# Timing-Constrained Global Routing with RC-Aware Steiner Trees and Routing Based Optimization

Dissertation

zur

Erlangung des Doktorgrades (Dr. rer. nat.)

der

Mathematisch-Naturwissenschaftlichen Fakultät

der

Rheinischen Friedrich-Wilhelms-Universität Bonn

vorgelegt von

**Rudolf Scheifele**

aus

Dresden

Bonn, März 2019

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

| | |
|---|---|
| 1. Gutachter: | Prof. Dr. Jens Vygen |
| 2. Gutachter: | Prof. Dr. Stephan Held |

| | |
|---|---|
| Tag der Promotion: | 7. Juni 2019 |
| Erscheinungsjahr: | 2019 |

# Acknowledgements

At this point I want to express my gratitude to my supervisor Professor Dr. Jens Vygen for his excellent guidance and valuable feedback. I also want to thank Professor Dr. Dr. h.c. Bernhard Korte for providing great working conditions at the Research Institute for Discrete Mathematics in Bonn.

Also, my thanks goes to my present and former colleagues for the friendly working atmosphere and productive collaboration in the past few years. In particular, I want to thank Dr. Dirk Müller for introducing me to BonnRouteGlobal and its implementation, Tilmann Bihler and Pietro Saccardi for the close collaboration on BonnRouteGlobal and for dealing with most of the matters concerning BonnRouteGlobal during the final periods of writing this thesis, and Markus Ahrens, Dr. Michael Gester and Niko Klewinghaus for their great work on BonnRouteDetailed. Furthermore, I am thankful to Siad Daboul, Professor Dr. Stephan Held, Anna Hermann, Daniel Rotter and Ulrike Schorr from the timing optimization team for their collaboration and help on timing related topics.

Further thanks goes to a number of people at IBM, especially Karsten Muuss, Dr. Sven Peyer and Dr. Christian Schulte from the routing team for the successful collaboration on BonnRoute in the last years, and Michael Kazda and Alexander Suess for the productive teamwork on the BonnRoute based optimization flow presented in this thesis.

Finally, my personal thanks goes to my parents Erika and Gustav, my sisters Anna and Edith, and my grandparents Christian and Pauline for all the support and encouragement they have given me throughout my whole life.

# Contents

CHAPTER 1

# Introduction

In today's world, computer chips are omnipresent and indispensable. They can be found in traditional computers, phones, cars and many other devices that are essential to modern living. Naturally, this need for computer chips creates a highly competitive market, and the usual objectives include making chips faster, smaller, less power consuming and less expensive to produce. As a result, modern computer chips are highly complex structures consisting of up to billions of interconnected transistors. Therefore, the VLSI design process used to create these modern computer chips is very sophisticated and requires contributions from many research areas.

In this thesis we are concerned with global routing, which is part of the VLSI routing step, whose task is to compute the layout of the wires on the chip. We present new theoretical results that are implemented and evaluated in practice. The underlying routing tool for implementing our results is BonnRoute [41], the routing tool of the BonnTools program suite [73] developed at the Research Institute for Discrete Mathematics in Bonn. BonnRoute is the main routing tool used by IBM during the design of processor chips that are among the fastest and most complex in the world and can therefore be considered as successful state-of-the-art router.

To introduce some VLSI design concepts and terminologies and to get a better understanding of the context in which global routing takes place, we outline a strongly simplified VLSI design flow in this introductory chapter. This flow is depicted in Figure 1.1 and explained in more detail in the remainder of this chapter. As it is not an essential part of this thesis, we omit any description of the manufacturing step. A more extensive introduction to VLSI design is given by Uyemura [115] and many other textbooks.

Throughout this thesis we assume that the reader is acquainted with basic concepts and terminologies in mathematics and computer science, in particular in the field of combinatorial optimization. For an introduction to combinatorial optimization, the reader is referred to the textbook by Korte and Vygen [75] or other textbooks on that topic. For examples of combinatorial optimization problems arising in VLSI design see Held et al. [52] and Korte and Vygen [74].

