ROUTING ALGORITHMS FOR HIGH-PERFORMANCE VLSI PACKAGING

BY
MUHAMMET MUSTAFA OZDAL

B.S., Bilkent University, 1999
M.S., Bilkent University, 2001

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2005

Urbana, Illinois

ROUTING ALGORITHMS FOR HIGH-PERFORMANCE VLSI PACKAGING

Muhammet Mustafa Ozdal, Ph.D.
Department of Computer Science
University of Illinois at Urbana-Champaign, 2005
Martin D. F. Wong, Adviser

We have seen dramatic advances in the IC technology in the past several years. The shrinkage of die sizes and the increase in functional complexities made the circuits more and more dense. Furthermore, the number of timing critical nets in a typical high-end design has increased considerably due to increasing clock frequencies. These factors have brought significant routing challenges that cannot be handled by traditional board routing algorithms. In this dissertation, we propose novel routing algorithms targeted at handling the challenges due to increasing package densities, and increasing clock frequencies.

Routing nets within minimum and maximum length bounds is an important requirement for high-speed VLSI packages. For this problem, we first propose a Lagrangian relaxation based length matching routing algorithm, where the objective of satisfying min-max length constraints is effectively incorporated into the actual routing problem. Our experiments demonstrate that our algorithm outperforms a commonly used ad hoc methodology, especially when the length constraints are tight. Although this algorithm can be used for more general routing problems, we also consider more restricted yet common problem instances, and propose more effective routing algorithms for them. Specifically, we first focus on the problem of two-layer bus routing between component boundaries. We model this problem as a job scheduling problem, and propose algorithms to solve it effectively. After that, we focus on the problem of routing bus structures between component boundaries on a single layer. For this, we propose algorithms that are proven to give close-to-optimal solutions.

As the package densities are increasing, routing nets from individual pins within dense components to the component boundaries (escape routing) is becoming the main bottleneck in terms of overall routability. Furthermore, solving the escape routing problem in each component independently is not an effective methodology for high-end board designs, since it ignores the wiring requirements between different components. For this, we propose novel models and algorithms to solve the escape routing problem in multiple components simultaneously, such that the number of crossings in the intermediate area (between components) is minimized. Our experiments demonstrate that these algorithms can reduce via requirements substantially,

compared to a net-by-net methodology. We also consider practical generalizations of these models, and discuss how to incorporate several high-speed design constraints into the framework of these algorithms. Finally, we focus on the problem of escape routing within dense pin clusters, which can have arbitrary convex boundaries. We propose a set of sufficient and necessary conditions that guarantee routability outside the escape boundaries. We also discuss how these conditions can be incorporated effectively into an escape routing algorithm.

To my family

ACKNOWLEDGMENTS

I would like to express my deepest gratitudes to my supervisor, Professor Martin D. F. Wong, for his guidance throughout this research work. The stimulating discussions with him were a tremendous help for completing this research work. He has always been supportive, understanding, and encouraging during my studies.

I am also indebted to the members of my dissertation committee, Professor Janak Patel, Professor Lenny Pitt, and Professor Josep Torrellas, for their interest and constructive comments.

I would also like to thank the Microelectronics Group in IBM for funding this project. In particular, I would like to thank Philip Honsinger, Dan Miller, John Ludwig, and George Katopis, for their invaluable discussions, and for providing the industrial problems used in our experiments.

I would also like to thank all the members of the VLSI CAD group at the University of Illinois at Urbana-Champaign. In particular, I would like to thank Lei Cheng, Liang Deng, Hua Xiang, and Yu Zhong, for their support, and for making my time enjoyable during my studies.

Finally, I am grateful to my family and all my friends for their infinite moral support and help during so many years.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

During the past several years, we have seen dramatic advances in the IC technology. The shrinkage of die sizes and the increase in functional complexities made the circuits more and more dense. So, boards and packages have reduced in size, while the pin counts have been increasing. For example, a multichip module (MCM) used in IBM eServer z900 [26] (introduced in 2000), contains 20 processor chips, 8 L2 cache chips, 2 system control chips, 4 memory bus adapter chips, and a clock chip – a total of 35 chips in one package. On the bottom of this MCM, there are 4224 I/O pins, within an area of 127-mm $\times$ 127-mm. In the subsequent generation of the same series, IBM eServer z990 [61] (introduced in 2003), the corresponding number of pins in an MCM has increased about 20%, with a decrease of almost 50% in the substrate area. With increasing pin densities of this pace, routing nets on boards beneath the component areas (escape routing) is increasingly becoming the main bottleneck in terms of overall routability [61]. Furthermore, the number of timing-critical nets in a typical high-end design has increased significantly due to increasing clock frequencies. While only 2-5% of the nets were timing-critical in the past, today this ratio can reach to 90% in a typical high-end design [60]. These factors bring significant routing challenges that cannot be handled by traditional board routing algorithms. Today, many high-end board designs in the industry are being routed using manual efforts [40], since the existing autorouters fail to produce acceptable solutions.

In this dissertation, we propose novel routing algorithms that can handle challenges due to increasing package densities, and increasing clock frequencies [41–48].

A typical printed circuit board (PCB) contains a number of different components such as MCMs, memory, or I/O modules. These components are mounted on or

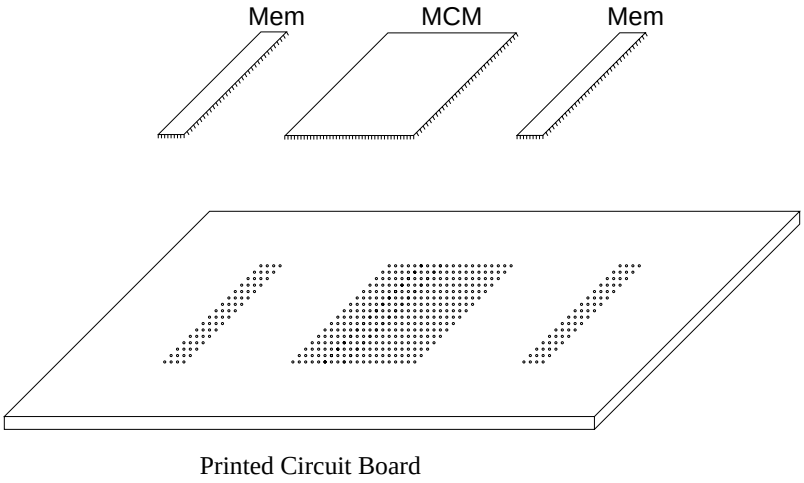


Figure 1.1: Different components are mounted on or plugged into a PCB. A pin array is created on the board corresponding to each component.

plugged in to the board, forming a set of dense pin arrays, as shown in Figure 1.1. The routing resources within such pin arrays are extremely limited due to the large number of pins, and tight clearance rules. Furthermore, there are large number of nets that need to be routed from their terminal pins to the corresponding component boundaries. On the other hand, the intermediate routing area on the board between different components has relatively few blockages, and the amount of available routing resources is relatively larger.

In accordance with this characteristics, we propose a problem decomposition that handles routing within dense pin arrays separately from the intermediate area routing. In other words, two separate problems are distinguished here: (1) routing nets from pin terminals to component boundaries (escape routing), and (2) routing nets between component boundaries (area routing). For escape routing, the main emphasis is on routability: routing as many number of nets as possible using the limited resources inside dense pin arrays. On the other hand, during area routing, we mainly focus on timing constraints due to high clock frequencies. Figure 1.2 illustrates a sample problem instance where the objective is to route a group of nets between two components, which are shown as dense pin arrays on the left and right sides of the figure.

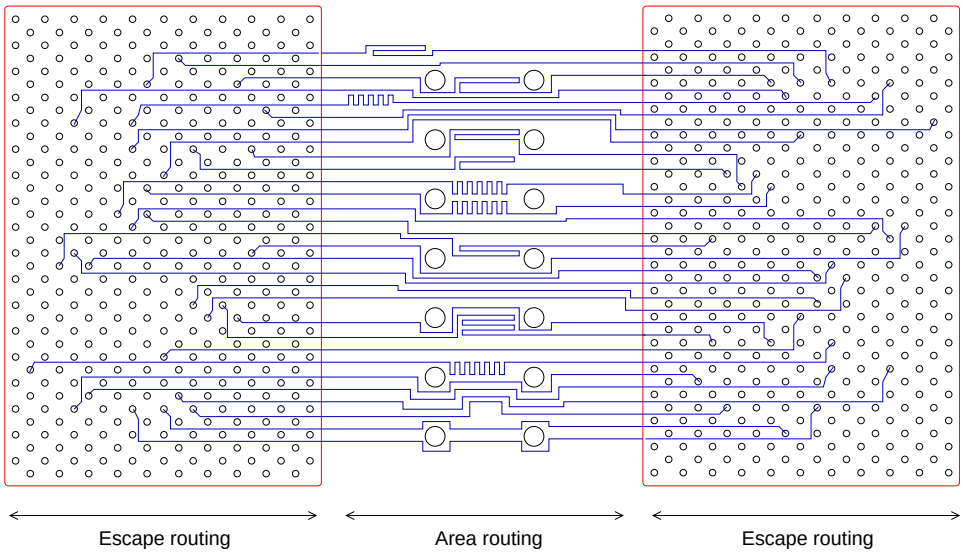


Figure 1.2: Escape routing and area routing solutions of a problem instance containing two components. The lengths of some nets in the intermediate area have been extended to satisfy the min-length constraints.

1.1 Overview of Dissertation

Timing constraints are commonly imposed on PCB bus structures, where data is clocked into registers or other circuits. For example, in the case of a 64-bit data bus, each bit travels over a different wire, and all 64 bits must arrive destination pins *approximately* at the same time. To achieve this, all the wires constituting this bus need to have *approximately* same lengths. The precision with which matching must be done is directly related to the clock frequency. As the clock frequency increases, the skew requirements on the propagation delays become more strict, and hence, a higher degree of length matching is required. There have been several algorithms proposed in the literature for the objective of minimizing path lengths, or satisfying prespecified maximum length constraints. However, the problem of routing nets with lower bound constraints has not been studied explicitly. As circuits start to use clock frequencies in the order of gigahertz in the current technology, the timing constraints become extremely tight, and more aggressive methods for achieving length bounds are needed in the industrial applications.

In Chapter 2, we propose a novel algorithm that incorporates the objective of satisfying min-max length constraints effectively into the original routing problem. Here, we model the problem of length matching as a constrained optimization prob-

lem, and use Lagrangian relaxation to obtain a new routing objective function. Then, we perform multiple routing iterations, each of which is guided by the global objective of length matching. In one iteration, we route nets and allocate resources so as to minimize our objective Lagrangian function, which captures both min and max length constraints for all nets. Our experiments show that this algorithm outperforms a commonly used ad hoc methodology.

In Chapter 3, we focus on a more restricted yet common length matching problem: routing nets between component boundaries using two x-y signal layers. Here, the component boundaries define a routing channel, and all net terminals are assumed to be aligned on the opposite sides of this channel. The objective is to route all nets while satisfying their min-max length constraints. Routability in a highly congested area is expected to be limited; so it is more effective to perform length extension (to satisfy min-length constraints) within the less congested areas. For example, the vertical layer of a horizontal problem is expected to be significantly less congested; so it makes more sense to perform length extension on this layer. In Chapter 3, we propose an algorithm that incorporates the objective of length extension into the actual routing algorithm. For a horizontal problem, our algorithm simultaneously extends the lengths of the nets and assigns them to vertical tracks. For this, we first model the routing problem as a *task scheduling problem with release times and deadlines*. Here, the min-max length constraints of a net correspond to the release times and deadlines of one task in the scheduling problem, and a vertical routing track corresponds to one machine. Although the scheduling problem is NP-complete even for the single machine case, we propose a polynomial-time optimal algorithm for one track, due to a special property of the given routing problem. In particular, our approach here is to process one routing track at a time and to choose the best subset of nets to be routed on each track. The algorithm we propose is guaranteed to find the optimal subset of nets together with the optimal solution with length extension on one track.

In Chapter 4, we focus on board designs that do not use any buried vias, due to high manufacturing costs. For such designs, each net needs to be routed on a single layer in a planar fashion. Similar to the problem of Chapter 3, we assume that boundaries of the components define a routing channel, and all net terminals are aligned on the opposite sides of the channel. The objective is to route all nets on a single layer such that each net satisfies its prespecified min-max length constraints.

For this problem, we propose an algorithm in Chapter 4, and we prove that it gives close-to-optimum routing solutions. In particular, if there exists a feasible routing solution for a given set of nets on a channel width of W, we prove that our algorithm is guaranteed to find a feasible solution for a channel width of $W + 3$. Since typical channel widths in the industry are on the order of hundreds, or even thousands, this difference is negligible in practice.

In Chapter 5, we propose algorithms for escape routing problem, which is defined as routing nets from their respective pins to the component boundaries. As mentioned above, we need more effective algorithms to solve the escape routing problem, due to increasing package densities. It is important here to note that escape routing for different components should not be considered independent of each other. In other words, we cannot just apply a traditional escape routing algorithm on different components independently. The reason is that such an approach ignores the connections between different components and increases the via requirements significantly. Especially in high-speed designs, these vias seriously degrade signal characteristics, add additional delay, decrease routing area, and lower the manufacturing yields. Furthermore, for some board designs, no buried vias are allowed for the purpose of limiting manufacturing costs [40]. For such designs, the nets need to be routed in a planar fashion on every layer. Hence, an escape routing algorithm that tries to minimize (or completely avoid) crossings in the intermediate area is crucial to handle the recent challenges encountered in board routing problems. For this reason, we propose algorithms in Chapter 5 to find the escape routing solutions of multiple components simultaneously such that the number of crossings in the intermediate area is minimized. For multilayer designs, the best layer assignment also needs to be determined during this process. Our approach to solve this problem is to process one layer at a time and to try to route as many planar nets as possible on each layer. For this purpose, we generate a number of escape patterns for each net and try to choose the maximum subset of patterns such that (1) at most one pattern is selected for each net, (2) there are no conflicts within components, and (3) there are no crossings in the channel. Note that even though we consider only a limited number of routing patterns for each net, there are exponential number of possible ways of selecting patterns for a set of nets. However, we propose a polynomial time optimal algorithm to select the best combination that gives the maximal planar routing solution. We also propose a faster randomized algorithm that gives almost as good results as the

optimal algorithm in practice. Experiments on industrial problems show that our algorithms can reduce the via requirements significantly, compared to a Pathfinder based net-by-net approach.

In Chapter 6, we propose further improvements for the escape routing algorithm foundations of which are presented in Chapter 5. Here, we propose three main improvements: (1) Escape patterns are generated based on the congestion levels inside the components and the number of crossings in the intermediate region, instead of simple straight connections. (2) An improved maximal planar route selection algorithm is proposed, which is general enough to handle multi-capacity escape slots, and high-speed design constraints. (3) Explicit discussion about how to handle various high-speed design constraints is given for this framework. Our experiments demonstrate that these improvements can reduce the via requirements of industrial test cases on average by 39%, compared to the basic algorithm of Chapter 5.

In Chapter 7, we study another important routing problem encountered in typical high-end MCM designs: routing within dense pin clusters. Pin clusters are often formed by pins that belong to the same functional unit or the same data bus, and can become bottlenecks in terms of overall routability. Typically, these clusters have irregular shapes, which can be approximated with rectilinear convex boundaries. Since such boundaries have often irregular shapes, a traditional escape routing algorithm may give unroutable solutions. In this chapter, we study how the positions of escape terminals on a convex boundary affect the overall routability. For this purpose, we propose a set of necessary and sufficient conditions to model routability outside a rectilinear convex boundary. Given an escape routing solution, we propose an optimal algorithm to select the maximal subset of nets that are routable outside the boundary. After that, we focus on an integrated approach to consider routability constraints (outside the boundary) during the actual escape routing algorithm. Here, we propose an optimal algorithm to find the best escape routing solution that satisfies all routability constraints. Our experiments demonstrate that we can reduce the number of layers by 17% on average by using this integrated methodology.

Finally, in Chapter 8, we give our concluding remarks and discuss future research directions.

Chapter 2

A Lagrangian Relaxation Based Length Matching Algorithm

2.1 Introduction

Routing nets within minimum and maximum length bounds is an important requirement for high-speed VLSI layouts. There have been several algorithms proposed for the objective of minimizing path lengths or satisfying prespecified maximum length constraints, especially in the context of timing-driven routing [5; 12; 13; 18; 35; 37; 52]. However, the problem of routing nets with lower bound constraints has not been studied explicitly in the literature. The main reason is that these bounds were loose most of the time, and non-sophisticated strategies (such as greedy length extension in post-processing) were sufficient for most applications. However as circuits start to use clock frequencies in the order of gigahertz in the current technology, the timing constraints become extremely tight, and more aggressive methods for achieving length bounds are needed in the industrial applications.

Timing constraints are commonly imposed on PCB bus structures, where data is clocked into registers or other circuits. For example, in the case of a 64-bit data bus, each bit travels over a different wire, and all 64 bits must arrive destination pins *approximately* at the same time. To achieve this, all the wires constituting this bus need to have *approximately* same lengths. The precision with which matching must be done is directly related to the clock frequency [54]. As the clock frequency increases, the skew requirements on the propagation delays become more strict, and hence, a higher degree of length matching is required.

A typical approach used for this problem is to route the nets using a conventional routing algorithm to satisfy max length constraints, and then perform *snaking* to extend the routes of the short nets during postprocessing. The main disadvantage of such an approach is that after all the nets have already been routed, the available routing space around short nets might be limited in dense designs. So, it is likely that some nets cannot be extended to satisfy minimum length constraints due to lack of routing space.

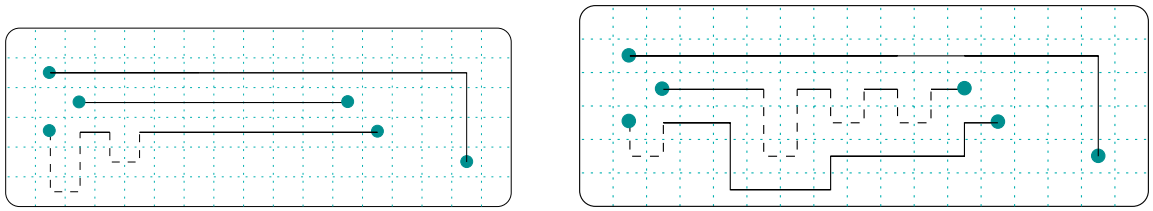
In this chapter, we propose a novel algorithm that incorporates the objective of satisfying min-max length constraints effectively into the original routing problem. For the ease of presentation, we will first focus on the length matching problem, and then we will extend our models for the general case where individual nets might have different lower and upper bound constraints. For this, we start with redefining the routing problem as follows: Find valid routes for all nets such that *(1) the length of the longest route is kept small, and (2) the shorter routes have available routing space around themselves such that it is possible to match all lengths by snaking at the end.* We propose effective algorithms in this chapter to handle both objectives simultaneously during routing.

As a motivating example, consider the circuit given in Figure 2.1(a). Here, there are three nets that need to be routed with equal lengths, and the figure illustrates a typical routing solution[1] given by a conventional router. Here, all nets were routed first, and then snaking was performed at the end for length matching. Observe that the top net turned out to be the longest one, with a path length[2] of 17. So, the length of the bottom net was extended by 6 through snaking. However, snaking was not possible for the middle net, because all routing resources around its route were used during routing. So, length matching fails in this example.

Figure 2.1(b) shows the solution given by the router we propose in this chapter. Observe that the lengths of these three nets are matched *exactly* through snaking. Here, our approach is to simultaneously route each net and allocate extra routing resources (i.e., grid cells) for them. After that, these extra resources are used for snaking. There are a couple of points worth mentioning here. First of all, the number of extra grid cells allocated for a net depends on the length of its route (i.e., more

[1]The underlying grid structure is also shown in this figure. Throughout the chapter, we assume that routing edges go center-to-center of each grid cell, as illustrated in this figure. Note that each grid cell is regarded as a *routing resource*.

[2]All the path lengths given in this chapter are in terms of number of grid cells spanned.



(a) (b)

Figure 2.1: Length matching based on (a) greedy snaking in postprocessing, and (b) resource allocation during routing. Dashed lines indicate snaking performed. Observe that the length of the middle route could not be extended in part (a).

grid cells are allocated for shorter nets, and vice versa). Here, it is likely that the actual routes of the nets will be affected because of this resource allocation. In this example, the bottom net is detoured so that there are enough resources allocated for the middle net. An important point here is that it is not the top net that is detoured for this purpose, because detouring the top net would increase the length of the longest route. In fact, we can say that the two objectives for length matching mentioned above are achieved simultaneously in this example.

As will be discussed in detail later, we perform multiple iterations, each of which is guided by the global objective of length matching. In one iteration, we route nets and allocate resources so as to minimize an objective Lagrangian function, which captures both min and max length constraints for all nets. Our low-level routing algorithm is based on Pathfinder negotiated congestion algorithm [2; 3; 19]. However, we propose a methodology to handle resource allocation simultaneously during path calculations, as opposed to the greedy algorithm above, which considers min constraints only in post-processing. In our approach, shorter nets automatically prefer the paths where they can allocate extra resources around.

The rest of the chapter is organized as follows. In Section 2.2, we summarize the relevant work in the literature and discuss why they are not applicable for this problem. Then, we propose a Lagrangian relaxation based algorithm that facilitates allocating extra resources during routing in Section 2.3. After that, we propose a graph model in Section 2.4 to perform resource allocation in accordance with snaking. Specifically, this model makes sure that if the number of extra grid cells allocated for net i is S_i, it is possible to extend length of net i by an amount equal to S_i through snaking. In other words, resource allocation is done in such a way that

every allocated grid cell can be used for snaking later. We then outline the low-level routing algorithm we use in Section 2.5. Then in Section 2.6, we briefly explain how to extend this method for more general problems. Note that even though we propose a systematic approach based on Lagrangian relaxation for this problem, the solution found is not guaranteed to be optimal, because of the non-convex nature of the problem. Furthermore, it is not guaranteed that a feasible solution will be found, even if one exists. However, we demonstrate the effectiveness of our heuristics through experiments in Section 2.7.

2.2 Related Work

There have been several routing algorithms proposed in the literature for the objective of satisfying maximum length constraints [5; 12; 13; 18; 35; 37; 52]. Typically, these algorithms try to keep the lengths of critical nets shorter, but they do not consider explicit minimum length constraints.

A related problem in the literature is the *zero/bounded skew clock tree routing* problem [32]. Here, the objective is to construct a clock tree such that the arrival times for all source-sink pairs are (almost) equal. However, our bus routing problem is different in the sense that each terminal pair belongs to a different net, and no overlaps are allowed between different pairs. On the other hand, in the clock tree routing problem, there is a single net (with multiple terminals), and the objective is to find a routing tree, instead of independent pairwise connections. Several algorithms have been proposed in the literature for this problem [11; 31; 32; 58]. However, they are based on tree construction methods most of the time, and they are not applicable to the case where each pairwise connection must be routed independent of each other.

If the length matching problem consists of only two nets, it is possible to use a wave expansion method to find a feasible solution [49]. Let us denote the source-sink pair of the two nets as (s_1, t_1) and (s_2, t_2). Here, waves are expanded originating from these four terminals, and their intersections are checked repeatedly. Namely, whenever waves from s_1 and t_1 meet, the length of the corresponding path is compared with every path between s_2 and t_2. This process continues until a match is found between the lengths of two paths. Note that this approach does not explicitly avoid short circuits between the two nets; so special care must be taken, such as restricting propagating waves to separate portions of the layout [49]. However, we cannot use

this technique in our bus routing problem, since the number of nets is typically much larger than 2. Here, the main problem is that conflicts between different nets cannot be detected during simultaneous wave expansions, and it would not be practical to limit the waves of all nets to separate regions when there are multiple nets.

On the other hand, some traditional routing tools allow users to specify length matching constraints. However, as the clock frequencies increase, the constraints for typical high-end circuits become extremely tight, and these tools fail to find a feasible routing solution for many high-end industrial designs. A commonly used practical approach here is to route all nets first, and then to perform length extension in post-processing. The disadvantage of such an approach is that after all nets have already been routed, the available routing space around short nets might be limited in dense designs. So, it is likely that some nets cannot be extended to satisfy min-length constraints due to lack of space. In Section 2.7, we will present an experimental comparison of this practical approach with our framework.

2.3 Routing Resource Allocation

2.3.1 Problem Formulation

The original length matching problem can be stated as follows. Given a circuit, and a set of nets $\mathcal{N}$, find a congestion-free routing solution for each net in $\mathcal{N}$ such that the maximum path length is minimized, and the difference between the minimum and maximum path lengths does not exceed the predefined tolerance value Δ. Here, the input circuit is assumed to be modeled as a uniform $n \times m$ grid structure, where each grid cell is marked as either a *routing resource*, or a *blockage*. Each net in $\mathcal{N}$ is assumed to have two fixed terminals on the grid structure. A routing solution for a set of nets $\mathcal{S}$ is defined to be *congestion-free* if and only if no *routing resource* on the grid is used by more than one net in $\mathcal{S}$.

To solve the length matching problem, we introduce two main objectives for the router: (1) to keep the path lengths of longer nets small, and (2) to allocate extra routing resources around shorter nets such that their lengths can be extended through snaking. Intuitively, we want to minimize the expression $\sum_{i \in \mathcal{N}}(\alpha_i L_i - \beta_i S_i)$, where L_i is the length of net i's route, S_i is the total number of extra grid cells allocated for net i, and α_i and β_i are weighting terms. One can argue that for long nets, α_i should

be large, giving priority to path length minimization. On the other hand, for short nets, β_i should be large, giving more priority to resource allocation. In this section, our focus will be on how to set and update these parameters dynamically such that the two main objectives are achieved simultaneously.

For simplicity of the presentation, we assume that routing will take place on one layer only. Furthermore, our focus will be to route only one bus; i.e., all the given nets need to be routed with the same length. However, it is straightforward to extend our models and algorithms to a multi-layer multi-bus routing problem, or to the general problem where each net has a different length constraint, as will be discussed in Section 2.6. Also, we introduce some restrictions for the resulting routing solutions. We assume that there is a *preferred direction* for each net, and all the snaking will be performed perpendicular to this direction. Furthermore, the resulting routes will not have any detour towards opposite of the preferred direction. For example, if the preferred direction is RIGHT, then snaking will be performed UP and DOWN (as in Figure 2.1); detouring towards LEFT will not be allowed. These restrictions are necessary for the models we propose. However, we believe that they will not degrade the solution quality, because a typical routing solution given by a conventional router would also satisfy these conditions. For simplicity of presentation, we will first assume that there is a *global* preferred direction for all nets. However, it is possible to generalize our models to the case where each net has an individual preferred direction, as will be discussed in Section 2.6.

2.3.2 Lagrangian Relaxation Based Resource Allocation

Lagrangian relaxation is a general technique for solving optimization problems with difficult constraints. The main idea is to replace each complicating constraint with a penalty term in the objective function. Specifically, each penalty term is multiplied by a constant called *Lagrangian multiplier (LM)*, and added to the objective function. The Lagrangian problem is now the optimization of the new objective function, where difficult constraints have been relaxed and incorporated into the new objective function. If the optimization is a minimization problem, then the solution of Lagrangian problem is guaranteed to be a lower bound for the original optimization. In fact, Lagrangian relaxation is a two-level approach: In the low level, the Lagrangian problem is solved for fixed LM values. In the high level, LM values are updated iteratively such that the optimal value obtained in the low level is as close to the real

optimal value as possible. Typically, a subgradient method is used to update LM values in the high level. Intuitively, the LM values corresponding to the constraints that are not satisfied in the current iteration are increased (hence, the weights of these constraints in the low-level objective function are increased), and vice versa. The iterations continue until a convergence criterion is satisfied. Further details can be found in various survey or tutorial papers about Lagrangian relaxation [21–23].

Length matching problem can be formulated as a constrained optimization problem. Assume that we *somehow* determine[3] a target length T; and our purpose is to route each net i in set $\mathcal{N}$ with a path length in the range $T - \Delta$ and T.

Based on the resource allocation idea we have discussed before, it is possible to give the following formulation:

$$\begin{aligned} &\textit{minimize} && \sum_{i \in \mathcal{N}} L_i \\ &\textit{subject to}: && \\ & && \forall i, \quad L_i \leq T \\ & && \forall i, \quad L_i + S_i \geq T - \Delta \end{aligned} \tag{2.1}$$

Again, L_i denotes the length of net i's route, and S_i denotes the number of extra grid cells allocated for net i. Suppose for now that it is possible to extend the length of net i by an amount up to S_i using snaking (in Section 2.4, we will propose a model that will facilitate this). Observe that the first constraint above simply states that the total length should not exceed the target length. On the other hand, with the second constraint we make sure that shorter nets allocate enough routing resources for snaking.

If we apply Lagrangian relaxation on this formulation, our objective becomes minimization of

$$\sum_{i \in \mathcal{N}} L_i + \sum_{i \in \mathcal{N}} \lambda_{iL}(L_i - T) - \sum_{i \in \mathcal{N}} \lambda_{iS}(L_i + S_i - T + \Delta) \tag{2.2}$$

Here, each λ_{iL} and λ_{iS} are Lagrangian multipliers corresponding to length and resource constraints given in the original formulation (2.1). Intuitively, we would

[3]Initially, T can be set based on the maximum Manhattan distance of the terminal positions of the input nets. If no routing solution is found with target length T, it can be increased gradually throughout the execution.

ROUTE-AND-LENGTH-MATCH (Inputs: $\mathcal{N}$, T)

Initialize λ_{iL}, λ_{iS} to zero for each $i \in \mathcal{N}$
while termination condition not occurred do
    route all nets for fixed λ_{iL}, λ_{iS} values (see Section 2.5)
    for each $i \in \mathcal{N}$ do
        check constraints for current route
        update λ_{iL} and λ_{iS} values accordingly
perform snaking using the extra grid cells allocated

Figure 2.2: High-level algorithm description.

want longer nets to have larger λ_{iL} values (so that length minimization is prioritized for them) and shorter nets to have larger λ_{iS} values (so that resource allocation is prioritized for them).

The high-level algorithm we propose for length matching during routing is given in Figure 2.2. For the following discussions in this section, assume that we have a subroutine for finding the routing solution that minimizes objective function (2.2), for fixed λ_{iL} and λ_{iS} values. Observe in Figure 2.2 that we iteratively call this subroutine, and update the Lagrangian multipliers until some convergence criterion is satisfied. We use an update scheme similar to subgradient method, but we have tailored it specifically for this problem. Given a routing solution in iteration k, and the current multiplier values λ_{iL}^k and λ_{iS}^k, the multipliers for iteration $k+1$ are calculated as follows:

$$\lambda_{iL}^{k+1} = \begin{cases} max(0, \lambda_{iL}^k - t_k(T - L_i)^\gamma) & \text{if } L_i \leq T, \\ \lambda_{iL}^k + t_k v_{iL}(L_i - T)^\gamma & \text{otherwise.} \end{cases} \tag{2.3}$$

$$\lambda_{iS}^{k+1} = \begin{cases} max(0, \lambda_{iS}^k - t_k(L_i + S_i - T + \Delta)^\gamma) & \text{if } L_i + S_i \geq T - \Delta, \\ \lambda_{iS}^k + t_k v_{iS}(T - \Delta - L_i - S_i)^\gamma) & \text{otherwise.} \end{cases} \tag{2.4}$$

Note that t_k is the step size used in subgradient method, and it is updated in each iteration such that it slowly converges to 0. Specifically, we use the convergence condition given by Held et al [28], which states that as $k \to \infty$, it should be the case that $t_k \to 0$ and $\sum_{i=1}^{k} t_i \to \infty$. The terms v_{iL} and v_{iS} denote the number of iterations the length constraint ($L_i \leq T$) and the resource constraint ($L_i + S_i \geq T - \Delta$) for

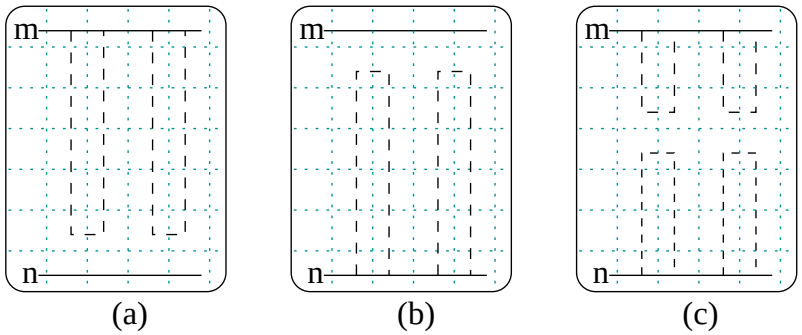


Figure 2.3: Parallel routing segments of net m and net n, together with allocated routing resources (indicated by dashed lines): (a) resource allocation if $\lambda_{mS} > \lambda_{nS}$, (b) resource allocation if $\lambda_{nS} > \lambda_{mS}$, and (c) desirable resource allocation if λ_{nS} is only slightly larger than λ_{mS}.

net i have been violated, respectively. If a constraint is not satisfied repeatedly for several iterations, then its multiplier is increased more rapidly. Finally, $\gamma \leq 1$ is a constant we have introduced for this problem, and it is used to smooth the effect of the amount of length or resource constraint violation, which can have large values. Our experiments have shown that setting it to a value as small as 0.1 gives decent results.

2.3.3 Handling Oscillation Problems

It is known that solution oscillation is a serious and inherent problem for Lagrangian relaxation based methods [24; 53]. Note that even if the Lagrangian multipliers converge to their optimal values in the subgradient method, the solution to the original problem might oscillate between two extremes with a slight change of the multipliers. Guan et al. [25] identify one cause of such a behavior as the existence of *homogeneous subproblems*. A similar problem also exists in the formulation we have given in Section 2.3.2.

Figure 2.3 illustrates this problem with an example of two parallel routing segments. Assume that both net m and net n need to allocate extra routing resources (i.e., grid cells) around their routes to satisfy their resource constraints. Observe that to minimize objective function (2.2), the intermediate grid cells should be allocated by net m or net n, depending on the values of λ_{mS} and λ_{nS}. Specifically, if $\lambda_{mS} > \lambda_{nS}$, then function (2.2) will be minimized if S_m has its maximum value. Hence, all the intermediate grid cells will be allocated by net m (Figure 2.3(a)). On the other hand,

if $\lambda_{nS} > \lambda_{mS}$, then S_n will be set to its maximum value as in Figure 2.3(b) to minimize the objective function. Note that even if the difference between two Lagrangian multipliers is infinitely small, the solution will be one of these extreme cases;[4] so the solution will always oscillate between these two. The desirable behavior would be as shown in Figure 2.3(c) when λ_{mS} and λ_{nS} are close to each other.

A typical remedy for this kind of a problem is to use *augmented Lagrangian relaxation* [53; 59], where a penalty term is added to the Lagrangian function to avoid oscillations. Using a similar idea, we can modify objective function (2.2) such that our new objective becomes the minimization of

$$\sum_{i \in \mathcal{N}} (L_i + \lambda_{iL} L_i - \lambda_{iS} S_i) + \sum_{i \in \mathcal{N}} \sum_{e \in P_i} \epsilon (s^e)^2 \tag{2.5}$$

where P_i denotes the path of net i, e denotes a unit edge (between two neighboring grid cells) in P_i, and s^e denotes the number of extra grid cells allocated around edge e, i.e., $\sum_{e \in P_i} s^e = S_i$.

Here, we first simplified the original function (2.2) by eliminating the constant terms. Then, we added the term $\sum_{i \in \mathcal{N}} \sum_{e \in P_i} \epsilon (s^e)^2$ as a penalty term for resource allocation. Note that, ϵ is expected to be a small constant compared to the initial step size t_0 used to update Lagrangian multipliers. Intuitively, we want the penalty term to be ineffective in earlier iterations, but as the multiplier values start to converge to their optimal values, we want it to effectively dampen the oscillations. Note that the resulting behavior will be similar to the one illustrated in Figure 2.3(c). Also as a side effect, we had to eliminate the term $-\sum_{i \in \mathcal{N}} \lambda_{iS} L_i$ from function (2.2). The reason can be explained by using the example given in Figure 2.3. Assume that both net m and n have small λ_L, but large λ_S values, and assume that λ_{nS} is slightly larger than λ_{mS}. Due to the penalty term added, it is possible that the term $-\lambda_{nS} L_n$ dominates instead of $-\lambda_{nS} S_n$; so, L_n will be maximized, instead of S_n. The result would be similar to the case shown in Figure 2.3(b), but this time with a *snaking-like* behavior[5] instead of resource allocation. So, we also need to remove the term $-\sum_{i \in \mathcal{N}} \lambda_{iS} L_i$. It is interesting to note here the similarity between the new objective function (2.5), and the intuitive formula $\sum_{i \in \mathcal{N}} (\alpha_i L_i - \beta_i S_i)$, given in Section 2.3.1.

[4] The case $\lambda_{mS} = \lambda_{nS}$ would give an arbitrary outcome, so we ignore this case in our discussions.

[5] The routing algorithm we use (Section 2.5) maximizes length if all the edge weights are negative.

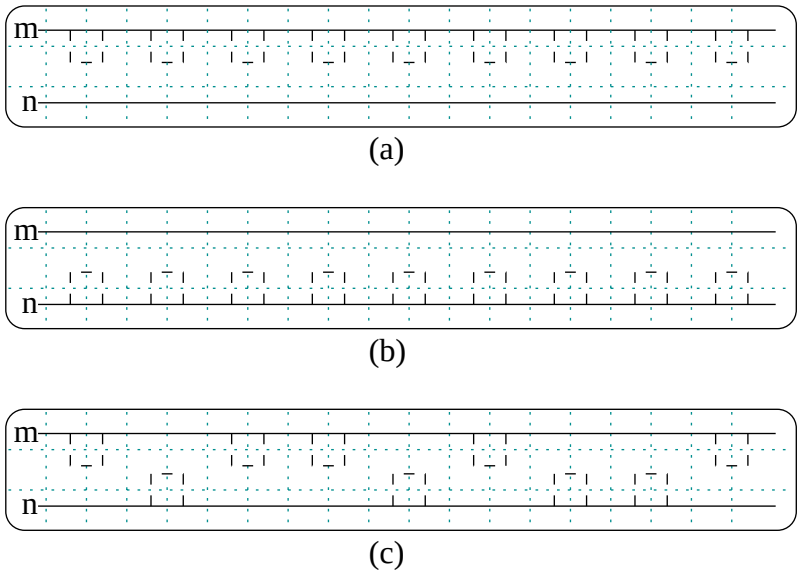


Figure 2.4: Parallel routing segments of net m and net n, together with allocated resources (indicated by dashed lines): (a) resource allocation if $\lambda_{mS} > \lambda_{nS}$, (b) resource allocation if $\lambda_{nS} > \lambda_{mS}$, and (c) desirable resource allocation if λ_{mS} is slightly larger than λ_{nS}.

Another source of possible oscillations is due to the fact that we route all nets using fixed Lagrangian multiplier values. As shown in Figure 2.4, if λ_{mS} is even slightly larger than λ_{nS}, all the intermediate grid cells would be allocated for net m, and vice versa, to minimize objective function (2.5). The reason for such a behavior is that the Lagrangian multipliers are updated only after the complete routing solution is found using the fixed multiplier values. For instance, assume that it is required to allocate extra grid cells for both net m and n to satisfy their resource constraints (i.e., as in Figure 2.4(c)). If the solution in iteration k is as in Figure 2.4(a), λ_{mS} would be decreased, and λ_{nS} would be increased for the next iteration. So, the solution in iteration $k+1$ would be as in Figure 2.4(b). Similar arguments suggest that the solution will always oscillate between these two extreme cases.

We propose a simple yet effective heuristic for this problem. First, we rewrite the objective function (2.5) without any modifications as follows:

$$\sum_{i \in \mathcal{N}} \sum_{e \in P_i} (1 + \lambda_{iL} - \lambda_{iS} s^e + \epsilon (s^e)^2) \tag{2.6}$$

Again, $e \in P_i$ is a unit edge in the path of net i. This formulation suggests that we need to access the variables λ_{iL} and λ_{iS} for each edge $e \in P_i$. To avoid the oscillation

problem described above, we will apply random smoothing each time such an access occurs. Specifically, instead of using λ_{iL}^k and λ_{iS}^k in iteration k, we will use

$$\lambda_{iL} = \alpha\lambda_{iL}^k + (1-\alpha)\lambda_{iL}^{k-1} \tag{2.7}$$

$$\lambda_{iS} = \alpha\lambda_{iS}^k + (1-\alpha)\lambda_{iS}^{k-1} \tag{2.8}$$

where α is a random number in the range $[0, 1]$, and it is regenerated for each access to λ_{iL} and λ_{iS} values. Observe that such a smoothing is not expected to affect the results if there are no oscillations (since the multiplier values in iterations $k-1$ and k would be consistent with each other). However, in case of oscillations as in Figures 2.4(a) and (b), the result is expected to turn out eventually as in Figure 2.4(c).

2.4 Graph Model

In this section, we propose a graph model that facilitates resource allocation during shortest path calculations. The significance of this model is that all the extra grid cells allocated for net i can be used for extending the length of net i through snaking. In other words, our low-level routing algorithm will operate on this graph, so that there will be a one-to-one correspondence between resource allocation (during routing) and snaking (in post-processing). For simplicity of the presentation, we will give the graph model in case the *preferred direction* (see Section 2.3.1) is RIGHT. It is straightforward to extend this model for the other directions.

