Shortest Paths and Steiner Trees in VLSI Routing

Dissertation

zur Erlangung des Doktorgrades

der Mathematisch-Naturwissenschaftlichen Fakultät

der Rheinischen Friedrich-Wilhelms-Universität Bonn

vorgelegt von

Sven Peyer

aus Erfurt

im Oktober 2007

Angefertigt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät
der Rheinischen Friedrich-Wilhelms-Universität Bonn

Erstgutachter: Professor Dr. Bernhard Korte
Zweitgutachter: Professor Dr. Jens Vygen

Tag der Promotion: 14. Dezember 2007

Die Theorie ist nicht die Wurzel,
sondern die Blüte der Praxis.

Ernst von Feuchtersleben (1806-1849)

Acknowledgments

I would like to express my gratitude to my supervisors, Professor Dr. Bernhard Korte and Professor Dr. Jens Vygen. I benefited a lot from their ideas, experience and guidance. This work would not have been possible without them, and I am happy to be part of their research team. Under their leading, the Research Institute for Discrete Mathematics at the University of Bonn provides optimal working conditions, which makes it a distinctive place of research with practical relevance. It is a great source of motivation and satisfaction to work on leading-edge technologies in a unique cooperation with industrial partners.

I am very thankful to Professor Dr. Matthias Müller-Hannemann, Professor Dr. Dieter Rautenbach and Professor Dr. Martin Zachariasen. They are great co-authors, and it was a pleasure to work together closely and investigate Steiner trees and shortest paths.

I particularly thank all my colleagues in the BonnRoute team, namely Dirk Müller, Dr. Tim Nieberg, Christian Panten, Dr. André Rohe and Christian Schulte. I enjoyed the daily discussions on BonnRoute and our joint coding work. There are many people at the institute who contributed to the routing project in some way or another, and my thanks go to all of them.

I am very grateful to all people at IBM Corporation who shared their knowledge of VLSI design with me, especially Dr. Markus Bühler, Dr. Jürgen Koehl, Karsten Muuss and Dr. Matthias Ringe.

My special thanks go to Dr. Ulrich Brenner and Dr. Jürgen Werber. We have been colleagues for ten years, and together we went through exciting times. Ulrich frequently cheered me on late at night when he came into my office and asked, “So, have you written a new page today?” Furthermore, I thank Christian, Dirk, Jürgen and Tim for reading substantial parts of my thesis and making valuable comments.

Naturally, many people at the institute were helpful because of their work in the background: Duke Keiper, Ralf Jann and Michael “James” Hahmann, just to mention three.

I have great friends who made life easier in difficult times and showed continued interest in the progress of my studies, but also helped me to take a break and to take my mind off this thesis. I can hardly list them all, but special thanks go to Dr. Ralf Häfner, Mechthild Köditz, Alexander Lademann, Dr. Falk Tschirschnitz and Lutz Volke.

I am very grateful to my dear parents, Marlies and Wolfgang Peyer, who always gave me the greatest possible assistance for my work. Thanks to them, I had a carefree student life.

Finally, I would like to thank my partner, Katrin Heyne, for her continuous loving support, tolerance and patience, and my children Paula and Felix, who did not see me a lot during the last months. There was one hard question Felix often asked me in the morning: “Dad, will you come home late tonight?” Sadly, I often had to say yes, but I am so fond of those incredible moments when I came home to my family and saw my kids run into me.

Contents

Chapter 1

Introduction

Very-large-scale integration (VLSI) design, the process of creating complex integrated circuits, is one of the most important and appealing application areas for mathematics. A major part is physical design; it leads to a wide range of combinatorial optimization problems which are of both theoretical and practical interest. Due to the rapid technology development and growing complexity of VLSI chips, tools based on very efficient algorithms are needed to cope with the requirements of a highly automated design process.

Over the last 20 years, the Research Institute for Discrete Mathematics at the University of Bonn has been developing the BonnTools, a VLSI toolkit for physical design, as part of a long-term cooperation with IBM Corporation. This software have been used on a large number of leading-edge chips in design centers all over the world. The package comprises applications for all major parts of physical design: placement, timing optimization, clock tree design and routing. In this work, we examine the problem of VLSI routing under theoretical as well as practical aspects. The practical implementation is called BonnRoute, the routing program of the BonnTools. Many of the most complex industrial chips have been designed using BonnRoute.

In this thesis, we start by presenting the routing problem in Chapter 2. Its task is to find disjoint wire connections between sets of points on the chip. For each individual set of points to be connected, specific constraints have to be taken into account. Moreover, routing blockages have to be avoided, and several other types of constraints have to be obeyed. The three main optimization goals are timing, power and yield. It is usually desirable to consider more than one of these objectives at the same time.

A simplified view of the VLSI routing problem is as follows. In a subgraph of a three-dimensional grid we look for vertex-disjoint Steiner trees connecting given terminal sets. (There are additional complications in practice, which do not change the core algorithmic problem much.) The standard approach, which is also taken by BonnRoute, is to split the routing problem into two major parts: first, the *global routing* consists in packing Steiner trees in a coarsened grid subject to edge capacities. As a second step, the *detailed routing* determines the exact layout, essentially by computing shortest paths sequentially

within the global routing corridors and using a subsidiary ripup-and-reroute strategy. In Chapter 2, we go into the theoretical details of both parts.

Steiner trees and shortest paths are the two main mathematical concepts in routing. The rectilinear Steiner tree problem in the plane asks for a minimum-length tree interconnecting a set of terminals and consisting only of horizontal and vertical line segments. For the instances which typically occur in VLSI design, rectilinear Steiner minimum trees (RSMTs) can today be computed quickly. As interconnect signal delays are becoming increasingly important, the length of paths in a tree — or even a measure which reflects delay directly — should be taken into account in the construction. In Section 3.1 we consider the problem of finding an RSMT that minimizes a secondary objective related to signal delay. As a major result of this thesis, we derive structural properties of RSMTs for which the weighted sum of path lengths from a designated source to the other terminals is minimized. We also present exact and heuristic algorithms for constructing RSMTs with the secondary objective of minimizing either the weighted sum of path lengths or the so-called Elmore delay, a standard wire delay approximation used in VLSI timing analysis. Finally, computational results for industrial designs are presented.

In Section 3.2 we consider the problem of finding a shortest rectilinear Steiner tree for a given set of points in the plane in the presence of rectilinear obstacles. The Steiner tree is not required to avoid the obstacles completely; however, if the Steiner tree intersects an obstacle, then no connected component of the induced subtree must be longer than a given fixed length. This reflects the insertion of repeaters in a large wiring tree, which are necessary for electrical correctness, but must not be placed on top of obstacles. We show that this problem can be approximated within twice the optimum length in polynomial time. Another main result of this thesis is a generalization of the Hanan grid theorem. Using this structural property, we show how to improve the performance guarantee of the approximation to a factor which is arbitrarily close to the best bound for the classical Steiner tree problem in graphs.

The second central concept in routing (besides Steiner trees) is to construct a shortest wiring connection between two metal components that must be connected electrically. This can be modeled as a shortest paths problem in a partial grid graph and can be solved with Dijkstra's algorithm, the classical method for finding shortest paths in digraphs with non-negative edge lengths. Since a major part of the detailed routing running time is spent in path-search routines, several speed-up techniques are used routinely today.

Routing speed-ups are a crucial lever for reducing the time needed for the overall design process. Goal-oriented modifications of Dijkstra's algorithm are typical approaches. A further important contribution of this thesis is a framework for speeding up Dijkstra's algorithm, which is presented in Chapter 4. Instead of labeling individual vertices, we start from a partition of the underlying graph into subgraphs and assign labels to the subgraphs. If their number is small compared to the order of the original graph and the shortest path problems restricted to these subgraphs are computationally easy, this approach will lead to a substantial reduction in running time. The framework is generic and can be specialized in different ways. We apply it to the VLSI routing problem, whose

computational challenge is due to the fact that we need to find millions of shortest paths in partial grid graphs with billions of vertices. In this context, the modified path search is applied twice: first in a coarse abstraction (where the labeled subgraphs are rectangles), and then in a detailed model (where the labeled subgraphs are intervals). Using the result of the first algorithm to speed up the second one via goal-oriented techniques leads to considerably improved running times. Experimental results on leading-edge industrial chips constitute a practical justification of our approach, complementing the theoretical worst-case time bounds.

For a routing tool to stay top for more than a decade, extensive efforts in coding, maintenance and testing are mandatory. In Chapter 5 we present computational results achieved by BonnRoute on real-world VLSI chips. They show that BonnRoute performs excellently on all traditional quality measures such as wire length and number of vias, but also on further criteria of equal importance in the every-day work of the designer. Due to today's time-to-market pressure it is also necessary to minimize the time needed to complete the full design process. Our experiments demonstrate that BonnRoute is a very effective and efficient routing tool and fulfills all requirements of state-of-the-art VLSI routing.

Chapter 2

Routing

The goal of this chapter is to give an introduction to routing in VLSI design and an overview of the program BonnRoute. Since routing is the last major step in the design flow we start with a short description of the overall VLSI design process. We set up the routing problem and formalize the routing task. In the main part of this chapter we present the key components of BonnRoute which comprises BonnRouteGlobal and BonnRouteLocal.

2.1 VLSI Design Flow

In this section we give a basic explanation of the VLSI design process, which consists of two main parts: logical and physical design.

The functionality of a chip has to be modeled first at the behavioral level and can be expressed by means of a *hardware description language (HDL)*. The two commonly used standard HDL formats are VHDL (IEEE [1994]) and Verilog (Thomas and Moorby [2002]). The HDL compiler translates the specification into a *register transfer level (RTL)* model and then into a logic description, which is not optimized yet. This is the first step of *logic synthesis*. Another important application of logic synthesis is logic optimization which is applied during many subsequent stages in the design flow. The objective of logic optimization is to find an equivalent description of the logic function such that the physical implementation of the chip is as compact as possible and timing constraints can be met. After initial logic optimization steps which result in a *netlist* with standard components (such as NANDs and NORs), the netlist must be mapped to books of a given library. A *library* is a set of logic components (*books*) which can be used to implement a given logic function. It contains standard books such as NAND, NOR, or INVERTER and more complex modules such as ADDER/MUX. There are many instances of each book which have distinct layout and different properties with respect to area consumption, load capacitance and timing. These instances are called *circuits* (or *gates*). A circuit contains

a set of (*pins*) which serve as connection points to other circuits. Pins of different circuits form a *net*, and they are connected by *wires*.

For the physical layout some parameters, such as the chip image, the number of routing layers, and the technology have to be specified in advance. The technology constraints define the physical characteristics of components of the chip, give capacitance and resistance values for wires and state so-called *design rules* which are discussed in detail in Section 5.3.2. For further reading on logic synthesis, see Devadas, Ghosh and Keutzer [1994].

The second main part of the VLSI design flow is the *physical design* of a chip which includes *placement*, *timing optimization* and *routing*.

In placement, all circuits of a chip have to be placed disjointly on the chip area. Although it is even *NP*-hard to decide whether there exists a feasible solution to this problem, in practice, such a solution can usually be found. This is due to the fact that most of the circuits have standard height and vary only in a few different widths. Moreover, the area customized by all circuits is sufficiently small compared to the entire chip area.

The objective of the placement step can be manifold: timing-critical nets should be realized as short as possible. This is done by imposing higher weights on nets which are identified as timing-critical. In practice, this approach is often managed in a loop which performs logic optimization and placement changes successively. Another requirement on placement is that the design is routable and does not contain highly congested regions. Here, a congestion estimator which runs fast and detects routing-critical areas reliably is essential to guarantee good results (Brenner and Rohe [2003]). For a good survey on theoretical as well as practical aspects of placement, see Brenner [2005].

The optimization of the timing behavior of a chip is an important task in the VLSI design process. Its main goal is to achieve timing closure, i.e. to meeting all timing constraints, for which various algorithms are applied. Here, we only briefly mention the major design steps in timing optimization.

The slack at a sink of a net is the difference of required and computed arrival time. A negative slack indicates that given timing constraints are not met. As feature size shrinks with new technologies, the interconnect delay becomes increasingly dominant over circuit delay. *Repeaters* (buffers and inverters) are used to repower a signal over a long distance. A repeater tree (also called fanout tree) should be constructed such that it maximizes worst slack and minimizes total wire length simultaneously.

The timing behavior of a gate strongly depends on two physical characteristics: input capacitance and driver strength, both depending on area and power consumption of the gate. A larger gate typically results in a smaller downstream delay and larger upstream delay. The task of gate sizing is to minimize an objective function, for example power or area consumption, while meeting all timing restrictions.

The threshold voltage of a gate also affects the timing behavior and power consumption of the gate. The higher the threshold, the higher the delay through the gate, but the lower the

leakage power consumption. This trade-off is subject of the V_t-assignment problem which aims at choosing the right threshold voltage for all circuits to minimize power consumption without violating timing constraints.

Most computations on a chip are synchronized by a periodic clock signal. This signal controls the times when bits are stored in storage elements and when they are released for computations. It can be shown that the overall cycle time can be decreased by assigning individual clock signal arrival times for the storage elements instead of having simultaneous clock signals. The task of clock skew scheduling and clock tree synthesis is to determine the best arrival times for clock signals and to build a clock tree which realizes this assignment.

Logic restructuring changes the logic structure of the netlist and is a mean to improve the worst slack of a net on top of timing optimization techniques. It can perform local exchange operations or even replace an entire path by another.

In practice, timing-closure can only be achieved by an iterative approach which calls optimization steps described above in a timing-driven placement loop. Among a whole bunch of literature on the subject of timing optimization we refer to Korte, Rautenbach and Vygen [2007].

There are many papers and books in the literature on VLSI design in general, e.g. consider the few survey books by Gerez [1998], Sherwani [1999], and Sait and Youssef [1999]. For a comprehensive and detailed work with the focus on theoretical aspects we refer to Vygen [2001]. A good overview of the mathematical components of the BonnTools is given by Korte, Rautenbach and Vygen [2007]. Finally, Alpert, Mehta and Sapatnekar [2008] publish a book on state-of-the-art VLSI algorithms.

2.2 The Routing Problem

We consider the stage of the design flow, where all placement and timing optimization steps are assumed to be finished. (This assumption is a slight simplification. In practice, it may be necessary to return to placement and timing steps in order to insert engineering change orders or to correct violations produced by routing.) The final task of the VLSI design flow is — informally expressed — to connect all nets of the netlist disjointly on the chip such that all given constraints are met. Thereby, properties of the nets have to be considered and blockages have to be avoided. We now discuss the instance of the routing task in more detail and formulate the routing problem finally.

2.2.1 Physical Description of a Chip

Let $A_0 := [x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}] \times [z_{\min}, z_{\max}] \subset \mathbb{R}^3$ be the three-dimensional chip area. We assume $z_{\min}, z_{\max} \in \mathbb{Z}_{\geq 0}$. A chip has a number of *wiring layers* $z_{\min}, z_{\min}+1, \ldots, z_{\max}$. For two adjacent wiring layers $z, z+1$ with $z \in \{z_{\min}, \ldots, z_{\max}-1\}$ the interval $(z, z+1)$

is called *via layer* which is referenced by index z. A (routing) *layer* is either a wiring layer or a via layer.

As the output of the logic optimization steps we have a fixed netlist, which consists of a set $\mathcal{N}$ of nets. Each net $N \in \mathcal{N}$ contains a set of pins of which one serves as the electrical source and all the others are sinks. All pins of a net have to be connected by wiring. Each pin can be decomposed into classes of so-called *soft* and *hard pin areas*, which are sets of rectangular metal shapes. Shapes of soft pin areas are not electrically connected to each other whereas shapes of hard pins are. In practice, most pins consist of only one class, they are either soft or hard. Most pins are located on the lower wiring layers. Some pins may be found on upper wiring layers, particularly those of macros. Figure 2.1 visualize parts of the structure of a real chip including pins and wires.

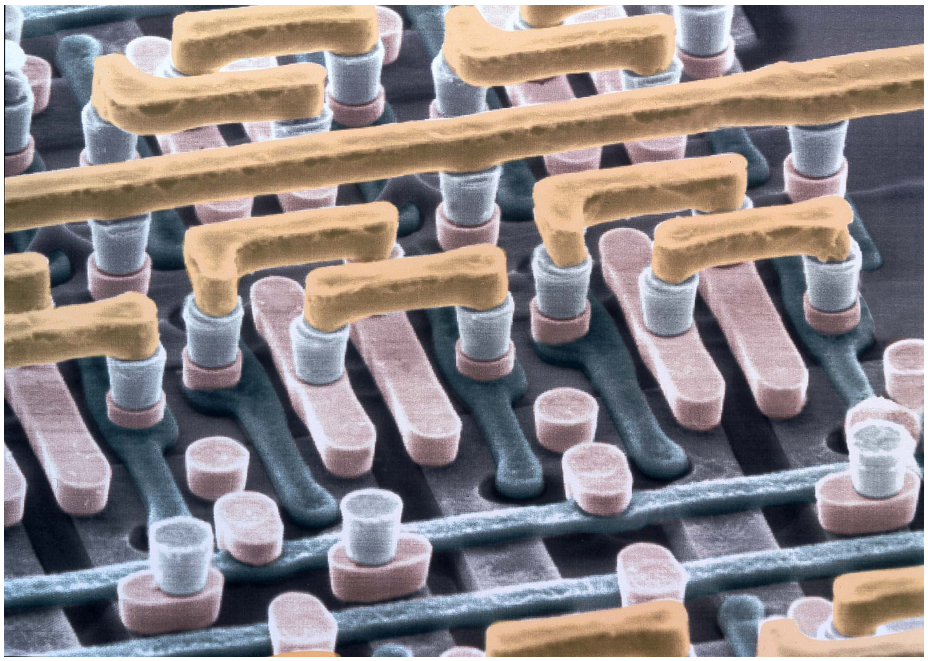


Figure 2.1: Photo of a real chip taken by an electron microscope. For the sake of exposure, silicon dioxide has been removed and the metal structure is artificially colored. It displays pins and wires of lower wiring layers, connected by vias (light blue).

From a manufacturing perspective, the width of a wire solely must not be smaller than a technology dependent minimum width. However, routing is often restricted to a set Ω of *wiretypes* in order to make routing manageable and to fulfill additional constraints (see Section 2.2.2). Each net is assigned a set $\mathcal{A}_N$ of pairs (ω, A) where wiretype $\omega \in \Omega$ is allowed to be used in the non-empty area $A \subseteq A_0$. The set $\mathcal{A}_N$ is called wiretyped routing area.

A wiretype $\omega \in \Omega$ is defined by a set of layer specific wiremodels and viamodels (at most one model per layer). Each *wiremodel* is a triple (s_z, R_z, C_z) defined on wiring

layer $z \in \{z_{\min}, \ldots, z_{\max}\}$. The shape of the wiremodel relative to a reference point (called *anchor point*) is defined by s_z. A very general description of a shape is a tuple $s_z = (o_W, o_E, o_S, o_N)$, defining overhang o. An explicitly required extra spacing may be associated with a wiremodel in order to force neighboring wires to keep a sufficiently large distance. For timing optimization and evaluation, vectors R_z and C_z give best, worst and nominal values for the resistance and capacitance per unit length of every wiretype on a wiring layer z. Resistance and capacitance of a wire with fixed length only depend on the width of the wire. The resistance of a wire is proportional to the length divided by the width, and the capacitance of a wire is proportional to the area of the wire.

A *viamodel* is defined similarly to a wiremodel. It is a tuple $(s_z^{\text{bot}}, s_z^{\text{mid}}, s_z^{\text{top}}, R_z)$ on via layer $(z, z+1)$ with $z \in \{z_{\min}, \ldots, z_{\max}-1\}$. The outline of the via is defined by its bottom pad shape s_z^{bot} on wiring layer z, its stem s_z^{mid} on via layer $(z, z+1)$ and its top pad shape s_z^{top} on wiring layer $z+1$; see Figure 2.2. Since a via has negligible capacitance, only its resistance R_z is given. Wiremodels and viamodels may contain some more parameters which we do not need for our description.

A *segment* is a triple $(\omega, (x_1, y_1, z_1), (x_2, y_2, z_2)) \in \Omega \times A_0 \times A_0$ with $x_1 \le x_2, y_1 \le y_2$, $z_2 \in \{z_1, z_1+1\} \subset \mathbb{Z}_{\ge 0}$ and either $x_1 \neq x_2$ or $y_1 \neq y_2$ or $z_1 \neq z_2$. The one-dimensional line defined by the two end points (x_1, y_1, z_1) and (x_2, y_2, z_2) is referred as the *stick-figure* of the segment. In our connectivity model, two segments are electrically connected if and only if their stick-figures intersect. Although shape connectivity is sufficient, stick-figure connectivity is helpful in the description of routing. For sake of simplicity, we assume that two wires intersect only at their endpoints which can always be achieved by splitting segments at their intersection. We further assume that a segment electrically connects a pin if and only if its stick-figure intersects the pin.

The length of a segment e is $l(e) := |x_2 - x_1| + |y_2 - y_1| + |z_2 - z_1|$. A segment defines the shape of a wire segment (or *wire* for short) if $z_1 = z_2$. The x-y-expansion of the wire is defined by the shape parameters s_{z_1} of the wiremodel $(s_{z_1}, R_{z_1}, C_{z_1})$ on wiring layer z_1 of wiretype ω. For $s_{z_1} := (o_W, o_E, o_S, o_N)$ it is $[x_1 - o_W, x_2 + o_E] \times [y_1 - o_S, y_2 + o_N] \times \{z_1\} \subseteq A_0$. We may assume $o_W + o_E = o_S + o_N$, and define the *width* of a wire by $o_W + o_E$. For $z_2 = z_1 + 1$, a via segment (or *via* for short) is defined by the viamodel $(s_{z_1}^{\text{bot}}, s_{z_1}^{\text{mid}}, s_{z_1}^{\text{top}}, R_{z_1})$. It connects locations of adjacent wiring layers of the same x- and y-coordinate and consists of three parts. The bottom pad of the via on wiring layer z_1 is $[x_1 - o_W, x_1 + o_E] \times [y_1 - o_S, y_1 + o_N] \times \{z_1\} \subseteq A_0$, where $s_z^{\text{bot}} := (o_W, o_E, o_S, o_N)$. The shapes of the stem and the top pad are defined similarly; see Figure 2.2 for illustration. Vias are undesired for several reasons, including their high electrical resistance and impact on manufacturing yield.

The *wiring* $W(N)$ of a net $N \in \mathcal{N}$ is the set of all segments assigned to that net. Clearly, a stick-figure connected set of wiring results in an electrically connected path. The wiring of the chip is the union of the wiring of its nets.

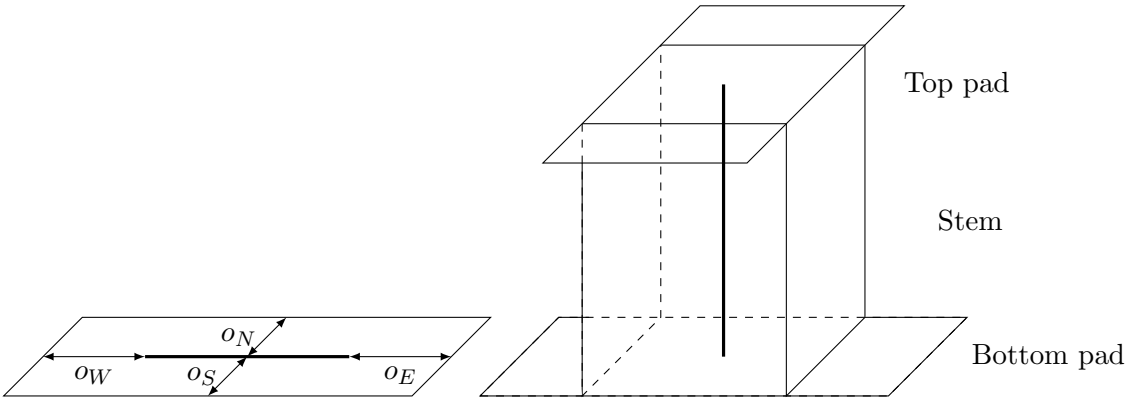


Figure 2.2: A wire model (left) and a via model (right). The corresponding stick-figure is depicted by a bold line.

2.2.2 Design Constraints

The instance of the routing problem contains various kinds of restrictions and guidance to routing. We distinguish between blockages which have to be avoided, design rules which have to be obeyed by routing, and constraints which help to complete the routing task in practice.

Parts of the chips which are not usable for wiring are modeled as a set of (rectangular) *blockages*. The segments of a net have to obey these blockages. There are different types of blockages: macros which can be large blocks of logic units, book blockages, and power rails. As the complete wiring of a chip is usually not done at once, but in several steps, some wires might already exist in the input of the routing task. Often, *pre-wiring* is not allowed to be changed. This is particularly applied in one of the last stages in the VLSI design process in which small modifications in the netlist are necessary (*engineering change order, ECO*). Here, it is desired that as few segments as possible change their layout. Pre-wires of a specific net serve as blockages for all other nets but can be used to close connections of the same net. Moreover, some user-defined blockages (so-called *reserved areas*) belong to the set of blockages.

Manufacturing process related *design rules*, defined for each routing layer separately, have to be respected by routing. Design rules have three main goals: first, required design rules must ensure that constraints of the manufacturing process are respected. They particularly define when shapes are connected and separated. Second, they are used to avoid that a signal is affected by another nearby signal (so-called *cross-talk*). Additionally, there are specific wiretypes to prevent cross-talk, so-called *shielded* or *isolated* wiretypes. Third, recommended design rules define additional restrictions to the layout of a design to decrease the failure probability of a chip further, i.e. to improve the expected manufacturing yield. We describe some important design rules in more detail in Section 5.3.2.

Each wiring layer is usually assigned a *preference direction* to efficiently use the routing space. In this work we restrict ourselves to horizontal and vertical directions. Consecutive wiring layers have different preference directions in practice, although this is neither important from theoretical perspective, nor from an implementation point of view. A *jog* is a wire segment running orthogonally to the preference direction within a wiring layer. Jogs may be necessary to close connections but they may block many wires in preference direction and should therefore be largely avoided.

A net must connect source and sink pins by a network which does not necessarily have to be a tree (McCoy and Robins [1994], Kahng, Liu and Mandoiu [2002]). Nevertheless, for timing analysis and in terms of minimizing net capacitance we assume that every net is connected by segments which form a Steiner tree.

Some further design specific constraints may be imposed to the routing task. Due to timing analysis some nets of the netlist are considered to be more critical than others. These nets can be assigned timing and capacitance constraints.

2.2.3 Optimization Goals

The optimization goal of the VLSI routing problem for a specific chip correlates with that of the whole design process of a chip and very much depends on the purpose of the chip. There are three main goals: meeting all timing constraints, minimizing power consumption and maximizing manufacturing yield. In practice, more than one objective is desirable at the same time. Hence, one has to find a good (weighted) trade-off between conflicting goals.

For processor chips and ASICs (*application specific integrated circuits*) the overall goal is usually given by the *maximum clock speed*. This objective is mainly addressed by logic synthesis and clock scheduling in previous design steps. The remaining task in routing is to construct a wiring which realizes the required timing. That is, the most timing-critical nets should be routed almost optimally, while other nets should be routed with minimum capacitance. In practice, most of the nets are not timing-critical. Applications for this can be found e.g. in servers, printers, DVD recorders and gaming.

Another important design criterion is *power consumption*, loosely speaking, it can be viewed as the (weighted) sum over all capacitances on the chip. As all circuits have been fixed before routing, the optimization goal in routing is to minimize the capacitance of the wiring of the chip. Chips with a low power consumption are needed, particularly, where the operation time for that chip is crucial, such as in battery-powered devises (mobile phones, microprocessors).

The last optimization goal is *manufacturing cost*. Decisions on many factors having an effect on this goal have already been made before routing (e.g. chip size, number of routing planes). But the wiring has a considerable impact on the *production yield* (yield for short) of a chip. Informally expressed, yield is the proportion of chips without any defect on the wafer. It is influenced by several components (see Section 5.3.2). During routing, this can

be taken into consideration in many ways, for example by spreading wires, decreasing the number of vias and avoiding crosstalk; see Section 5.3.3.

2.2.4 Problem Formulation

The task of VLSI routing problem can now be formulated as follows:

VLSI Routing Problem	
Instance:	• a netlist $\mathcal{N}$; • for each net $N \in \mathcal{N}$ a set $\mathcal{A}_N$ of wiretyped routing areas; • a set of blockages; • a set of design rules; • a set of design specific constraints; • an optimization goal.
Task:	For each net $N \in \mathcal{N}$, find a set of segments within $\mathcal{A}_N$ which electrically connects its pins and is separated from blockages and shapes of other nets such that all design rules and constraints are met and the goal is optimized.

2.3 Routing Grid

In order to simplify the routing process, especially in 90 nm and older technologies, all metal shapes, such as pins and blockages, are well aligned to a layer-dependent, predefined grid. The distance between adjacent tracks in the grid, often called the *wiring pitch*, is usually the minimum width of a wire plus the minimum distance of two wires on that layer. Therefore, libraries of those technologies are also called *gridded*. Wiring can be restricted to follow these tracks of the grid without sacrificing routing space. For this reason, most industrial routing tools make use of that property and create a fully *on-grid* wiring.

The graph typically used for modeling the search space for VLSI routing is a three-dimensional grid graph. Let $G_0 = (V_0, E_0)$ be the infinite three-dimensional grid graph with vertex set $V_0 := \mathbb{Z}^3$, in which two nodes $(x, y, z) \in V_0$ and $(x', y', z') \in V_0$ are joined by an edge if and only if $|x - x'| + |y - y'| + |z - z'| = 1$.

As mentioned in the previous section, going from one wiring layer to an adjacent wiring layer and also going orthogonally to the preference direction within one wiring layer is costly for routing. This is typically modeled by edge lengths $c : E(G_0) \to \mathbb{Z}_{>0}$ that prefer edges within one wiring layer in preference direction: for each wiring layer $z \in [z_{\min}, z_{\max}]$ there are constants $c_{z,1}, c_{z,2}, c_z \in \mathbb{Z}_{>0}$ such that for all $(x, y) \in [x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$

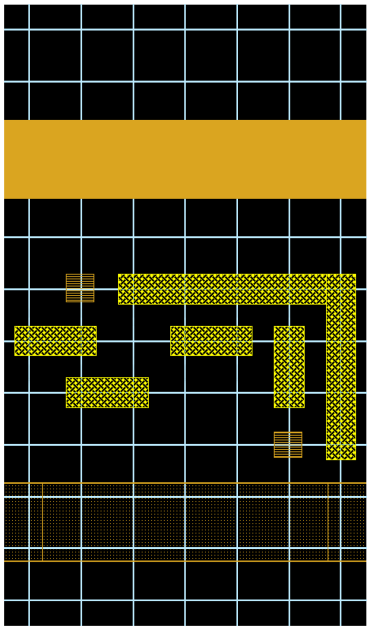
$$\begin{aligned} c(((x, y, z), (x + 1, y, z))) &= c_{z,1}, \\ c(((x, y, z), (x, y + 1, z))) &= c_{z,2} \quad \text{and} \\ c(((x, y, z), (x, y, z + 1))) &= c_z \quad (z < z_{\max}), \end{aligned}$$

i.e., in each wiring layer defined by some fixed z-coordinate there are fixed lengths for edges in x- and y-direction and there is a fixed length for all edges leading to the wiring layer above.

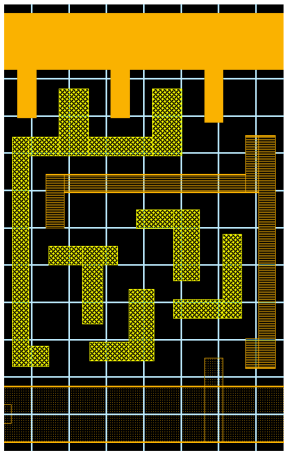
Typical values we use in practice and throughout this work are 1 and 4 for edges within one wiring layer in and orthogonal to the preference direction, and 13 for vias.

In newer technologies, such as 65 nm and beyond, several reasons make routing much more difficult. Routing tools are faced with shapes which do not have that grid property any longer. They must follow complex variable rules in width and spacing which force the wiring to become *off-grid*. Moreover, the structure of power and blockages, especially those inside circuits, is much more complex than for older technologies, thus even pin access becomes difficult.

For an example of comparable circuits of a gridded and gridless library, see Figure 2.3.



90 nm technology (gridded)



65 nm technology (gridless)

Figure 2.3: The layout of a circuit (NAND4) in a gridded (left) and a gridless (right) 12-track library with pins (yellow), blockages (orange, striped), supply voltage (orange, filled) and ground (orange, dotted). For the gridded library, all shapes are simple, lie on pre-defined tracks (white) and have a symmetric overhang with respect to the tracks. This is in contrast to the circuit of the gridless library. There, the structure is composed of many non-trivial shapes with non-regular positions relatively to the grid. Note that the castellated power supply terminals on the top additionally contribute to a difficult pin access.

For gridless instances, routing tools cannot afford to work on the manufacturing grid, the smallest geometry manufacturable by a fab. Standard grid-based routing tools (where

all wires are assigned on pre-defined tracks) make use of a regular detailed grid graph as described above. Although searching for paths on a uniform grid graph is much easier, it may waste space available for routing. Our detailed routing tool BonnRouteLocal uses a very efficient data structure to represent all shapes which is able to answer queries extremely fast (Hetzel [1995], Panten [2005]). Although path search utilizes an underlying grid structure, also given in the example of Figure 2.3, it is not restricted to grid-based routing algorithm. Each shape is associated with one or more nodes of the routing grid which represent areas containing the shape.

2.4 Complexity

The VLSI Routing Problem contains the following simplified problem which can be formulated in terms of a network problem:

Simplified VLSI Routing Problem	
Instance:	• The infinite three-dimensional space $\mathbb{Z}^3$; • a netlist $\mathcal{N}$, each net $N \in \mathcal{N}$ consists of a set of terminals $T(N) \subset \mathbb{Z}^3$;
Task:	For each net $N \in \mathcal{N}$, find a Steiner tree which connects the terminal set $T(N)$ in $\mathbb{Z}^3$ such that the total netlength $l(\mathcal{N}) := \sum_{N \in \mathcal{N}} \sum_{e \in W(N)} l(e)$ is minimized.

The Simplified VLSI Routing Problem includes many *NP*-hard combinatorial optimization problems, e.g. for $|T(N)| = 2$ for all $N \in \mathcal{N}$ the vertex-disjoint paths problem in undirected grid graphs (Kramer and van Leeuwen [1984]), and for $|\mathcal{N}| = 1$ and all terminals lying in one layer the rectilinear Steiner tree problem (Garey and Johnson [1977]). Hence, the Simplified VLSI Routing Problem is also an *NP*-hard problem.

2.5 Routing Approaches

Many routing approaches in integrated circuit design systems have been developed during the last 40 years. First papers described wiring programs for printed circuit boards (Dunne [1967], Fisk, Caskey and West [1967], Heiss [1968]). The main goal was to interconnect a given set of terminals in a grid of a few hundred channels on two layers. At about the same time, the theory on shortest path algorithms received a lot of attention in literature (cf. Section 4.1), which in turn improved routing in practice of circuit designs considerably. On the other hand, some theoretical results were motivated by applications in circuit layout; for example Mikami and Tabuchi [1968] proposed a line search algorithm, and the concept of line expansion was first described by Heyns, Sansen and Beke [1980].

Due to the huge instance size of today's VLSI chips, most routing tools consist of at least two main parts: *global routing* and *detailed routing*. In global routing, each net is assigned a *global routing corridor* in which it is realized in detailed routing. As global routing works

on a much coarser instance than detailed routing, it runs much faster than the latter and is able to globally optimize a given optimization goal such as minimizing netlength or maximizing yield. The task of detailed routing is to electrically connect each net of the chip within the global routing corridor.

There are mainly two approaches for global routing. Sequential algorithms route the nets one by one in a maze running fashion (based on Lee [1961]), with a line search algorithm (Mikami and Tabuchi [1968], Hightower [1969], Soukup [1978]) or line expansion technique (Heyns, Sansen and Beke [1980]). Multicommodity flow-based algorithms model the global routing problem as a multi-terminal multicommodity flow problem, which is solved approximately in several iterations (Carden IV, Li and Cheng [1996], Albrecht [2001]). Today's global routing algorithms are mostly congestion and performance driven (Vygen [2004], Müller [2006]).

Most detailed routing programs split the task to connect a net in a sequence of shortest path searches for which standard algorithms and speed-up techniques (e.g. goal-oriented or bi-directional) are applied; see Section 4.1.2. Two different approaches are commonly used in practice: in a cell-based approach, the global routing corridor is partitioned into a sequence of small areas (cells) within local connections are realized. The overall path is searched from one cell to another. These cells may form a channel expanding only a few tracks, called channel routing (e.g. Hashimoto and Stevens [1971], Rivest and Fiduccia [1982], Tseng and Sechen [1999]), or consist of only a small cell with connection points on all sides, called switch-box routing (e.g. Hitchcock [1969], Hamachi and Ousterhout [1984], Huijbregts and Jess [1993]). In a point-to-point approach, the connection is searched without breaking the task into multiple connections. Various methods are applied in practice. Margarino et al. [1987] presented the idea of expanding tiles which is similar to the line expansion technique (see Section 4.1.2). Tseng and Sechen [1999] applied an improved version in their channel based router. Hetzel [1998] developed a goal-oriented and interval-based version of Dijkstra's algorithm which has been used in XRouter, the predecessor routing tool of BonnRoute, used within IBM for many years. Main parts of the current path search in BonnRoute is still based on his algorithm, see Section 2.6.2. Zheng, Lim and Iyengar [1996] restricted the search space to an implicitly represented sparse strong connection graph which is part of the Hanan grid induced by the boundaries of all obstacles. Following their framework, Cong, Fang and Khoo [2001] and Li, Chen and Lin [2007] presented combined approaches of a connection graph based router with tile-expansion.

Although path search plays a dominant role in constructing the wiring of a net, a few routing programs apply multi-terminal routing. Huijbregts and Jess [1993] propose an algorithm which routes multi-terminal nets without partitioning them into 2-terminal nets. They show that their algorithm constructs minimum cost paths.

In order to handle huge instance sizes, some routing engines use intermediate steps or apply a multi-level routing system. In track assignment (Kay and Rutenbar [2001], Batterywala et al. [2002]), long routes are embedded within their global routing corridor, i.e. their ordering is fixed within tiles spanning a very few tracks. The goal of this approach is to

address problems arising from signal delay, cross-talk and other constraints at moderate computational complexity. For easy chips, track routing can reduce the running time of detailed routing significantly. For complex and dense chips, this approach often leads to a tremendous increase in rip-up and reroute sequences to withdraw and fix wrong decisions made in track assignment. Instead of a track assignment step, Cong, Fang and Khoo [2001] add a congestion driven wire-planning stage between global and detailed routing that plans the route of each net based on available routing resources and individual requirements of that net. In the last few years, several multilevel routing systems have been proposed (Ho et al. [2003], Cong et al. [2005], Chen, Chang and Lin [2006]). They include a coarsening and an uncoarsening stage. Starting with a fine tile partitioning of the entire chip, the coarsening stage iteratively merges tiles. At each level, it estimates routing resources. At the coarsest level, an initial route is constructed (e.g. by a multicommodity flow algorithm). Some multi-level routing engines also perform a track assignment at that level. The uncoarsening pass moves from a global to a local view. At each level, tile-to-tile-paths are searched and results of the previous level are refined. At the finest level, a final path search finds the exact connections within each tile. In contrast, Chen, Chang and Lin [2006] proposed a reversed flow, i.e. first an uncoarsening stage followed by a coarsening stage.

For many years, the classical optimization goals of routing algorithms were netlength and number of vias. With decreasing feature size, wire spacing has become significant for yield, power consumption and timing of a chip, and is taken into account by the work of Huijbregts, Xue and Jess [1995], Tseng, Scheffer and Sechen [2001], and Müller [2006]. Moreover, all modern routing systems provide a good rip-up and reroute strategy to revise decisions already made in an earlier stage of the routing algorithm.

Finally, we would like to remark that some work has also been spent on diagonal wiring which was already mentioned in a paper by Heiss [1968]. The use of 45°-segments for routing is reported, for example, in Lodi [1988], Chiang and Sarrafzadeh [1991], Natarajan et al. [1992] and Ho et al. [2005]. For 45°-routing, Teig [2002] introduced the term X-architecture. Analogously, a Y-Architecture allowing wires routed in three directions (0°, 120° and 240°) was proposed by Chen et al. [2003a] with an analysis presented in Chen et al. [2003b]. The practical relevance of both architectures is unclear. All current real-word VLSI chips allow wiring for only two directions (horizontal and vertical), which will certainly remain the approach for the next years.

2.6 Some Key Components of BonnRoute

In this section we focus on key algorithmic components of BonnRoute, the routing tool developed at the Research Institute for Discrete Mathematics at the University of Bonn. Computational results of BonnRoute achieved on modern industrial chips are presented in Chapter 5.

In contrast to some of today's industrial routers, BonnRoute does not follow a hierarchical approach and does not have the intermediate step of track assignment, mainly for three

reasons: BonnRouteGlobal contains a very accurate capacity estimation providing a guidance for detailed routing which is well solvable without resorting to large safety margins in routing capacities. Moreover, it is able to incorporate timing and other constraints into its optimization goal. Finally, the core routine of BonnRouteLocal, the shortest path algorithm, is extremely fast.

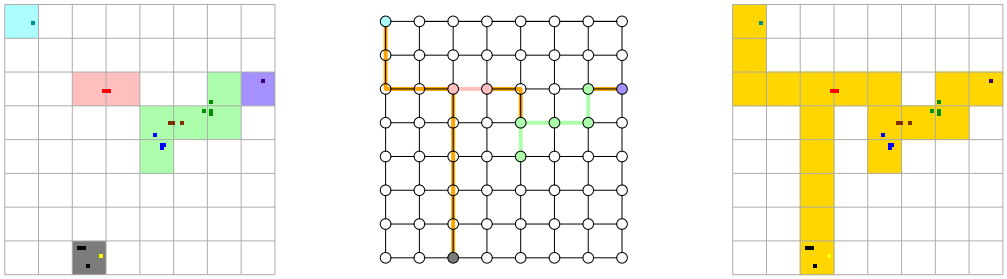
For very complex and dense chips it may be helpful to split the two-stage flow of global and detailed routing further. Short nets occupy routing capacities on lower layers and may block wiring resources of long-distance nets without being seen by global routing. Müller [2002] implemented a very accurate capacity estimation into BonnRouteGlobal which is able to respect capacity of blockages and pre-wiring very well. He proposed to pre-route short nets in a separate step prior to global routing and showed that the new approach reduces the number of overflows and rip-ups drastically. Moreover, running time of the overall flow can be saved by up to 60 %.

2.6.1 Global Routing

As already explained in the previous section, for every desired electrical connection the global routing phase determines a three-dimensional subgraph G of G_0 in which the connection has to be realized. The main tasks of global routing are elimination of congestion and timing problems on a global level and providing corridors for detailed routing as a guidance while optimizing the overall goal. In this section we describe BonnRouteGlobal, the global routing program which is part of BonnRoute.