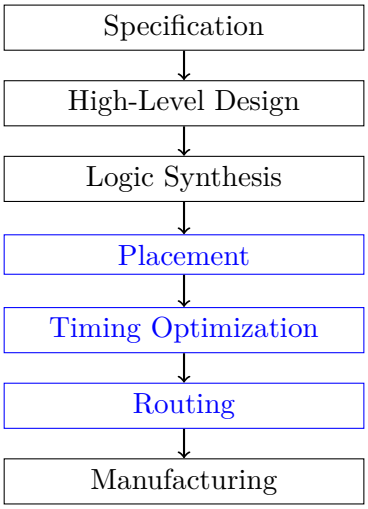



FIGURE 1.1. A strongly simplified VLSI design flow. The steps colored in blue constitute the physical design step.

In Section 1.1 we give a brief overview on logic design, in particular the output of this step that serves as input for physical design. We then turn towards physical design in Section 1.2, but only describe placement and timing optimization in this section. The last step, routing, is covered in Section 1.3, as it is the most important one for this thesis. At last, Section 1.4 contains a preview of the contents of this thesis.

## 1.1. Specification and Logic Design

The starting point of the VLSI design process is a *system specification*, where the functionality of the product and design goals are defined. These specifications are then encoded during *high-level design* in a *hardware description language* (HDL), which is a computer language similar to a programming language that is used to describe the structure of an electronic circuit. The most commonly used HDLs at this point in time are VHDL and Verilog. This abstract description of the circuit is then translated by a *logic synthesis* tool into a *netlist*, which contains the set of circuits of the chip and the logical connectivity information that controls the signal flow. As logic synthesis is not the focus of this thesis, we leave it at that very brief description and refer the reader to [**30**, **33**, **48**] for more on this topic.

Instead, we focus our attention on the output of logic synthesis for the remainder of this section, as the output of logic synthesis constitutes (part of) the input for physical design. We introduce circuits and books in Section 1.1.1, pins in Section 1.1.2 and the netlist in Section 1.1.3. These are logical concepts in the sense that they are used to describe

the logic of the chip, but as these logical concepts must be implemented physically later, they are of course also central to physical design.

**1.1.1. Circuits and Books.** A circuit consists of multiple interconnected transistors and either represents a boolean function, in which case it is called a *gate*, or is a memory element, also called a *register*. Most gates compute an elementary boolean function like $NAND$ and $NOR$, but also more complex boolean functions are possible. Memory elements are controlled by periodic clock signals and are used to store information at one point in time and release it later.
Although it would be theoretically possible to design chips on a transistor level, this is usually not done on a large scale due to the complexity involved. Instead, there is a predefined *library* containing *books*, which are blueprints for the physical implementation of specific circuits. When a circuit with a given functionality is required, a book implementing this functionality can just be taken from the library. As there are often multiple books with different physical properties available for implementing the same logic, picking the right books is a difficult optimization problem. However, this problem is not tackled during logic design, but later during timing optimization (cf. Section 1.2.2). For more on transistor-level layout and library design we refer to Cremer [31], Hougardy et al. [58] and Schneider [109].

**1.1.2. Pins.** The interface between a circuit and the chip it belongs to is given by the *pins* of the circuit. Each circuit contains a set of *input pins* and a set of *output pins*: Input signals for gates are received at the input pins, and the results of the boolean functions computed are present at the output pins. Likewise, signals to be stored in registers are present at their input pins, and signals to be released by registers are present at their output pins. Here, at any given point in time we associate each pin with one of two voltage levels $V_{ss} < V_{dd}$ representing the logical values zero and one. A signal can then be regarded as a voltage change at a given pin, as it is explained in more detail later in Section 3.1. Since the input pins of a circuit are receiving signals and the output pins are distributing signals, we classify the former as *sink pins* and the latter as *source pins*.
In addition to circuit pins, the chip also contains a set of pins which constitute the interface to the exterior context in which the chip is used. These are called *primary input* and *primary output* pins, and are classified as source and sink pins, respectively. Note that while input pins of circuits are sink pins, primary input pins are source pins, with an analogous statement holding for output pins of circuits and primary output pins. In fact, the chip itself can be considered as a circuit, and if viewed from the inside, a primary input pin has the role of distributing input signals, while a primary output pin has the role of receiving output signals.