As a first step, we define a supernode corresponding to each routing grid cell. A supernode N is defined to contain three subnodes: u_N, d_N, and s_N. Each subnode corresponds to a different state of N in terms of the direction of the incoming edge. Namely, u_N, d_N, and s_N define the cases where the incoming edge to N is *upwards*, *downwards*, and *straight*, respectively. Figure 2.5 illustrates this graph model with an example. Here, supernodes A, B, and C correspond to three neighboring routing grid cells, where B and C are right and down neighbors of A, respectively. All eleven edges are illustrated separately with the corresponding physical explanation. For instance, the edge $s_A \rightarrow s_B$ corresponds to the case where the incoming edge to A is straight, and the connection from A to B is also straight. As another example, the edge $u_A \rightarrow d_B$ corresponds to the case where the incoming edge to A is upwards, and the connection from A to B is through allocating some of the top grid cells. Note that in this case, the direction of the incoming edge to B (from A) is regarded as

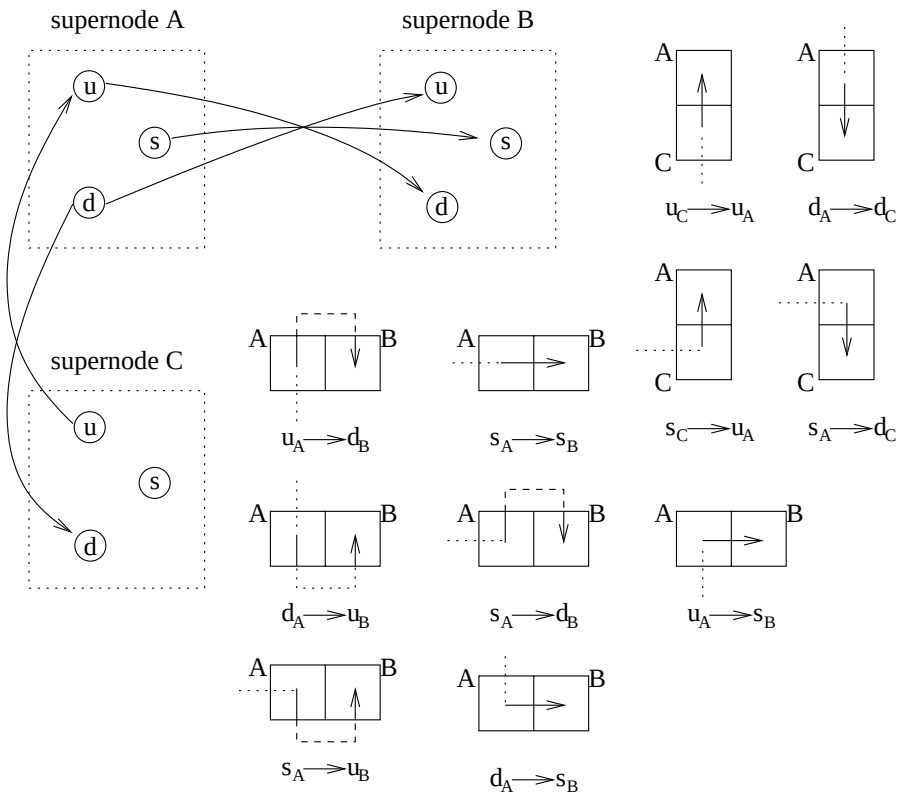


Figure 2.5: Three supernodes together with their subnodes are displayed on the upper left corner. Only 5 of the 11 edges are drawn in the big picture for clarity. All the 11 edges between supernodes A, B, and C are illustrated separately on the right.

downwards, assuming that the allocated grid cells will be used for snaking later.

One point to observe in Figure 2.5 is that resource allocation is possible only through the edges $u_A \rightarrow d_B$, $d_A \rightarrow u_B$, $s_A \rightarrow d_B$, and $s_A \rightarrow u_B$. This guarantees that all the allocated grid cells during min-cost path calculations can be used for snaking later. The issues such as assigning weights to these edges, determining the amount of resource allocation, etc. will be discussed in Section 2.5. However, we can state the following lemma based on the discussions above.

Lemma 2.1 *Let $\mathcal{R}$ be the original routing grid, and let $\mathcal{G}$ be the corresponding graph model. For any* valid *route (with snaking) in $\mathcal{R}$, there exists a corresponding path (with extra resource allocation) in $\mathcal{G}$. Furthermore, for any path P in $\mathcal{G}$, a route can be constructed in $\mathcal{R}$ such that all extra allocated resources are used for snaking.*

Figure 2.6 shows an example path on the routing grid, and its graph representation. Here, resource allocation is performed for two edges, and the notation *extra=4*

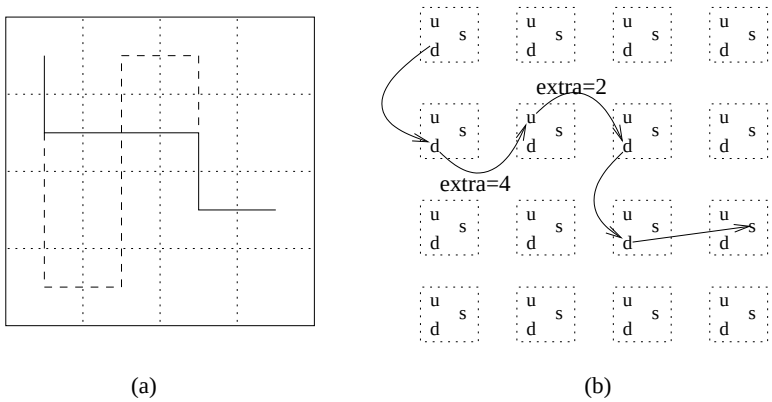


Figure 2.6: (a) An example routing segment, where allocated grid cells are shown with dashed lines, and (b) the corresponding path in our graph model.

in part (b) means that four extra grid cells are allocated around this edge. Observe that a total of six grid cells is allocated in part (a), and it is possible to extend the length of this path from 5 to 11 if all these grid cells are used for snaking.

2.5 Routing Nets

In this section we describe the methodology we use to route all nets $i \in \mathcal{N}$, given fixed λ_{iL} and λ_{iS} values. We will first give a brief overview of the Pathfinder negotiated congestion algorithm in Section 2.5.1. Then, we will discuss how to incorporate our Lagrangian cost functions into this methodology in Section 2.5.2.

2.5.1 Negotiated Congestion Algorithm

Our low-level routing algorithm is based on the Pathfinder negotiated congestion algorithm, which was originally proposed for FPGA routing problem [2; 3; 19]. The main idea here can be summarized as follows. First, every net is routed individually, regardless of any overuse (i.e., congestion) of routing grid cells. Then the nets are ripped-up and rerouted one by one iteratively. In each iteration, the congestion cost of each grid cell is updated based on the current and past overuse of it. By increasing the congestion cost of an overused grid cell gradually, the nets with alternative routes are forced not to use this grid cell. Eventually, only the net that needs to use this grid cell most ends up using it. More details about this heuristic-based routing algorithm can be found in [19].

In our implementation, we have used the following congestion cost function for grid cell g in iteration k:

$$congestion\ \ cost(g,k) = k^{\varphi} \,.\, t_g \,.\, (1 + history(g,k)) \tag{2.9}$$

where φ is a constant parameter, t_g is the number of nets that are passing through grid cell g in the current iteration, and $history(g,k)$ is the *congestion history* of grid cell g. In the beginning of the algorithm, the congestion history of each grid cell is initialized to zero. Then, after each iteration k, the congestion history of grid cell g is updated as follows:

$$history(g,k+1) = history(g,k) + v_{gc} \,.\, max(0, t_g - 1) \tag{2.10}$$

where v_{gc} denotes the number of consecutive turns in which grid cell g has been congested. Observe here that when a grid cell is congested for multiple iterations consecutively, its history is incremented by a larger value; hence, its congestion cost increases more rapidly. On the other hand, when a grid cell is not congested in the current iteration (i.e., when t_g is 0 or 1), its congestion history remains unchanged.

In cost function (2.9), φ is a user-defined parameter, and it is used to control how fast the congestion costs are increased in the later iterations. In practice, this parameter is set empirically, based on a trade-off between solution qualities and execution times.

2.5.2 Incorporating Length Matching Objectives

In one iteration of the negotiated congestion routing algorithm, each routing grid cell has a fixed congestion cost value, as defined in Equation (2.9). The problem now is to find the best route and resource allocation for each net i, based on fixed congestion costs and fixed λ_{iL} and λ_{iS} values. Specifically, we want to find path P_i for each net i that minimizes the following expression:

$$\sum_{e \in P_i} (1 + \lambda_{iL} - \lambda_{iS} s^e + \epsilon (s^e)^2 + c^e) \tag{2.11}$$

where s^e is the number of extra grid cells allocated around edge e, and c^e is the total congestion cost of the grid cells *occupied* by edge e. Observe that this expression is obtained by incorporating congestion costs into the Lagrangian objective function (2.6), defined in Section 2.3.3.

To find the best path for net i, we model the routing grid as a graph using the model described in Section 2.4. Based on objective function (2.11), the weight of edge e is defined as:

$$weight(e) = \min_{s^e} \{1 + \lambda_{iL} - \lambda_{iS} s^e + \epsilon (s^e)^2 + c^e\} \tag{2.12}$$

As described in Section 2.4, some types of edges are not suitable for resource allocation. If e is such an edge, then s^e is set to zero, and c^e is set to the sum of the congestion costs of the two grid cells connected by this edge. Otherwise, s^e is selected[6] so as to minimize $weight(e)$ in Equation (2.12). Note that increasing s^e means allocating more grid cells, hence possibly increasing c^e. Here, the optimal value of s^e depends on the value of λ_{iS} (i.e., the importance of resource allocation constraint), and congestion costs of the grid cells around this edge. For this graph model and the weight function, we can state the following theorem:

Theorem 2.1 *Let $\mathcal{R}$ be the original routing grid, and let $\mathcal{G}$ be the corresponding graph model as defined in Section 2.4, and edge weights set based on Equation (2.12). The shortest path P in $\mathcal{G}$ corresponds to the best route (with resource allocation) in $\mathcal{R}$ that minimizes objective function (2.11).*

Proof. From Lemma 2.1, we know that there is a one-to-one correspondence between any valid route in $\mathcal{R}$, and any path in $\mathcal{G}$. Furthermore, consider an arbitrary path in the form: $v_0 \rightsquigarrow v_i \rightarrow v_j \rightsquigarrow v_n$, where v_0 and v_n are the terminal nodes, and v_i and v_j are any intermediate neighbouring nodes. According to the graph model given in Section 2.4, the types of vertices v_i and v_j completely determine whether space allocation is possible around the edge $(v_i \rightarrow v_j)$. Furthermore, changing the amount of space allocation around this edge does not affect the paths $v_0 \rightsquigarrow v_i$ and $v_j \rightsquigarrow v_n$, since all paths are defined to be monotonic in the horizontal direction (see Section 2.3.1). In other words, the *amount* of space allocation on a particular edge does not affect the solution for the rest of the path. So, the value s^e around any edge e must be selected as defined in Equation (2.12) to minimize objective function (2.11). ■

[6] In our implementation, we have defined a small preset upper bound value (e.g., 10) for s^e, and we tried each even number between 0 and this upper bound to find the optimal s^e value that minimizes the weight function defined.

```
ROUTE-ALL-NETS (Inputs: λ_iL, λ_iS values for each net i)
initialize congestion cost of each grid cell to zero
while a congestion-free routing solution not found do
    for each net i ∈ 𝒩 do
        calculate edge weights
        find min cost path for net i
    increase congestion costs of overused grid cells
```

Figure 2.7: Low-level algorithm description to route nets based on fixed Lagrangian multiplier values

After setting the edge weights, the next step is to find the minimum cost path for net i. Intuitively, defining edge weights as in Equation (2.12) has two important consequences. Shorter nets (with large λ_{iS} values) will automatically prefer the routes where they can allocate enough resources around. On the other hand, longer nets will probably not be detoured despite congestion costs, because λ_{iL} will dominate weight function (2.12) for small or moderate congestion levels. Closer examination of the edges illustrated in Figure 2.5 will reveal that our graph is in fact a dag (directed acyclic graph). It is known that the shortest path problem can be solved for a weighted dag in linear time [16].

The overall method described in this section is summarized in Figure 2.7.

2.6 Generalizing the Models

The models and algorithms in the previous sections mainly focus on routing a single bus on a single layer, and it is assumed that all routes are monotonic in one direction. However, it is straightforward to extend these ideas to more general cases.

For a multilayer layout, we can use a 3-D grid model, where the third dimension corresponds to interlayer connections. Here, the graph model proposed in Section 2.4, and the weight calculation scheme given by Equation (2.12) can be applied to each layer independently. However, the main difference here is in modeling interlayer connections. Assume that grid cells A and B are in different layers, and a via connection is possible between them. To model such a connection, we need to create edges between all subnodes of A and B. Since resource allocation is not applicable here, the

weight of these edges would only reflect the length requirement and congestion. For example, we can modify equation (2.12) for this purpose as follows:

$$weight(e_{via}) = (\lambda_{iL} d_{via} + c^e) \times via\_penalty \tag{2.13}$$

Note that an interlayer connection is likely to have different delay characteristics than a regular intralayer connection; so it might be necessary to use a circuit-dependent factor d_{via} to model such difference. Furthermore, since via connections are typically undesired, the constant *via_penalty* is used to avoid using these edges unless they are really necessary. After that, we can use our low-level routing algorithm on this 3-D grid structure. However, note that since this multilayer graph structure is not acyclic, we need to use a shortest path algorithm that can handle edges with negative weights,[7] such as Bellman-Ford algorithm [16]. Figure 2.8 shows a sample routing solution on 2 layers. Here, each of nets 2 and 3 is routed on a single layer, while net 1 uses a via to switch layers. Note that the only difference in the multilayer routing model is that a net can use a via to go in the third dimension during path calculations. By assigning a high cost to interlayer connections, we can avoid using vias if they are not really needed.

Also, we can extend these models for routing multiple buses together. For this, we need to modify the original formulation (2.1) such that each bus uses a different target length T. We can also extend this formulation to the most general case, where each net has different upper and lower bound constraints:

$$\begin{aligned} &minimize && \sum_{i \in \mathcal{N}} L_i \\ &subject\ \ to: && \\ & && \forall i, \quad L_i \leq T_i^{ub} \\ & && \forall i, \quad L_i + S_i \geq T_i^{lb} \end{aligned} \tag{2.14}$$

[7]Edges with negative weights are possible due to the weight function given in Equation 2.12. However, it is guaranteed that there is no negative-weight cycle, since we assume that each net has the same preferred direction in all layers. For example, if the preferred direction is RIGHT, then there will be no detour towards LEFT. Hence, a cycle cannot contain a horizontal edge. Since resource allocation is not possible around vertical edges, it is guaranteed that all edges in a cycle have positive weights.

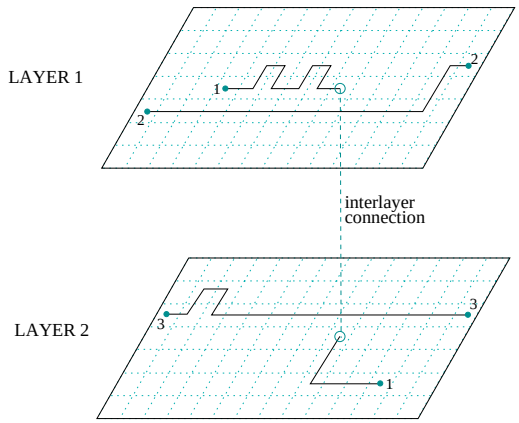


Figure 2.8: A sample solution with 3 nets routed on 2 layers. The interlayer connection for net 1 is illustrated with a dashed line. Note that snaking is performed on each layer the same way as in a single-layer model.

where T_i^{ub} and T_i^{lb} are the upper and lower bounds for net i. Note that such a modification in constraints would only effect the update schedule of Lagrangian multipliers. Namely, the multipliers for iteration $k+1$ would be calculated as follows:

$$\lambda_{iL}^{k+1} = \begin{cases} max(0, \lambda_{iL}^k - t_k(T_i^{ub} - L_i)^\gamma) & \text{if } L_i \leq T_i^{ub}, \\ \lambda_{iL}^k + t_k v_{iL}(L_i - T_i^{ub})^\gamma & \text{otherwise.} \end{cases} \tag{2.15}$$

$$\lambda_{iS}^{k+1} = \begin{cases} max(0, \lambda_{iS}^k - t_k(L_i + S_i - T_i^{lb})^\gamma) & \text{if } L_i + S_i \geq T_i^{lb}, \\ \lambda_{iS}^k + t_k v_{iS}(T_i^{lb} - L_i - S_i)^\gamma) & \text{otherwise.} \end{cases} \tag{2.16}$$

It is also possible to generalize our algorithms to the case where each net has an individual *preferred direction*, instead of a single global *preferred direction* for all nets. In other words, some nets can be specified as monotonic in the horizontal direction, while some others are monotonic in the vertical direction. Note that monotonicity of routes in one direction is required for the graph model we propose in Section 2.4, which ensures that all the extra routing resources allocated by our low-level routing algorithm can be used for length extension. In section 2.4, we have given a graph model for routes that are monotonic in the horizontal direction, and it is straightforward to generalize it for monotonicity in the vertical direction. Since our low-level routing algorithm (given in Figure 2.7) routes nets one by one, different graph models can be used for different nets, based on the prespecified *preferred directions*.[8] As an

[8]The preferred direction for a net can be determined heuristically based on the relative positions

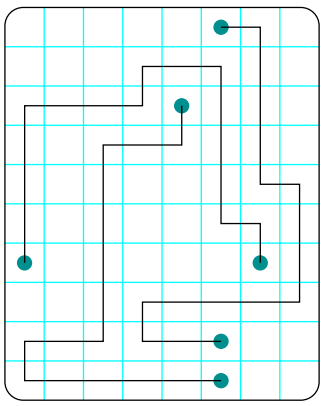


Figure 2.9: A sample routing solution where two nets are monotonic in the vertical direction, and one net is monotonic in the horizontal direction. The lengths of all three nets have been matched by our algorithm.

example, consider Figure 2.9, where two nets are monotonic in the vertical direction, and one net is monotonic in the horizontal direction. Our algorithm has successfully found the routing solution where all three nets have exactly the same length.

2.7 Experimental Results

In this section, we compare our framework with a commonly used greedy approach. In this approach, a negotiated-congestion routing algorithm (similar to Pathfinder [2; 3; 19]) is used to find a congestion-free routing solution for all nets. As described before, the main idea here is to route each net regardless of any congestion in the beginning; then the costs for congested routing resources are increased gradually, forcing the nets to use alternative routes. Note that this algorithm does not explicitly consider the objective of length matching during path calculations. Instead, after a conflict-free routing solution is found for all nets, a greedy post-processing method is used for the purpose of length matching. Here, each net is processed (from shortest to longest), and jogs are inserted (as in Figure 2.1) until it satisfies the min-length constraint.

During implementation of our algorithm, we have used a heuristic to expedite the convergence of the solution. Here, after all the nets are routed for fixed λ_{iL} and λ_{iS}

of its terminals. It is also possible to try both directions one by one, and then choose the one that gives the better route.

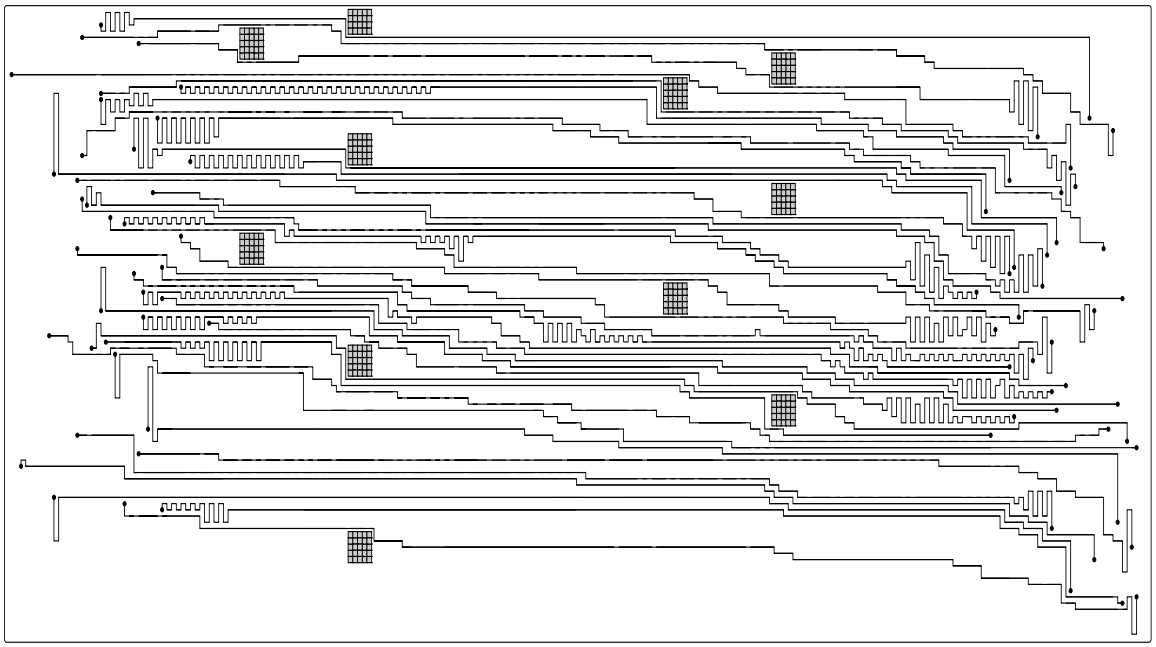


Figure 2.10: A sample routing solution using Lagrangian relaxation based resource allocation.

values in one iteration, it might be the case that some nets have more than necessary allocated resources. Our heuristic is to deallocate the extra resources from all these nets and to greedily allocate the available routing grids for shorter ones. We have observed that this heuristic decreases the running time of our algorithm.

We have implemented all these algorithms in C++, and we have performed our experiments on an Intel Xeon 2.4Ghz system with 512MB memory, and a Linux operating system.

For illustration purposes, we have applied our algorithm on a relatively small-sized single-layer bus routing problem,[9] the outcome of which is displayed in Figure 2.10. Here, there are various nets that are routed[10] with *almost* the same path lengths. Specifically, we have set the constant Δ in objective function (2.1) to 1 in our experiments.[11] In accordance with this constraint, the difference between the minimum and

[9] Although a single-layer routing solution is illustrated here, our algorithm applies equally well to multi-layer problems.

[10] We have not fine-tuned our program to reduce the number of bend points. However, if these are undesirable, it is possible to eliminate them in postprocessing.

[11] The length of a route can be extended only by an even number of grid cells. So, if there are two different nets (one with an even path length, one with an odd path length), their lengths can

Table 2.1: Properties of test problems

Test Problem	Vertical Spacing avg	Vertical Spacing stdev	Manhattan Dist. avg	Manhattan Dist. stdev	Grid size	Net count	Layer count
B1	3.54	2.25	106	14.43	150×356	99	one
B2	2.74	2.24	106	7.19	150×280	100	one
B3	2.66	1.72	107	17.31	150×261	96	one
B4	2.23	1.38	107	17.07	150×222	97	one
B5	3.29	2.64	117	15.05	150×459	133	two
B6	2.50	1.68	116	10.37	150×357	135	two
B7	2.30	1.43	117	18.47	150×325	133	two
B8	1.93	1.37	116	11.19	150×277	134	two
B9	1.81	1.18	117	15.74	150×231	118	two
B10	1.73	0.97	117	14.88	150×250	134	two

maximum path lengths is only one grid cell in the solution of Figure 2.10. Observe that snaking could be performed even in the dense areas of the layout. Furthermore, the heights of these jogs are usually small (i.e., 1 or 2 grid cells most of the time), mainly due to the methods proposed in Section 2.3.3 to avoid solution oscillations.[12] As a result, multiple nets effectively share the available routing resources such that all satisfy their min-length constraints.

We have also performed experiments on test problems properties of which are summarized in Table 2.1. Here, *vertical spacing* is measured in terms of the number of grid cells between the terminal points of adjacent nets, and it indicates how dense the problem is. On the other hand, *Manhattan distance* is given in terms of number of grid cells between two terminals of the same net. The deviation in this value is a good indicator for the amount of snaking needed to be performed. Each bus given in this table has around a hundred nets, and the objective is to route them and match their lengths. Note also that the underlying grid sizes are between 150×222 and 150×459, depending on the problem size. Similar to Figure 2.10, the nets in these problems are monotonic in the same direction. Furthermore the net terminals in the single-layer problems are ordered as in Figure 2.10 to ensure that a planar routing solution exists. On the other hand, the terminals in the two-layer problems are not ordered, since via usage is permitted for these problems.

be matched only up to 1 grid cell difference.

[12] As mentioned before, we use a preset upper bound value for s^e in function (2.12), effectively limiting the maximum height of a jog. However, in this figure the jogs have heights even smaller than this upper bound most of the time.

Table 2.2: Routing results on test problems

Test	GREEDY SNAKING				LR-BASED ROUTING			
Problem	minL	maxL	stdev	time	minL	maxL	stdev	time
B1	140	141	0.50	0:08	140	141	0.50	0:08
B2	99	127	2.78	0:11	121	122	0.50	0:19
B3	91	142	7.12	0:14	145	146	0.50	0:41
B4	66	150	10.71	0:12	145	146	0.50	3:27
B5	150	151	0.50	0:22	150	151	0.50	0:23
B6	109	140	3.04	0:25	139	140	0.50	0:47
B7	132	161	3.44	0:24	160	161	0.50	0:49
B8	103	147	6.78	0:24	144	145	0.50	4:25
B9	100	152	4.91	0:20	151	152	0.50	0:44
B10	96	152	7.75	0:25	151	152	0.50	9:46

As described before, our formulations involve some parameters due to the high-level Lagrangian relaxation framework (Figure 2.2) and the low-level negotiated congestion algorithm (Figure 2.7). In our experiments, we have set these parameters as follows. In the update schedule for Lagrangian multipliers given in Equations (2.3) and (2.4), we have set the step size t_k such that the convergence criterion given by Held et al[28] is satisfied, i.e., as $k \rightarrow \infty$, it should be the case that $t_k \rightarrow 0$ and $\sum_{i=1}^{k} t_i \rightarrow \infty$, where k is the current iteration number. Specifically, we have used the function $t_k = 1/\sqrt{k}$ for this purpose. As mentioned before, exponent γ in these equations is expected to have a small value to smooth the effect of the amount of length or resource constraint violations. So, we have set $\gamma = 0.1$ in our experiments. Similarly, we have set ϵ in the Lagrangian cost function (given in Equation (2.5)) to 0.1. Remember that the penalty term $\sum_{i \in \mathcal{N}} \sum_{e \in P_i} \epsilon(s^e)^2$ in this function has been introduced to avoid potential solution oscillations. Setting ϵ to such a small value makes the penalty term ineffective in earlier iterations, and dampens the oscillations as multiplier values start to converge to their optimal values, as described in Section 2.3.3. For the congestion cost function given in Equation (2.9), we have empirically set φ to 0.4. As discussed earlier, the main idea of the Pathfinder algorithm is to gradually increase the congestion costs of the overused grid cells. Here, parameter φ determines how fast the congestion costs are incremented. Finally, for the two-layer problems in Table 2.1, we have used Equation (2.13) to set the weights of via edges, as described in Section 2.6. In our experiments, we have empirically set the *via penalty* multiplier in this equation to 4 to discourage via usage.

We have executed both the greedy algorithm mentioned before and the Lagrangian relaxation (LR)-based routing algorithm on these test problems. The comparison of the results are given Table 2.2. Here, *minL, maxL*, and *stdev* denote the minimum path length, maximum path length, and standard deviation in path lengths, respectively. All results are given in terms of the number of grid cells spanned. Also, the execution times of these algorithms are given under columns *time*, and they are reported with *min:sec* units. Observe that the greedy method fails to match lengths especially when the problem is dense or when the variation in net lengths is large. However, our method performs multiple iterations in such cases to effectively find the solution that satisfies length constraints. Due to these multiple iterations, the execution time increases; nevertheless, the feasible solution is obtained eventually. In these experiments, we have observed that the high-level Lagrangian framework (as given in Figure 2.2) took less than 10 iterations most of the time. On the other hand, the low-level negotiated congestion algorithm (as given in Figure 2.7) took around 50-100 iterations. We have also observed that most of the nets are routed without congestion in the first few iterations; then the remaining few nets *negotiate* congested resources in the later iterations.

2.8 Conclusions

We have proposed an algorithm for routing nets within minimum and maximum length bounds. We can summarize our contributions in this chapter as (1) a high-level Lagrangian relaxation framework that guides the routing iterations such that length matching objectives are eventually satisfied, (2) incorporating the resource allocation objectives (which are guided by a Lagrangian function) into a state-of-the-art routing algorithm, (3) a special graph model $\mathcal{G}$ such that the shortest path in $\mathcal{G}$ corresponds to the optimal resource allocation for the current Lagrangian multipliers of a net. Our experiments indicate that our algorithm can be effectively used for routing nets with min-max length constraints, even in the situations where the greedy strategy fails to satisfy these constraints.

Chapter 3

A Two-Layer Bus Routing Algorithm for High-Performance Boards

3.1 Introduction

In this chapter, we focus on a more restricted yet common length matching problem: routing nets between component boundaries using two x-y signal layers. Here, the component boundaries define a routing channel, and all net terminals are assumed to be aligned on the opposite sides of this channel. We assume that each layer is assigned a primary routing direction of either horizontal or vertical. A sample routing solution is illustrated in Figure 3.1, where there are two bus structures: (1) a vertical bus between MCM and I/O module, and (2) a horizontal bus between MCM and memory module. Observe in the horizontal layer that the congestion in the area corresponding to the vertical problem (i.e., the area between MCM and I/O) is considerably lower than the congestion in the area corresponding to the horizontal problem (i.e., the area between MCM and MEM), and vice versa. The main reason is that the horizontal distance between terminals of a net in a horizontal problem corresponds to the distance between two different components, and it is typically much larger than the respective vertical distance.

Routability in a highly congested area is expected to be limited; so it will be more effective to perform length extension (to satisfy min-length constraints) within the less congested areas. For example, the vertical layer of a horizontal problem is

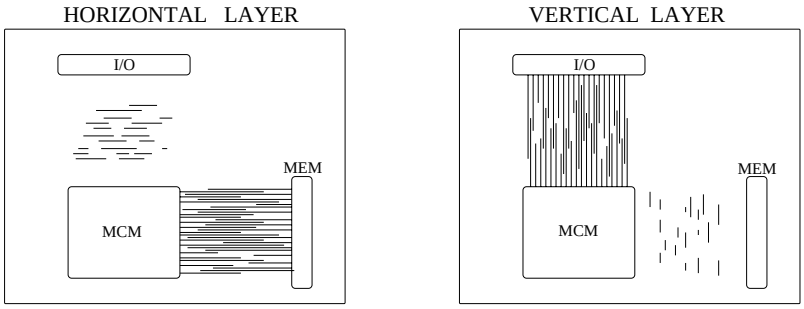


Figure 3.1: A typical two-layer routing solution. There are two separate bus structures here: (1) between MCM and I/O, and (2) between MCM and memory. No length extension (to satisfy min-length constraints) has been performed yet.

expected to be significantly less congested; so it makes more sense to perform length extension on this layer. In this chapter, we propose an algorithm that incorporates the objective of length extension into the actual routing algorithm. For a horizontal problem, our algorithm simultaneously extends the lengths of the nets and assigns them to vertical tracks.

The Lagrangian relaxation based length matching algorithm proposed in Chapter 2 can handle more general routing problems. However, the algorithm proposed in this chapter has some distinct advantages on its target class of problems. First of all, we route multiple nets simultaneously on one track in an optimal way, instead of using a net-by-net approach, which has no theoretical guarantees. Furthermore, the routing solutions are more uniform in this algorithm. That is, all nets use two vias, and the number of bends (due to length extension) is at most four for each net. Also, we consider a certain type of length extension methodology here, which is especially effective for this target class of problems.

The rest of the chapter is organized as follows. In Section 3.2, we describe the target problem in more detail, and discuss why simple ad hoc methodologies are not sufficient for this problem. Then, we propose an algorithm in Section 3.3 based on some assumptions about input circuits. In Section 3.4, we relax these assumptions, and discuss how to generalize this algorithm. Finally, we perform experiments in Section 3.5 to show the effectiveness of our algorithm compared to the Lagrangian relaxation framework.

3.2 Problem Formulation and Motivation

For a given set of nets $\mathcal{N}$, and min-max length constraints T_i^{min}, T_i^{max} for each net i, our purpose is to find a two-layer routing solution such that all length constraints are satisfied, and the routing resources are utilized most effectively. We assume that routing within dense components (escape routing) has already been accomplished[1] by the earlier stages of the routing system; hence all terminals are now aligned on the opposite sides of the channel. At first glance, this problem may seem similar to the traditional *channel routing* problem [7; 27; 55; 63], which has been studied extensively in the literature. However, the existence of min-max length constraints due to the high-speed design rules makes this problem significantly different from the traditional problem.

For simplicity of presentation, we will first focus on a restricted problem instance, where (1) each pair of adjacent terminals is separated by at least one grid cell on each side, and (2) no obstacles are found within the routing area. The algorithms we propose in Section 3.3 will be based on these assumptions; however, we will extend our algorithms in Section 3.4 for the general case.

All algorithms in this chapter will be presented for a horizontal problem (i.e., terminals are aligned on the left and right sides of the channel); however it is trivial to modify them for a vertical problem. Let us denote the horizontal routing layer as the *primary layer* and the vertical routing layer as the *secondary layer*. As mentioned before, since routing resources are very scarce on the primary layer, length extension will be performed on the secondary layer to satisfy all min-length constraints.

Figure 3.2 illustrates a sample routing solution for 12 nets, where the dashed lines indicate length extension performed on each net. Observe that layers 1 and 2 are primarily for routing horizontal and vertical segments, respectively. However small deviations from the primary directions are allowed on each layer. For instance, there are small diagonal segments (for alignment) on layer 1 and small horizontal segments (for length extension) on layer 2. Furthermore, the second layer is defined to consist of *vertical tracks*, where each track has width equal to the sum of via diameter and wire width (plus clearance between them). For example, five vertical tracks are used on layer 2 of this figure. Note that via diameters are typically much larger than wire widths, so the increase in track widths due to length extension is normally negligible.

[1] We propose such escape routing algorithms in Chapters 5 and 6.

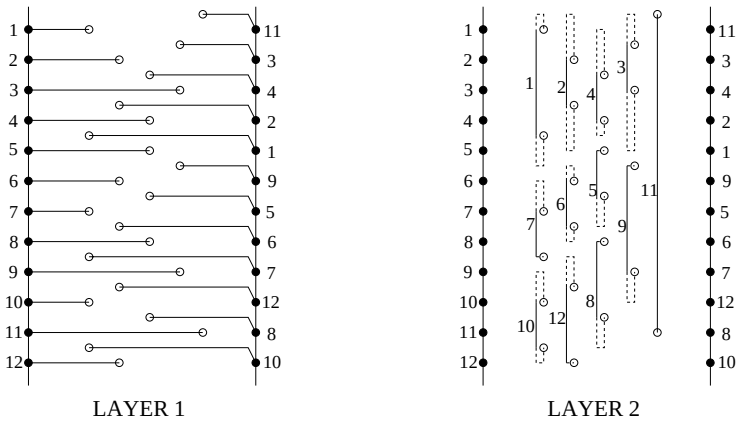


Figure 3.2: A sample routing solution on two layers, where each net has individual min-max length constraints. The terminals for 12 nets are aligned on the left and right side of the channel. Two vias (represented as empty circles) are used to route each net. The dashed lines on layer 2 indicate the length extension performed to satisfy min-length constraints.

It is important here to note that length extension needs to be performed simultaneously while determining the positions of vertical segments on the secondary layer. In the example of Figure 3.2, the number of vertical tracks used is kept minimum (i.e., 5 vertical tracks used on the second layer), and the routing resources are utilized most effectively. However, this utilization will be significantly reduced if length extension is performed as a separate step in the routing process. For instance, one can use a traditional channel routing algorithm (such as left-edge algorithm [27]) first to assign routing segments to the vertical tracks, and then extend the lengths of vertical segments in post-processing. Figure 3.3(a) shows the corresponding solution of the left-edge algorithm. Observe that segments have been assigned to vertical tracks without considering the min-length constraints. So, it is not guaranteed that there is enough space around each net such that its length will be successfully extended in post-processing. For instance, consider net 5 in this figure, which is assigned to the second vertical track between the segments of nets 1 and 7. Obviously, its length cannot be extended in post-processing (due to lack of space), and its min-length constraint will be violated. Specifically, there are four nets in this example of which length constraints will not be satisfied even after post-processing: nets 1, 5, 7, and 9. This example clearly demonstrates that min-length constraints need to be considered during the actual routing process, not just as a post-processing step.

Another approach here can be to extend the lengths of vertical segments using a

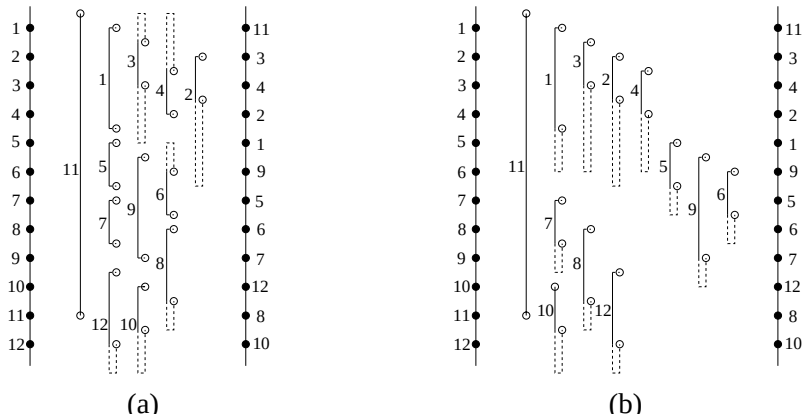


Figure 3.3: Alternative routing solutions corresponding to Figure 3.2, if (a) length extension is performed in post-processing, and (b) length extension is performed in preprocessing. In (a), min-length constraints of nets 1, 5, 7 and 9 are violated. In (b), the number of vertical tracks necessary increases to 8 (from 5). For clarity, only the results on the secondary layer are illustrated.

predefined pattern in preprocessing, and then to apply a traditional channel routing algorithm to assign them to vertical channels. Figure 3.3(b) shows such an example, where segments have been extended (from bottom) first; then the left edge algorithm has been applied on the extended segments. The disadvantage of this approach is that the routing algorithm has no control over the length extension process; so the resulting solution cannot utilize the routing resources most efficiently. In this example, eight vertical tracks are used to obtain a feasible solution, while Figure 3.2 shows that five tracks would be sufficient to satisfy all length constraints. This example shows that performing length extension as a preprocessing step is also not an effective strategy.

The algorithms we propose in this chapter handle length extension and track assignment simultaneously, so that a feasible routing solution is obtained while using a minimum number of vertical tracks. For instance, observe in Figure 3.2 that net 1 is extended both from top and from bottom, and such an extension allows three nets to fit on one track. The next section describes our models and algorithms in more detail.

3.3 Algorithm Description

3.3.1 Routing Model

Routing on the horizontal layer is straightforward, because of the assumptions that there are no obstacles in the routing area and that each adjacent pair is separated by at least one grid cell (see Section 3.4 for the general case without these assumptions). As illustrated in the example of Figure 3.2, a horizontal connection[2] is possible from each terminal on one side of the channel to the other side, without any conflicts with others. So, the main problem here is to determine the positions of vertical segments on the secondary layer. Once the positions of these vertical segments are fixed, the horizontal segments on the first layer can be connected to them using vias, as illustrated in this example.

Figure 3.4 shows an example illustrating the way length extension is performed on vertical segments. Here, assume that we need to extend the length of the segment in part (a) by 16 units (in terms of grid cells) to satisfy its min-length constraint. Figure 3.4(b) shows eight possible configurations, each of which is the extended version of the original segment. As mentioned before, one via and one wire is defined to fit on a *vertical track* together; so each extended segment in this figure is defined to be on a single track. Figure 3.4(c) gives a simpler representation, where a solid line represents the original segment and a dashed line represents the extended length.

Min length constraint for a net directly determines the minimum length requirement for its vertical segment. Let x_i denote the amount of length extension required to satisfy min length constraint of net i. The value of x_i is simply equal to the Manhattan distance between net i's terminal points subtracted from its min length constraint. Here, the vertical segment of net i must be extended by at least $x_i/2 - 1$ from top or from bottom, as in Figure 3.4(c). In the example of Figure 3.4, x_i is given as 16, and the vertical segment of net i needs to be extended by at least 7 units.[3] These concepts are formalized by the following definitions:

Definition 3.1 *The routing solution for net i is defined based on the position of its*

[2]A small diagonal segment might be necessary to align the horizontal segments on opposite sides, as shown in Figure 3.2.

[3]As shown in Figure 3.4(b), the extended part actually consists of two adjacent vertical wire segments and one unit of horizontal wire segment. However for simplicity, we represent it as a single wire as in part (c).