The chip area $[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$ is partitioned into a set $\mathcal{R}$ of axis-parallel rectangular regions. Each such region spans about 50–100 channels in horizontal and vertical direction. A global routing *tile* is a pair of a region $R \in \mathcal{R}$ and a wiring layer z with $z_{\min} \leq z \leq z_{\max}$ and is associated with a node of the global routing grid graph $G = (V(G), E(G))$. Two nodes of G are joined by an edge if and only if for their corresponding tiles (R_1, z_1), (R_2, z_2) hold: if there exist $r_1 \in R_1$ and $r_2 \in R_2$ with $(r_1, r_2) \in E(G_0)$, then $(R_1, R_2) \in E(G)$. For each net N of the netlist $\mathcal{N}$, a global routing net (*gnet*) is defined as follows: Define $P(N)$ to be the set of pins of net N. For a electrically connected component $c \subseteq P(N) \cup W(N)$ let $t(c)$ be a minimal set of tiles covering all shapes of c in wiring layers. A partition of $t(c)$ into a maximum number of pairwise non-overlapping sets build the set of global pins (*gpins*). Every gpin corresponds to a set of nodes in G, which form a multi-terminal. For each net N, global routing builds a Steiner tree $S(N)$ in G connecting multi-terminals which correspond to gpins of N. The set of nodes $V(S(N)) \subseteq V(G)$ correspond to a set of tiles which form the global routing area for net N. Some sophisticated post-processing is necessary to support detailed routing, for example add neighbored wiring layers to allow detailed routing to change layers. See Figure 2.4 for a two-dimensional illustration.

The task of global routing is to construct Steiner trees in G such that the detailed router can connect all nets within that area simultaneously and the overall goal is optimized. In order to solve the global routing problem we need to formulate a graph-theoretical problem.



(a) Pins and gpins (b) gnet (c) Global routing area

Figure 2.4: For a net with eight pins (depicted in different colors), five gpins are created (a). In (b), a Steiner tree connects the five corresponding multi-terminals in G which results in a global routing area as shown in (c).

For this, G is a capacitated and weighted undirected grid graph $G = (V(G), E(G), u, l)$, where $u : E(G) \to \mathbb{R}_{\geq 0}$ and $l : E(G) \to \mathbb{R}_{\geq 0}$. The length $l(e)$ of an edge $e = \{v, w\} \in E(G)$ is the distance of the midpoints of the tiles corresponding to v and w with respect to the unit edge lengths c as defined in Section 2.3.

After partitioning the chip area into tiles, $V(G)$, $E(G)$ and l are determined. The first important problem to set up the global routing task is to compute the edge capacities of $E(G)$. The capacity $u(e)$ of an edge $e = \{v, w\} \in E(G)$ is an estimation how many detailed wires of unit length can be routed between $t(v)$ and $t(w)$. It also has to consider blockages and resources of nets whose pins lie within one tile only. Capacity estimation is crucial for the quality of global routing. It can be considered as a vertex-disjoint path problem. There is a commodity for each wiring layer. The paths are allowed to use resources of the same layer in preference wiring direction and in adjacent layers in the orthogonal wiring direction. Then the capacity $u(\{v, w\})$ is the number of vertex disjoint paths between tiles $t(v)$ and $t(w)$ in a solution which maximizes the total number of vertex-disjoint paths over all commodities.

Solving each commodity independently with a maximum flow algorithm is far too slow and too optimistic. For example, on a chip with about one billion paths an implementation of the Goldberg-Tarjan-algorithm takes more than a week. For the same chip, BonnRouteGlobal computes a set of vertex-disjoint paths by a new and extremely fast multicommodity flow heuristic in five minutes (Müller [2002]). The basic idea is to perform an augmenting path algorithm which exploits the special structure of the instance. Here, a bit pattern based path search is performed where bit vectors encode blockages and flows can be found by a sequence of logical operations on bit vectors. Each augmenting path requires only $O(k)$ constant-time bit pattern operations, where k is the number of edges orthogonal to the preferred wiring direction of the corresponding layer. The capacity estimation finds a feasible integral multicommodity flow solution whose value differs only by about 10 % from the (weak) upper bound derived from the maximum-flow algorithm. This bit pattern based and far better approach for capacity estimation allows to pre-route short nets which lie within one tile only (Müller [2002]).

After capacity estimation, the global routing instance is fully specified. The global routing task can now be expressed in graph-theoretical terms:

VLSI Global Routing Problem	
Instance:	• A global routing graph G with edge capacities and edge lengths; • a set $\mathcal{N}$ of nets; • for each net $N \in \mathcal{N}$ a wiretype area $\mathcal{A}_N$; • a set of design rules; • a set of design specific constraints; • an optimization goal.
Task:	For each net $N \in \mathcal{N}$, find a Steiner tree in G such that the edge capacities are respected, the design rules and constraints are met and the overall goal is optimized.

The design rules imposed to the global routing task do not cover most of the design rules specified by the manufacturing process for the VLSI Global Routing Problem as they cannot be respected by global routing. However, some of design rules, such as minimum space restrictions, can be taken into account by BonnRouteGlobal. The set of design specific constraints may cover rules to respect timing and capacitance constraints or to increase yield. They can be charged to nets individually or to groups of nets which is shown in the formulation of the derived mathematical problem below.

Basically, the global routing problem amounts to a Steiner tree packing problem in a graph with edge capacities. A simplified version can be stated as follows: For each net $N \in \mathcal{N}$, let $\mathcal{Y}_N$ be a set of feasible Steiner trees for N in G, and $w(e, N) \in \mathbb{R}_{>0}$ the maximum width of net N on edge $e \in E(G)$. The function w is derived from the wiretyped routing areas $\mathcal{A}_N$, which allows to set wiretypes depending on planes and regions. An integer programming formulation for the simplified VLSI Global Routing Problem is as follows:

$$
\begin{aligned}
\min \quad & \sum_{N \in \mathcal{N}} \sum_{e \in E(G)} \Big(l(e) \sum_{Y \in \mathcal{Y}_N : e \in E(Y)} x_{N,Y} \Big) \\
\text{s.t.} \quad & \sum_{N \in \mathcal{N}} \sum_{Y \in \mathcal{Y}_N : e \in E(Y)} \omega(N,e) x_{N,Y} \leq u(e) \qquad \text{for all } e \in E(G) \\
& \sum_{Y \in \mathcal{Y}_N} x_{N,Y} = 1 \qquad \text{for all } N \in \mathcal{N} \\
& x_{N,Y} \in \{0,1\} \qquad \text{for all } N \in \mathcal{N}, Y \in \mathcal{Y}_N
\end{aligned}
\tag{2.1}
$$

For a net N, the set $\mathcal{Y}_N$ of feasible Steiner trees can be obtained by a Steiner tree algorithm which construct feasible trees for the given application. In Chapter 3 we present two Steiner tree algorithms which can be used to build feasible Steiner trees for $\mathcal{Y}_N$.

A Steiner tree $Y \in \mathcal{Y}_N$ is chosen for net N if and only if the decision variable $x_{N,Y}$ is 1. If each net consists of only two pins, $\mathcal{Y}_N$ contains all possible paths in G connecting both pins

of N, $u \equiv 1$, $w \equiv 1$ and $l \equiv 0$, the global routing problem reduces to the undirected edge-disjoint paths problem. As the undirected edge-disjoint paths problem is *NP*-complete, even in many special cases (Vygen [1995], Nishizeki, Vygen and Zhou [2001], Marx [2004]), the decision version of the simplified version of the VLSI Global Routing Problem is NP-complete. The fractional relaxation of the above program (allowing $x_{N,Y} \in [0,1]$ for all $N \in \mathcal{N}, Y \in \mathcal{Y}_N$) results in the undirected multicommodity flow problem. This can be solved in polynomial time by means of linear programming. As the LP formulation is rather huge for today's instance sizes, it is desirable to apply an efficient combinatorial algorithm to solve the problem approximately. The first fully polynomial approximation scheme for the multicommodity problem was developed by Shahrokhi and Matula [1990], while Carden IV, Li and Cheng [1996] first applied this approach to global routing. An initial implementation in BonnRouteGlobal is due to Albrecht [2001] who adapted an approximation algorithm by Garg and Könemann [1998] to global routing.

The simplified description (2.1) does not consider objective functions other than netlength. In particular, it is not able to take timing, power or yield into account. For example, coupling capacitance could be ignored in older technologies, whereas it becomes increasingly important with new technologies. A first extended formulation taking space-dependent costs into account was introduced by Vygen [2004]. He proposed a global routing algorithm which finds a solution arbitrarily close to the optimum. We briefly give the main idea.

For a net $N \in \mathcal{N}$ on edge $e \in E(G)$ there is a maximum capacitance cost $l(e,N)$ which is attained at minimum space to neighboring shapes on both sides. We assume that it reduces linearly by an amount of at most $v(e,N)$ units of coupling capacitance with increasing extra space until an extra maximum spacing $s(e,N) \in \mathbb{R}_{\geq 0}$ is reached. Note that coupling capacitance does not depend linearly on the spacing between two shapes in practice. Nevertheless, this simplification gives very good results in BonnRoute. Let $y_{e,N} \in [0,1]$ denote the fraction of possible extra space assigned to net $N \in \mathcal{N}$ on edge $e \in E(G)$. Then, the space requirement of net N on edge $e \in E(Y_N)$ ($Y_N \in \mathcal{Y}_N$) is $w(e,N) + y_{e,N}s(e,N)$, resulting in $l(e,N) - y_{e,N}v(e,N)$ units of capacitance consumed.

Moreover, it is possible to bound the capacitance used by subsets of $\mathcal{N}$. This can, for example, be used to restrict capacitance along timing-critical paths. Let $\mathcal{M}$ a family of subsets of $\mathcal{N}$ with $\mathcal{N} \in \mathcal{M}$, and $U : \mathcal{M} \to \mathbb{R}_{\geq 0}$ the capacitance bound for family $\mathcal{M}$. Further, we can specify weights $c(M,N) \in \mathbb{R}_{\geq 0}$ for $N \in M \in \mathcal{M}$. With this notation and enhancements, the global routing problem can now be formulated as follows: for each net $N \in \mathcal{N}$, find a Steiner Tree $Y_N \in \mathcal{Y}_N$ and numbers $y_{e,N} \in [0,1]$ for each $e \in E(Y_N)$, which

$$\text{minimize} \quad \sum_{N \in \mathcal{N}} c(\mathcal{N},N) \sum_{e \in E(Y_N)} (l(e,N) - y_{e,N}v(e,N))$$

$$\begin{aligned} \text{s.t.} \quad \sum_{N \in \mathcal{N}: e \in E(Y_N)} (w(e,N) + y_{e,N}s(e,N)) &\leq u(e) \quad \text{for all } e \in E(G) \\ \sum_{N \in M} c(M,N) \sum_{e \in E(Y_N)} (l(e,N) - y_{e,N}v(e,N)) &\leq U(M) \quad \text{for all } M \in \mathcal{M} \end{aligned}$$

The objective function above minimizes the overall capacitance, i.e., the power consumption of the chip. This integer program can be reformulated whose relaxation then results to the following linear program:

$$\begin{aligned}
&\min\ \lambda \\
&\text{s.t.} \\
&\sum_{Y\in\mathcal{Y}_N} x_{N,Y} = 1 && \text{for all } N\in\mathcal{N} \\
&\sum_{N\in M} c(M,N)\left(\sum_{Y\in\mathcal{Y}_N}\sum_{e\in E(Y)} l(e,N)x_{N,Y} - \sum_{e\in E(G)} v(e,N)y_{e,N}\right) \le \lambda U(M) && \text{for all } M\in\mathcal{M} \\
&\sum_{N\in\mathcal{N}}\left(\sum_{Y\in\mathcal{Y}_N:e\in E(Y)} w(e,N)x_{N,Y} + s(e,N)y_{e,N}\right) \le \lambda u(e) && \text{for all } e\in E(G) \\
&\sum_{Y\in\mathcal{Y}_N:e\in E(Y)} x_{N,Y} \ge y_{e,N} \ge 0 && \text{for all } e\in E(G), N\in\mathcal{N} \\
&\sum_{Y\in\mathcal{Y}_N:e\in E(Y)} x_{N,Y} \ge y_{e,N} \ge 0 && \text{for all } e\in E(G), N\in\mathcal{N} \\
&x_{N,Y} \ge 0 && \text{for all } N\in\mathcal{N}, Y\in\mathcal{Y}_N
\end{aligned} \tag{2.2}$$

Vygen [2004] developed a fully polynomial approximation scheme for this linear program and its dual. This algorithm always gives a fractional dual solution.

Theorem 2.1 (Vygen [2004])**.** *Given an approximation parameter ϵ_0, one can find $\epsilon, \epsilon_1, \epsilon_2 \in \mathbb{R}_{>0}$ and $t \in O\left(\frac{\log(|E(G)|+|\mathcal{M}|)}{\epsilon_0^2}\right)$ such that with parameters $\epsilon, \epsilon_1, \epsilon_2$ and t, a $(1+\epsilon_0)$-optimal feasible solution to the global routing LP* (2.2) *can be computed.*

The objective function of the above description is minimizing power consumption formulated in terms of capacitance. Recently, Müller [2006] showed how to use Vygen's algorithm to optimize manufacturing yield. The main idea is to replace capacitance by costs representing the sensitivity of the layout to random defects.

After solving the linear program, randomized rounding based on methods by Raghavan and Thompson [1987] is applied. Vygen [2004] proved that, after rounding the rational solution, the maximum integral violation λ can be bounded. The small integrality gap together with a feasible solution of the dual LP provides a certificate for feasibility of the instance.

Finally, rip-up and reroute is applied to fix problems caused by randomized rounding. Running time of critical parts of BonnRouteGlobal can efficiently be parallelized. Müller [2006] gives a parallelized version of the approximation algorithm by Vygen, which scales very well with the number of processors.

Saxena, Shelar and Sapatnekar [2007] published a comprehensive book on estimation and optimization of congestion in VLSI routing. For more detailed information on theoretical aspects as well as computational results of BonnRouteGlobal we refer to Müller [2002], Vygen [2004] and Müller [2006].

2.6.2 Detailed Routing

The task of detailed routing is to realize desired connections of a net within the corridors computed by global routing before, respecting all given constraints.

The VLSI Detailed Routing Problem equals the VLSI Routing Problem under the additional constraint, that the routing area for each net $N \in \mathcal{N}$ is restricted to a set $\mathcal{A}_N$ of wiretyped global routing corridors.

Note that our formulation primarily requires to find a feasible rather than optimizes a given optimization goal. The primary task of the VLSI Detailed Routing Problem is to determine a feasible layout of the metal realizations of all nets. Of course, objectives such as netlength and number of vias are taken into account. We assume that all optimization is already done in previous steps. For example, timing-critical nets are assigned wiretypes with a sufficiently large extra spacing requirement, and restricted sets of layers they should be routed on. In global routing, it is possible to incorporate a optimization goals. Further, we suppose that all timing constraints set up for global routing leave some margin such that timing specification for the chip are unlikely to be violated by detailed routing — assuming that the wiring is composed of shortest paths and the area of the global router is respected by the detailed router. Some post-processing can be applied after connecting all nets to increase manufacturing yield (Schulte [2006], Bickford et al. [2006]).

Due to the huge instance sizes of detailed routing, i.e. millions of vertex-disjoint Steiner trees to be found in a graph with billions of nodes, we can not afford to determine all nets simultaneously. Therefore, we route the nets one at a time. Each net has to obey distance rules to shapes of blockages and pins, and to shapes of already routed nets by obeying distance rules. We also restrict paths to corridors computed by global routing. This information is essential for detailed routing for two reasons: first, capacity estimation in global routing assures that all paths can be realized within the computed corridors. Second, restricting to only a small fraction of the entire chip area speeds up path search tremendously.

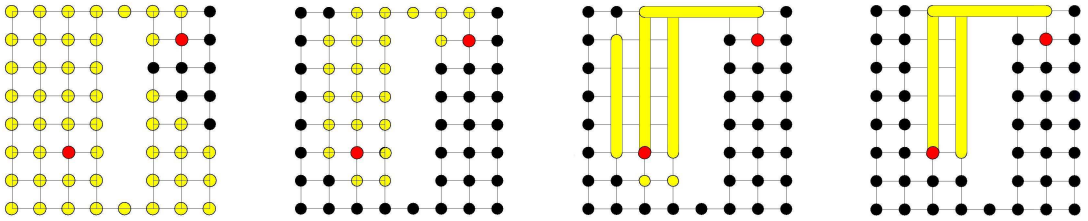
Since global routing more or less specifies a tree structure for each net, in practice sufficient quality is attained by composing Steiner trees from paths in detailed routing.

So the key component of BonnRouteLocal is its path search, which contains two important ideas allowing to find millions of paths in only a few hours: it is goal oriented and interval based approach. With the help of a *future cost* estimate, which is a lower bound on the distance from a set of nodes to a set of targets, it is possible to guide path search towards the target. The second factor used in the path search is the way in which we store distance information. In contrast to the original Dijkstra's algorithm which labels individual nodes, we store consecutive nodes in preference wiring direction in intervals. We combine nodes if they are equal with respect to certain properties, and if their distance to the target can be expressed very efficiently. Our path search then labels intervals instead of individual nodes. The following theorem shows that this can be done in a time complexity which depends linearly on the number of intervals. Note that the number of intervals is typically about 25 times smaller than the number of nodes:

Theorem 2.2 (Hetzel [1998]). *The running time of path search is $O((d+1)I \log I)$, where d is the detour (actual path length minus lower bound) from the source to the target, and I is the number of intervals.*

In Chapter 4 we give a generalization of Dijkstra's algorithm which allows to compute a better future cost estimate for detailed routing as well as to generalize Hetzel's algorithm.

Figure 2.5 is a very simplified comparison of path search algorithms without and with future cost, both based on nodes and intervals. This example indicates that the goal-oriented interval based path search performs the smallest number of labels.



(a) node based (b) goal-oriented node based (c) interval based (d) goal-oriented interval based

Figure 2.5: Four different methods to find a shortest path from the red vertex on the bottom left to the red vertex on the upper right part. Labeled points or intervals are depicted yellow. In all cases, the running time is roughly proportional to the labeled nodes (50 versus 24) or intervals (7 versus 4).

Like most routing tools, BonnRouteLocal also contains a rip-up and reroute strategy to overcome blockages caused by already realized wire segments. When routing a net which can only be connected by removing wire segments of neighbored nets, our rip-up and reroute algorithm determines a set of wires to be removed in order to close the connection. After that, disconnected connections are closed again by finding an alternative route if possible. Hetzel [1998] presents an efficient rip-up and reroute algorithm by extending an algorithm by Raith and Bartholomeus [1991].

Although the above mentioned components (global routing corridors and goal oriented, interval based path search) are absolutely necessary to handle today's large chips instances, some sophisticated features have been shown to be useful to run detailed routing successfully. At this point we would like to name only two strategies: The ordering in which the nets are routed is crucial for routability and running time. Weighted (mostly timing-critical) nets are routed first to ensure that they are connected shortest possible to avoid detours due to other wires. Nets assigned a wide wiretype are also preferred over other nets as it is more difficult to connect them with increasing wiring density. Among all nets with equal weight and wiretype properties, those with pins which are difficult to access are routed before nets with easily accessible shapes. The second strategy is to restrict the routing area of a net with more than two pins further such that it guides the path search to label only the part of the global routing area which contains the closest target.

A multi-threaded implementation for BonnRouteLocal is described in Panten [2005]. For more theoretical as well as practical details on BonnRouteLocal we refer to Hetzel [1995] and Rohe [2001].

Chapter 3

Minimum Steiner Tree Algorithms

The rectilinear Steiner tree problem is a key problem in VLSI layout. It appears as a subproblem in several applications, e.g. in inverter tree and clock tree algorithms as well as in global and detailed routing. Due to technological constraints on the orientation of wires, interconnections in VLSI design usually are rectilinear Steiner trees. To handle the Steiner tree problem efficiently in practice, it is almost always mapped to a two–dimensional problem. Although this simplification is clearly a restriction, its solution is sufficient for most algorithms working with Steiner trees.

In the problem we are given a finite set of (electrical) *terminals*, assumed to be a set of points Z in the plane. A rectilinear Steiner tree in the plane is a tree that interconnects a given set of points using only horizontal and vertical line segments. The line segments of the tree are denoted *edges*. Edges meet only at *vertices* in the tree and no vertex intersects the interior of an edge. Note that all the terminals are vertices. For a given tree T we let $V(T)$ denote its vertices and $E(T)$ its edges. Rectilinear Steiner trees are assumed to have no overlapping edges, which are clearly sub-optimal with regard to total length. Therefore, every vertex has at most one incident edge in each of the four directions. The *degree* of a vertex is the number of edges it is incident to. A *Steiner point* is a non-terminal vertex of degree three or four, while a *corner point* is a non-terminal vertex of degree two where the two edges meeting at a corner point are perpendicular. Non-terminal vertices of degree two with two colinear incident edges are removed by merging both edges. We assume w.l.o.g. that all interconnections between terminals and/or Steiner points are shortest rectilinear paths and that no two corner points are adjacent in the tree, i.e., staircase connections are not allowed. Thus, interconnections between terminals and/or Steiner points consist of at most two edges. In this chapter, distances are always measured based on the l_1 metric if not otherwise stated. The rectilinear distance between vertices u and v is denoted by $|uv|$, whereas the length $|T|$ of a tree T is the sum of the lengths of all its edges. A shortest rectilinear Steiner tree is called *Steiner minimum tree (SMT)*. The problem to find a rectilinear Steiner minimum tree connecting a given set of terminals in the plane is well-known to be NP-hard (Garey and Johnson [1977]).

The literature on the Steiner tree problem is very comprehensive. For an introduction see, for example, the monographs by Hwang, Richards and Winter [1992], and Prömel and Steger [2002].

In this chapter, we examine two different questions arising in the construction of rectilinear Steiner minimum trees with additional objectives and constraints: first, we consider the problem of finding a rectilinear Steiner minimum tree that — as a secondary objective — minimizes a signal delay related objective. In the second section, we consider the problem of finding a shortest rectilinear Steiner tree in the presence of rectilinear obstacles. The Steiner tree is allowed to run over obstacles; however, if we intersect the Steiner tree with some obstacle, then no connected component of the induced subtree must be longer than a given fixed length. As an example, both algorithms can be applied to build feasible Steiner trees used in the integer programming formulation for global routing; see Section 2.6.1.

Main parts of Sections 3.1 and 3.2 follow Peyer, Zachariasen and Jørgensen [2004], and Müller-Hannemann and Peyer [2003], respectively.

3.1 Minimum Steiner Trees With Secondary Objectives

In this section we discuss the problem to construct a tree — among all shortest-length rectilinear Steiner trees — which minimizes a given objective. We mainly focus on the problem where the objective is defined by the sum of weighted path lengths. We can use those trees in the design flow, for example, as an initial solution for the integer programming formulations (2.1) and (2.2) for the VLSI Global Routing Problem.

This section is organized as follows: after the formulation of the Rectilinear Steiner Tree Problem with Weighted Sum of Path Lengths Secondary Objective (RSTPWP) in Section 3.1.1, we give some basic notation and definitions in Section 3.1.2 which are needed in this section. Structural results for optimal solutions to RSTPWP are presented in Section 3.1.3, and an exact algorithm for solving the problem is presented in Section 3.1.4. In Section 3.1.5 we give a heuristic framework for solving RSTPWP and similar problems, including secondary objectives based on the Elmore delay model. Comprehensive experimental results are finally presented in Section 3.1.6

3.1.1 Problem Formulation

In the problem we consider we assume that a tree is *rooted* at the *source* $r \in Z$, while the remaining terminals in Z are the *sinks*. Thus, the tree is actually a *Steiner arborescence* for Z rooted at the source. An electrical signal should propagate from the source to the sinks via the constructed tree. When constructing the tree, several conflicting objectives must be taken into account. In particular, the following two objectives need to be considered:

- The total length of the tree should be minimized since this reduces area requirements, congestion and power consumption.

- The signal delay from the source to the sinks should be minimized since this reduces the overall clock cycle time.

An optimal solution for the problem that only considers the first objective is a *rectilinear Steiner minimum tree (RSMT)*. This problem has received significant attention in the literature (Hwang, Richards and Winter [1992], Kahng and Robins [1995], Zachariasen [2001a]), and RSMTs of any practical size can be computed quickly (Warme, Winter and Zachariasen [2000, 2001]). Minimizing total length has traditionally been the prime objective since this objective is also reasonably good with respect to signal delay in practice. Furthermore, for most terminal sets (also called *nets*), signal delay is not important; these nets are not part of the *critical signal path* of the chip. However, for those nets that are part of the critical signal path, signal delay is obviously very important.

In the past, the problem of minimizing sink delay was mainly attacked by using geometrical approaches. The delay of a wire was assumed to be linear in its length. So-called shallow-light algorithms limit the delay by bounding the radius of the tree (Nastansky, Selkow and Stewart [1974], Cong et al. [1992], Khuller, Raghavachari and Young [1995], Naor and Schieber [1997]). A similar approach is due to Alpert et al. [1993] who present a tradeoff between Prim's and Dijkstra's algorithm. Cong, Leung and Zhou [1993] justify that a Manhattan Steiner arborescence has good approximating properties with respect to delay. For newer VLSI fabrication technologies interconnect delays are becoming increasingly dominating when compared to gate delays (Cong et al. [1997]), thus linear delay approximation is not sufficient anymore. Therefore, algorithms directly incorporate a better delay approximation function (Prasitjutrakul and Kubitz [1990], Boese, Kahng and Robins [1993], Hu, Hou and Sapatnekar [1999], Lin, Liu and Hwang [2001]). For a comparison between several different performance-driven Steiner tree construction algorithms, we refer to Alpert et al. [2006]. Boese et al. [1995a] proved that minimizing the sum of weighted sink delays can be solved to optimality on the Hanan grid. This is not true for minimizing the maximum sink delay as shown by Boese et al. [1994]. For a good overview, see also Kahng and Robins [1995].

In this section we consider the problem of constructing RSMTs — which have minimum total length — that are as good as possible with respect to some signal delay objective. Therefore, without sacrificing minimum total length, we try to improve signal delay (if possible), that is, consider signal delay as a *secondary* objective when constructing RSMTs. The proposed algorithms can therefore be used to improve all minimum-length interconnections on the chip. However, for some nets on the critical signal path, it may be necessary to sacrifice minimum total length using alternative methods (Boese et al. [1995b], Kahng and Robins [1995], Peyer [2000]). Alpert et al. [2006] show that commonly used algorithms constructing timing-driven Steiner trees add only at most 2 % – 4 % extra wire length while improving the signal delay from source to sinks. The construction of (alternative) rectilinear Steiner trees that are good with respect to routability was considered by Bozorgzadeh, Kastner and Sarrafzadeh [2001].

For a given tree T spanning Z, let $P_T(r, z_i)$ be the path from the source r to a sink $z_i \in Z \setminus \{r\}$ in T and $|rz_i|_T$ its length (or the distance in T from r to z_i). Furthermore,

let $w_i > 0$ be a positive weight for sink z_i. We mainly focus our study on the following problem:

RECTILINEAR STEINER TREE PROBLEM WITH WEIGHTED SUM OF PATH LENGTHS SECONDARY OBJECTIVE (RSTPWP)	
Instance:	• A terminal set Z in the plane; • a designated source $r \in Z$; • weights $w_i > 0$ for all sinks $z_i \in Z \setminus \{r\}$.
Task:	Construct an $RSMT_r$ such that $\sum_{z_i \in Z \setminus \{r\}} w_i \lvert r z_i \rvert_T$ is minimized.

An optimal solution to RSTPWP is denoted by $RSMT_r$. See Figure 3.1 for an illustration.

Figure 3.1: Two RSMTs for the same set of terminals (depicted in black circle). The RSMT on the right has better signal delay properties than the RSMT on the left. In fact, the RSMT on the right is an optimal solution to RSTPWP since all paths from the source r to the sinks are shortest rectilinear paths.

The objectives of RSTPWP are motivated by VLSI design where it is important to build trees not only as short as possible but also with good timing properties. A signal which is propagated from the source through a tree must fulfill specified timing constraints. These constraints can be approximately reflected by weights w_i for all sinks $z_i \in Z \setminus \{r\}$, where critical sinks receive a higher weight than less critical sinks.

The advantage of the problem formulation of RSTPWP is that it is simple and does not use any timing parameter. However, the weights must be chosen carefully in order to appropriately express criticality of sinks. A commonly used delay approximation is due to Elmore [1948]. The *Elmore delay* model serves as a good estimation for computing the signal delay from the source to the sinks in a tree T. Given a source resistance R_d, resistance R_{unit} and capacitance C_{unit} per wire unit, and load capacitances c_i for every sink $z_i \in Z \setminus \{r\}$, the Elmore delay $\text{del}_T(z_i)$ of a sink z_i is defined as

$$\text{del}_T(z_i) := R_d C_{T,r} + \sum_{e=(u,v) \in E(P_T(r,z_i))} r_e \left(\frac{c_e}{2} + C_{T,v} \right),$$

where $r_e := R_{\text{unit}} \cdot \lvert uv \rvert_T$ and $c_e := C_{\text{unit}} \cdot \lvert uv \rvert_T$ denote the resistance respectively the capacitance of edge (u, v) and $C_{T,v}$ the downstream capacitance of the subtree of T rooted at vertex v. For more details about the Elmore delay model, see Peyer [2000].

3.1.2 Basic Notation and Definitions

For a given tree T rooted at r we use the following notation for a vertex $u \in V(T) \setminus \{r\}$ (see also Figure 3.2):

- $P(u)$: Predecessor or parent of u.
- $S(u)$: Successor of u, i.e., child of u that is colinear with $P(u)$ and u. If no such child exists then $S(u) = \text{NIL}$.
- $L(u)$: Left child of u when looking from $P(u)$ towards u. If no such child exists then $L(u) = \text{NIL}$.
- $R(u)$: Right child of u when looking from $P(u)$ towards u. If no such child exists then $R(u) = \text{NIL}$.

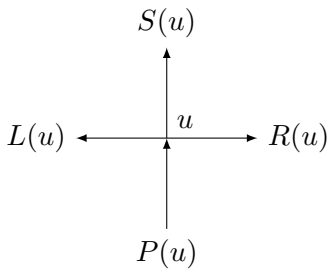


Figure 3.2: Neighboring vertices of u in a rectilinear Steiner tree.

A vertex u is called a *T-vertex* if both $L(u)$ and $R(u)$ exist, but $S(u)$ does not exist; otherwise u is called a *non T-vertex*.

A sequence of one or more adjacent, colinear edges is called a *segment*. A *maximal segment* is a segment which is not contained in any other segment. A *complete segment* is a segment whose interior vertices all are Steiner points and which is not contained in any other segment having only Steiner points as interior vertices. Note that any edge is contained in exactly one maximal/complete segment.

The *entering vertex* of a segment S is the (unique) vertex on the segment that is closest to the source. If the entering vertex is not the source itself, the *entering edge* of S is the (unique) edge having the entering vertex as its head. (The entering edge of a maximal segment S is always perpendicular to S.) Similarly, a *leaving edge* of S is an edge for which the tail belongs to S while the head does not.

The *Hanan grid* $H(Z)$ for the terminal set Z is obtained by drawing vertical and horizontal lines through each point in Z. Correspondingly, the *Hanan grid graph* $HGG(Z)$ is defined as follows: the set of intersections in $H(Z)$ are the vertices and a pair of vertices is connected if and only if the corresponding intersection points are adjacent in the Hanan

grid. The length l_{uv} of an edge $\{u, v\}$ in $HGG(Z)$ is the (Euclidean) distance between the corresponding Hanan grid intersections. We denote by $\mathcal{T}(Z)$ the set of subtrees of $HGG(Z)$ interconnecting Z and rooted at $r \in Z$.

3.1.3 Full Steiner Trees for RSTPWP

In this section we present a structural result characterizing optimal solutions to RSTPWP. This result can, e.g., be used in a pre-processing phase to reduce the graph instances for the exact algorithm presented in Section 3.1.4.

A *full Steiner tree (FST)* is a Steiner tree for which all terminals are leaves. All interior vertices in an FST are Steiner points or corner points. An optimal solution $RSMT_r$ to RSTPWP decomposes into directed FSTs, i.e., each FST has one designated terminal as its local root and all edges are directed away from this root. In the following we give a characterization of FSTs in any $RSMT_r$. Let F be an FST in an $RSMT_r$ with local root r_F. For an edge (u, v) we let $F_{(u,v)}$ denote the subtree of F rooted at u and containing v.

Lemma 3.1. *Let u be an internal non T-vertex in F for which $L(u)$ exists. Let v be the endpoint of the complete segment that contains the edge $(u, L(u))$ and which is on the same side of u as $L(u)$ on the complete segment. Then, v is a terminal.*

Proof. (In this and the following two lemmas we assume w.l.o.g. that the vertex u has its neighbors geometrically oriented as in Figure 3.3 such that $P(u)$ is below u.) Assume for the sake of contradiction that v is a non-terminal. Let $u_1 := u, u_2 := L(u), u_3, \ldots, u_k := v$ with $k \geq 2$ be the vertices on the segment uv. If we move uv and all its vertices up or

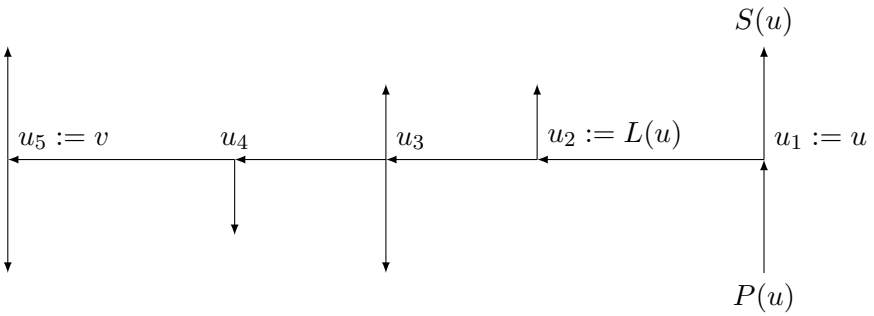


Figure 3.3: Proof illustration, Lemma 3.1.

down, the change in length of F is *linear* in the movement. Since F is an RSMT, the change must actually be zero. Thus, uv can be moved towards $P(u)$ without changing the length of F. Note that the path length from $P(u)$ to all vertices which are both above segment uv and adjacent to one vertex of $u_1, \ldots, u_k$ does not change as uv is moved towards $P(u)$. However, if there exists an adjacent vertex $w \neq P(u)$ below uv, then the path length to w decreases. Since w is either a terminal, or at least one terminal belongs

to the subtree rooted at w (and all the path weights are positive), the $RSMT_r$ is not optimal with respect to the weighted sum of path lengths secondary objective.

If there is no adjacent vertex except for $P(u)$ below uv, then v must be a corner point connected directly to u (since otherwise F is clearly not length-optimal). Now, if u is a corner point, too, then uv is part of a staircase connection. Otherwise, $S(u)$ must exist, but in this case F is clearly not length-optimal — a contradiction. □

We get a similar result for Lemma 3.1 when u is an internal non T-vertex and $R(u)$ exists.

Lemma 3.2. *Let u be an internal non T-vertex in F for which $L(u)$ exists. Then $F_{(u,L(u))}$ contains no corner point.*

Proof. From Lemma 3.1 we know that the endpoint v of the complete segment containing $(u, L(u))$ (and which is on the same side of u as $L(u)$ on the complete segment) is a terminal. Thus, none of the vertices on this segment are corner points. Now we recursively repeat this argument for all the interior vertices on uv; since these vertices are non T-vertices, the endpoints of the complete segments given by their left/right edges also must be terminals. Thus, the whole subtree is exhausted without encountering a corner point. □

We get a similar result for Lemma 3.2 when u is an internal non T-vertex and $R(u)$ exists.

Lemma 3.3. *Let u be an (internal) T-vertex in F. Let $v_L v_R$ be the complete segment that contains $(u, L(u))$ and $(u, R(u))$ such that v_L (respectively v_R) is on the same side of u as $L(u)$ (respectively $R(u)$) on $v_L v_R$. Then either v_L or v_R (or both) are terminals.*

Proof. We use the proof technique from Lemma 3.1. Assume that both v_L and v_R are non-terminals, such that the segment $v_L v_R$ only contains non-terminals. Then $v_L v_R$ can be moved freely up or down without changing the length of F. There must be at least one adjacent vertex except for $P(u)$ below $v_L v_R$, since otherwise the tree is not length-optimal. By moving $v_L v_R$ towards $P(u)$, the path length to this vertex decreases while the path length to no other vertex increases — a contradiction to the optimality of F with regard to the secondary objective. □

Theorem 3.4. *A full Steiner tree in an optimal solution to RSTPWP has at most one corner point.*

Proof. Consider the local root r_F of some FST F. Since r_F is a terminal it has exactly one out-going edge in F; let $r_F v$ be the complete segment that contains this edge. Since all interior Steiner points on $r_F v$ (if any) are non T-vertices, their left/right subtrees contain no corner point by Lemma 3.2. Therefore, we only need to consider vertex v and its subtrees (if any). We distinguish between three cases:

- v is a terminal: Then F clearly contains no corner point.
- v is a corner point: In this case v is the only corner point in F since the subtree given by v has no corner point by Lemma 3.2.
- v is a T-vertex: Let $v_L v_R$ be the complete segment containing v as defined by Lemma 3.3. Either v_L or v_R (or both) are terminals; assume w.l.o.g. that v_R is a terminal. Then the subtree $F_{(v,R(v))}$ contains no corner point (we use the same arguments as in the proof of Lemma 3.2).

  Thus, if F contains a corner point it is in $F_{(v,L(v))}$. Repeat the same arguments recursively for $F_{(v,L(v))}$ using v as the local root. Note that a single path is followed through F and as soon as one corner point is identified (if any), it will be the only corner point in F.

Therefore, in all three cases we conclude that F has at most one corner point. □

Let us now consider the Hanan grid $H(Z)$ for Z (defined in Section 3.1.2). Zachariasen [2001b] showed that there *exists* an optimal solution to RSTPWP in $H(Z)$. We can now prove the following stronger result:

Theorem 3.5. *An optimal solution to RSTPWP must be part of the Hanan grid $H(Z)$ for Z.*

Proof. Consider a Steiner point s in an optimal solution to RSTPWP. Clearly s is part of exactly one maximal horizontal segment and one maximal vertical segment. Consider the horizontal segment and assume that it contains no terminal. Then the entering edge of the segment must be perpendicular to the segment. By applying Lemma 3.1 and 3.3 to the entering vertex, we can prove that the segment *must* contain a terminal. Similarly, the vertical maximal segment must contain a terminal. Thus, s is a Hanan grid intersection point. □

3.1.4 Exact Algorithm

In order to construct an optimal solution to RSTPWP it is sufficient to compute an optimal solution in the corresponding Hanan grid graph by Theorem 3.5. The input is an undirected edge-weighted graph $G = (V, E)$ in which a set of terminals $Z \subseteq V$ and a source $r \in Z$ are given. Every sink $z_i \in Z \setminus \{r\}$ is assigned a path length weight $w_i > 0$.

In this section we give a (mixed) integer programming (IP) formulation for the general graph problem; this formulation is solved by standard branch-and-cut methods. The IP formulation is essentially the so-called directed formulation for the Steiner tree problem in graphs (Aneja [1980]). In addition, a *flow* from the source to the sinks measures the value of the secondary objective, i.e., the weighted sum of path lengths. Let $G_d = (V, E_d)$ be a directed graph having the same vertices as G and two directed opposite edges for each

edge in G. We assume that every edge $(u, v) \in E_d$ has a positive integer-valued weight $l_{uv} = l_{vu}$. This makes it easier to handle the secondary objective as the tree length can be assumed to be integral.

For any non-empty set $S \subset V$ define $\delta^+(S) := \{(u, v) \in E_d \ : \ u \in S \text{ and } v \in V \setminus S\}$ to be the set of edges leaving S and ending in $V \setminus S$. Two variables are defined for an edge $(u, v) \in E_d$: a decision variable $x_{uv} = 1$ if and only if edge $(u, v) \in E_d$ is chosen to be part of the Steiner tree and 0 otherwise, while a variable f_{uv} gives the amount of flow traversing the edge; $f_{uv} = 0$ if the edge is not part of the Steiner tree.

An IP formulation for the graph version of RSTPWP is then

$$\begin{aligned}
\min \quad & \sum_{(u,v)\in E_d} l_{uv}(x_{uv} + f_{uv}) \\
\text{s.t.} \quad & \sum_{(u,v)\in\delta^+(S)} x_{uv} \geq 1 && \text{for all } S \subset V,\ r \in S,\ (V \setminus S) \cap Z \neq \emptyset \quad (3.1) \\
& \sum_{(u,v)\in E_d} f_{uv} - \sum_{(v,u)\in E_d} f_{vu} = D_v && \text{for all } v \in V \setminus \{r\} \quad (3.2) \\
& f_{uv} \geq 0 && \text{for all } (u, v) \in E_d \\
& f_{uv} \leq x_{uv} && \text{for all } (u, v) \in E_d \\
& x_{uv} \in \{0, 1\} && \text{for all } (u, v) \in E_d \quad (3.3)
\end{aligned}$$

The constraints (3.1) and (3.3) are directed Steiner tree formulation constraints. The path length objective is measured by sending a certain amount of flow from the source to the sinks. The flow demand D_v in constraint (3.2) is zero for a non-terminal vertex $v \in V \setminus Z$, that is, we require flow-conservation at non-terminals. The demand D_{z_i} for a sink $z_i \in Z \setminus \{r\}$ is proportional to its path length weight w_i and is defined as follows: Let L be an upper bound on any path length (e.g., the total length of all edges), and let $W := \sum_{z_i \in Z\setminus\{r\}} w_i$ be the total path length weight for all sinks. Then we set $D_{z_i} := w_i/(LW)$.

Consider a sink $z_i \in Z \setminus \{r\}$. The contribution to the objective function of the flow from r to z_i is at most $L \cdot w_i/(LW) = w_i/W$. The total contribution is bounded by $\sum_{z_i \in Z\setminus\{r\}} w_i/W = 1$. Consequently, the tree constructed must have minimum length as edge-weights were assumed to be integer-valued.

The branch-and-cut algorithm used to solve the problem is basically the one by Koch and Martin [1998], but without the pre-processing algorithm for reducing the size of the problem. The traditional branching strategy which branches on variables is used; a fractional edge-variable with LP-value closest to 0.5 is selected. Note that it is enough to ensure that all edge-variables x_{uv} have integer value. When this is the case the flow-variables are set accordingly.

Computational results for this algorithm are presented in Section 3.1.6. It is well-known that solving Steiner tree problems in the Hanan grid graph is computationally difficult

due to a high degree of symmetry (Koch and Martin [1998]). Graph reduction methods for the ordinary Steiner tree problem on the Hanan grid graph are proposed by Winter [1995]. However, not all the proposed reduction tests — and in particular not the more powerful tests — generalize to RSTPWP. Another avenue for reducing the Hanan grid is to generate full Steiner trees and overlay these on the Hanan grid (Zachariasen [1999]). Preliminary results with FST generation for RSTPWP, based on the structural property stated in Theorem 3.4, appear to make it possible to reduce the Hanan grid significantly — but we do not elaborate on this subject in this work.

3.1.5 Heuristics

The overall heuristic approach considered in this section is the following. Assume we are given some RSMT T for the set of terminals Z. Specify a series of (local) modifications to T that retain total minimum length while decreasing the weighted sum of path lengths — or some other delay related objective.

Boese et al. [1995b] gave such a post-processing enhancement algorithm, denoted *Global Slack Removal (GSR)*. This algorithm removes so-called *V's* and *U's* from the tree until no removals are possible; these operations are illustrated in Figure 3.4.