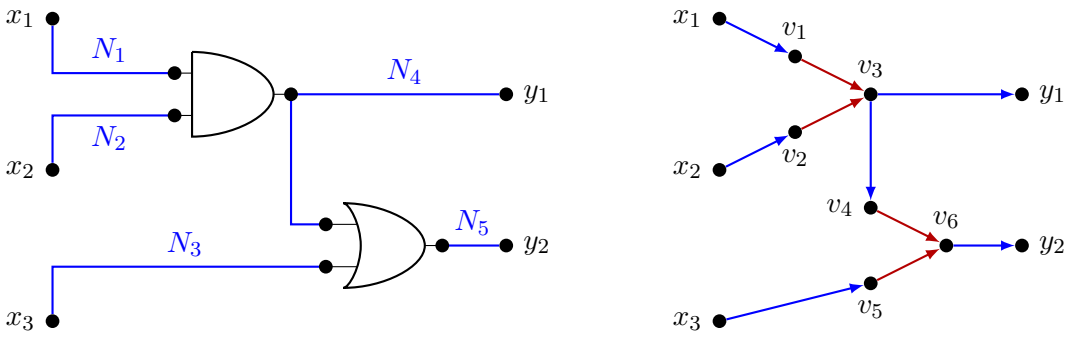



FIGURE 1.2. Left: A simple netlist with five nets $N_1, \ldots, N_5$, input signals $x_1, x_2, x_3$ and output signals $y_1, y_2$. The boolean functions computed are $y_1 = x_1 \wedge x_2$ and $y_2 = (x_1 \wedge x_2) \vee x_3$. Right: The corresponding netlist graph. We have vertices $x_1, x_2, x_3$ for the primary input pins, $y_1, y_2$ for the primary output pins and $v_1, \ldots, v_6$ representing circuit pins. Edges corresponding to nets are shown in blue while circuit internal edges are shown in red.

**1.1.3. The Netlist.** Logical connectivity information is encoded by *nets*: A net is a set of pins that contains exactly one source pin and at least one sink pin. In this context, we often call the source pin of the net — and sometimes also the circuit this source pin belongs to — the *driver* of the net. Moreover, every pin on the chip belongs to exactly one net. The set of nets is then called the *netlist*, and it is often associated with a *netlist graph*. The netlist graph is an acyclic digraph $G = (V, E)$ whose edge directions indicate signal flow: $V$ is the set of pins on the chip, and there is an edge $(v, w) \in E(G)$ if $v$ is the source pin of the net containing $w$, or if $v$ and $w$ are input and outpin pins of the same gate. A very simple example of a netlist and its corresponding netlist graph is shown in Figure 1.2.

## 1.2. Physical Design

The netlist computed by logic synthesis serves as input for the *physical design* step, where a detailed construction plan for physical components like wires and transistors is computed. In addition to the netlist, the input for the physical design step includes the rectangular chip area, the number of available chip layers and the locations of primary input and output pins. We assume the lowest layer to be the *placement layer* containing all transistors, while we assume all other layers to be *routing layers* containing wires connecting the nets. Our simplified physical design flow from Figure 1.1 consists of three steps: *placement*, *timing optimization* and *routing*. This is only a very brief and simplified description of physical design — a more comprehensive overview on physical design and the algorithms used in it is given by Alpert et al. [5].

We describe placement in Section 1.2.1 and timing optimization in Section 1.2.2. As the routing step is central to this thesis, we cover it in its own section, namely Section 1.3.