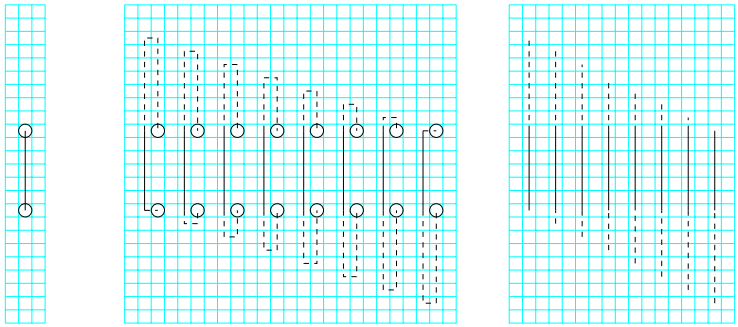


Figure 3.4: (a) A vertical segment that needs to be extended by 16 units. (b) Eight possible configurations corresponding to the extended segment. (c) Each configuration is represented as a single line, where dashed lines represent the length extension.

vertical segment, and it is denoted as $R_i = (t_i, r_i, \ell_i)$*, where* t_i *is the track number,* r_i *is the top row, and* ℓ_i *is the length of the vertical segment of net* i*. Here,* ℓ_i *is determined directly from the min length constraint of net* i*, as discussed before. Furthermore,* r_i *must be chosen such that* $r_i^{min} \leq r_i \leq r_i^{max}$*, where* $r_i = r_i^{min}$*, and* $r_i = r_i^{max}$ *correspond to the extreme cases where no length extension is performed from the bottom, and from the top, respectively. The main idea is illustrated in Figure 3.5.*

Definition 3.2 *The routing problem for a given set of nets is defined as finding a solution* $R_i = (t_i, r_i, \ell_i)$ *for each net* i *such that (1) no two vertical segments on the*

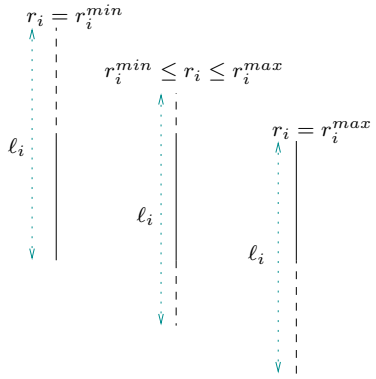


Figure 3.5: Three different cases for the vertical segment of net i are illustrated. Here, length ℓ_i is fixed, since it is determined by the min-length constraint. However, the top row r_i can vary between r_i^{min} and r_i^{max}. The solid and dashed lines here represent the original and extended segments, respectively.

same track overlap with each other, and (2) the number of vertical tracks used is minimized.

Note that the min length constraints are captured by the target length ℓ_i defined for each net i, while the max length constraints do not need to be considered explicitly. The reason is that each net is routed using the minimum possible length in this algorithm.

3.3.2 Algorithm Proposed

The problem defined by Definitions 3.1 and 3.2 is actually a special case of the *task scheduling problem with release times and deadlines on a multi-computer*. Here, each vertical track can be considered as a computer; each vertical routing segment can be considered as a task with length ℓ_i, release time r_i^{min}, and deadline r_i^{max}. Note that this problem is known to be an NP-complete problem in the strong sense, even in the case where there is a single computer [4]. However, the special property of our problem will allow us to give a polynomial-time exact algorithm for the single-track case.

Here, our approach will be to process one track at a time and to *pack* as many routing segments as possible on each track. Note that the best routing configuration for each net should also be determined simultaneously during this process. The following definition gives a formal description of this objective.

Definition 3.3 *The problem of single-track assignment is defined as follows: Given a set of nets $\mathcal{N}$, and a set of vertical segments for each net i in $\mathcal{N}$, the objective is to select a subset of these vertical segments, such that (1) at most one vertical segment is selected corresponding to each net i, (2) the selected segments do not overlap with each other, and (3) maximum resource utilization is achieved on one track (i.e., the number of grid cells unused is kept minimum).*

As an example, consider Figure 3.6(a), where there are four different nets, and each net has multiple routing configurations. The corresponding optimal solution for single-track assignment is shown in Figure 3.6(b). Observe that 21 out of 22 grid cells have been utilized on this track.

As mentioned above, this problem is a special case of the task scheduling problem on a single computer, which is an NP-complete problem in the strong sense. However,

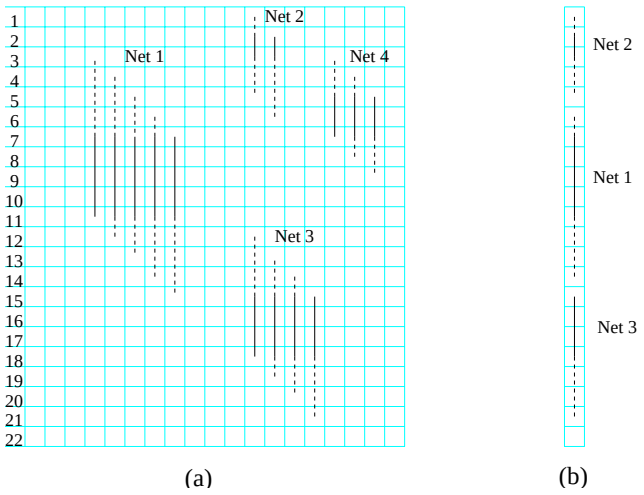


Figure 3.6: (a) A sample single-track assignment problem with 4 nets. Multiple routing configurations are illustrated for each net. (b) The optimal solution, which utilizes 21 out of 22 grid cells of one track. The dashed lines indicate the length extension performed.

we propose an algorithm in Figure 3.7, which is guaranteed to find the optimal solution in polynomial time (due to the special property that will be given in Lemma 3.2). Here, the main idea is to represent each row of the track as a vertex, and to model each vertical segment as a zero-weight edge between the respective rows. Furthermore, there is a unit-weight edge from each $v[j]$ to $v[j+1]$, which corresponds to the case where the grid cell on row j is unused. Then, the shortest path from the first row to the last row is computed to find the optimal assignment with the maximum resource utilization. Intuitively, the weight of an edge from $v[k]$ to $v[m]$ indicates the number of grid cells that will be wasted between rows k and $m-1$ if this edge is selected. So, the shortest path from the top row to the bottom row will give us the assignment with the minimum waste. Figure 3.8 illustrates the graph model corresponding to the problem given in Figure 3.6(a). The highlighted path in this graph is the shortest path, and it corresponds to the optimal solution in Figure 3.6(b). For example, the edge from $v[1]$ to $v[6]$ on the shortest path corresponds to the vertical segment of net 2 from row 1 to row 5. The formal analysis of this algorithm is given as follows.

Lemma 3.1 *Consider any pair of edges $e_i, e_j \in \mathcal{G}$. If there exists a path P such that $e_i, e_j \in P$, then the vertical segments corresponding to e_i and e_j are guaranteed not to overlap with each other.*

SINGLE-TRACK-ASSIGNMENT ($\mathcal{N}$: set of nets, t: current track)

create a graph $\mathcal{G}$ as follows:

  for each row j of track t

    create a vertex $v[j]$

    add a unit-weight edge from $v[j-1]$ to $v[j]$

  for each net i in $\mathcal{N}$

    for $r_i = r_i^{min}$ to r_i^{max} do

      create a zero-weight edge from $v[r_i]$ to $v[r_i+\ell_i+1]$

Compute the shortest path P from v[1] to v[last] in $\mathcal{G}$

for each edge $e \in P$

  if e is a zero-weight edge from $v[k]$ to $v[m]$

    select the vertical segment that spans rows from k to m-1

Figure 3.7: Algorithm for selecting the maximal subset of non-overlapping vertical segments

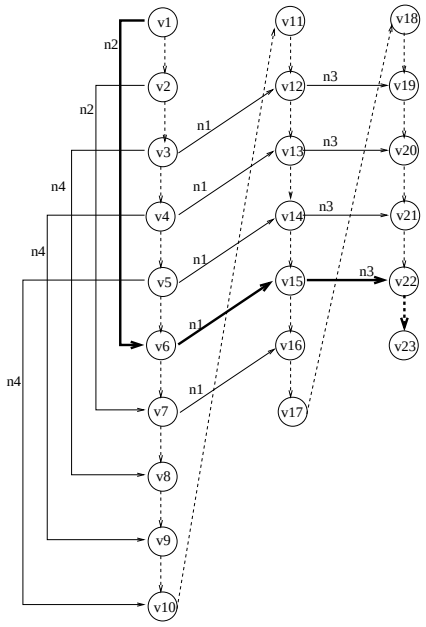


Figure 3.8: The graph model corresponding to the problem of Figure 3.6(a). The edges corresponding to net segments (solid arrows) have zero weights, while the others (dashed arrows) have unit weights. The shortest path (with total weight 1) is highlighted, and it corresponds to the optimal solution in Figure 3.6(b).

PROOF. The direction of edges in $\mathcal{G}$ are always towards larger vertex indices. Furthermore, for a vertical segment that spans rows k to m, the corresponding edge will be from $v[k]$ to $v[m+1]$. Hence, any edge selected after this edge will correspond to a segment starting from row $m+1$. So, any pair of edges in a path cannot correspond to overlapping net segments. ■

Lemma 3.2 *Consider the set of edges E_n corresponding to net n. There exists no path P in $\mathcal{G}$ such that $e_i, e_j \in P$ and $e_i, e_j \in E_n$. In other words, a path cannot contain two edges corresponding to different vertical segments of the same net.*

PROOF. There are different vertical segments corresponding to net n, because there are different ways of extending the length of n. However, as can be seen in Figure 3.5, the original segment (represented with solid lines) is always fixed; hence all vertical segments corresponding to the same net will overlap with each other. Also, from Lemma 3.1, a path cannot contain edges corresponding to overlapping net segments. ■

Theorem 3.1 *The shortest path in $\mathcal{G}$ between the first and last vertices corresponds to the optimal solution of the single-track assignment problem defined in Definition 3.3.*

PROOF. From Lemmas 3.1 and 3.2, the set of edges on any path corresponds to a valid assignment on one track. Furthermore, there is a path in $\mathcal{G}$ corresponding to any valid assignment on one track. Since the unit weighted edges in $\mathcal{G}$ correspond to the unused rows of the track, the total weight of path P will be equal to the number of rows wasted. So, the shortest path from the top row to the bottom row will correspond to the optimal assignment with maximum utilization. ■

Theorem 3.2 *Let H denote the number of rows in the channel, and x_i denote the amount of length extension required to satisfy min-length constraint of net i. The time complexity of the algorithm given in Figure 3.7 is $O(H + \sum_{i \in N} x_i)$.*

PROOF. There are $O(x_i)$ different vertical segments defined for each net. So, the number of edges in $\mathcal{G}$ is $O(H + \sum_{i \in N} x_i)$, while the number of vertices is $O(H)$. Furthermore, $\mathcal{G}$ is a directed acyclic graph, and the shortest path can be computed in linear time [16]. ■

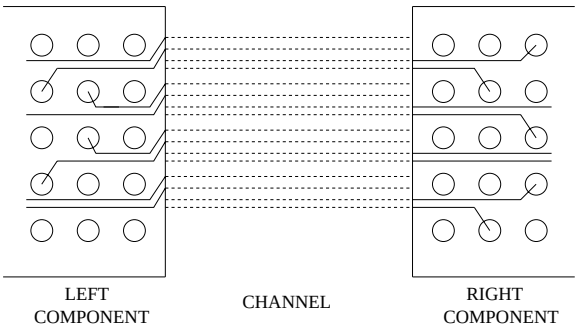


Figure 3.9: In a typical board routing problem, nets escape from dense components (solid line segments), and the input to the bus routing problem is defined as a set of terminals aligned on the opposite sides of a channel. Since the diameters of the pins within components are much larger than wire widths, these terminals are typically well-separated. So, it is possible to align all horizontal segments (dashed lines) such that there are no overlaps between them.

3.4 Generalization of the Algorithm

In the previous section, we assumed that each pair of adjacent terminals is separated by at least one grid cell. However, this is not absolutely necessary, as long as it is possible to extend the horizontal segments from one side of the channel to the other side without any conflicts, as illustrated in Figure 3.9. If this is the case, then there will be no restrictions on the positions of the vertical segments, and the same algorithm in Section 3.3 can be used without a change. Note that this assumption is reasonable for a typical industrial circuit, since the pin diameters within chip components are much larger than the wire widths; so there will be enough routing space to align horizontal segments from both sides without any overlaps (as in Figure 3.9).

Actually, this corresponds to the unrestricted case of the original channel routing problem [27], where there are no vertical constraints; i.e., net segments can be assigned to tracks without any ordering constraints. On the other hand, if overlaps are possible on the horizontal layer, then we need to define *pin constraints* to avoid overlaps. For instance, assume that the horizontal segment of net i originating from a left terminal overlaps with the horizontal segment of net j originating from a right terminal. In that case, the vertical segment of net i must be assigned to a track which is to the left of the vertical segment of net j to avoid an overlap on the horizontal layer. Note that if there were no min-max length constraints, this would correspond to the problem of *channel routing with vertical constraints*, which has been studied in the literature.

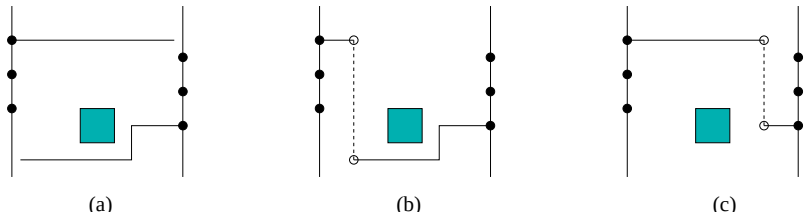


Figure 3.10: (a) The horizontal segments of a net are not entirely straight due to an obstacle. (b) The vertical segment is assigned on a track to the left of the obstacle. (c) The vertical segment is assigned on a track to the right of the obstacle. The solid and the dashed lines represent the routing segments on the horizontal and vertical layers, respectively. For clarity, only one net is displayed, and length extension on the vertical layer is not shown.

This problem has been shown to be NP-complete [36; 56]; however there have been several algorithms proposed that give sufficiently good results [7; 20; 55; 63]. If the assumption of well-separated terminals (mentioned above) is not valid for a circuit, we can use similar ideas to extend our algorithm to the general case. In particular, we can define pin constraints indicating the ordering of the vertical segments, and then perform track assignment based on this ordering. Since our algorithm processes one track at a time, we can simply consider the set of nets that do not violate the ordering constraints for the track that is being processed.

We can also generalize our algorithm to the case where there are some obstacles in the routing region. If the obstacles are on the horizontal layer, then the horizontal segments will not be entirely straight, as shown in Figure 3.10(a). Furthermore, the y coordinates of the vertical segments will depend on the track on which it is assigned, as illustrated in parts (b) and (c) of the same figure. Since the algorithm we propose processes one track at a time, the appropriate vertical segments corresponding to each net can be determined for each track, and the algorithm given in Figure 3.7 can still be used to choose the best subset. On the other hand, we can handle the obstacles on the vertical layer by simply removing the edges corresponding to vertical segments that overlap with an obstacle on the current track.

3.5 Experimental Results

We have performed experiments to compare the two-layer routing algorithm proposed in this chapter with the Lagrangian relaxation (LR) based methodology of Chapter 2.

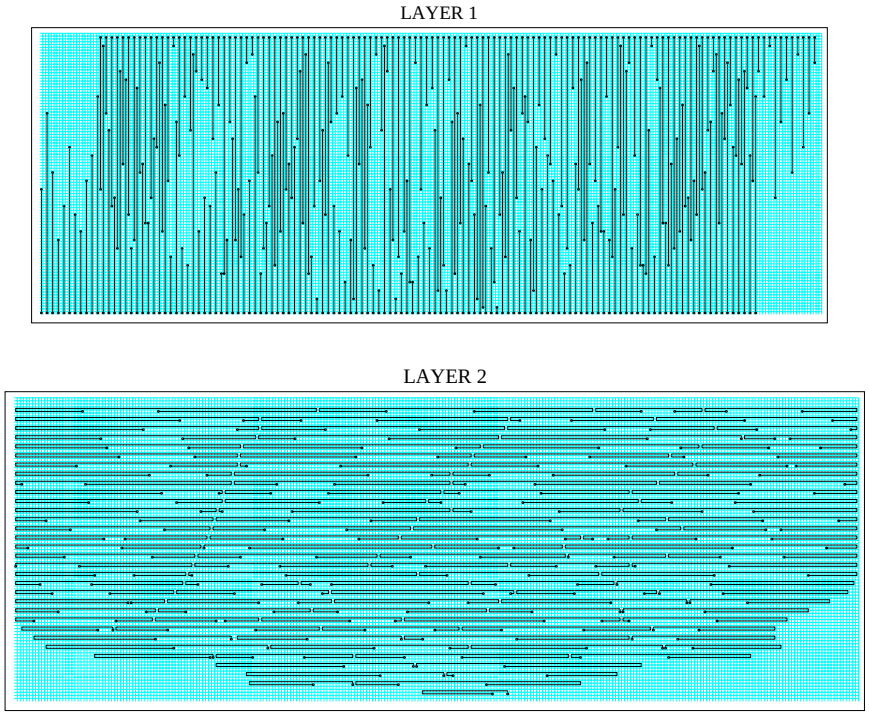


Figure 3.11: A sample two-layer solution for 128 nets. A vertical problem is illustrated here; hence length extension is performed on the horizontal (second) layer. The length constraints for all nets have been satisfied in this solution.

All algorithms in this section have been implemented in C++, and experiments were performed on an Intel Pentium 4 2.4GHz system with 1GB memory, and a Linux operating system.

A sample output of this two-layer routing algorithm is illustrated in Figure 3.11, for a routing problem with 128 nets. Here, each net has individual length constraints, and terminals are aligned on the top and bottom sides of the channel. Since this is a vertical problem, length extension is performed on the horizontal (second) layer. Observe that nets have been assigned to tracks and their lengths have been extended so that maximum resource utilization is achieved on each track.

The experiments we have performed on test problems are given in Table 3.1. Here, "*avg. spacing*" is measured in terms of the number of grid cells between terminal points of adjacent nets, and it indicates how dense the problem is. On the other hand, columns "*length avg*" and "*length stdev*" give statistical information about net target lengths. Each problem in this table contains between 100 and 300 nets, with individual length constraints for each net. The grid size for the smallest circuit in

Table 3.1: Comparison of the two-layer routing algorithm proposed in this chapter with the Lagrangian relaxation based methodology.

				TWO-LAYER		LR-BASED	
Input Prob.	Avg. spacing	Length avg	Length stdev	# nets failed	time (m:s)	# nets failed	time (m:s)
B1	3.40	150.1	39.9	0	0:01	5	38:21
B2	3.19	138.5	18.6	0	0:01	6	44:52
B3	3.46	151.6	40.1	0	0:01	21	57:06
B4	2.91	160.8	38.0	0	0:01	4	46:32
B5	3.16	183.9	30.0	0	0:01	19	52:50
B6	2.94	174.7	18.1	0	0:01	48	52:45
B7	2.12	157.4	20.7	0	0:01	45	73:50
IBM1	7.75	417.6	35.8	3	0:01	3	2:29
IBM2	6.41	382.0	46.0	1	0:01	1	1:50
IBM3	9.16	427.6	73.8	0	0:01	0	7:06

this table is 100×330, and the largest one is 290×776. The last three problems here have been extracted from an IBM design, corresponding to the bus routing problems between MCM, memory and STI modules. Here, layer assignment and routing inside chips have been performed by the previous phases of the routing system; so the input for the bus routing problem is a set of non-crossing nets on each layer. While the first seven circuits in this table have a single layer pair, the IBM circuits have multiple layer pairs.

The results in this table indicate that the two-layer routing algorithm performs significantly better than the LR-based approach on most circuits, in terms of both quality and run time. The solution quality for LR-based approach degrades especially when the average spacing between nets decrease, or the target lengths increase (hence more aggressive length extension required). The main reason for this is that the LR-based approach uses a variant of Pathfinder [3; 19] algorithm in the low level, where routing conflicts are resolved through *negotiations*. As the problems get denser, these negotiations take more and more time, and they do not always successfully lead to a good result. On the other hand, this two-layer routing algorithm performs length extension in a fast and effective way on its target class of problems.

3.6 Conclusions

We have proposed a routing algorithm with the objective of satisfying length constraints of high-speed printed circuit boards. The main idea is to perform length extension on the secondary layer (e.g. vertical layer for a horizontal problem), where routing congestion is typically much lower than the primary layer. We have proposed an optimal algorithm to select the best subset of nets to assign to a single track, while satisfying the length constraints. Our experiments show that compared to the more general Lagrangian relaxation framework of Chapter 2, this algorithm performs considerably better on its target class of problems.

Chapter 4

An Algorithmic Study of Single-Layer Bus Routing For High-Speed Boards

4.1 Introduction

In this chapter, we focus on board designs that do not use any buried vias, due to high manufacturing costs. For such designs, each net needs to be routed on a single layer in a planar fashion. Similar to the problem of Chapter 3, we assume that boundaries of the components define a routing channel, and all net terminals are aligned on the opposite sides of the channel. The objective is to route all nets on a single layer such that each net satisfies its prespecified min-max length constraints.

Figure 4.1 illustrates this problem with an example. Here, we assume that layer assignment has already been performed and that all nets have been routed from their individual pins to chip boundaries.[1] The problem is to route nets between pairs of components such that all nets belonging to the same bus have *approximately* the same length. In the example of Figure 4.1, two separate bus structures are displayed. However, in reality there may be more than one bus structure interleaved with each other, or there may be individual nets not belonging to any bus. For this reason, we will focus on the general problem, where each net has individual min and max length

[1]Since via usage is not allowed, the previous phases of the routing system make sure that net ordering within a single layer is compatible among different components. We propose such escape routing algorithms in Chapters 5 and 6.

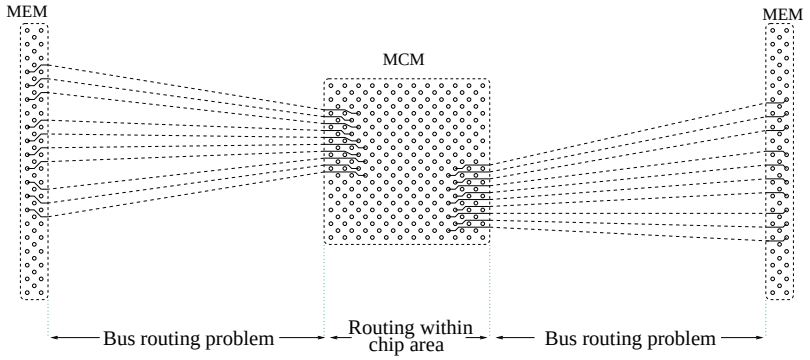


Figure 4.1: Two separate bus structures are displayed between MCM and memory modules of a sample board.

constraints.

Actually, this problem is similar to the *river routing* [39] problem, in the sense that all terminal points are aligned with each other on two opposite sides of the circuit, and a single-layer routing solution is desired. River routing has been studied extensively in the literature [30; 38; 50; 51; 65]. A common application for river routing has been routing bus structures across a channel [30]. Today, we face a similar problem for high-performance bus routing, with the additional constraint that each net must be routed within min-max length constraints due to very high clock frequencies.

In this chapter, we first define the problem of *min-area max-length routing* in Section 4.2. The objective here is to route each net within its max-length constraint, while allocating at least a prespecified amount of routing area around it. Then in post-processing, snaking can be performed within this area to extend the lengths of *short* nets. Intuitively, shorter nets belonging to a bus need to allocate more area around their routes so that length matching will be possible in the end. We propose a linear-time optimal algorithm for this problem in Section 4.2. Then, we extend this algorithm in Section 4.3 to solve a general river routing problem with min-max length constraints, and we prove that this algorithm is near-optimal. Finally in Section 4.4, we perform experiments to compare these algorithms with the Lagrangian relaxation methodology.

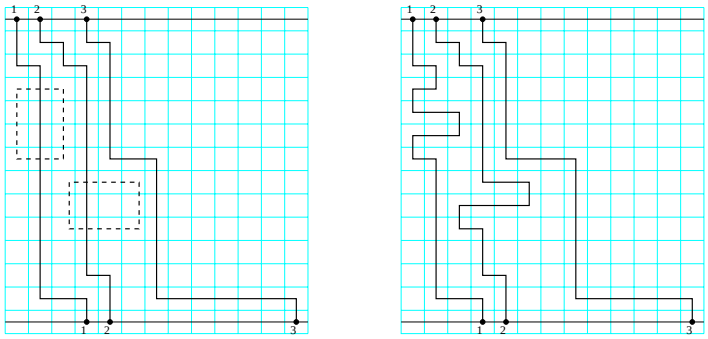


Figure 4.2: (a) A sample routing solution with area allocation, where dashed lines represent the allocated areas around routes. (b) The final routing solution, where shorter nets have extended their lengths using the allocated areas.

4.2 Min-Area Max-Length Routing

4.2.1 Problem Formulation

Our objective here is to find a routing solution, where each net has some extra space allocated around its route. The idea is that if a short net allocates *enough* routing resources around itself, it is possible to extend its length in post-processing using those extra resources. Figure 4.2(a) gives a sample routing solution, where shorter nets have allocated extra routing areas around their routes. Figure 4.2(b) illustrates how those areas can be used for matching the lengths of all three nets. Based on this idea, we assume that each min-length constraint (T_i^{min}) can actually be given as a min-area constraint (A_i^{min}), where area of route i (A_i) is defined as the total number of grid cells allocated by route i.

This problem can formally be stated as follows: Given a set of nets $\mathcal{N}$, and min-area (A_i^{min}), max-length (T_i^{max}) constraints for each net i, find a single-layer routing solution such that $A_i \geq A_i^{min}$, and $T_i \leq T_i^{max}$, where A_i, T_i denote the area and the length of route i, respectively.

Here, we assume that there is an underlying routing grid where routes go center-to-center of grid cells. As mentioned in the previous section, we also assume that all terminal points are aligned on two opposite sides of the circuit. For simplicity of presentation, we will give our algorithms for the case where terminal points are at the topmost and bottommost rows of the grid.

ROUTE-WITH-MIN-AREA-MAX-LENGTH-CONSTRAINTS
find the leftmost boundary L_i for each net i
find the rightmost boundary R_i for each net i
for each net i from left to right
    while Route(L_i, L_{i+1}) does not satisfy min-area constraint
        flip an *appropriate* corner of L_{i+1} rightwards

Figure 4.3: High-level algorithm for routing problem with min-area max-length constraints

4.2.2 An Optimal Algorithm

The high level description of our algorithm is given in Figure 4.3. The algorithm starts with finding the leftmost boundary (L_i) and the rightmost boundary (R_i) for each net i. Here, L_i and R_i define the interval within which net i must be routed. These boundaries depend on (1) the boundaries to the left and right of net i, and (2) the maximum detour allowed for net i due to its max-length constraint. Figure 4.4 illustrates the idea for left boundaries. The right boundaries can also be found similarly. Note here that, a boundary L_i follows L_{i-1} to the left as long as max-length constraint of net i is not violated. For example, L_5 in Figure 4.4 stops short of following L_4 due to the max-length constraint of net 5.

After finding the initial positions of all left and right boundaries, the algorithm attempts to find a valid route for each net, starting from the leftmost one. At any point in time, route of net i is defined based on L_i and L_{i+1} as follows: *The main route of net i follows the trail of L_i as close as possible; and all the remaining grid cells between L_i and L_{i+1} are allocated by net i as extra routing area.* Since the left and right boundaries are defined based on the max-length constraints, it is guaranteed that any route within those boundaries will satisfy max constraints. So, the algorithm checks only the min-area constraints. The strategy here is to incrementally *flip* the left boundary of the right neighbor until the area constraint of the current net is satisfied.

Before giving details of the *flip* operations, we need to define some properties for boundaries.

Definition 4.1 *A boundary is defined to be* monotonic *if its trail contains no detour, and* nonmonotonic *otherwise. A monotonic boundary can be either* falling *or* rising,

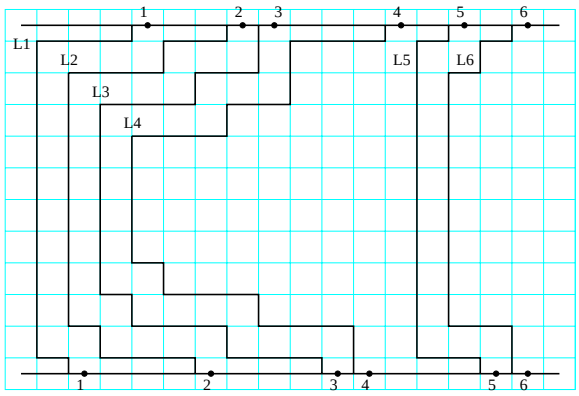


Figure 4.4: Left boundaries L_1-L_6 for six nets. Terminal points of the nets are shown as filled circles. A feasible route for net i cannot cross L_i at any point.

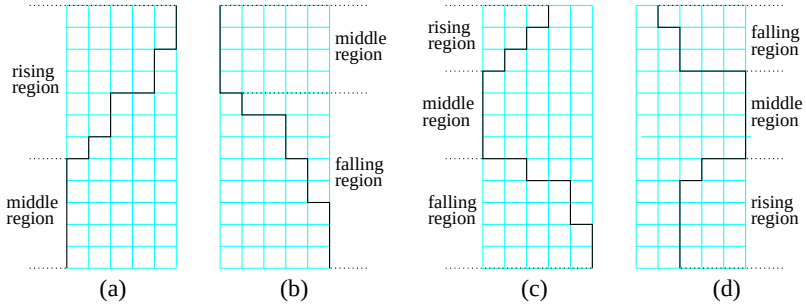


Figure 4.5: Types of boundaries: (a) rising monotonic, (b) falling monotonic, (c) concave nonmonotonic, and (d) convex nonmonotonic.

depending on the relative positions of its terminals. A nonmonotonic boundary can be either concave *or* convex*, depending on the direction of the detour. A boundary is defined to have three regions:* rising*,* middle*, and* falling *regions.*

These concepts are illustrated in Figure 4.5. Note that a rising monotonic boundary has an empty falling region, and vice versa. Due to the algorithm we use, a nonmonotonic boundary cannot have detours in two different directions at any point in time; so this case is not defined. It is also possible to show that a left boundary cannot be convex nonmonotonic in the beginning of the algorithm. Similarly, a right boundary cannot be concave nonmonotonic.

Based on Definition 4.1, we can define the operation *flip* L_i as follows:

- *If* L_i *is rising monotonic*: Flip the top corner of the leftmost segment of L_i that is not adjacent to R_i. If no such corner exists, see the special case below.

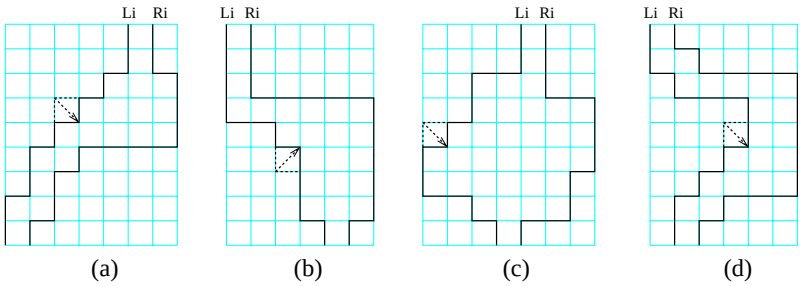


Figure 4.6: Next *flip* on left boundary L_i if L_i is (a) rising monotonic, (b) falling monotonic, (c) concave nonmonotonic, and (d) convex nonmonotonic. The dashed lines illustrate the *flip* operation. The final L_i is shown with solid lines.

- *If L_i is falling monotonic*: Flip the bottom corner of the leftmost segment of L_i that is not adjacent to R_i. If no such corner exists, see the special case below.
- *If L_i is concave nonmonotonic*: Flip the top corner of the middle region. Since R_i is guaranteed not to be concave nonmonotonic, such a flip will always be possible.
- *If L_i is convex nonmonotonic*: Consider the leftmost segment of L_i that is not adjacent to R_i. If this segment is in the rising region of L_i (as in Figure 4.6(d)), then flip its top corner. If it is in the falling region of L_i, then flip its bottom corner. If it is in the middle region, then see the special case below.

Figure 4.6 illustrates each case with an example. Recall that the algorithm given in Figure 4.3 processes nets from left to right, and L_i is flipped to allocate more area for net $i-1$. It is obvious that L_i cannot be pushed on or beyond any segment of R_i, since that would make it impossible to route the next net.

There are 3 special cases mentioned above that need to be handled separately. These are illustrated in Figure 4.7. Observe that we can flip the rightmost segments either from the top or from the bottom. We have to make this decision so that the following invariant is maintained throughout the execution: *Each boundary is one of the following: (1) rising monotonic, (2) falling monotonic, (3) concave nonmonotonic, and (4) convex nonmonotonic.* Figure 4.8(b) shows an example where this invariant is violated because of an incorrect decision. On the other hand, if the flip is performed from the bottom as in part (c), then the invariant is successfully maintained. Note that flipping L_i modifies some of the boundaries L_j $(j > i)$ to the right of L_i, as in the example of Figure 4.8. Here, we need to check the invariant for all boundaries

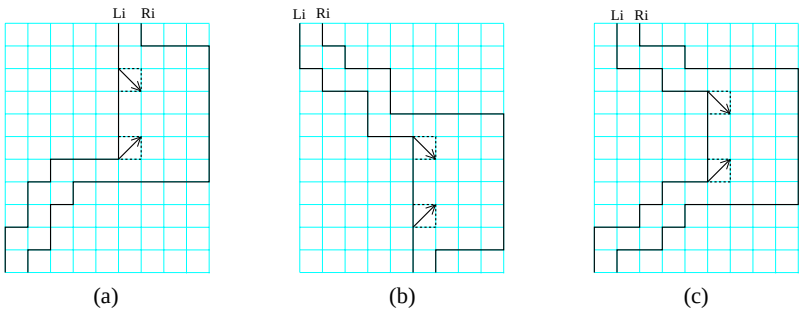


Figure 4.7: Special cases corresponding to (a) rising monotonic L_i, (b) falling monotonic L_i, and (c) convex nonmonotonic L_i. The dashed lines illustrate 2 alternative *flip*s for each case.

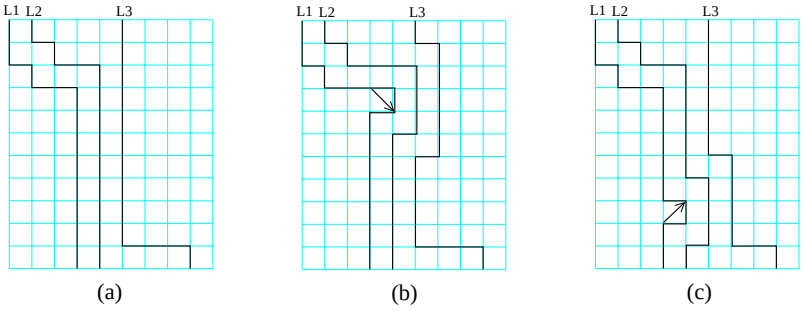


Figure 4.8: Illustrating two alternative *flip* operations for a special case: (a) A flip is to performed at the rightmost segment of L_1. (b) Flip is performed from the top, and L_3 violates the invariant. (c) Flip is performed from the bottom, and the invariant is maintained.

between L_i and L_k, where k is the smallest value such that $k > i$, and L_k will not be convex nonmonotonic after the flip. Note that the boundaries to the right of L_k do not need to be checked, because only convex nonmonotonic boundaries can cause others (to the right) to violate the invariant.

A sequence of flips on an initially *concave nonmonotonic* boundary L_i is illustrated in Figure 4.9 as a general example. As mentioned above, it is always possible to flip the top corner of a concave nonmonotonic boundary. So, a sequence of flips are performed on L_i, until it becomes *falling monotonic*. Then, it stays monotonic until all its segments except the rightmost one are pushed adjacent to R_i. After that, it becomes *convex nonmonotonic*, and its middle region is flipped until there remains a trail only with a single width for net i.

The example of Figure 4.9 suggests that we do not need to perform flips only one unit at a time, as shown in Figure 4.6. Instead, we can use two binary searches to

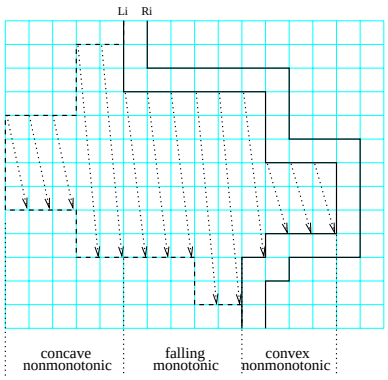


Figure 4.9: A sequence of flips on left boundary L_i is illustrated. The initial and final positions of L_i are shown with dashed and solid lines, respectively. Each columnwise flip is shown as a dotted arrow. The intervals within which L_i is monotonic or nonmonotonic are also specified.

find the minimum number of flips necessary to satisfy the min area requirement of the current net. Here, the first binary search performs flips one column at a time (as in Figure 4.9) to find the minimum number of columnwise flips. Then, the second binary search finds the minimum number of single-unit flips necessary on the last columnwise flip.

The time complexity of this algorithm is $O(A)$, where A is the area of the channel. The operations of finding the left and right boundaries for all nets dominate this time complexity. Note that we need $O(logA_i)$ iterations in the binary search described above to route net i, where A_i denotes the total area between L_i and L_{i+1}. The area constraint for net i can be checked in constant time in each iteration.[2] Note also that checking the invariants for special cases (as in Figure 4.7) does not affect the time complexity. The main reason is that we need to make this check only once for each net: for the last partial column.

4.2.3 Proof of Correctness

We will prove that if a feasible solution exists for a given min-area max-length routing problem, our algorithm is guaranteed to find it.

Lemma 4.1 *Consider the initial positions of two adjacent boundaries L_i and L_{i+1}.*

[2]We assume that a boundary L_j $(j > i+1)$ is updated lazily; i.e., only before net $j-1$ is to be routed.

It is guaranteed that every segment in the rising and falling regions of L_{i+1} is adjacent to either some segment of L_i or the channel boundary. In other words, extra routing space between L_i and L_{i+1} can only exist to the left of the middle region of L_{i+1}.

PROOF. The main intuition is that maximum detour for L_{i+1} is determined by the position of its middle region. The bends that are not in the middle region of L_{i+1} must be due to the blockage of L_i. ■

Definition 4.2 *A grid cell in the final routing solution is defined to be* critically allocated *iff its removal causes either a route to be disconnected or a min-area constraint to be violated.*

We can argue that the grid cells that are not critically allocated may cause some min-area constraints to be violated. As an example, consider the interval between L_4 and L_5 in Figure 4.4. If the min-area constraint of net 4 is not large enough, there will be several grid cells here that are not critically allocated. On the other hand, the min-area constraint for net 6 can be violated, since it cannot be routed further to the left due to the blockage of L_5. In other words, some routing resources are wasted in one part of circuit, while there are not enough resources in other parts. Our optimality proof will be based on the fact that our algorithm uses routing resources at least as efficiently as any feasible solution.

Remark 4.1 *If there is no extra space between the initial positions of L_i and L_{i+1}, then all grid cells between the final positions of L_i and L_{i+1} will be* critically allocated.

Lemma 4.2 *If there is some extra space between the initial positions of L_i and L_{i+1}, then either (1) all grid cells between the final positions of L_i and L_{i+1} will be* critically allocated*, or (2) the final position of L_{i+1} will be the same as its initial position.*

PROOF. The left boundary L_{i+1} will be flipped only after all extra routing space between L_i and L_{i+1} is critically allocated by some net to the left of net $i+1$. From Lemma 4.1, we know that the extra space must be to the left of the middle region of L_{i+1}. Since the algorithm continuously flips L_i from its leftmost available segment, the position of L_{i+1} will stay the same until all extra space to its left has been critically allocated. ■

The following discussion will be based on comparison of the solution of our algorithm with an arbitrary feasible solution. Let T_i, A_i denote the length of and area allocated for net i, respectively, in our solution. (Note that since we assume that the route of net i follows the trail of L_i, the length of L_i is also equal to T_i.) Let T_i^F and A_i^F denote the corresponding quantities in the arbitrary feasible solution.

Lemma 4.3 *Consider the final positions of left boundaries in the solution of our algorithm. If all grid cells have been critically allocated between L_1 and L_n, then there exists no convex nonmonotonic left boundary L_i, $1 \leq i \leq n$, such that $T_i > A_i^F$.*

PROOF. By contradiction, consider the smallest i such that L_i is convex nonmonotic, and $T_i > A_i^F$. We know that no left boundary can be convex nonmonotonic in the beginning of our algorithm. So, before obtaining the final L_i, our algorithm must have tested a configuration L_i' such that $T_i' = A_i^F$, and no more flip is possible on L_i' without increasing its length T_i'. Note that in the feasible solution considered above, all grid cells allocated for nets $[1, i)$ must be within the region between L_1 and L_i', because L_1 is the absolute leftmost boundary for any solution, and L_i' is the rightmost possible boundary if net i has a length of at most A_i^F. So, we can say that the set of grid cells allocated for net $[1, i)$ in the feasible solution is a proper subset of the region between L_1 and L_i. Now, consider two cases:

- Case 1: There is no L_j $(j < i)$ that is concave nonmonotonic and $T_j > A_j^F$. In other words, for all k, $1 \leq k < i$, it is the case that $T_k \leq A_k^F$; hence $A_k \leq A_k^F$, since all grid cells between L_1 and L_n are assumed to be critically allocated. However, we have shown above that the number of grid cells between L_1 and L_i is greater than the number of grid cells allocated for nets $[1, i)$ in the feasible solution, if L_i is convex nonmonotonic and $T_i > A_i^F$. This is a contradiction, and our proof is complete for this case.

- Case 2: There is at least one L_j $(j < i)$ that is concave nonmonotonic and $T_j > A_j^F$. Now, consider the largest such j value. Since net j already satisfies its min-area constraint (i.e., $T_j > A_j^F$), L_{j+1} will not be flipped by net j. Note that this means L_{j+1} cannot be convex nonmonotonic; hence $j < i - 1$. By the same arguments above, the set of grid cells allocated for nets $[j + 1, i)$ in the feasible solution must be a proper subset of the region between L_{j+1} and L_i. Also, we know that for all k, $j + 1 \leq k < i$, it is the case that $T_k \leq A_k^F$;

hence $A_k \leq A_k^F$, due to the assumption of critical allocation. Again, this is a contradiction, and the proof is complete.