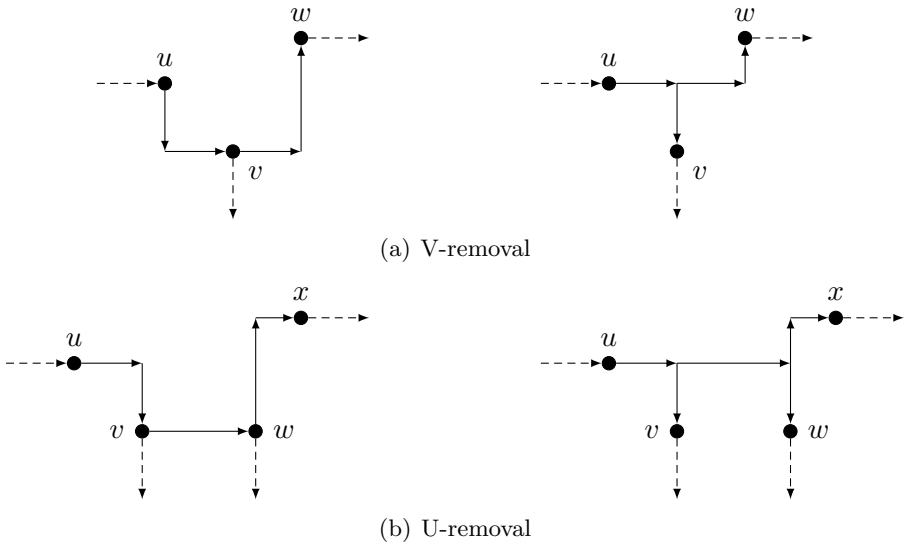


Figure 3.4: GSR operations. (a) V-removal: Sequence of three vertices u, v and w in increasing distance from the source; the subtree is replaced by a shortest path from u to w and a connection to v. Note that a V-removal is not applicable to any length-optimal tree. (b) U-removal: Sequence of four vertices u, v, w and x in increasing distance from the source; the subtree is replaced by a shortest path from u to x and connections to v and w.

The local modifications performed by GSR are special cases of a so-called *segment slide*, which is defined in Section A. In Section B, we give an algorithm to identify a "best"

segment slide in linear time, and in Section C we describe a new greedy method called *extended* GSR (XGSR) for performing a series of slides according to different objectives, including weighted sum of path lengths and weighted sum of Elmore delays.

Although we only consider input trees that are RSMTs, the XGSR algorithm may also be applied to trees that are not length-optimal. The only requirement is that the tree is rooted at some source $r \in Z$ and that all corner point connections have been oriented, that is, corner points are vertices in the trees (as defined in Section 3.1.2). However, some technical difficulties arise when the input tree is not length-optimal, since this may create overlapping edges. These difficulties are ignored in this work.

A Segment Slides

Consider a vertex $u \neq r$ in the tree T. This vertex defines a unique maximal segment MS containing u and being perpendicular to the edge $(P(u), u)$. As a degenerate case MS may consist solely of the vertex u, but let us assume that MS contains at least one edge. Also, w.l.o.g. let the edge $(P(u), u)$ be vertical with u above $P(u)$ such that MS is horizontal (Figure 3.5(a)).

Let $u_1, u_2, \ldots, u_k$ be the vertices on MS from left to right, where $u = u_m$ for some $m \in \{1, \ldots, k\}$. Let S be any segment given by a subsequence of vertices $u_l, \ldots, u, \ldots, u_r$ where $1 \leq l \leq m \leq r \leq k$, i.e., u belongs to the segment S. A *segment slide* for S is defined as a vertical (downward) movement of its vertices $u_l, \ldots, u, \ldots, u_r$ and edges such that all vertices are moved the same distance $\epsilon > 0$. The new vertices are denoted $u'_l, \ldots, u', \ldots, u'_r$ (Figure 3.5(b)). Depending on whether the original vertices are terminals or Steiner points — and in which directions these vertices are connected — it may be necessary to keep the old vertex and connect the original and new vertex (details are given in Section B).

We are obviously interested in segment slides that do not increase total tree length. For RSMTs the change in tree length should be precisely zero. As shown in Section C, performing a segment slide that does not increase total tree length cannot make the weighted sum of path lengths (or Elmore delays) worse. Clearly, V-removals and U-removals (Figure 3.4) are special cases of segment slides. Also, segment slides are strictly more powerful: Figure 3.6 gives a tree that contains no V's or U's, but for which there exists a segment slide that transforms it into an optimal solution to RSTPWP. However, it is also easy to construct instances for which no segment slide is possible and the tree is not an optimal solution to RSTPWP (Figure 3.7).

B Identifying Best Segment Slides

The segment S consists of the vertices $u_l, \ldots, u_r$. The change in tree length can be computed by adding up the contributions from each vertex. Assuming that each vertex is moved by distance $\epsilon > 0$, the change in tree length for a vertex is either $-\epsilon$, 0 or $+\epsilon$. We say that the vertex has value -1, 0 or $+1$, respectively. Case analysis gives the following values for moving a vertex $v \in \{u_l, \ldots, u_r\}$:

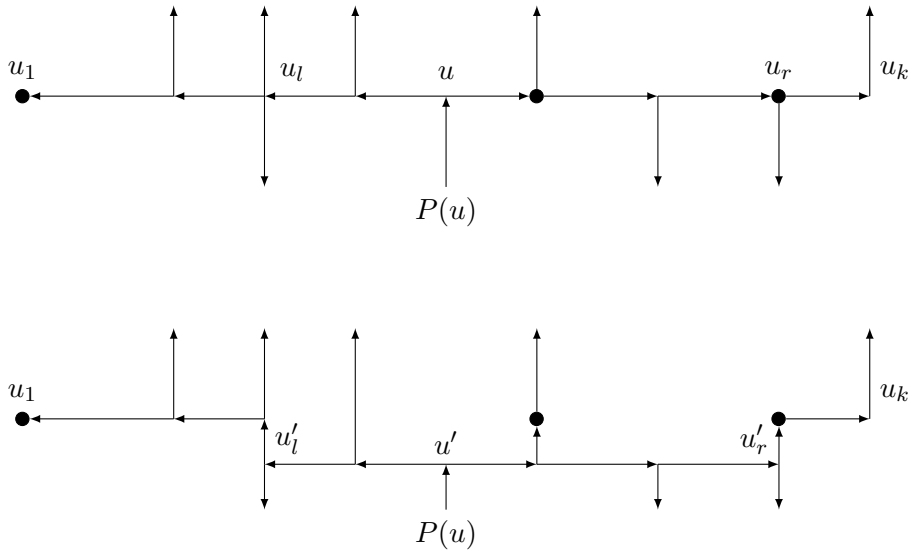
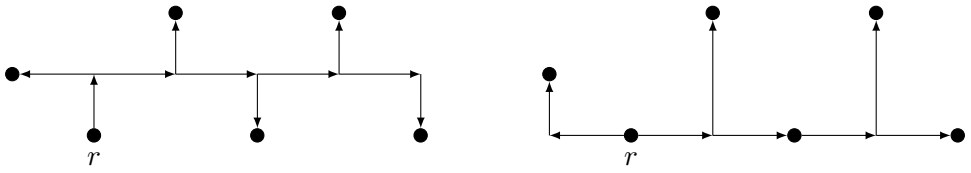


Figure 3.5: Segment slide example.



(a) RSMT (b) $RSMT_r$

Figure 3.6: (a) An RSMT that cannot be improved using GSR; (b) optimal solution to RSTPWP.

(a) RSMT (b) $RSMT_r$

Figure 3.7: (a) An RSMT that cannot be improved using segment slides; (b) optimal solution to RSTPWP.

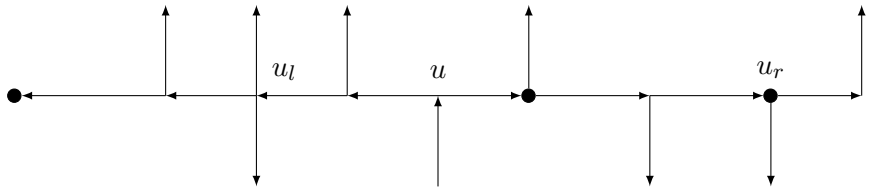
Vertex v is an endpoint ($v = u_l$ or $v = u_r$) If v is a corner point with a leaving edge going down it has value -1. If v has no leaving edge going down it has value $+1$, and it has value 0 otherwise.

Vertex v is an interior point ($v \neq u_l$ and $v \neq u_r$) If v is a Steiner point with a leaving edge going down and no leaving edge going up it has value -1. If v has no leaving edge going down it has value $+1$, and it has value 0 otherwise.

We now give an algorithm to find the best (sub)segment for a given maximal segment defined by an entry vertex u. Consider procedure BestSlide given on page 38. The segment is the best segment in the sense that it decreases total tree length by as much as possible as a function of ϵ (in case of a tie the longest segment is returned). The functions EvalNodeEndpoint(v) and EvalNodeInterior(v) are assumed to return the values given above. Note that endpoint evaluation does not depend on whether $v = u_l$ or $v = u_r$. In Section C we give various options for choosing the best segment among *all* maximal segments in the tree.

The algorithm BestSlide(u) first finds the best leftmost endpoint v_l and then the best rightmost endpoint v_r on the maximal segment. The overall best segment is then either segment $v_l, \ldots, u$, segment $u, \ldots, v_r$ or segment $v_l, \ldots, v_r$. An example is given in Figure 3.8.

Clearly, BestSlide(u) runs in linear time in the number of edges on the maximal segment. Since every edge belongs to exactly one maximal segment, running BestSlide(u) for all vertices $u \in V(T) \setminus \{r\}$ takes *linear time* in the number of edges (or vertices) in the tree.



	$+1$	0	$+1$	-1	$+1$	-1	0	
$+1$	$+1$	0	$+1$	0	$+1$	0	0	$+1$
$+3$	$+2$	$+1$	$+1$		$+1$	$+1$	0	$+1$

Figure 3.8: Identifying a best slide for a maximal segment in the example of Figure 3.5: The first line of $+1$,0 and -1's gives the value of moving the corresponding vertex downwards as *interior* vertex. The second line gives the corresponding *endpoint* values. The third line gives the accumulated value of every endpoint. The best slide is the segment $u_l, \ldots, u_r$ which has total value zero, i.e., does not change the length of the tree.

Procedure BESTSLIDE(u)

```
   // find best leftmost endpoint v_l of maximal segment given by u
 1 v = L(u); Δ = 0; Δ_l = ∞; v_l = u;
 2 while v ≠ NIL do
 3     δ = Δ + EVALNODEENDPOINT(v); // evaluate v as an endpoint
 4     if δ ≤ Δ_l then
 5         Δ_l = δ; v_l = v;
 6     end
 7     Δ = Δ + EVALNODEINTERIOR(v); v = S(v);
 8 end
   // find best rightmost endpoint v_r of maximal segment given by u
 9 v = R(u); Δ = 0; Δ_r = ∞; v_r = u;
10 while v ≠ NIL do
11     δ = Δ + EVALNODEENDPOINT(v); // evaluate v as an endpoint
12     if δ ≤ Δ_r then
13         Δ_r = δ; v_r = v;
14     end
15     Δ = Δ + EVALNODEINTERIOR(v); v = S(v);
16 end
   // find best combined slide (either both sides or only left or right)
17 Δ = ∞; u_l = NIL; u_r = NIL;
18 δ = Δ_l + Δ_r + EVALNODEINTERIOR(u); // both sides
19 if δ < Δ then
20     Δ = δ; u_l = v_l; u_r = v_r;
21 end
22 δ = Δ_l + EVALNODEENDPOINT(u); // left side
23 if δ < Δ then
24     Δ = δ; u_l = v_l; u_r = u;
25 end
26 δ = Δ_r + EVALNODEENDPOINT(u); // right side
27 if δ < Δ then
28     Δ = δ; u_l = u; u_r = v_r;
29 end
30 return (Δ, u_l, u_r)
```

C XGSR Algorithm

The XGSR algorithm (Figure 3.2) is a greedy method for performing a series of segment slides. In each iteration, XGSR identifies the overall best segment slide to perform and applies it to the tree, that is, slides the segment until one of its vertices overlaps with a neighboring vertex.

First, the overall best segment slide decrease total length as much as possible (for RSMTs only zero value segment slides are considered). Second, the *gain* of the segment slide is maximized. The gain is a measure of how much the segment slide improves the chosen secondary objective: given a segment $u_l, \ldots, u_r$ with entry vertex u, we assume that the function ComputeGain(u, u_l, u_r) returns the gain obtained by sliding the segment; this function is assumed to return zero when the segment slide increases the total tree length. ApplySegmentSlide(T, u, u_l, u_r) applies a slide of that segment to tree T.

Algorithm 3.2: XGSR

```
   // iteratively find slide with best gain and apply it
 1 g* = ∞;
 2 while g* > 0 do
 3     Δ* = ∞; u* = NIL; u_l* = NIL; u_r* = NIL; g* = 0;
 4     forall u ∈ V(T) \ {r} do
           // find best slide for maximal segment given by u
 5         (Δ, u_l, u_r) = BestSlide(u);
           // is tree length decrease better or equal to best seen so far?
 6         if (Δ < ∞) and (Δ ≤ Δ*) then
 7             g = ComputeGain(u, u_l, u_r);
 8             if g > g* then
                   // // this is the best segment slide seen so far
 9                 Δ* = Δ; u* = u; u_l* = u_l; u_r* = u_r; g* = g;
10             end
11         end
12     end
       // apply segment slide to tree if gain is positive
13     if g* > 0 then
14         T = ApplySegmentSlide(T, u*, u_l*, u_r*);
15     end
16 end
17 return T
```

We experimentally evaluate the following four secondary objectives:

1. Weighted sum of path lengths (RSTPWP): $\sum_{z_i \in Z \setminus \{r\}} w_i |rz_i|_T$.
2. Maximum path length: $\max_{z_i \in Z \setminus \{r\}} |rz_i|_T$.

3. Weighted sum of Elmore delays (see Section 3.1.2): $\sum_{z_i \in Z \setminus \{r\}} w_i \operatorname{del}_T(z_i)$.
4. Maximum Elmore delay: $\max_{z_i \in Z \setminus \{r\}} \operatorname{del}_T(z_i)$.

The corresponding gain function is equal to the *decrease* in the secondary objective function. The first two gain functions can be computed in linear time for *all* segment slides, while Elmore delay computations take linear time for *each* segment slide (Peyer [2000]).

D Properties of the XGSR Algorithm

In this section we present some theoretical properties related to the XGSR algorithm. First we give a general result concerning terminal sets of size 4. Then we prove that XGSR cannot make any of the secondary objectives proposed earlier worse.

Lemma 3.6. *Applying* XGSR *to an RSMT where* $|Z| \leq 4$ *produces an optimal solution to RSTPWP.*

Proof. For $|Z| \leq 3$ any RSMT is also an optimal solution to RSTPWP, since all inter-terminal paths are shortest rectilinear paths. We now show that for $|Z| = 4$, after applying XGSR to an RSMT, all source-sink paths are shortest rectilinear paths. Therefore, the tree is obviously an optimal solution to RSTPWP.

Assume on the contrary that T is the output of XGSR, and that there exists a sink z_i for which the path $P := P_T(r, z_i)$ is not a shortest rectilinear path. This implies that P contains an edge $(P(u), u)$, a segment S of vertices $u, \ldots, v$ and an edge (v, w) that together form a non-optimal subpath (Figure 3.9a). We may w.l.o.g. assume that neither u nor v are corner points — otherwise we may flip the corner(s) to form another non-optimal subpath. Now, u is either a sink or a Steiner point being the root of a subtree containing at least one sink not belonging to P. The same holds for vertex v. The vertex w is either identical to z_i or is the root of a subtree containing z_i. Thus, the vertices u, v and w represent three distinct sinks. Since $|Z| = 4$ the tree spans no more sinks, and therefore the segment S has no interior vertices, i.e., S is an edge (u, v) that can be slid towards $P(u)$ without changing the length of T. This contradicts the assumption that no path length improving segment slide exists. □

Note that Lemma 3.6 also proves that GSR, given some RSMT, constructs an optimal solution to RSTPWP; the segment S in the proof is in fact part of a U that should have been removed by GSR. For $|Z| \geq 5$ not all source-sink paths need to be shortest rectilinear paths (Figure 3.9b). Furthermore, neither GSR nor XGSR always construct an optimal solution to RSTPWP for $|Z| \geq 5$ as shown in the experimental section (see Table 3.5).

The following lemma justifies the use of ApplySegmentSlide in XGSR in order to minimize weighted sum of path lengths.

Lemma 3.7. *Given a tree* $T_1 \in \mathcal{T}(Z)$*, let* $T_2 \in \mathcal{T}(Z)$ *be the output tree of subroutine* ApplySegmentSlide *in* XGSR *if applied to* T_1*. Then*

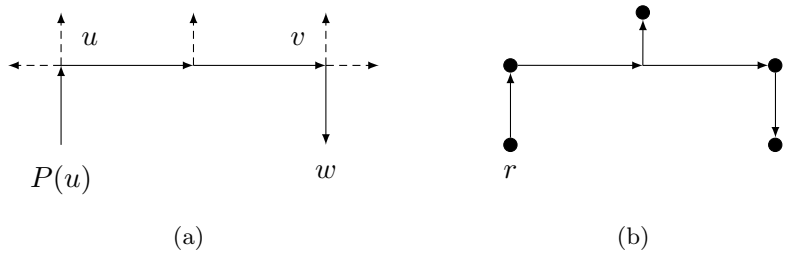


Figure 3.9: (a) Proof illustration of Lemma 3.6; (b) A 5-terminal set for which one source-sink path is not a shortest rectilinear path.

(i) $|T_2| \leq |T_1|$,

(ii) $|rz|_{T_2} \leq |rz|_{T_1}$ *for all terminals* $z \in Z$,

(iii) there is at least one terminal $z \in Z$ *with* $|rz|_{T_2} < |rz|_{T_1}$.

Proof. (i) For any cost function, XGSR slides a segment with entry vertex u only if the return value Δ of BestSlide(u) is non-positive, that is, the tree does not increase its total length.

(ii) Let S_1 be the segment in T_1, and S_2 the segment in T_2 after sliding S_1. W.l.o.g., S_1 is a horizontal segment which is slid downwards. Let (x^l, y_i) and (x^r, y_i) be the coordinates of the leftmost and rightmost vertex of S_i $(i = 1, 2)$ with $x^l < x^r$ and $y_2 < y_1$. Let B denote the induced subgraph of $HGG(Z)$ for which $v = (x_v, y_v) \in V(B)$ if and only if $x^l \leq x_v \leq x^r$ and $y_v \in \{y_1, y_2\}$. For each terminal $z \in Z$ consider the intersection of $P_{T_1}(r, z)$ and $P_{T_2}(r, z)$ with B (Figure 3.10). A segment slide only changes those paths which intersect B. Obviously, $B \cap P_{T_1}(r, z) \neq \emptyset$ if and only if $B \cap P_{T_2}(r, z) \neq \emptyset$. Hence, if $B \cap P_{T_1}(r, z) = \emptyset$ then $|rz|_{T_2} = |rz|_{T_1}$. Now consider a terminal z with $B \cap P_{T_1}(r, z) \neq \emptyset$. $P_{T_1}(r, z)$ enters B in $HGG(Z)$ at p and leaves B at some vertex q (or ends in $z \in V(B)$). Obviously, a slide does not change the entering or leaving vertex. But $P_{T_2}(r, z) \cap B$ is a shortest p-q-path in B. Hence, $|pq|_{T_2} \leq |pq|_{T_1}$. Since $P_{T_2}(r, p) = P_{T_1}(r, p)$ and $P_{T_2}(q, z) = P_{T_1}(q, z)$, the path length from r to z does not increase: $|rz|_{T_2} \leq |rz|_{T_1}$.

(iii) This follows immediately from the condition $g^* > 0$ for which ApplySegmentSlide is applied. □

By Lemma 3.7, the total tree length does not increase and no source-sink-path gets longer if ApplySegmentSlide is applied to a tree. These properties give rise to the following definition.

Definition 3.8. *A function f is* weakly decreasing w.r.t. g *if $f(g(T)) \leq f(T)$ for every tree $T \in \mathcal{T}(Z)$, and f is* strongly decreasing w.r.t. g *if $f(g(T)) < f(T)$ for every tree $T \in \mathcal{T}(Z)$.*

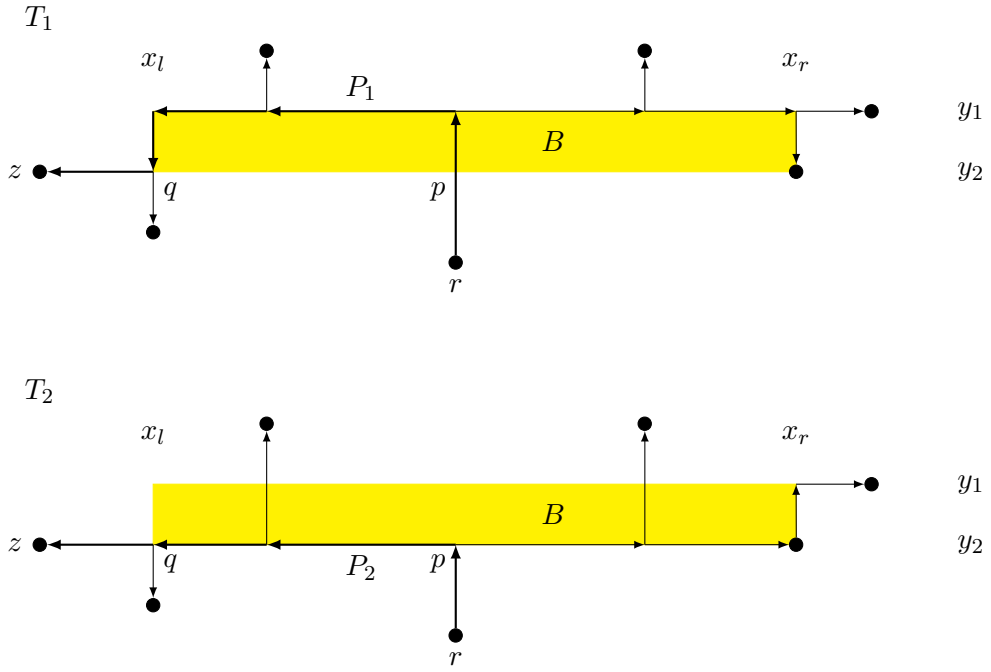


Figure 3.10: A segment slide does not change the entering or leaving vertex of a path from the source r to the sink z in B. (The induced subgraph B of $HGG(Z)$ is marked with a yellow background.)

So the total tree length $f(T) := |T|$ is a weakly decreasing function, while the weighted sum of path lengths $f(T) := \sum_{z_i \in Z \setminus \{r\}} w_i |r z_i|_T$ is a strongly decreasing function w.r.t. ApplySegmentSlide. A similar result can be achieved for the Elmore delay function:

Lemma 3.9. *The Elmore delay function* del *is weakly decreasing w.r.t.* ApplySegmentSlide.

Proof. First we show that a certain path delay does not increase at any sink if a subtree is moved distance $D > 0$ closer to the source while increasing the length of the subtree by at most D. Consider the subtree T_1^v of T_1 rooted at v where p is on the path from the root r to vertex u (Figure 3.11).

By deleting edge (u, v) and reconnecting T_1^v via two edges (p, w) and (w, v) the whole subtree is moved towards the source by a distance of $D := |pu|$. Let T_2 be the resulting tree of that replacement. Furthermore, denote by $\mathrm{del}_T(p, v)$ the Elmore delay in T from p to v. Then we have

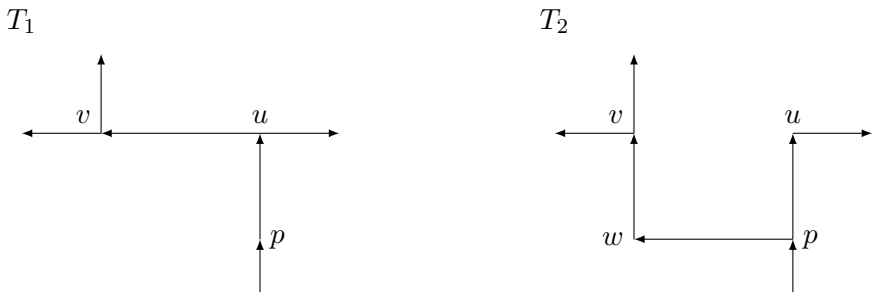


Figure 3.11: Moving the subtree rooted at v closer to the source does not increase the Elmore delay at any sink

$$\begin{aligned}
\mathrm{del}_{T_1}(p,v)/R_{\mathrm{unit}} &= |pu|\left(\tfrac{c_{(p,u)}}{2}+C_{T_1,u}\right)+|uv|\left(\tfrac{c_{(u,v)}}{2}+C_{T_1,v}\right)\\
&\geq D\left(\frac{C_{\mathrm{unit}}D}{2}+C_{\mathrm{unit}}|uv|+C_{T_1,v}\right)+|uv|\left(\frac{C_{\mathrm{unit}}|uv|}{2}+C_{T_1,v}\right)\\
&= (D+|uv|)\left(\frac{C_{\mathrm{unit}}|uv|}{2}+C_{T_1,v}+\frac{C_{\mathrm{unit}}D}{2}\right)\\
&= (D+|pw|)\left(\frac{C_{\mathrm{unit}}|pw|}{2}+C_{T_2,v}+\frac{C_{\mathrm{unit}}D}{2}\right)\\
&= |pv|\left(\frac{C_{\mathrm{unit}}|pv|}{2}+C_{T_2,v}\right)\\
&= \mathrm{del}_{T_2}(p,v)/R_{\mathrm{unit}}
\end{aligned}$$

Therefore, $\mathrm{del}_{T_2}(p,z) \leq \mathrm{del}_{T_1}(p,z)$ for all sinks z in T_1^v. Sliding a segment can be viewed as a series of edge slides. By repeating the above argument it is shown that a segment slide does not increase the path delay from p to any sink in the subtree T_1^v. Since we allow only those slides which do not increase total tree length — and therefore do not increase the length of the subtree rooted at p — $\mathrm{del}_{T_1}(p)$ does not increase either. Moreover, $\mathrm{del}_{T_2}(u) < \mathrm{del}_{T_1}(u)$ holds because the capacitance of $C_{T_1,v} + C_{\mathrm{unit}}|uv|$ does not affect the delay from p to u in T_2 anymore. Since all other edges remain the same, the Elmore delay does not increase at any sink of the whole tree by performing ApplySegmentSlide. □

From Lemma 3.7 and 3.9 it follows that the four secondary objectives proposed in Section C are weakly decreasing w.r.t. ApplySegmentSlide.

E Running Time of the XGSR Algorithm

The running time of XGSR is mainly determined by the number of applied segment slides. So far it is not clear whether XGSR stops at all. This question is answered by the following lemma:

Lemma 3.10. *The number of iterations of the algorithm* XGSR *is* $O(n^3)$.

Proof. By assumption, XGSR starts with a tree where each Steiner point is a Hanan vertex. Sliding a segment of a tree T results in a new tree having the same property. For $i \in \{1, \ldots, n-1\}$ let P_i be the path from the source r to terminal z_i in T, and $H(P_i)$ be the set of Hanan vertices covered by all edges of P_i. By construction, for each path P_i a slide does not increase the number of Hanan vertices covered by P_i. Moreover, by Lemma 3.7(iii), there is at least one terminal z_j for which the length of P_j decreases. Hence $|H(P_j)|$ decreases, too. Therefore, the sum of all covered Hanan vertices $\sum_{i=1}^{n-1} |H(P_i)|$ decreases by at least one when a slide is made. Initially, each path covers at most n^2 Hanan vertices of the Hanan grid, so all paths cover at most $n^2(n-1)$ vertices (counting vertices several times if covered by several paths). Since n Hanan vertices are covered by terminals, XGSR stops after at most $n(n^2-n-1)$ iterations. □

The following example gives a lower bound of n^2-n for the number of iterations in XGSR (see Figure 3.12). For n being a multiple of 4, we define the set of terminals as follows: There are $\frac{n}{4}$ vertices $\{(0,0),\ (2,0), (4,0), \ldots, (\frac{n}{2}-2, 0)\}$, $\frac{n}{4}$ vertices $\{(0,2),\ (2,2), (4,2), \ldots, (\frac{n}{2}-2, 2)\}$ and $\frac{n}{2}$ vertices $\{(\frac{n}{2}, 0),\ (\frac{n}{2}, \frac{4}{n-2}), (\frac{n}{2}, \frac{8}{n-2}), \ldots, (\frac{n}{2}, 2)\}$. The root r is (0,0) and the rightmost vertical segment contains $\frac{n}{2}$ equidistant vertices. Consider now the RSMT which zig-zags from right to left. Sliding the rightmost horizontal segment up requires $\frac{n}{2}-1$ steps, sliding the second-right horizontal segment (together with the rightmost horizontal) again takes $\frac{n}{2}-1$ steps etc. Altogether, it takes $\frac{n(n-2)}{8}$ steps until every path from r to a terminal is as short as possible. (Note that in this case all Hanan vertices are covered at any step during the algorithm, so the total number of Hanan vertices covered does not decrease.)

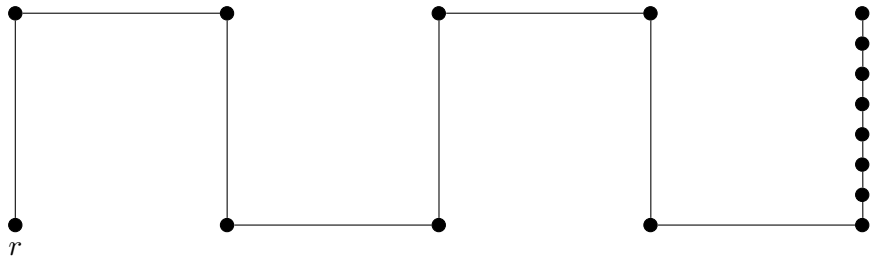


Figure 3.12: Example for $n = 16$.

Lemma 3.11. *There exist RSMTs for which* XGSR *terminates after* $\Omega(n^2)$ *iterations.*

As already pointed out, applying BestSlide(u) for all $u \in V(T) \setminus \{r\}$ and running ApplySegmentSlide takes $O(n)$ time. Thus, we obtain the following lemma.

Lemma 3.12. *(i) Let* $O(t(n))$ *be the running time of* ComputeGain *for some function* t. *Then, the running time of* XGSR *is* $O\left(n^3 \max(n, t(n))\right)$.

Chip	Johannes	Ilse	Aidan	Heinz	Johanna	
Technology	350 nm	250 nm	180 nm	130 nm	130 nm	All
Metal	Al	Al	Cu	Cu	Cu	
#nets in total	180 129	681 665	727 020	3 516	735 355	2 327 685
#nets with 2– 3 pins (%)	75.76	80.16	73.89	80.29	79.84	77.75
#nets with 4– 5 pins (%)	12.84	10.52	12.37	12.00	11.60	11.61
#nets with 6– 7 pins (%)	4.40	2.11	6.12	2.82	3.42	3.95
#nets with 8–10 pins (%)	4.62	3.40	5.95	2.07	2.19	3.90
#nets with 11–20 pins (%)	1.91	2.96	0.53	2.33	1.96	1.99
#nets with 21–40 pins (%)	0.37	0.80	0.82	0.49	0.68	0.62
#nets with ≥ 41 pins (%)	0.10	0.05	0.32	0.00	0.31	0.18
$R_dC_{T,r}$ proportion (%)	89.74	81.54	96.40	97.50	98.09	92.38

Table 3.1: Chip characteristics and net statistics. The total number of nets on each chip and their size distribution is given. The $R_dC_{T,r}$ proportion row is explained in the text.

(ii) If the gain is determined by the weighted sum of path lengths or the maximum path length, then XGSR *runs in* $O(n^4)$ *time.*

(iii) If the gain is determined by the weighted sum of Elmore delays, or the maximum Elmore delay then XGSR *runs in* $O(n^5)$ *time.*

Based on experiences and experimental results presented in the next section, we conjecture that XGSR actually terminates after $O(n^2)$ iterations — giving a running time of $O(n^3)$ for path length secondary objectives.

3.1.6 Experimental Results

The goals of our experiments are threefold: first, we investigate how much the delay of an (arbitrary) RSMT can be improved. Second, we compare our new XGSR heuristic to the GSR heuristic. Finally, we show that both heuristics perform very well in the sense that most of the trees constructed are optimal for RSTPWP; we do this by computing optimal solutions to RSTPWP using the exact algorithm described in Section 3.1.4.

All experiments with GSR and XGSR were made on an IBM S85 machine with 18 RS64 IV processors running at 600 MHz (all programs were run sequentially, and each processor is comparable to a 650 MHz Pentium III). The exact algorithm was run on a 933 MHz Pentium III.

All test instances are from real chips, made available by courtesy of IBM. Characteristics and net statistics for the five chips are given in Table 3.1.

Thinner wires in newer chip technologies result in a smaller capacitance and a larger resistance per wire unit. In addition, copper (Cu) has better thermal properties and a smaller resistance than aluminum (Al). Therefore, it is necessary to take the technology

parameters into account when studying delay properties of chip nets. The general tendency is that interconnect delays are becoming increasingly dominating when compared to gate delays (Cong et al. [1997]).

The $R_dC_{T,r}$ *proportion* in Table 3.1 gives the average percentage of the $R_dC_{T,r}$ term in the Elmore delay formula (see Section 3.1.2) relative to the maximum sink delay for the RSMT. This term is directly proportional to the length of the net and thus a constant for an RSMT. The percentage is quite high and gives a bound on the possible delay improvement for the net, e.g., for the newest chip the average delay improvement can be at most 1.91 %. It should be noted that an increasing $R_dC_{T,r}$ proportion for newer technologies is not a tendency that usually should be expected; rather, it means that the newer chips in this case have been better optimized for delay than the older chips.

As pointed out in Section D, none of the secondary objectives considered can be improved for nets having 2 or 3 terminals. In order to have more uniform data, we also excluded nets with more than 40 terminals. The nets of size 4 to 40 were divided into five groups as shown in Table 3.1. In total, we performed experiments on 509,792 nets from the five chips. All path length and Elmore delay weights were set to 1 in our experiments.

RSMTs were constructed using the exact algorithm of Hetzel [1995]. This algorithm has no knowledge of the source of the net and does not attempt to optimize any secondary objective. The RSMTs were used as input for GSR and XGSR; below we report on the improvement of the secondary objectives chosen in Section C.

In Table 3.2 we present the main results of our study. Both heuristics GSR and XGSR are able to improve each of the secondary objectives considerably; the average improvement of the Elmore delay is smaller, but the maximum improvement is still significant. The improvements obtained by GSR and XGSR are similar, but XGSR is clearly better. This is illustrated by the columns GSR+ and XGSR+ which give the fraction of nets for which one heuristic is strictly better than the other; for the larger nets, XGSR obtains better solutions for a considerable fraction of the nets while GSR almost never is better.

In Table 3.3 and 3.4 we give detailed results for each of the four (large) chips. Table 3.3 presents results for RSTPWP while Table 3.4 presents results for the weighted sum of Elmore delays secondary objective. The path length improvement becomes larger for newer technologies, while the Elmore delay improvement appears to decrease for newer technologies (which is related to the fact that the more recent chips are better optimized for delay — giving fewer opportunities for improvement).

In Table 3.2 the results of the exact algorithm for RSTPWP are also presented (column OPT). All instances are also solved to optimality. For instances with up to 10 terminals, the exact algorithm needs less than one second on average, while the average running time for the size group 11–20 terminals is 12 seconds. For the larger instances, a substantial computing effort is needed for some instances. The result shows that XGSR produces excellent solutions; the average excess of the secondary objective from the optimal solution is less than 0.1 % for nets having at most 7 terminals and less than 0.2 % for nets having up to 20 terminals. Furthermore, as shown in Table 3.5, most of the trees constructed

Size	GSR		XGSR		OPT		GSR+	XGSR+
Weighted sum of path lengths (RSTPWP)								
4–5	1.69	(40.37)	1.69	(40.37)	1.69	(40.37)	0.00	0.00
6–7	2.44	(43.96)	2.46	(43.96)	2.50	(47.20)	0.00	0.79
8–10	2.92	(46.99)	2.98	(46.99)	3.07	(46.99)	0.03	2.34
11–20	2.43	(43.29)	2.51	(43.29)	2.70	(45.55)	0.07	5.56
21–40	2.69	(41.95)	2.83	(41.95)	3.54	(69.34)	0.10	13.46
Average	2.14	(46.99)	2.16	(46.99)	2.23	(69.34)	0.02	1.47
Maximum path length								
4–5	1.78	(48.95)	1.78	(48.95)			0.00	0.00
6–7	2.41	(51.69)	2.44	(51.69)			0.00	0.54
8–10	2.94	(55.06)	3.01	(55.06)			0.00	1.55
11–20	2.76	(45.40)	2.87	(45.40)			0.00	3.52
21–40	3.05	(57.53)	3.24	(57.53)			0.03	7.09
Average	2.22	(57.53)	2.26	(57.53)			0.00	0.90
Weighted sum of Elmore delays								
4–5	0.07	(22.20)	0.07	(22.20)			0.00	0.00
6–7	0.10	(21.30)	0.10	(21.30)			0.00	0.80
8–10	0.12	(16.21)	0.12	(16.21)			0.03	2.36
11–20	0.21	(19.33)	0.22	(19.33)			0.08	5.56
21–40	0.23	(12.54)	0.24	(12.54)			0.14	13.40
Average	0.10	(22.20)	0.10	(22.20)			0.02	1.47
Maximum Elmore delay								
4–5	0.09	(29.45)	0.09	(29.45)			0.00	0.00
6–7	0.14	(21.38)	0.14	(21.38)			0.00	0.82
8–10	0.17	(20.34)	0.17	(20.34)			0.03	2.30
11–20	0.29	(27.72)	0.30	(27.72)			0.06	5.50
21–40	0.32	(17.32)	0.33	(17.32)			0.11	11.66
Average	0.14	(29.45)	0.14	(29.45)			0.01	1.40

Table 3.2: Average improvement of secondary objectives in percent for GSR and XGSR (maximum improvement given in parenthesis). For RSTPWP the column OPT gives the improvement of the optimal solution. In column GSR+ (resp. XGSR+) the fraction of nets for which GSR (resp. XGSR) is strictly better than XGSR (resp. GSR) is given.

Size	GSR		XGSR		GSR+	XGSR+
Johannes						
4–5	1.47	(37.92)	1.47	(37.92)	0.00	0.00
6–7	2.16	(37.73)	2.18	(37.73)	0.00	0.71
8–10	2.30	(38.11)	2.35	(38.11)	0.08	2.41
11–20	2.68	(40.53)	2.77	(40.53)	0.14	5.72
21–40	3.45	(30.50)	3.65	(30.50)	0.57	14.57
Average	1.88	(40.53)	1.90	(40.53)	0.04	1.28
Ilse						
4–5	1.27	(33.95)	1.27	(33.95)	0.00	0.00
6–7	2.23	(42.26)	2.25	(42.26)	0.01	0.78
8–10	2.82	(46.70)	2.89	(46.70)	0.05	2.73
11–20	1.70	(34.66)	1.76	(34.66)	0.10	4.91
21–40	1.46	(33.19)	1.53	(33.19)	0.16	12.62
Average	1.71	(46.70)	1.74	(46.70)	0.03	1.81
Aidan						
4–5	1.45	(40.37)	1.45	(40.37)	0.00	0.00
6–7	2.28	(40.85)	2.29	(40.85)	0.00	0.76
8–10	3.01	(46.99)	3.07	(46.99)	0.01	2.07
11–20	2.03	(43.29)	2.12	(43.29)	0.00	7.69
21–40	2.61	(25.05)	2.74	(25.05)	0.05	12.24
Average	2.06	(46.99)	2.08	(46.99)	0.01	1.22
Johanna						
4–5	2.35	(38.66)	2.35	(38.66)	0.00	0.00
6–7	2.95	(43.96)	2.99	(43.96)	0.00	0.86
8–10	3.16	(37.79)	3.23	(37.79)	0.04	2.46
11–20	3.51	(39.27)	3.63	(39.27)	0.01	5.87
21–40	4.07	(41.95)	4.29	(41.95)	0.02	15.79
Average	2.72	(43.96)	2.75	(43.96)	0.01	1.55

Table 3.3: Average improvement in percent for GSR and XGSR for *weighted sum of path lengths* and each of the four large chips (maximum improvement given in parenthesis). In column GSR+ (resp. XGSR+) the fraction of nets for which GSR (resp. XGSR) is strictly better than XGSR (resp. GSR) is given.

Size	GSR		XGSR		GSR+	XGSR+
Johannes						
4–5	0.11	(22.20)	0.11	(22.20)	0.00	0.00
6–7	0.20	(21.30)	0.20	(21.30)	0.01	0.73
8–10	0.19	(10.44)	0.20	(10.44)	0.11	2.42
11–20	0.33	(14.48)	0.34	(14.48)	0.17	5.66
21–40	0.61	(10.31)	0.65	(10.31)	0.57	14.86
Average	0.17	(22.20)	0.17	(22.20)	0.05	1.28
Ilse						
4–5	0.16	(22.18)	0.16	(22.18)	0.00	0.00
6–7	0.21	(13.94)	0.21	(13.94)	0.02	0.80
8–10	0.18	(16.21)	0.18	(16.21)	0.04	2.73
11–20	0.29	(16.44)	0.31	(16.44)	0.11	4.91
21–40	0.37	(12.54)	0.39	(12.54)	0.09	12.62
Average	0.19	(22.18)	0.20	(22.18)	0.03	1.81
Aidan						
4–5	0.03	(14.99)	0.03	(14.99)	0.00	0.00
6–7	0.08	(12.51)	0.08	(12.51)	0.00	0.77
8–10	0.09	(8.39)	0.09	(8.39)	0.01	2.10
11–20	0.19	(19.33)	0.20	(19.33)	0.03	7.69
21–40	0.20	(9.00)	0.21	(9.00)	0.07	12.26
Average	0.07	(19.33)	0.07	(19.33)	0.01	1.23
Johanna						
4–5	0.03	(8.12)	0.03	(8.12)	0.00	0.00
6–7	0.05	(11.89)	0.05	(11.89)	0.00	0.86
8–10	0.06	(5.40)	0.06	(5.40)	0.03	2.47
11–20	0.07	(3.96)	0.07	(3.96)	0.04	5.89
21–40	0.05	(1.97)	0.05	(1.97)	0.22	15.51
Average	0.04	(11.89)	0.04	(11.89)	0.01	1.54

Table 3.4: Average improvement in percent for GSR and XGSR for *weighted sum of Elmore delays* and each of the four large chips (maximum improvement given in parenthesis). In column GSR+ (resp. XGSR+) the fraction of nets for which GSR (resp. XGSR) is strictly better than XGSR (resp. GSR) is given.

Size	RSMT	GSR	XGSR	Not-Opt	GSR-Opt	XGSR-Opt
4–5	70.18	99.94	99.94	158	0.00	0.00
6–7	46.45	98.50	99.27	1 380	0.22	51.81
8–10	29.85	95.80	98.08	3 818	0.71	54.50
11–20	15.62	88.80	94.01	4 709	0.53	46.80
21–40	3.09	71.28	82.40	5002	0.28	38.88
Average	52.01	97.06	98.41	15067	0.46	46.09

Table 3.5: Percentage of optimal solutions for RSTPWP. The columns RSMT, GSR and XGSR give the percentage of optimal solutions for RSMT, GSR and XGSR, respectively. The Not-Opt column gives the number of nets for which either GSR or XGSR does not find an optimal solution; among these the percentage of nets actually solved to optimality by GSR and XGSR is given in the columns GSR-Opt and XGSR-Opt, respectively.

by GSR and XGSR are optimal. On average, more than 98 % of the trees constructed by XGSR are optimal solutions to RSTPWP. Among nets for which either GSR or XGSR do not find an optimal tree, GSR constructs an optimal solution for less than 1 % of the nets, while XGSR constructs an optimal solution for almost 50 % of the nets. Figure 3.13 and Figure 3.14 give two examples for which XGSR finds an optimal solution while GSR does not, while Figure 3.15 shows a net for which neither GSR nor XGSR find an optimal solution

A simple measure of the delay properties of a net is the *maximum detour* for that net, i.e., the maximum source-sink tree-distance to rectilinear distance ratio (see Table 3.6). For nets of size 4–5 this ratio is on average 1.078 for RSMTs; after applying XGSR using the maximum path length objective this ratio has dropped to 1.014. For nets of size 6–7 the ratio drops from 1.211 to 1.110, and for the largest group (21–40 terminals), the ratio drops from 2.159 to 2.058. For smaller nets the improvement is therefore significant.