**1.2.1. Placement.** During placement, circuits are placed non-overlapping on the chip area such that objectives like timing and wire length are optimized. There are different types of placement problems: During *floorplanning*, a relatively small number of large circuits is to be placed. These can be smaller chips contained in a top level chip when hierarchical design is used, or other predesigned structures like intellectual property blocks designed by another company. On the other hand, during *standard cell placement*, all circuits classified as *standard cell* are placed. Here, a cell is a standard cell if it has some fixed uniform height. This height is defined by the distance of power supply stripes, called *power rails*, which are arranged in a regular pattern across the placement layer and also some routing layers of the chip. The standard cell height is then the maximum height that makes it possible to place the cell between two adjacent power rails. Floorplanning and standard cell placement can be done combined or separately. A third placement problem is the *placement legalization problem*, where an infeasible (i.e. not overlap-free) placement is given and the task is to compute a feasible placement that is close to the input placement. The placement legalization problem occurs continually during timing optimization and therefore also during our routing based optimization flow from Chapter 7.
All circuits (including large units placed during floorplanning) are placed on the placement layer, but they might also occupy space on routing layers. This occupied space is seen as blockage by the router. Large units might occupy a lot of space even on higher layers, while standard cells only block a small amount of space on the lowest layers. Throughout this thesis we assume that a feasible placement is given in the input for routing and do not examine this problem any further. For a comprehensive overview on the placement problem the reader is referred to Alpert et al. [5], Brenner et al. [20] and Struzyna [114].

**1.2.2. Timing Optimization.** After placement has finished, timing optimization takes place. The main task in timing optimization is to ensure that all signals arrive in the correct time window so that the chip can operate at the desired frequencies — if the result of logic synthesis and placement was to be routed directly, meeting timing constraints would be impossible for all but the simplest designs. There are several operations which can be performed during timing optimization, including (but not limited to) the following:

- *Buffering:* As will become evident after looking at the Elmore delay model [35, 99] from Section 3.4, long nets and nets with many pins must be subdivided into

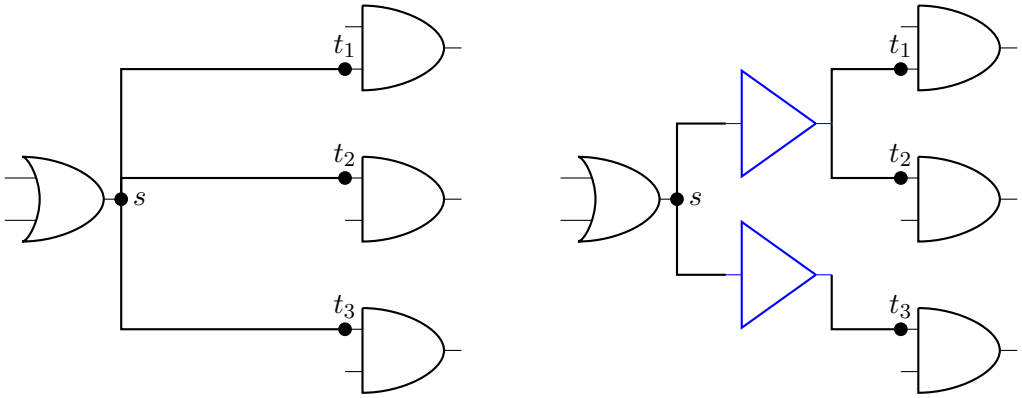



FIGURE 1.3. The net $\{s, t_1, t_2, t_3\}$ (left) is subdivided into three nets by insertion of two buffers (right).

smaller nets in order to avoid excessive signal delays. This is done by inserting *buffers* or a suitable number of *inverters*, which implement the identity and *NOT* function, respectively. For simplicity, we only use the term buffer to cover buffers and inverters for the rest of this thesis.

The buffering problem involves two major subproblems: The first one, *buffer insertion*, consists of inserting buffers into the netlist, usually based on the structure of some routing tree given for a certain net. The second one, *buffer tree generation*, consists of computing the routing tree that is used for buffer insertion. While buffer insertion can be performed on an estimated routing tree, it is more accurate to combine global routing and buffering, as was done by Rotter [97]. Further references regarding buffering include Bartoschek [8] and Hu et al. [60]. A simple illustration of the buffering process is given by Figure 1.3.

- *Gate sizing:* The library (cf. Section 1.1.1) usually contains multiple books implementing the same boolean function, but having largely different electrical properties. The electrical properties roughly correlate to the size of the book: Larger books have a lower internal resistance and switch faster on their own, but consume more area and power and have bigger input pins that slow down predecessor nets. The task in gate sizing is to pick a suitable book for each gate (or circuit) in the netlist, as the netlist output by logic synthesis is logically correct, but not well optimized with respect to gate sizes. An overview on the gate sizing problem is given by Held and Hu [51] and Schorr [110].
- *Vt optimization:* The voltage threshold, or Vt level, denotes the voltage at which a transistor switches. A lower Vt level results in faster switching and therefore faster signal propagation, but it also increases power consumption.