■

Theorem 4.1 *Assume that a feasible solution exists for a given problem containing nets* $[1..n]$. *If our algorithm* critically allocates *all grid cells between final positions of* L_1 *and* L_n, *then it is guaranteed that the solution found will be feasible.*

PROOF. The proof is based on induction on the number of concave nonmonotonic left boundaries L_i for which $T_i > A_i^F$:

- Base case: There exists no L_k $(1 \leq k \leq n)$ for which $T_k > A_k^F$. For all k, $1 \leq k \leq n$, it will be the case that $T_k \leq A_k^F$; hence, $A_k \leq A_k^F$, due to critical allocation. By contradiction, consider the leftmost net j of which min-area constraint has been violated. This means that L_{j+1} has been maximally flipped to the right in our algorithm. On the other hand, any feasible solution for nets $[1, j]$ must be within the region between L_1 and L_{j+1}, by definition. Since we have already shown that $A_k \leq A_k^F$ for all nets, it must be the case that $A_k = A_k^F$ for all k, $1 \leq k \leq j$. This contradicts with the assumption that min-area constraint for net k is not satisfied.

- General case: Consider the smallest j $(1 \leq j \leq n)$ for which $T_j \geq A_j^F$. The same proof above applies to the nets $[1, j)$; hence, their min-length constraints must have been satisfied. Now, consider the subproblem containing nets $(j, n]$. Since $T_j > A_i^F$, and L_j is concave nonmonotonic, the area between L_j and R_n is a superset of the set of grid cells allocated by nets $(j, n]$ in the feasible solution. Hence, our inductive hypothesis applies for it.

■

Theorem 4.2 *If there exists a feasible solution for a given min-area max-length routing problem, then it is guaranteed that the given algorithm is going to find it.*

PROOF. The proof is based on (reverse) induction, where the base case contains only the rightmost net n. It is obvious that the theorem is correct for the base case. Now we will prove that the theorem holds for an input problem containing nets $[1..n]$. Consider the smallest j value $(1 \leq j < n)$ such that there are some grid cells

not critically allocated between L_j and L_{j+1}. If no such j exists, then the proof is complete due to Theorem 4.1. Otherwise, the final position of L_{j+1} will be the same as its initial position due to Lemma 4.2. This implies that the subproblem containing nets $[j+1..n]$ remains unmodified; hence our induction hypothesis applies for it. On the other hand, we can use Theorem 4.1 to show that the solution found for nets $[1..j]$ is feasible since all grid cells are critically allocated between L_1 and L_j. As a result, a feasible solution will be found (if one exists) for all nets in the given input problem. ■

4.3 Bus Routing with Min-Max Length Constraints

4.3.1 Problem Formulation

In this section, we extend the algorithm given in Section 4.2 to the problem of *river routing with min-max length constraints*. The main difference here is that the minimum constraints are also given as exact length bounds, instead of min-area requirements.

In the original river routing problem [50], the input is a single-layer rectangular routing channel, and a set of two-terminal nets, where all terminals are aligned at the top and the bottom of the channel. In this section, we extend this problem for the case where all nets have to be routed within prespecified min-max length bounds.

Most of the existing work on river routing assume monotonic routes both in horizontal and vertical directions, since it does not hurt routability [39]. However in our case, we will need explicit detours to satisfy min-length constraints. So, we assume monotonicity only in the vertical direction. In other words, routes are allowed to proceed in three directions: *left, right*, and *down*.

4.3.2 Routing Algorithm

For this problem, we use almost the same algorithm given in Figure 4.3. The main difference here is that we need to check min-length constraint of *Route*(L_i, L_{i+1}) in each iteration, instead of the min-area constraint. Recall that it was trivial to calculate the total area between L_i and L_{i+1}, after each flip on L_{i+1}. However, we now need to calculate the maximum length achievable by route i, so that we can

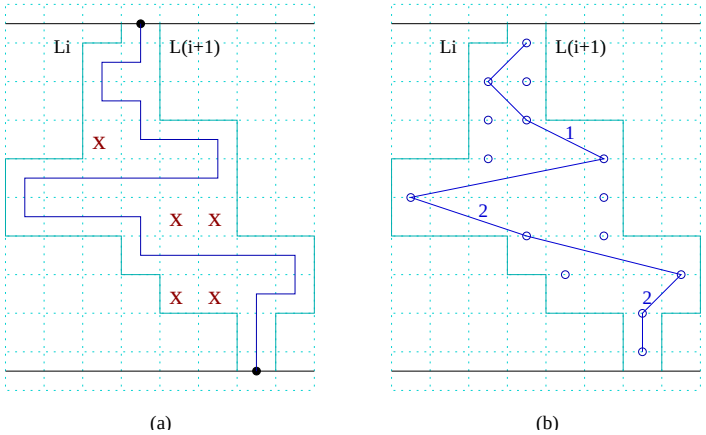


Figure 4.10: (a) A sample max-length route, where wasted cells are marked with 'X'. (b) The corresponding shortest path in $\mathcal{G}$. For clarity, only the edges on the shortest path, and only the non-zero edge weights are displayed.

determine whether min-length constraint for net i can be satisfied within the interval defined by L_i and L_{i+1}.

For the purpose of calculating the maximum-length route efficiently, we define a graph $\mathcal{G}$, corresponding to the interval between L_i and L_{i+1}. For each row, two vertices are defined on the leftmost and rightmost horizontal grid edges within the interval.[3] Then, an edge is defined from each vertex in row k to each vertex in row $k+1$, corresponding to the paths between the respective cells. The weight of an edge is defined to be the number of grid cells *wasted* (i.e., not used by the route), if this edge is selected. Figure 4.10(a) shows a maximum-length route between two terminal points, and (b) shows the corresponding path in the graph model $\mathcal{G}$.

It is straightforward to show that the shortest path between the top and bottom vertices in $\mathcal{G}$ corresponds to the maximum-length route in the original problem; and the total path length in $\mathcal{G}$ is actually equal to the number of grid cells wasted by this route. Observe in the example of Figure 4.10(a) that there are five grid cells wasted within the given interval, and the length of the corresponding shortest path in (b) is also five.

Based on this graph model, the time complexity of calculating the maximum length achievable by $Route(L_i, L_{i+1})$ is $O(W)$, where W denotes the channel width, i.e., the number of rows between the top and bottom terminal points. As discussed in

[3]If there is a single grid edge in a particular row, there will be a single vertex defined. However for consistency of presentation, assume that the two vertices overlap with each other in this case.

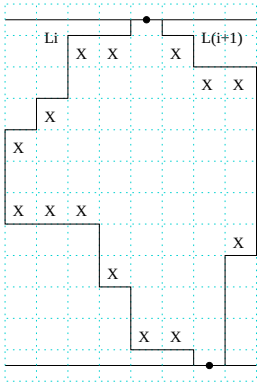


Figure 4.11: A sample interval illustrating the upper bound on $waste(i)$. It is possible to construct a route that wastes at most the grid cells marked with 'X'.

Section 4.2.2, we can find the optimum position of L_{i+1} using two binary searches on the *flip sequence*. So, we need $O(logA_i)$ iterations to route net i, where A_i denotes the total area between L_i and L_{i+1}. As a result, the overall time complexity of routing all nets within min-max length bounds becomes: $O(nWlogA)$, where n is the number of nets, and A is the total area of the channel. Note that since $nW \leq A$, the time bound can also be stated as $O(AlogA)$.

4.3.3 Analysis of the Algorithm

We are going to show that the algorithm described in the previous subsection is near-optimal.

Definition 4.3 *Let $Route(L_i, L_{i+1})$ denote the route with the maximum length within the interval of L_i and L_{i+1}. The* waste *due to net i (denoted as $waste(i)$) is defined as the number of grid cells that are within the respective interval, but not used by $Route(L_i, L_{i+1})$.*

Observe in the example of Figure 4.10(a) that $waste(i)$ is 5, since the max-length route of net i cannot use 5 grid cells.

Lemma 4.4 *For each net i, $waste(i)$ is guaranteed to be less than the sum of horizontal spans of the trails corresponding to L_i, and L_{i+1}.*

PROOF. Figure 4.11 illustrates the idea with an example. Assume that we construct a path in the corresponding graph $\mathcal{G}$ by always choosing edges going from a vertex

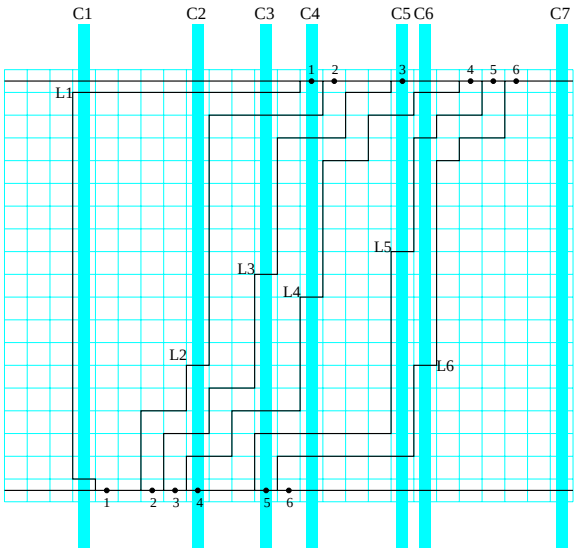


Figure 4.12: The final positions of left boundaries are illustrated for 6 nets. For each left boundary L_i, the leftmost flipped column (if any) C_i is also highlighted. The extra grid cells between L_i and L_{i+1} exist only at columns within the interval $[C_i, C_{i+1}]$. For consistency, C_7 is defined to be the rightmost channel boundary.

adjacent to L_i to another vertex adjacent to L_{i+1}, or vice versa. We can show that such a path will only waste the grid cells marked with 'X' in Figure 4.11 in the worst case. Since the shortest path in $\mathcal{G}$ (corresponding to the longest route in the channel) is guaranteed to waste at most as many grid cells as this path, the lemma follows. ■

For the following analysis, let C_i denote the leftmost column of left boundary L_i that has been flipped. If L_i has never been flipped, let it denote the leftmost column of L_i.

Remark 4.2 *For each i, $1 \leq i < n$, it is the case that $C_i < C_{i+1}$.*

Lemma 4.5 *The extra grid cells allocated for net i can only be at columns within the interval $[C_i, C_{i+1}]$. In other words, there exists only a single trail between L_i and L_{i+1} outside the interval $[C_i, C_{i+1}]$ (see Figure 4.12). For consistency of presentation, C_{n+1} is defined to be the rightmost channel boundary, for a set of n nets.*

Proof. Consider a left boundary L_i and the corresponding column C_i. By definition, no segment of L_i to the right of C_i has been flipped. So, any bend on L_i that is to the right of C_i must either be adjacent to the channel boundary or be due to the

blockage of L_{i-1}. So, all nets to the left of net i can have only a single trail at columns to the right of C_i. On the other hand, remember that our algorithm always flips the leftmost segment of L_i that is not adjacent to R_i. So, all segments of L_i to the left of C_i (if any) must have been flipped until the corresponding right boundaries. Hence, for each net j, $j \geq i$, there can be at most a single trail to the left of C_i. These concepts are illustrated by an example in Figure 4.12. ■

Theorem 4.3 *For a given routing problem with min-max length constraints, the number of grid cells* wasted *by all nets will be less than $4H$, where H is the number of grid cells in one row of the channel.*

PROOF. Due to Lemma 4.5, the grid cells wasted by net i can only be at columns within the interval $[C_i, C_{i+1}]$. Also, the total size of the horizontal trails of L_i and L_{i+1} in the interval $[C_i, C_{i+1}]$ will be at most $2(C_{i+1} - C_i + 1)$. So, due to Lemma 4.4, we can state that $waste(i) \leq 2(C_{i+1} - C_i + 1)$. As a result, the total number of grid cells wasted by n nets will be at most $2(H + n - 1)$, which is less than $4H$. ■

Definition 4.4 *Let $\mathcal{C}^W$ denote a river routing problem with channel width W, and with length constraints T_i^{min}, T_i^{max} for each net i. The projection of $\mathcal{C}^W$ onto a channel width of W-k (denoted as $\mathcal{C}^{W-k}$) is defined to be the same routing problem as $\mathcal{C}^W$, except that the channel width in $\mathcal{C}^{W-k}$ is W-k, and length constraints are T_i^{min}-k and T_i^{max}-k, respectively for each net.*

For the rest of the analysis, we will use the following notations:

- $\mathcal{C}_L^W$: the given min-length max-length routing problem with length constraints T_i^{min}, T_i^{max} for each net i.
- $\mathcal{C}_L^{W-3}$: the projection of $\mathcal{C}_L^W$ onto channel width W-3, with length constraints T_i^{min}-3 and T_i^{max}-3, for each net i.
- $\mathcal{C}_A^{W-3}$: the min-area max-length routing problem obtained by replacing min-length constraints of $\mathcal{C}_L^{W-3}$ with min-area constraints A_i^{min} for each net i, where A_i^{min} is defined as follows: If the minimum detour required to satisfy the min-length constraint of net i in C_L^{W-3} is even, then $A_i^{min} = T_i^{min} - 3$; otherwise,[4]

[4]Let MD_i denote the Manhattan distance between the terminals of net i. It is obvious that any valid route for net i will have a length of $MD_i + d_i$, where d_i is an even number. If the minimum detour required to satisfy the min-length constraint $T_i^{min} - 3$ is an odd number, a feasible solution will have a length of at least $T_i^{min} - 2$, since odd detour is not possible.

$A_i^{min} = T_i^{min} - 2$.

- S_A^{W-3}: the solution to $\mathcal{C}_A^{W-3}$ produced by our min-area max-length routing algorithm (given in Section 4.2).
- S_A^W: the *projection* of S_A^{W-3} to channel width of W (see Definition 4.5).
- S_L^W: the solution obtained after routing each net within the area allocated for it in S_A^W (as in the example of Figure 4.10).

Our objective in the following analysis is to show that if a feasible solution to $\mathcal{C}_L^{W-3}$ exists, then our min-length max-length routing algorithm is guaranteed to find a feasible solution to $\mathcal{C}_L^W$.

Definition 4.5 *Assume that our min-area max-length routing algorithm is applied on $\mathcal{C}_A^{W-3}$. As described in Section 4.2, a number of flips are performed on each left boundary L_i during the execution of the algorithm. Let $\#fc_i^{W-3}$ denote the number of columnwise flips, and $\#fs_i^{W-3}$ denote the number of single flips on the last flipped column of left boundary L_i (see Figure 4.13(a)). The* projected solution *$\mathcal{S}_A^W$ is defined to be constructed from $\mathcal{S}_A^{W-3}$ as follows:*

- *Find the leftmost and rightmost boundaries L_i and R_i for each net i for channel width W.*
- *For each net i:*
  - *Perform $\#fc_i^W$ columnwise flips on L_i, where $\#fc_i^W = \#fc_i^{W-3}$.*
  - *Perform $\#fs_i^W$ single flips on the next column of L_i, where $\#fs_i^W$ is defined as follows: If $\#fs_i^{W-3}$ is equal to 0, then $\#fs_i^W$=0; else if L_i is convex nonmonotonic, then $\#fs_i^W = \#fs_i^{W-3} + 2$; otherwise $\#fs_i^W = \#fs_i^{W-3} + 1$.*
- *The route of net i in $\mathcal{S}_A^W$ is defined to follow the trail of L_i as close as possible; and all the grid cells between L_i and L_{i+1} are defined to be allocated by net i.*

It is important to note here that *solution projection* is not an actual part of the algorithm we have proposed, but it is only used for correctness analysis. Projection of a concave nonmonotonic left boundary from channel width $W-3$ to channel width W is illustrated in Figure 4.13, as an example.

For the rest of the analysis, we will use the following notation:

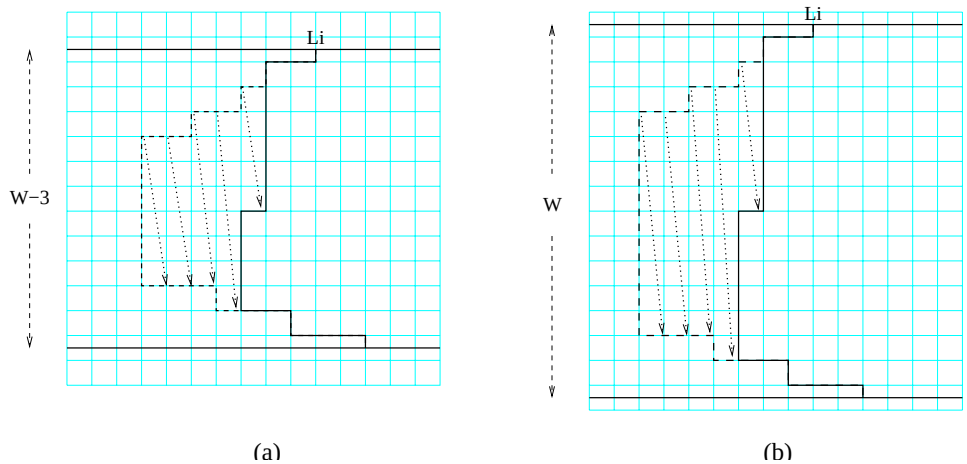


Figure 4.13: Illustration of solution projection from channel width $W-3$ to W. The flips performed are shown by dotted arrows, and the final positions of L_i are shown by solid arrows. (a) The original output of the min-area max-length routing algorithm, where $\#fc_i^{W-3} = 4$, and $\#fs_i^{W-3} = 5$. (b) The projected solution, where $\#fc_i^{W} = 4$, and $\#fs_i^{W} = 6$.

- C_i: The leftmost column of left boundary L_i that has been flipped. If L_i has never been flipped, let it denote the leftmost column of L_i.
- A_i^{W-3}, A_i^W: The number of grid cells (i.e., area) allocated for net i in $\mathcal{S}_A^{W-3}$, and $\mathcal{S}_A^W$, respectively.
- T_i^{W-3}, T_i^W: The length of net i in $\mathcal{S}_A^{W-3}$, and $\mathcal{S}_A^W$, respectively.
- T_i^{min}: The min-length constraint for net i in $\mathcal{C}_L^W$.

Remark 4.3 *The C_i values of $\mathcal{S}_A^W$ are the same as those of $\mathcal{S}_A^{W-3}$.*

Lemma 4.6 *For any i, $1 \leq i \leq n$, if $C_{i+1} = C_i + 1$ in $\mathcal{S}_A^W$, and if C_L^{W-3} has a feasible solution, then it is guaranteed that the min-length constraint of net i in $\mathcal{S}_L^W$ is satisfied.*

PROOF. We know that $A_i^W \geq A_i^{W-3} + 3$, due to the increase in the channel width. Since $C_{i+1} = C_i + 1$, and due to Lemma 4.5, the extra grid cells allocated for net i can only be at columns C_i and C_{i+1}. Figure 4.14 illustrates examples for different left boundary types. Observe that if the number of extra grid cells allocated for net i is even, then net i can be routed between L_i and L_{i+1} without *wasting* any grid cell. In

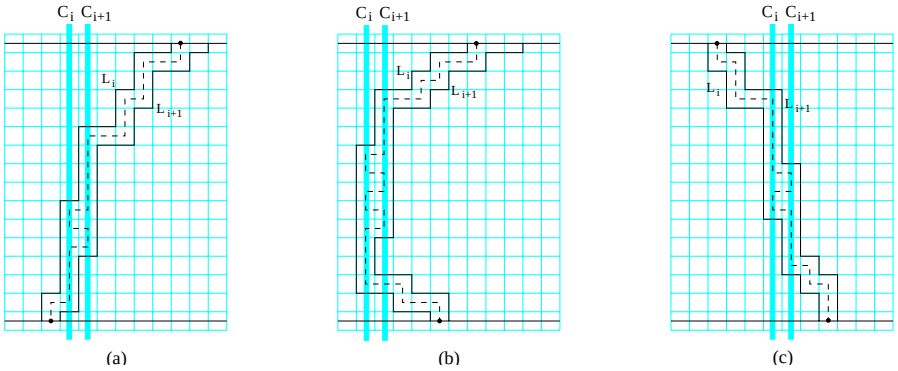


Figure 4.14: Examples illustrating the final positions of different left boundary types, where $C_{i+1} = C_i + 1$. The dashed lines indicate the route for net i between left boundaries L_i and L_{i+1} for each case. If the number of extra grid cells between L_i and L_{i+1} is even, then no grid cell is wasted by the corresponding route.

this case, we will have $T_i^W = A_i^W \geq A_i^{W-3} + 3 \geq T_i^{min}$. Remember from the definition of $\mathcal{C}_A^{W-3}$ in Section 4.3.3 that the number of extra grid cells required to satisfy the min-area constraint of net i in $\mathcal{C}_A^{W-3}$ is always even. Since the number of extra grid cells in $\mathcal{S}_A^W$ will be at least as large as the number of extra grid cells in $\mathcal{S}_A^{W-3}$, the lemma follows. ■

Lemma 4.7 *For any i, $1 \leq i \leq n$, if $C_{i+1} > C_i + 1$ in $\mathcal{S}_A^W$, and if C_L^{W-3} has a feasible solution, then it is guaranteed that the min-length constraint of net i in $\mathcal{S}_L^W$ is satisfied.*

PROOF. The proof is based on case-by-case analysis of different types of left boundaries L_i and L_{i+1}. Note that since a left boundary can be one of the four types described in Definition 4.1, there are 16 different cases for L_i and L_{i+1}. Figure 4.15 illustrates 4 of these cases, and it is straightforward to extend the ideas here for the remaining 12 cases.

The following notations are used in Figure 4.15:

- s_i: the number of columns between (but excluding) C_i and C_{i+1}, i.e., $s_i = C_{i+1} - C_i - 1$.
- $waste(i)$: the number of grid cells wasted by the longest route within the region between L_i and L_{i+1} (see Definition 4.3).
- $\Delta area(i)$: The area increase between L_i and L_{i+1} after solution projection, i.e., $\Delta area(i) = A_i^W - A_i^{W-3}$.

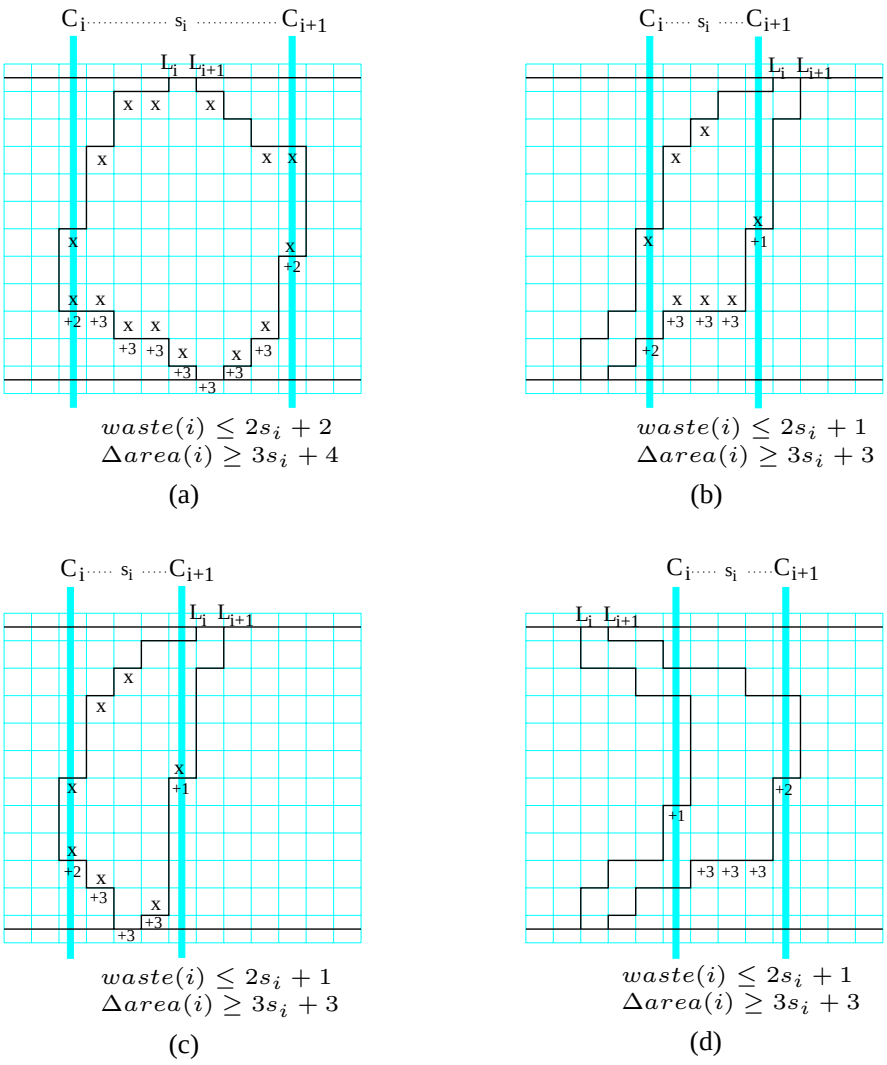


Figure 4.15: Calculation of $waste(i)$ and $\Delta area(i)$ values for different types of L_i-L_{i+1} pairs. The grid cells marked with 'X' are the cells that will be wasted in the worst case by the longest route for net i. The number of such grid cells gives an upper bound for $waste(i)$. The increase in the number of grid cells (due to solution projection) at each column between L_i and L_{i+1} is indicated below each column. Note that the lower bound for $\Delta area(i)$ is calculated as the sum of these values between C_i and C_{i+1}.

Due to Lemma 4.5, the grid cells *wasted* between L_i and L_{i+1} can only be at columns within the interval $[C_i, C_{i+1}]$. To find an upper bound for $waste(i)$, we can construct a route that *snakes* between boundaries L_i and L_{i+1}, as described in the proof of Lemma 4.4. In each case illustrated in Figure 4.15, the grid cells that would be wasted in the worst case by such a route are marked with an 'X'. Since the route with the maximum length is guaranteed to waste at most this many grid cells, the number of grid cells marked with 'X' in each case gives us an upper bound for $waste(i)$.

Due to the increase of channel width from $W-3$ to W, we know that the number of grid cells at each column will increase by 3. To calculate $\Delta area(i)$, we should consider the extra grid cells that are in the region between L_i and L_{i+1}. From Definition 4.5, we know that no segment of L_i in the interval (C_i, C_{i+1}) has been flipped. On the other hand, the segments of L_{i+1} (if any) in the interval (C_i, C_{i+1}) have been flipped maximally, i.e., until the right boundary R_{i+1}. So, we can state that the 3 extra grid cells at each column in the interval (C_i, C_{i+1}) are all in the region between L_i and L_{i+1}. To find the corresponding area increase at columns C_i and C_{i+1}, we should consider the last flip performed on L_i and L_{i+1}, respectively. These flips are illustrated in Figure 4.15 for each case, and the corresponding increase at these columns are marked below. For instance, if L_i is concave nonmonotonic as in Figure 4.15(a), then the corresponding area increase on column C_i will be 2. The reason is that the last flip on L_i will have $\#fs_i^W = \#fs_i^{W-3} + 1$ (see Definition 4.5), and 1 out of 3 extra grid cells on column C_i will be outside the region between L_i and L_{i+1}.

Based on these considerations, the values of $waste(i)$ and $\Delta area(i)$ are given for each case in Figure 4.15. Note that since $C_{i+1} > C_i + 1$, s_i will have a value of at least 1. So, for each case, we will have $\Delta area(i) - waste(i) \geq 3$. Based on this, we can state that $A_i^W \geq A_i^{W-3} + waste(i) + 3 \geq T_i^{min} + waste(i)$. This means that we can find a routing solution for each net i that satisfies the min-length constraint T_i^{min} within the region between L_i and L_{i+1} in $\mathcal{S}_L^W$. ■

Now, we can prove the correctness of our algorithm as follows.

Remark 4.4 *A feasible solution to $\mathcal{C}_L^{W-3}$ is also a feasible solution to $\mathcal{C}_A^{W-3}$.*

Remark 4.5 *If there exists a feasible solution to C_A^{W-3}, then S_A^{W-3} is guaranteed to be feasible (due to Theorem 4.2).*

Lemma 4.8 *It is possible to obtain a feasible S_L^W, which satisfies all length constraints of $\mathcal{C}_L^W$, by projecting S_A^{W-3} to S_A^W, and then routing each net within the area allocated for it in S_A^W.*

PROOF. The proof directly follows due to Lemmas 4.6 and 4.7. ■

Remark 4.6 *S_L^W is within the solution space explored by our min-length max-length routing algorithm.*

Theorem 4.4 *Assume that $\mathcal{C}_L^W$ is the given min-length max-length routing problem, and $\mathcal{C}_L^{W-3}$ is the projected problem onto channel width W-3. If a feasible solution exists for $\mathcal{C}_L^{W-3}$, then our algorithm is guaranteed to find a feasible solution for $\mathcal{C}_L^W$.*

PROOF. From Remarks 4.4 and 4.5, and Lemma 4.8, we know that S_L^W will be feasible if there exists a feasible solution to $\mathcal{C}_L^{W-3}$. From Lemma 4.6, our algorithm will find either S_L^W or another feasible solution. ■

4.3.4 Discussions and Practical Considerations

Recall that the given algorithm in Section 4.2 finds the optimal solution if the minimum constraints are given as area constraints. Theorem 4.3 suggests that, if a prespecified amount of routing area is allocated for each net using this optimal algorithm, then all routing resources within the allocated areas will be successfully used for length extension, except for at most a number of $4H$ grid cells. Since the total area of the channel is WH, where W is the channel width, and typically is much larger than 4, the number of grid cells wasted will be negligible.

On the other hand, Theorem 4.4 makes a stronger statement, giving an approximation factor for the actual routing problem with min-max length constraints. One implication of this theorem is that if we are given a bus routing problem that has a feasible solution, we can guarantee to match the lengths of all nets by extending the channel width by three units. The reason is that all length constraints T_i^{min} and T_i^{max} are increased by exactly the same amount (i.e., three units); and our algorithm is guaranteed to find a feasible solution for the extended channel.

Furthermore, typical industrial circuits have channel widths containing hundreds or even thousands of grid cells. One can argue that if a feasible solution exists for the original channel width W, most probably a feasible solution will exist for the channel

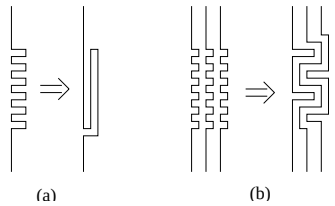


Figure 4.16: Sample post-processing techniques to reduce number of bends. (a) Adjacent jogs are replaced with a longer segment. (b) The jogs for adjacent nets are merged together.

width W-3, since the difference will be almost negligible. So, in practice the proposed algorithm will find the feasible solution for the original problem without the need for extending the channel length.

Note here that our algorithm in this section uses a certain type of snaking, and it does not explore the whole solution space; e.g., the route in Figure 4.10 cannot go up-and-down, since it must be monotonic in the channel direction. Yet, Theorem 4.4 states that the solution found by our algorithm is guaranteed to be *close* to the (most general) optimal solution. In other words, our solution will be sufficiently close to optimum even though we consider only a particular type of snaking. However, it is possible to use different types of length-extension methods to reduce number of bends (see discussion below). Note here that our general framework (given in Figure 4.3) allows using such alternative approaches, as long as it is possible to check the maximum length achievable for net i within the area between L_i and L_{i+1}. Yet another practical approach can be to use the min-area max-length routing algorithm given in Section 4.2.2 first, and then to perform any type of length extension in post processing using the allocated areas.

The type of snaking we use in this algorithm (as illustrated in Figure 4.2) is also frequently used in current industrial circuits [40]. However, it is possible to reduce the number bends by using some post-processing methods, if necessary. Two possible techniques are illustrated in Figure 4.16. Here, part (a) gives a straightforward replacement of adjacent jogs with a single longer detour. On the other hand in part (b), the jogs for three nets are merged to obtain a solution with less number of bends.

Table 4.1: Comparison of the single-layer routing algorithm proposed in this chapter with the Lagrangian relaxation based methodology

			SINGLE-LAYER		LR-BASED	
Circuit	Avg. spacing	Length stdev	# nets failed	time (min:sec)	# nets failed	time (min:sec)
C1	2.59	31.96	0	0:02	1	12:54
C2	2.80	47.80	0	0:02	2	7:45
C3	2.18	66.12	1	0:02	7	20:41
C4	1.81	44.62	0	0:03	18	21:30
C5	1.53	11.13	0	0:02	21	8:31
C6	1.64	41.26	0	0:02	29	17:58
C7	1.52	53.46	1	0:02	86	45:34
IBM_1	7.75	35.78	3	0:02	3	6:32
IBM_2	6.41	45.98	1	0:02	1	4:31
IBM_3	9.16	73.76	0	0:03	0	4:09

4.4 Experimental Results

We have performed experiments to compare the single-layer routing algorithm proposed in this chapter with the Lagrangian relaxation based methodology of Chapter 2. All algorithms in this section have been implemented in C++, and experiments were performed on an Intel Pentium 4 2.4GHz system with 512MB memory, and a Linux operating system.

For the purpose of illustration, we have first applied our algorithm on a test circuit, shown in Figure 4.17. Each net in this circuit has prespecified min-max length constraints, and the solution displayed satisfies all those constraints.

Then we have performed experiments on test circuits, as demonstrated in Table 4.1. Here, "*avg. spacing*" is measured in terms of the number of grid cells between terminal points of adjacent nets, and it indicates how dense the problem is. On the other hand, column "*length stdev*" gives the standard deviation in net target lengths. Each problem in this table contains around 200-300 nets, with individual length constraints for each net. The grid size for the smallest circuit in this table is 390×200, and the largest one is 776×290. The last three problems here have been extracted from an IBM design, corresponding to the bus routing problems between MCM, memory and STI modules. Here, layer assignment and routing inside chips have been performed by the previous phases of the routing system, as described in Section 4.1, and the input for the bus routing problem is a set of non-crossing nets

on each layer. While the first seven circuits in this table have single layers, the IBM circuits have multiple layers.

As seen in this table, our algorithm performs significantly better than the Lagrangian relaxation based approach on most circuits, in terms of both quality and run time. As the average spacing between nets decrease, the solution quality for LR-based approach degrades, and it takes more time to find a routing solution. The main reason for this is that LR-based approach uses a variant of Pathfinder [3; 19] algorithm in the low level, where routing conflicts are resolved through *negotiations*. As the problems get denser, these negotiations take more and more time, and they don't always successfully lead to a good result. On the other hand, the algorithm we propose in this chapter has optimality guarantees on its target class of problems, and it finds a good solution as long as it exists. Observe that, our algorithm has also very good run-time characteristics, since the underlying flip operations can be done in a fast and efficient way.

4.5 Conclusions

We have proposed two algorithms for high-performance bus routing problem. The first algorithm is for the problem of routing with min-area max-length constraints, where length matching is assumed to be performed in post-processing using the allocated areas. The second algorithm extends these ideas to the problem where minimum constraints are given as exact length bounds. The first algorithm is proven to be optimal, while the second one is provably close-to-optimal. The respective time complexities of these algorithms are given as $O(A)$ and $O(AlogA)$, where A is the area of the intermediate region between chips. Our experiments demonstrate the effectiveness of these algorithms on the target class of problems, compared to the more general Lagrangian relaxation based methodology of Chapter 2.

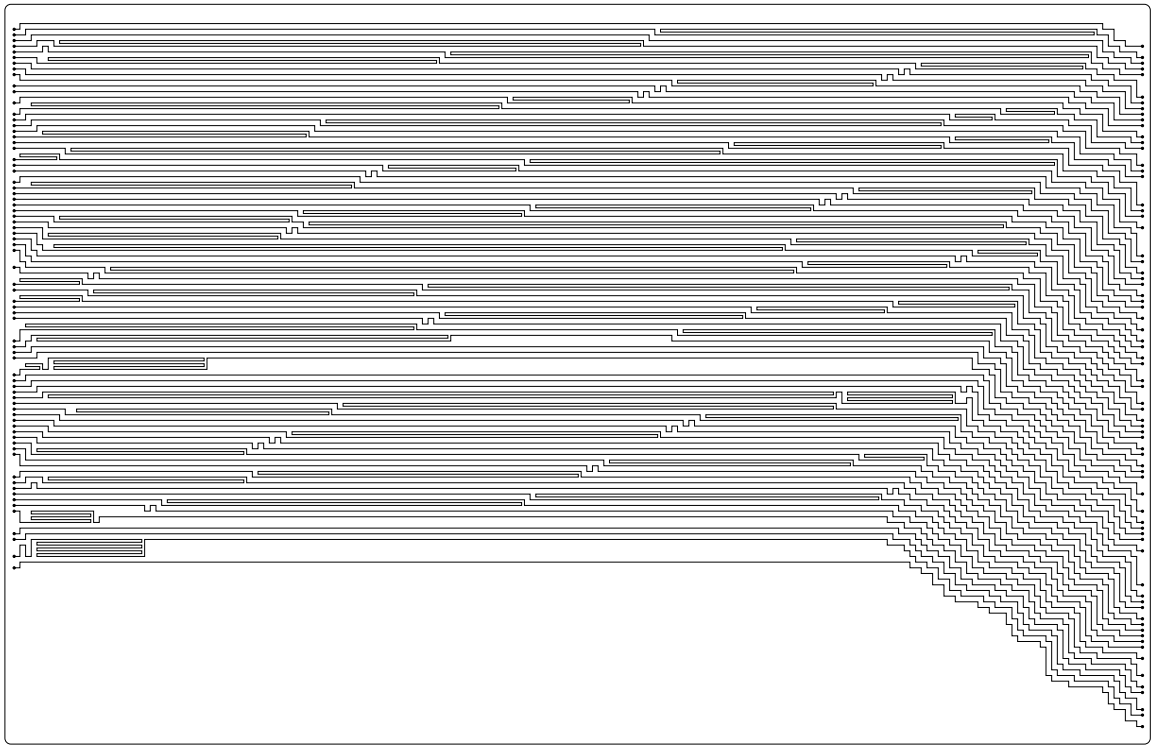


Figure 4.17: The output of our algorithm on a bus-routing problem. Only part of the circuit is displayed here due to space limitations. The post-processing technique illustrated in Figure 4.16(a) has been applied on the output of the algorithm given in Section 4.3 to obtain the final routing solution. Note that although a small problem is chosen here for illustration purposes, typical industrial problems have channel widths on the order of hundreds or even thousands, and our algorithm is scalable for such large problems.

Chapter 5

Algorithms for Simultaneous Escape Routing and Layer Assignment of Dense PCBs

5.1 Introduction

The shrinkage of die sizes and the increase in functional complexities in the past several years made the circuits more and more dense. So, boards and packages have reduced in size, while the pin counts have been increasing. For example, a multichip module (MCM) used in IBM eServer z900 [26] (introduced in 2000), contains 20 processor chips, 8 L2 cache chips, 2 system control chips, 4 memory bus adapter chips, and a clock chip – a total of 35 chips in one package. On the bottom of this MCM, there are 4224 I/O pins within an area of 127-mm $\times$ 127-mm. In the subsequent generation of the same series, IBM eServer z990 [61] (introduced in 2003), the corresponding number of pins in an MCM has increased about 20%, with a decrease of almost 50% in the substrate area. With increasing pin densities of this pace, routing nets on boards beneath the component areas (escape routing) is increasingly becoming the main bottleneck in terms of overall routability [61]. Traditional board routing algorithms cannot handle designs with such complexities, and many high-end boards in the industry today require manual efforts for routing. In a typical design cycle of a high-end board, manual routing efforts take about a month [40], and new effective routing algorithms are necessary to significantly reduce this time. In this chapter, we focus on board-level routing beneath dense components, and we propose algorithms

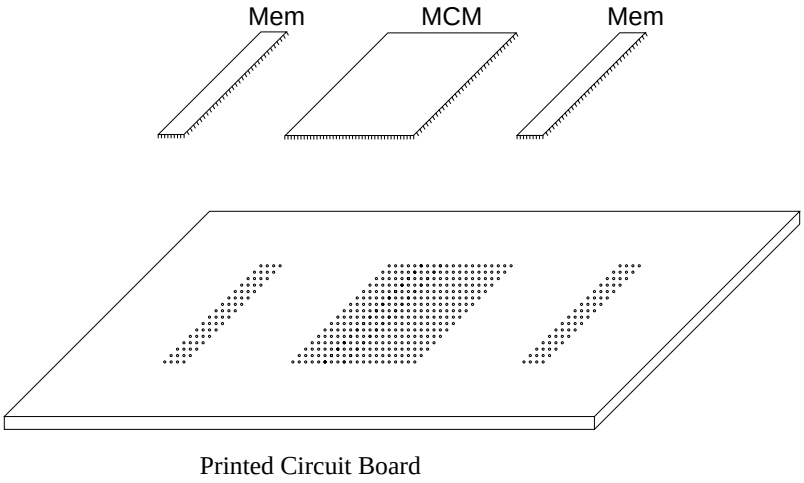


Figure 5.1: Different components are mounted on or plugged into a PCB. A pin array is created on the board corresponding to each component.

that address these challenges effectively.

A typical PCB contains a number of different components such as MCMs, memory, or I/O modules. These components are mounted on or plugged in to the board, forming a set of dense pin arrays, as shown in Figure 5.1. In this chapter, we will focus on the type of board designs where each component pin is accessible from all layers, as will be discussed in detail below. The routing resources within such pin arrays are extremely limited due to the large number of pins, and tight clearance rules. Furthermore there are large number of nets that need to be routed from their terminal pins to the corresponding component boundaries. On the other hand, the intermediate routing area on the board between different components has relatively few blockages, and the amount of available routing resources is relatively larger. So, it is clear that the escape routing problem requires a special focus in a board routing framework.