Size	Number	Average Ratio			Maximum Ratio		
		RSMT	GSR	XGSR	RSMT	GSR	XGSR
4–5	268 803	1.078	1.014	1.014	3.000	3.000	3.000
6–7	91 513	1.211	1.111	1.110	4.217	3.476	3.476
8–10	90 271	1.359	1.247	1.244	5.052	4.633	4.633
11-20	41 836	1.508	1.433	1.431	6.423	6.015	6.015
21-40	17 369	2.159	2.065	2.058	15.323	14.677	13.839

Table 3.6: Average and maximum ratio of the maximum source-sink tree-distance to rectilinear distance for RSMT, GSR and XGSR. For XGSR the maximum path length objective was applied. (Almost the same results are achieved with the total path length objective.)

The average number of iterations (or segment slides) performed by GSR and XGSR is given in Figure 3.16. For XGSR we present data for RSTPWP, but the results for the

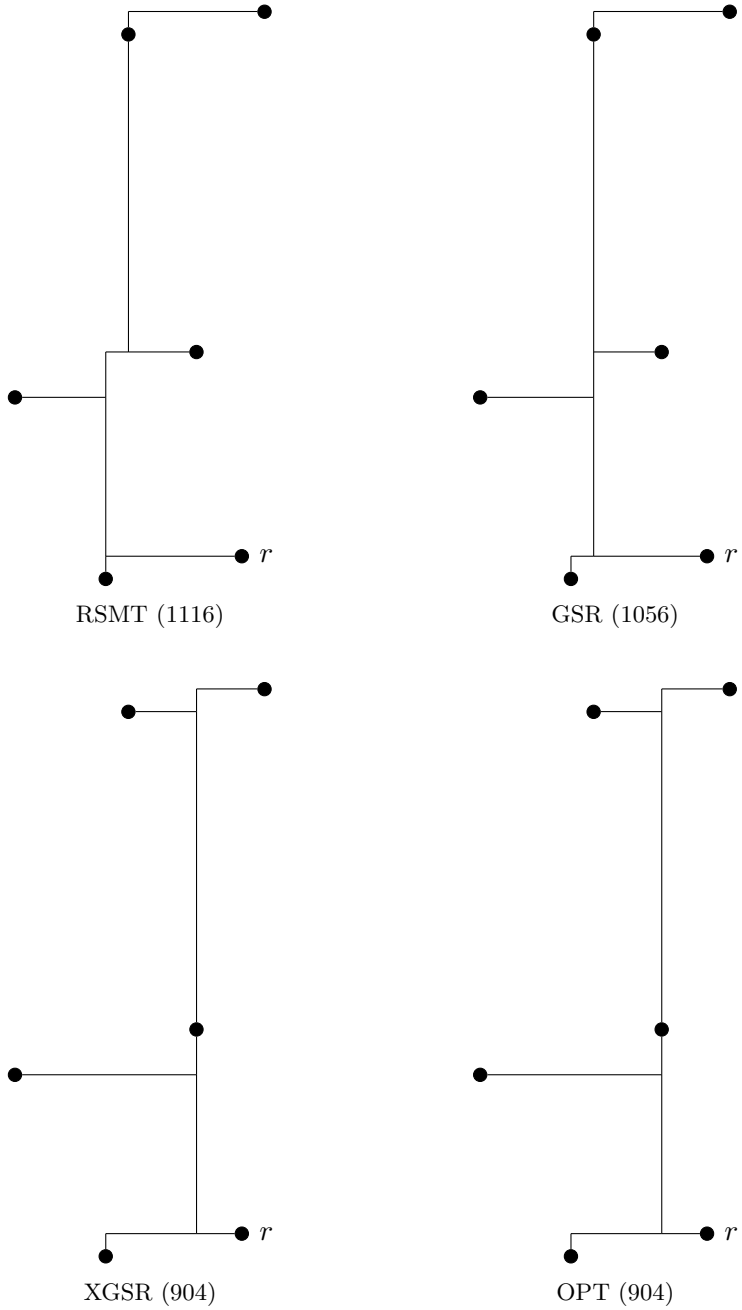


Figure 3.13: Chip net example. RSMT, GSR, XGSR and OPT, optimal solution to RSTPWP, are shown. The weighted sum of path lengths is given for each tree. In this example the XGSR solution (which is optimal) is better than the GSR solution, since XGSR shifts the whole vertical segment rather than just a part of it.

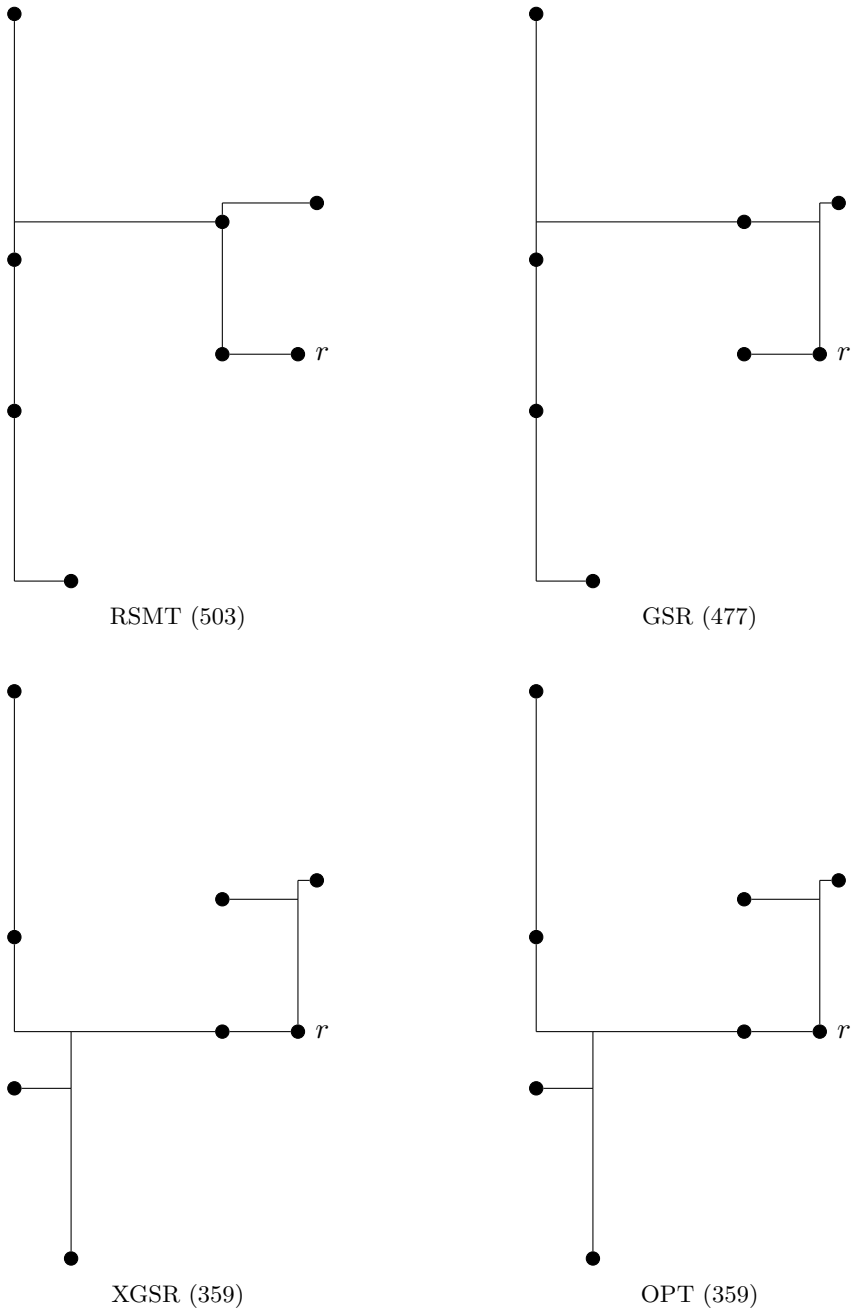


Figure 3.14: Chip net example. RSMT, GSR, XGSR and OPT, optimal solution to RSTPWP, are shown. The weighted sum of path lengths is given for each tree. In this example GSR and XGSR differ since they shift segments in a different order. GSR shifts the rightmost vertical segment to the right in the first move, while XGSR shifts the topmost horizontal segment down in the first move.

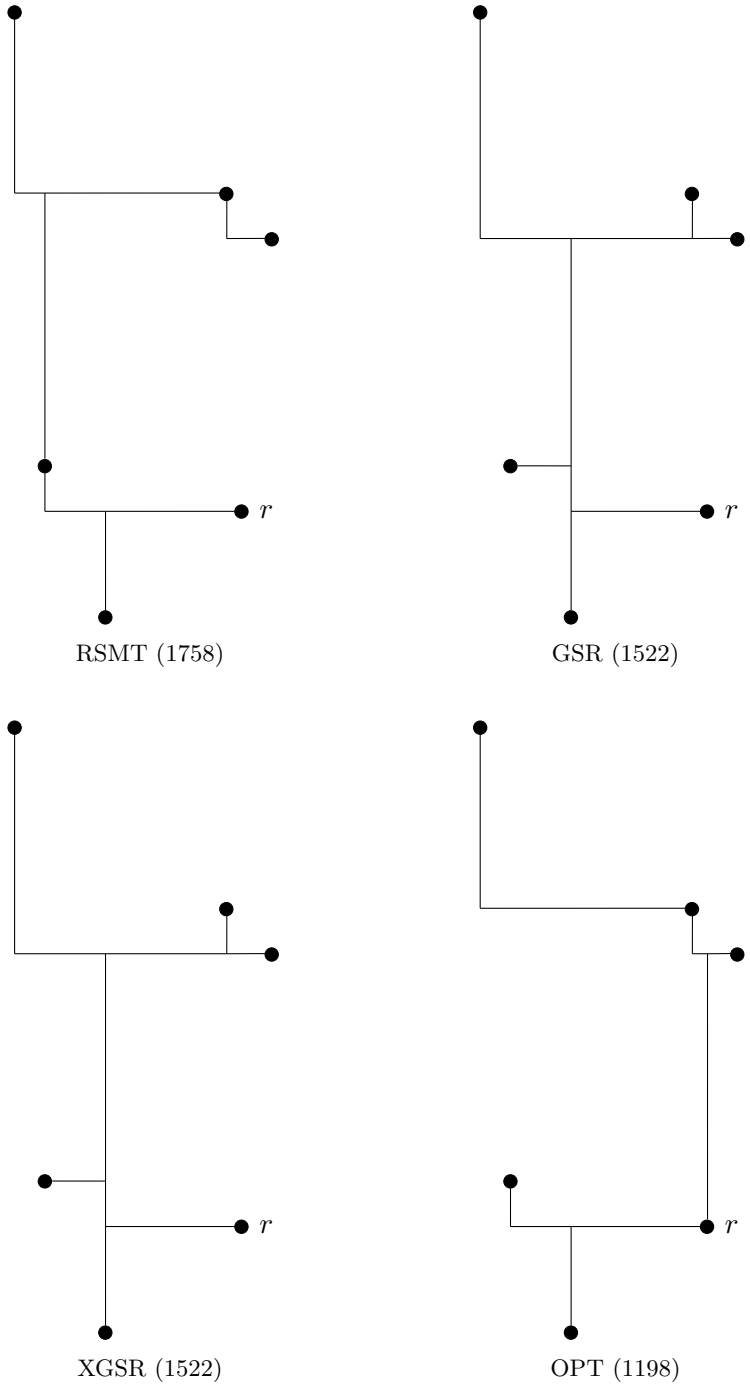


Figure 3.15: Chip net example. RSMT, GSR, XGSR and OPT, optimal solution to RSTPWP, are shown. The weighted sum of path lengths (WPL) is given for each tree. In this example, neither GSR nor XGSR find an optimal solution as they are not able to spend a vertical segment near the source while saving another vertical segment of the same length in the upper part of the tree.

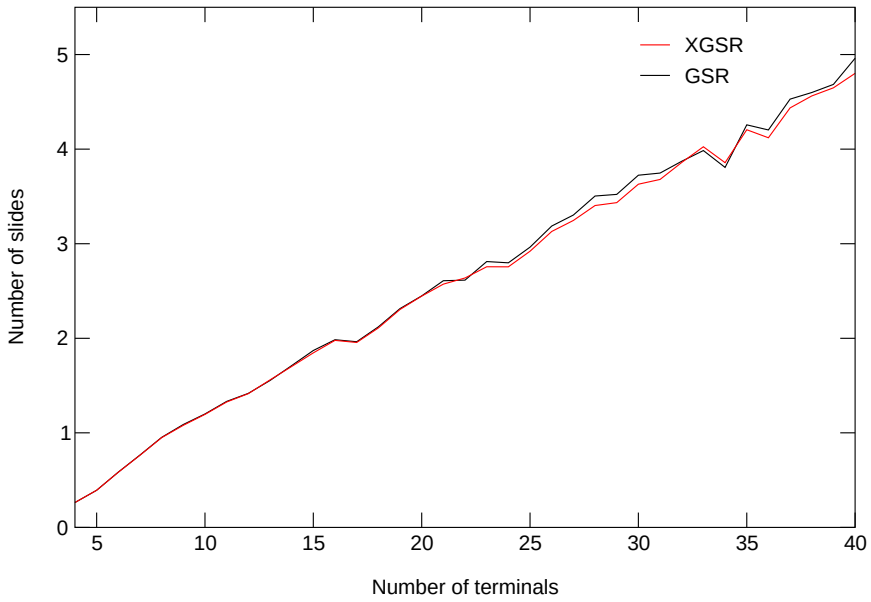


Figure 3.16: Number of segment slides for heuristics (problem RSTPWP).

other secondary objectives are almost identical. Clearly, the upper bound on the number of segment slides given by Lemma 3.10 is overly pessimistic, and in practice the average number of segment slides grows linearly. XGSR performs — as could be expected — slightly fewer segment slides than GSR.

Before we give some details on the running time of GSR and XGSR, it should be noted that the trees obtained by minimizing the weighted sum of path lengths were almost the same as those obtained by minimizing, e.g., the weighted sum of Elmore delays. That is, using a computationally "cheaper" gain function in XGSR reduces the running time without any noteworthy change in the resulting tree.

In Figure 3.17, we present running times for GSR and XGSR (problem RSTPWP). On average, XGSR is about twice as slow as GSR. Obviously, the more sophisticated changes that are made to the tree and the greedy selection of these come at an additional cost. But these extra costs are fairly limited. For the Elmore delay secondary objectives the running times of XGSR are significantly higher. This is due to each segment slide being evaluated in $O(n)$ time. However, the running times are still moderate compared to the computational effort of constructing an RSMT — less than 50 ms even for the largest nets. The total running time for XGSR on *all* 509,792 nets in this study (using the weighted sum of Elmore delays secondary objective) was approximately 5 minutes.

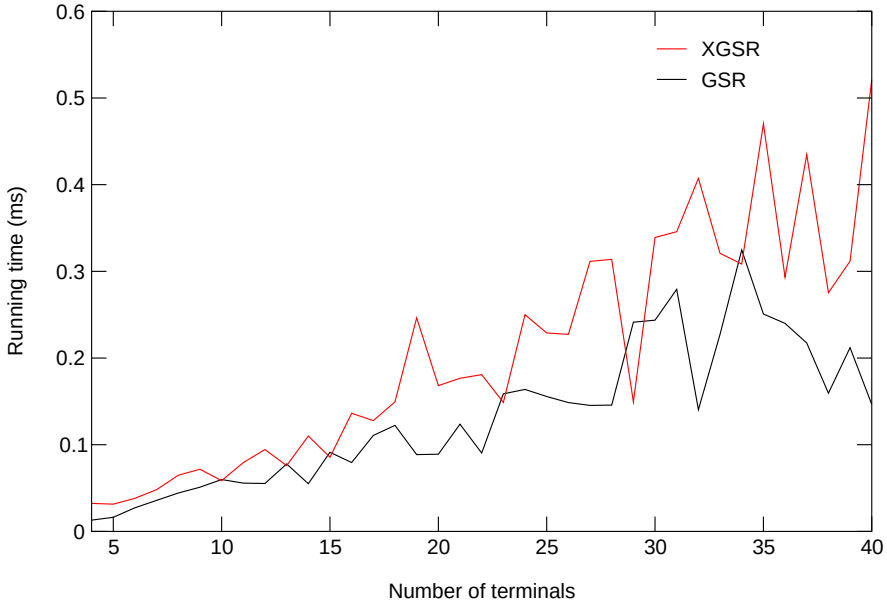


Figure 3.17: Running time for heuristics (problem RSTPWP).

3.2 Minimum Steiner Trees With Obstacles

The second problem we examine in this chapter addresses the rectilinear Steiner tree problem in the presence of obstacles. In contrast to problems discussed in the literature, we allow parts of the Steiner tree to run over obstacles, but with a given length restriction on each of these parts. This problem of length–restricted Steiner minimum trees is inspired by VLSI design as we discuss in Section 3.2.1. We propose an efficient 2-approximation algorithm in Section 3.2.2. Based on structural properties for optimum solutions derived in 3.2.3, we give an improved approximation guarantee in 3.2.4.

3.2.1 Problem Formulation

Throughout this section, an *obstacle* is a connected region in the plane bounded by one or more simple rectilinear polygons such that no two polygon edges have an inner point in common (i.e. an obstacle may contain holes). For a given set $\mathcal{B}$ of obstacles we require the obstacles to be disjoint, except for possibly a finite number of common points (corners of obstacles). By ∂B we denote the boundary of an obstacle B. Every obstacle B is weighted with a factor $w_B \geq 1$; regions not occupied by an obstacle and boundaries of obstacles all have unit weight. These weights are used to compute a weighted tree length which we want to minimize.

Note that we allow trees to run over obstacles, however, we introduce length restrictions for those portions of a tree T which do so. Namely, for each obstacle $B \in \mathcal{B}$ and given

obstacle dependent parameter $L_B \in \mathbb{R}_{\geq 0}$, we require the following for each strictly interior connected component T_B of $(T \cap B) \setminus \partial B$: The (weighted) length $\ell(T_B)$ of such a component must not be longer than the given length restriction L_B. Note that the intersection of a Steiner minimum tree with an obstacle may consist of more than one connected component and that our length restriction applies individually for each connected component. So we can consider the following problem:

Length–Restricted Steiner Tree Problem (LRSTP)	
Instance:	• A set of terminal points Z in the plane; • a set of obstacles $\mathcal{B}$ such that no terminal point lies in the interior of some obstacle; • length restrictions $L_B \in \mathbb{R}_{\geq 0}$ for each obstacle B.
Task:	Find a rectilinear Steiner tree T of minimum (weighted) length such that for each obstacle $B \in \mathcal{B}$, all connected components T_B of $(T \cap B) \setminus \partial B$ satisfy $\ell(T_B) \leq L_B$.

An optimum solution of an instance of the Length-Restricted Steiner Tree Problem is called a *length-restricted Steiner minimum tree (LRSMT)*. Obviously, LRSTP is an NP-hard problem as it contains the rectilinear Steiner minimum tree problem as a special case.

The motivation to study the Length-Restricted Steiner Tree Problem stems from its application in the construction of buffered routing trees in VLSI design (see for example Alpert et al. [2001], Chen, Pedram and Buch [2002], Hu et al. [2002], Alpert et al. [2002], Hrkic and Lillis [2003], Kahng and Liu [2003], Dechu, Shen and Chu [2005]). Consider a signal source r to be connected to a set of sinks S. This gives us an instance of the rectilinear Steiner tree problem with the terminal set $Z := \{r\} \cup S$. A *routing tree* is a tree rooted at the source such that each sink is a leaf. (Note that the latter condition can always be achieved be inserting edges of length zero to sinks which do not fulfill it.) A *buffered routing tree* T is a routing tree with buffers located on its edges. A *buffer* (also called *repeater*) is a circuit which strengthens a signal without logically changing it.

The *subtree driven by a buffer* b *(or the source)* is the maximal subtree of T which is rooted at b and has no internal buffers. The *capacitive load* of a subtree driven by b is the sum of the subtree wire capacitance and the input pin capacitances of its leaves. The source, as well as each type of buffer, respectively, can only drive a certain maximum load. Hence, insertion of buffers in a routing tree may be necessary. One can choose from a set of buffer types with different characteristics. Roughly speaking, a larger and therefore stronger buffer type has a larger input capacitance but causes a smaller delay. (Instead of inserting buffers it is also possible to insert *inverters* to the tree, as long as the signal parity at the sinks does not change.) There might be large macros circuits (such as data caches or processor cores) over which wires can run because there are several wiring layers, but no buffer circuit can be placed in the area covered by the obstacle (there is only one layer in which circuits are realized).

In real world applications, most obstacles are rectangles or of very low complexity. Figure 3.18 gives an impression of the shape, size and distribution of obstacles on typical chip designs.

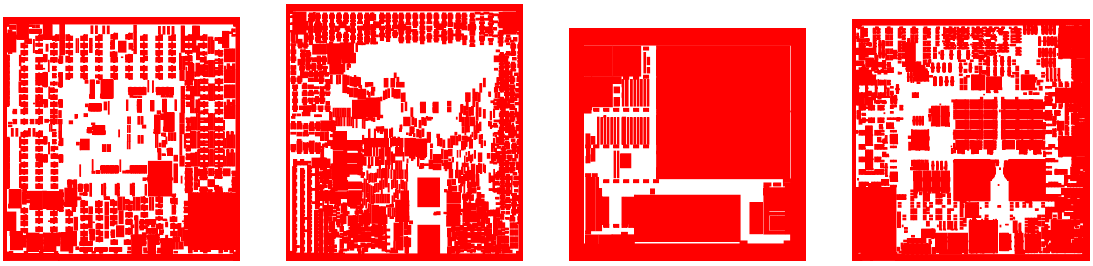


Figure 3.18: Typical shape and distribution of obstacles (macros and other circuits) on current chip designs by IBM.

W.l.o.g., we can assume a unit weight cost function w. If $w_B > 1$ for some obstacle B, then replace L_B by $\frac{L_B}{w_B}$ and the length restriction on obstacle B does not change. For simplicity, we use the same length restriction for all obstacles in our formulation. However, all our results carry over to the case that each obstacle B has an individual length restriction L_B. In particular, by setting $L_B = 0$ for an obstacle, we can model the case that the interior of B must be completely avoided.

Electrical correctness and minimization of power consumption for non-critical nets with respect to timing motivates the minimum cost buffered routing problem, which we shall define now. The *cost* of a buffered routing tree may, for example, be its total capacitance (wire capacitance plus input capacitance of buffers) as a measure for power consumption, or merely the number of inserted buffers.

MINIMUM COST BUFFERED ROUTING PROBLEM	
Instance:	• A source r and sinks $s_1, \ldots, s_k$; • input capacitances of the sinks; • a library of available buffer types with input capacitances and upper load constraints.
Task:	Find a minimum cost buffered routing tree connecting the source to all sinks such that the capacitive load of the source and all inserted buffers is within the given load constraints.

Alpert et al. [2001] give approximation algorithms for the MINIMUM COST BUFFERED ROUTING PROBLEM in a scenario without obstacles for a single buffer type. Their algorithms use approximations of the rectilinear Steiner minimum tree as a subroutine because such trees yield a lower bound on the necessary wiring capacitance. However, in the presence of large obstacles a feasible buffering of a given tree might not be possible any more.

We introduce length restrictions on obstacles to overcome this problem as length restrictions limit the wire capacitance of a connected tree component which runs over some blocked area. This is still a simplified model because the load of a subtree also crucially depends on the input capacitances of its leaves. One way to get rid of this complication would be to require that each internal connected component running over an obstacle has not only a length restriction but also a given upper bound on the number of its leaves (a *fanout restriction*). A second possibility is to introduce a family of length restriction parameters $L_1 \geq L_2 \geq \cdots \geq L_i \geq \ldots$ with the interpretation that for a component T_B with i leaves the length constraint $\ell(T_B) \leq L_i$ applies. In both models it is then always possible to insert additional buffers into a tree such that no load violations occur. Moreover, the length restriction parameter has to be chosen carefully with respect to the available buffer library and technology parameters, for example unit wire capacitance.

As a first step in extending the approximation results for the Minimum Cost Buffered Routing Problem to the case with obstacles, we look for good approximation algorithms for LRSTP with one of these additional types of restrictions. For simplicity of presentation in this section we consider only the version of LRSTP as defined in the Length–Restricted Steiner Tree Problem. However, fanout restrictions as well as fanout dependent length restrictions are easily incorporated into the algorithmic approach and change none of the results with respect to approximation guarantees and asymptotic running times.

Given a set of terminals in the plane without obstacles, the shortest rectilinear Steiner tree can be approximated in polynomial time to any desired accuracy using an approximation scheme by Arora [1998] or Mitchell [1999].

An obstacle which has to be avoided completely is referred to as *hard obstacle*. Most previous work dealing with obstacles considered hard obstacles. We define the term *Hanan grid* similar to the previous section: given a finite point set Z in the plane and a set of obstacles $\mathcal{B}$, the Hanan grid is obtained by constructing a vertical and a horizontal line through each point of Z and a line through each edge used in the description of the obstacles (Hanan [1966]). The importance of the Hanan grid lies in the fact that it contains a rectilinear Steiner minimum tree. Ganley and Cohoon [1994] observed that the rectilinear Steiner tree problem with hard obstacles can be solved on a slightly reduced Hanan grid. An approximation factor of 2 can easily be obtained by computing the minimum spanning tree on the Hanan grid (after deleting edges on obstcales). Several more variants and generalizations of the Steiner tree problem are solvable on the Hanan grid; for a survey see the catalog by Zachariasen [2001b]. As a consequence, all these variants can be solved as instances of the Steiner tree problem in graphs. (Given a connected graph $G = (V, E)$, a length function ℓ, and a set of terminals $Z \subseteq V$, a *Steiner tree* T for Z is a tree in G containing all vertices of Z. T is a *Steiner minimum tree* for Z if its length is minimum among all Steiner trees for Z.) The best available polynomial-time approximation algorithm for the Steiner problem in general graphs has an approximation guarantee $\alpha = 1 + \frac{\ln 3}{2} \approx 1.55$; see Robins and Zelikovsky [2005]. Recently, Müller-Hannemann and Tazari [2007] proposed a near linear time approximation scheme for Steiner minimum trees in the presence of hard obstacles.

Their result applies for the Euclidean metric and also for all uniform orientation metrics, i.e. particularly the rectilinear and octilinear metrics.

Miriyala, Hashmi and Sherwani [1991] solved the case of a single rectangular hard obstacle to optimality and approximated the Steiner tree for a set of rectangular hard obstacles provided that all terminals lie on the boundary of an enclosing rectangle, a so–called switchbox instance. Slightly more general, a switchbox instance with a constant number of rectangular hard obstacles can be solved exactly in linear time as was shown by Chiang, Sarrafzadeh and Wong [1992].

Rectilinear shortest path problems with hard obstacles and weighted versions have achieved a lot of attention. The strongest theoretical result for this kind of problems has been given by Chen, Klenk and Tu [2000] who provide a data structure to answer two-point shortest rectilinear path queries for arbitrary weighted, rectilinear obstacles. Such a data structure can be constructed in $O(n^2 \log^2 n)$ time and space and allows to find a shortest path in $O(\log^2 n + k)$ time, where n is the number of obstacle vertices and k denotes the number of edges on the output path. Many efficient obstacle-avoiding rectilinear Steiner tree constructions have been proposed in literature. We refer to the recent work by Lin et al. [2007].

Rectilinear shortest path problems with length restrictions have first been considered by Müller-Hannemann and Zimmermann [2003] who showed that these problems can easily be solved to optimality (see also Section 3.2.2).

3.2.2 A 2-Approximation

We now show that the Length–Restricted Steiner Tree Problem can be approximated within a factor of 2 in polynomial time. In this connection, instances of the Length–Restricted Steiner Tree Problem with only two terminals, i.e. the Length–Restricted Shortest Path Problem (LRSPP), are of special interest for several reasons. In contrast to the general Length–Restricted Steiner Tree Problem, such instances can be solved to optimality in polynomial time. Müller-Hannemann and Zimmermann [2003] analyzed the LRSPP and used it as a subroutine for constructing slack-optimized buffer and inverter trees. An efficient solution to the LRSPP is the basis for our 2-approximation of the Length–Restricted Steiner Tree Problem. We summarize the most important properties of the LRSPP for later use.

Lemma 3.13 (Müller-Hannemann and Zimmermann [2003])**.** *Given two terminals s and t, a set of obstacles $\mathcal{B}$ and a length restriction L, there is an optimal length–restricted s–t–path using only Hanan grid edges.*

This property does not hold for Steiner trees. A small counter–example with three terminals is shown in Figure 3.19.

For a set $\mathcal{B}$ of obstacles described by $m_{\mathcal{B}}$ edges (in total) and a set Z of terminals, the size of the associated Hanan grid may have as many as $O\left((m_{\mathcal{B}} + |Z|)^2\right)$ vertices. For

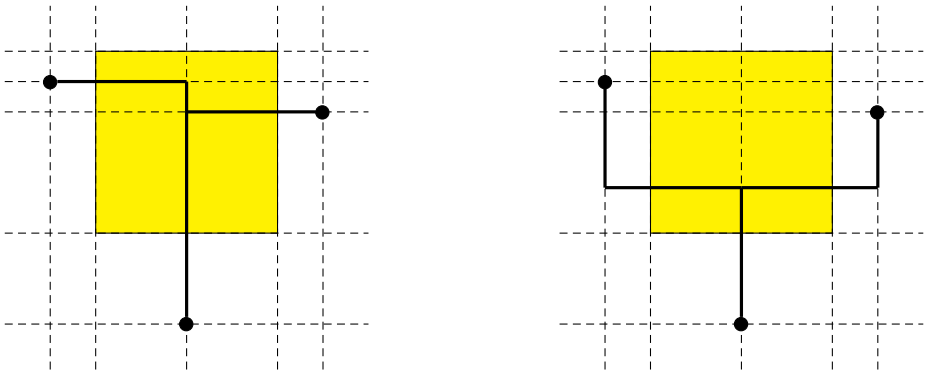


Figure 3.19: A small rectilinear Steiner tree instance with three terminals: A Steiner minimum tree without a length restriction lies on the Hanan grid (left), whereas a Steiner minimum tree with such a restriction on the rectangular obstacle does not always lie on the Hanan grid (right).

many applications, see again Figure 3.18, this is by far too pessimistic. Therefore, in the following we use the actual size of the Hanan grid as a measure of our algorithm's complexity.

Lemma 3.14 (Müller-Hannemann and Zimmermann [2003])**.** *Given a Hanan grid with n vertices, there is a graph G with $O(n)$ vertices and edges in which all s–t–paths are feasible length–restricted paths and which contains an optimal length–restricted s–t–path for any pair s, t of terminals. Such a graph can be constructed in $O(n)$ time.*

Lemma 3.15 (Müller-Hannemann and Zimmermann [2003])**.** *Given a weighted rectilinear subdivision of the plane with an associated Hanan grid of size n where a subset of the regions are obstacles, the weighted shortest path problem with a given length restriction L can be solved by Dijkstra's algorithm in $O(n \log n)$ time.*

To obtain a 2-approximation for LRSTP, we use well-known 2-approximations for the Steiner tree problem in graphs. Consider an instance $G = (V, E, \ell; Z)$ of the Steiner tree problem in graphs, where V is the vertex set and E the edge set of a connected graph with edge length function ℓ, and Z denotes the terminal set. The *distance network* $N_d = (Z, E_Z, d)$ is a complete graph defined on the set of terminals Z: for each pair $z_1, z_2 \in Z$ of terminals there is an edge with exactly the length $d(z_1, z_2)$ of a shortest z_1–z_2-path in G.

For every vertex $z \in Z$ let $N(z)$ be the set of vertices in V that are closer to z (with respect to d) than to any other vertex in Z. More precisely, we partition the vertex set V into sets $\{N(z) : z \in Z\}$ with $N(z_1) \cap N(z_2) = \emptyset$ for $z_1, z_2 \in Z, z_1 \neq z_2$ with the property

$$v \in N(z_1) \Rightarrow \; d(v, z_1) \leq d(v, z_2) \text{ for all } z_2 \in Z,$$

resolving ties arbitrarily. The *modified distance network* $N_d^* = (Z, E^*, d^*)$ is a subgraph of N_d defined by

$$E^* := \{(z_1, z_2) \mid z_1, z_2 \in Z \text{ and there is an edge } (u, v) \in E \text{ with } u \in N(z_1), v \in N(z_2)\},$$

and

$$d^*(z_1, z_2) := \min\{d(z_1, u) + \ell(u, v) + d(v, z_2) \mid (u, v) \in E, u \in N(z_1), v \in N(z_2)\},$$

for $z_1, z_2 \in Z$.

Mehlhorn [1988] showed that (a) every minimum spanning tree of N_d^* is also a minimum spanning tree of N_d, and that (b) N_d^* can be computed in $O(n \log n + m)$ time (namely, by one single-source shortest path computation plus bucket sorting).

Algorithm 3.3: MEHLHORN'S ALGORITHM

Input: Graph $G = (V, E, \ell; Z)$
Output: A Steiner Tree T for G

1 Compute the modified distance network N_d^* for $G = (V, E, \ell; Z)$;
2 Compute a minimum spanning tree T_d^* in N_d^*;
3 Transform T_d^* into a Steiner tree T for G by replacing every edge of T_d^* by its corresponding shortest path in G;
4 Return T;

Given an instance $G = (V, E, \ell; Z)$ of the Steiner tree problem in graphs with $n = |V|$ and $m = |E|$, MEHLHORN'S ALGORITHM computes a Steiner tree with a performance guarantee of 2 in $O(n \log n + m)$ time (Mehlhorn [1988]).

Theorem 3.16. *Length–restricted Steiner trees can be approximated with a performance guarantee of* 2 *in* $O(n \log n)$ *time.*

Proof. Using the results of the previous section, we can efficiently build up the modified Hanan grid G' from Lemma 3.14. We apply MEHLHORN'S ALGORITHM to G' and obtain a performance guarantee of 2. The claim on the running time follows immediately as $O(m) = O(n)$. Finally, the obtained tree is feasible, as no tree in G' violates any length restriction. □

We finish this section by showing that the approximation guarantee for MEHLHORN'S ALGORITHM is tight. The *Steiner ratio* is the least upper bound on the length of a minimum spanning tree in the distance network divided by the length of a minimum Steiner tree for all instances of the Steiner tree problem. We extend this notion to length restrictions. The *length–restricted Steiner ratio* is the least upper bound on the length of a minimum spanning tree in the distance network containing a length–restricted shortest path between any pair of terminals divided by the length of an LRSMT for all instances of the LENGTH–RESTRICTED STEINER TREE PROBLEM. In the case without obstacles the

Steiner ratio is $\frac{3}{2}$ as was shown by Hwang [1976]. However, in the scenario with obstacles and length restrictions the corresponding Steiner ratio is worse, namely 2, and therefore not better than for the Steiner tree problem in general graphs.

Lemma 3.17. *The length–restricted Steiner ratio is* 2.

Proof. Clearly, the Steiner ratio is not larger than 2 by the approximation guarantee from Theorem 3.16. To prove that the Steiner ratio is exactly 2, we provide a class of input instances for which the length of the minimum spanning tree in the distance network (based on length–restricted shortest paths) becomes arbitrarily close to twice the length of a length–restricted Steiner minimum tree.

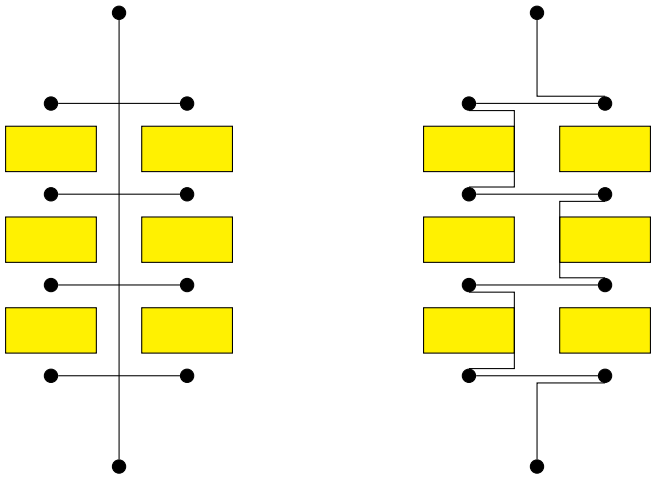


Figure 3.20: Schematic view on the instance class with a Steiner ratio of 2: the LRSMT (left) and a minimum spanning tree in the distance network based on length–restricted shortest paths (right).

For a given length restriction L, we define an instance with $k = 2r$ terminals as follows, see Figure 3.20. The terminal positions are $p_1 = (0, 0)$, $p_{2i} = (-rL, iL+3i)$ for $i = 1, \ldots, r-1$ and $p_{2i+1} = (rL, iL + 3i)$ for $i = 1, \ldots, r - 1$ and $p_k = (0, rL + 3r)$. There are $k - 2$ rectangular obstacles $B_1, \ldots B_{k-2}$. We specify each rectangle by giving two opposite corner points, namely the lower left corner ℓc_i and the upper right corner rc_i for $i = 1, \ldots k - 2$. Precisely, we define $\ell c_{2i-1} = (-2rL, iL + 3i + 1)$, $rc_{2i-1} = (-1, iL + 3i + L + 2)$ and $\ell c_{2i} = (1, iL + 3i + 1)$, $rc_{2i} = (2rL, iL + 3i + L + 2)$, for $i = 1, \ldots, r - 1$. The length of a length–restricted Steiner minimum tree $LRSMT$ is

$$|LRSMT| = 2r^2L - rL + 3r.$$

The length of a minimum spanning tree in the distance network, however, is

$$|MST| = 4r^2L - 3rL + r + 4.$$

Hence

$$\lim_{r\to\infty} \frac{|MST|}{|LRSMT|} = 2.$$

□

3.2.3 The Structure of Length–Restricted Steiner Minimum Trees

The purpose of this section is to characterize the structure of length–restricted Steiner minimum trees. In particular, we define a finite graph (a variant of the Hanan grid) which always contains an optimal solution.

Let Z be a set of terminals with $|Z| \geq 4$ and T be a Steiner minimum tree for Z. Then T is called a *fir tree* (see Figure 3.21) if and only if every terminal has degree one in T and one of the following two conditions is satisfied (possibly after reflection and/or rotation):

1. All Steiner points lie on a vertical line and every Steiner point is adjacent to exactly one horizontal edge, and these horizontal edges alternatingly extend to the left and to the right. The topmost Steiner point is adjacent to a vertical edge ending in a terminal, the lowest Steiner point is adjacent to a vertical edge either ending in a terminal or at a corner. In the latter case, the horizontal edge extends to the opposite side than the horizontal edge of the lowest Steiner point. (Types (I) and (II) in Figure 3.21)

2. All but one Steiner point lie on a vertical line. Every Steiner point but the exceptional one is adjacent to exactly one horizontal edge which alternatingly extend to the left and to the right and ends in a terminal. The exceptional Steiner point is incident to two horizontal edges, one of which ends in a terminal. The other edge is a connection to the lowest Steiner point on the vertical line by a corner from the opposite side than the horizontal edge of the lowest Steiner point. Finally, the topmost and the exceptional Steiner point are both adjacent to a vertical edge that extend upwards and downwards, respectively, and ends in a terminal. (Type (III) in Figure 3.21)

The vertical line connecting all or all but one Steiner point is called the *stem* of the fir tree, all horizontal edges are called *legs*. An edge is called *interior* with respect to some obstacle B if it is contained in B and does not completely lie on the boundary of B.

Lemma 3.18. *Let Z be a terminal set on the boundary of an obstacle B such that in every length–restricted Steiner minimum tree for Z*

1. *all terminals are leaves, and*

2. *all tree edges are interior edges with respect to B.*

Then there exists a length–restricted Steiner minimum tree T for Z such that it either is a fir tree or has one of the following five shapes (possibly after reflection and/or rotation):

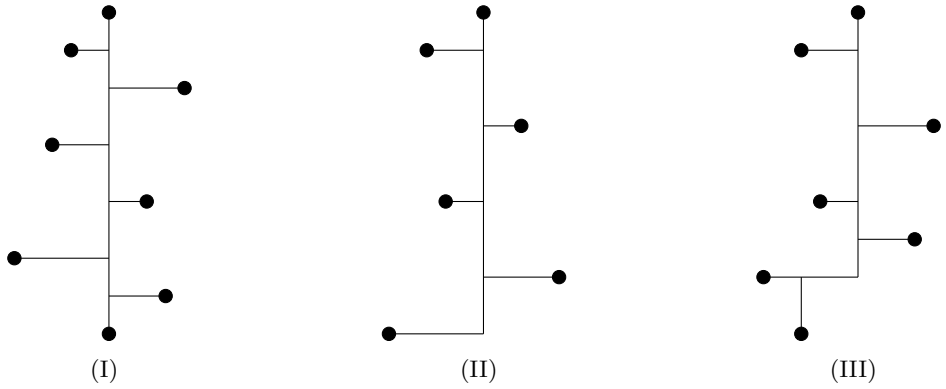
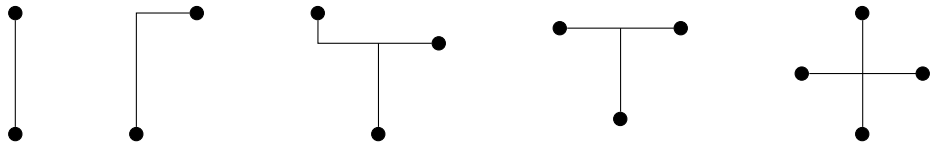


Figure 3.21: The three different types of fir trees.



Proof. Almost the same characterization as claimed in the lemma is well-known for rectilinear Steiner trees without obstacles, see, for example, the monograph by Prömel and Steger [2002], Chapter 10. The general proof idea is to start with an arbitrary optimal tree and then to perform a series of modifications which does not increase the tree length until the tree has the claimed structure (or some terminal is no leaf any more and so violates our assumptions). The only small difference is that this lemma assumes that every optimal tree lies strictly inside a bounded region. So this gives another reason to stop the series of modifications but clearly does not change the structure of the resulting tree if the instance satisfies the premises of this lemma. □

Trees of the fourth and fifth shape in Lemma 3.18 are called *T-shaped* and *cross-shaped*, respectively. The two horizontal edges of a T-shaped tree are its *stem*. Note that the lemma asserts the following property: for a set of terminals located on the boundary of an obstacle there is either a LRSMT of the described structure or the tree decomposes into at least two instances with fewer terminals.

Based on the structural insights from the previous lemma, we can now define a variant of the Hanan grid, which we call *augmented Hanan grid*.

Definition 3.19 (augmented Hanan grid)**.** *Given a set Z of points in the plane, a set of rectilinear obstacles $\mathcal{B}$ and a length restriction $L \in \mathbb{R}_{\geq 0}$, the* augmented Hanan grid *is the graph induced by the following lines:*

1. *for each point $(x, y) \in Z$, there is a vertical and a horizontal line going through (x, y),*
2. *each edge of each obstacle is extended to a complete line, and*

3. *for each obstacle $B \in \mathcal{B}$, include a line going through the stem of all those T-shaped trees, and all those fir trees of type (I) or of type (III) which have exactly length L, have only interior edges, and an arbitrary, odd set of points located on the boundary of B as their terminals.*

From its definition it is not clear whether the augmented Hanan grid has polynomial size and can be efficiently constructed. For instances with rectangular obstacles both properties hold: We observe that we need at most four additional lines per obstacle and that we can find all candidate lines easily.