There are usually a few different Vt levels available for each book, and the task in Vt optimization is to pick one for each circuit to improve timing without increasing power consumption too much. For more on Vt optimization see Daboul et al. [32] and Schorr [110].

- *Local placement changes:* Although timing is considered as major objective during placement, one can often improve timing further by deploying additional local placement changes later in the flow. Here, "local" can refer to moving only local parts of the netlist, e.g. a single circuit, but can also entail restricting changes to a local area. A method for improving timing with local placement changes is given by Bock et al. [14].
- *Local logic restructuring:* The netlist output by logic synthesis is logically correct, but may not be optimal with respect to timing behavior. This is particularly true as the timing of the chip is strongly influenced by placement and timing optimization, and both these steps are running after logic synthesis in the flow. Therefore, logic restructuring can be applied to improve timing. As the problem of deciding whether two netlists are logically equivalent contains the 3-SAT problem and is therefore $NP$-hard, very complex changes are usually avoided. For more on local logic restructuring see Held and Spirkl [55] and Werber [121].

The above steps are interrelated in the sense that to give meaningful input to one step, the other steps must already have been performed to a reasonable degree. For this reason, better results can be obtained if different timing optimization steps are combined and iterated during the course of the physical design flow. Held [50] covers several aspects of timing optimization in more detail.

As already indicated by the above description of the individual timing optimization steps, there is often a trade-off between improving timing and keeping power consumption in check. Routability also has to be considered during timing optimization. This is usually done by using various kinds of routability estimates ranging from rather simple estimates incorporated in the timing optimization tools to more accurate estimates given by a global router. An accurate method to estimate routability and timing properties of wires is given in Chapter 7, where we deal with routing based optimization, i.e. a method to perform timing optimization based on a routing that is incrementally updated by our global router.

## 1.3. Routing

The final step in our simplified physical design flow is routing. We start with routing basics in Section 1.3.1, continue with global routing — the main focus of this thesis — in

Section 1.3.2 and then introduce track assignment and detailed routing in Sections 1.3.3 and 1.3.4, respectively. Our description of the various routing steps is tailored to BonnRoute. Most of it should be generally applicable, but for other routers, individual details may vary. An improvement of the routing step that is not discussed in this introductory chapter is routing based (timing) optimization. We discuss this in detail in Chapter 7.

**1.3.1. Routing Basics.** The basic task in routing is to connect the pins of each net by metal wires, i.e. realizing the logical connectivity of the netlist by a physical implementation. These metal wires are always extending in $x$- or $y$-direction — they are never running diagonal. In principle, diagonal wiring could be used to obtain better routing solutions, but this would complicate manufacturing and physical design considerably.
Wires are arranged in *wiring layers*, which are located above the placement layer. Different wiring layers are separated from each other by insulating material, but metal holes called *vias* can be cut into the insulating material to establish electrical connectivity between different wiring layers. For this reason, the layers between wiring layers are called *via layers*.
A central constraint for routing is that wires and vias of different nets have to obey certain *minimum distance rules*: If wires or vias of different nets are placed too close to each other, then this can very likely result in a short after manufacturing, rendering the chip useless. To allow for a better packing of wires, one uses the concept of *preference directions*: Each wiring layer has a preference direction that is either horizontal or vertical, and the large majority of the wiring on that layer will extend into that direction. Depending on the layer and the technology, wires running orthogonal to the preference direction of the layer (called *jogs*) may or may not be allowed. However, they should be used sparingly, as they can be a barrier for many other wires running in preference direction. For the sake of a simpler description we assume for the rest of this section that jogs are not allowed. Naturally, adjacent routing layers have different preference directions in current technologies.
It is natural (but not mandatory) to use a graph model for solving the routing problem. The underlying graph is called *track graph*, as it is based upon *routing tracks*: On each layer, tracks extending into the preference direction of the layer are spanned over the chip area. At each point where $x$- and $y$-coordinates of two different tracks on adjacent layers coincide, vertices are added to the track graph on both layers, and both these vertices are connected by an edge representing a possible via position. Neighboring vertices on the same track and same layer are also connected by an edge. If some edges are not usable for the router (e.g. due to routing blockages), then these edges can be omitted when constructing the track graph. An illustration of the track graph is given by Figure 1.4.