It is important here to note that escape routing problem for different components should not be considered independent of each other. That is, we cannot just apply a conventional escape routing algorithm [16] on different components independently. The reason is that such an approach ignores the connections between different components, and increases the via requirements significantly. Especially in high-speed designs, these vias seriously degrade signal characteristics, add additional delay, decrease routing area, and lower the manufacturing yields. Furthermore, for some board designs, no buried vias are allowed, for the purpose of limiting manufacturing costs

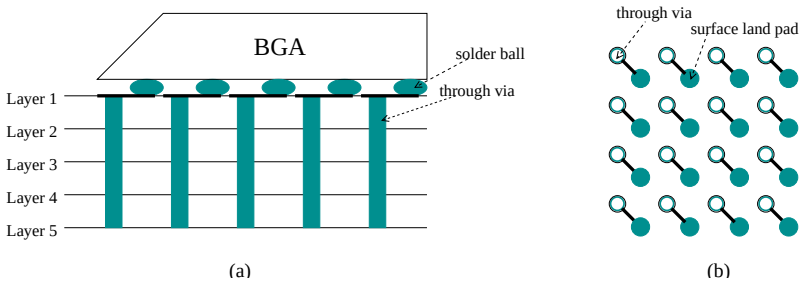


Figure 5.2: A BGA package is mounted on a PCB, and through vias are used to connect component pins to inner layers of the board. (a) Cross-sectional view. (b) Top-side view. In the context of board-level routing, each through via here will be regarded as a component pin.

[40]. For such designs, the nets need to be routed in a planar fashion on every layer. Hence, an escape routing algorithm that tries to minimize (or completely avoid) crossings in the intermediate area is crucial. For this reason, we propose algorithms in this chapter to find the escape routing solutions of different components simultaneously such that the number of crossings in the intermediate area is minimized. For multilayer designs, the best layer assignment also needs to be determined during this process. Figure 5.3 illustrates a sample problem, and Figure 5.4 gives a two-layer solution.

Since the routing resources inside dense components are extremely limited, we assume that via usage is not allowed within components. So, the escape routing solution has to be conflict-free within components on every layer. On the other hand, via usage is possible in the intermediate areas, where there are relatively few routing blockages. However, since vias adversely affect routability and signal delay characteristics, and they lower manufacturing yield, we try to minimize the number of vias through crossing minimization. So, our objective is to find the best conflict-free escape routing solution inside components that will minimize the number of crossings in the intermediate area.

As mentioned before, for the models and algorithms we propose in this chapter, we assume that each component pin can be accessed from every layer of the input PCB. For example in IBM eServer z900 [26], pin grid array (PGA) connectors are used to connect components or daughtercards into the PCBs. PGA-based connectors have a grid of metal leads as their pins, which are plugged into the PCBs, making each pin directly accessible from the inner layers of the board. However, our algorithms

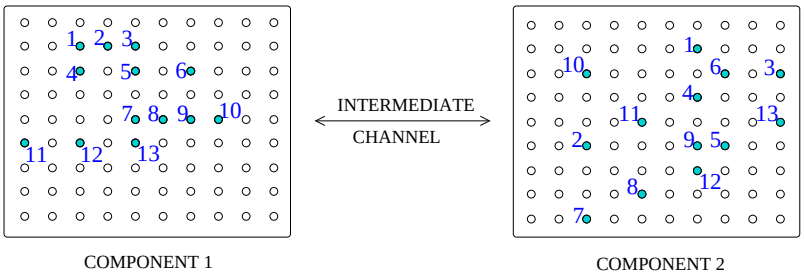


Figure 5.3: A sample escape problem with 13 nets on two components. Each terminal pin is labeled with its net index. The problem is to find a conflict-free routing solution within components, and to minimize crossings in the channel.

are also applicable to surface-mount type packages, if plated through holes (PTHs) [10] (a.k.a through vias) are used to connect component pins to inner board layers. For example, the MCMs in IBM eServers p690 and z990 use land grid array (LGA) connectors [15], which are mounted on the board surface, and connected to PTHs on the board. Similar statements can be made for a ball grid array (BGA)-type package that is mounted on a grid of PTHs, where the through-via pitch is equal to the ball pitch of the BGA. Typically *dog bone pattern*-type routing is used in such cases to connect the component balls to the through-vias on the board surface [8], as illustrated in Figure 5.2. In these cases, we will regard each such through via as a *component pin* in the context of board-level routing. We will not go into further details of these issues in this chapter; instead, we will focus on escape problem from the perspective of board-level routing where each component pin is accessible from every layer, either directly or by through vias.

The rest of this chapter is organized as follows. In Section 5.2, we give a formal description of this problem, and discuss how it relates to the existing work in the literature. Then, we outline our solution approach in Section 5.3. Mainly, we process one layer at a time, and try to route as many non-crossing nets as possible on each layer. In Section 5.4.1, we model the maximal planar routing problem as a *longest path with forbidden pairs* (LPFP) problem, and propose an efficient checkerboard-based graph model for it in Section 5.4.2. Although the general LPFP problem is NP-complete, the special structure of our problem allows us to propose a polynomial-time exact algorithm in Section 5.4.3. Then, we propose a fast and effective randomized algorithm in Section 5.4.4 for large circuits. In Section 5.5, we discuss generalizations of our models and algorithms. Finally, we demonstrate the effectiveness of our algorithms

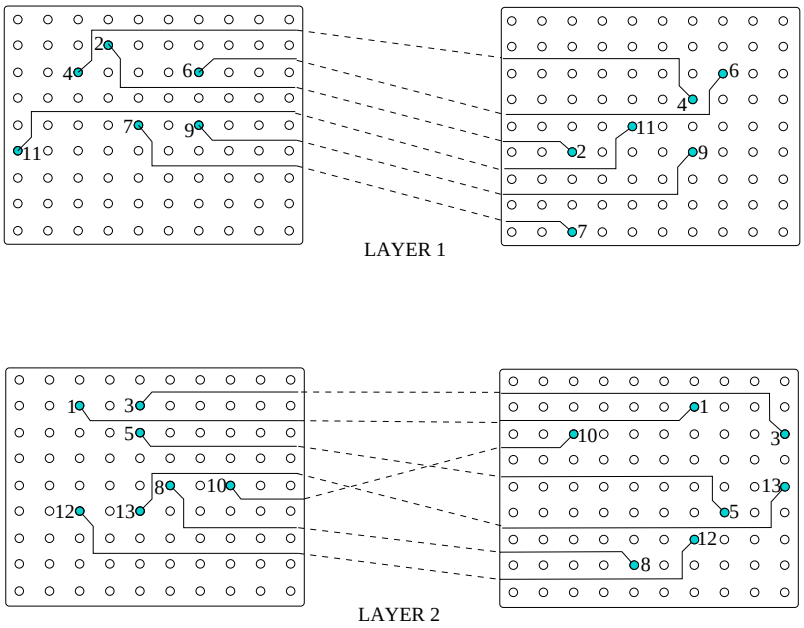


Figure 5.4: A sample solution for the problem given in Figure 5.3. Escape routes are illustrated with solid lines within components. Channel segments are shown with dashed lines.

through experiments in Section 5.6.

5.2 Problem Formulation and Related Work

Let a component be defined as a 2-D array of pins that span multiple layers. The input circuit is assumed to contain two components separated by a channel between them. A two-terminal net specifies two pins as its endpoints, which are assumed to be in different components by definition. For simplicity of presentation, we will assume that only one net can be routed between adjacent rows and columns of component pins[1]. An *escape route* for a given net is defined as the route from one of its terminal pins to the respective component boundary. Two escape routes r_i and r_j within the same component are defined to *conflict* with each other iff r_i and r_j cannot exist together in a permissible one-layer planar routing solution. Given an input circuit and a set of two-terminal nets, the problem is to find an escape routing solution for each net, and assign them to different layers such that (1) conflict-free routing

[1]In Chapter 6, we will discuss how to handle components with multiple routing tracks between adjacent pins.

solution is obtained within each component at every layer, and (2) the number of crossings in the intermediate channel is minimized. Here, routing conflicts are not allowed inside the components, because routing resources within components are too scarce to allow via usage. On the other hand, via usage is allowed in the intermediate channel between components; hence crossings are allowed here. However, since vias have adverse affects on routability and signal delay characteristics, and they lower manufacturing yield, our objective is to minimize the number of vias through crossing minimization.

Figure 5.3 illustrates a sample escape problem with 13 nets in two components, and Figure 5.4 gives a two-layer solution. As mentioned earlier, it is assumed that each pin spans multiple layers; so it is possible to assign the route for each net to any layer. In the given solution, 6 nets are routed on layer 1 without any crossings in the channel. On the other hand, the channel segment of one net (net 10) on layer 2 crosses with others. This crossing can be avoided in the later stages of the routing system by using a via for only net 10. So we can state that the escape routing solution given in Figure 5.4 helps the objective of via minimization since it minimizes the crossings in the channel.

A related problem in the literature is the *pin assignment* problem [6; 34; 62]. Its objective is to determine the positions of pins on chip boundaries such that a cost function is minimized. However this problem ignores escape routing inside the components. Another related problem is the *k-layer topological via minimization* problem [14], where the objective is to determine the topological routing of a set of nets on k routing layers such that the total number of vias is minimized. It has been shown that the general case of this problem is NP-complete, and an algorithm has been proposed for the case of a crossing channel, where nets have fixed pin positions on chip boundaries [14]. However, escape routing is not considered in this problem. On the other hand, in our problem we need to find the escape routes simultaneously while assigning nets to different layers for via minimization. In other words, the pin positions of nets are not fixed on component boundaries, but they are determined based on the escape routes. For instance in the example of Figure 5.4, the ordering of nets within components is not necessarily the same as the ordering on component boundaries,[2] since this ordering further reduces the number of crossings.

[2]For example, in the left component of first layer, net 4 escapes to row 1, and net 2 escapes to row 3, although the terminal of net 2 is above net 4 within the component.

5.3 Methodology

We use a two-phase approach for this problem: (1) for each layer l, pack as many non-crossing routes as possible on l, and (2) distribute the remaining nets to available layers, this time allowing crossings in the intermediate channel.

In the first phase, we process one layer at a time and try to find the maximum subset of available nets that can be routed without any crossings on that layer. The first layer in Figure 5.4 is an example output of this phase. Specifically, the maximum non-crossing subsets for layer 1 and layer 2 have been found to be $\{2, 4, 6, 7, 9, 11\}$, and $\{1, 3, 5, 8, 12, 13\}$, respectively, for this problem. The details of the algorithm we propose for this phase are presented in Section 5.4.

Then in the second phase, the nets that have not been routed are distributed to available layers. In our sample problem, net 10 does not belong to any of the planar subsets of phase 1. So, an escape routing solution is found for it in the second phase on layer 2. Observe in Figure 5.4 that although it has a conflict-free routing solution within the components, it crosses with nets 5 and 13 in the channel.

For the second phase, we use a negotiated congestion based net-by-net approach similar to Pathfinder [19]. The main idea is to allow routing conflicts in the beginning, and then to iteratively rip-up and reroute nets, while gradually increasing the costs of conflicted routing resources. By doing so, nets with alternative routes are forced not to use the conflicted resources, and eventually a conflict-free routing solution is obtained. Note here that we discourage ripping up the nets routed in the first phase by using relatively higher costs for conflicts with these nets.

5.4 Maximal Planar Routing

5.4.1 Algorithm Outline

Given a set of nets, our objective is to find the maximum subset that can be routed on one layer without any conflicts. For this purpose, we define a number of routing patterns for each net, and we propose algorithms to choose the best possible combination of these patterns. For simplicity of presentation, we will focus on a horizontal problem, where one component is to the right of another.

Our main assumption in the following algorithm is that the vertical span of escape routes within components will be limited in a typical solution, as in Figure 5.4, where

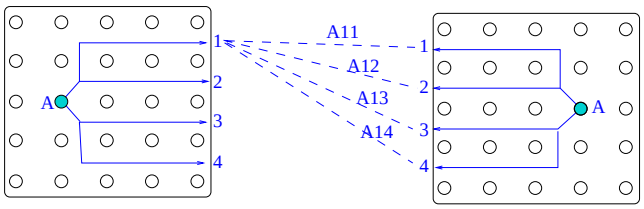


Figure 5.5: Routing patterns considered for net A. Only 4 out of 16 patterns are shown here for clarity.

an escape route spans at most 2 rows. The main reason is that large vertical spans within components block other escape routes; so we need small vertical spans for maximal routing. Furthermore, we have observed this behavior for a great majority of nets in typical manual industrial solutions. Based on this, we define 16 possible configurations for each net,[3] as shown in Figure 5.5. Namely, we consider 4 escape routes for a net within each component, so that it can escape from one of the 4 neighboring rows of its terminal pin. Note that any one of the 4 escape routes within each component can be selected, and so there are $4 \times 4 = 16$ possible routing patterns for each net. Let A_{ij} denote the configuration where net A escapes to row i in the first component, and to row j in the second component. In Figure 5.5, some sample routing patterns are illustrated. As seen in this figure, we consider only simple escape routes, each of which has a single horizontal segment. This assumption can be generalized for more general patterns, as will be discussed in Chapter 6.

Now, the problem can be stated as to select the maximum subset of patterns for a given set of nets such that (1) at most one pattern is selected for each net, (2) there are no conflicts within components, and (3) there are no crossings in the channel. Note that even though we consider only a limited number of routing patterns for each net, there are exponential number of possible ways of selecting patterns for a set of nets. However, we will propose a polynomial time algorithm to select the best combination that gives the maximal planar routing solution.

If every net had only one possible routing pattern (instead of 16), and if there were no conflicts between different nets within components, then we could use a longest path algorithm to find the maximal subset of non-crossing nets [14]. However, we have to consider escape routes within components, and try to find the best possible escape route for each net simultaneously while finding the optimal subset of non-conflicting

[3]In Section 5.5, we discuss possible extensions to relax this assumption.

and non-crossing nets. For this purpose, we will define a graph model $\mathcal{G}$, and a set of forbidden pairs $\mathcal{F}$ (such that $\mathcal{F}$ contains pairs of vertices from $\mathcal{G}$) as follows:

- For each routing pattern, a vertex exists in $\mathcal{G}$.
- Let u, v be two vertices in $\mathcal{G}$ corresponding to routing patterns U_{ij} and V_{kl}, respectively.[4] An edge from u to v exists in $\mathcal{G}$ iff the channel segment of U_{ij} is strictly above the channel segment of V_{kl}, i.e., $i < k$ and $j < l$. (e.g., A_{12} in Figure 5.5 would be strictly above A_{34}.)
- Let u, v be vertices in $\mathcal{G}$. Forbidden pair (u, v) exists in $\mathcal{F}$ iff at least one the following conditions is the case:
  1. u and v correspond to the same net.
  2. The routing patterns corresponding to u and v *conflict* with each other (as defined in Section 5.2) in at least one component.

It is straightforward to show that $\mathcal{G}$ is in fact a directed acyclic graph (dag). We can state that if a path exists from vertex u to vertex v in $\mathcal{G}$, then it is guaranteed that the channel segments of the corresponding routing patterns do not cross with each other. Hence, the longest path in $\mathcal{G}$ will correspond to the maximum set of routing patterns that have no crossings in the channel. However, we also need to consider the conflicts within components, as defined by the forbidden-pair set $\mathcal{F}$. The following theorem gives a formal description of this problem:

Theorem 5.1 *The problem of finding the maximum subset of non-crossing and non-conflicting routing patterns is equivalent to the longest path with forbidden pairs (LPFP) problem on* $\{\mathcal{G}, \mathcal{F}\}$.

LPFP problem [17] for a graph $\mathcal{G}$, and a vertex-pair set $\mathcal{F}$ is defined as finding the longest path P in $\mathcal{G}$ such that P contains at most one vertex from each pair of vertices in $\mathcal{F}$. In other words, if $(u, v) \in \mathcal{F}$, then a permissible path in $\mathcal{G}$ cannot contain both u and v. The general LPFP is known to be an NP-complete problem [1]. However, the following property of our problem will enable us to propose a polynomial time algorithm in Section 5.4.3.

[4] As before, U_{ij} denotes the routing pattern where net U escapes to row i in the first component, and row j in the second component.

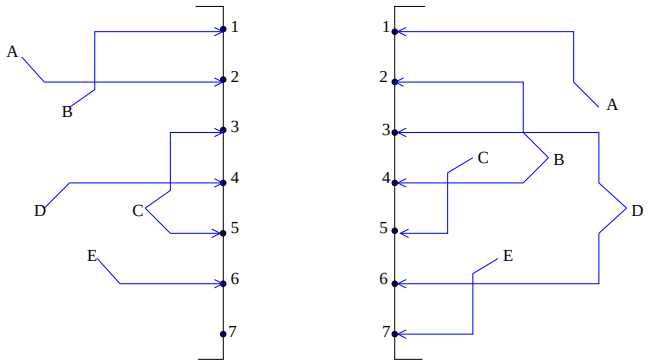


Figure 5.6: A sample escape routing problem for five nets. For clarity, only one or two routing patterns are defined for each net (instead of 16 as in the actual algorithm).

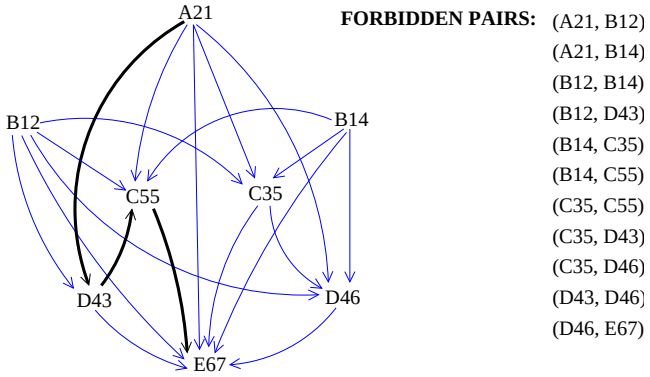


Figure 5.7: The graph model corresponding to the problem given in Figure 5.6. The longest path without forbidden pairs is illustrated with the thick lines.

Lemma 5.1 *For any forbidden vertex pair $(u, v) \in \mathcal{F}$, if v is reachable from u in $\mathcal{G}$, then the maximum path length (in terms of number of edges) from u to v is guaranteed to be less than or equal to 3.*

PROOF. An edge from w to t exists only if the corresponding routing pattern of t escapes to rows strictly below those of w (by definition). Furthermore, the vertical spans of routing patterns are limited. Hence, if u and v conflict with each other within a component, then this means that their escape routes are on *nearby* rows. It is possible to show by case-by-case analysis that u and v cannot escape to rows separated by more than 3 rows if $(u, v) \in \mathcal{F}$. So, the maximum path length between conflicting vertices in $\mathcal{G}$ can be at most 3. ■

Figure 5.6 gives a sample problem with a limited number of patterns defined

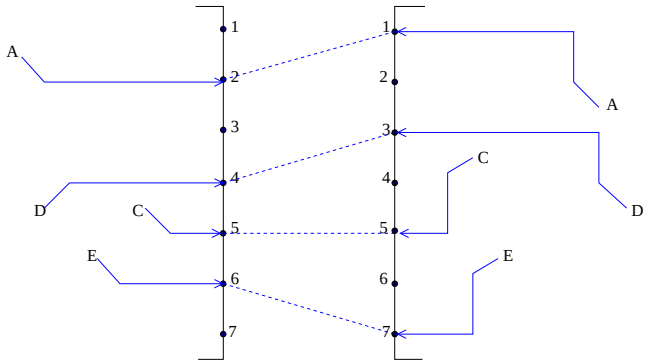


Figure 5.8: The actual maximal planar routing solution corresponding to the path given in Figure 5.7.

for each net.[5] The graph model corresponding to these patterns is illustrated in Figure 5.7. Observe that the longest path without forbidden pairs on this graph is given as $A_{21} \rightarrow D_{43} \rightarrow C_{55} \rightarrow E_{67}$. The actual solution corresponding to this path is also shown in Figure 5.8.

5.4.2 Checkerboard Graph Model

In the graph model described in Section 5.4.1, an edge exists from vertex u to every vertex v of which channel segment is strictly below u. So, the number of edges in $\mathcal{G}$ is $O(n^2)$, where n is the number of nets. In this section, we will describe a more structured graph model with less number of nets.

Let us consider a (conceptual) checkerboard structure with size $r \times r$, where r is the number of rows in a component. As before, let A_{ij} denote the routing pattern where net A escapes to row i in the first component, and to row j in the second component. The main idea here is to (conceptually) assign each routing pattern A_{ij} to cell (i, j) of the checkerboard, as shown in Figure 5.9. We can formally define a graph model $\mathcal{G}_C$ based on this conceptual structure as follows:

- For each cell (i, j) of the checkerboard, a vertex e_{ij} with zero weight exists in $\mathcal{G}_C$.
- For each routing pattern U_{ij}, a vertex u_{ij} with unit weight exists in $\mathcal{G}_C$.

[5] Only one or two patterns are defined for each net for clarity of the figure. In our actual algorithm, there are 16 patterns defined for each net.

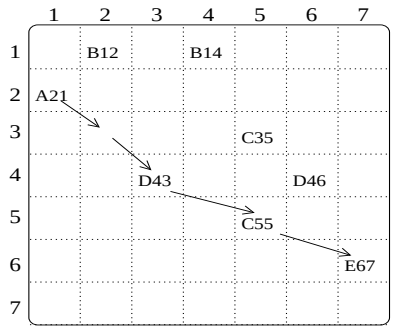


Figure 5.9: The checkerboard structure corresponding to the graph of Figure 5.7. For clarity, only the edges on the longest path are illustrated.

- Let u_{ij} and v_{kl} be vertices in $\mathcal{G}_C$. An edge from u to v exists in $\mathcal{G}_C$ iff ($k = i+1$ AND $l > j$) OR ($l = j+1$ AND $k > i$). In other words, an edge exists only between adjacent rows or adjacent columns of the checkerboard, and the direction is always towards south-east.

Figure 5.9 shows the checkerboard structure corresponding to the graph given in Figure 5.7. For clarity, the vertices with zero weights, and the edges between adjacent rows and columns are omitted in this figure. The corresponding longest path without forbidden pairs is also illustrated here. Observe that this path traverses the empty cell $(3, 2)$ on the checkerboard in addition to the selected routing patterns. Intuitively, this empty cell corresponds to the unused connection from row 3 to row 2 of the channel illustrated in Figure 5.8.

This graph structure is in fact very similar to the one proposed in Section 5.4.1. The main difference is that edges exist only between neighboring routing patterns here, which brings an asymptotic reduction in the problem complexity. For the following analysis, let r and c denote the number of rows and columns of the components, respectively; let s denote the size of the components (i.e., $s = rc$); and let n denote the number of nets. Furthermore, let us assume that the components have constant aspect ratios, i.e., $r = \Theta(c) = \Theta(s^{1/2})$.

Lemma 5.2 *The total number of vertices in $\mathcal{G}_C$ that are assigned to row i of the checkerboard is $O(s^{1/2})$; similarly, the number of vertices assigned to column j is $O(s^{1/2})$, where $1 \leq i, j \leq r$.*

Proof. Remember that each net has a constant number of routing patterns, and each routing pattern has a limited (constant) vertical span (as shown in Figure 5.5).

So, the total number of nets escaping to row k of a component is $O(c)$. Furthermore, the routing patterns assigned to row i of the checkerboard are the ones escaping to row i of the first component, by definition. Similarly, the patterns assigned to column j of the checkerboard are the ones escaping to row j of the second component. In addition, each row and column of the checkerboard contains $O(r)$ zero-weighted vertices, corresponding to the cells of the checkerboard. Hence, the total number of vertices assigned to any row or column of the checkerboard is $O(r + c) = O(s^{1/2})$. ■

Lemma 5.3 *The number of vertices in $\mathcal{G}_C$ is $O(n + s)$, and the number of edges is $O(ns^{1/2} + s^{3/2})$.*

PROOF. For each net, a constant number of routing patterns are defined, and there is a zero-weighted vertex corresponding to each checkerboard cell. Hence, the number of vertices in $\mathcal{G}_C$ is $O(n + s)$. The edges are only between adjacent rows and columns of the checkerboard structure; so each vertex has $O(s^{1/2})$ incoming edges (due to Lemma 5.2). As a result, the number of edges in $\mathcal{G}_C$ is $O(n + s) \times O(s^{1/2}) = O(ns^{1/2} + s^{3/2})$. ■

Assuming that the component pins are densely populated (i.e., $n = \Theta(s)$), the number of edges in graph $\mathcal{G}_C$ is in fact $O(n^{3/2})$. This is an asymptotic reduction in complexity, compared to the graph model described in Section 5.4.1, which has $O(n^2)$ number of nets. Hence, the checkerboard structure will be helpful to reduce the complexity of the exact algorithm we propose in Section 5.4.3. Furthermore, the structured view of a checkerboard will help us to propose a very effective randomized algorithm in Section 5.4.4.

5.4.3 Exact Algorithm for LPFP Problem

As mentioned earlier, the exact algorithm is possible due to the special property of the input graph as given in Lemma 5.1. Our approach will be to perform a graph transformation such that the longest path on the transformed graph will be equivalent to the solution of the LPFP problem on the original graph. This transformation will be described in Definition 5.3; however to give an intuition about this process, we will first describe simpler versions of this transformation in Definitions 5.1 and 5.2.

The notations we will use in this section are as follows: The input problem is given in the form $\{\mathcal{G}, \mathcal{F}\}$, where $\mathcal{G}$ is a directed acyclic graph, and $\mathcal{F}$ is the set containing forbidden vertex pairs. Consider two vertices u and v in $\mathcal{G}$. We denote u

as a *parent* of v if there is an edge $u \rightarrow v$ in $\mathcal{G}$. On the other hand, u is denoted as a *grandparent* of v if there is a vertex w such that the edges $u \rightarrow w$ and $w \rightarrow v$ exist in $\mathcal{G}$. For consistency, we assume that each vertex has a parent-grandparent pair of NULL-NULL.

Definition 5.1 *First-order transformation of $\mathcal{G}$ (denoted as $\mathcal{G}^1$) is defined as follows:*

- *For each vertex u in $\mathcal{G}$, there is a vertex u' in $\mathcal{G}^1$.*
- *There exists an edge $u' \rightarrow v'$ in $\mathcal{G}^1$ iff:*
  1. *The edge $u \rightarrow v$ exists in $\mathcal{G}$.*
  2. *(u, v) is not a forbidden pair*

Remark 5.1 *If the maximum path length between any forbidden pair (u, v) in $\mathcal{G}$ is at most 1, then the longest path in $\mathcal{G}^1$ is the exact solution to LPFP problem in $\mathcal{G}$.*

Definition 5.2 *Second-order transformation of $\mathcal{G}$ (denoted as $\mathcal{G}^2$) is defined as follows:*

- *For each vertex u in $\mathcal{G}$, there is a set of vertices U in $\mathcal{G}^2$ such that $U[i]$ corresponds to the i^{th} parent of u. In other words, number of vertices in U is equal to the number of parents of u.*
- *There exists an edge from $U[i]$ to $V[j]$ in $\mathcal{G}^2$ iff:*
  1. *u is the j^{th} parent of v in $\mathcal{G}$.*
  2. *(u, v) is not a forbidden pair.*
  3. *(i^{th}-parent-of-u, v) is not a forbidden pair.*

As an example, consider graph $\mathcal{G}$ with forbidden pairs in Figure 5.10. Second order transformation of this graph is shown in Figure 5.11. Observe that there is a group of vertices in the transformed graph corresponding to each vertex in $\mathcal{G}$. For instance, there is set D, containing 4 vertices in Figure 5.11, corresponding to vertex d in $\mathcal{G}$. Here, each vertex in set D corresponds to one parent of d, and it is connected to that parent if they are not forbidden pairs. As mentioned earlier, we assume that each vertex in $\mathcal{G}$ has a (pseudo) parent of NULL; hence an extra vertex with no parent

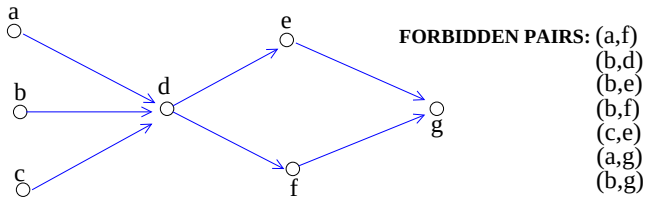


Figure 5.10: A sample graph $\mathcal{G}$, and a set of forbidden pairs.

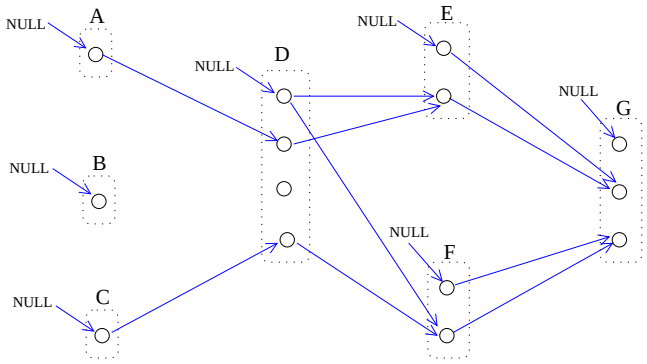


Figure 5.11: Second-order transformation of graph $\mathcal{G}$ in Figure 5.10. A set of vertices indicated with dotted lines correspond to each vertex of $\mathcal{G}$.

is created in each set. For instance, the extra vertex in set D corresponds to the case where the path starts with d in $\mathcal{G}$, i.e., a NULL parent. The following lemma gives the rationale behind this transformation:

Lemma 5.4 *Consider two vertices w and v in $\mathcal{G}$ such that the maximum path length from w to v is at most 2. If (w, v) is a forbidden pair, then there exists no path from vertex set W to vertex set V in $\mathcal{G}^2$.*

PROOF. If the maximum path length from w to v is 1, then the proof is straightforward. Otherwise, consider any path of the form $w \rightarrow u \rightarrow v$. Assume that w is the i^{th} parent of u, and u is the j^{th} parent of v. Due to rule (1) in Definition 5.2, edges from vertex set W to vertex set U in $\mathcal{G}^2$ can only be to $U[i]$. Due to rule (3), an edge from $U[i]$ to $V[j]$ exists only if (w, v) is not a forbidden pair. Hence, if (w, v) is a forbidden pair, a path from W to V cannot exist. ■

As an example, consider the forbidden pairs (a, f), (b, d), (b, e), (b, f), (c, e) in Figure 5.10, each having a maximum path length of 2 between the pairs. Observe that there is no path in the transformed graph of Figure 5.11 between the corresponding set of vertices.

Lemma 5.5 *If there is a path from w to v in $\mathcal{G}$ such that no pair of vertices on the path is a forbidden-pair, then there will be at least one path of the same length in $\mathcal{G}^2$ from vertex set W to vertex set V.*

PROOF. Since there are no forbidden pairs in the path from w to v in $\mathcal{G}$, only the first rule of Definition 5.2 will apply, and the proof of the lemma follows directly. ■

Theorem 5.2 *If the maximum path length between any forbidden pair (u, v) in $\mathcal{G}$ is at most 2, then the longest path in $\mathcal{G}^2$ is the exact solution to LPFP problem on $\mathcal{G}$.*

PROOF. It follows directly from Lemma 5.4 and 5.5. ■

Definition 5.3 *Third-order transformation of $\mathcal{G}$ (denoted as $\mathcal{G}^3$) is defined as follows:*

- *For each vertex u in $\mathcal{G}$, there is a 2-D array of vertices U in $\mathcal{G}^3$ such that $U[i][j]$ corresponds to the i^{th} parent of u and the j^{th} parent of i^{th} parent of u. In other words, for each parent-grandparent pair of u, there exists a corresponding vertex in set U.*
- *There exists an edge between $U[i][j]$ and $V[k][l]$ in $\mathcal{G}^3$ iff:*
  1. *u is the k^{th} parent of v in $\mathcal{G}$.*
  2. *$l = i$.*
  3. *(u, v) is not a forbidden pair.*
  4. *(i^{th}-parent-of-u, v) is not a forbidden pair.*
  5. *(j^{th}-parent-of-i^{th}-parent-of-u, v) is not a forbidden pair.*

Figure 5.12 illustrates the third-order transformation of the graph given in Figure 5.10. Here, it is again assumed that the first parent of each vertex is NULL. For instance, $G[2][1]$ (i.e., the first vertex on the second row of vertex set G) corresponds to the vertex pair $(e, NULL)$ in the original graph, since e is the second parent of g, and NULL is the first parent of e.

Lemma 5.6 *Consider two vertices w and v in $\mathcal{G}$ such that the maximum path length from w to v is at most 3. If (w, v) is a forbidden pair, then there is no path from vertex set W to vertex set U in $\mathcal{G}^3$.*

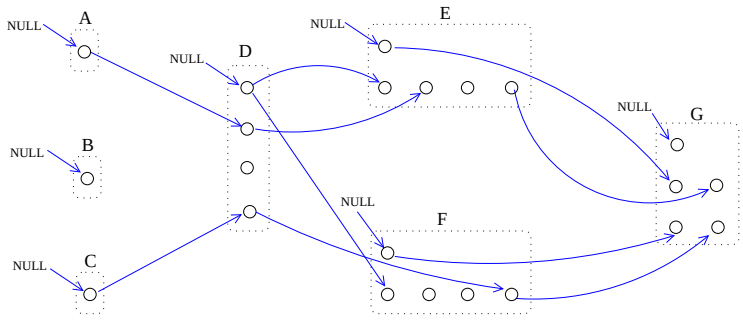


Figure 5.12: Third-order transformation of graph $\mathcal{G}$ in Figure 5.10. A set of vertices indicated with dotted lines correspond to each vertex of $\mathcal{G}$.

PROOF. If the maximum path length from w to v is 1 or 2, then the proof is similar to that of Lemma 5.4. Otherwise, consider any path of the form $w \rightarrow y \rightarrow u \rightarrow v$, where w is the j^{th} parent of y, y is the i^{th} parent of u, and u is the k^{th} parent of v. Any edge in $\mathcal{G}^3$ from vertex set W to vertex set Y can only be to $Y[j][.]$ due to rule (1) in Definition 5.3. Similarly, any edge from $Y[j][.]$ to vertex set U can only be to $U[i][j]$ due to rules (1) and (2) in Definition 5.3. Finally, an edge from $U[i][j]$ to vertex set V exists only if (w, v) is not a forbidden pair, due to rule (5). So, a path from w to v cannot exist if w and v conflict with each other. ■

Observe in Figure 5.12 that there is no path between vertex sets corresponding to the forbidden pairs in Figure 5.10. For example, (a, g) is a forbidden pair, and there is no path between vertex set A and vertex set G in the transformed graph.

Lemma 5.7 *If there is a path from w to v in $\mathcal{G}$ such that no pair of vertices on the path is a forbidden pair, then there will be at least one path of the same length in $\mathcal{G}^3$ from vertex set W to vertex set V.*

PROOF. Since there are no forbidden pairs in the path from w to v in $\mathcal{G}$, only the first and second rules of Definition 5.3 will apply, and the proof of the lemma follows directly. ■

Theorem 5.3 *If the maximum path length between any forbidden pair (u, v) in $\mathcal{G}$ is at most 3, then the longest path in $\mathcal{G}^3$ is the exact solution to LPFP problem on $\mathcal{G}$.*

PROOF. It follows directly from Lemma 5.6 and 5.7. ■

Let $\mathcal{G}_C$ denote the acyclic checkerboard graph structure described in Section 5.4.2. Due to Lemma 5.1, we can apply a third-order transformation on $\mathcal{G}_C$ to obtain $\mathcal{G}_C^3$,

and we can find the exact solution to the LPFP problem by using a linear-time longest path algorithm [16] on $\mathcal{G}_C^3$. From Theorem 5.1, this solution corresponds to the maximal planar routing solution to our original problem.

The following theorem gives the asymptotic time complexity of this algorithm.

Theorem 5.4 *Let s and n denote the size of the components, and the number of nets, respectively. The time complexity of the exact algorithm proposed in this section is $O(ns^{3/2} + s^{5/2})$.*

PROOF. Each vertex v in $\mathcal{G}_C$ has $O(s^{1/2})$ parents, and $O(s^{1/2})$ grandparents, due to Lemma 5.2. Since there is a vertex $V[i][j]$ in $\mathcal{G}_C^3$ corresponding to each parent i and grandparent j of v in $\mathcal{G}_C$, the number of vertices in $\mathcal{G}_C^3$ will be $O(s) \times O(n+s) = O(ns + s^2)$. Note here that $O(n + s)$ is the number of vertices in $\mathcal{G}_C$, as stated in Lemma 5.3. Furthermore, closer examination of Definition 5.3 will reveal that the number of edges entering to each vertex $V[k][l]$ in $\mathcal{G}_C^3$ is $O(s^{1/2})$. Hence, the total number of edges in $\mathcal{G}_C^3$ will be $O(ns^{3/2} + s^{5/2})$. Since the longest path algorithm used on acyclic $\mathcal{G}_C^3$ has linear time complexity in terms of the input graph size, the total time complexity of our algorithm is $O(ns^{3/2} + s^{5/2})$. ■

Although this time complexity is acceptable for moderate component sizes, the algorithm might not be scalable for very large circuits. In the next subsection, we propose a scalable randomized algorithm as an effective alternative for large circuits.

5.4.4 Randomized Algorithm for LPFP

As stated by Lemma 5.1, the vertices that conflict with each other are always *close* to each other in graph $\mathcal{G}$. Intuitively, if we somehow generate subpaths by grouping the *nearby* vertices together, then we can obtain a graph where there are no conflicts between groups that are far away from each other. The algorithm we propose in this section makes use of this idea, and uses randomization to group the nearby vertices together, and handle forbidden pairs accordingly.

Figure 5.13 gives the outline of the randomized algorithm we propose for the checkerboard graph model described in Section 5.4.2. The first step here is to define subproblems on the checkerboard structure, as shown in Figure 5.15. Then, we randomly generate a predefined number of *permissible* subpaths for each subproblem. Figure 5.14 gives the algorithm we use to generate random subpaths for one subproblem. Observe that for each checkerboard cell C at the last row of a subproblem, we

RANDOM-LPFP

Define horizontal subproblems (with 3 rows) on the checkerboard

Randomly generate subpaths P_j^i within each subproblem i

Create a graph $\mathcal{G}_R$ as follows:

–A vertex v_j^i exists in $\mathcal{G}_R$ corresponding to each subpath P_j^i

–Weight of v_j^i is equal to size of P_j^i

–An edge from v_j^i to v_k^{i+1} exists iff:

(1) P_k^{i+1} is completely to the south-east of P_j^i

(2) The last element of P_k^{i+1} is separated from the last element of P_j^i by at least 2 columns

(3) There exists no forbidden pair (u, w) such that $u \in P_j^i$ and $w \in P_k^{i+1}$

Return the longest path in $\mathcal{G}_R$

Figure 5.13: Randomized algorithm for LPFP problem on a checkerboard graph where the maximum path length between any forbidden pair is at most 3.

keep the K/r longest subpaths ending at C. Note that our purpose here is not just to find the best possible subpath, but instead to find various (possibly on the order of thousands) *good* subpaths for each subproblem. After that, we merge them in an optimal way by applying a longest path algorithm on the directed acyclic graph $\mathcal{G}_R$, which is defined in Figure 5.13. The following lemma explains the rationale behind this model:

Lemma 5.8 *Consider two subpaths P_j^i and P_k^l $(i < l)$ in subproblems i and l, respectively. If there is a forbidden pair (u, w) such that $u \in P_j^i$ and $w \in P_k^l$, then there exists no path between the corresponding vertices v_j^i and v_k^l in $\mathcal{G}_R$.*

PROOF. If $l = i+1$, this check is done explicitly by rule (3), as given in Figure 5.13. Otherwise, assume that $l \geq i+2$, and there is a path from P_j^i to P_k^l in $\mathcal{G}_R$. It is obvious that P_j^i and P_k^l are separated by at least 3 checkerboard rows, since there is at least one subproblem between them. Furthermore due to rule (2), there are at least 3 columns between the last element of P_j^i, and the first element of P_k^l. Since the maximum path length between a forbidden pair can be at most 3 in the original graph (as stated in Lemma 5.1), there exists no forbidden pair (u, w) such that $u \in P_j^i$ and $w \in P_k^l$. ■

GENERATE-SUBPATHS(Subproblem i: between rows T_i and B_i)

for a fixed number of M iterations do:

  $u \leftarrow$ a random vertex at row T_i

  $P \leftarrow \{u\}$ // initialize the subpath

  repeat:

    $v \leftarrow$ a random vertex for which edge $u \rightarrow v$ exists,

      and (w, v) is not a forbidden pair for any $w \in P$

    $P = P \cup \{v\}$

    $u \leftarrow v$

  until v is not at row B_i

  Let C be the checkerboard cell that contains the last v

  Let $\mathcal{P}_C$ denote the set of previously recorded subpaths ending at $\mathcal{C}$.

  If $|\mathcal{P}_C| < K/r$, where r is the number of component rows

    record P

  else if there exists a subpath $P' \in \mathcal{P}_C$ such that P' is shorter than P

    replace P' with P

  else

    discard P

Figure 5.14: Algorithm to generate a set of $O(K)$ random subpaths between rows T_i and B_i of the checkerboard.

Due to this lemma, we can use a simple longest path algorithm on $\mathcal{G}_R$ without the need of checking forbidden pairs. This longest path will correspond to the optimal combination of subpaths that were randomly generated. If we can generate a large variety of random paths, we can expect the final solution to be sufficiently close to the optimal planar routing solution.

For the complexity analysis of this randomized algorithm, let us first focus on the subpath generation phase given in Figure 5.14. In one iteration of this algorithm, a subpath P is generated and evaluated. Generation of one subpath P takes constant time, since P can contain at most 3 routing patterns, by definition. The evaluation of P can also be performed in constant time by using efficient bucket-based data structures.[6] Since M iterations are performed in Figure 5.14 for one subproblem, and there are $O(r)$ subproblems (where r is the number of rows in the components), the total time complexity of the subpath generation phase is $O(Mr)$. Note here that the number of subpaths recorded at the end of this phase for each subproblem is $O(K)$, where $K \leq M$. After this phase, a graph $\mathcal{G}_R$ is created, as shown in Figure 5.13. The number of vertices in $\mathcal{G}_R$ is equal to the total number of recorded subpaths, which is $O(Kr)$. The edges in $\mathcal{G}_R$ are only between vertices that correspond to adjacent subproblems. Hence the number of edges in $\mathcal{G}_R$ is $O(K^2r)$. As mentioned before, computing the longest path for a directed acyclic graph (dag) has linear time complexity in terms of the input graph size [16]. As a result, the total time complexity of this algorithm is $O(Mr+K^2r)$. Here, we can set K and M to large values (possibly on the order of thousands) so that a large number of subpaths are generated for each subproblem, and various path combinations are explored for the solution. Yet the algorithm will still have good run-time characteristics, as will be demonstrated in Section 5.6.