Lemma 3.20. *If all obstacles in $\mathcal{B}$ are rectangles, then we have to add at most $O(|\mathcal{B}|)$ additional lines to the ordinary Hanan grid to obtain the augmented Hanan grid.*

Proof. Consider one rectangular obstacle $R \in \mathcal{B}$ of dimension $a \times b$. Let us fix one of its four sides s, say the bottom side of length a. If $L \leq a$, no T-shaped tree can have its leg on side s. If $a < L < a + b$, the only position of the stem of any possible T-shaped tree of length L with its leg being located somewhere on s is exactly $L - a$ units above side s. Note that no T-shaped tree is possible if $L \geq a+b$. Conversely, fir trees are only possible if $L > a + b$. Consider a fir tree of the following kind: its stem runs parallel to s and x units above s, $\ell_b \geq 2$ of its legs are pointing downwards and $\ell_b - 1$ legs are pointing upwards. The length of such a fir tree T is a function of ℓ_b and x, namely $L(T, \ell_b, x) = a + (\ell_b - 1)b + x$. As $0 \leq x < b$, there can be at most one solution (ℓ_b, x) of $L = L(T, \ell_b, x)$. Note that the case with more legs pointing upwards than downwards is counted for the upper side of the rectangle. Hence, we have to include at most one additional line for each side of R. □

Similarly, but with more involved counting arguments, one can show that the size of the augmented Hanan grid is still polynomially bounded if each obstacle can be described by at most k edges, where k is some given constant.

Next we prove that the augmented Hanan grid has the desired property to contain an optimal solution.

Lemma 3.21. *The* Length–Restricted Steiner Tree Problem *has an optimal solution which lies completely on the augmented Hanan grid.*

Proof. Choose T among all optimal trees such that (a) T has the structure described in Lemma 3.18 inside obstacles, and (b) T has the fewest number of (inclusion-maximal) segments q which do not lie on the augmented Hanan grid among all those optimal trees which already fulfill (a). Assume $q > 0$ and let s be a non-Hanan segment. Without loss of generality we assume that s is a horizontal segment. Consider first the case that one endpoint of s, say the left endpoint p_ℓ, is the corner point of a fir tree of type (II) lying within some obstacle B, and that s is not parallel to the stem (which means that the fir tree is rotated). Denote by p_B the nearest intersection point of segment s with B as seen from the left endpoint of s. In such a situation, we can modify the fir tree inside B such that it has the same total length, does not use the edge $\overline{p_\ell p_B}$ and does not create a new

non-Hanan segment. A similar modification may be necessary for the right endpoint p_r of s. After that modification, we can be sure that no endpoint of the remaining part of s ends as a leg in a corner point of a fir tree of the second type lying within some obstacle B.

Now let m_u, m_b be the number of edges which end in s from above, and from below, respectively. If $m_u \neq m_b$, say $m_u > m_b$, then we can slide segment s upwards and thereby strictly reduce the length of T, which contradicts its optimality. Thus $m_u = m_b$. Hence, we can slide segment s upwards until it first hits a line of the augmented Hanan grid (or until it overlaps with a preexisting segment of T, which would contradict its minimality and so does not occur). This modification does not change the length of the tree, but property (a) may now be violated. However, it is easy to modify the tree further such that it fulfills property (a) inside obstacles and keeps its minimum length without introducing any new non-Hanan segment. Hence, the modified tree contradicts our initial choice of T.

It remains to argue that the sliding operation does not violate the length restriction for some obstacle. Assume that some part of s, say s', belongs to a subtree which causes a length violation for some obstacle B'. Then s' belongs to a fir tree. If it is the stem of the tree, then the sliding operation would either not change the length of the fir tree (namely, if the fir tree has an even number of terminals) or it would stop at the latest at the additional line of the Hanan grid which has been inserted for this type of fir tree and no violation would occur. Otherwise, if it is not part of the stem, the length of the tree remains invariant (or become even shorter if two edges of the fir tree collapse). □

3.2.4 Improved Approximation for Rectangular Obstacles

In this section, we focus on improved approximation guarantees for instances where all obstacles are rectangles. The basic idea is to construct an instance of the Steiner tree problem in graphs with the property that a Steiner tree in the constructed graph immediately translates back to a feasible length–restricted rectilinear Steiner tree. In addition, the construction is designed to guarantee that the Steiner minimum tree in the graph is not much longer than the length–restricted Steiner minimum tree. This is inspired by approximation algorithms for rectilinear Steiner trees which rely on k-restricted Steiner trees (Zelikovsky [1992], Berman and Ramaiyer [1994]). We say that a Steiner tree is a *k-restricted* Steiner tree if each full component spans at most k terminals.

To formalize this general idea, we do the following. Given an instance of the Length–Restricted Steiner Tree Problem with rectangular obstacles and an integer $k \geq 2$, we construct the graph G_k in three steps:

1. build up the augmented Hanan grid;

2. delete all vertices and incident edges of the augmented Hanan grid that lie in the strict interior of some obstacle;

3. for each obstacle R, consider each c-element subset of distinct vertices on the boundary of R for $c = 2, \ldots, k$. Compute an (unrestricted) Steiner minimum tree for such a vertex set. If the length of this tree is less or equal to the given length bound L and if the tree has no edge lying on the boundary of R, then add this tree to the current graph and identify the leave vertices of the tree with the corresponding boundary vertices of R.

The following lemma shows that the construction of G_k can be done efficiently. In particular, in Step 3 we do not have to consider all c-element subsets of vertices on the boundary of a rectangle explicitly. It suffices to enumerate only those subsets of vertices which have Steiner minimum trees according to Lemma 3.18.

Lemma 3.22. *If the augmented Hanan grid has n vertices, then*

(a) G_2 has at most $O(n)$ vertices and edges, and can be constructed in $O(n)$ time, and

(b) G_k has at most $O(n^{k-2})$ vertices and edges and can be constructed in $O(n^{k-2})$ time for any $k \geq 3$.

Proof. For an instance with c terminals located on the boundary of a rectangle, the Steiner minimum tree can be found in time $O(c)$ as shown by Agarwal and Shing [1990] and Cohoon, Richards and Salowe [1990]. That is, for a given tree with $c \leq k$ terminals we can verify in constant time whether it is optimal and satisfies the given length bound L. It remains to show how many trees have to be examined for an arbitrary rectangle R with n_R grid vertices. As shown in Lemma 3.14, there are only $O(n_R)$ many paths (i.e. trees with two terminals) to be considered, which already implies the claim for $k = 2$. Trees with three terminals can be assumed to be T-shaped. A T-shaped tree is uniquely characterized by its Steiner point and by the direction of its leg. This implies that there are at most $O(n_R)$ many trees with three terminals. A Steiner tree with four terminals is either cross-shaped or a fir tree. Clearly, there are only $O(n_R)$ many cross-shaped trees. Fir trees with $c \geq 4$ terminals are completely determined by specifying its $c - 2$ Steiner points and the direction of the first leg. (Given the Steiner point locations, there are only two possible directions.) This implies that there are at most $O(n_R^{c-2})$ possible fir trees. Summing up these possibilities over all rectangles and all $c \leq k$ yields $O(n^{k-2})$ additional vertices and edges in G_k, $k \geq 3$. □

The following lemma yields the basis for our improved approximation guarantee.

Lemma 3.23. *Let R be a rectangular obstacle and Z a set of terminals on its boundary. Then G_3 contains a Steiner minimum tree that is at most $\frac{5}{4}$ times as long as the length–restricted Steiner minimum tree for Z. For $k \geq 4$, G_k contains a Steiner minimum tree that is at most $\frac{2k}{2k-1}$ times as long as the length–restricted Steiner minimum tree for Z.*

Proof. Let $LRSMT$ be a length–restricted Steiner minimum tree for the terminal set Z. By Lemma 3.21, we may assume that $LRSMT$ lies on the augmented Hanan grid. Thus,

we may further assume that $LRSMT$ is a full Steiner tree and contains no edge on the boundary of R, as otherwise $LRSMT$ can be decomposed into smaller instances which fulfill the hypotheses of this lemma. If $|Z| \leq k$, then G_k contains $LRSMT$ by construction. Otherwise, by Lemma 3.18, $LRSMT$ must be a fir tree with $|Z| > k$ terminals or a cross-shaped tree if $|Z| = 4$ and $k = 3$.

Zelikovsky [1992] and Berman and Ramaiyer [1994] defined four 3-restricted Steiner trees that each span the same terminals as $LRSMT$ with a total length five times $|LRSMT|$. Thus, the shortest of the four trees has length at most $\frac{5}{4}|LRSMT|$. For $k \geq 4$, Borchers et al. [1998] are able to define a collection of $2k - 1$ k-restricted full Steiner trees with total length at most $2k$ times the length of any full tree. As $LRSMT$ is length-feasible, all full components used in these k-restricted Steiner trees are also length-feasible as they are strictly shorter. The lemma follows, since G_k contains all k-restricted full Steiner trees used in these constructions. □

Combining our previous observations, we obtain the main result of this section.

Theorem 3.24. *Using a polynomial-time approximation algorithm for the ordinary Steiner tree problem in graphs with an approximation guarantee α, we obtain a family of polynomial-time approximation algorithms for the* Length–Restricted Steiner Tree Problem *subject to rectangular obstacles with performance guarantee $\frac{5}{4}\alpha$ and $\frac{2k}{2k-1}\alpha$ for any $k \geq 4$, respectively.*

Proof. Our approximation algorithms solve the Steiner tree problem in graphs on G_k, for $k \geq 3$. For rectangular obstacles, the augmented Hanan grid has polynomial size and can be constructed in polynomial time. Hence, the algorithms also run in polynomial time.

Let $LRSMT$ be a length–restricted Steiner minimum tree. By Lemma 3.21, we may assume that $LRSMT$ lies on the augmented Hanan grid. Hence, all edges of $LRSMT$ which are not interior edges of obstacles are also edges of G_k. Now consider an arbitrary obstacle R such that the optimal tree restricted to this obstacle $T_R := LRSMT \cap R$ has edges interior to R. The edges of T_R are not necessarily represented in G_k. However, for each connected component of T_R we may consider its leaves Z as terminals of a Steiner tree instance on obstacle R. By Lemma 3.23, the length of a Steiner minimum tree in G_3 for the set Z is at most $\frac{5}{4}$ times as long as the length of the corresponding component of T_R. Similarly, the length of a Steiner minimum tree in G_k, $k \geq 4$, for the set Z is at most $\frac{2k}{2k-1}$ times as long as the length of the corresponding component of T_R. Replacing the subtrees T_R by an optimal solution in G_k, gives an overall tree in G_k with the claimed length bound. □

We note again that a similar result holds for a scenario with general obstacles provided each obstacle is bounded by only a constant number of edges.

Finally, we would like to mention that concepts used in this section are also applied to prove similar approximability results for octilinear Steiner trees (Müller-Hannemann and Schulze [2006]).

Chapter 4

Shortest Paths

The shortest path problem is one of the most elementary, important and well-studied algorithmic problems in graph theory. In a very general setting it can be formulated as follows: for a given digraph $G = (V(G), E(G))$, edge lengths $c : E(G) \to \mathbb{R}$ and vertex sets $S, T \subseteq V(G)$, find a path P in G from source S to target T whose total length $c(E(P)) = \sum_{e \in E(P)} c(e)$ is minimum. An optimum solution to this problem may not exist if there is a cycle of negative total length. Therefore, we assume that c is *conservative*, i.e. G does not contain a cycle of negative total length. The shortest path problem is a special case of the transshipment problem (Orden [1956]) and can be solved by means of linear programming (Dantzig [1957]). Hence, finding a shortest path is polynomially solvable. If not defined otherwise, let $n := |V(G)|$ and $m := |E(G)|$ throughout this chapter.

A framework used by many algorithms to solve shortest path problems may be stated in terms of a label-setting based algorithm as follows: every vertex $v \in V(G)$ is assigned a distance value $\gamma(v)$. Initially, $\gamma(v) := 0$ for all $v \in S$, and $\gamma(v) := \infty$ for all other vertices. Now, in each step of the algorithm, an edge $(u, v) \in E(G)$ with $\gamma(v) > \gamma(u) + c((u, v))$ is chosen and $\gamma(v)$ is set to $\gamma(u) + c((u, v))$. The algorithm stops if a target vertex receives its final distance value or none of the distance values can be improved. There are many variants of that scheme to improve the running time; see Section 4.1. The main goal is to prune the number of label adjustments and examine only a small portion of the original graph.

Note that our formulation does not explicitely construct a shortest path. But this information can easily be retrieved by keeping track of predecessor information during the algorithm.

In this chapter we restrict the shortest path problem to non-negative integral edge length. So, we consider the following problem:

Integral Shortest Path Problem	
Instance:	• A directed graph $G = (V(G), E(G))$; • edge lengths $c : E(G) \to \mathbb{Z}_{\geq 0}$; • vertex sets $\emptyset \neq S \subseteq V(G)$ and $T \subseteq V(G)$.
Task:	If T is non-empty and reachable from S in G, find the length of a shortest path from S to T in G, i.e. a path P of minimum total length $c(E(P))$. Otherwise, compute the length of shortest paths from S to all vertices in $V(G)$ which are reachable from S.

Using an algorithm by Johnson [1977] based on Ford [1956] and Bellman [1958] it is possible to convert conservative edge length $c : E(G) \to \mathbb{R}$ to non-negative edge length in $O(n(n+m)\log n)$ time while preserving shortest paths.

The Integral Shortest Path Problem appears in countless practical applications, for example in motion planning (Fitch, Butler and Rus [2001]), road networks (Klunder and Post [2006]), modeling timetable information (Müller-Hannemann et al. [2007]), and network routing (Medhi and Ramasamy [2007]). See also Gallo and Pallottino [1988] and Cherkassky, Goldberg and Radzik [1996].

The motivation for the research presented here originates from the routing problem in VLSI design where the time needed to complete the full design process is one of the most crucial issues to be addressed. In typical problem instances today millions of shortest paths have to be found in graphs with billions of vertices. Therefore, no algorithm which does not heavily exploit the specific instance structure can lead to an acceptable running time.

Various methods are proposed in literature to solve the Integral Shortest Path Problem. Some of them are of pure theoretical interest, others are derived from a practical background. We give an overview on techniques to find shortest paths in Section 4.1. In Section 4.2 we present a generic version of our new algorithm which can be viewed as a generalization of Dijkstra's algorithm (Dijkstra [1959]). Section 4.3 is devoted to two applications of our strategy which reduce the overall running time of VLSI routing significantly as shown with experimental results in Section 4.4. Main results of this chapter have been recently published by Peyer, Rautenbach and Vygen [2006].

4.1 Shortest Paths Algorithms

In Section 4.1.1, we introduce commonly used speed-up techniques for the Integral Shortest Path Problem in general graphs. Since our special focus lies on grid graphs, main results on grid graphs are presented in Section 4.1.2.

4.1.1 General Graphs

Shimbel [1955] was probably the first who described a polynomial-time algorithm for the shortest path problem allowing conservative edge lengths running in $O(n^4)$. Ford [1956] and Bellman [1958] independently gave similar algorithms with time complexity $O(nm)$. Their running time of $O(nm)$ is still the fastest known strongly-polynomial algorithm for the shortest path problem.

A number of algorithms has been proposed to solve the INTEGRAL SHORTEST PATH PROBLEM efficiently. An algorithm comparable to that of Ford and Bellman was developed by Moore [1959] for non-negative edge lengths with the same time complexity. Dantzig [1960] (with a refinement of Minty [1957]) proposed to choose an edge (u, v) for which $\gamma(v) + c((u, v))$ is smallest possible and achieved an algorithm with running time $O(n^2 \log n)$. For a survey on the early history of shortest path algorithms, see Schrijver [2005].

The basic strategy for solving the INTEGRAL SHORTEST PATH PROBLEM is the following algorithm given by Dijkstra [1959]:

Algorithm 4.1: DIJKSTRA'S ALGORITHM

```
1  Set γ(v) := 0 for v ∈ S; Set γ(v) := ∞ for v ∈ V(G) \ S;
2  Set P := ∅ and Q := S;
3  while Q ≠ ∅ do
4      Choose a vertex v ∈ Q with γ(v) = min{γ(w) : w ∈ Q};
5      Set Q := Q \ {v} and P := P ∪ {v};
6      if v ∈ T then
7          return γ(v);
8      end
9      forall w ∈ V(G) \ P with (v, w) ∈ E(G) do
10         if γ(w) > γ(v) + c((v, w)) then
11             Set γ(w) := γ(v) + c((v, w));
12             Set Q := Q ∪ {w};
13         end
14     end
15 end
16 return ∞;
```

The algorithm follows the general framework described at the beginning of that chapter. The set of vertices of $V(G)$ is partitioned into three sets: the set $\mathcal{Q}$ contains all vertices which have already received a feasible distance label possibly not minimum. $\mathcal{P}$ is the set of all vertices known to have received a feasible minimum distance label. These are called *permanently labeled* (or *scanned*). For all remaining vertices $v \in V \setminus (\mathcal{Q} \cup \mathcal{P})$ holds $\gamma(v) = \infty$. In each iteration of the algorithm a vertex of minimum distance value is moved from $\mathcal{Q}$ to $\mathcal{P}$, and all its neighbors are updated. The operation in line 11 of DIJKSTRA'S ALGORITHM is called *labeling step*. The algorithm stops as soon as a target vertex is

permanently labeled. If T is empty or not reachable from S the algorithm returns ∞. Dijkstra [1959] shows that after the execution of DIJKSTRA'S ALGORITHM, each vertex $v \in \mathcal{P}$ is labeled with the length $\gamma(v)$ of its shortest path to S.

The running time of DIJKSTRA'S ALGORITHM as formulated above is $O(n^2)$. It can be decreased if the order in which vertices are removed from $\mathcal{Q}$ is carefully chosen. The set $\mathcal{Q}$ is kept as a data structure supporting operations to insert an element, to decrease the key of an element and to delete an element with minimum key. Much effort has been spent to build an efficient data structure which improves the time complexity of DIJKSTRA'S ALGORITHM. If $\mathcal{Q}$ is implemented as a Fibonacci heap, the running time can be decreased to $O(m + n \log n)$ for arbitrary non-negative edge lengths (Fredman and Tarjan [1987]), which is the fastest known strongly polynomial implementation. This result has been improved further by Thorup [2004] for non-negative integral edge lengths to $O(m+n \log\log n)$. For undirected graphs even a linear running time can be achieved for integral lengths (Thorup [1999]), or on average in a randomized setting (Goldberg [2001], Meyer [2001]).

If $c(v) \leq C$ for all $v \in V(G)$ and some integer C, Dial [1969] formulated an $O(m + nC)$-algorithm using $O(nC)$ space which works well for small values of C. Dial's idea is to maintain an array $B = [0, \ldots, nC]$ of buckets and store labeled vertices v in $B[\gamma(v)]$. This allows inserting a vertex and decreasing the key in constant time, and finding a vertex of minimum key in $O(C)$ time. The space bound can be improved to $O(C)$ by using C many buckets and working modulo $C + 1$. Alternatively, the time bound can be reduced to $O(m + D)$ for a biggest distance D in G. Many improvements using different data structures and computational models can be found in literature; for a short survey see Thorup [2004].

The main idea of a *goal-oriented* path search (often called A^*) is to scan fewer vertices such that path search is guided from the source towards the target. Goal-oriented techniques have been proven to be very powerful in reducing running time of shortest path algorithms in practice, so that they are often used in combination with other techniques. They have first been described as heuristics in the artificial-intelligence setting by Doran [1967], Hart, Nilsson and Raphael [1968], and later by Rubin [1974].

For this, it uses a lower bound $\pi(v)$ on the distance from each $v \in V(G)$ to T in order to get a better estimation on the length of a path from S to T using vertex v. At each step in DIJKSTRA'S ALGORITHM, a vertex $v \in \mathcal{Q}$ with minimum *potential cost* $\gamma(v) + \pi(v)$ is picked to become permanently labeled. Optimality is not guaranteed in general, but can be shown if π satisfies

$$c_\pi((v,w)) := c((v,w)) - \pi(v) + \pi(w) \geq 0 \quad \text{for all } (v,w) \in E(G). \tag{4.1}$$

We call c_π the *reduced cost* with respect to π and π is a *feasible node potential* if the above *consistency condition* (4.1) holds. If π gives exact distances to T, the goal-oriented path search scans only vertices on shortest paths from S to T. It further can be shown that for a feasible node potential π and any two vertices v and w in $V(G)$, a shortest v-w-path in G with respect to c is a shortest v-w-path in G with respect to c_π, and vice versa.

Let π and π' two feasible potential functions such that $\pi(t) = \pi'(t) = 0$ for all $t \in T$ and $\pi(v) \leq \pi'(v)$ for all $v \in V(G) \setminus T$. Then, it is easy to observe that the set of vertices scanned by A^* using π' is contained in the set of vertices scanned by A^* using π (assuming a reasonable tie-breaking rule; see, for example, Goldberg and Harrelson [2005]). Since DIJKSTRA'S ALGORITHM can be viewed as a goal-oriented path search using zero potential cost, any A^*-algorithm with a non-negative feasible node potential scans no more vertices than DIJKSTRA'S ALGORITHM. This property leads to the task to compute good lower bounds. Lower bounds which are easy to compute are based on metric dependent distances (Rubin [1974], Sedgewick and Vitter [1986]). In the last few years, mainly two techniques to determine a good lower bound have been discussed in the literature: distances obtained from graph condensation (Müller-Hannemann and Weihe [2001]) and obtained by landmarks (Goldberg and Harrelson [2005]). These approaches require a pre-processing of the entire graph prior to path search.

Another fundamental approach in the field of shortest path searches is a *bi-directional* search. Although Dantzig [1960] already mentioned this idea, the pioneering work is attributed to Pohl [1971]. In a bi-directional search two searches are performed simultaneously. One (forward) path search applies DIJKSTRA'S ALGORITHM on the original graph searching from S to T. The other (backward) path search operates on the reverse graph $\bar{G} := (V(G), \{(v, u) : (u, v) \in E(G)\})$ and starts at T searching for a shortest path to S. The algorithm stops when a vertex v has been permanently labeled by both path searches. Note that upon stopping this vertex v does not necessarily have to be a vertex of the shortest path. The shortest path can be composed of two shortest paths of both searches. The performance of a bidirectional path search depends — among other things such as data structures — on the strategy in which order forward and backward path search are executed; see Luby and Radge [1989].

For many instances that arise in practice, the graphs have some underlying geometric structure which can also be exploited to speed up shortest path computations. Techniques using a *pre-processing* stage have become popular in the last few years in the context of public railroad transport and navigation on road networks. They all have in common to spend some time to retrieve local information which lead to a speed-up in the actual query time. Those techniques are applicable in practice where many shortest paths are queried on a static input graph. Instances might allow a natural *hierarchical decomposition*, where additional edges are added to the given graph which represent shortest paths between corresponding vertices. This way, the input graph can be coarsened leading to a sparse graph. Previous works include a multi-level approach (Schulz, Wagner and Zaroliagis [2002], Holzer, Schulz and Wagner [2006]), the concept of highway hierarchies (Sanders and Schultes [2005, 2006]) and the usage of so–called transit nodes (Bast et al. [2007]). Another pre-processing strategy *attaches labels* on vertices or edges in advance to indicate whether a certain vertex or edge has to be considered during the path search. This includes the approaches of reach-based routing (Gutman [2004]), landmarks (Goldberg and Harrelson [2005]), geometric containers (Wagner and Willhalm [2003]), edge labels and arc flags (Schulz, Wagner and Weihe [2000], Möhring et al. [2005]).

For a comprehensive study on combinations of speed-up techniques we refer to Holzer et al. [2005]. Computational experiments were presented on the 9th DIMACS Implementation Challenge [2006] for shortest paths instances. Excellent surveys on shortest paths algorithms published during the last years can be found in Wagner and Willhalm [2007], and Sanders and Schultes [2007].

4.1.2 Grid Graphs

Grid graphs play an important role in the application of VLSI design as we have described in Section 2.3. Although many algorithms for general graphs are also applicable for grid graphs, special algorithms have been developed which exploit the specific grid structure. In this section we give a brief overview of research on shortest path algorithms in grid graphs. Many of the approaches described below have been used by routing programs as we discussed in Section 2.5.

Maze running algorithms are widely used for routing in grid graphs. They are special implementations of DIJKSTRA'S ALGORITHM. Lee [1961] (with a slight correction by Rubin [1974]) was the first who formulated an algorithm for the special case $c \equiv 1$. Instead of implementing $\mathcal{Q}$ as a priority queue, Lee's algorithm uses a much simpler first-in-first-out queue. Although his algorithm has a running time of $O(n)$, it was considered to be very time and memory consuming. Therefore, subsequent papers proposed enhancements to Lee's algorithm. Rubin [1974] suggested to use a "rough prediction of the cost function" from a vertex to the target. He gives a short proof for his goal-oriented approach and shows that the search space does not become larger. If the integral edge lengths are bounded by C, Hoel [1976] gave an $O(nC)$-algorithm working with C many buckets instead of $nC + 1$ buckets as in Dial [1969]. Combining ideas by Rubin and Hoel, Hadlock [1977] proposed an algorithm with time complexity between $O(\sqrt{n})$ and $O(n)$ for grid graphs with uniform edge lengths. Finally, we mention a generic algorithm by Johann and Reis [2000] which simultaneously uses goal-oriented and bi-directional methods.

A disadvantage of maze running algorithms is that they cannot make use of uniform structures in large grid graphs. *Line search* algorithms are developed to overcome this drawback. Instead of generating wave fronts of labeled vertices, they search for escape lines to find a way towards the target. Those escape lines do not need to follow an underlying grid, so that line search algorithms are sometimes called *gridless* in literature. Line search algorithms do not necessarily find a shortest path. The first work by Mikami and Tabuchi [1968] minimizes only the number of bends, while the algorithms by Hightower [1969] and Dobes [1977] are not even guaranteed to find a path if there exists one. A similar approach was given by Soukup [1978] who combined the line search approach to run over long distances with a Lee-type expansion technique to label around obstacles. The latter two algorithms do find a connection between source and target, but not necessarily the shortest one. If the grid graph is represented by a set $\mathcal{I}$ of lines, Lipski [1984] derived an $O(|\mathcal{I}| \log |\mathcal{I}|)$-algorithm which minimizes the number of interval changes.

The principle of *line expansion* was first developed by Heyns, Sansen and Beke [1980] and improved by Sato, Kubota and Ohtsuki [1990] later. In contrast to line search algorithms, line expansion always finds an existing path, but not necessarily a shortest one. The idea is to consider not only escape lines but also all lines running orthogonal to them. This approach can be viewed as a labeling algorithm of two-dimensional rectangles. It is therefore also called *area search*.

For more information on many of the above-mentioned algorithms we refer to Lengauer [1994] and Hetzel [1995].

Hetzel [1998] combined the ideas of maze running and line search and proposed to represent the partial grid graph by a set of intervals of adjacent vertices and to label these intervals rather than individual vertices in his goal-oriented version of Dijkstra's Algorithm. Part of our work described in the following sections is a generalization of Hetzel's algorithm. Motivated by Hetzel and not much different, Shenoy and Nicholls [2002] presented an efficient database to answer region queries efficiently in their interval-based path search.

Many algorithms have been discovered in the field of computational geometry. They are mainly of theoretical interest and can hardly be applied in practice. The main approach here is to develop efficient data structures which reduce the path search complexity; see, for example, Chen and Xu [2001].

Finally, we mention the work of Xing and Kao [2002] who consider a graph which is part of the Hanan grid induced by all obstacles, and propagate piecewise linear functions on the edges of this graph.

4.2 Generalizing Dijkstra's Algorithm

The new algorithm which we call GeneralizedDijkstra provides a speed-up technique for Dijkstra's Algorithm in two ways. First, it can directly be applied to propagate distance labels through a graph. The main difference to other techniques is that our approach labels sets of vertices instead of individual vertices. It is beneficial if vertices can be grouped such that their distances can be expressed by a relatively simple distance function and updating neighboring sets works fast. In VLSI routing, the vertex sets are one-dimensional intervals of the three-dimensional grid. A second application of GeneralizedDijkstra is the goal-oriented search. Many techniques to determine a good lower bound have been discussed in the literature, see previous section. The approach proposed in this chapter is another method to compute lower bounds: it computes shortest distances from the target to all vertices in a supergraph G' of the reverse graph of the input graph. Here, G' must be chosen such that it approximately reflects the original distances and allows a partition of the vertex set in order to perform a fast propagation of distance labels. In our VLSI application, G' is the subgraph representing the global routing corridor, which is the union of only few rectangles.

One of our main proposals is to introduce three levels of hierarchy. The vertices of the graph are the elements of the lowest and most detailed level. The middle level is a partition

of the vertex set. Instead of labeling individual vertices as in the original version of DIJKSTRA'S ALGORITHM, we label the elements of this middle level. Finally, the top level is a partition of the middle level, i.e. several of the vertex sets of the middle level are related. The role of the top level of the hierarchy is that it allows to delay certain labeling operations. We perform labeling operations between elements of the middle level that are contained in the same element of the top level instantly, whereas all other labeling operations are delayed. Depending on the structure of the underlying graph, the well-adapted choice of the hierarchy and the implementation of the labeling operations, we can achieve running time reductions both in theory and in practice.

Throughout the rest of this chapter, we use the following notation. For vertices $u, v \in V(G)$ we denote by $\text{dist}_{(G,c)}(u, v)$ the minimum total length of a path in G from u to v with respect to c, or ∞ if v is not reachable from u. For a given non-empty source set $S \subseteq V(G)$ we define a function $d : V(G) \to \mathbb{Z}_{\geq 0} \cup \{\infty\}$ by

$$d(v) := \text{dist}_{(G,c)}(S, v) := \min\{\text{dist}_{(G,c)}(s, v) \mid s \in S\}$$

for $v \in V(G)$. If we are given a target set $T \subseteq V(G)$ we want to compute the distance

$$d(T) := \text{dist}_{(G,c)}(S, T) := \min\{d(t) \mid t \in T\}$$

from S to T in G with respect to c, or ∞ if T is not reachable from S.

Instead of labeling individual vertices with distance-related values, we label subgraphs of G induced by subsets of vertices with distance-related functions. Therefore, we assume to be given a set $\mathcal{V}$ of disjoint subsets of $V(G)$ and subsets $\mathcal{S}$ and $\mathcal{T}$ of $\mathcal{V}$ such that

$$V(G) = \bigcup_{U \in \mathcal{V}}^{\cdot} U \quad \text{and} \quad S = \bigcup_{U \in \mathcal{S}}^{\cdot} U \quad \text{and} \quad T = \bigcup_{U \in \mathcal{T}}^{\cdot} U.$$

We require that the graph $\mathcal{G}$ with $V(\mathcal{G}) := \mathcal{V}$ and

$$E(\mathcal{G}) := \{(U, U') \mid \exists u \in U, u' \in U', (u, u') \in E(G) \text{ with } c((u, u')) = 0\}$$

is acyclic. (Note that we do not need to assume this for G. Moreover, one can always get this property by contracting strongly connected components of $(V(G), \{e \in E(G) : c(e) = 0\}$.) Therefore, there is a topological order $V_1, V_2, \ldots, V_{|\mathcal{V}|}$ of $\mathcal{V}$ with $i < j$ if $(V_i, V_j) \in E(\mathcal{G})$. For $U \in \mathcal{V}$ we define the *index* of U to be $I(U) = i$ iff $U = V_i$.

Throughout the execution of the algorithm and for every $U \in \mathcal{V}$ we maintain a function $d_U : U \to \mathbb{Z}_{\geq 0} \cup \{\infty\}$ which is an upper bound on d, i.e.

$$d_U(v) \geq d(v) \text{ for all } v \in U, \tag{4.2}$$

and a feasible potential on $G[U]$, i.e.

$$d_U(v) \leq d_U(u) + c((u, v)) \text{ for all } (u, v) \in E(G[U]), \tag{4.3}$$

where $G[U]$ denotes the subgraph of G induced by U.

Initially, we set

$$d_U(v) := \begin{cases} 0 & \text{for } v \in U \in \mathcal{S}, \\ \infty & \text{for } v \in U \in \mathcal{V} \setminus \mathcal{S}. \end{cases}$$

We want to make use of a specific structure of the graph $\mathcal{G}$ and distinguish between two different labeling operations. For this, we additionally require a partition of $\mathcal{V}$ into $N \geq 1$ sets $\mathcal{V}_1, \ldots, \mathcal{V}_N$, called *blocks*, and a function $\mathcal{B} : \mathcal{V} \to \{\mathcal{V}_1, \ldots, \mathcal{V}_N\}$ such that

$$\begin{aligned} \forall\, 1 \leq i \leq N : &\quad \emptyset \neq \mathcal{V}_i \subseteq \mathcal{V}, \\ \forall\, U \in \mathcal{V} : &\quad U \in \mathcal{B}(U), \\ \forall\, 1 \leq i < j \leq N : &\quad \mathcal{V}_i \cap \mathcal{V}_j = \emptyset. \end{aligned}$$

Clearly, $\mathcal{V} = \bigcup_{i=1}^{N} \mathcal{V}_i$ and $V(G) = \bigcup_{i=1}^{N} \bigcup_{U \in \mathcal{V}_i} U$ by the definition of blocks.

A central concept of our algorithm GeneralizedDijkstra is the following: We distinguish between two operations for a vertex set $U \in \mathcal{V}$ which is chosen to label its neighbors: U *directly updates* the neighboring sets within the same block and *registers* labeling operations to vertex sets in different blocks for a later use. This approach has two advantages: First, many registered labeling operations may never have to be performed if a target vertex is reached before the registered operations would be processed. Second, if sets in $\mathcal{V}$ typically have few of their neighboring sets within the same block, update operations between blocks may be much more efficient when performed at once instead of one after another. For a schematic illustration of our algorithm see Figure 4.1. Two more examples are given in Section 4.3.

Our algorithm maintains a function $\text{key} : \mathcal{V} \to \mathbb{Z}_{\geq 0} \cup \{\infty\}$ and a queue $\mathcal{Q} = \{U \in \mathcal{V} \mid \text{key}(U) < \infty\}$, allowing operations to insert an element, to decrease the key of an element and to delete an element of minimum key. At any stage in the algorithm, for each $U \in \mathcal{V}$, $\text{key}(U)$ is the minimum distance label of any vertex in U that was decreased after the last time that U was deleted from $\mathcal{Q}$. After U has updated its neighbors or registered a labeling operation, $\text{key}(U)$ is set to infinity. It can be reset to a value smaller than infinity, as soon a $d(u)$ is reduced for a vertex of $u \in U$ by an update operation onto U.

For two sets $\mathcal{U}, \mathcal{U}' \subseteq \mathcal{V}$ and a queue $\mathcal{Q} \subseteq \mathcal{V}$ we use the following update operation which clearly maintains (4.2) and (4.3):

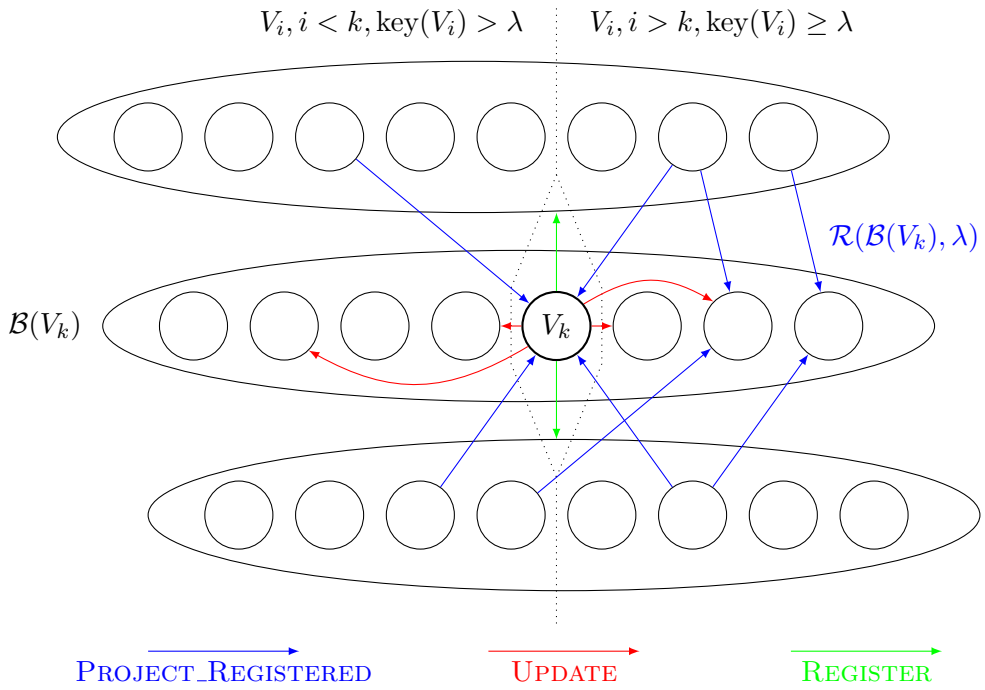


Figure 4.1: Schematic view of vertex sets V_i (circles) and blocks $\mathcal{V}_i$ (ellipses). The left-to-right order of the vertex sets is a topological order of $\mathcal{G}$. The arcs show update operations. If V_k and $\mathcal{B}(V_k)$ are selected in step 7 of GENERALIZED DIJKSTRA, then $\text{key}(V_i) > \lambda$ and $\mathcal{R}(V_i, \lambda) = \emptyset$ for $i = 1, \ldots, k-1$, and this property is maintained. Then first all registered updates onto block $\mathcal{B}(V_k)$ are performed (PROJECT_REGISTERED, blue arcs), then the elements of $\mathcal{B}(V_k)$ with key λ are scanned in their order (we show V_k only), updates within the block are performed directly (UPDATE, red arcs), and updates to other blocks are registered (REGISTER, green arcs). Each V_k is chosen at most once in phase λ.

Procedure UPDATE($\mathcal{U} \to \mathcal{U}', \mathcal{Q}$)

```
1  forall U' ∈ U' do
2      forall v ∈ U' do
3          Set δ := min_{U∈U} min_{u∈U} {d_U(u) + dist_{(G[{u}∪U'],c)}(u, v)};
4          if δ < d_{U'}(v) then
5              Set d_{U'}(v) := δ;
6              Set key(U') := min{key(U'), δ};
7              Set Q := Q ∪ {U'};
8          end
9      end
10 end
```

The actual labeling operation is done by UPDATE. For each vertex set $U' \in \mathcal{U}'$ and each vertex $v \in U'$, it computes the minimum distance in the subgraph determined by U' and all neighboring vertices in vertex sets of $\mathcal{U}$. If the distance label at v can be decreased, the label of v and the key of U' are updated. If U' is not in the queue, it will enter. Of course, this is not done sequentially for each single vertex. Here, the advantage of our algorithm becomes apparent: instead of performing the labeling steps on a vertex by vertex basis, it rather updates the distance function of neighboring vertex sets in one step. The more carefully a partition of V is chosen and the simpler d_U becomes, the faster UPDATE can work. In our main algorithm, UPDATE is called for a single vertex set $\mathcal{U} := \{U\}$ updating its neighbors in $\mathcal{B}(U)$, and for a set of vertex sets with registered labels updating their neighbors in one block.

The second operation is the registration of labeling operations to be postponed. For this, we define a set $\mathcal{R}(\mathcal{U}, \lambda)$ for each block $\mathcal{U} \in \{\mathcal{V}_1, \dots, \mathcal{V}_N\}$ and $\lambda \in \mathbb{Z}_{\geq 0}$, which consists of all vertex sets which might cause a label of value λ in some vertex set in $\mathcal{U}$. This set is filled by the following routine, where $E_G(U, U') := \{(u, u') \in E(G) \mid u \in U, u' \in U'\}$ and $\min \emptyset := \infty$.

Procedure REGISTER$(U \to \mathcal{U}', \mathcal{R})$

```
1 Set λ' := min_{U'∈𝒰'} min {δ | δ = d_U(u) + c((u,v)) < d_{U'}(v), (u,v) ∈ E_G(U,U')};
2 if λ' ≠ ∞ then
3     Set ℛ(𝒰', λ') := ℛ(𝒰', λ') ∪ {U};
4 end
```

REGISTER is called for a vertex set U and some block $\mathcal{U}' \neq \mathcal{B}(U)$. It computes the minimum label λ' which improves a label of at least one vertex in a neighboring vertex set of U in $\mathcal{U}'$ and registers U in $\mathcal{R}(\mathcal{U}', \lambda')$. If no label can be decreased, U will not be registered.

Given a block $\mathcal{U}$ and a key λ, we apply two major subroutines: First, all labeling operations of registered vertex sets of $\mathcal{U}$ at value λ are performed onto block $\mathcal{U}$ in PROJECT_REGISTERED. Afterwards, all vertex sets in $\mathcal{U}$ containing vertices with key λ update their neighbors within the same block and register labeling operations in different blocks in procedure PROJECT_FROMBLOCK. We use the notation $\mathcal{Q}_\lambda := \{U \in \mathcal{Q} \mid \text{key}(U) = \lambda\}$ for $\lambda \in \mathbb{Z}_{\geq 0}$.

Procedure PROJECT_REGISTERED$(\mathcal{U}, \lambda, \mathcal{Q}, \mathcal{R})$

```
1 if ℛ(𝒰, λ) ≠ ∅ then
2     UPDATE(ℛ(𝒰, λ) → 𝒰, 𝒬);
3     Set ℛ(𝒰, λ) := ∅;
4 end
```

Procedure PROJECT_FROMBLOCK($\mathcal{U}, \lambda, \mathcal{Q}, \mathcal{R}$)

```
1  while there is an element U ∈ Q_λ ∩ U do
2      Choose U ∈ Q_λ with minimum index;
3      Set Q := Q \ {U} and key(U) := ∞;
4      if U ∈ T then
5          return λ;
6      end
7      forall U' ∈ {B(U') | U' ∈ V \ {U} with E_G(U, U') ≠ ∅} do
8          if U' = U then
9              Set J_U := {U' ∈ U \ {U} | E_G(U, U') ≠ ∅};
10             UPDATE({U} → J_U, Q);
11         end
12         else
13             REGISTER(U → U', R);
14         end
15     end
16 end
```

PROJECT_REGISTERED makes up for all postponed labeling steps onto block $\mathcal{U}$ at label λ. After this, $\mathcal{R}(\mathcal{U}, \lambda)$ is empty. PROJECT_FROMBLOCK goes over all vertex sets $U \in \mathcal{U}$ in the queue whose key equals the current label λ according to their topological order. If a target vertex set is the minimum element in the queue, the overall algorithm stops. Otherwise, all neighboring vertex sets in the same block are directly labeled by UPDATE whereas labeling operations to neighbors in different blocks are registered by REGISTER.

Finally, we can formulate the overall algorithm. It performs labeling operations as long as no set in $\mathcal{T}$ has received a final label and there are still labeling operations which need to be executed. It runs in so-called *phases* where in the phase at key λ all vertices with distance λ receive their final label. In phase with key λ, the block $\mathcal{U}$ is chosen which includes the vertex set of minimum index containing a vertex with key λ. If no such vertex exists, a block $\mathcal{U}$ with postponed labels of value λ is taken. For $\mathcal{U}$, PROJECT_REGISTERED and PROJECT_FROMBLOCK are called in that order. PROJECT_REGISTERED must be called before labeling steps from vertices in vertex sets of $\mathcal{U}$ are performed within $\mathcal{U}$ in order to update vertices at label λ by neighbors of different blocks. Otherwise, PROJECT_FROMBLOCK might be not operate efficiently because a vertex might get a label λ at a later time in the algorithm which again requires an update operation for neighbors in $\mathcal{U}$. This loop at key λ is done as long as their is still a vertex to be scanned or a block with postponed labels. The algorithm stops as soon as a vertex in $\bigcup \mathcal{T}$ receives its final label and returns it, or it returns infinity to indicate that $\bigcup \mathcal{T}$ is not reachable.