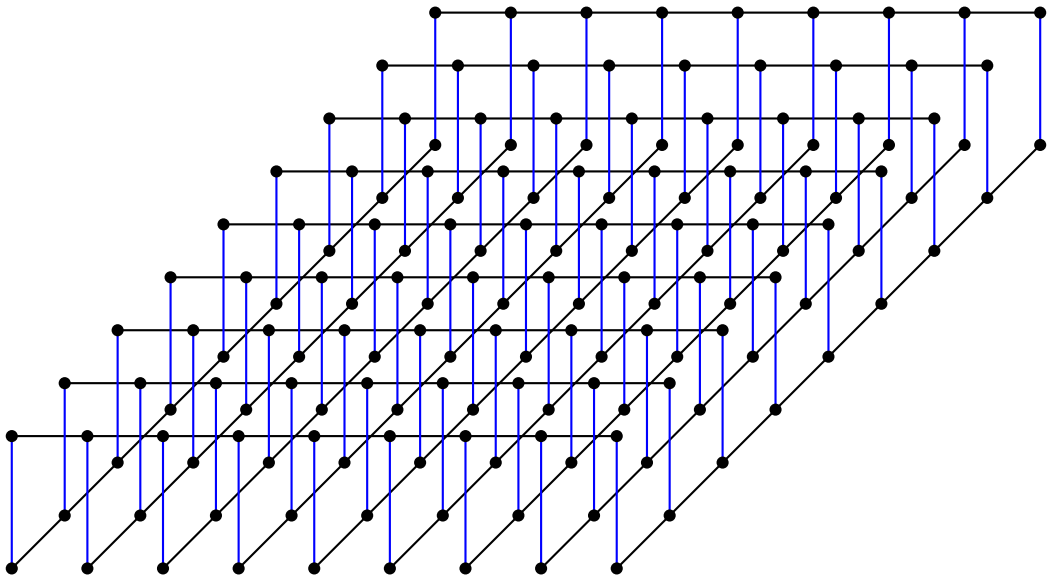


FIGURE 1.4. Illustration of a track graph with two routing layers. Blue edges correspond to vias, while black edges correspond to wires.

The distance between neighboring tracks, called *track pitch*, is naturally chosen as the minimum distance such that two minimum width wires can be placed on neighboring tracks without causing a minimum distance rule violation. Here, the track pitch is not the same on every layer. It is generally larger on higher layers in order to accomodate thicker wires with better electrical properties. Modeling minimum distance rules through the construction of the track graph is more or less simple as long as only minimum width wires with minimum spacing requirements are used, but the router must also be able to deal with wires of different *wire types*, which define the exact geometries and spacing requirements of wires and vias on different layers.

Using the track graph, a simplified version of the routing problem can be formulated as a Steiner tree packing problem in graphs: Each pin corresponds to a vertex (or a set of vertices) in the track graph, and the wiring of a net is represented by a Steiner tree connecting the pins. Steiner trees of different nets must be vertex-disjoint in order to avoid violations of minimum distance rules. From a theoretical perspective, vertex-disjoint packing of Steiner trees in grid graphs is $NP$-complete [76, 77]. In practice, the routing problem is complicated further by the following factors:

- **Complex minimum distance rules:** The construction of the track graph does not ensue complete adherence to minimum distance rules, e.g. when different wire types are used. There are also specific spacing rules for vias. This makes compliance to minimum distance rules more difficult.