Figure 5.15(a) illustrates a sample checkerboard with 9 rows, and 3 subproblems. For each subproblem, a subpath is selected, and they are merged to obtain a path of 8 routing patterns. The solution corresponding to this path is illustrated in Figure 5.15(b).

[6]Namely, we can create 3 buckets for each checkerboard cell C, each bucket corresponding to a linked list of subpaths having the same length. Using this, we can find a subpath $P' \in \mathcal{P}_C$ such that P' is shorter than P, and replace it with P in constant time.

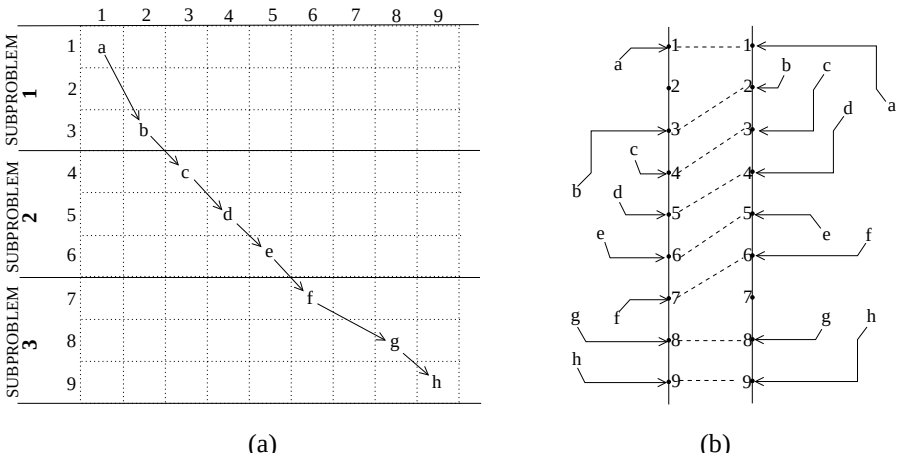


Figure 5.15: (a) A sample checkerboard structure with 3 subproblems. The selected subpaths in each subproblem are $\{a11, b32, c43\}$, $\{d54, e65, f76\}$, and $\{g88, h99\}$, respectively. (b) The corresponding escape routing solution.

5.5 Generalizations of the Algorithms

In the algorithms of Section 5.4, we have considered only 16 routing patterns for each net. The rationale behind this assumption has been discussed in Section 5.4.1. However, it is also possible to extend our algorithms such that more routing patterns are considered. Assume that a net is allowed to escape from one of the V neighboring rows of its terminal. (We have assumed that $V = 4$ in the previous sections). The graph model described in Section 5.4.1 can be used with small modifications for different V values. However, for the exact maximal planar routing algorithm in Section 5.4.3, we would need a $(V-1)^{st}$*-order transformation* on the input graph. Note that the size of the transformed graph would be exponential in V, and this approach could be impractical for large V values. However, the randomized algorithm we propose in Section 5.4.4 can easily be generalized for arbitrary V values. Namely, only two modifications are needed in the algorithm described in Figure 5.13. First, the subproblem sizes need to be $V-1$, instead of 3. Then, the second rule for edge creation in $\mathcal{G}_R$ needs to be changed such that P_k^{i+1} and P_j^i are separated by $V-2$ columns, instead of 2 columns. Hence the randomized algorithm would still be scalable for large V values. Furthermore it is also possible to generalize the types of escape patterns used in the proposed algorithms, as will be discussed in detail in Chapter 6.

Another assumption we have made in the previous sections is that the problem

consists of two components separated by a channel. For a general design, we can apply these algorithms on different pairs of components independently. As future work, we need an algorithm that automatically identifies the best pairs of components to be routed on each layer of a complex design. Once the component pairs are identified, the algorithms given in this chapter can be applied on each pair independently. For a typical industrial board today, it is reasonable to expect large bus structures (each containing a large number of nets) between different pairs of components. Furthermore, in high-speed designs, there are additional spacing requirements between nets belonging to different buses due to noise considerations. For such designs, it is highly preferable to route nets belonging to the same bus together, and to minimize adjacencies between nets belonging to different bus structures. So, identifying different component pairs and solving the escape problem for each pair separately will be an effective approach.

It is also possible to merge more than one component to obtain a (conceptual) super-component, and apply our algorithms on super-component pairs. As an example, consider Figure 5.16, which illustrates components from a real industrial design. For this circuit, we can define two separate subproblems: (1) two memory modules on the left, and the left half of the MCM, and (2) two memory modules on the right, and the right half of the MCM. For the first subproblem, we can define one super-component as the concatenation of the two memory units on the left, and the other super-component as the two quadrants on the left half of the MCM. Applying our algorithms on these super-components will give an escape routing solution for the three components simultaneously. Section 5.6 gives further details about our experiments on this and other industrial problems.

5.6 Experimental Results

For evaluation of our algorithms, we have extracted escape problems corresponding to different components of an industrial circuit from IBM, for which the current industrial routers fail to produce a routing solution. Typically, the industrial tools do not use the problem formulation proposed in this chapter. A common approach used to route board designs in the industry is to perform global routing, followed by iterations of rip-up and reroute techniques. However, such an approach fails when there are a large number of nets of which terminals are inside very dense pin arrays.

Table 5.1: Comparison of randomized and exact algorithms

			EXACT-PLANAR		RANDOM-PLANAR	
Input	# nets	# layers	# planar nets	time (min:sec)	# planar nets	time (min:sec)
IBM_MEM1	213	4	196	3:34	198	0:31
IBM_MEM2	213	4	191	3:33	190	0:34
IBM_STI	352	5	319	21:52	313	1:44

As discussed before, applying escape routing on each component independently is not an effective approach, either.

For experimental comparisons, we have used a special implementation of the Pathfinder [19] algorithm that recognizes the special property of this problem. In particular, a special graph structure is used so that the dense component areas are modeled as detailed grids, and the intermediate areas between components are modeled as coarse-grain connections. The purpose here is to find the detailed escape routing solutions inside dense components, while minimizing the number of crossings in the intermediate areas. We have implemented all our algorithms in C++, and performed our experiments on an AMD Athlon 1.3 GHz system with 512MB memory, and a Linux operating system. For the randomized algorithm, we have used a fixed random seed throughout our experiments. We have observed that changing the random seed does not have a considerable effect on the routing results.

First, we have performed experiments to evaluate the effectiveness of the randomized maximal planar routing algorithm given in Section 5.4.4. Table 5.1 gives comparison of this algorithm with the exact algorithm described in Section 5.4.3.

Note that the exact algorithm is guaranteed to route maximum number of planar nets on one layer. However, it does not guarantee the optimal result on multiple layers, since we process one layer at a time. As can be seen from this table, the randomized algorithm gives results almost as good as the exact algorithm, requires less running time, and is more scalable for larger circuits. Therefore, we have used the randomized algorithm as the underlying maximal planar routing algorithm in the next set of experiments.

We have then implemented the methodology described in Section 5.3. Namely, the maximal planar routing solution is found for each layer, and then the remaining nets are distributed to all layers at the end. For comparison purposes, we have used the Pathfinder [19] based algorithm described above. We have fine-tuned this algorithm

Table 5.2: Comparison of our methodology with a net-by-net approach

			OUR METHOD		NET-BY-NET	
Input	# nets	# layers	# crossing nets	time (min:sec)	# crossing nets	time (min:sec)
IBM_MEM1	213	4	8	0:38	41	5:14
IBM_MEM2	213	4	19	0:43	32	4:33
IBM_STI	352	5	24	0:27	62	11:23
IBM_MEMG1	452	8	82	2:59	164	62:51
IBM_MEMG2	454	8	101	2:24	174	52:32

such that the number of crossing nets (in the channel) is minimized. Table 5.2 gives comparison of the results. Here, the number of crossing nets can also be viewed as the number of nets that need to use vias in the *area routing stage*. Observe that our methodology results in substantially less number of crossing nets for all problems. On average, 14% and 28% of all nets are crossing in the solution of our methodology, and the net-by-net approach, respectively. So, we can say that our algorithms reduce the via requirements significantly. Furthermore, the execution times of our method are much lower, since we calculate the best set of planar nets simultaneously in an efficient way. On the other hand, the net-by-net approach requires multiple iterations to *negotiate* routing resources among different nets.

We also illustrate a sample solution for one layer of a circuit in Figure 5.16. Actually, this figure contains two separate problems: (1) the memory units on the left and MCM, and (2) the memory units on the right and MCM. As mentioned in Section 5.5, we have grouped multiple components together to obtain two *super-components* separated by a channel, for each problem. Although the exact area routing will be determined by a later stage, we also display the non-crossing channel segments in this figure.

5.7 Conclusions

We have proposed an exact and a randomized algorithm for simultaneous escape routing and layer assignment problem for boards with dense components. The experimental results show that the randomized algorithm gives results as good as the exact algorithm, and is much faster. We also show that the methodology we propose produces considerably better results than a net-by-net approach.

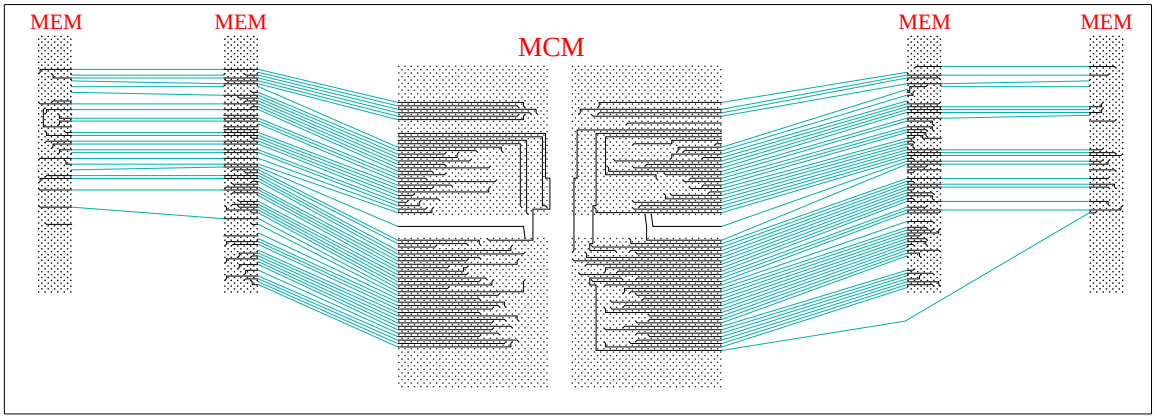


Figure 5.16: A sample solution for one layer (out of 8) of a problem containing an MCM and 4 memory units. The non-crossing channel connections are illustrated as straight (dotted) lines between components, while the escape routing solutions are shown with solid lines inside the components. 120 (out of 906 total) nets have been assigned to this layer, and 109 of them have non-crossing channel segments.

Chapter 6

An Escape Routing Framework for Dense Boards with High-Speed Design Constraints

6.1 Introduction

In Chapter 5, we proposed fundamental algorithms for solving the escape routing problem in multiple components simultaneously. However, these algorithms have been given under some simplifying assumptions for the ease of presentation. In this chapter, we generalize these models and algorithms, and present an improved escape routing framework targeted for dense boards with high-speed design constraints.

We propose an escape routing framework in this chapter for the purpose of solving the escape routing problem in multiple components simultaneously, so that the number of crossings in the intermediate area is minimized, and high-speed design constraints are satisfied. Figure 6.1 illustrates a one-layer escape routing solution for two components. In this figure, nets have been routed from their terminal pins to the corresponding component boundaries. Here, only one net (net D) crosses with the others in the intermediate area. For this net, the area router will need to use a via to resolve the crossing. As mentioned before, the number of crossings in the intermediate area is a good measure for the via requirements of an escape routing solution.

Compared to the algorithm proposed in Chapter 5, this algorithm brings three main improvements: (1) more general escape patterns are considered within the

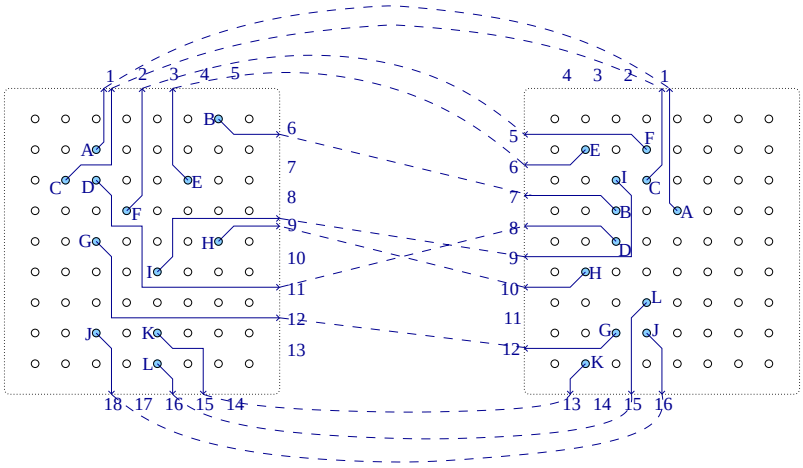


Figure 6.1: An escape routing solution for 12 nets. The escape slots are identified on the boundaries of components. The connections in the intermediate area are shown by dashed lines.

framework, instead of simple straight connections (Section 6.3); (2) an improved maximal planar route selection algorithm is proposed, which is general enough to handle multi-capacity escape slots, and high-speed design constraints (Section 6.4); and (3) explicit discussion about handling various high-speed design constraints is given for this framework (Section 6.5). Our experiments in Section 6.6 show that our algorithm reduces the via requirements of industrial test cases on average by 39%, compared to the basic algorithm proposed in Chapter 5.

The rest of this chapter is organized as follows. We give the formal description of this problem in Section 6.2. Our methodology to solve this problem is based on generating a number of different routing alternatives for each net and then selecting the maximum planar subset of escape patterns on each layer. In Section 6.3, we propose an algorithm to generate escape patterns based on congestion levels within the components and the crossings in the intermediate area. Then, we propose a randomized algorithm in Section 6.4 for the problem of maximum planar route selection. In Section 6.5, we discuss how to handle high-speed design constraints within the framework of this algorithm. Finally, we demonstrate the effectiveness of this algorithm on industrial test cases in Section 6.6.

6.2 Problem Formulation and Methodology

We will present our generalized models and algorithms in such a way that the chapter is overall self-contained. The theoretical results given in Chapter 5 are still relevant in this chapter. However, we will adapt a slightly different presentation, which is more suitable for the general escape routing framework that will be proposed in this chapter.

Let a *component* be defined as a 2-D array of pins, where each pin spans multiple layers, and routing tracks are defined on each layer between adjacent rows and columns of pins. An *escape segment* is defined to be a route from a pin inside the component to an *escape slot* on the component boundary. For a component, a set of *escape slots* are defined on its boundary, defining the permissible end-points of escape segments originating from the pins. Due to limited routing resources, buried vias are not allowed inside the components. So, routing within the component area needs to be planar on every layer. Two escape segments corresponding to two different nets are defined to have a *conflict* inside the component if they cannot be routed together on the same layer in a planar fashion. The number of routing tracks at each row and column is pre-determined based on the pin diameters, wire widths, pin spacings, and clearance constraints. In a feasible solution, the number of escape segments passing through a row/column of the component cannot exceed the corresponding capacity of that row/column.

Let us assume that the input problem consists of only two components, which are denoted as *left* and *right* components, respectively, for simplicity of presentation. A *net* is assumed to have two terminals, one in each component. An escape pattern P_i for net i is defined to be the combination of two escape segments originating from the terminals of net i in the left and right components. Two escape patterns P_i and P_j corresponding to nets i and j are defined to have a *conflict* iff their escape segments have conflicts within at least one component. Note here that a pair of non-conflicting escape patterns P_i and P_j can have a *crossing* in the intermediate area between components, depending on the relative ordering of their escape slots. Since buried vias are allowed in the intermediate area between components, these crossings are allowed in a feasible solution. However, the number of crossings need to be minimized for the objective of via minimization.

Based on these definitions, the simultaneous escape routing problem for a set of

nets can be stated as follows: *Find an escape pattern P_i for each net i, and assign it to a layer such that: (1) no pair of escape patterns on the same layer conflict with each other, (2) the capacity constraints on all rows and columns of the components are satisfied, and (3) the number of crossings in the intermediate area is minimized.*

Figure 6.1 illustrates a sample one layer solution for 12 nets. The number of escape slots in the left and right components are 18 and 16, respectively. While the slots on the left component are numbered increasing in the clockwise direction, the slots on the right component are numbered increasing in the counter clockwise direction. Among the 12 nets routed on this layer, only one (net D) crosses with the others in the intermediate area. Some of the escape slots (slots 1 and 9 in the left component, slot 1 in the right component) are used by more than one escape segments. This is allowed in a feasible solution as long as the capacity constraints are not violated.

Our methodology to solve this problem is similar to the one proposed in Chapter 5. Namely, we process one layer at a time, and route as many noncrossing nets as possible on each layer. After finding a maximal planar routing solution for all layers, we distribute the remaining nets to available layers, this time allowing crossings in the intermediate area. In the rest of the chapter, we will focus on the problem of maximal planar routing. For the second phase, we use a Pathfinder-based algorithm to distribute the remaining nets to available layers.

Our algorithm for maximal planar routing consists of two phases: (1) Generate a number of different routing alternatives for each net, and (2) select the maximal subset of routing patterns that will give a feasible planar routing solution for the current layer. Compared to the algorithm of Chapter 5, our main contributions in this chapter can be summarized as follows. For the first phase, we propose an algorithm to generate routing patterns based on the congestion levels inside the components, and the number of crossings in the intermediate area (Section 6.3). For the second phase, we propose a more sophisticated randomized algorithm, which can also be generalized to handle high-speed design constraints (Sections 6.4 and 6.5).

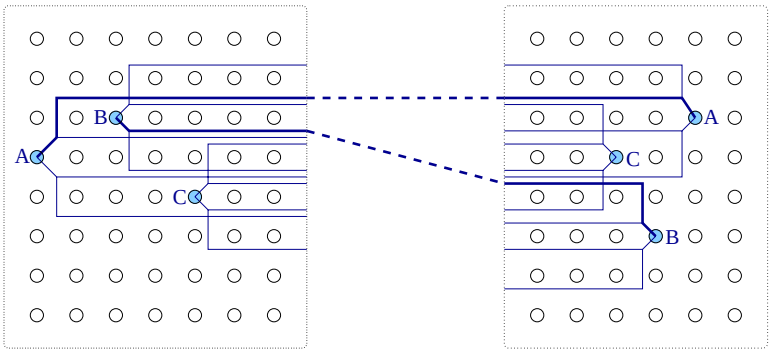


Figure 6.2: The output of a simple pattern generation technique for 3 nets. The maximal planar subset is highlighted with bold lines.

6.3 Escape Pattern Generation

6.3.1 Motivation

In this section, we describe an algorithm to generate a number of different routing alternatives for each net. In Chapter 5, a simple pattern generation methodology has been used for this purpose. In particular, 4 escape segments are generated for each net within each component, for a total of $4 \times 4 = 16$ escape patterns. The escape segments generated in the basic algorithm have vertical spans of at most 2 rows, as illustrated in Figure 6.2. The justification here is that escape patterns with large vertical spans block other patterns; so small vertical spans are needed for maximal planar routing. However, nonregular escape patterns with larger vertical spans, as illustrated in Figure 6.3, may be helpful in some situations. For example if we use the simple pattern generation technique of Chapter 5, as in Figure 6.2, then only 2 out of 3 nets will be routed in a planar fashion, as highlighted in the figure. However, if we use a more intelligent pattern generation algorithm as in Figure 6.3, we can route all 3 nets in a planar fashion. With this motivation, we propose an algorithm in this section that generates escape patterns based on congestion levels within the components, and the crossings in the intermediate area.

Our objective here is to generate escape patterns with low congestion levels inside the components, and small number of crossings in the intermediate area. However, the patterns generated need to satisfy the following two properties:

Consider any pair of escape segments S_i and S_j generated within the same component. Let V be a constant input parameter.

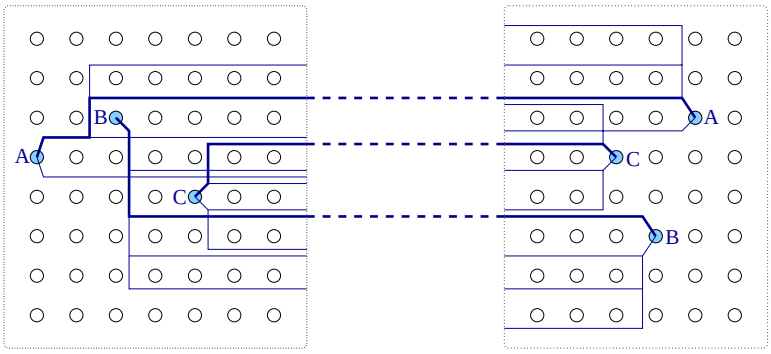


Figure 6.3: Pattern generation with the objective of low congestion levels, and small number of crossings. The maximal planar subset is highlighted with bold lines.

Property 6.1 *If S_i and S_j correspond to the same net (i.e., S_i and S_j originate from the same terminal), then it must be the case that $|S_i.slot - S_j.slot| < V$.*

Property 6.2 *If S_i and S_j have a* conflict*, then it must be the case that $|S_i.slot - S_j.slot| < V$.*

Here, the notation $S.slot$ denotes the index of the *escape slot* of segment S, as defined in Section 6.2. Intuitively, the segments belonging to the same net, and the conflicting segments must escape to slots that are *close* to each other on the component boundary. In the examples of Figure 6.2 and 6.3, these two properties hold for $V = 4$. As will be discussed in detail in Section 6.4, our *maximal planar route selection* algorithm will be based on the assumption that these two properties hold. Furthermore, we will show in Section 6.4 that a polynomial-time optimal algorithm exists for *maximal planar route selection* problem if these two properties hold for a constant V value.

6.3.2 The Algorithm

Given a simultaneous escape routing problem, as defined in Section 6.2, we start with sorting all the net terminals based on their distances to the closest escape slots on the component boundaries. Then, we process these terminals in sorted order, starting from the terminal closest to an escape slot. Originating from each terminal, we generate a number of escape segments, by using the algorithm outlined in Figures 6.4 and 6.5.

```
GENERATE-ESCAPE-SEGMENTS(G, t, V, T)
  // G: the grid graph corresponding to the component
  // t: the terminal from which the segments will be generated
  // V: the input parameter
  // T: the set of target escape slots
 for index ← 1 to V do
    S ← GENERATE-ONE-ESCAPE-SEGMENT(G, t, T)
    add S to the set of escape segments originating from t
    T ← T ∩ ({s : S.slot−V < s < S.slot+V}\{S.slot})
      // limit the target slot range to satisfy Property 6.1
```

Figure 6.4: High-level algorithm to generate a number of V escape segments originating from terminal t.

Figure 6.4 displays the high-level algorithm used to generate a number of routing segments originating from a given terminal t. Here, graph G is used to model the routing resources of the input component. As described in Section 6.2, a component is assumed to be a 2-D array of pins, with rows and columns of routing tracks between adjacent pins. Also, a set of target *escape slots* $\mathcal{T}$ is specified for terminal t as input to the algorithm of Figure 6.4. Although $\mathcal{T}$ can be set such that it contains all escape slots on the component boundary, it is also possible to set it based on the length constraints of the corresponding net, as will be discussed in Section 6.5. Observe in Figure 6.4 that after an escape segment S is generated from terminal t, the set $\mathcal{T}$ is restricted to the escape slots that are within the neighbourhood of escape slot of S. The purpose here is to make sure that Property 6.1 is maintained for the segments generated from terminal t.

Before describing the low-level algorithm, we need to make the following definition:

Definition 6.1 *Let $v.segments$ denote the set of escape segments passing through vertex v. An escape slot s is defined to be* reachable *from vertex v iff for each escape segment $S \in v.segments$, it is the case that $|S.slot - s| < V$, where V is the input parameter specified in Property 6.2. The set of* reachable *escape slots from vertex v is denoted as $v.reachableSlots$*

Remark 6.1 *Consider a path P in grid graph G that starts at terminal t, and ends*

```
GENERATE-ONE-ESCAPE-SEGMENT(G, t, T)
pQ ← an empty priority queue
for each vertex v ∈ G that is adjacent to terminal t do
  v.label ← 0
  v.targetSlots ← T
  pQ ← pQ ∪ {v}
while pQ not empty do
  u ← pQ.extractMin()
  if u corresponds to an escape slot then terminate loop
  for each edge (u → v) ∈ G do
    if (u.targetSlots ∩ v.reachableSlots ≠ ∅)
        && (u.label + cost(u → v) < v.label) then
      v.label ← u.label + cost(u → v)
      v.targetSlots ← u.targetSlots ∩ v.reachableSlots
        // limit the target slot range to satisfy Property 6.2.
      v.parent ← u
      pQ ← pQ ∪ {v}
construct escape segment S by backtracking from escape slot u
```

Figure 6.5: Low-level algorithm to generate one escape segment originating from terminal t.

at escape slot s. If $s \in v.reachableSlots$ for each $v \in P$, then it is guaranteed that path P satisfies Property 6.2.

The low-level algorithm used to generate one escape segment is given in Figure 6.5. This is basically a variant of Dijkstra's shortest path algorithm [16]. As an additional constraint, we make sure that Property 6.2 is satisfied, by restricting the target slot range when a conflict with an existing pattern is possible. The cost of edge $(u \rightarrow v)$ is computed by the following formula:

$$cost(u \rightarrow v) = \alpha.dist(u \rightarrow v) + \beta.cong(v) + \gamma.cross(v) \tag{6.1}$$

Here, α, β, and γ are scaling factors for *distance*, *congestion*, and *crossing* cost metrics, respectively. Congestion cost for vertex v is computed based on the number of escape segments passing through v. Before generating any escape pattern, we

first estimate the congestion values for individual vertices through path analysis. As we generate escape segments, we gradually replace these estimations with actual congestion values. If v is a vertex corresponding to an escape slot, we also compute a crossing cost based on the estimated number of crossings in the intermediate area.

6.4 Maximal Planar Route Selection

In this section, it is assumed that a number of escape patterns that maintain Property 6.1 and Property 6.2 have been generated. The objective now is to select the maximum number of escape patterns such that: (1) at most one pattern for each net is selected, (2) the segments of the selected patterns do not *conflict* with each other within components (i.e., they are routable in a planar fashion on the same layer), and (3) the selected patterns have no crossing in the intermediate area.

6.4.1 Problem Modeling

Let $P.slotL$ and $P.slotR$ denote the escape slots of escape pattern P in the left and right components, respectively. Furthermore, let us assume that a unique rank is assigned for each escape segment within a component, indicating the relative ordering between different segments. As an example, consider the segments of nets I and H in the left component of Figure 6.1. Although these two segments escape to the same slot (slot 9), the rank of net I's segment must be less than the rank of the corresponding segment of net H. Let $P.rankL$ and $P.rankR$ denote the rank of pattern P in the left and right components, respectively. For simplicity of presentation, we will first consider the problem with unit slot capacities in the following definitions.

Definition 6.2 *The* less-than predicate *for two escape patterns is defined as follows:* $P_i \prec P_j$ *iff* $P_i.rankL < P_j.rankL$ *and* $P_i.rankR < P_j.rankR$.

Note here that the precedence relation defined above is *transitive*; i.e., if $P_i \prec P_j$ and $P_j \prec P_k$, then $P_i \prec P_k$. Based on this property, we can give the following definitions:

Definition 6.3 *A pattern sequence* $\mathcal{S}$ *is defined to be an ordered set of patterns* $\{P_1, P_2, ..., P_n\}$ *such that if* $i < j$ *then* $P_i \prec P_j$.

Definition 6.4 *A pattern sequence $\mathcal{S}$ is defined to be* permissible *iff it contains no pair of conflicting patterns.*[1]

Theorem 6.1 *For a given set of escape patterns, the longest permissible pattern sequence $\mathcal{S}$ is equivalent to the maximum subset of patterns that can be routed on one layer in a planar fashion.*

Theorem 6.2 *For a given set of escape patterns, assume that Property 6.1 and Property 6.2 are satisfied for a constant V value. Then, there is a polynomial-time optimal algorithm to solve the maximal planar route selection problem.*

PROOF. As given in Theorem 6.1, the maximal planar route selection problem is equivalent to finding the longest permissible pattern sequence among a given set of patterns. If Property 6.1 and 6.2 are satisfied, then we can use a dynamic programming algorithm to solve this problem. As an example, consider the simplified problem where $V = 1$; i.e., there is only one pattern corresponding to each net, and no pair of patterns conflict with each other. In this case, a simple dynamic programming based algorithm that computes the longest sequence ending at each pattern will be sufficient. We can use the same intuition to devise an algorithm for the general case, where V has an arbitrary constant value. Let $\mathcal{P}_V$ be the set of all different permutations of the given patterns with size V. The main idea here is to compute each longest sequence that has its last V patterns the same as an element of set $\mathcal{P}_V$. For example, let us consider $p_v \in \mathcal{P}_V$ (where p_v consists of V patterns). We can find the longest sequence that has its last V elements the same as p_v in $O(n)$ time. (We need to consider the longest subsequences that have its last $V-1$ patterns the same as the first $V-1$ patterns in p_v.) Based on these ideas, we can show that such an algorithm will have a time complexity of $o(n^{V+1})$, where n is the number of nets, and V is a constant value. Note here that this is a loose upper bound, and more efficient algorithms can be devised for fixed V values. In particular, the exact algorithm proposed for $V = 4$ in Chapter 5 has a time complexity of $O(ns^{3/2} + s^{5/2})$, where s is the size of the components. ■

Although a polynomial-time optimal algorithm exists for this problem, its high time complexity makes it impractical for large circuits. For this reason, we propose a

[1]We denote two patterns P_i and P_j as *conflicting* iff they cannot occur together in a valid planar escape routing solution.

PLANAR-ROUTE-SELECTION($\mathcal{P}$: a set of patterns)

map each pattern in $\mathcal{P}$ to a cell of checkerboard $\mathcal{C}$

rowwise partition $\mathcal{C}$ into subproblems with $V-1$ rows each

randomly generate a set of subsequences in each subproblem

create a graph $\mathcal{G}_R$ as follows:

- A vertex v_j^i exists in $\mathcal{G}_R$ corresponding to each subsequence S_j^i in subproblem i.
- The weight of v_j^i is equal to the number of patterns in S_j^i.
- Let x_j^i and x_k^{i+1} denote the x coordinates of the checkerboard cells of the last patterns in subsequences S_j^i and S_k^{i+1}.
  An edge from v_j^i to v_k^{i+1} exists in $\mathcal{G}_R$ iff:
  (1) all patterns in S_j^i are to the left of all patterns in S_k^{i+1}
  (2) $x_k^{i+1} > x_j^i + V - 2$,
  (3) no pattern in S_j^i conflict with a pattern of S_k^{i+1}.

return the longest path in $\mathcal{G}_R$

Figure 6.6: High-level description of the randomized planar route selection algorithm

much faster randomized algorithm in the next subsection, which gives solutions that are very close to optimum in practice. As mentioned before, we will also discuss how to handle high-speed design constraints within the framework of this algorithm in Section 6.5. The algorithm proposed in the next subsection can also handle multi-capacity slots, as given by the following definition.

Definition 6.5 *Assume that each escape slot is defined to have a particular capacity, as defined in Section 6.2. A sequence $\mathcal{S}$ is defined to be* capacity constrained *iff the number of patterns in $\mathcal{S}$ that use a particular escape slot is less than or equal to the corresponding slot capacity.*

6.4.2 A Randomized Algorithm

In this section, we propose a randomized algorithm to solve the *capacity-constrained longest permissible sequence* problem for a given set of escape patterns. The high-level algorithm is given in Figure 6.6. Compared to the algorithm given in Chapter 5, the main improvement is our randomized subsequence generation algorithm, as given

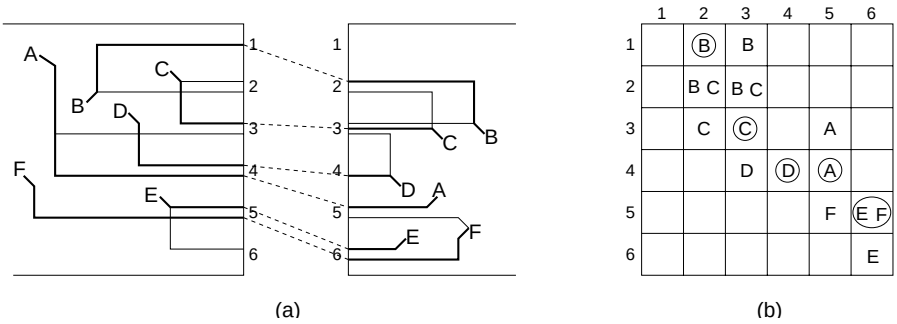


Figure 6.7: (a) A set of routing patterns defined for 6 nets. (b) The corresponding checkerboard model. For clarity, only one or two escape segments are illustrated for each net. The maximum planar subset is highlighted in both figures.

in Figure 6.9. This algorithm not only improves the routing results considerably (Section 6.6), but also is general enough to handle multi-capacity escape slots and typical high-speed design constraints (Section 6.5). We also prove later in this section that the average-time complexity of this algorithm is linear in the component sizes (Theorem 6.3).

The (conceptual) *checkerboard* model introduced in the algorithm of Figure 6.6 is defined in a similar way as in Chapter 5.

Definition 6.6 *Let $\#sL$ and $\#sR$ denote the number of escape slots defined on the left and right components, respectively. Let $\mathcal{C}$ be a (conceptual) checkerboard with $\#sL$ rows and $\#sR$ columns. An escape pattern P is defined to be* mapped *to cell (i, j) of checkerboard $\mathcal{C}$ iff $P.slotL = i$ and $P.slotR = j$.*

Figure 6.7 illustrates a sample escape problem and the corresponding checkerboard model. Let us consider two patterns P_i and P_j on this checkerboard. If P_j is below and to the right of P_i (e.g., $P_i = B12, P_j = C33$), then $P_i \prec P_j$, as defined in Definition 6.2. If P_j is above and to the right of P_i (e.g., $P_i = D43, P_j = A35$), then neither $P_i \prec P_j$ nor $P_j \prec P_i$. Otherwise, if P_i and P_j are on the same row (e.g., $P_i = D44, P_j = A45$), or the same column (e.g., $P_i = C33, P_j = D43$), or the same cell (e.g., $P_i = E56, P_j = F56$), then we need to check the ranks of P_i and P_j to determine the relationship between these patterns. For instance, ranks of $E56$ in both left and right components are less than those of $F56$ (since the corresponding escape segments of net E are above those of net F); hence $E56 \prec F56$.

After mapping each pattern to a checkerboard cell, $\mathcal{C}$ is rowwise partitioned into subproblems. Then a set of *capacity-constrained permissible* subsequences is randomly

generated within each subproblem, as will be described in detail later in this section. After that, these subsequences are merged together to obtain the *capacity-constrained longest permissible sequence*. For this purpose, a graph model $\mathcal{G}_R$ is defined in Figure 6.6, which satisfies the following lemma.

Lemma 6.1 *Consider two subsequences S_j^i and S_k^l in subproblems i and l, respectively. If there is a path between the corresponding vertices v_j^i and v_k^l in $\mathcal{G}_R$, then it is guaranteed that S_j^i and S_k^l contain no patterns that conflict with each other.*

This lemma is similar to Lemma 5.8, which was given for the restricted problem instances of Chapter 5. Based on this lemma, we can compute the longest path in acyclic graph $\mathcal{G}_R$ to find the best combination of subsequences generated in different subproblems. We can then merge these subsequences to obtain the longest permissible sequence. Figure 6.8 illustrates the execution of the randomized algorithm on a sample problem. Here, assume that a number of patterns have already been mapped to this checkerboard, and the conflicts between patterns are as listed in this figure. A small set of randomly generated subsequences[2] is shown for each subproblem on the right. Corresponding to each subsequence, there is a vertex in $\mathcal{G}_R$, and edges between them are created based on the rules defined in Figure 6.6. For instance, there is no edge from $\{A, B, C, E\}$ to $\{I, J, L, M\}$ because patterns E and I are conflicting. Similarly, there is no edge from $\{A, B, F, G\}$ to $\{I, J, L, M\}$, because it violates rule (2) in Figure 6.6. The longest path in $\mathcal{G}_R$, corresponding to the *capacity-constrained longest sequence* is also highlighted in this figure.

The algorithm we use to generate a set of random subsequences is outlined in Figure 6.9. In the beginning, this recursive function is called for each cell on the first row of the given subproblem, with argument *subseq* set to $\emptyset$. In one recursive call, first it is checked whether the partial subsequence generated so far is good enough to store. This decision is made by comparing the weight of the current subsequence *subseq* with the weights of the subsequences already stored for this subproblem. Let tx denote the x-coordinate of the checkerboard cell corresponding to the last pattern in *subseq*. The weight of *subseq* is compared with only the subsequences that end at column tx of the checkerboard. An input parameter determines how many subsequences can be stored

[2]For clarity, only 3 or 4 subsequences are given in this example. Normally, hundreds or even thousands of subsequences are generated for each subproblem to obtain a good variety.

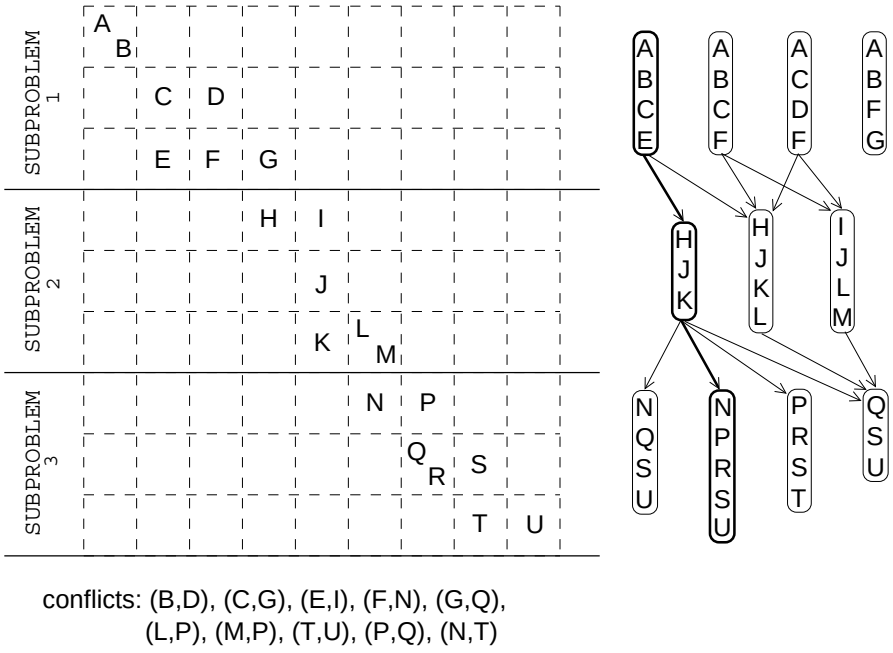


Figure 6.8: Illustration of the randomized algorithm given in Figure 6.6 on a sample checkerboard. For clarity, ranks of the patterns are not displayed. The set of subsequences generated for each subproblem are shown on the right, together with the corresponding graph $\mathcal{G}_R$, and the (highlighted) longest path. It is assumed here that each escape slot has a capacity of two.

GENERATE-SUBSEQ$(x, y, subseq)$

// (x,y): coordinate of the current checkerboard cell

// subseq: the partial subsequence generated so far

if cell (x, y) is not within subproblem boundaries

    terminate recursion

if *subseq* is *good*

    store *subseq* in candidate set of the subproblem

Let P' be the last pattern in *subseq*

$\mathcal{T} \leftarrow \{P : P' \prec P$ (see Definition 6.2) AND

$((x \leq P.slotR \leq x + \Delta$ AND $P.slotL = y)$ OR

$(y \leq P.slotL \leq y + \Delta$ AND $P.slotR = x))$ AND

capacity of $(P.slotR, P.slotL)$ not fully used AND

P has no conflict with $subseq\}$

for each pattern $P \in \mathcal{T}$ do

    randomly determine whether to *accept* or *reject* P

    if P is *accepted*

        GENERATE-SUBSEQ$(P.slotR, P.slotL, subseq \cup \{P\})$

GENERATE-SUBSEQ$(x + 1, y + 1, subseq)$

Figure 6.9: Algorithm to generate a set of random subsequences

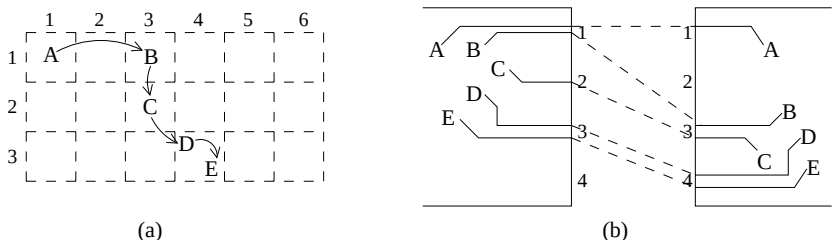


Figure 6.10: (a) A subsequence on the checkerboard, and (b) the corresponding escape patterns.

corresponding to each column.[3] If the partial subsequence *subseq* is to be stored, a previously stored subsequence with less weight may need to be replaced. Note here that our purpose is to generate a large variety of *good* subsequences for the given subproblem, instead of generating only the *best* ones. The variety among subsequences is obtained by making sure that a subsequence ending at a particular checkerboard column does not replace another subsequence ending at a different column.