Algorithm 4.6: GeneralizedDijkstra$(G, c, \mathcal{V}, \mathcal{B}, \mathcal{S}, \mathcal{T})$

```
1  Set key(U) := 0 and d_U(v) := 0 for U ∈ S and v ∈ U;
2  Set key(U) := ∞ and d_U(v) := ∞ for U ∈ V \ S and v ∈ U;
3  Set R(U, λ) := ∅ for U ∈ {V_1 ..., V_N} and λ ∈ Z_{≥0};
4  Set Q := S and λ := 0;
5  while Q ≠ ∅ or R(U, λ') ≠ ∅ for some λ' ≥ λ and some U ∈ {V_1, ..., V_N} do
6      while Q_λ ≠ ∅ or R(U, λ) ≠ ∅ for some U ∈ {V_1, ..., V_N} do
7          Choose U ∈ {V_1, ..., V_N} s.t. arg min{I(U) | U ∈ Q_λ or R(U, λ) ≠ ∅} ∈ U;
8          Project_Registered(U, λ, Q, R);
9          Project_FromBlock(U, λ, Q, R);
10     end
11     Set λ :=   min {μ : Q_μ ≠ ∅ or R(U, μ) ≠ ∅ for some U ∈ {V_1, ..., V_N}} ;
12 end
13 return ∞
```

Theorem 4.1. *The algorithm* GeneralizedDijkstra *calculates the correct distance from* $S := \bigcup \mathcal{S}$ *to* $T := \bigcup \mathcal{T}$*. If there is no path from* S *to* T*, the algorithm computes the minimum distance from* S *to all reachable vertices in* G *and returns* ∞*.*

Proof: We first show that for every $\lambda \in \mathbb{Z}_{\geq 0}$

$$\bigcup_{U \in \mathcal{V}} \{v \in U \mid d_U(v) = \lambda\} = \bigcup_{U \in \mathcal{V}} \{v \in U \mid d(v) = \lambda\} \tag{4.4}$$

holds after execution of phase λ which ends when λ is increased in line 11 of GeneralizedDijkstra. For contradiction, assume that there is a λ for which equation (4.4) does not hold. Choose λ minimum possible. By (4.2), $\lambda = d(v) < d_U(v)$ for some $U \in \mathcal{V}$ and $v \in U$. We choose $U \in \mathcal{V}$ with $I(U)$ minimum possible, and $v \in U$ and a shortest path P from S to v such that $|E(P)|$ is minimum. Let u be the predecessor of v on P. Hence, $\lambda' := d(u) \leq \lambda$ and, by the choice of v, $d_{U'}(u) = \lambda'$ for $U' \in \mathcal{V}$ with $u \in U'$. We show

$$d_U(v) \leq d_{U'}(u) + c((u, v)). \tag{4.5}$$

It directly follows for $U = U'$ by (4.3).

For $U \neq U'$, let $\mathcal{U} := \mathcal{B}(U)$ and $\mathcal{U}' := \mathcal{B}'(U)$. If $\lambda' < \lambda$, then U must have been updated directly from U' (if $\mathcal{U} = \mathcal{U}'$) or registered in Project_FromBlock$(\mathcal{U}', \lambda', \mathcal{Q}, \mathcal{R})$ (if $\mathcal{U} \neq \mathcal{U}'$) in phase λ'. In the latter case, d_U is updated by Update$(\mathcal{R}(\mathcal{U}, \lambda) \to \mathcal{U}, \mathcal{Q})$ in Project_Registered$(\mathcal{U}, \lambda, \mathcal{Q}, \mathcal{R})$. If $\lambda' = \lambda$, then $c((u, v)) = 0$. In this case, $I(U') < I(U)$ and U' must have been removed from $\mathcal{Q}$ before U (line 2 of Project_FromBlock). Consequently, U has been directly updated by U' in Project_FromBlock$(\mathcal{U}', \lambda, \mathcal{Q}, \mathcal{R})$ (if $\mathcal{U} = \mathcal{U}'$), or U' was registered in $\mathcal{R}(\mathcal{U}, \lambda)$ and has updated its neighbored vertex set U in Project_Registered$(\mathcal{U}, \lambda, \mathcal{Q}, \mathcal{R})$. For $\lambda' < \lambda$ as well as for $\lambda' = \lambda$, we conclude

$$d_U(v) \leq d_{U'}(u) + \mathrm{dist}_{(G[\{u\} \cup U'], c)}(u, v) \leq d_{U'}(u) + c((u, v)),$$

proving inequality (4.5) for $U \neq U'$. By (4.5) and our assumption on v and u,

$$d_U(v) \leq d_{U'}(u) + c((u,v)) = d(u) + c((u,v)) = d(v) < d_U(v),$$

which is a contradiction. This concludes the proof of (4.4).

All phases with key less than λ have been finished already when phase λ is being processed. By (4.4), a vertex $v \in U$ with $d(v) < \lambda$ cannot get a distance label $d_U(v) = \lambda$ after phase $\lambda - 1$.

It follows that

$$\{v \in U : d_U(v) = \lambda\} \subseteq \{v \in U : d(v) = \lambda\} \tag{4.6}$$

holds after a vertex set U has been removed from Q at key λ.

Therefore, an element $T \in \mathcal{T}$ is removed from the queue at minimum distance $\lambda = d(T)$ if T is reachable from S. Otherwise, GeneralizedDijkstra stops and returns ∞ as soon as $\mathcal{Q}$ is empty and there are no labeling registrations left in $\mathcal{R}$. By (4.4), the distance from S to any vertex $v \in V(G)$ that is reachable from S is given by $d_U(v)$ for $v \in U \in \mathcal{V}$. □

In addition to the computed distance from S to T or from S to all vertices, we are often also interested in shortest paths. These can be derived easily from the output of GeneralizedDijkstra. Alternatively, one could store predecessor information during the shortest path computation encapsulated in Update.

The theoretical running time of GeneralizedDijkstra as well as its performance in practice depends on the structure of the underlying graph G and its partition into vertex sets and blocks. In the special case of $N = 1$, the running time reduces to

$$O\left((\Lambda + 1)(|\mathcal{V}| \log |\mathcal{V}| + \phi |E(\mathcal{G})|)\right),$$

where Λ is the length of a shortest path from S to T and ϕ is the running time of lines 2 and 3 of Update. If $N = 1$ and all vertex sets are singletons, GeneralizedDijkstra equals Dijkstra's Algorithm with a running time of $O\left(|V(G)| \log |V(G)| + |E(G)|\right)$. The essential difference is that, for general vertex sets, an element of the queue can enter the queue again if it contains more than one vertex. Consequently, there are at most $\Lambda + 1$ many queries for the smallest element of the queue. Both time bounds assume that $\mathcal{Q}$ is implemented by a Fibonacci heap (Fredman and Tarjan [1987]).

GeneralizedDijkstra is primarily suitable for graphs with a regular structure, for which the ground set V can be partitioned into a set $\mathcal{V}$ of vertex sets with easily computable functions d_U for all $U \in \mathcal{V}$, e.g. linear functions.

4.3 Applications in VLSI Routing

In this section we describe two applications of GeneralizedDijkstra in VLSI routing where the algorithm is particularly efficient. In the following, we use the instance of the

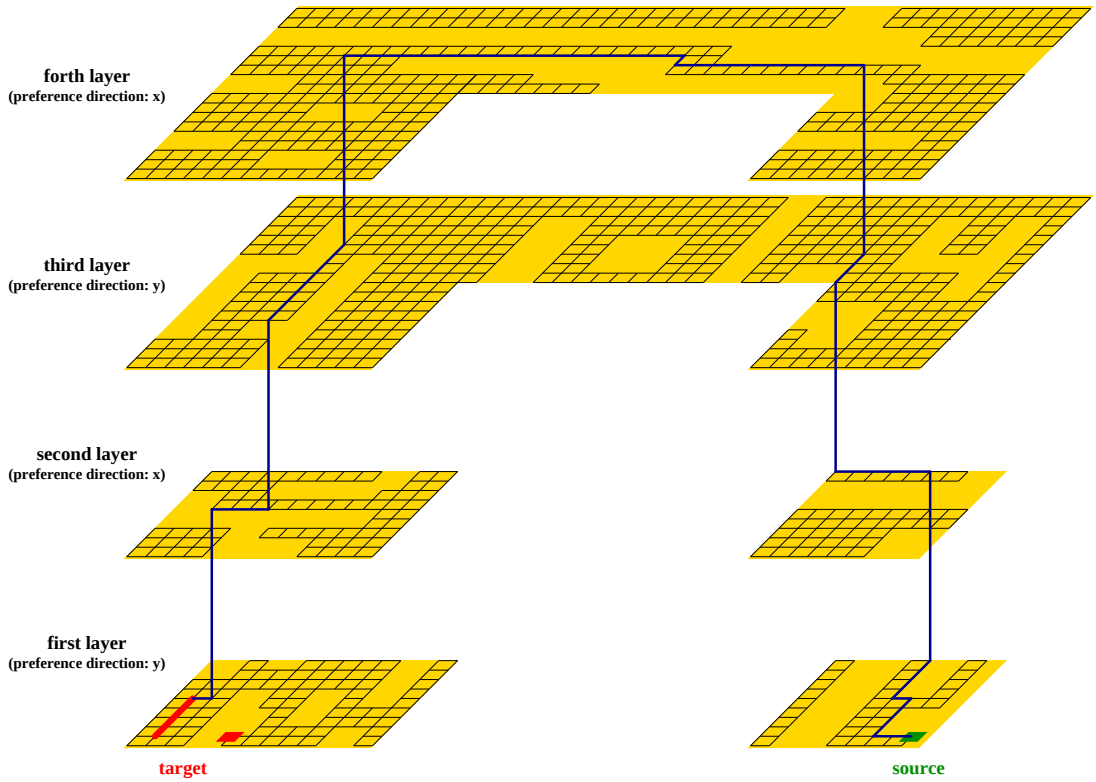


Figure 4.2: The source vertex (green) on the lower right corner and the target vertices (red) on the lower left corner of the picture are connected by a shortest path (blue) of length 153. The corridor determined by global routing (yellow) runs over four different layers in this example. The costs of edges running in and orthogonal to the preference direction are 1 and 4, respectively, the cost of a via is 13.

detailed routing problem and its solution shown in Figure 4.2 in order to demonstrate these applications.

4.3.1 Labeling Rectangles

For speeding up the computation of a shortest path in a subgraph of the three-dimensional grid graph G_0 (defined in Section 2.3) using a goal-oriented approach one can use a good lower bound on the distances. In order to determine this lower bound, we consider just the corridor computed by global routing and neglect obstacles and previously determined paths intersecting it. Since these corridors are produced by global routing as a union of relatively few rectangles. GeneralizedDijkstra computes distances efficiently. In this first application we use only one block, i.e. $N = 1$. Consequently, there are no registrations and Project_Registered is not called.

Let $G = (V(G), E(G))$ be a subgraph of the infinite graph G_0 induced by a set $\mathcal{V}$ of *rectangles*, where a rectangle is a set of the form

$$[x_1, x_2] \times [y_1, y_2] \times \{z_1\} \quad := \quad \big\{(x, y, z) \in \mathbb{Z}^3 \mid x_1 \le x \le x_2,\, y_1 \le y \le y_2,\, z = z_1\big\}$$

for integers $x_1 \le x_2$, $y_1 \le y_2$ and z_1. Note that $x_1 = x_2$ or $y_1 = y_2$ is allowed, in which case the rectangles are intervals or just single points. Two rectangles R and R' are said to be *adjacent* if $G_0[R \cup R']$ is connected.

We assume that $\mathcal{V}$ satisfies the following condition which can always be ensured by iteratively splitting rectangles while only slightly increasing the number of rectangles for typical VLSI instances: for every two rectangles $R, R' \in \mathcal{V}$ with $R = [x_1, x_2] \times [y_1, y_2] \times \{z_1\}$ and $R' = [x'_1, x'_2] \times [y'_1, y'_2] \times \{z'_1\}$ we have that either $x_1 > x'_2$ or $x_2 < x'_1$ or $(x_1, x_2) = (x'_1, x'_2)$, and similarly, either $y_1 > y'_2$ or $y_2 < y'_1$ or $(y_1, y_2) = (y'_1, y'_2)$. Clearly, this condition implies that each rectangle has at most six adjacent rectangles. As an example, Figure 4.3 shows the partition of the routing area of the instance in Figure 4.2 into such rectangles.

The set $Z := \{z \in \mathbb{Z} \mid \exists\, x, y \in \mathbb{Z} \text{ with } (x, y, z) \in V(G)\}$ contains all relevant z-coordinates, and the sets $C_i := \{0\} \cup \{c \in \mathbb{Z} \mid |c| = c_{z,i} \text{ for some } z \in Z\}$ contain all relevant edge lengths in x-direction ($i = 1$) and y-direction ($i = 2$), respectively. Let $k_i := |C_i|$ for $i = 1, 2$.

Due to the bounded number of different edge lengths, it is possible to store the function d_R corresponding to the function d_U of Section 4.2 implicitly as a minimum over $k_1 k_2$ linear functions assigned to each rectangle R:

$$d_R((x, y, z)) := \min\{d_{(R,c_1,c_2)}(x, y) \mid (c_1, c_2) \in C_1 \times C_2\},$$

where with $(c_1, c_2) \in C_1 \times C_2$ we associate a linear function $d_{(R,c_1,c_2)} : \mathbb{Z}^2 \to \mathbb{Z}_{\ge 0} \cup \{\infty\}$ of the form

$$d_{(R,c_1,c_2)}(x, y) = c_1(x - x_1) + c_2(y - y_1) + \delta_{(R,c_1,c_2)}.$$

See Figure 4.3 for examples of these functions.

All information on $d_{(R,c_1,c_2)}$ is contained in the offset value $\delta_{(R,c_1,c_2)}$. Initially,

$$\delta_{(R,c_1,c_2)} := \begin{cases} 0 & \text{for } (R, c_1, c_2) \in \mathcal{S} \times \{0\} \times \{0\}, \\ \infty & \text{for } (R, c_1, c_2) \in (\mathcal{V} \setminus \mathcal{S}) \times \{0\} \times \{0\}. \end{cases}$$

During the execution of the algorithm these values are updated as follows: Let $(R, c_1, c_2) \in \mathcal{V} \times C_1 \times C_2$, and let $R' \in \mathcal{V} \setminus \{R\}$ be adjacent to R. For $(x', y', z') \in R'$ let

$$\begin{aligned} &d_{((R,c_1,c_2)\to R')}(x', y') \\ &\qquad := \min\{d_{(R,c_1,c_2)}(x, y) + \mathrm{dist}_{(G_0[R\cup R'],c)}((x, y, z), (x', y', z')) \mid (x, y, z) \in R\}. \end{aligned}$$

The main observation established by the next lemma is that the function $d_{((R,c_1,c_2)\to R')}$ is of the same form as $d_{(R',c'_1,c'_2)}$ for appropriate values of c'_1 and c'_2.

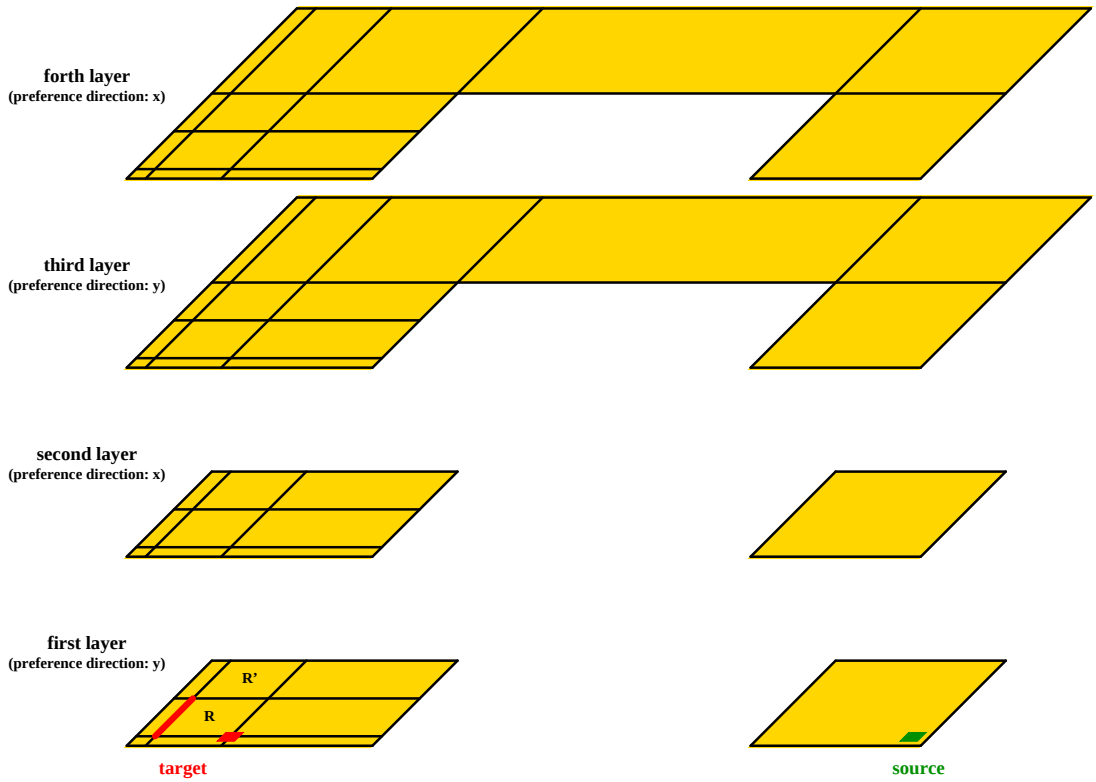


Figure 4.3: The corridor of the global routing for the instance depicted in Figure 4.2 is partitioned into rectangles to propagate distance functions. As an example, the function d_R of the rectangle R containing all target vertices at its boundary is given by $\min(4(x-x_1), -4(x-x_1)+(y-y_1)+16)$. The function $d_{R'}$ of the adjacent rectangle R' is $\min(4(x-x'_1)+(y-y'_1), -4(x-x'_1)+(y-y'_1)+20)$, where $x'_1 = x_1$ and $y'_1 = y_1 + 4$.

Lemma 4.2. *If $R \in \mathcal{V}$ and $R' = [x'_1, x'_2] \times [y'_1, y'_2] \times \{z'\} \in \mathcal{V}$ are adjacent, and $(c_1, c_2) \in C_1 \times C_2$, then there are $c'_1 \in C_1$, $c'_2 \in C_2$ and $\delta' \in \mathbb{Z}_{\geq 0}$ such that*

$$d_{((R,c_1,c_2)\to R')}(x', y') = c'_1(x' - x'_1) + c'_2(y' - y'_1) + \delta'$$

for all $(x', y', z') \in R'$.

Proof: We give details for just one case, since the remaining cases can be proved using similar arguments. Therefore, we assume that $R = [x_1, x_2] \times [y_1, y_2] \times \{z\}$ and $R' = [x_2 + 1, x_3] \times [y_1, y_2] \times \{z\}$ For $(x, y, z) \in R$ and $(x', y', z) \in R'$ we have

$$\mathrm{dist}_{(G_0[R\cup R'],c)}((x, y, z), (x', y', z)) = c_{z,1}|x - x'| + c_{z,2}|y - y'|.$$

Since $x < x'$, this simplifies to $c_{z,1}(x'-x)+c_{z,2}(y'-y)$ for $y \leq y'$ and $c_{z,1}(x'-x)-c_{z,2}(y'-y)$ for $y \geq y'$. Hence, by definition, $d_{((R,c_1,c_2)\to R')}(x',y')$ equals

$$\min \left\{ \min_{x_1\leq x\leq x_2} \min_{y_1\leq y\leq y'} \left(c_{z,1}(x'-x)+c_{z,2}(y'-y)+c_1(x-x_1)+c_2(y-y_1)+\delta_{(R,c_1,c_2)}\right), \right.$$
$$\left. \min_{x_1\leq x\leq x_2} \min_{y'\leq y\leq y_2} \left(c_{z,1}(x'-x)-c_{z,2}(y'-y)+c_1(x-x_1)+c_2(y-y_1)+\delta_{(R,c_1,c_2)}\right) \right\}.$$

Depending on the signs of $(c_1 - c_{z,1})$, $(c_2 - c_{z,2})$ and $(c_2 + c_{z,2})$, this minimum is attained — independently of the specific value of (x', y') — by setting

$$x := \begin{cases} x_1 & \text{if } c_1 - c_{z,1} \geq 0, \\ x_2 & \text{if } c_1 - c_{z,1} < 0 \end{cases} \quad \text{and} \quad y := \begin{cases} y_1 & \text{if } c_2 - c_{z,2} \geq 0, \\ y' & \text{if } c_2 - c_{z,2} < 0 \text{ and } c_2 + c_{z,2} \geq 0, \\ y_2 & \text{if } c_2 + c_{z,2} < 0 \end{cases}$$

and the desired result follows. □

This lemma shows that we can store d_R implicitly as a vector with k_1k_2 entries $\delta_{(R,c_1,c_2)}$ and that a labeling operation from one rectangle R to another R' can be done by manipulating the entries of the vector corresponding to $d_{R'}$. This can be done in constant time regardless of the cardinalities of R and R'.

We can now apply GeneralizedDijkstra implementing Update with the following operation:

Procedure Project_Rectangle($R \to \mathcal{U}', \mathcal{Q}$)

```
1  forall R' ∈ U' do
2      forall c_1 ∈ C_1 and c_2 ∈ C_2 do
3          Compute c'_1, c'_2, δ' with d_((R,c_1,c_2)→R')(x', y') = c'_1(x' − x'_1) + c'_2(x' − x'_2) + δ';
4          if δ' < δ_(R',c'_1,c'_2) then
5              Set δ_(R',c'_1,c'_2) := δ';
6              Set key(R') := min{key(R'), δ'};
7              Set Q := Q ∪ {R'};
8          end
9      end
10 end
```

Theorem 4.3. *If $d_{(R,c_1,c_2)}$ for $(R, c_1, c_2) \in \mathcal{V} \times C_1 \times C_2$ are the functions produced at termination by* GeneralizedDijkstra$(G, c, \mathcal{V}, \mathcal{S})$ *implementing* Update *with procedure* Project_Rectangle*, then $d((x, y, z)) = d_R((x, y, z))$ for all $(x, y, z) \in V(G)$. The corresponding running time of* GeneralizedDijkstra *is $O\left(k_1^2k_2^2|\mathcal{V}|\log|\mathcal{V}|\right)$.*

Proof. From Lemma 4.2, it follows that Project_Rectangle takes $O(k_1k_2)$ time. The number of deletions of elements from the queue is bounded by the total number of updates from any rectangle to any adjacent one, which is at most $6k_1k_2|\mathcal{V}|$. □

In our implementation, we improved the running time by applying PROJECT_RECTANGLE to triples $(R, c_1, c_2) \in \mathcal{V} \times C_1 \times C_2$ instead of rectangles, where the current key is attained by $d_{(R,c_1,c_2)}$. Here, PROJECT_RECTANGLE takes constant time and the number of updates from any rectangle to any adjacent one is still bounded by $O(k_1 k_2 |\mathcal{V}|)$. This leads to an overall running time of $O\left(k_1 k_2 |\mathcal{V}| \log(k_1 k_2 |\mathcal{V}|)\right)$. As mentioned before, in a typical VLSI instance k_1 and k_2 can be as small as 5; see experimental results in Section 4.4.

The example shown in Figure 4.4 illustrates that the lower bound on the distances to the target can considerably be improved in a goal-oriented path search if the global routing corridor is respected.

4.3.2 Labeling Intervals

As a second application of GENERALIZEDDIJKSTRA we consider the core routine in detailed routing. Its task is to find a shortest path connecting two vertex sets S and T in an induced subgraph G of G_0 with respect to costs $c : E(G) \to \mathbb{Z}_{\geq 0}$, where we can assume $c((u,v)) = c((v,u))$ for every edge $(u,v) \in E(G)$. We use the variant of GENERALIZEDDIJKSTRA as described in Section 4.3.1 to compute distances $\pi(w)$ from T to each $w \in V(G)$ in a supergraph G' of G where G' is determined by the corresponding global routing corridor. Hence, $\pi(w)$ is a lower bound for the distance of each $w \in V(G)$ to T in G with respect to c, $\pi(t) = 0$ for all $t \in T$, and $\pi(u) \leq \pi(v) + c((u,v))$ for all $(u,v) \in E(G)$. We call $\pi(w)$ the *future cost* of vertex w. The function π is used to define reduced costs $c_\pi((u,v)) := c((u,v)) - \pi(u) + \pi(v) \geq 0$ for all $(u,v) \in E(G)$. We apply GENERALIZEDDIJKSTRA to find a path P from $s \in S$ to $t \in T$ in G for which $\sum_{e \in E(P)} c(e) = \pi(s) + \sum_{e \in E(P)} c_\pi(e)$ is minimum by setting $d_U(v) := \pi(v)$ for $v \in U \in \mathcal{S}$ in line 1 and using c_π instead of c.

We generalize Hetzel's algorithm (Hetzel [1998]) by using a more sophisticated future cost function π as described in the previous section. Note that Hetzel's algorithm strongly relies on the fact that $\pi(v)$ is the l_1-distance between v and T for all $v \in V(G)$.

As most wires use the cheap edges in preference direction, it is natural to represent the subgraph of G induced by a layer z by a set of intervals in preference direction. Horizontal intervals are rectangles of the form $[x_1, x_2] \times \{y\} \times \{z\}$, and vertical intervals are rectangles of the form $\{x\} \times [y_1, y_2] \times \{z\}$. Typically, the number of intervals is approximately 25 times smaller than the number of vertices. Hetzel showed how to operate DIJKSTRA'S ALGORITHM on such intervals, but his algorithm works only for reduced costs defined with respect to l_1-distances. Clearly, the l_1-distance is often a poor lower bound. Therefore, our generalization allows for significant speed-ups. For example, in Figure 4.3 the l_1-distance is 36, our lower bound is 130, and the actual distance is 153. The improved lower bound of 130 includes 78 units for six vias, and 16 units for the necessary detours in x- and y-direction.

We again apply GENERALIZEDDIJKSTRA and the vertex sets in $\mathcal{V}$ are the respective intervals. Figure 4.5 illustrates the set of intervals for the example in Figure 4.2. In rare cases some intervals have to be split in advance to guarantee that π is monotone on an interval.

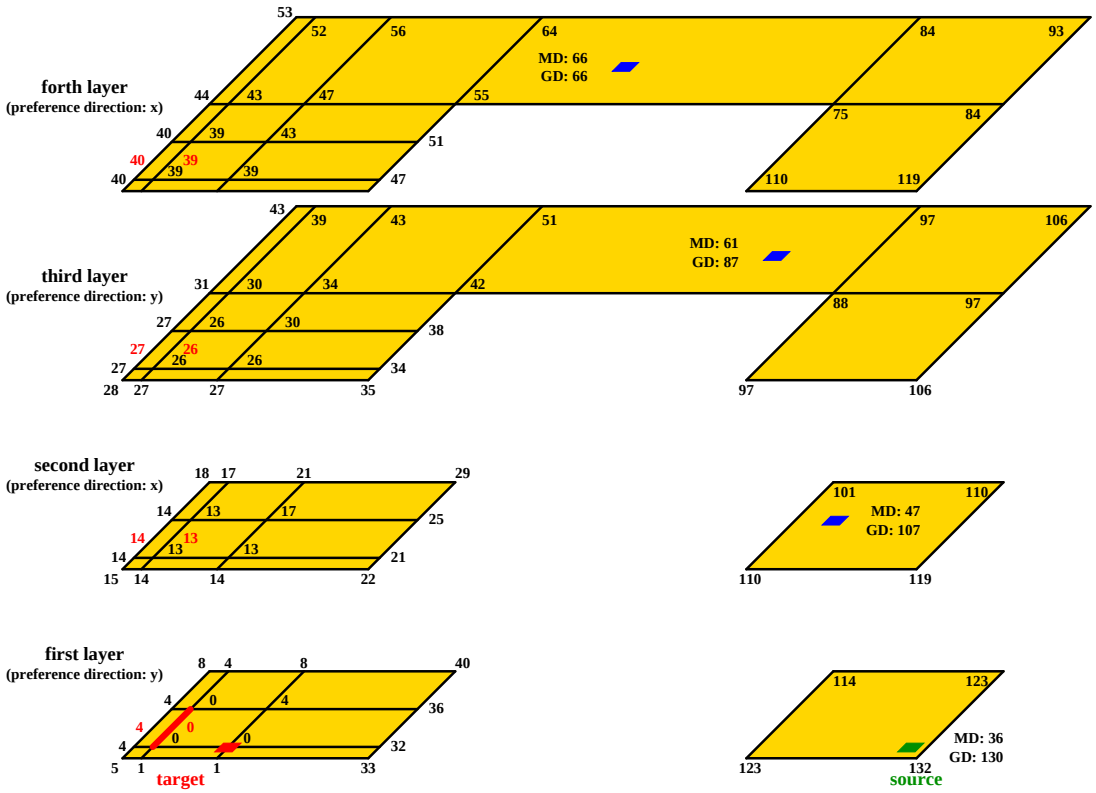


Figure 4.4: A comparison of two different computations of lower bounds to the target in a goal-oriented path search for vertices of the instance depicted in Figure 4.2. Black numbers at nodes and red numbers at the boundary of rectangles correspond to the distance functions computed by GeneralizedDijkstra. For four different nodes in the graph, the lower bound to the target evaluated with GeneralizedDijkstra ("GD") is always at least as large as the lower bound based on the Manhattan distance (respecting only via costs and neglecting penalty costs for jogs), referred to "MD". Especially in the near of the source both bounds differ significantly in that example because "GD" is able to account for vias necessary to find a path within the global routing corridor while "MD" cannot.

A *tag* is a triple (J, v, δ), where J is a subinterval of U, $v \in J$, and $\delta \in \mathbb{Z}_{\geq 0}$. At any stage we ensure to have a set of tags on each interval.

A tag (J, v, δ) on $U \in \mathcal{V}$ represents the distance function $d_{(J,v,\delta)} : U \to \mathbb{Z}_{\geq 0} \cup \{\infty\}$, defined as follows: Assume that $U = [x_1, x_5] \times \{y\} \times \{z\}$ is a horizontal interval, $J = [x_2, x_4] \times \{y\} \times \{z\}$, and $v = (x_3, y, z)$ with $x_1 \leq x_2 \leq x_3 \leq x_4 \leq x_5$. Then $d_{(J,v,\delta)}((x, y, z)) := \delta + \mathrm{dist}_{(G[J],c_\pi)}(v, (x, y, z))$ for $x_2 \leq x \leq x_4$ and $d_{(J,v,\delta)}((x, y, z)) := \infty$ for $x \notin [x_2, x_4]$. For vertical intervals, these definitions and properties carry over analogously. For $v \in U$ we define $d_U(v)$ to be the minimum of the $d_{(J,v,\delta)}(v)$ over all tags (J, v, δ) on U. However, this function d_U is not stored explicitly.

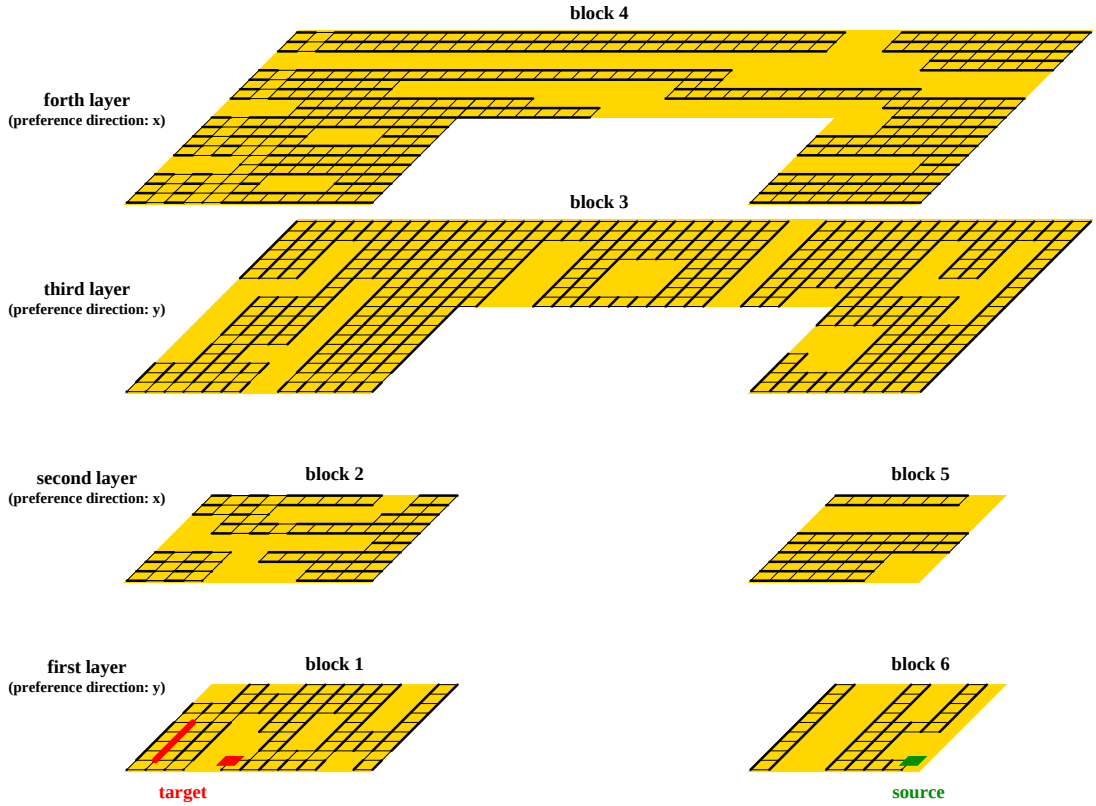


Figure 4.5: The intervals of the detailed routing for the instance depicted in Figure 4.2. For each layer, all intervals are sets of adjacent vertices according to the preference wiring direction of the layer. A block in this example is the set of intervals belonging to the same shaped area.

At any stage, we have the following properties:

- If $(x, y, z), (x + 1, y, z), (x + 2, y, z) \in J$ for some tag (J, v, δ), then $c_\pi(((x, y, z), (x + 1, y, z))) = c_\pi(((x+1, y, z), (x+2, y, z)))$ and $c_\pi(((x+2, y, z), (x+1, y, z))) = c_\pi(((x + 1, y, z), (x, y, z)))$.
- The tags on an interval U are stored in a search tree and in a doubly-linked list for U, both sorted by keys, where the key of a tag (J, v, δ) is the coordinate of v which corresponds to the preference direction.
- If there are two tags $(J, v, \delta), (J', v', \delta')$ on an interval U, then
  - $v \neq v'$.
  - $J \cap J' = \emptyset$
  - $d_{(J,v,\delta)}(v') > \delta'$.

The last condition says that no redundant tags are stored. If there are k tags on U, the search tree allows us to compute $d_U(v)$ for any $v \in U$ in $O(\log k)$ time.

We can also insert another tag in $O(\log k)$ time and remove redundant tags in time proportional to the number of removed tags. As every inserted tag is removed at most once, the time for inserting tags dominates the time for removing redundant tags.

For a fixed layer z let $V_z(G')$ be the set of vertices of the supergraph G' in this layer. Then $G'[V_z(G')]$ can be decomposed into a set of maximally connected subgraphs which become the blocks in GeneralizedDijkstra (Figure 4.5). Obviously, this fulfills the requirements of blocks given in Section 4.2. It is easy to see that the topological order of the intervals in $\mathcal{V}$ can be chosen according to non-ascending future cost values $\min\{\pi(v) \mid v \in U\}$.

The priority queue $\mathcal{Q}$ of GeneralizedDijkstra works with buckets $B_{\delta,\mathcal{W},I}$ with key $\delta \in \mathbb{Z}_{\geq 0}$, block $\mathcal{W} \in \{\mathcal{V}_1 \ldots, \mathcal{V}_N\}$ and index $I \in \{1, \ldots, |\mathcal{V}|\}$. An element $U \in \mathcal{Q} \cap \mathcal{W}$ is contained in bucket $B_{\text{key}(U),\mathcal{W},I(U)}$. The nonempty buckets are stored in a heap and processed in lexicographical order. This ensures that an interval is removed from $\mathcal{Q}$ only once per key.

The main difficulty is that an interval can have a large number of neighbors, particularly on an adjacent layer with orthogonal wiring direction. Although a single Update operation to these neighbors is fast, we cannot afford to perform all the resulting operations separately. Fortunately, there is a better way.

For neighbored intervals in the same layer we need to insert at most one tag. This is due to the monotonicity of the function π on an interval. We can register labeling operations on intervals in adjacent layers in constant time by updating $\mathcal{R}$. Finally, Project_Registered is performed in a single step for all intervals in $\mathcal{U}$ which are already registered at key λ. Set $R := \mathcal{R}(\mathcal{U}, \lambda)$ for short. We maintain a sweepline to process the elements in R which costs $O(|R| \log |R|)$ time. Hetzel [1998] showed that each interval in $\mathcal{U}$ needs to be updated by at most one interval in R. Adding the time for searching neighbored intervals in R and for the labeling operation itself, the overall running time for Project_Registered applied to block $\mathcal{U}$ is $O(|R| \log |R| + |\mathcal{U}|(\log |R| + \log |\mathcal{U}| + \log \Delta_{\mathcal{U}}))$, where $\Delta_{\mathcal{U}}$ is the maximum number of tags in an interval in $\mathcal{U}$. Since $\Delta_{\mathcal{U}}$ can be bounded by the number of intervals in $\mathcal{U}$ (Hetzel [1998]), we get the following theorem:

Theorem 4.4. *If* GeneralizedDijkstra *is applied to a set $\mathcal{V}$ of intervals partitioning the vertex set $V(G)$ of a detailed routing instance (G, c, S, T) and reduced costs c_π with respect to a feasible potential π, then its running time is*

$$O\left(\min\{(\Lambda + 1)|\mathcal{V}| \log |\mathcal{V}|, |V(G)| \log |V(G)|\}\right),$$

where Λ is the length of a shortest path from S to T with respect to c_π.

4.4 Experimental Results

We analyze two applications of GeneralizedDijkstra of Section 4.3. The algorithm is implemented in the detailed path search of BonnRoute, which is a state-of-the-art routing tool used by IBM. Our experiments are made sequentially on an AMD-Opteron machine with 64 GiB memory and four processors running at 2.6 GHz.

We run our algorithm on 12 industrial VLSI designs from IBM. Table 4.1 gives an overview on our testbed which consists of seven 130 nm and five 90 nm chips of different size and of different number $|Z|$ of layers. Here, the distance between adjacent tracks of a chip is usually the minimum width of a wire plus the minimum distance of two wires. We test our algorithm on about 30 million path searches in total. We use the standard costs with which BonnRoute is applied with in practice. These costs are 1 and 4 for edges running in and orthogonal to the preference direction, respectively, and 13 for a via.

Chip	Tech	Image Size $\times 10^3$ Tracks	$\lvert Z\rvert$	#Paths $\times 10^3$	$\lvert V\rvert$ $\times 10^6$	$\lvert\mathcal{V}\rvert$ $\times 10^6$	$\lvert R\rvert$ $\times 10^6$	$\lvert R_{\text{hyb}}\rvert$ $\times 10^6$
Bill	130 nm	26 × 26	7	32	10 385	529	20	5
Paul	90 nm	24 × 24	8	148	19 391	762	393	12
Hannelore	90 nm	36 × 33	8	263	40 434	1 616	903	24
Elena	130 nm	19 × 19	6	896	126 562	5 494	1 707	78
Heidi	130 nm	23 × 23	7	1 537	231 558	8 836	3 172	177
Garry	130 nm	26 × 26	7	1 810	305 100	10 725	3 886	237
Edgar	90 nm	40 × 40	8	1 866	263 836	10 456	3 843	236
Ralf	130 nm	26 × 26	7	2 902	472 826	18 330	9 827	243
Monika	130 nm	35 × 35	7	3 006	513 863	20 323	8 535	263
Edmund	90 nm	44 × 44	9	4 288	573 697	26 290	5 012	358
Hermann	130 nm	46 × 46	7	5 509	899 406	37 768	12 415	626
David	90 nm	53 × 53	9	7 786	953 092	49 341	14 870	1 328
All				30 043	4 410 150	190 470	64 583	3 587

Table 4.1: Testbed characteristics

In Table 4.1, Columns 6–9 give the sum of all instances for each chip. The number of grid vertices of all detailed routing instances sums up to 4.4 trillion. The seventh column of Table 4.1 shows that the number $|\mathcal{V}|$ of intervals is by a factor of about 23 smaller than the number $|V|$ of individual vertices. (In practice, the number of labeled intervals is even smaller by an additional factor of about 3.) This confirms observations by Hetzel [1998].

In Table 4.2 we compare a node-based ("classical") and interval-based ("old") path search, both goal-oriented using l_1-distances as future cost (that is, "classical" corresponds to the algorithm by Rubin [1974] and "old" to Hetzel [1998]). The interval-based implementation of Dijkstra's Algorithm decreases the number of labels by a factor of about 9.3, while the running time is improved by a factor of 16 on average. All running times include the time for performing initialization routines for future cost queries. These numbers confirm that it is absolutely necessary to apply an interval-based path search in order to get an acceptable running time.

Chip	Number of Labels ($\times 10^6$)			Running Time (sec)		
	classical	old	factor	classical	old	factor
Bill	23 940	1 333	18.0	46 818	1 037	45.2
Paul	6 079	624	9.7	8 008	580	13.8
Hannelore	19 621	2 025	9.7	26 849	1 546	17.4
Elena	33 727	4 277	7.9	53 776	4 604	11.7
Heidi	52 889	6 426	8.2	73 156	6 160	11.9
Garry	83 910	10 330	8.1	121 189	9 715	12.5
Edgar	139 139	13 319	10.4	193 642	12 020	16.1
Ralf	81 436	10 682	7.6	123 421	11 651	10.6
Monika	95 803	12 650	7.6	135 286	12 949	10.4
Edmund	206 289	20 768	9.9	315 541	26 304	12.0
Hermann	366 455	37 363	9.8	661 441	51 190	12.9
David		(56 872)		1 859 491	88 639	21.0
All	1 109 288	119 797	9.3	3 618 618	226 395	16.0

Table 4.2: Comparison of the node-based (classical) and interval-based (old) path search. The results were obtained by two different runs for each of the two approaches. One run was optimized with respect to running time, the other — much more time consuming — run counts each single label step. Running the classical approach on David already took more than a week. Therefore, we have not evaluated the number of labels for the classical path search on David which would have taken several weeks of computing time.

Second, we compare the performance of the detailed path search in BonnRoute based on different future cost computations: Hetzel's original path search using l_1-distances for future cost values (Hetzel [1998]), and a new version as described in Section 4.3.1.

The number $|R|$ of rectangles used to perform GeneralizedDijkstra on rectangles as described in Section 4.3.1 is given in the second to last column of Table 4.1. As this number is only by a factor of about 3 smaller than the number of intervals, and applying GeneralizedDijkstra on rectangles can take a significant running time, it is not worthwhile to perform this pre-processing on all instances. Rather we would like to apply our new approach only to those instances where the gain in running time of the core path search routine is larger than the time spent in pre-processing. As we cannot know this a priori, we need to set up a good heuristic criterion to estimate whether this effort is expected to pay off. While in one of our test scenarios ("new") we run Generalized Dijkstra on rectangles for each path search to obtain a better future cost, the second scenario ("hybrid") does this only if all of the following three conditions hold:

- the global routing corridor is the union of at least three three-dimensional cuboids.
- the number of target shapes is at most 20, and
- the l_1-distance of source and target is at least 50.

Otherwise the l_1-distance is used as future cost. Table 4.3 shows that in the hybrid scenario, extensive pre-processing is used approximately in a quarter of all path searches. We compared it to a third scenario ("old") where the l_1-distance is always used (as proposed in Hetzel [1998]). All scenarios run GeneralizedDijkstra on intervals as described in Section 4.3.2. The last column of Table 4.1 lists the number $|R_{\text{hyb}}|$ of rectangles of the pre-processing step for the hybrid scenario, where we account for one rectangle if the old l_1-based approach is applied. This shows that the instance size of GeneralizedDijkstra on rectangles in the hybrid scenario is significantly smaller than that of the actual interval-based path search.

Chip	#Paths	
	all ($\times 10^3$)	ext pp
Bill	32	36.5 %
Paul	148	18.3 %
Hannelore	263	18.6 %
Elena	896	20.2 %
Heidi	1 537	23.7 %
Garry	1 810	26.9 %
Edgar	1 866	25.1 %
Ralf	2 902	17.7 %
Monika	3 006	18.6 %
Edmund	4 288	24.6 %
Hermann	5 509	24.2 %
David	7 786	33.4 %
All	30 043	25.5 %

Table 4.3: Portion of total path searches for which extensive preproccesing (ext pp) is applied in the hybrid scenario

In Tables 4.4 and 4.5, we compare the number of labels and the length of detours, again summed up over all instances, for each chip of our testbed. The detour of a path is given by the length of the path minus the future cost value of the source. For each chip, the sum of all path lengths, given in the second column of Table 4.5, was about the same for all three scenarios. (They all guarantee to find shortest paths, but they may find different shortest paths occasionally, leading to different instances subsequently.)

The number of labels clearly decreases by applying the new approach. In the hybrid and new scenario, the total number of labels can be reduced by about 38 % and 49 %, respectively, on average. The total length of detours decreases by about 36 % and 56 %, respectively, on average.

In Table 4.6, we present the main result of our study which is a significant improvement in running time of the interval-based path search when using the hybrid scenario compared to the old scenario.

Chip	Number of Labels ($\times 10^6$)				
	old	hybrid	Δ_{hyb}	new	Δ_{new}
Bill	1 333	544	−59.2 %	464	−65.2 %
Paul	624	350	−43.9 %	248	−60.2 %
Hannelore	2 025	823	−59.3 %	682	−66.3 %
Elena	4 277	2 755	−35.6 %	2 099	−50.9 %
Heidi	6 426	3 958	−38.4 %	2 779	−56.8 %
Garry	10 330	5 930	−42.6 %	4 601	−55.5 %
Edgar	13 319	6 237	−53.2 %	5 054	−62.1 %
Ralf	10 682	6 409	−40.0 %	4 507	−57.8 %
Monika	12 650	7 670	−39.4 %	5 587	−55.8 %
Edmund	20 768	11 384	−45.2 %	8 791	−57.7 %
Hermann	37 363	22 208	−40.6 %	18 732	−49.9 %
David	56 872	40 451	−28.9 %	36 046	−36.6 %
All	176 669	108 719	−38.5 %	89 590	−49.2 %

Table 4.4: Number of labels for the interval-based path search in three different scenarios. The improvement in the number of labels is given by Δ_{hyb} and Δ_{new} for the hybrid and the new scenario, respectively.

Chip	Length of Paths ($\times 10^3$)	Length of Detours ($\times 10^3$)				
		old	hybrid	Δ_{hyb}	new	Δ_{new}
Bill	59 361	1 454	686	−52.8 %	403	−72.3 %
Paul	38 544	5 688	4 363	−23.3 %	2 838	−50.1 %
Hannelore	113 392	9 443	5 659	−40.1 %	3 748	−60.3 %
Elena	248 355	37 443	24 922	−33.4 %	14 853	−60.3 %
Heidi	404 564	58 608	40 979	−30.1 %	22 903	−60.9 %
Garry	605 396	77 874	48 461	−37.8 %	30 325	−61.1 %
Edgar	809 167	76 813	47 205	−38.5 %	26 718	−65.2 %
Ralf	661 519	112 238	78 915	−29.7 %	47 280	−57.9 %
Monika	729 992	118 016	83 559	−29.2 %	52 316	−55.7 %
Edmund	1 232 087	192 218	130 483	−32.1 %	85 654	−55.4 %
Hermann	2 314 273	266 446	155 045	−41.8 %	97 189	−63.5 %
David	2 830 553	565 832	358 058	−36.7 %	284 172	−49.8 %
All	10 047 203	1 522 073	978 335	−35.7 %	668 399	−56.1 %

Table 4.5: Length of paths and detours for the interval-based path search in three different scenarios. The improvement in the length of detours is given by Δ_{hyb} and Δ_{new} for the hybrid and the new scenario, respectively.

Chip	old	hybrid	init	Δ_{hyb}	new	init	$\overline{\Delta}_{\text{hyb}}$
Bill	1 037	525	12	−49.4 %	518	46	−54.5 %
Paul	580	459	25	−20.9 %	1 590	1 187	−30.7 %
Hannelore	1 546	1 031	48	−33.3 %	4 355	3 370	−36.2 %
Elena	4 604	3 811	169	−17.2 %	8 191	4 833	−27.1 %
Heidi	6 160	5 077	382	−17.6 %	14 571	10 199	−29.0 %
Garry	9 715	8 142	529	−16.2 %	20 552	13 247	−24.8 %
Edgar	12 020	8 627	555	−28.2 %	19 898	12 215	−36.1 %
Ralf	11 651	9 599	544	−17.6 %	42 862	33 944	−23.4 %
Monika	12 949	11 030	575	−14.8 %	39 502	29 541	−23.1 %
Edmund	26 304	20 667	890	−21.4 %	32 184	14 078	−31.2 %
Hermann	51 190	39 568	1 450	−22.7 %	83 235	43 248	−21.9 %
David	88 639	80 941	3 213	−8.7 %	123 010	47 836	−15.2 %
All	226 395	189 477	8 392	−16.3 %	390 468	213 744	−21.9 %

Table 4.6: Running time (in sec) for the interval-based path search in three different scenarios. The improvement in running time for the hybrid scenario is given by Δ_{hyb} for which an upper bound $\overline{\Delta}_{\text{hyb}}$ can be found in the last column. (We observed running time variations up to 4.3 % with identical code on the same instance. Therefore, we have run our program three times on each instance and computed the average CPU time.)

All running times for old, hybrid and new include the time for performing initialization routines for future cost queries. For the hybrid and new scenario we also give the time spent in the pre-processing routine of the new approach, which contains initializing the corresponding graph and performing GeneralizedDijkstra on rectangles. Although 4.4 % of the running time of hybrid is spent in the initialization step, we obtain a total improvement of 16.3 % in comparison to the old scenario. The best result of 33.3 % was obtained on Hannelore, the worst of 8.7 % on David which shows by far the least improvement. Most reductions in running time are in the range of 15–20 %. Only a carefully chosen heuristic enables us to identify those instances for which it is worthwhile to spend time on the more expensive computation of a better future cost. In the hybrid scenario the new approach is called for 25.5 % of the path searches, which, however, takes most of the running time. We would get a theoretical improvement of 21.9 % if we ran the new scenario and did not take initialization time into account (see last column of Table 4.6). This shows that our criteria in the hybrid scenario are close to optimal. (Due to unreliability in measuring CPU time, we observed an improvement Δ_{hyb} in running time on Hermann which is bigger than its upper bound $\overline{\Delta}_{\text{hyb}}$.)

The combination of both techniques — the interval-based path search and the usage of improved future cost values — substantially speed up the core routine of detailed routing, one of the most time-consuming steps in the layout process. Thus, a significant reduction of the overall turn-around time from a couple of days to a few hours can be achieved.

Chapter 5

BonnRoute in Practice

In this chapter, we focus on BonnRoute, the routing tool developed at the Research Institute for Discrete Mathematics at the University of Bonn. We present data of typical state-of-the-art industrial VLSI chips and show computational results BonnRoute achieves during operation on these chips in practice.

5.1 Success of BonnRoute

The BonnTools program package comprises a set of applications providing solutions for the physical design of integrated circuits. They are developed at the Research Institute for Discrete Mathematics in Bonn and cover placement, timing optimization, clock synthesis and routing. During 20 years of cooperation with IBM, they have been used intensively on leading-edge designs in many design centers all over the world. Since 2005, The BonnTools are also incorporated into the design system of Magma Design Automation. For a comprehensive overview on the main ideas of the mathematical algorithms used in the BonnTools, we refer to Korte, Rautenbach and Vygen [2007].

BonnRoute is the routing tool of the BonnTools. The IBM design center in Böblingen (Germany) has started to use *XRouter*, the predecessor of BonnRoute, in 1992. Five years later, the cooperation was extended to IBM in the U.S.A., where BonnRoute became the only router within IBM and its customers. BonnRoute supports all recent and new technologies. It mainly consists of the programs BonnRouteGlobal for global routing and BonnRouteLocal for detailed routing. It is applicable flexibly, can be driven in various operating modes and is parallelized very efficiently based on a shared memory architecture; see Section 2.6 for a more detailed description of the core routines of BonnRouteGlobal and BonnRouteLocal.

So far, BonnRoute has been successfully used for the layout of more than one thousand different chips. It has shown its strength on various complex industrial designs, which we want to demonstrate by giving the following examples:

- BonnRoute has already been applied to a chip with 11 million nets, 1,300 m of total netlength and about 100,000,000 vias.
- In 2002, the wiring of IBM's fastest ASIC was realized with BonnRoute. It was a custom integrated circuit (North Bridge memory controller) for the Apple G5 processor which has been sold more than a million times.
- BonnRoute was involved in the set of tools which achieved IBM's fastest ASIC turnaround time.
- IBM's densest and largest chip with respect to image size (referenced by Hermann in tables below) with 18.3 mm edge length was routed by BonnRoute in 2003.
- BonnRoute has also been used for routing another large chip of the same image size: TRIPS (Tera-op, Reliable, Intelligently adaptive Processing System) which is a prototype chip developed at the University of Texas at Austin and designed in cooperation with IBM (McDonald, Burger and Keckler [2005]).
- Two of IBM's first ASICs in 65 nm technology are currently routed by BonnRoute.

5.2 Orders of Magnitude

In this section we characterize our testbed consisting of 17 state-of-the-art chips from IBM. In the subsequent section we present results we have achieved on these chips.

The CMOS (*Complementary Metal Oxide Semiconductor*) technology is the dominant technology for manufacturing integrated circuits. There are different levels of CMOS technologies which are specified by their process generation. The *process generation* is referred to by the drawn length of the silicon gate between the source and drain terminals in field effect transistors (FETs). The three widely used technologies for state-of-the-art chips are 130 nm, 90 nm and 65 nm technologies. The generations after that will be 45 nm and 32 nm. (In contrast, the feature size for chips manufactured in 1989 was 1000 nm.) For further insight into technological details in general and on CMOS in detail, see e.g. Weste and Eshraghian [2002].

Our testbed contains seven chips in a 130 nm technology, five chips for a 90 nm and five for a 65 nm technology. Table 5.1 gives an overview of some key figures we discuss further.

The length and width of an entire chip area, presented in the third column of Table 5.1, is typically between 7 mm and 18 mm. In hierarchical designs, some small parts of the chip are defined as RLMs (*Random Logic Macros*) which form a well defined unit of logic. RLMs can be designed independently from other RLMs and are connected among each other and to the peripheral devices at a later stage, called top-level routing. (In our testbed, three instances are RLMs: Tara, Benedikt and Tina.)

Column 4 of Table 5.1 shows the area size in terms of channels. As explained in Section 2.3, routing tools usually make use of a regular grid to efficiently solve the routing

Chip	Tech	Image Size (mm)	Image Size (10^3 Channels)	#Wiring Layers	#Nets ($\times 10^3$)
Bill	130 nm	10.2 × 10.3	26 × 26	7	11
Paul	90 nm	6.6 × 6.8	24 × 24	8	68
Hannelore	90 nm	10.0 × 9.2	36 × 33	8	140
Elena	130 nm	7.5 × 7.6	19 × 19	6	421
Tara	65 nm	4.4 × 3.9	22 × 20	7	509
Benedikt	65 nm	3.2 × 2.9	16 × 15	7	528
Tina	65 nm	2.4 × 4.0	12 × 20	7	534
Dorothea	65 nm	6.1 × 6.1	30 × 30	10	679
Edgar	90 nm	11.2 × 11.2	40 × 40	8	772
Heidi	130 nm	9.3 × 9.4	23 × 23	7	777
Garry	130 nm	10.2 × 10.3	26 × 26	7	828
Ralf	130 nm	10.2 × 10.3	26 × 26	7	1 350
Monika	130 nm	13.8 × 13.9	35 × 35	7	1 503
Hermann	130 nm	18.3 × 18.4	46 × 46	7	2 332
Edmund	90 nm	12.4 × 12.4	44 × 44	9	2 609
Larry	65 nm	14.0 × 14.0	70 × 70	10	3 589
David	90 nm	14.8 × 14.8	53 × 53	9	5 751

Table 5.1: Characteristics of our testbed consisting of 17 IBM chips

task. The number of channels corresponds to the number of coordinates in the routing grid. It is determined by the image size divided by the pitch, which depends on the technology: Pitches are 400 nm, 280 nm and 200 nm for 130 nm, 90 nm and 65 nm technology, respectively. Decreasing pitches are the reason that larger chips in one technology may have a smaller number of channels compared to a smaller chip in a newer technology. For example, Hermann (130 nm) and Edmund (90 nm) have about the same number of channels in horizontal and vertical direction, although the image size of Edmund is less than half of the image size of Hermann.

For current chips, the number of wiring layers is between 6 and 10. RLM instances typically have fewer wiring layers in order to leave wiring space for the top-level routing.

The number of nets is in the range of millions; the maximum number of nets BonnRoute has routed on a chip is around 11 millions. In our tests we have run chips with up to 5.8 million nets (column 6 of Table 5.1). These numbers contain not only signal nets but also nets with wider wiretypes for I/O nets and clock nets. Each signal net contains between three and four pins on average. RLMs usually do not have I/O nets.

The width of a wire may also vary with the layer it is running on. For example, in a 130 nm technology with seven layers the width of the wiretype `single` is 200 nm on wiring layers 1–4, whereas it is 400 nm on wiring layers 5–7. Table 5.2 shows the width of wires of four different wiretypes for 130 nm, 90 nm and 65 nm technologies. The majority of the nets are routed with the standard wiretype `single` which is normally the wiretype defining the

Wiring Layer	130 nm			90 nm			65 nm		
	`single`	`double`	`io`	`single`	`double`	`io`	`single`	`double`	`io`
1	200	600	-	160	440	-	100	300	-
2	200	600	-	140	420	-	100	300	-
3	200	600	-	140	420	-	100	300	-
4	200	600	-	140	420	20 180	100	300	-
5	400	400	18 840	140	420	20 180	200	200	-
6	400	400	18 840	140	420	20 180	200	200	-
7	400	400	18 840	280	280	19 800	200	200	-
8	-	-	-	280	280	19 800	400	400	12 000
9	-	-	-	-	-	19 800	400	400	12 000
10	-	-	-	-	-	-	2400	-	31 000

Table 5.2: Widths of wires of different wiretypes for 130 nm, 90 nm and 65 nm IBM technologies (in nm)

thinnest wires. Other often used wiretypes are `double` and `triple`. Timing-critical nets typically have been an assigned wiretype with thick wires. The thickest wires are used for I/O nets.

Although the number of wiretypes of a technology is in the range of 50–100, the number of wiretypes actually used for a given instance is typically less than 20.

The minimum space allowed between two neighboring shapes on a fixed layer normally equals the minimum width of any wire on that layer. For the technologies in Table 5.2 it is 200 nm in the 130 nm technology, 140 nm in the 90 nm technology, and 100 nm in the 65 nm technology.

Figure 5.1 shows a part of a modern industrial chip in a 65 nm technology, routed with BonnRoute.

5.3 Experimental Results

In the previous section we have introduced characteristic data of some state-of-the-art chips. We now present experimental results we achieved with BonnRoute on the testbed presented in Table 5.1. We distinguish between traditional measures such as wire length, running time or memory consumption, and *wiring integrity errors* which is the sum of all errors a routing tool leaves to be fixed by the designer. At the end of this section, we briefly introduce functions of BonnRoute which support yield improvement.

5.3.1 Traditional Criteria

In practice, the main objective of BonnRoute is to complete the wiring of a chip without violating design rules. Thereby, an objective function is optimized (timing, power consumption or yield) which is mainly considered by global routing already. Two aspects have

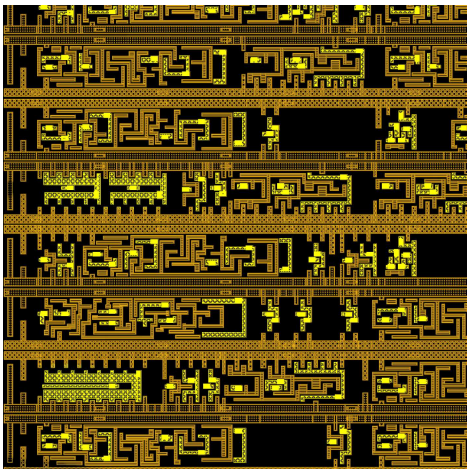
also to be taken into account by BonnRoute: the consumption of wiring space on the chip (wire length and number of vias), and resources of the software program consumed on the operating system (running time and memory consumption). In this section, we focus on these traditional performance measures and turn to design rules in the next section.

Chip	Tech	#Nets ($\times 10^3$)	L_D (m)	L_G (m)	$\frac{L_D - L_G}{L_G}$	L_S (m)	$\frac{L_D - L_S}{L_S}$	GR Cong.
Bill	130 nm	11	23.36	23.31	0.2 %	23.27	0.4 %	79.2 %
Paul	90 nm	68	9.87	9.83	0.4 %	9.75	1.3 %	87.7 %
Hannelore	90 nm	140	30.30	30.19	0.4 %	30.20	0.3 %	87.8 %
Elena	130 nm	421	92.02	92.21	−0.2 %	89.06	3.3 %	90.1 %
Tara	65 nm	509	50.68	47.73	6.2 %	48.14	5.3 %	86.5 %
Benedikt	65 nm	528	29.64	27.41	8.1 %	28.70	3.3 %	67.8 %
Tina	65 nm	534	43.11	41.12	4.8 %	41.16	4.7 %	83.4 %
Dorothea	65 nm	679	86.08	82.87	3.9 %	81.32	5.8 %	90.4 %
Edgar	90 nm	772	211.26	210.02	0.6 %	210.44	0.4 %	86.0 %
Heidi	130 nm	777	150.33	149.41	0.6 %	149.15	0.8 %	87.7 %
Garry	130 nm	828	221.03	220.58	0.2 %	218.15	1.3 %	88.2 %
Ralf	130 nm	1 350	236.15	236.99	−0.4 %	231.17	2.2 %	88.5 %
Monika	130 nm	1 503	262.96	263.75	−0.3 %	259.49	1.3 %	88.0 %
Hermann	130 nm	2 332	862.53	862.05	0.1 %	842.50	2.4 %	91.1 %
Edmund	90 nm	2 609	328.07	326.06	0.6 %	324.07	1.2 %	88.8 %
Larry	65 nm	3 589	380.05	360.44	5.4 %	365.88	3.9 %	89.2 %
David	90 nm	5 751	768.23	757.28	1.4 %	743.63	3.3 %	91.6 %
All		22 401	3 785.67	3 741.25	1.2 %	3 696.08	2.4 %	

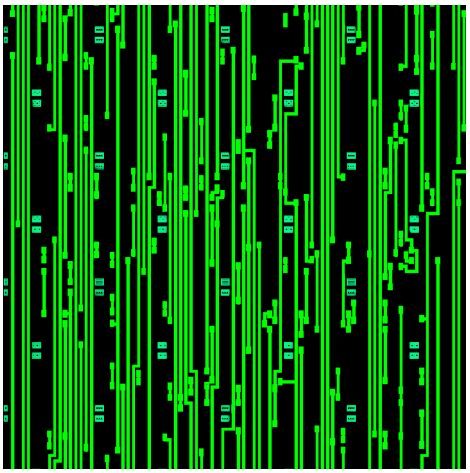
Table 5.3: Detailed routing length (L_D) compared to global routing length (L_G) and the length of a Steiner minimum tree (L_S). "GR Cong." gives the average congestion of the worst global routing edges whose crossing wires sum up to 20 % of the total wire length.

Table 5.3 presents wire length statistics of BonnRoute on all chips of our testbed. In the fourth column, we give the wire length L_D of detailed routing. To measure the quality of L_D, we compare it to two other results: first, we determine the difference of L_D to the estimated wire length L_G computed by global routing. This number is small if detailed routing follows the global routing corridors. The fourth column from the right shows that the difference is about 1.2 % on average. It is less than 0.5 % on average for chips in 130 nm and 90 nm technologies and 5.4 % on average for 65 nm.

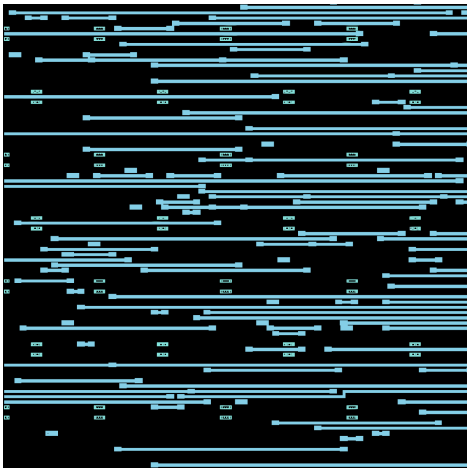
A lower bound on the wiring length of a net is the length of a two-dimensional Steiner minimum tree (pins are projected onto the x-y-plane). Steiner minimum trees can be computed by algorithms as described in Chapter 3. The second column from the right shows that detailed routing differs from the length of a Steiner minimum tree by 2.4 % on average on all chips. The average differences are 2.1 % and 4.3 % for chips in 130 nm and 90 nm technologies and 65 nm technology, respectively. The increase observed for 65 nm



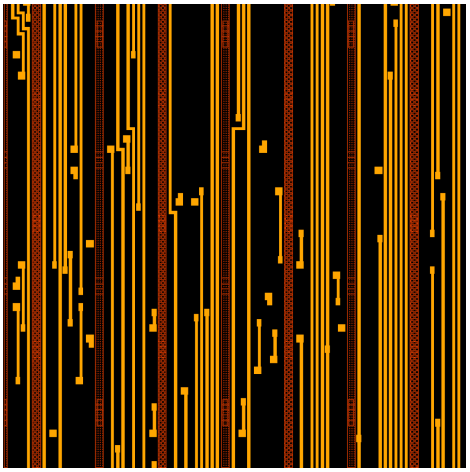
Wiring layer 1



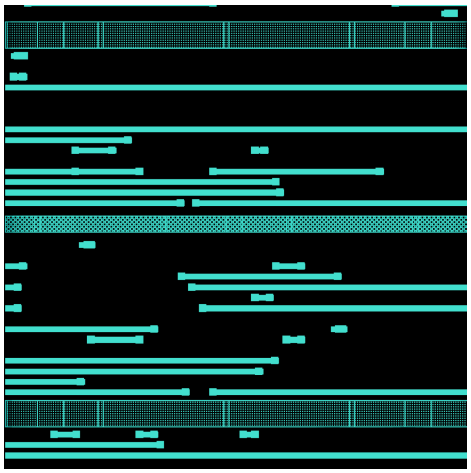
Wiring layer 2



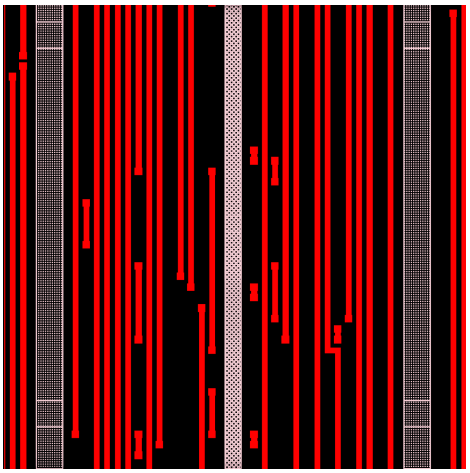
Wiring layer 3



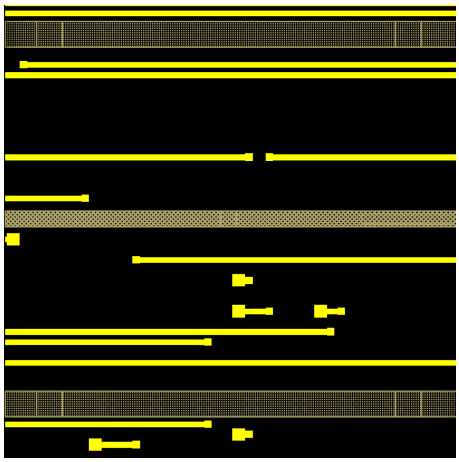
Wiring layer 4



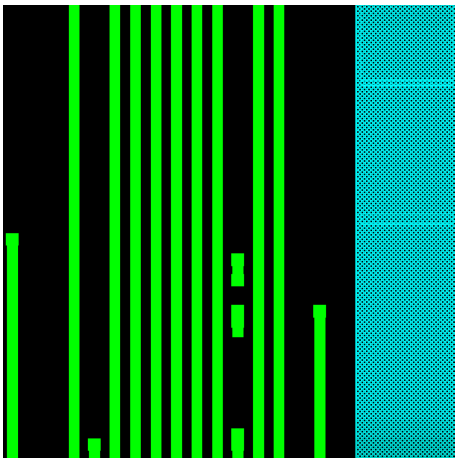
Wiring layer 5



Wiring layer 6



Wiring layer 7



Wiring layer 8

Wiring layer 9

Wiring layer 10

Figure 5.1: Part of signal wiring of Dorothea (65 nm technology) on all ten wiring layers. The depicted part has a size of 17.4 μm×17.4 μm (87×87 channels, about seven circuit rows), which corresponds to a 122,000th of the chip area. Most pins (yellow, dotted) and blockages of circuits (orange, striped) are located on layer 1, the lowest wiring layer. It is mainly used for pin access and does not serve for wire segments to run over long distances. Wires are depicted as filled segments, mostly running in preference wiring direction. The preference direction of wiring layers 1, 3, 5, 7 and 9 is horizontal, whereas it is vertical for wiring layers 2, 4, 6, 8 and 10. In this example, all wires are assigned wiretype `single`. The width of wires on wiring layers 1–4 equals the minimum width of any wire of that technology. Compared to the minimum wire width, the wire width is twofold on wiring layers 5–7, and fourfold on wiring layers 8 and 9 (cf. Table 5.2). Via pads are displayed by small filled rectangles which are slightly wider than their attached wire segments. Power structure (shadowed) can be found on wiring layers 1 and 4–10. They are connected by power vias which can be seen by shadowed rectangles on wiring layers 2 and 3.

chips can be explained by a combination of causes such as the complicated pin access in the gridless library and the usage of extra wiring required to fulfill new design rules.

In the last column of Table 5.3 we list the average congestion of a subset of all global routing edges; more precisely, we sort the edges by congestion and consider only the most congested ones such that the wires crossing these edges make up 20 % of the total global wire length. This measure gives a good estimation of the overall congestion of a design. For less recent technologies, the difference between L_D and L_S correlates very well with this congestion measure. On chips with a large increase in wire length (e.g. Elena, Hermann and David), the congestion exceeds the 90 % mark. We do not observe a similar correlation for chips in 65 nm technology, which still needs to be investigated.

As detailed wire length has a direct influence on the overall capacitance, it is desirable to minimize total wire length — provided that all other design-specific constraints can be met. However, a net topology of minimum length may be less desirable than a different one which leads to improved timing (e.g. Peyer [2000]), and detours can be acceptable to achieve shortest possible connections for other, timing-critical nets.

Table 5.4 gives statistics on the number of vias, running time and memory consumption. Our experiments were made sequentially on an AMD-Opteron machine with 64 GiB memory and four processors running at 2.6 GHz.

Chip	Tech	#Nets ($\times 10^3$)	#Vias ($\times 10^3$)	#Vias per Pin	Running time (CPU) global	detailed	Memory (GiB)
Bill	130 nm	11	91	1.83	0:10:53	0:23:20	1.74
Paul	90 nm	68	428	1.60	0:14:38	0:56:02	2.39
Hannelore	90 nm	140	767	1.44	0:30:41	1:53:22	3.38
Elena	130 nm	421	2 494	1.53	0:43:35	3:08:20	2.45
Tara	65 nm	509	4 951	2.64	0:37:08	3:33:15	5.52
Benedikt	65 nm	528	4 163	2.23	0:30:19	2:37:42	4.50
Tina	65 nm	534	5 142	2.44	0:21:56	2:55:51	4.80
Dorothea	65 nm	679	6 581	2.70	2:49:26	6:15:38	8.41
Edgar	90 nm	772	5 700	1.84	1:42:47	10:18:34	7.32
Heidi	130 nm	777	4 453	1.56	0:38:53	4:12:17	4.08
Garry	130 nm	828	5 277	1.70	1:06:28	5:50:04	4.44
Ralf	130 nm	1 350	7 921	1.50	1:26:08	8:37:58	5.68
Monika	130 nm	1 503	8 349	1.51	1:26:49	9:57:36	8.54
Hermann	130 nm	2 332	17 384	1.90	7:55:35	23:03:12	12.15
Edmund	90 nm	2 609	16 520	1.85	3:54:13	28:08:54	15.08
Larry	65 nm	3 589	32 035	2.36	7:12:27	29:18:16	41.04
David	90 nm	5 751	43 648	2.25	16:38:52	91:29:30	32.79

Table 5.4: Statistics on number of vias, running time and memory consumption

A reduction in the number of vias usually has a positive effect on yield because vias are much more crucial to yield loss than wiring. Although experience shows that BonnRoute

produces about 10 % less vias than other industrial routing tools, it is not a priori clear how to give a quantitative interpretation of the number of vias spent in detailed routing.

The number of vias per pin is a good measure to assess the total number of vias. On average, detailed routing takes about two vias per pin. It averages 1.9 % on instances of 130 nm and 90 nm technologies. This figure is higher for 65 nm technology (2.4 %), which is justified by three reasons: the number of wiring layers is larger for 65 nm, the lowest wiring layer can rarely be used for wiring, and the average wiring space per layer is decreased (only four single-wide wiring layers instead of up to six in 90 nm).

Routing takes a large portion of the overall completion time for a chip (turn-around time). Thus, it is of great importance to keep the running time of BonnRoute as small as possible. Much effort has been spent to decrease running time of BonnRoute, for example speeding up the detailed path search using the techniques and approaches presented in this thesis (cf. Chapter 4). In Table 5.4, the running time is split up into global and detailed routing. Global routing is much faster than detailed routing. Its running time very much depends on the congestion of a design (cf. last column of Table 5.3). BonnRouteGlobal takes almost eight hours on Hermann which is one of the densest design, whereas it only takes about half the time on Edmund. Once the global routing has found a solution without overloaded global routing edges, the running time of detailed routing is not as highly dependent on routing congestion as global routing. For example, the running time of BonnRouteLocal on Edmund is larger than the running time on Hermann although they have comparable number of nets and Hermann is more congested than Edmund. David is the most difficult design in our testbed. It takes more than 100 hours to complete the routing task.

As mentioned above, our experiments were made on a standard AMD-Opteron machine running at 2.6 GHz. Similar types of machines are also used in design centers at IBM. (Experiments on an Intel dual quad-core Xeon running at 2.66 GHz have shown that the running time may still be decreased; for example the running time of detailed routing on Larry dropped from 29:18:16 to 21:42:24 hours.)

On average, BonnRoute — running sequentially — can complete 1.9 million nets per day. This can be improved further by BonnRouteGlobal and BonnRouteLocal in a multi-threaded mode. Experimental results show that the MCF-algorithm of BonnRouteGlobal scales very well with the number of processors used (Müller [2006]). For BonnRouteLocal running with four processors we have seen a speed-up of the main routing loop of more than three on large instances, leading to an overall speed-up of up to 2.5 (Panten [2005]).

Low memory consumption is another important requirement to a state-of-the-art routing tool. Many of today's computing machines have 64 GiB RAM. At the same time, current chip instances grow in size and complexity such that only a sophisticated memory management can lead to a reasonable and still feasible memory consumption. Due to the usage of a gridless library in the 65 nm technology, memory consumption of BonnRoute is higher on chips in the 65 nm technology than for chips in older technologies. This is confirmed by the results presented in the last column of Table 5.4.

5.3.2 Design Rules

In this section we first give a short introduction into the manufacturing process of wiring layers of integrated circuits in order to motivate and understand various types of design rules routing is faced with. After that, we describe the most important design rules in more detail. We conclude this section with experimental results of BonnRoute with respect to the total number of wiring integrity errors.

A *wafer* is a thin disk of semiconducting material, typically silicon, and is cut into many chips. The manufacturing of a wafer is a very complex photolithographic process which is done layer by layer. The fabrication process of each individual layer is a complex flow of several optical, chemical and mechanical process steps where each single step can potentially alter the shapes to be produced.

The fabrication process starts with forming the P and N transistors on the silicon wafer. Then the wiring and via layers are formed. Each metal layer starts with depositing an insulation layer to the previous layer. Typical insulation materials are silicon dioxide (SiO_2) or materials with a lower dielectric constant (so-called low-k material). Then this material is covered with a thin film of photoresist. With the help of a mask, the light-sensitive resist is exposed to light such that it becomes less chemically robust against certain edging materials. So, the desired patterns or their inverse are printed on the photo resist. This exposure step is also called optical lithography. In the etching step, chemical materials selectively remove the photoresist and the underlying dielectric material from the layer according to desired patterns on that layer. This step results in holes (for vias) or trenches (for wires) which is filled with copper. As copper can diffuse into the dielectric, holes and trenches are sealed with a thin barrier film (called liner) before. The photoresist is removed after it is no longer needed. The final step in the fabrication of a layer is planarization (often done by chemical-mechanical polishing) to provide a flat surface for the subsequent layer on top.

During these manufacturing steps various types of fabrication errors can occur, such as mask misalignment, rough surfaces or changes in the process parameters, leading to shape distortions. Shapes can be typically degraded in mask creation, due to exposure and resist variations, and during etching (Lavin and Liebmann [2002]). As a consequence, there are two main effects on the circuit function, performance and reliability caused by shape distortions: variations can lead to a loss of connectivity (*opens*) as well as to undesirable connectivity (*shorts*). Another problem arises when shapes change their physical outline and when spacing between different shapes becomes too small. This can result in a worse timing behavior or even in local errors.

Another source of failures on a chip are particles which contaminate the fabrication process of chips. Although all steps of the manufacturing process are performed in an extremely clean environment, a certain but very low level of contamination cannot be avoided. Small particles can result in missing material (if covering patterns prevent shapes from etching) or in extra material (if particles create extra shapes in mask creation). For a schematic picture see Figure 5.2.

Figure 5.2: A particle (dark brown) contaminates the fabrication of wiring (yellow) which can result in a metal defect (brown).

In practice, there are two main approaches to avoid failures on the chip caused by the manufacturing process: prevention and correction. Prevention is mainly done by imposing *design rules* (also called *ground rules*) which specify constraints on geometric properties and relations between different shapes. Their task is to ensure that a chip works correctly after manufacturing even when some small distortions in fabrication occur (within some tolerance). As design rules define constraints for a routing tool, they must comprise a relatively small set of simple geometric constraints. Therefore, they cannot cover all possible shape manipulations and a second approach is necessary as a post-processing solution after routing. Here, the most important correction method is *optical proximity correction* (OPC) to create mask shapes where an inverse distortion is applied to patterns in order to compensate for shape distortions in one of the fabrication steps. Other correction techniques are, for example, filling and slotting to provide a certain shape density affecting the etching and planarization process. One problem of post-processing is that some manipulations may not be possible late in the design process. Another drawback is that full-chip shape processing tend to result in an unacceptable running time and data size. Thus, issues of post-processing are moved to an earlier step in the design flow, e.g. formulated as a new or more complex design rule. For more information, see Lavin and Liebmann [2002].

All design rules aim at reducing the likelihood of failures in the fabrication process. They can be classified into three different types. *Hard required* rules specify a limit in order to achieve at least some minimum acceptable yield, whereas *soft required* rules shall guarantee that the timing constraints of the chip are met. Design rules of both types must be satisfied by the wiring. As manufacturing has a significant impact on yield, *recommended* rules are additionally specified to improve yield further. It is not the prime objective to follow those rules, but desirable if all other design rules can be met. Recommended rules are often formulated as hard required rules to affect yield mandatorily. Yield can often be improved further by tightening thresholds of required design rules.

Design rules vary with technologies and they are specified individually for each layer. In the following, we describe the most important design rules in more detail, assuming some fixed wiring layer.

Minimum Width Rule
The minimum width rule sets the minimum width any wire on a specified layer must have. It is mainly imposed by the resolution of the photolithographic process. Although the ultraviolet light used for exposure has a wavelength of 200 nm to 400 nm, it is possible by some tricks (phase shifting) to create structures with width of less than 100 nm. There always exists a wire (usually called `single`) having minimum width, see Table 5.2.

Minimum Spacing Rule
Different shapes which do not intersect must keep a minimum distance to each other. The main reason is to ensure that they do not cause a short after fabrication. In practice, the distance is defined by the Euclidian distance. Another reason for setting up a spacing rule is crosstalk where a signal is affected by another nearby signal. Increased spacing (as well as wire shielding) is used in practice to prevent crosstalk between timing-critical nets.

The minimum spacing between shapes of minimum width is usually the same as the minimum width. In general, the spacing which must be kept between two shapes depends on the width of both shapes. As the fabrication process (especially in exposure and etching) is highly optimized for minimum feature size objects, processes are less adjusted for larger shapes. This leads to different kinds of variations for large shapes which can be mitigated by increasing the required minimum space to large shapes. Therefore, minimum allowed spacing to large shapes is usually larger than to thin shapes. Distance requirements for shapes of the same net are typically more relaxed than those for different nets.

Minimum Area Rule
Very small metal shapes without any connection to other shapes can detach from the glass underneath and move to another place on the chip causing a short.

Minimum Enclosed Area Rule
A minimum enclosed area is a region completely enclosed by metal shapes. This non-metal region is made of dielectric material which can detach if its area fall below a certain threshold interfering exposure at other places of the chip. Therefore, non-metal regions must not be smaller than a given threshold.

Short Edge Rule
OPC which is applied to compensate for optical and other process distortions becomes difficult on certain geometric configurations which, therefore, have to be avoided. Here, the configuration of two (or more) consecutive short edges on the border of a metal shape becomes increasingly important with 65 nm and newer technologies. The short edge rule specifies a threshold for the length of the involved edges.

Via Reliability Rule
Defects in large metal shapes tend to move to locations of attached vias. Therefore, thin vias meeting large shapes are unstable and must be replaced by multi-cut vias. (A multi-cut via on a via layer is compounded by several vias of a standard via type which is the only via type defined on the given via layer.)

Floating Gate Rule
During fabrication, various steps can provoke an electro-static discharge and a destruction of already connected gates; particularly polishing, the final step of manufacturing a layer, is done in a mechanical and chemical process. Two solutions are applied in practice. First, large electrically connected components in one layer can be avoided by breaking up long segments and jumping to layers above. Second, a floating gate diode which is able to discharge an electrical charge can be added between a transistor input gate and ground. Inserting such an element, however, is possible only if there is sufficient space in the placement layer. It can be easily applied after routing and is chosen if the first repair method fails.

With the exception of the floating gate rule, all of the above design rules can be specified as input to BonnRoute, which then has to respect them. Floating gate problems are not handled by BonnRoute since there already exists a post-processing step within the design flow of IBM. Table 5.5 shows the results we have obtained with BonnRoute on chips of our testbed.

The wiring of all chips was checked by ChipEdit, IBM's graphical physical design editor whose checking engine serves as the sign-off tool for the designer to release a chip. (ChipEdit counts some types of failures twice, such that the actual number of failures is even smaller.)

The number of connections which are not closed (opens, column 3 in Table 5.5) very much depends on the instance. Opens can be caused either by dense regions in the neighborhood of pins making pin access difficult (local problem), or by congested regions in which some connections cannot be realized (global problem). Column 4 and 5 of Table 5.5 show that BonnRoute rarely creates minimum space violations between segments of different nets, but leaves a few minimum space violations between segments of the same net. It turns out that BonnRoute violates the minimum area rule and the short edge rule only for the 65 nm technology because both rules are automatically fulfilled in older technologies as long as all segments are assigned to pre-defined tracks on the routing grid, which in turn is defined by the minimum pitch. All other rules listed in Table 5.5 are respected by BonnRoute with only very few exceptions.

Although BonnRoute has proved to be a very successful routing tool, Table 5.5 shows that the final wiring still contains a few routing errors which are left to be fixed more or less manually by the designer. The challenge for BonnRoute is to decrease the total number of failures - in the best case to zero. The only class of failures which might not get fixed by any routing tool are opens due to non-accessible pins or congested regions. Such an instance has to be thoroughly investigated for the reasons of the routing failures. Power structure or the placement may cause non-accessible pins and, therefore, have be revised. In the case of routing congestion, placement changes are required before rerouting the design. In rare cases, a designer has to fix the problem manually. It is generally accepted that about 50 remaining failures per one million nets can be expected to be fixed manually by a designer with current design sizes.

Chip	#Nets ($\times 10^3$)	Opens	MinSpace diff	MinSpace same	Min Area	Min EArea	Short Edge	Via Rel	Others	WIE
130 nm technology										
Bill	11									
Elena	421	6		16					1	23
Heidi	777									
Garry	828	59		3					9	71
Ralf	1 350	1				1		1	1	4
Monika	1 503	10				4			2	16
Hermann	2 332	4		5					3	12
90 nm technology										
Paul	68	4							3	7
Hannelore	140	1		16						17
Edgar	772	25		6		113				144
Edmund	2 609	222		21		4		51	5	303
David	5 751	65	11	33				22	4	135
65 nm technology										
Tara	509	17		30	14	1			1	63
Benedikt	528	7			7				1	15
Tina	534			1	2	1			8	12
Dorothea	679	34		8	58		12		6	118
Larry	3 589	10		3	58	5			20	96

Table 5.5: Results on wiring integrity errors of BonnRoute (empty entries stand for 0). Columns 3–10 give detailed statistics on the problems BonnRoute leaves after routing. In column 10 all failures not described in this section are summed up. The last column contains the total sum of all failures. The following abbreviations are used in the table: *MinSpace diffnet*: Minimum space violations between segments of different nets; *MinSpace same*: Minimum space violations between segments of the same net; *Min EArea*: Minimum Enclosed Area; *Via Rel*: Via Reliability; *WIE*: wiring integrity errors.