The next step of the recursive algorithm is to find the set of patterns $\mathcal{T}$ that can be added to the partial subsequence *subseq*. Here, this selection is done based on the invariant that *subseq* remains *permissible* (Definition 6.4) and capacity constrained (Definition 6.5). In one recursive iteration, we consider the patterns that are (1) on cell (x, y), (2) on column x, and (3) on row y of the checkerboard. To limit the search space, we only consider patterns that are within Δ-neighbourhood of (x, y), where Δ is an input parameter, typically set to a value less than five. Figure 6.10 illustrates the physical meaning of selecting patterns from the same cell, row, or column of the checkerboard.

After finding the candidate pattern set $\mathcal{T}$, we consider each P in $\mathcal{T}$, and randomly decide whether to accept or reject P. Here, the probability of *accepting* pattern P is set so that the expected number of escape patterns that can be selected from set $\mathcal{T}$ is equal to a fixed input parameter.[4] In other words, this probability is inversely proportional to the number of candidate patterns in $\mathcal{T}$. If P is *accepted*, then another recursive call is made starting from the current checkerboard cell. After all patterns in

[3] In our experiments, the maximum number of subsequences that can be stored corresponding to each column is set to 50.

[4] We have set the expected number of patterns that can be selected at each recursive iteration to 7 in our experiments. The execution time of the subsequence generation phase can be controlled by this parameter.

$\mathcal{T}$ are considered, a recursive call to cell $(x+1, y+1)$ is made to continue subsequence generation without selecting any pattern from the current level. the main purpose here is to have a good variety in the generated subsequences.

For the following complexity analysis, we assume that parameter V given in Properties 6.1 and 6.2, and all the slot capacities are constants (i.e., have complexity $O(1)$).

Lemma 6.2 *Let $\mathcal{R}$ be the recursion tree of the function GENERATE-SUBSEQ given in Figure 6.9. The following two properties hold for $\mathcal{R}$: (1) The maximum depth of $\mathcal{R}$ is $O(1)$, and (2) the number of recursive calls made from a node in $\mathcal{R}$ is $O(1)$ on the average.*

PROOF. At each recursive call, either a pattern P is added to the partial subsequence, or the x and y coordinates are both incremented by 1. Since each subproblem consists of V rows, and escape slot capacities are constants, the maximum length of any subsequence is $O(1)$. Hence, the maximum recursion depth is $O(1)$. Furthermore, we randomly decide whether to accept or reject pattern P such that the expected number of patterns selected in each iteration is constant. As a result, the number of recursive calls made from a node in $\mathcal{R}$ is $O(1)$ on the average. ■

Lemma 6.3 *The recursive function GENERATE-SUBSEQ(x,y,subseq) is invoked only a constant number of times for each checkerboard cell (x, y).*

PROOF. Our proof is based on induction on the depth of the recursive tree $\mathcal{R}$. Obviously, the checkerboard cell at the root of $\mathcal{R}$ is called only a constant number of times (base case). Let us consider a grid cell (x, y), and let us assume that the induction hypothesis holds for all cells called before (x, y). From the algorithm of Figure 6.9, we know that only the cells that are in the Δ-neighbourhood of (x, y) can make a call to (x, y). Since Δ is constant, the lemma follows due to the induction hypothesis. ■

Theorem 6.3 *The total average-time complexity of subsequence generation for all subproblems is $O(n + s^2)$, where n is the number of nets, and s is the number of escape slots on the component boundaries.*

PROOF. We will first prove that the average-time complexity for subproblem i is $O(n_i + s)$, where n_i is the number of patterns mapped to a cell within subproblem

i. In one recursive call, all patterns P mapped to cells in the Δ-neighborhood of cell (x, y) are processed to determine set $\mathcal{T}$. Since Δ is constant, and due to Lemma 6.3, each pattern is processed only a constant number of times. Furthermore, the average number of nodes in a recursion tree $\mathcal{R}$ is $O(1)$, due to Lemma 6.2. Since there are s separate recursion trees (each root corresponding to a cell on the first row of the current subproblem), the average-time complexity for one subproblem is $O(n_i + s)$. Based on this, the total average-time complexity for all subproblems can be written as $\sum_{1 \leq i \leq s/V} O(n_i + s) = O(n + s^2)$. ■

Theorem 6.4 *Let K denote the maximum number of subsequences that can be stored for each subproblem. The average time complexity for the proposed randomized planar route selection algorithm is $O(n + s^2 + K^2 s)$, where n is the number of nets, and s is the number of escape slots on component boundaries.*

PROOF. In graph $\mathcal{G}_R$ (defined in the beginning of this section), there is a vertex corresponding to each subsequence generated. Since there are $s/V = O(s)$ subproblems, the number of vertices in $\mathcal{G}_R$ is $O(Ks)$. The edges in $\mathcal{G}_R$ are only between vertices that correspond to adjacent subproblems. Hence, the number of edges in $\mathcal{G}_R$ is $O(K^2 s)$. Since, $\mathcal{G}_R$ is acyclic, computing the longest path has linear time complexity in the graph size [16], which is $O(K^2 s)$. As given in Theorem 6.3, the average time complexity of subsequence generation is $O(n + s^2)$; so the proof is complete. ■

6.5 Handling High-Speed Design Constraints

In the following subsections, we discuss how to generalize the algorithms given in Sections 6.3 and 6.4 to handle different high speed design constraints.

6.5.1 Maximum Length Constraints

Board designers specify maximum length constraints for critical nets to limit the maximum arrival times. We can handle these constraints during the pattern generation phase of our framework. Specifically, we can restrict the set of target escape slots (parameter $\mathcal{T}$ in Figure 6.4) such that the escape segments with long detours are avoided. Furthermore, remember that an escape pattern is created by merging two escape segments from the left and right components. It is possible to check the

maximum length constraints during this step, and eliminate the patterns that violate the corresponding constraints.

6.5.2 Minimum Length Constraints

Minimum length constraints are typically enforced for nets belonging to a bus structure, with the objective of matching the signal arrival times. Typically, the length of a *short* net needs to be extended to satisfy its min-length constraint. Since the routing resources within components are extremely limited, it makes more sense to perform length extension in the intermediate area between components, in a later stage of the routing system. However, we can also modify our randomized planar route selection algorithm (Section 6.4) such that the patterns that satisfy min bounds are preferred over the others. For this purpose, we can assign a weight to each escape pattern, based on its length and the corresponding min length constraint. Then, the randomized algorithm given in Section 6.4 can be used to select the permissible pattern sequence with the largest weight.

6.5.3 Adjacency Constraints for Noise Avoidance

Adjacency constraints between different nets are defined by designers to avoid crosstalk problems. A typical adjacency constraint between nets n_i and n_j can be stated as follows [40]: *If n_i and n_j are routed adjacent to each other on the same layer, then their routes need to be separated by at least k routing tracks.* Such a constraint is enforced typically on signal nets that belong to different bus structures. In the context of the model defined in Section 6.4.1, we can restate this constraint as follows: *If the patterns corresponding to n_i and n_j are adjacent in a permissible pattern sequence $\mathcal{S}$, then the escape slots of these patterns need to be separated by at least k routing tracks.* This constraint can be handled effectively in the subsequence generation algorithm given in Figure 6.9 by comparing the last pattern in the partial subsequence *subseq* with the candidate pattern P. Specifically, the following line needs to be added immediately after set $\mathcal{T}$ is defined in Figure 6.9:

$$\mathcal{T} \leftarrow \mathcal{T} \cap \{P : \text{if } (P', P) \text{ has a } k\text{-adjacency constraint, then there are } k \text{ empty tracks between } P' \text{ and } P \text{ in both left and right components}$$

By adding this line, we make sure that only the subsequences that do not violate adjacency constraints are generated. In addition, we also need to check these constraints for subsequences in neighbouring subproblems. Specifically, we need to add the following rule while defining the edges of $\mathcal{G}_R$ in Figure 6.6:

Let v_j^i and v_k^{i+1} denote two vertices in $\mathcal{G}_R$ corresponding to subsequences S_j^i and S_k^{i+1}, which have been generated in subproblems i and $i+1$, respectively. Let P_j^i denote the last pattern in subsequence s_j^i, and P_k^{i+1} denote the first pattern in subsequence S_k^{i+1}. If (P_j^i, P_j^{i+1}) have an adjacency constraint of at least k tracks, then an edge from v_j^i to v_k^{i+1} (in $\mathcal{G}_R$) exists only if there are at least k tracks between P_j^i and P_k^{i+1} in both components.

These two modifications are sufficient to ensure that the output of our algorithms satisfy all adjacency constraints.

6.5.4 Differential Pairs

A differential pair is a complementary pair of nets that provide noise immunity. The two nets within a differential pair need to be routed parallel to each other, separated by a specific distance as long as possible. Let us consider two nets n_i and n_j that belong to a differential pair. During pattern generation, we can identify the pairs of escape segments corresponding to n_i and n_j that adhere to these constraints. In the context of the model defined in Section 6.4.1, a pattern corresponding to n_i can exist in a permissible sequence $\mathcal{S}$ only if it is *adjacent* to an acceptable segment of n_j. This constraint can be explicitly checked in the subsequence generation algorithm of Figure 6.9 by comparing the last pattern in the partial subsequence *subseq* with the candidate pattern P. Specifically, the following code segment needs to be added immediately after set $\mathcal{T}$ is defined in the algorithm of Figure 6.9:

```
Let P'' be the second-to-last pattern in subseq
if P' belongs to a differential pair AND
    (P'', P') is not a differential pair then
        T ← T ∩ {P : (P', P) is a differential-pair}
```

By adding these lines, we make sure that patterns belonging to a differential pair always occur together in a subsequence. However, we also need to check differential pairs that are in two adjacent subproblems. For this purpose, we need to add the following rule while defining the edges of $\mathcal{G}_R$ in Figure 6.6:

Table 6.1: Comparison of the proposed framework with the basic algorithm given in Chapter 5

			PROPOSED FRAMEWORK		BASIC ALGORITHM	
Input	# layers	# nets	nonplanar nets	time (m:s)	nonplanar nets	time (m:s)
IBM1	5	426	25	1:56	50	0:37
IBM2	5	428	35	0:58	44	0:30
IBM3	4	352	22	1:04	48	0:33
IBM4	5	312	47	1:22	68	0:54
IBM5	3	226	6	0:25	18	0:18
IBM6	5	441	35	1:01	50	0:32

Let P^i_{j-1} and P^i_j denote the second-to-last and last patterns in subsequence s^i_j; let P^{i+1}_k denote the first pattern in subsequence S^{i+1}_k. Assume that P^i_j is part of a differential pair, and (P^i_{j-1}, P^i_j) is not a differential pair. If this is the case, then an edge from v^i_j to v^{i+1}_k (in $\mathcal{G}_R$) exists only if (P^i_j, P^{i+1}_k) is a differential pair.

These two modifications are sufficient to handle the differential pair constraints.

6.6 Experimental Results

We have performed experiments on escape problems extracted from a real industrial board design, for which current industrial tools fail to produce a routing solution. We have implemented our algorithms in C++, and performed the experiments on an Intel Pentium 4 2.4GHz system with 1GB memory, and a Linux operating system. The input parameter V given in Properties 6.1 and 6.2 is set to 4 in our experiments.

In Section 5.6, our experiments have illustrated that the randomized algorithm of Chapter 5 outperforms the net-by-net methodology. Here, we use that randomized algorithm for comparison with the algorithm we propose in this chapter. Table 6.1 gives the results obtained on industrial test cases. As mentioned before, layers are processed one by one, and the maximal planar routing solution is found for each layer. The number of nets that could not be routed in a planar fashion is given for each problem under columns *nonplanar nets*. These nets will be distributed to available layers later, allowing crossings in the intermediate channel. As discussed before, a *crossing* net will need to use a via during the later stages of the routing system. The results in Table 6.1 indicate that our algorithm reduces the via requirements on the average by 39%, for the given industrial test cases. A sample output of our maximal

planar routing algorithm for one layer is illustrated in Figure 6.11.

6.7 Conclusions

We have proposed an algorithm to solve the escape routing problem in multiple components simultaneously. Compared to the algorithm proposed in Chapter 5, the main improvements can be summarized as follows. First, we propose a more intelligent pattern generation algorithm based on congestion levels in the components and the number of crossings in the intermediate area. Then, we propose a more sophisticated randomized algorithm for the maximal planar routing problem. We also show how to handle typical high speed design constraints within the framework of this algorithm. Our experiments show that our algorithm can reduce the via requirements significantly.

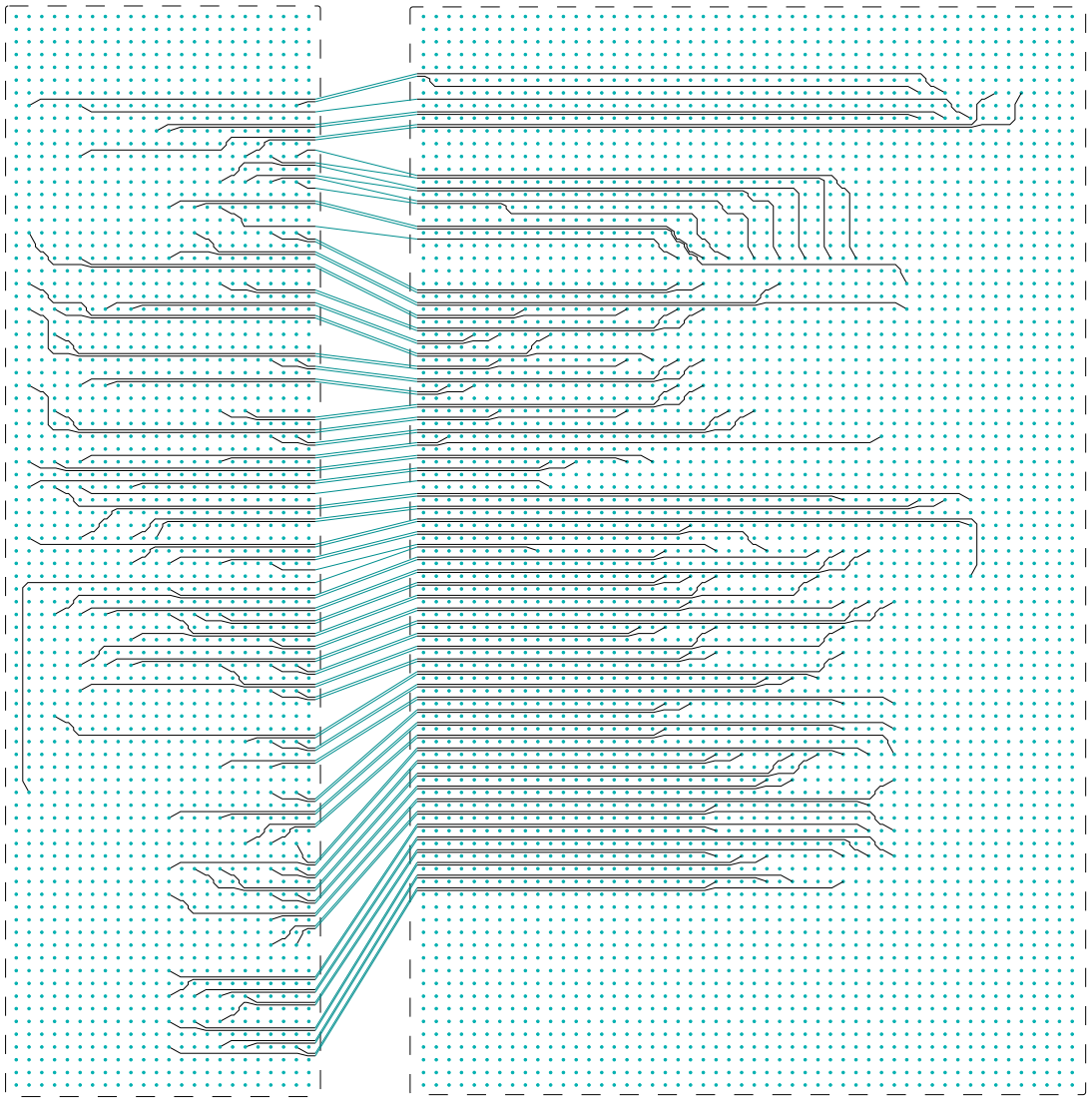


Figure 6.11: A planar escape routing solution is illustrated for two components. 111 nets have been routed on this layer. The connections in the intermediate area are shown as straight lines between components.

Chapter 7

Optimal Routing Algorithms for Pin Clusters in High-Density Packages

7.1 Introduction

One of the most difficult parts of the package routing problem is routing within dense pin clusters [33]. Both packaging hierarchy and functional hierarchy imply potential pin clustering at hierarchical interfaces. These clusters are formed typically by pins that belong to the same functional unit or the same data bus. The highest wire demand is typically within such pin clusters and in proximity of the cluster perimeters. As additional objectives (such as delay and noise optimizations) impose further constraints on the routing problem, getting the connections started correctly from the clustered pin areas is becoming an increasingly important issue.

A cluster of pins from a real MCM design (from IBM) is illustrated in Figure 7.1. As seen in this figure, these pin clusters typically have irregular shapes. The empty areas in these clusters can be due to islands of voltage pins, which are routed in dedicated voltage layers. They can also be due to blind or buried vias that do not span all layers of the package. In surface mount type (SMT) components, such as ball grid arrays (BGAs), I/O signals are typically transferred to inner component layers using blind or buried vias [8; 60]. These vias are used to carry the I/O signals from bare chips (on the top layer) to the layers on which the corresponding nets are routed. In other words, once a net is routed on one layer, its pin is not extended

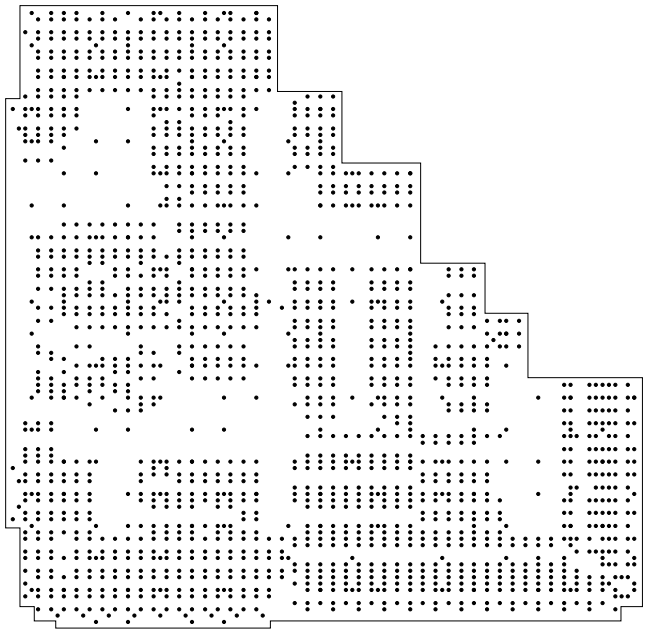


Figure 7.1: A cluster of pins from a chip mounted on a ceramic MCM module from a real IBM design. The convex boundary enclosing the pin cluster is also illustrated.

further down the layer stack. As a result, the cluster of pins typically *shrinks* as we go further down the layer stack. Due to all these factors, the pin clusters often have irregular shapes, as shown in Figure 7.1.

The escape routing problem has been studied extensively in the literature [9; 16; 29; 57; 64] to route nets from individual pins to a boundary. However, a rectangular boundary is assumed in these algorithms most of the time, and the effects of irregular boundaries are not considered. Normally, a traditional escape routing algorithm can also be applied on a pin cluster with an irregular boundary. However, the problem here is that routability outside the boundary is not guaranteed, since the routes of nets that escape to this boundary may conflict with each other outside. Figure 7.2 gives a small example to illustrate this problem in more detail. Assume that a set of nets have been routed to a set of escape terminals on the boundary, as shown in Figure 7.2(a). Here, one problem is how to determine whether all nets that have escaped to the boundary can be successfully routed outside, since some of the escaped nets can conflict with each other. In this example, there are 14 nets that have successfully escaped to the boundary; however only 9 of them can be successfully routed outside, as shown in Figure 7.2(b). In Section 7.3, we propose a set of necessary and sufficient conditions to determine routability based on only the positions of escape terminals on an arbitrary

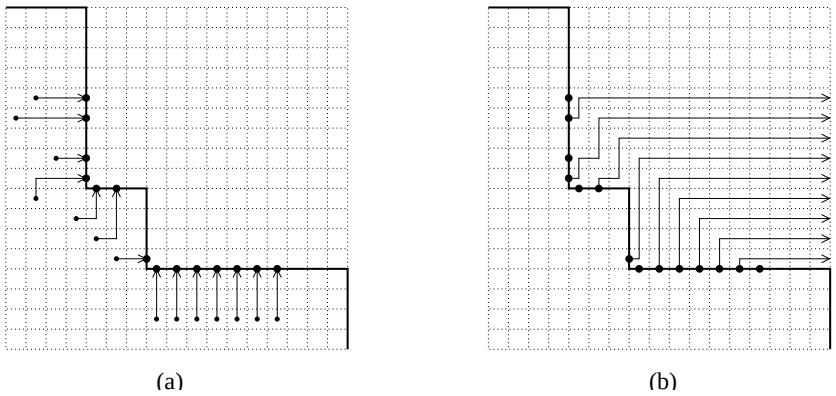


Figure 7.2: (a) An escape routing solution for 14 nets from pins to a convex boundary. (b) Only 9 out of 14 escaped nets can be routed outside due to conflicts with each other.

convex boundary. Another problem here is how to determine the maximal routable subset if a given set of escape terminals is not routable. A maximal routable subset is shown in Figure 7.2(b), together with a feasible routing solution, corresponding to the escape terminals in Figure 7.2(a). For this purpose, we propose an optimal algorithm in Section 7.4. In this algorithm, the optimal subset is determined based on only the positions of the escape terminals, without performing any routing outside the boundary. After that, we focus on an integrated approach in Section 7.5 to consider *routability outside* during the actual escape routing algorithm. In other words, instead of using a two-step methodology (escape routing followed by routability analysis), we directly find the escape routing solution such that routability outside is also guaranteed. For example, Figure 7.3(a) shows a different escape routing solution for the problem in Figure 7.2. Here, all the nets that have escaped are routable, as shown in Figure 7.3(b). The proposed algorithm for this purpose is also proven to be optimal.

The rest of this chapter is organized as follows. We give a formal description of this problem in Section 7.2. Then in Section 7.3, we propose a set of necessary and sufficient conditions that exactly model routability outside the given convex boundary. Based on these constraints, we propose an optimal algorithm in Section 7.4 to select the maximal subset of routable escape terminals. After that, we propose an integrated approach in Section 7.5 that incorporates the routability constraints into the original escape routing algorithm in an optimal way. In section 7.6, we present our experimental results, and demonstrate the effectiveness of our algorithms.

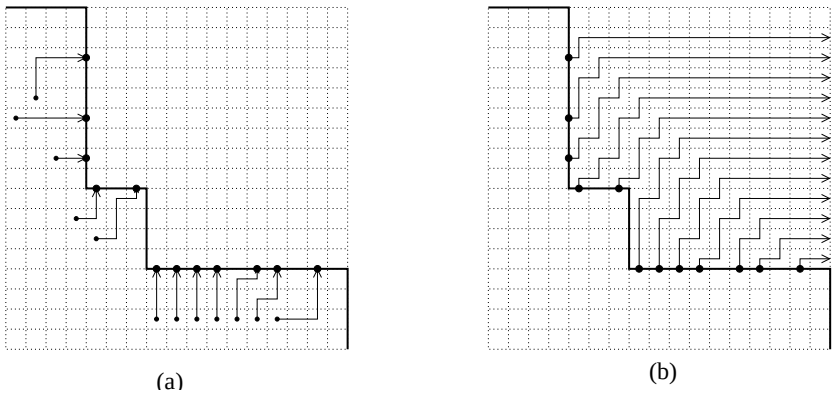


Figure 7.3: (a) A different escape routing solution for the problem of Figure 7.2. (b) All 12 nets are routable outside the boundary.

7.2 Problem Formulation

Let $\mathcal{P}$ denote a cluster of pins, and let B denote the rectilinear convex boundary enclosing $\mathcal{P}$. Our purpose is to find a routing solution from each pin in $\mathcal{P}$ to an *escape terminal* on B. Here, the scarcity of routing resources inside dense pin clusters do not allow usage of additional buried vias. Hence, the escape routing solution needs to be planar on every layer.

As illustrated in Figure 7.2, the nets escaping to an irregular boundary can have conflicts with each other outside. A given escape routing solution is defined to be *routable outside* iff all nets escaping to boundary B can be routed without conflicts, as illustrated in Figure 7.3(b). Here, let us assign a unique index to every escape terminal on boundary B, as shown in Figure 7.4. Furthermore, let $\#t(x, y)$ denote the number of nets escaping to escape terminals in the interval $[x, y]$, e.g. $\#t(8, 13) = 5$ in part (a), and $\#t(8, 13) = 3$ in part (b) of Figure 7.4. Here, the first problem we focus on is how to determine whether a given escape routing solution is also *routable outside*, using these $\#t(x, y)$ values. If the given solution is not routable outside, the next problem becomes how to select the maximal subset of routable escape terminals. Finally, the third problem is how to find an escape routing solution in an optimal way such that overall routability is guaranteed. We study these problems in this chapter, and propose models and algorithms to solve each of them optimally.

For simplicity of presentation, we will consider only a single layer. In other words, our objective will be to find the maximal escape routing solution on one layer, given a set of candidate pins. It is possible to process layer by layer, and apply our algorithms

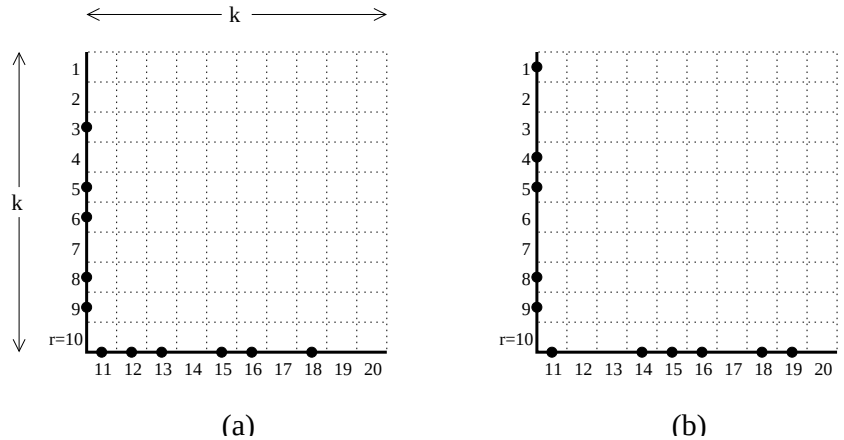


Figure 7.4: A boundary with a single corner is illustrated, where filled circles represent the escape terminals at which an escape route ends (the escape routes inside are not shown for clarity). Two examples with different terminals are given in (a) and (b).

on every layer. However, our algorithms can also be extended to multilayer problems in a straightforward way, by duplicating the given constraints for every layer.

7.3 Constraint Modeling

Our purpose in this section is to investigate the relationship between escape terminal positions on a given convex boundary and the overall routability. For this, we define a set of necessary and sufficient conditions that exactly model *routability outside* the boundary. For simplicity of presentation, we will first focus on a single corner of a given convex boundary in Section 7.3.1 and then generalize this model for an arbitrary convex boundary in Section 7.3.2. This constraint modeling will be especially useful since it can be incorporated into the original escape routing algorithm in such a way to guarantee routability outside the boundary.

7.3.1 Corner Constraints

In this section, we will consider a boundary with a single corner, as shown in Figure 7.4. Here, let r and $r+1$ denote the escape terminals on the corner, and let

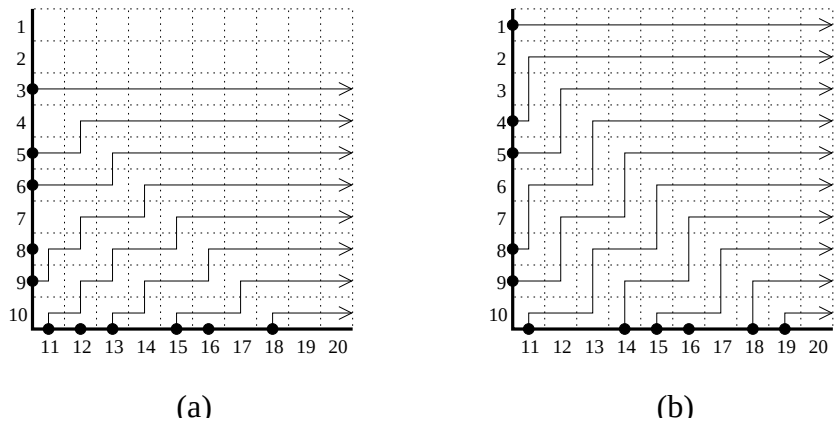


Figure 7.5: Routing solutions outside the boundaries for the problem given in Figure 7.4. Three and one nets are unroutable in the solutions of parts (a) and (b), respectively.

k denote the width of one side of the corner.[1] Furthermore, let $\#t(x, y)$ denote the number of nets that have escaped to terminals in the interval $[x, y]$. Observe in the example of Figure 7.4 that $r = 10$, and $k = 10$. Also, $\#t(9, 12) = 3$ in part (a), and $\#t(9, 12) = 2$ in part (b), etc. The following theorem defines the necessary and sufficient conditions for routability:

Theorem 7.1 *An escape routing solution is routable if and only if* $\#t(r\text{-}i\text{+}1, r\text{+}i) \leq i$, *for each* i, $1 \leq i \leq k$. *In other words, routability is guaranteed if and only if the number of nets escaping to terminals in the interval* $[r\text{-}i\text{+}1, r\text{+}i]$ *is less than or equal to* i, *for each* i.

As an example, let us consider the boundary given in Figure 7.4, where the escape terminals are marked from 1 to 20. Here, the following conditions are necessary and sufficient for routability: $\#t(10, 11) \leq 1$, $\#t(9, 12) \leq 2$, ..., $\#t(1, 20) \leq 10$. The given escape routing solution in part (a) violates the conditions $\#t(9, 12) \leq 2$, $\#t(8, 13) \leq 3$, and $\#t(5, 16) \leq 6$; hence, 3 out of 11 nets are unroutable, as illustrated in Figure 7.5(a). Similarly, the solution in part (b) violates the condition $\#t(5, 16) \leq 6$, resulting in one unroutable net.

PROOF. NECESSITY: We first prove that the constraints given in Theorem 7.1 are necessary for routability outside. For any i value, let D_i denote the diagonal line

[1]For simplicity, assume that the widths of both sides are equal as shown in Figure 7.4. The generalization will be given in Section 7.3.2.

spanning the grid cells that have Manhattan distance of i to the corner, as shown in Figure 7.6(a). It is obvious that the number of *outlets* (i.e., grid cells through which nets can escape) on D_i is equal to i. Since a net escaping to a terminal in the interval $[r-i+1, r+i]$ must use an outlet on D_i, the necessity of constraint $\#t(r-i+1, r+i) \leq i$ follows. ■

PROOF. SUFFICIENCY: Let us make the following inductive hypothesis: *If the constraints $\#t(r-i+1, r+i) \leq i$ are satisfied for each i, $1 \leq i \leq k$, then all nets escaping to terminals in the interval $[r-k+1, r+k]$ can escape to diagonal D_k.* Again, D_k denotes the diagonal line spanning the grid cells that have Manhattan distance of k to the corner, as shown in Figure 7.6(a). It is straightforward to show that this hypothesis holds for the base case $k = 1$. Now let us assume that it holds for $k = m$, and we will prove it for $k = m + 1$. For this, we need to consider two cases:

- Case 1: $\#t(r-m+1, r+m) < m$. Here, since there are less than m outlets used on diagonal D_m, there is at least one outlet unused (shown as a hollow circle in Figure 7.6(b)). Even if there are two nets escaping to terminals $r-m$ and $r+m+1$, all nets will still be routable to diagonal D_{m+1}, as shown in Figure 7.6(b).

- Case 2: $\#t(r-m+1, r+m) = m$. For the constraint $\#t(r-m, r+m+1) \leq m+1$ be satisfied, only one net can escape to terminals $r-m$ and $r+m+1$. As shown in Figure 7.6(c), and (d), all nets will still be routable to diagonal D_{m+1}, as long as only one of the terminals $r-m$ or $r+m+1$ is selected.

So, the inductive proof is complete. ■

7.3.2 Generalization to Arbitrary Convex Boundaries

In this section, the idea presented in Section 7.3.1 is generalized to arbitrary convex boundaries. For a rectilinear convex boundary, we can make the following definition:

Definition 7.1 *A rectilinear convex boundary is defined to have four different regions: falling-right, falling-left, rising-left, and rising-right regions, as illustrated in Figure 7.7.*

It is obvious that nets escaping to one boundary region (e.g. falling-right region) do not interfere with nets escaping to other regions outside the boundary. In other

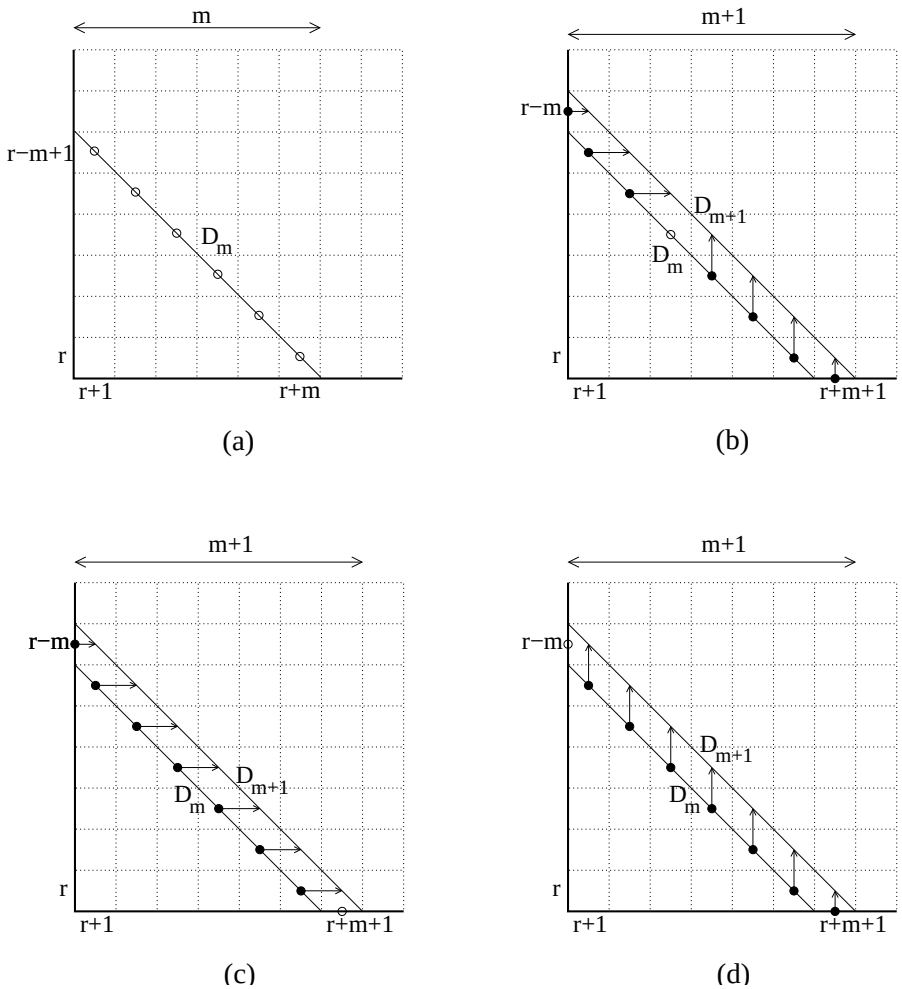


Figure 7.6: (a) Diagonal D_m has m escape outlets (shown as hollow circles). (b) If there is an unused outlet on D_m, all nets are routable to D_{m+1}, even if both escape terminals $r - m$ and $r + m + 1$ are selected. (c,d) If all outlets on D_m are occupied, then routability is guaranteed as long as at most one of terminals $r - m$ and $r + m + 1$ is selected.

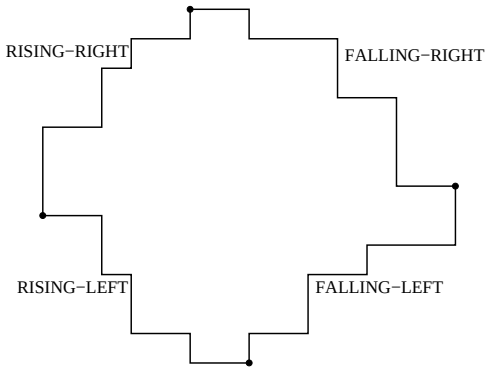


Figure 7.7: Different boundary regions of a rectilinear convex boundary are illustrated.

words, we can consider each of falling-right, falling-left, rising-left, and rising-right regions independent of each other while determining routability outside the boundary. So, in the rest of this section, we will propose the necessary and sufficient conditions for only a falling-right boundary region. It is straightforward to generalize these conditions for other region types.

For the ease of presentation, we will define the routability conditions using the algorithm given in Figure 7.10. This algorithm is based on boundary transformations that are defined in Definitions 7.2 and 7.3. It is important here to note that these are only conceptual transformations used for the purpose of presentation. In other words, these transformations are not actually performed (i.e., the original boundary still remains intact); however, they are used conceptually to generate the set of necessary and sufficient conditions for routability. In the following, let H-segment, V-segment, and D-segment denote horizontal, vertical, and diagonal boundary segments, respectively.

Definition 7.2 *Consider a corner of a falling-right boundary where a V-segment is followed by an H-segment. We can (conceptually) transform this corner as shown in Figure 7.8(a), and add the explicit constraint* $\#t(r, r+1) \leq 1$*, where the escape terminals on the corner are denoted as* r *and* $r+1$*. Intuitively, replacing a corner with a diagonal as in this figure implies that we do not have to consider this corner anymore for routability analysis, as long as the constraint* $\#t(r, r+1) \leq 1$ *is satisfied.*

Definition 7.3 *Consider a falling-right boundary that contains a V-segment, followed by a D-segment, followed by an H-segment. We can (conceptually) transform*

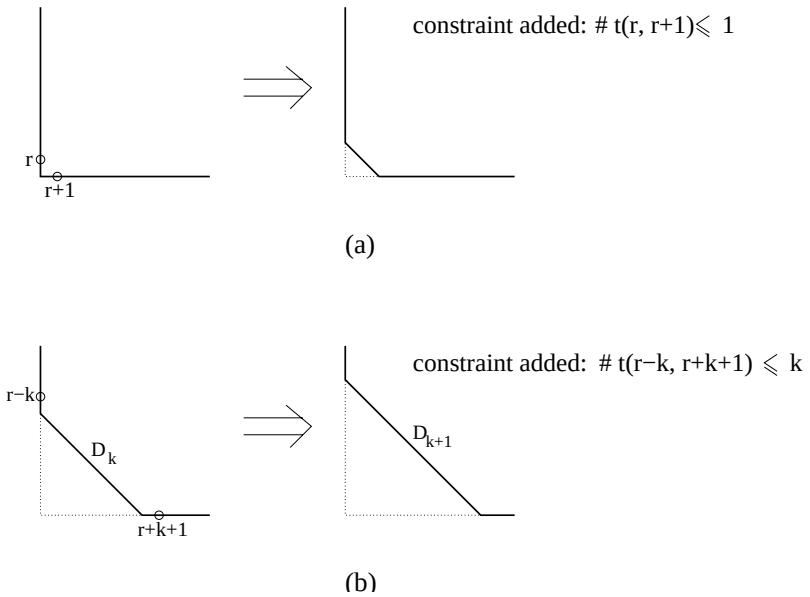


Figure 7.8: Illustration of the boundary transformations given in (a) Definition 7.2, and (b) Definition 7.3. The corresponding constraints generated are also shown.

this boundary as shown in Figure 7.8(b), and add the explicit constraint $\#t(r-k, r+k+1) \leq k+1$*, where* r *and* k *are as defined in this figure.*

Lemma 7.1 *Let* B *denote the original escape boundary, and let* B' *denote the boundary after one of the transformations given in Definitions 7.2 and 7.3 is applied on* B*. The routability characteristics of* B *is equivalent to the routability characteristics of* B' *iff the additional constraint introduced during the transformation is satisfied.*

PROOF. The proof is very similar to the inductive proof of Theorem 7.1. ■

Intuitively, we can continue performing boundary transformations, and defining new conditions, until the transformed boundary is guaranteed to be routable. The following lemma states the routability condition for a falling-right boundary.

Lemma 7.2 *A falling-right boundary* B *is guaranteed to be routable outside if there is no H-segment after a V-segment in* B*.*

PROOF. Figure 7.9 shows the main intuition. In part (a), there is no H-segment after a V-segment, and all nets escaping to all terminals on the boundary are routable. On the other hand, there is an H-segment after a V-segment in part (b), and routing conflicts are possible outside the boundary. ■

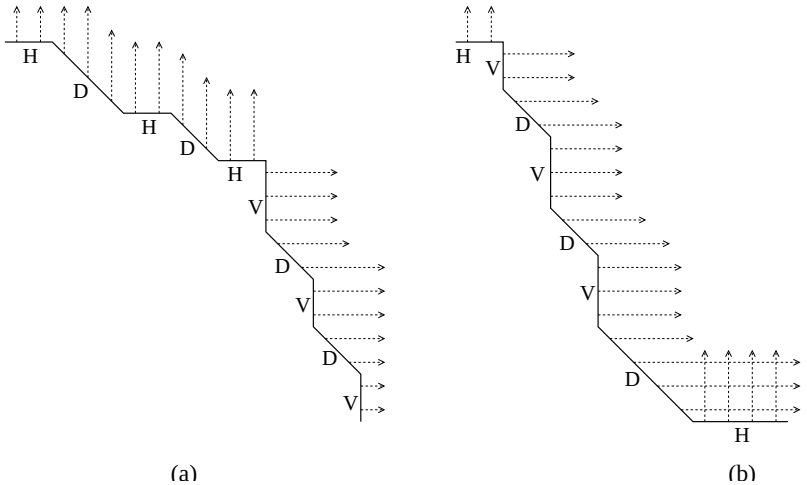


Figure 7.9: Illustration of routing conflicts outside a falling-right boundary. (a) There is no H-segment after a V-segment; hence conflict-free routing is possible outside. (b) The H-segment after V-segment causes routing conflicts.