5.3.3 Manufacturing Yield

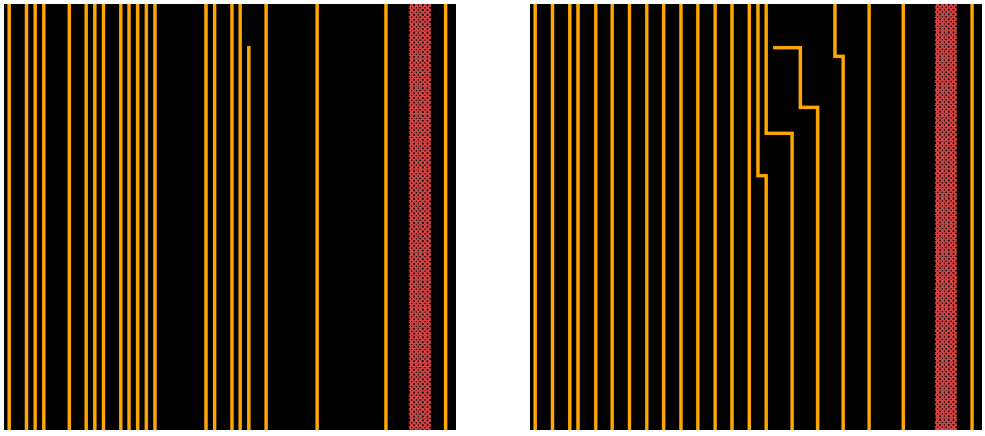
BonnRoute offers optional functionality to be driven in a yield aware mode. Yield is the average proportion of chips on the wafer without any defect. A widely used method to compute the expected number of faults within chip-level wiring is the Monte Carlo dot-throwing approach (Maly [1985]). It computes critical area values planewise, sums up these values and finally multiplies the result with chip area.

Based on a formulation of the global routing problem by Vygen [2004] which takes yield into account, Müller [2006] presents results of BonnRoute showing that the expected num-

ber of defects in wiring can be reduced by more than 10 % on state-of-the-art industrial chips when driving BonnRoute in yield aware.

Yield can also be improved in detailed routing by adding links to the wiring of a net to create redundancy for wires and vias. Particularly, via opens highly contribute to yield loss. Therefore, so-called *redundant vias* are inserted into full chip wiring. That is, each via is substituted by a multi-cut via wherever applicable. However, this approach is less efficient for newer technologies as it requires additional, partly wrong-way metal on one of the adjacent wiring layers. The percentage of successfully added redundant vias highly depends on technology and density of the design. Experiments of Bickford et al. [2006] show a variance between 88 % in a 130 nm technology and below 70 % in a 90 nm technology. For that reason, BonnRoute has incorporated another post-processing method which inserts *local loops* to a fully wired design. It considerably increases the percentage of redundant vias compared to previous approaches. Addition of local loops in the vicinity of single vias increases the robustness to via opens and high resistance vias. It does not generate wrong-way wiring, and the impact on timing is negligible. As it is applied after the design is fully wired, it has also no impact on wirability. Bickford et al. [2006] show that it is superior to add first local loops and insert redundant vias afterwards. This approach achieves a significant reduction in via open critical area by 82 % on average. (The insertion of redundant vias without local loops reduces critical area by 77 % only.) In 2007, BonnRouteLocal was successfully used to complete the chip David for fabrication including the insertion of local loops.

Furthermore, BonnRoute has been recently utilized to introduce so-called *global loops* to wiring for two main reasons: first, adding global loops increases robustness against open defects further and is more efficient in the sense that more wiring and vias are protected relative to the number of wires and vias added. Second, global loops reduce on-chip variations in timing (Panitz et al. [2007]).



Before wire spreading After wire spreading

Figure 5.3: Part of wiring of chip Elena on wiring layer 4

Wire spreading can also be done as a post-processing routine in BonnRouteLocal where a reduction of the critical area by 1 % to 10 % can be attained (Schulte [2006]). A small part of the fourth wiring layer of chip Elena is depicted in Figure 5.3 where the effect of wire spreading is apparent.

Although all of the above methods are expensive with respect to running time, it is worthwhile to spend extra time in a final call of BonnRoute after having met all specified design rules and constraints.

Bibliography

Agarwal, P. K. and Shing, M.-T. [1990]: Algorithms for the special cases of rectilinear Steiner trees: I. points on the boundary of a rectilinear rectangle. *Networks* 20, pp. 453–485, 1990.

Albrecht, C. [2001]: Global routing by new approximation algorithms for multicommodity flow. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 20, pp. 622–632, 2001.

Alpert, C. J., Gandham, G., Hrkic, M., Hu, J., Kahng, A. B., Lillis, J., Liu, B., Quay, S. T., Sapatnekar, S. S., Sullivan, A. J. [2002]: Buffered Steiner Trees for Difficult Instances. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 21, pp. 3–14, 2002.

Alpert, C. J., Kahng, A. B, Sze, C. N. and Wang, Q. [2006]: Timing-Driven Steiner Trees are (Practically) Free. In *Proceedings of the 43rd Conference on Design Automation (DAC'06)*, pp. 389–392, 2006.

Alpert, C. J., Hu, T. C., Huang, J. H. and Kahng, A. B. [1993]: A Direct Combination of the Prim and Dijkstra Constructions for Improved Performance-Driven Global Routing. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, pp. 1868–1872, 1993.

Alpert, C. J., Kahng, A. B., Liu, B., Mandoiu, I. I. and Zelikovsky, A. [2001]: Minimum-Buffered Routing of Non-Critical Nets for Slew Rate and Reliability Control. In *Proceedings of the 2001 IEEE/ACM International Conference on Computer-Aided Design*, pp. 408–415, 2001.

Alpert, C. J., Mehta D. P. and Sapatnekar, S. S. [2008]: Handbook of Algorithms for Physical Automation. CRC Press, to appear in 2008.

Aneja, Y. P. [1980]: An Integer Linear Programming Approach to the Steiner Problem in Graphs. *Networks* 10, pp. 167–178, 1980.

Arora, S. [1998]: Polynomial Time Approximation Schemes for the Euclidean Traveling Salesman and other Geometric Problems. *Journal of the ACM* 45, pp. 753–782, 1998.

Bast, H., Funke, S., Matijevic, D., Sanders, P. and Schultes D. [2007]: In Transit to Constant Time Shortest-Path Queries in Road Networks. In *Proceedings of the 9th Workshop on Algorithm Engineering and Experiments (ALENEX 07)*, 2007.

Batterywala, S. Shenoy, N., Nicholls, W. and Zhou, H. [2002]: Track Assignment: A Desirable Intermediate Step Between Global Routing and Detailed Routing. In *Proceedings of the 2002 IEEE/ACM International Conference on Computer-Aided Design*, pp. 59–66, 2002.

Bellman, R. [1958]: On a routing problem. *Quarterly of Applied Mathematics* 16, pp. 87–90, 1958.

Berman, P. and Ramaiyer, V. [1994]: Improved Approximations for the Steiner tree problem. *Journal of Algorithms* 17, pp. 381–408, 1994.

Bickford, J., Hibbeler, J., Bühler, M., Koehl, J., Müller, D., Peyer, S. and Schulte, C.: Yield Improvement by Local Wiring Redundancy. In *Proceedings of the 7th International Symposium on Quality Electronic Design (ISQED)*, pp. 473-478, 2006.

Boese, K. D., Kahng, A. B., McCoy, B. A. and Robins, G. [1994]: Rectilinear Steiner Trees with Minimum Elmore Delay. In *Proceedings of the 31th Conference on Design Automation (DAC'94)*, pp. 381–386, 1994.

Boese, K. D., Kahng, A. B. and Robins, G. [1993]: High-Performance Routing Trees With Identified Critical Sinks. In *Proceedings of the 30th Conference on Design Automation (DAC'93)*, pp. 182–187, 1993.

Boese, K. D., Kahng, A. B., McCoy, B. A. and Robins, G. [1995a]: Rectilinear Steiner Trees with Minimum Elmore Delay. Computer Science Department, University of California, LA, 1995.

Boese, K. D., Kahng, A. B., McCoy, B. A. and Robins, G. [1995b]: Near-optimal Critical Sink Routing Tree Constructions. *IEEE Transactions on Computer-Aided Design* 14(12), pp. 1417–1436, 1995.

Borchers, A., Du, D.-Z., Gao, B. and Wan, P. [1998]: The k-Steiner Ratio in the Rectilinear Plane. *Journal of Algorithms* 29, pp. 1–17, 1998.

Bozorgzadeh, E., Kastner, R. and Sarrafzadeh, M. [2001]: Creating and Exploiting Flexibility in Steiner Trees. In *Proceedings of the 38th Conference on Design Automation (DAC'01)*, pp. 195–198, 2001.

Brenner, U. [2005]: *Theory and Practice of VLSI Placement*. PhD thesis, Institute for Discrete Mathematics, University of Bonn, 2005.

Brenner, U. and Rohe, A. [2003]: An effective congestion-driven placement framework. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 22, pp. 387–394, 2003.

Carden IV, R. C., Li, J. and Cheng, C.-K. [1996]: A global router with a theoretical bound on the optimal solution. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 15, pp. 208–216, 1996.

Chen, T.-C., Chang, Y.-W. and Lin, S.-C. [2006]: A Novel Framework for Multilevel Full-Chip Gridless Routing. In *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 636–641, 2006.

Chen, H., Cheng, C.-K., Kahng, A., Mandoiu, I. .I., Wang, Q. and Yao, B. [2003]: The Y-Architecture for On-Chip Interconnect: Analysis and Methodology. In *Proceedings of the 40th Conference on Design Automation (DAC'03)*, pp. 13–19, 2003.

Chen, D. Z., Klenk, K. S. and Tu, H. T. [2000]: Shortest path queries among weighted obstacles in rectilinear plane. *SIAM Journal on Computing* 29, pp. 1223–1246, 2000.

Chen, W., Pedram, M. and Buch, P. [2002]: Buffered Routing Tree Construction Under Buffer Placement Blockages. In *Proceedings of 7th ASP-DAC and 15th International Conference on VLSI Design*, pp. 381–386, 2002.

Chen, D. Z. and Xu, J. [2001]: An efficient direct approach for computing shortest rectilinear paths among obstacles in a two-layer interconnection model. *Computational Geometry* 18, pp. 155–166, 2001.

Chen, H., Yao, B., Zhou, F. and Cheng, C. K. [2003]: The Y-Architecture: Yet Another On-Chip Interconnect Solution. In *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 840–846, 2003.

Cherkassky, B. V., Goldberg, A. V. and Radzik, T. [1996]: Shortest Paths Algorithms: Theory and Experimental Evaluation. *Mathematical Programming* 73, pp. 129–174, 1996.

Chiang, C. and Sarrafzadeh, M. [1991]: Wirability of knock-knee layouts with 45°wires. In *IEEE Transactions on Circuits and Systems* 38, pp. 613–624, 1991.

Chiang, C., Sarrafzadeh, M. and Wong, C. K. [1992]: An Algorithm for Exact Rectilinear Steiner Trees for Switchbox with Obstacles. In *IEEE Transactions on Circuits and Systems — I: Fundamental Theory and Applications* 39, pp. 446–455, 1992.

Cohoon, J. P., Richards, D. S. and Salowe, J. S. [1990]: An optimal Steiner tree routing algorithm for a net whose terminals lie on the perimeter of a rectangle. In *IEEE Transactions on Computer-Aided Design* 9, pp. 398–407, 1990.

Cong, J., Fang, J. and Khoo, K. [2001]: DUNE — A Multilayer Gridless Routing System. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 20, pp. 633–647, 2001.

Cong, J., Fang, J., Xie, M. and Zhang, Y. [2005]: MARS — A Multilevel Full-Chip Gridless Routing System. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 24, pp. 382–394, 2005.

Cong, J., He, L., Khoo, K.-Y., Koh, C.-K. and Pan Z. [1997]: Interconnect Design for Deep Submicron ICs. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design; Digest of Technical Papers (ICCAD '97)*, pp. 478–487, 1997.

Cong, J., Kahng, A. B., Robins, G., Sarrafzadeh, M. and Wong, C. K. [1992]: Provably Good Performance-Driven Global Routing. *IEEE Transactions on Computer-Aided Design* 11(6), pp. 739–752, 1992.

Cong, J., Leung, K.-S. and Zhou, D. [1993]: Performance-Driven Interconnect Design Based on Distributed RC Delay Model. In *Proceedings of the 30th Conference on Design Automation (DAC'93)*, pp. 606–611, 1993.

Dantzig, G.B. [1957]: Discrete-Variable Extremum Problems. *Operations Research* 5, pp. 266–277, 1957.

Dantzig, G.B. [1960]: On the Shortest Route through a Network. *Management Science* 6, pp. 187–190, 1960.

Dechu, S., Shen, Z. C., and Chu, C. C [2005]: An Efficient Routing Tree Construction Algorithm With Buffer Insertion, Wire Sizing and Obstacle considerations. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 24, pp. 600–607, 2005.

Devadas, S., Ghosh, A. and Keutzer, K. W. [1994]: Logic Synthesis. McGraw-Hill Series on Computer Engineering, 1994.

Dial, R. B. [1969]: Algorithm 360: shortest-path forest with topological ordering. *Communications of the ACM* 12, pp. 632–633, 1969.

Dijkstra, E. W. [1959]: A note on two problems in connexion with graphs. *Numerische Mathematik* 1, pp. 269–271, 1959.

9th DIMACS Implementation Challenge: The Shortest Path Problem (2006). http://www.dis.uniroma1.it/˜challenge9/papers.shtml

Dobes, I. R. [1977]: A multi-contouring algorithm. *ACM SIGDA Newsletter* 7, pp. 2–11, 1977.

Doran, J. [1967]: An Approach to Automatic Problem-Solving. *Machine Intelligence* 1, pp. 105–127, 1967.

Dunne, G. V. [1967]: The design of printed circuit layouts by computer, In *Proceedings of the 3rd Australian Computer Conference*, pp. 419–423, 1967.

Elmore, W. C. [1948]: The Transient Response of Damped Linear Networks with Particular Regard to Wideband Amplifiers. *Journal of Applied Physics* 19, pp. 55–63, 1948.

Fisk, C. J., Caskey, D. L. and West, L. E. [1967]: Topographic simulation as an aid to printed circuit board design. In *Proceedings of the 4th Conference on Design Automation (DAC'67)*, pp. 17.1–17.23, 1967.

Fitch, R., Butler, Z. and Rus, D. [2001]: 3D Rectilinear Motion Planning with Minimum Bend Paths, In *Proceedings of the International Conference on Intelligent Robots and Systems*, 2001.

Fredman, M. L. and Tarjan, R. E., [1987]: Fibonacci heaps and their uses in improved network optimization problems. *Journal of the ACM* 34, pp. 596–615, 1987.

Ford Jr, L. R., [1956]: Network Flow Theory, Paper P-923, The RAND Corporation, 1956.

Gallo, G. and Pallottino, S. [1988]: Shortest Paths Algorithms. *Annals of Operations Research* 13, pp. 3–79, 1988.

Ganley, J. L. and Cohoon, J. P. [1994]: Routing a Multi-Terminal Critical Net: Steiner tree construction in the presence of obstacles. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, pp. 113–116, 1994.

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

Garg, N. and Könemann, J. [1998]: Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In *Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science*, pp. 300–309, 1998.

Gerez, S. H. [1998]: Algorithms For VLSI Design Automation. Wiley, 1998.

Goldberg, A. V. [2001]: A Simple Shortest Path Algorithm with Linear Average Time. In *Proceddings of the 9th European Symposium on Algorithms (ESA 2001)*, Lecture Notes in Computer Science (LNCS) 2161, 230–241, 2001.

Goldberg A. V. and Harrelson, C. [2005]: Computing the Shortest Path: A* Search Meets Graph Theory. In *Proceedings. of the 16th Annual (ACM-SIAM) Symposium on Discrete Algorithms (SODA)*, pp. 156–165, 2005.

Gutman, R [2004]: Reach-based routing: A new approach to shortest path algorithms optimized for road networks. In *Proceedings of the 6th Workshop on Algorithm Engineering and Experiments (ALENEX'04)*, pp. 100–111, 2004.

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

Hamachi, G. T. and Ousterhout, J. K. [1984]: A switchbox router with obstacle avoidance. In *Proceedings of the 21st Design Automation Conference (DAC'84)*, pp. 173–179, 1984.

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

Hart, P. E., Nilsson, N. J. and Raphael, B. [1968]: A formal basis for the heuristic determination of minimum cost paths in graphs. *IEEE Transactions on Systems Science and Cybernetics, SSC* 4, pp. 100–107, 1968.

Hashimoto, A. and Stevens, J. [1971]: Wire routing by optimizing channel assignment within large apertures. In *Proceedings of the 8th Design Automation Conference (DAC'71)*, pp. 155–169, 1971.

Heiss, R. [1968]: A path connection algorithm for multi-layer boards. In *Proceedings of the 5th Conference on Design Automation (DAC'68)*, pp. 6.1–6.14, 1968.

Hetzel, A. [1995]: *Verdrahtung im VLSI-Design: Spezielle Teilprobleme und ein sequentielles Lösungsverfahren*. PhD thesis, Institute for Discrete Mathematics, University of Bonn, 1995.

Hetzel, A. [1998]: A sequential detailed router for huge grid graphs. In *Proceedings of Design, Automation and Test in Europe (DATE 1998)*, pp. 332–339, 1998.

Heyns, W., Sansen, W. and Beke, H.: A Line-Expansion Algorithm for the General Routing Problem with a Guaranteed Solution. In *Proceedings of the 17th Design Automation Conference*, pp. 243–249, 1980.

Hightower, D. W. [1969]: A solution to line-routing problems on the continuous plane. In *Procesdings of the 6th Design Automation Conference*, pp. 1–24, 1969.

Hitchcock, R. B. [1969]: Cellular wiring and the cellular modeling technique. In *Proceedings of the 6th Conference on Design Automation (DAC'69)*, pp. 25–41, 1969.

Ho, T.-Y., Chang, C.-F., Cheang, Y.-W. and Chen, S.-J. [2005]: Multilevel full-chip routing for the X-based architecture In *Proceedings of the 42nd Conference on Design Automation (DAC'05)*, pp. 597–602, 2005.

Ho, T.-Y., Chang, Y.-W., Chen, S.-J. and Lee, D. T. [2003]: A Fast Crosstalk- and Performance-Driven Multilevel Routing System. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design (ICCAD '03)*, pp. 382–387, 2003.

Hoel, J. H. [1976]: Some Variations of Lee's Algorithm. *IEEE Transactions on Computers* 25, pp. 19–24, 1976.

Holzer, M., Schulz, F. and Wagner, D. [2006]: Engineering multi-level overlay graphs for shortest-path queries. In *Proceedings of the 8th Workshop on Algorithm Engineering and Experiments (ALENEX'06)*, 2006.

Holzer, M., Schulz, F., Wagner, D. and Willhalm, T. [2005]: Combining speed-up techniques for shortest-path computations. *Journal of Experimental Algorithmics (JEA)* 10, 2005.

Hrkic, M. and Lillis, J. [2003]: Buffer Tree Synthesis with Consideration of Temporal Locality, Sink Polarity Requirements, Solution Cost, Congestion and Blockages. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 22, pp. 481-491, 2003.

Hu, J., Alpert, C. J., Quay, S. T. and Gandham, G. [2002]: Buffer Insertion with Adaptive Blockage Avoidance. In *Proceedings of International Symposium on Physical Design (ISPD'02)*, pp. 92–97, 2002.

Hu, J., Hou, H. and Sapatnekar, S. S. [1999]: Non-Hanan Routing. *IEEE Transactions on Computer-Aided Design* 18(4), pp. 436–444, 1999.

Huijbregts, E. P. and Jess, J. A. G. [1993]: General Gate Array Routing using a k–Terminal Net Routing Algorithm with Failure Prediction. In *IEEE Transactions on VLSI Systems* 1, pp. 473–481, 1993.

Huijbregts, E. P., Xue, H. and Jess, J. A. G. [1995]: Routing for Reliable Manufacturing. *IEEE Transactions on Semiconductor Manufacturing* 8, pp. 188–194, 1995.

Hwang, F. K. [1976]: On Steiner minimal trees with rectilinear distance. *SIAM Journal on Applied Mathematics* 30, pp. 104–114, 1976.

Hwang, F. K., Richards, D. S. and Winter, P. [1992]: *The Steiner Tree Problem.* Annals of Discrete Mathematics 53. Elsevier Science Publishers, Netherlands, 1992.

IEEE Standard VHDL Language Reference Manual. IEEE Publications, 1994.

Johann, M. and Reis, R. [2000]: Net by Net Routing with a New Path Search Algorithm. In *Proceedings of the 13th Symposium on Integrated Circuits and Systems Design* 19, pp. 144–149, 2000.

Johnson, D. B. [1977]: Efficient Algorithms for Shortest Paths in Sparse Networks. *Journal of the ACM* 24, pp. 1–13, 1977.

Kahng, A. B., Liu, B. and Mandoiu, I. I. [2002]: Non-tree routing for reliability and yield improvement. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD '02)*, pp. 260–266, 2002.

Kahng, A. B. and Liu, B [2003]: Q-Tree: A New Iterative Improvement Approach for Buffered Interconnect Optimization. In *Proceedings of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI'03)*, pp. 183-188, 2003.

Kahng, A. B. and Robins, G. [1995]: *On Optimal Interconnections for VLSI.* Kluwer Academic Publishers, Boston, 1995.

Kay, R. and Rutenbar, R. A. [2001]: Wire Packing - A Strong Formulation of Crosstalk-Aware Chip-Level Track/Layer Assignment with an Efficient Integer Programming Solution. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 20, pp. 672-679, 2001.

Khuller, S., Raghavachari, B. and Young, N. [1995]: Balancing Minimum Spanning and Shortest-Path Trees. *Algorithmica* 14(4), pp. 305–321, 1995.

Klunder, G. A. and Pst, H. N. [2006]: The Shortest Path Problem on Large-Scale Real-Road Networks. *Networks* 48, pp. 182–194, 2006.

Koch, T. and Martin, A. [1998]: Solving Steiner Tree Problems in Graphs to Optimality. *Networks* 33, pp. 207–232, 1998.

Korte, B., Rautenbach, D. and Vygen, J. [2007]: BonnTools: Mathematical innovation for layout and timing closure of systems on a chip. *Proceedings of the IEEE* 95, pp. 555–572, 2007.

Kramer, M. R. and van Leeuwen, J. [1984]: The complexity of wire-routing and finding the minimum area layouts for arbitrary VLSI circuits. In: Advances in Computing Research 2: VLSI Theory (F.P. Preparata, ed.), pp. 129–146, JAI Press, London 1984.

Lavin, M. and Liebmann, L. [2002]: CAD Computation for Manufacturing: Can We Save VLSI Technology from Itself? In *Proceedings of the 2002 IEEE/ACM International Conference on Computer-Aided Design*, pp. 424–431, 2002.

Lee, C. [1961]: An algorithm for path connections and its applications. *IRE Transactions on Electronic Computing* EC-10, pp. 346–365, 1961.

Lengauer, T. [2004]: Combinatorial Algorithms For Integrated Circuit Layout. Wiley, 1994.

Li, Y.-L., Chen, H.-Y. and Lin, C.-T. [2007]: NEMO: A New Implicit-Connection-Graph-Based Gridless Router With Multilayer Planes and Pseudo Tile Propagation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 26, pp. 705–718, 2007.

Lin, C-W., Chen, S.-Y., Li, C.-F., Chang, Y.-W. and Yang, C.-L. [2007]: Efficient Obstacle-Avoiding Rectilinear Steiner Tree Construction. In *Proceedings of International Symposium on Physical Design (ISPD'07)*, pp. 127–134, 2007.

Lin, L., Liu , Y. and Hwang, T. [2001]: Construction of Minimal Delay Steiner Tree Using Two-pole Delay Model. In *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 126–132, 2001.

Lipski, Jr., W. [1984]: An $O(n \log n)$ Manhattan path algorithm. *Information Processing Letters* 19, pp. 99–102, 1984.

Lodi, E. [1988]: Routing Multiterminal Nets in a Diagonal Model. In *Proceedings of the 1988 Conference on Information Sciences and Systems*, pp. 899–902, 1988.

Luby, M. and Ragde, P. L. [1989]: A bidirectional shortest-path algorithm with good average-case behavior. *Algorithmica* 4, pp. 551–567, 1989.

Maly, W. [1985]: Modeling of lithography related yield losses for CAD of VLSI circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 4, pp. 166–177, 1985.

Margarino, A., Romano, A., De Gloria, A., Curatelli, F. and Antognetti, P. [1987]: A Tile-Expansion Router. *IEEE Transactions on CAD of Integrated Circuits and Systems* 6, pp. 507–517, 1987.

Marx, D. [2004]: Eulerian disjoint paths problem in grid graphs is NP-complete. *Discrete Applied Mathematics* 143, pp. 336–341, 2004.

McCoy, B. A. and Robins, G [1994]: Non-Tree Routing. In *Proceedings of the European Conference on Design Automation*, pp. 430–434, 1994.

Medhi, D. and Ramasamy, K. [2007]: Network Routing: Algorithms, Protocols, and Architectures. Morgan Kaufmann, 2007.

Mehlhorn, K. [1988]: A faster approximation algorithm for the Steiner problem in graphs. *Information Processing Letters* 27, pp. 125–128, 1988.

Meyer, U. [2001]: Single-Source Shortest Paths on Arbitrary Directed Graphs in Linear Average Time. In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 797–806, 2001.

Mikami, K. and Tabuchi, K. [1968]: A computer program for optimal routing of printed circuit conductors. *International Federation for Information Processing (IFIP)* Congress, pp. 1475-1478, 1968.

Minty, G. J. [1957]: A comment on the shortest-route problem. *Operations Research* 5, p. 724, 1957.

Miriyala, S., Hashmi, J. and Sherwani, N. [1991]: Switchbox Steiner tree problem in presence of obstacles. In *EEE/ACM International Conference on Computer Aided Design; Digest of Technical Papers (ICCAD '91)*, pp. 536–539, 1991.

Mitchell, J. S. B. [1999]: Guillotine Subdivisions Approximate Polygonal Subdivisions: A Simple Polynomial-Time Approximation Scheme for Geometric TSP, k-MST, and Related Problems. *SIAM Journal on Computing* 28, pp. 1298–1309, 1999.

Möhring, R. H., Schilling, H., Schütz, B., Wagner, D. and Willhalm, T. [2005]: Partitioning graphs to speed up Dijkstra's algorithm. In *Proceedings of the 4th International Workshop on Efficient and Experimental Algorithms (WEA 2005)*, Lecture Notes in Computer Science (LNCS) 3503, pp. 189–202, 2005.

Moore, E. F. [1959]: The shortest path through a maze. In *Proceedings of the International Symposium on the Theory of Switching*, pp. 285–292, 1959.

Müller, D. [2002]: Bestimmung der Verdrahtungskapazitäten im Global Routing von VLSI-Chips. Master's thesis, Research Institute for Discrete Mathematics, University of Bonn, 2002.

Müller, D. [2006]: Optimizing yield in global routing. In *IEEE/ACM International Conference on Computer Aided Design; Digest of Technical Papers (ICCAD 06)*, pp. 480–486, 2006.

Müller-Hannemann, M. and Peyer, S. [2003]: Approximation of Rectilinear Steiner Trees with Length Restrictions on Obstacles. In *Proceedings of the 8th Workshop on Algorithms and Data Structures (WADS 2003)*, Lecture Notes in Computer Science (LNCS) 2748, pp. 207–218, 2003.

Müller-Hannemann, M., Schulz, F., Wagner, D. and Zaroliagis, C. [2007]: Timetable Information: Models and Algorithms. Lecture Notes in Computer Science (LNCS) 4359, pp. 67-89, 2007.

Müller-Hannemann, M. and Schulze, A. [2006]: Approximation of Octilinear Steiner Trees Constrained by Hard and Soft Obstacles. In *Proceedings of the 10th Scandinavian Workshop on Algorithm Theory (SWAT 2006)*, Lecture Notes in Computer Science (LNCS) 4059, pp. 242–254, 2006.

Müller-Hannemann, M. and Tazari, S. [2007]: A Near Linear Time Approximation Scheme for Steiner Tree among Obstacles in the Plane. In *Proceedings of the 10th Workshop on Algorithms and Data Structures (WADS 2007)*, Lecture Notes in Computer Science (LNCS) 4619, pp. 151-162, 2007.

Müller-Hannemann, M. and Weihe, K. [2001]: Pareto Shortest Paths is Often Feasible in Practice. In *Proceedings of the 5th International Workshop on Algorithm Engineering*, Lecture Notes in Computer Science (LNCS) 2141, pp. 185–198, 2001.

Müller-Hannemann, M. and Zimmermann, U. [2003]: Slack Optimization of Timing-Critical Nets. In *Proceedings of the 11th Annual European Symposium on Algorithms (ESA 2003)*, Lecture Notes in Computer Science (LNCS) 2832, pp. 727–739, 2003.

Naor, J. and Schieber, B. [1997]: Improved Approximations for Shallow-Light Spanning Trees. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science*, pp. 536–541, 1997.

Nastansky, L., Selkow, S. M. and Stewart, N. F. [1974]: Cost-Minimal Trees in Directed Acyclic Graphs. *Z. Oper. Res.* 18, pp. 59–67, 1974.

Natarajan, S., Sherwani, N., Holmes, N. D. and Sarrafzadeh, M. [1992]: Over-the-Cell Channel Routing For High Performance Circuits. In *Proceedings of the 29th Conference on Design Automation (DAC'92)*, pp. 600–603, 1992.

Nishizeki, T,, Vygen, J. and Zhou, X. [2001]: The edge-disjoint paths problem is NP-complete for series-parallel graphs. *Discrete Applied Mathematics* 115, pp. 177–186, 2001.

Orden, A. [1956]: The transhipment problem. *Management Science* 2, pp. 276–285, 1956.

Panitz, P. V., Olbrich, M., Barke, E. and Koehl, J. [2007]: Robust Wiring Networks for DfY Considering Timing Constraints. In *Proceedings of the 17th Great Lakes Symposium on VLSI 2007*, pp. 43–48, 2007.

Panten, C. [2005]: Paralleles Verdrahten von VLSI-Chips auf Shared-Memory-Basis. Master's thesis, Research Institute for Discrete Mathematics, University of Bonn, 2005.

Peyer, S. [2000]: Elmore-Delay-optimale Steinerbäume im VLSI-Design. Master's thesis, Research Institute for Discrete Mathematics, University of Bonn, 2000.

Peyer, S., Rautenbach, D. and Vygen, J. [2006]: A Generalization of Dijkstra's Shortest Path Algorithm with Applications to VLSI Routing. Report No. 06964, Research Institute for Discrete Mathematics, University of Bonn, 2006.

Peyer, S., Zachariasen, M. and Jørgensen, D. G. [2004]: Delay-related secondary objectives for rectilinear Steiner minimum trees. *Discrete Applied Mathematics* 136, pp. 271–298, 2004.

Pohl, I. [1971]: Bi-directional Search. *Machine Intelligence* 6, pp. 124–140, 1971.

Prasitjutrakul, S. and Kubitz, W. J. [1990]: A Timing-Driven Global Router for Custom Chip Design. In *Proceedings of the International Conference on Computer-Aided Design*, pp. 606–611, 1990.

Prömel, H. J. and Steger, A. [2002]: The Steiner Tree Problem: A Tour Through Graphs, Algorithms, and Complexity. Advanced Lectures in Mathematics, Vieweg, 2002.

Prabhakar, R. and Thompson, C. D. [1987]: Randomized rounding: a technique for provably good algorithms and algorithmic proofs. *Combinatorica* 7, pp. 365–374, 1987.

Raith, M. and Bartholomeus, M. [1991]: A new hypergraph based rip-up and reroute strategy. In *Proceedings of the 28th Conference on Design Automation (DAC'91)*, pp. 54–59, 1991.

Rivest, R. L. and Fiduccia, C, M. [1982]: A greedy channel router. In *Proceedings of the 19th Design Automation Conference (DAC82)*, pp. 418–424, 1982.

Robins, G. and Zelikovsky, A. [2005]: Tighter Bounds for Graph Steiner Tree Approximation. *SIAM Journal on Discrete Mathematics* 19, pp. 122–134, 2005.

Rohe, A. [2001]: Sequential and Parallel Algorithms for Local Routing. PhD thesis, Institute for Discrete Mathematics, University of Bonn, 2001.

Rubin, F. [1974]: The Lee Path Connection Algorithm. *IEEE Transactions on Computers* C-23, pp. 907–914, 1974.

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

Sanders, P. and Schultes, D. [2005]: Highway Hierarchies Hasten Exact Shortest Path Queries. In *Proceedings of the 13th Annual European Symposium on Algorithms (ESA 2005)*, Lecture Notes in Computer Science (LNCS) 3669, pp. 568–579, 2005.

Sanders, P. and Schultes, D. [2006]: Engineering Highway Hierarchies. ESA 2006: 804-816 In *Proceedings of the 14th Annual European Symposium on Algorithms (ESA 2006)*, pp. 804–816, 2005.

Sanders, P. and Schultes, D. [2007]: Engineering Fast Route Planning Algorithms. In *Proceedings of the 6th International Workshop on Efficient and Experimental Algorithms (WEA 2007)*, Lecture Notes in Computer Science (LNCS) 4525, pp. 23–36, 2007.

Sato, M., Kubota, K. and Ohtsuki, T. [1990]: A hardware implementation of gridless routing based on content addressable memory. In *Proceedings of the 27th Design Automation Conference*, pp. 646–649, 1990.

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

Schrijver, A. [2005]: On the History of Combinatorial Optimization (till 1960). In: Handbook of Discrete Optimization (K. Aardal, G.L. Nemhauser, R. Weismantel, eds.), Elsevier, Amsterdam, pp. 1–68, 2005.

Schulte, C. [2006]: Yield-Optimierung im Detailed Routing. Master's thesis, Research Institute for Discrete Mathematics, University of Bonn, 2006.

Schulz, F., Wagner, D. and Weihe, K. [2000]: Dijkstra's algorithm on-line: An empirical case study from public railroad transport. *Journal of Experimental Algorithmics* 5, 2000.

Schulz, F., Wagner, D. and Zaroliagis, C. [2002]: Using Multi-Level Graphs for Timetable Information in Railway Systems. In *Proceedings of the 4th International Workshop on Algorithm Engineering and Experiments (ALENEX 02)*, Lecture Notes in Computer Science LNCS 2409, pp. 43–59, 2002.

Sedgewick, R. and Vitter, J. [1986]: Shortest Paths in Euclidean Graphs. *Algorithmica* 1, pp. 31–48, 1986.

Shahrokhi, F. and Matula, D. W. [1990]: The Maximum Concurrent Flow Problem. *Journal of the ACM* 37, pp. 318–334, 1990.

Shenoy, N. V. and Nicholls, W. [2002]: An efficient routing database. In *Proceedings of the 39th Design Automation Conference (DAC02)*, pp. 590–595, 2002.

Sherwani, N. A. [1999]: Algorithms For VLSI Physical Design Automation. Kluwer, 1999.

Shimbel, A. [1955]: Structure in communication nets. In *Proceedings of the Symposium on Information Networks*, pp. 199–203, 1955.

Soukup, J. [1978]: Fast Maze Router. In *Proceedings of the 15th Design Automation Conference*, pp. 100–102. 1978.

Teig, S. [2002]: The X architecture: not your father's diagonal wiring. In *Proceedings of the 2002 international workshop on system-level interconnect prediction*, pp. 33–37, 2002.

Thomas, D. E. and Moorby, P. R. [2002]: The Verilog Hardware Description Language. Kluwer, 2002.

Thorup, M. [1999]: Undirected Single-Source Shortest Paths with Positive Integer Weights in Linear Time. *Journal of the ACM* 46, pp. 362–394, 1999.

Thorup, M. [2004]: Integer priority queues with decrease key in constant time and the single source shortest paths problem. *Journal of Computer and System Sciences* 69, pp. 330–353, 2004.

McDonald, R., Burger, D. and Keckler, S. [2005]: The Design and Implementation of the TRIPS Prototype Chip. *Symposium on High Performance Chips (HotChips)* 17, 2005.

Tseng, H.-P. and Sechen, C. [1999]: A Gridless Multilayer Router for Standard Cell Circuits Using CTM Cells. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 18, pp. 1462–1479, 1999.

Tseng, H.-P., Scheffer, L. and Sechen, C. [1999]: Timing- and Crosstalk-Driven Area Routing. *IEEE Transactions on CAD of Integrated Circuits and Systems* 20, pp. 528–544, 2001.

Vygen, J. [1995]: NP-completeness of some edge-disjoint paths problems. *Discrete Applied Mathematics* 61, pp. 83–90, 1995.

Vygen, J. [2001]: Theory of VLSI Layout. Habilitation, University of Bonn, 2001.

Vygen, J. [2004]: Near-Optimum Global Routing with Coupling, Delay Bounds, and Power Consumption. In *Proceedings of the 10th Integer Programming and Combinatorial Optimization (IPCO 2004)*, Lecture Notes in Computer Science (LNCS) 3064, pp. 308–324, 2004.

Wagner, D. and Willhalm, T. [2003]: Geometric Speed-Up Techniques for Finding Shortest Paths in Large Sparse Graphs. In *Proceedings of the 11th Annual European Symposium on Algorithms (ESA 2003)*, Lecture Notes in Computer Science (LNCS) 2832, pp. 776–787, 2003.

Wagner, D. and Willhalm, T. [2007]: Speed-Up Techniques for Shortest-Path Computations. Symposium on Theoretical Aspects of Computer Science (STACS), pp. 23–36, 2007.

Warme, D. M., Winter, P. and Zachariasen, M. [2000]: Exact Algorithms for Plane Steiner Tree Problems: A Computational Study. In D.-Z. Du, J. M. Smith and J. H. Rubinstein, editors, *Advances in Steiner Trees*, pp. 81–116, Kluwer Academic Publishers, Boston, 2000.

Warme, D. M., Winter, P. and Zachariasen, M. [2001]: GeoSteiner 3.1. Department of Computer Science, University of Copenhagen (DIKU), `http://www.diku.dk/geosteiner/`, 2001.

Weste, N. H. E. and Eshraghian, K. [2002]: Principles of CMOS VLSI Design: A Systems Perspective with Verilog/VHDL Manual. Addison Wesley, 2002.

Winter, P. [1995]: Reductions for the Rectilinear Steiner Tree Problem. *Networks* 26, pp. 187–198, 1995.

Xing, Z. and Kao, R. [2002]: Shortest Path Search Using Tiles and Piecewise Linear Cost Propagation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 21, pp. 145–158, 2002.

Zachariasen, M. [1999]: Rectilinear Full Steiner Tree Generation. *Networks* 33, pp. 125–143, 1999.

Zachariasen, M. [2001a]: The Rectilinear Steiner Tree Problem: A Tutorial. In: Steiner Trees in Industry (D.-Z. Du and X. Chen, ed.), pp. 467–507, Kluwer Academic Publisher, Boston, 2001.

Zachariasen, M. [2001b]: A Catalog of Hanan Grid Problems. *Networks* 38, pp. 76–83, 2001.

Zelikovsky ,A. Z. [1992]: An $\frac{11}{8}$-approximation algorithm for the Steiner problem in networks with rectilinear distance. *Coll. Math. Soc. J. Bolyai* 60, pp. 733–745, 1992.

Zheng, S.Q., Joon Shink Lim and Iyengar, S.S. [1996]: Finding Obstacle-Avoiding Shortest Paths Using Implicit Connection Graphs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 15, pp. 103–110, 1996.

Summary

Routing is one of the major steps in very-large-scale integration (VLSI) design. Its task is to find disjoint wire connections between sets of points on a chip, subject to numerous constraints. In this thesis, we present new theoretical results on Steiner trees and shortest paths, the two main mathematical concepts in routing. In the practical part, we give computational results of BonnRoute, a VLSI routing tool developed at the Research Institute for Discrete Mathematics at the University of Bonn.

The VLSI Routing Problem can be seen as a generalized packing problem for Steiner trees in three-dimensional grid graphs, where the trees are subject to further technology-related constraints. To solve this problem, BonnRoute follows a two-stage approach, which consists of so-called global and detailed routing steps. For each set of metal components to be connected, global routing reduces the search space by computing corridors in which detailed routing sequentially determines the desired connections as shortest paths.

The problem of finding a rectilinear Steiner minimum tree (RSMT) for a given set of points in the plane is NP-complete, but it can be solved extremely fast in practice. However, length alone is not the only criterion for today's VLSI instances, since interconnect delays are becoming increasingly important. Therefore, we examine the problem of constructing RSMTs which minimize a signal-delay-related function as a secondary objective. We give a mixed integer programming formulation for the rectilinear Steiner tree problem with the weighted sum of path lengths as secondary objective (RSTPWP), which can be solved to optimality by standard branch-and-bound methods. By deriving structural properties, we prove that each optimal solution to RSTPWP has its Steiner points on the Hanan grid. We additionally present a heuristic algorithm for constructing RSMTs with various secondary objectives. Experiments on industrial chips show that the algorithm improves the delay properties of RSMTs without increasing total tree length, and that it solves more than 98 % of the instances of RSTPWP optimally.

We further consider the problem of finding a shortest rectilinear Steiner tree in the plane in the presence of rectilinear obstacles. The Steiner tree is allowed to run over obstacles; however, if it intersects an obstacle, then no connected component of the induced subtree must be longer than a given fixed length. We show that this problem can be approximated with a performance guarantee of 2 in $O(n \log n)$ time, where n denotes the number of nodes of the Hanan grid defined by terminals and obstacles, and that there are optimal length-restricted Steiner trees with a special structure. In particular, we prove that a certain

graph (called augmented Hanan grid) always contains an optimal solution. Based on this structural result, we give an approximation scheme for the special case that all obstacles are of rectangular shape or are represented by at most a constant number of edges. The restrictions on the obstacles ensure that the augmented Hanan grid has polynomial size. For such a scenario, we introduce another class of auxiliary graphs with $O(n^{k-2})$ nodes and edges, parameterized by some integer $k \geq 3$, on which we solve a related Steiner tree problem (now n denotes the size of the augmented Hanan grid). This yields a $\frac{2k}{2k-1}\alpha$-approximation for any $k \geq 4$, where α denotes the performance guarantee for the ordinary Steiner tree problem in graphs. For $k = 3$, we obtain a factor of $\frac{5}{4}\alpha$.

Turning to the shortest paths problem, we present a new generic framework for Dijkstra's algorithm for finding shortest paths in digraphs with non-negative integral edge lengths. Our key concept is to label entire subgraphs instead of single vertices. In our algorithm, called GeneralizedDijkstra, we introduce three levels of hierarchy to structure the graph. The vertices of the original graph are the elements of the bottom level, and the middle level is a partition of these vertices. The top level is a partition of the middle level; its purpose is to delay certain labeling operations. Distances are propagated between elements of the middle level, but we perform direct labeling operations only between those elements that are contained in the same element of the top level, whereas all other labeling operations are delayed and thus have the potential to become unnecessary. The algorithm is suitable for graphs with a regular structure, such as partial grid graphs, where the number of involved subgraphs is small compared to the order of the original graph and the shortest path problems restricted to these subgraphs are computationally easy.

GeneralizedDijkstra is applied twice in the context of the VLSI Routing Problem, where we need to find millions of shortest paths in partial grid graphs with billions of vertices. In the first application, the original graphs correspond to the global routing corridors; each corridor is the union of a small number of rectangles. The distance labels that are output of this algorithm can be used as estimates for the remaining cost to the target in a goal-oriented path search. In a second application, we label one-dimensional intervals in the three-dimensional partial grid graph used for modeling detailed routing. This generalizes an algorithm by Hetzel [1998] from l_1-distance to arbitrary non-negative edge costs. Using the result of the first application as a pre-processing step in connection with goal-oriented techniques in the second one, we decrease the number of labels by up to 50 %. This leads to an average running-time reduction of over 16 % on leading-edge industrial chips.

Finally, we present computational results of our routing program BonnRoute, obtained on real-world VLSI chips. BonnRoute fulfills all requirements of modern VLSI routing and has been used by IBM and its customers over many years to produce more than one thousand different chips. To demonstrate the strength of BonnRoute as a state-of-the-art industrial routing tool, we show that it performs excellently on all traditional quality measures such as wire length and number of vias, but also on further criteria of equal importance in the every-day work of the designer.