CREATE-CONSTRAINT-FOREST(falling-right boundary B)

$\mathcal{C} \leftarrow \emptyset$ // the set of necessary and sufficient conditions

while there is no H-segment after a V-segment in B

perform a (conceptual) boundary transformation

add the corresponding constraint into $\mathcal{C}$.

create the constraint forest $\mathcal{F}$ as follows:

for each constraint in $\mathcal{C}$, a node exists in $\mathcal{F}$.

Node u is a parent of node v in $\mathcal{F}$ iff u has the smallest interval that is a proper superset of v's interval.

return $\mathcal{F}$

Figure 7.10: Algorithm to generate the set of necessary and sufficient conditions for a given falling-right boundary.

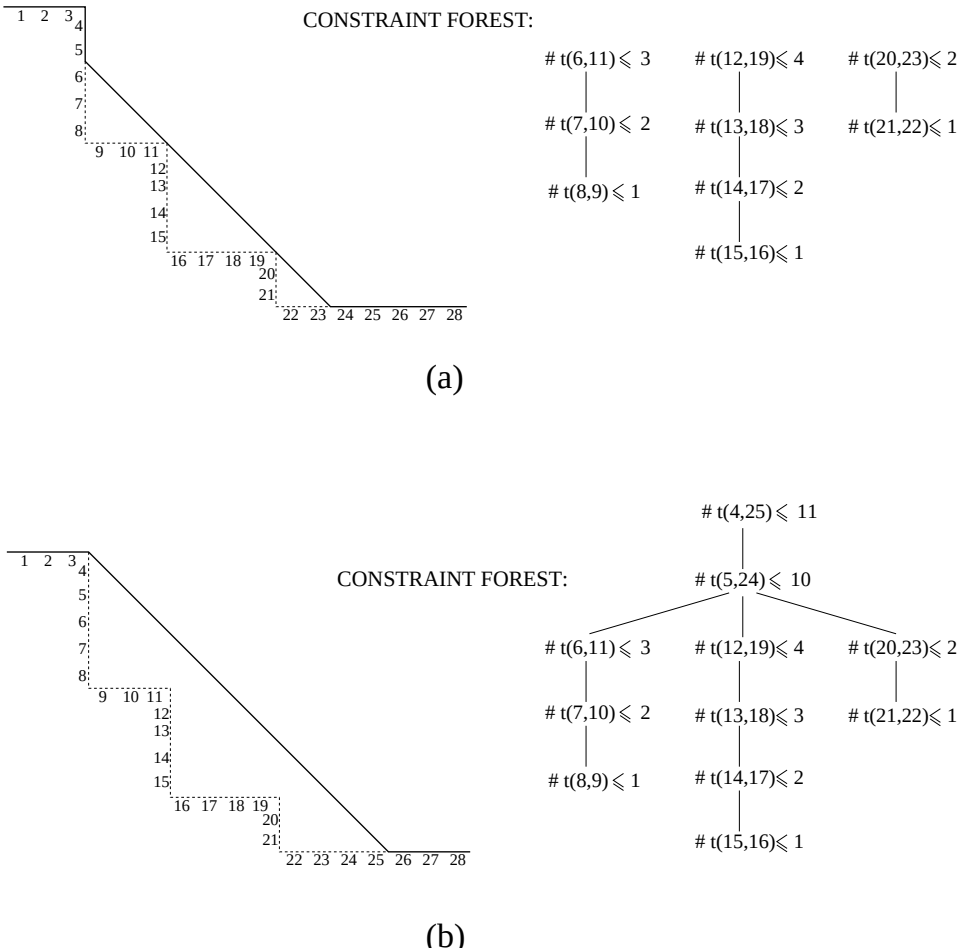


Figure 7.11: Illustration of constraint forest generation on a convex boundary with 28 escape terminals. The original boundary is shown with dotted lines. (a) The boundary after the first set of transformations, and the corresponding partial forest. (b) The final boundary, and the constraint forest generated.

Figure 7.10 gives the algorithm we use to generate the set of necessary and sufficient conditions corresponding to a given falling-right boundary. This set of conditions is represented as a *constraint-forest* $\mathcal{F}$, where each node in $\mathcal{F}$ corresponds to a constraint in the form $\#t(x, y) \leq z$; i.e., the number of nets escaping to terminals in the interval $[x, y]$ is less than or equal to z. Here, if node u is a parent of node v, then the constraint interval corresponding to node u is guaranteed to be a proper superset of the constraint interval corresponding to node v. Figure 7.11 illustrates the constraint forest generation process with an example.

7.4 Selection of Maximal Routable Escape Terminals

In this section, we assume that escape routing to an arbitrary convex boundary has already been performed, and our purpose is to select the maximum subset of terminals that can be routed outside without any conflicts. For this, we make use of constraint forest $\mathcal{F}$, which was defined in Section 7.3. Before giving the details of this algorithm, we will make some observations about the properties of $\mathcal{F}$ as follows.

Remark 7.1 *Consider two nodes u and v in constraint forest $\mathcal{F}$. If u is an ancestor of v, then the interval corresponding to u is a proper superset of the interval corresponding to v.*

Remark 7.2 *Consider two nodes u and v in constraint forest $\mathcal{F}$. If u is neither ancestor nor descendant of v, then the intervals corresponding to u and v do not overlap.*

Remark 7.3 *Consider a non-leaf node u that has the constraint $\#t(x, y) \leq z$, The union of the constraints corresponding to all children of node u is equivalent to $\#t(x+1, y-1) \leq z-1$.*

Remark 7.4 *The number of nodes in constraint forest $\mathcal{F}$ is linear in the number of escape terminals on the boundary.*

These observations directly follow from the definition of the constraint forest. Readers can refer to Figure 7.11 for an example.

The algorithm we propose for selection of maximal routable escape terminals is given in Figure 7.12. The recursive function given in this figure needs to be called for each root node in the constraint forest $\mathcal{F}$. Intuitively, we first process the children of the current node r, and find the maximal set of escape terminals that satisfy the descendant constraints. Then, we consider the constraint at r, which is $\#t(x, y) \leq z$. From Remark 7.3, we know that the escape terminals in the interval $[x+1, y-1]$ have already been processed by the descendants of node r. So, we consider only the escape terminals x and y here. If the size of the selected terminal set T is still less than z, then we add these terminals to T, making sure that the constraint $\#t(x, y) \leq z$ is not violated. The following theorem states the optimality and the time complexity of this algorithm.

```
SELECT-ESCAPE-TERMINALS(Node r)
T ← ∅          // the selected terminal set
for each child u of r do
    T ← T ∪ SELECT-ESCAPE-TERMINALS(u)
Let #t(x, y) ≤ z be the constraint corresponding to r
If there is an escape route ending at terminal x
    T ← T ∪ {x}
If there is an escape route ending at terminal y
    if |T| < z
        T ← T ∪ {y}
return T
```

Figure 7.12: The algorithm to select the maximal routable escape terminals. This algorithm needs to be called for each root node in the constraint forest.

Theorem 7.2 *The algorithm proposed in Figure 7.12 returns the maximal subset of escape terminals that are routable outside the boundary. The time complexity of this algorithm is linear in the number of escape terminals on the boundary.*

PROOF. The time complexity of the algorithm directly follows from Remark 7.4, since each node in the forest is processed only once. For optimality, let us first consider Remark 7.2, which indicates that different subtrees rooted at r specify constraints for non-overlapping intervals. In other words, terminal selection in each subtree can be performed independent of each other. Now, we will prove the optimality of this algorithm using induction. As the base case, let us consider a forest consisting of only leaf nodes. It is obvious that our algorithm will give the optimal solution, since each leaf is independent of each other (due to Remark 7.2). Now, assume that the inductive hypothesis holds for each child subtree of node r; i.e., each recursive call to a child of r returns the optimal solution. Let us denote the constraint corresponding to node r as $\#t(x, y) \leq z$. We know that the intervals corresponding to different subtrees do not overlap (due to Remark 7.2) and that the union of the intervals considered in r's child subtrees is $[x+1, y-1]$ (due to Remark 7.3). From the inductive hypothesis, we can state that after the recursive calls, T contains the maximal routable set of escape terminals in the interval $[x + 1, y - 1]$. So, while processing node r, we only need

to consider whether we should add escape terminals x and y into T. Note that the maximum number of escape terminals that can be selected in the interval $[x, y]$ is z due to the constraint at node r. Now, let us consider two cases: (1) $\#t(x+1, y-1) < z-1$, and (2) $\#t(x+1, y-1) = z-1$. In the first case, both x and y can be added to T, if there are escape routes ending at these terminals; hence, the optimal solution in the interval $[x, y]$ is obtained. In the second case, we need to make sure that the number of selected terminals does not exceed z before selecting terminals x or y. However, we know that the maximum size of T can be z in any routable solution; hence the optimal solution in the interval $[x, y]$ is still maintained. So, our inductive proof is complete. ■

7.5 Routability-Driven Escape Routing

In the previous sections, we have assumed that the escape routing solution has already been found, and we have proposed a set of constraints to determine the routability outside the boundary. In this section, we propose an integrated approach to solve the escape routing problem in such a way that routability outside is guaranteed. For this purpose, we define a flow network corresponding to the constraint forest proposed in Section 7.3, and then we augment it to the original flow network which corresponds to the escape routing problem.

It is well known that the problem of escape routing can be solved optimally using network flow [16]. In the literature, there have also been different improvements proposed for the purpose of reducing execution time and space requirements [9; 29]. Our constraint models can be applied to different flow models; however we will focus on the basic network flow formulation for simplicity of presentation.

Let us assume that flow network $\mathcal{N}$ is modeled corresponding to the original escape problem (to a convex boundary) as follows: For each grid cell, there is a vertex in $\mathcal{N}$, with node capacities equal to 1. The vertices corresponding to the neighboring grid cells are connected by edges in $\mathcal{N}$. Furthermore, there are two special vertices: the *source* and the *sink* vertices in the flow network. There is an edge from the *source* vertex to every vertex that corresponds to a grid cell on which a net terminal exists. Similarly, there is an edge to the *sink* vertex from every vertex that corresponds to a grid cell on the boundary. It is known that the maximum flow from the *source* vertex to the *sink* vertex in $\mathcal{N}$ gives the optimal solution for the escape routing problem.

Further details about this basic network flow formulation can be found in [16]. Note here that this formulation does not consider the routability constraints outside the boundary, and it is possible to obtain a routing solution that is not routable outside the convex boundary, as illustrated in Figure 7.2. For the purpose of incorporating the routability constraints, we define the following flow network $\mathcal{N}_C$:

Definition 7.4 *The flow network $\mathcal{N}_C$ corresponding to constraint forest $\mathcal{F}$ is created as follows:*

- *Create a t-vertex corresponding to each escape terminal on the convex boundary. Set the capacity of each t-vertex to 1.*
- *Create a c-vertex corresponding to each node in the constraint forest $\mathcal{F}$.*
- *Consider each c-vertex v_c, which corresponds to the constraint $\#t(x, y) \leq z$. Set the capacity of v_c to z. Then, create the incoming edges to v_c as follows:*
  - *Create an edge to v_c from t-vertex corresponding to escape terminal x.*
  - *Create an edge to v_c from t-vertex corresponding to escape terminal y.*
  - *Create edges to v_c from the c-vertices that correspond to children of v_c (in the constraint forest $\mathcal{F}$).*
- *Consider each c-vertex v_r that corresponds to a root node in the constraint forest $\mathcal{F}$. Create an edge from v_r to sink vertex of $\mathcal{F}$.*

The flow network corresponding to the constraint forest of Figure 7.11(b) is illustrated in Figure 7.13 as an example. Note that the size of $\mathcal{N}_C$ is linear in the number of escape terminals on the convex boundary, due to Remark 7.4.

Definition 7.5 *The flow network $\mathcal{N}_C$ (corresponding to constraint forest $\mathcal{F}$) can be augmented to the original flow network $\mathcal{N}$ (corresponding to the escape routing problem inside the convex boundary) as follows:*

Consider each vertex v in $\mathcal{N}$ that corresponds to an escape terminal on the boundary. If this escape terminal has a constraint associated with it in constraint forest $\mathcal{F}$, then

- *The edge from v to the sink vertex is removed.*

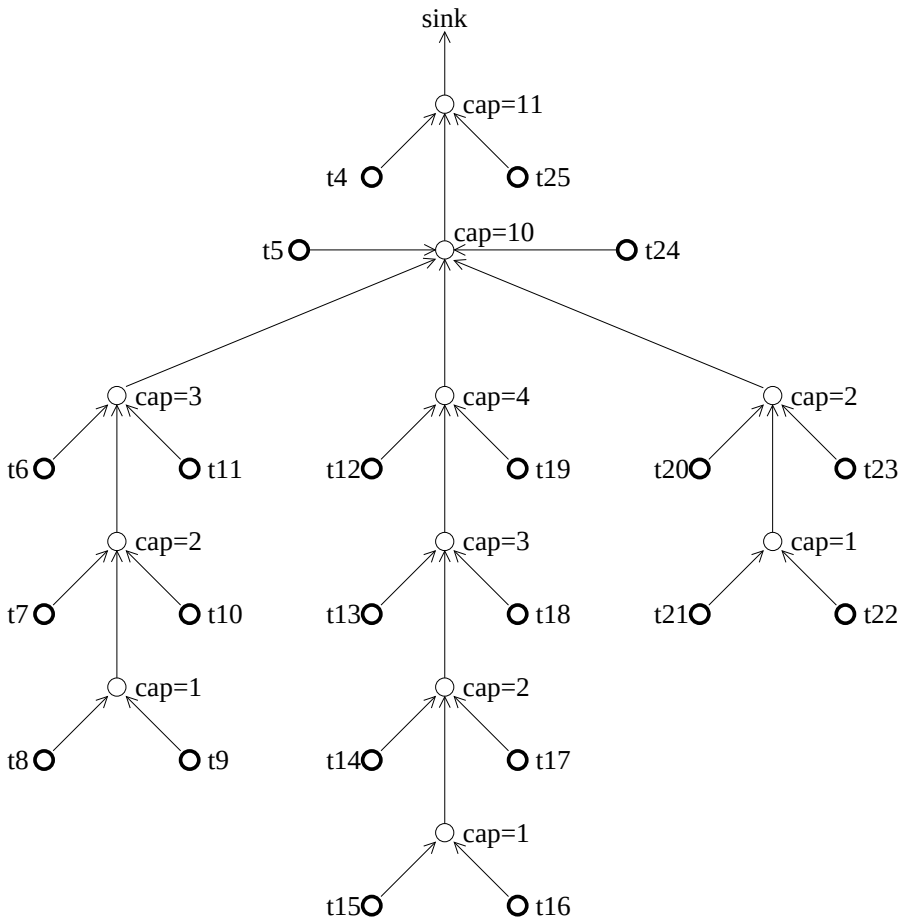


Figure 7.13: The flow network $\mathcal{N}_C$ corresponding to the constraint forest given in Figure 7.11(b). The dark and light circles represent t-vertices and c-vertices, respectively. The capacities of c-vertices, and the terminal indices of t-vertices are also shown.

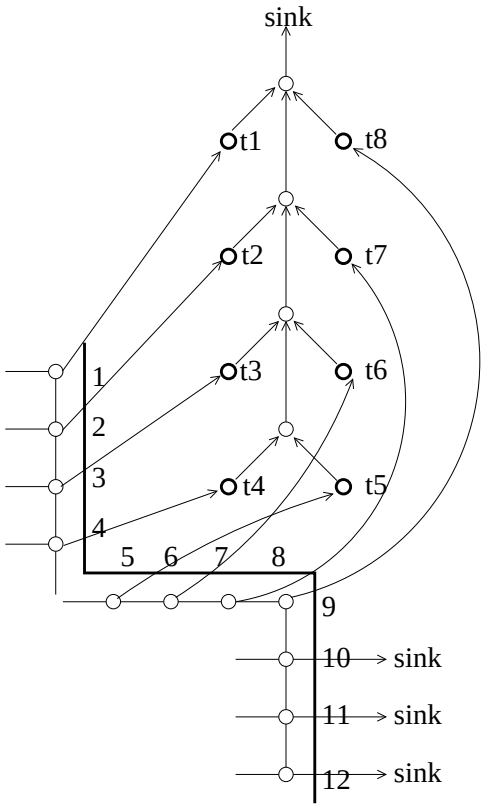


Figure 7.14: An example illustrating how to augment constraint network $\mathcal{N}_C$ to the original flow network $\mathcal{N}$. Here, an edge exists from each terminal vertex in $\mathcal{N}$ to the corresponding t-vertex in $\mathcal{N}_C$.

- *An edge is created from v to the corresponding t-vertex in $\mathcal{N}_C$.*

This augmentation process is illustrated in Figure 7.14 with a simple example. Here, the terminals on the corner (terminals 1-8) are connected to the corresponding t-vertices in $\mathcal{N}_C$. Since no constraint is associated with terminals 9-12, they are still connected to the *sink* vertex directly.

Theorem 7.3 *Let $\mathcal{N}$ denote the original flow network corresponding to the escape problem inside the boundary. Let $\mathcal{N}_C$ denote the constraint flow network as given in Definition 7.4. Assume that we augment $\mathcal{N}_C$ to $\mathcal{N}$ as described in Definition 7.5 to obtain the final flow network $\mathcal{N}_F$. The maximum flow on $\mathcal{N}_F$ will give the optimal escape routing solution that is also routable outside the convex boundary.*

PROOF. Here, we need to prove that there is a one-to-one correspondence between the maximal flow in $\mathcal{N}_F$ and the maximal escape routing solution that is also routable

outside the convex boundary. First, we will prove that any valid flow solution in $\mathcal{N}_F$ corresponds to a valid escape routing solution. We can state that any valid flow solution in $\mathcal{N}_F$ must satisfy all the conditions defined in Section 7.3, since $\mathcal{N}_C$ models the constraint forest exactly. We have also shown in Section 7.3 that these constraints are *sufficient* for routability outside the convex boundary. Hence, there is a valid escape routing solution corresponding to any flow in $\mathcal{N}_F$. Then, we can prove that there is a flow in $\mathcal{N}_F$ corresponding to any valid escape routing solution. We have proven in Section 7.3 that the constraints defined are *necessary* for routability outside the convex boundary. So, any valid escape routing solution must satisfy all these constraints; hence must have a corresponding valid flow in $\mathcal{N}_F$. ■

7.5.1 Discussions

A straightforward approach here could be to define a *rectangular* bounding box for the pin clusters, instead of a rectilinear convex boundary. In that case, we do not need to worry about the routability constraints outside, since all escape terminals are on a rectangular boundary. However, this approach increases the complexity of the escape problem *inside* the boundary. For example, let us consider the boundary segment in Figure 7.4. Defining a bounding box instead of this convex boundary would require the $k \times k$ grid outside the boundary to be included in the flow network inside. Depending on the convexity of the boundary, this approach can increase the size of the flow network quadratically. However, the constraint network we have defined in this section has only linear size in the number of escape terminals on the boundary. So, when we augment it to the original flow network, the complexity of the network flow algorithm does not increase.

Besides, in industrial practice, escape routing problem is not always solved by flow-based methodologies. This is mainly due to the high complexities and lack of constraint-handling capabilities of network flow algorithms. So, different heuristics for escape routing are being used in practice. A sample heuristic here can be based on routing pins to the closest escape terminals on the boundary. This heuristic would especially work well if most of the pins are close to the escape boundary. However, if we define the boundary as a bounding box instead of a rectilinear convex boundary, the pins will get farther away from the boundary, and the escape routing problem inside will get considerably more difficult. On the other hand, the routability constraints we have defined in this chapter can easily be incorporated into different heuristic-based

Table 7.1: Comparison of routability-driven escape routing with the traditional algorithm

PIN CLUSTER		TRADITIONAL ESCAPE ROUTING		ROUTABILITY-DRIVEN ESCAPE ROUTING	
Area	# Pins	# layers	time	# layers	time
7167	1687	13	1:12	10	0:57
8530	2080	14	1:44	11	1:26
9237	3742	26	3:34	21	2:53
9930	4885	31	5:02	26	4:13
10620	5984	38	7:04	32	5:51
12534	7638	47	11:02	40	8:59

escape routing algorithms, and they are applicable to arbitrary rectilinear convex boundaries.

7.6 Experimental Results

We have performed experiments to evaluate the practical effectiveness of the models and algorithms we have proposed. We have implemented all algorithms in C++, and performed the experiments on a Linux system with Intel Centrino 1.5GHz processor, and 512MB memory.

For comparison purposes, we have applied a network flow based escape routing algorithm on a set of test circuits, and then used the optimal algorithm (proposed in Section 7.4) to select the maximal subset of routes that are also routable outside the boundary. In other words, escape routing is performed without considering routability outside in the beginning, and then the unroutable nets for the current layer are removed. The results of this methodology are given in Tables 7.1 and 7.2 under the columns *traditional escape routing*. We have also implemented the integrated approach (proposed in Section 7.5), which considers routability constraints outside the boundary during the actual escape routing algorithm. A sample routing solution using this integrated methodology is given in Figure 7.15. Note here that all the necessary and sufficient conditions defined in Section 7.3 are satisfied in this solution, and it is guaranteed that all nets can be routed outside without any conflicts.

Table 7.1 gives the final routing results corresponding to these two methodologies. When routability outside the boundary is not considered during the actual escape

Table 7.2: Single-layer routing characteristics of the traditional and routability-driven escape routing algorithms

PIN CLUSTER		TRADITIONAL ESCAPE ROUTING			ROUTABILITY-DRIVEN ESCAPE ROUTING		
Area	# Pins	# escape	# routable	time	# escape	# routable	time
7167	1687	308	239	0:14	290	290	0:13
8530	2080	322	264	0:17	310	310	0:17
9237	3742	329	270	0:21	319	319	0:21
9930	4885	337	287	0:24	331	331	0:24
10620	5984	344	285	0:27	333	333	0:26
12534	7638	368	299	0:35	353	353	0:34

routing, more routing layers are needed, as can be observed in this table. However, when these constraints are integrated into the actual escape routing algorithm, the number of necessary layers decreases by 17% on average.

We have also listed the number of nets routed on the first layers of each circuit in Table 7.2. These quantities are important to observe the characteristics of these algorithms more closely, because each algorithm tries to route the maximal number of nets on the first layer. In this table, we list not only the number of nets that have escaped to the convex boundary, but also the number of nets that are routable outside. On average, the traditional escape routing algorithm routes 3.7% more nets on the first layer. However, on average 18.1% of these nets are not routable outside the boundary; so they need to be removed from the solution of this layer (and need to be propagated to the lower layers). However, when we consider routability constraints during the actual escape routing algorithm, it is guaranteed that the escape routing solution found is completely routable outside.

7.7 Concluding Remarks

In this chapter, we have studied the escape routing problem of irregular-shaped pin clusters, which are encountered frequently in in high-end MCMs. We have shown that routing nets to the cluster boundary without considering routability outside may lead to inferior solutions. We have proposed a set of necessary and sufficient conditions that model routability based on the positions of escape terminals on the boundary. Then, we have proposed an algorithm that selects the optimal subset

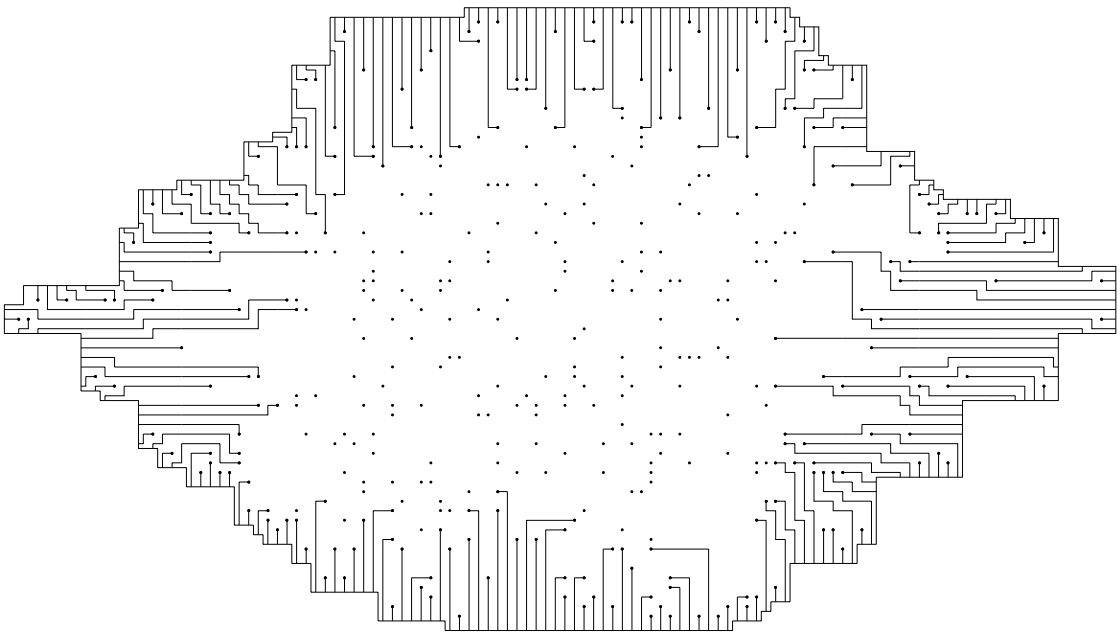


Figure 7.15: The escape routing solution on one layer of a sample pin cluster. 227 out of 414 nets have been routed on this layer. The solution found is also guaranteed to be routable outside.

of escape routes that are also routable outside. This algorithm is especially useful when a traditional routing algorithm is applied on a cluster of pins with a convex boundary. Then, we have shown how to integrate these constraints into the original routing algorithm without losing optimality. Our experiments have shown that the integrated methodology can reduce the number of layers by 17% on average.

Chapter 8

Conclusions and Future Directions

In this dissertation, we have proposed routing algorithms for high-performance VLSI packaging. Our objective has been to handle routing challenges due to increasing package densities, and high clock frequencies.

We have first focused on the problem of routing nets within tight min and max length constraints. This problem is becoming more and more important due to increasing clock frequencies, and increasing numbers of high-speed bus structures in the current high-end VLSI packages. In Chapter 2, we have proposed a Lagrangian relaxation based length matching routing algorithm, where the objective of satisfying min-max length constraints is effectively incorporated into the actual routing problem. Although this algorithm can be used for more general routing problems, we have also considered more restricted yet common problem instances and proposed more effective algorithms in Chapters 3 and 4. Specifically, we have focused on the two-layer bus routing problem between component boundaries in Chapter 3. We have modeled this problem as a job scheduling problem, and proposed algorithms to effectively solve it. In Chapter 4, we have focused on the problem of routing bus structures between component boundaries on a single layer. For this, we have proposed algorithms that are proven to give close-to-optimal solutions.

In the second half of the dissertation, we have focused on the escape routing problem, which is defined as routing nets from individual pins within dense components to the component boundaries. Due to increasing package densities, the escape routing problem is increasingly becoming the main bottleneck in terms of overall routability [61]. In Chapter 5, we have proposed fundamental models and algorithms to solve the escape routing problem in two components simultaneously, such that the number

of crossings in the intermediate area is minimized. Then, in Chapter 6, we have focused on the practical aspects of this problem, and we have proposed improvements on the fundamental models and algorithms of Chapter 5. Finally, in Chapter 7, we have focused on the escape routing problem within dense pin clusters, which can have arbitrary convex boundaries. Here, we have proposed a set of sufficient and necessary conditions that guarantee routability outside the escape boundary. We have also discussed how these conditions can be used effectively within an escape routing framework.

An important future research direction here is to develop a package-level routing system based on the fundamental algorithms proposed in this dissertation. As noted in Chapter 5, the basic assumption used in our simultaneous escape routing algorithm is that the problem consists of two components separated with a channel. In Section 5.5, we have discussed how to generalize this assumption by merging multiple components into (conceptual) super-components, and applying our algorithms on these super-components. However, for complex board designs, we need an algorithm that automatically identifies the best pairs of components to be routed on each layer of a complex design. Once the component pairs are identified for the current layer, the proposed escape routing algorithms can be applied on each component pair separately. As mentioned before, typical industrial boards today contain large bus structures between pairs of components. Basically, we need a *bus planning* algorithm that identifies the best subset of buses to be routed on the current layer. After that, the escape routing algorithm given in Chapter 5 can be used to determine the routing solutions of individual nets from their terminal pins to the corresponding component boundaries. Once all nets are routed to their component boundaries, it is possible to use one of the area routing algorithms we have proposed in Chapters 2, 3, and 4 to route nets between different component boundaries. In particular, if buried vias are allowed, the two-layer routing algorithm proposed in Chapter 3 can be used to route nets between pairs of component boundaries. Otherwise, the single-layer routing algorithm proposed in Chapter 4 can be used for this purpose. For the remaining nets that do not belong to any regular bus structure (i.e., for which a well defined channel cannot be defined), the general length-matching routing algorithm given in Chapter 2 can be used.

References

[1] P. Berman and G. Schnitger. On the complexity of approximating the independent set problem. *Inform. and Comput.*, 96:77–94, 1992.

[2] V. Betz and J Rose. Directional bias and non-uniformity in FPGA global routing architectures. In *Proc. of ICCAD*, pages 652–659, 1996.

[3] V. Betz and J. Rose. VPR: A new packing, placement and routing tool for FPGA research. In *Proc. of the 7th International Workshop on Field-Programmable Logic*, pages 213–222, 1997.

[4] J. Blazewicz, K. H. Ecker, and E. Pesch. *Scheduling Computer and Manufacturing Processes*. Springer, 2001.

[5] K. D. Boese, J. Cong, A. B. Kahng, K. S. Leung, and D. Zhou. On high-speed VLSI interconnects: Analysis and design. In *Proc. Asia–Pacific Conf. Circuits Syst.*, pages 35–40, 1992.

[6] H. N. Brady. An approach to topological pin assignment. *IEEE Trans. on Computer Aided Design*, 3:250–255, 1984.

[7] M. Burstein and R. Pelavin. Hierarchical channel router. In *Proc. of IEEE/ACM 20th Design Automation Conference*, pages 591–597, 1983.

[8] G. Capwell. High density design with MicroStar BGAs. Application Report SPRA471A, Texas Instruments, 1998.

[9] W.-T. Chan, F. Y. L. Chin, and H.-F. Ting. Escaping a grid by edge-disjoint paths. In *Proc. of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, pages 726–734, 2000.

[10] T Cohen. Practical guidelines for the implementation of back drilling plated through hole vias in multi-gigabit board applications. In *Proc. of IEC DesignCon*, 2003.

[11] J. Cong, A. B. Kahng, C.-K. Koh, and C.-W. A. Tsao. Bounded-skew clock and steiner routing. *ACM Trans. on Design Automation of Electronic Systems*, 3(3):341–388, 1998.

[12] J. Cong, A. B. Kahng, G. Robins, M. Sarrafzadeh, and C. K. Wong. Performance-driven global routing for cell based IC's. In *Proc. of IEEE Int. Conf on Computer Design*, pages 170–173, 1991.

[13] J. Cong, A. B. Kahng, G. Robins, M. Sarrafzadeh, and C. K. Wong. Provably good performance-driven global routing. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 11(6):739–752, 1992.

[14] J. Cong and C. L. Liu. On the k-layer planar subset and topological via minimization problems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 10, 1991.

[15] J. S. Corbin, C. N. Ramirez, and D. E. Massey. Land grid array sockets for server applications. *IBM Journal of Research and Development*, 46:763–778, 2002.

[16] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, 1992.

[17] P. Crescenzi and V. Kann. A compendium of NP optimization problems. http://www.nada.kth.se/ viggo/problemlist.

[18] A. E. Dunlop, V. D. Agrawal, D. N. Deutsch, M. F. Jukl, P. Kozak, and M Wiesel. Chip layout optimization using critical path weighting. In *Proc. of Design Automation Conference*, pages 133–136, 1984.

[19] C. Ebeling, L. McMurchie, S. A. Hauck, and S. Burns. Placement and routing tools for the triptych FPGA. *IEEE Trans. on VLSI*, pages 473–482, 1995.

[20] S. C. Fang, W. S. Feng, and S. L. Lee. A new efficient approach to multilayer channel routing problem. In *Proc. of the 29th ACM/IEEE Design Automation Conference*, pages 579–584, 1992.

[21] M. L. Fisher. The Lagrangian relaxation method for solving integer programming problems. *Management Science*, 27(1):1–18, 1981.

[22] M. L. Fisher. An applications oriented guide to Lagrangian relaxation. *Interfaces*, 15(2):10–21, 1985.

[23] A. M. Geoffrion. Lagrangian relaxation and its uses in integer programming. *Mathematical Programming*, 2:82–114, 1974.

[24] X. Guan, P. B. Luh, and L. Zhang. Nonlinear approximation method in Lagrangian relaxation-based algorithms for hydrothermal scheduling. *IEEE Transactions on Power Systems*, 10(2):772–778, 1995.

[25] X. H. Guan, Q. Z. Zhai, and F Lai. New Lagrangian relaxation based algorithm for resource scheduling with homogeneous subproblems. *Journal of Optimization Theory and Applications*, 113(1):65–82, 2002.

[26] H. Harrer, H. Pross, Winkel T.-M, W. D. Becker, H. I. Stoller, M. Yamamoto, S Abe, B. J. Chamberlin, and G. A Katopis. First- and second-level packaging for the IBM eServer z900. *IBM Journal of Research and Development*, 46:397–420, 2002.

[27] A. Hashimoto and J. Stevens. Wire routing by optimizing channel assignment within large apertures. In *IEEE Proc. of 8th Design Automation Workshop*, pages 214–224, 1971.

[28] M. H. Held, P. Wolfe, and H. D. Crowder. Validation of subgradient optimization. *Mathematical Programming*, 6(1):62–88, 1974.

[29] J. Hershberger and S. Suri. Efficient breakout routing in printed circuit boards. In *Proc. of the thirteenth annual symposium on Computational geometry*, pages 460–462, 1997.

[30] C. Hsu. General river routing algorithm. In *Proc. of 20th Design Automation Conference*, 1983.

[31] D. J.-H. Huang, A. B. Kahng, and C.-W. A. Tsao. On the bounded-skew clock and Steiner routing problems. In *Proc. 32nd ACM/IEEE Design Automation Conf.*, pages 508–513, 1995.

[32] A. B. Kahng and G. Robins. *On Optimal Interconnections in VLSI*. Kluwer Academic Publishers, 1995.

[33] G. A. Katopis, W. D. Becker, T. R Mazzawy, H. H. Smith, C. K. Vakirtzis, S. A. Kuppinger, B. Singh, P. C. Lin, J. Bartells, G. V. Kihlmire, P. N. Venkatachalam, H. I. Stoller, and J. L Frankel. MCM technology and design for the S/390 G5 system. *IBM Journal of Research and Development*, 43, 1999.

[34] N. L. Koren. Pin assignment in automated printed circuit board design. In *Proc. of the ninth design automation workshop on Design automation*, pages 72–79, 1972.

[35] E. Kuh, M. A. B. Jackson, and M Marek-Sadowska. Timing-driven routing for building block layout. In *Proc. IEEE International Symposium on Circuits and Systems*, pages 518–519, 1987.

[36] A. S. LaPaugh. *Algorithms for Integrated Circuit Layout: An Analytic Approach*. PhD thesis, Laboratory for Computer Science, MIT, Cambridge, MA, 1980.

[37] S. Lee and M. D. F. Wong. Timing-driven routing for FPGAs based on Lagrangian relaxation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 506–510, 2003.

[38] C. E. Leiserson and R. Y. Pinter. Optimal placement for river routing. In *Proc. of the CMU Conf. on VLSI Systems and Computations*, 1981.

[39] T. Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout*. Wiley, New York, 1990.

[40] J. Ludwig. IBM Systems Group. Private communication, 2004.

[41] M. M. Ozdal and M. D. F. Wong. Length matching routing for high-speed printed circuit boards. In *Proc. of IEEE Intl. Conf. on Computer-Aided Design (ICCAD)*, Nov. 2003.

[42] M. M. Ozdal and M. D. F. Wong. A provably good algorithm for high performance bus routing. In *Proc. of IEEE/ACM Intl. Conf. on Computer-Aided Design (ICCAD)*, Nov. 2004.

[43] M. M. Ozdal and M. D. F. Wong. Simultaneous escape routing and layer assignment for dense PCBs. In *Proc. of IEEE/ACM Intl. Conf. on Computer-Aided Design (ICCAD)*, Nov. 2004.

[44] M. M. Ozdal and M. D. F. Wong. A two-layer bus routing algorithm for high-speed boards. In *Proc. of IEEE Intl. Conf. on Computer Design (ICCD)*, Oct. 2004.

[45] M. M. Ozdal and M. D. F. Wong. An algorithmic study of single-layer bus routing for high-speed boards. *IEEE Trans. on Computer-Aided Design*, (to appear).

[46] M. M. Ozdal and M. D. F. Wong. A two-layer bus routing algorithm for high-speed boards. *ACM Trans. on Design Automation of Electronic Systems*, (to appear).

[47] M. M. Ozdal, M. D. F. Wong, and P. Honsinger. An escape routing framework for dense boards with high-speed design constraints. In *Proc. of IEEE/ACM Intl. Conf. on Computer-Aided Design (ICCAD)*, Nov. 2005 (to appear).

[48] M. M. Ozdal, M. D. F. Wong, and P. Honsinger. Optimal routing algorithms for pin clusters in high-density multichip modules. In *Proc. of IEEE/ACM Intl. Conf. on Computer-Aided Design (ICCAD)*, Nov. 2005 (to appear).

[49] M. Pecht and Wong Y. T. *Advanced Routing of Electrical Modules*. CRC Press Inc., 1996.

[50] R. Y. Pinter. On routing two-point nets across a channel. In *Proc. of the 19th Design Automation Conference*, 1982.

[51] R. Y. Pinter. River routing: Methodology and analysis. In *Proc. of the 3rd Caltech Conf. on VLSI*, 1983.

[52] S. Prastjutrakul and W. J. Kubitz. A timing-driven global router for custom chip design. In *Proc. of IEEE Intl. Conf. on Computer-Aided Design*, pages 48–51, 1990.

[53] A. Renaud. Daily generation management at Electricite de France: Form planning toward real time. *IEEE Transaction on Automatic Control*, 38(7):1080–1093, 1993.

[54] L. W. Ritchey. Busses: What are they and how do they work? *Printed Circuit Design Magazine*, 2000.

[55] R. L. Rivest and C. M. Fiduccia. A greedy channel router. In *Proc. of IEEE/ACM 19th Design Automation Conference*, pages 418–424, 1982.

[56] Szymanski. Dogleg channel routing is NP-complete. *IEEE Transactions on Computer-Aided Design*, CAD-4(1):31–41, 1985.

[57] A. Titus, B. Jaiswal, T. Dishongh, and A. N Cartwright. Innovative circuit board level routing designs for BGA packages. *IEEE Trans. on Advanced Packaging*, 27, 2004.

[58] C.-W. A. Tsao and C.-K. Koh. UST/DME: A clock tree router for general skew constraints. *ACM Transactions on Design Automation of Electronic Systems*, 7:359–379, 2002.

[59] S. J. Wang, S. M. Shahidehpour, D. S. Kirschen, S. Mokhtari, and G. D. Irisari. Short-term generation scheduling with transmission constraints using augmented Lagrangian relaxation. *IEEE Transactions on Power Systems*, 10(3):1294–1301, 1995.

[60] D. Wiens. Printed circuit board routing at the threshold. *The Board Authority*, pages 44–47, December 2000.

[61] T.-M. Winkel, W. D. Becker, H. Harrer, H. Pross, D. Kaller, B. Garben, B. J. Chamberlin, and S. A. Kuppinger. First- and second-level packaging of the z990 processor cage. *IBM Journal of Research and Development*, 48:379–394, 2004.

[62] H. Xiang, X. Tang, and D. F. Wong. An algorithm for simultaneous pin assignment and routing. In *Proc. of Intl. Conf. on Computer Aided Design*, 2001.

[63] T. Yoshimura and E. S. Kuh. Efficient algorithms for channel routing. *IEEE Transactions on Computer-Aided Design*, CAD-1(1):25–35, 1982.

[64] M. Yu and W. W. Dai. Single-layer fanout routing and routability analysis for ball grid arrays. In *Proc. of Intl. Conf. on Computer-Aided Design (ICCAD)*, pages 581–586, 1995.

[65] H. Zhou and M. D. F. Wong. Optimal river routing with crosstalk constraints. *ACM Transactions on Design Automation of Electronic Systems*, 3(3):496–514, 1998.

AUTHOR'S BIOGRAPHY

Muhammet Mustafa Ozdal received both his B.S. degree in electrical engineering (1999), and his M.S. degree in computer engineering (2001) from Bilkent University, Turkey. He is currently a Ph.D. candidate in the Department of Computer Science at the University of Illinois at Urbana-Champaign. His current research interests include algorithms for computer aided design of VLSI circuits with a primary focus on physical design algorithms. Following the completion of his Ph.D., he will begin to work in the Design Technology Department of Intel Corporation.