- **Complex same-net rules:** In addition to minimum distance rules, which are concerned with shapes of different nets or blockage shapes, there are also rules prescribed for the routing of a single net, called *same-net rules*. An example for such a rule is the minimum area rule, which dictates that any connected component of wiring on a given wiring layer must cover a certain minimum area. Some same-net rules can be hard or runtime-intensive to incorporate into an automatic routing tool. For instance, the above mentioned minimum area rule will in general not be followed when a standard path search algorithm (e.g. Dijkstra's algorithm [34]) is run on the track graph to connect two points. As outlined by Gester [40] (Section 5.4.2), many same-net rules can be incorporated into a path search algorithm by propagating multiple labels, but this usually results in a significant running time increase.
- **The graph is huge:** As the track pitch is very small compared to the size of the chip, the track graph can become huge. This results in track graphs with billions of vertices where millions of Steiner trees have to be packed. Therefore, running time is always a critical factor and naive approaches are most often not feasible.

The set of rules, called *design rules*, is primarily given by the foundry with the goal of preventing manufacturing problems. Usually, the number of design rule violations should be zero at the end of the physical design phase, but in some cases a few non-fatal violations might be tolerated, although this can have a negative impact on the manufacturing yield. For more on design rules in VLSI routing see Schulte [111].
On instances occuring in practice it is usually hard to find any routing solution that closes all connections while satisfying the given design rules. However, finding any routing solution is not good enough for the design of high-performance chips. In particular timing, power consumption and manufacturing yield are strongly influenced by the routing result and should be optimized accordingly. Incorporating these objectives makes the routing problem even harder. In this thesis we mainly deal with the optimization of timing during routing.
Due to the complexity involved, the routing task is usually split into two or even three major steps. The first of these steps is global routing, and it is the main focus of this thesis. The second step, track assignment, is optional, and the third step is detailed routing. We describe these three routing steps in the following sections.

**1.3.2. Global Routing.** Due to the inherent complexities of the routing problem outlined in Section 1.3.1, trying to solve it by packing Steiner trees directly in the track graph without any preparations is most likely futile. To this end, *global routing* is performed as the first step in the routing process: During global routing, a rough

layout called *global route* or *global wires* is computed for the routing of every net, which is then used as a guidance for the later routing steps to compute the actual detailed routing. To make this process feasible, design rules are only modeled indirectly and in a strongly simplified manner during global routing. Moreover, global routing takes place on a coarsened and much smaller version of the track graph, called the *global routing graph*. This central data structure is introduced in Section 1.3.2.1. In Section 1.3.2.2 we touch the subject of estimating global routing capacities and discuss global routing objectives in Section 1.3.2.3. Finally, we deal with the consideration of timing during global routing in Section 1.3.2.4. In this chapter we describe global routing in a rather descriptive and informal way. Formal definitions are given later in Chapter 2.
Global routing has many different applications during physical design — for example, it can be used to evaluate routability during placement and timing optimization, and it can be interwoven with the buffering step. In this thesis, we are only concerned with global routing as a preparation for track assignment and detailed routing, although part of the results from this thesis can certainly also be applied in other contexts.

1.3.2.1. *The Global Routing Graph.* The global routing graph is constructed as depicted in Figure 1.5: First, the chip area $[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$ is subdivided into a grid of rectangular *global routing tiles* through sequences $x_{\min} = x_0 < x_1 < \ldots < x_n = x_{\max}$ and $y_{\min} = y_0 < y_1 < \ldots < y_m = y_{\max}$ of *x-cuts* and *y-cuts*, which define the boundaries of the global routing tiles. This divides the chip area into $mn$ rectangular regions, and all vertices of the track graph that are located in the same region and on the same layer are contracted into a single vertex. Via edges are then added between vertices corresponding to the same region on adjacent layers, and wiring edges extending into the preference direction of the layer are added between vertices corresponding to neighboring regions on the same layer. Wiring edges orthogonal to the preference direction of the layer are not added, as jogs should be short and therefore negligible during global routing. A formal definition of the global routing graph is given later by Definition 2.8.
The size of the global routing graph is determined by the size of the track graph and the sets of $x$- and $y$-cuts. Therefore, there is a trade-off between running time and precision: Fewer cuts result in a smaller graph and therefore presumably less running time, but may also result in a loss of precision. The cuts can be chosen to be equidistant, and the distance may be increased on large designs to reduce the size of the global routing graph. However, it can be beneficial to choose non-equidistant cuts, e.g. in order to align global routing tiles with large blockages, as these can cause inaccuracies in the global routing model, in particular when running through a tile. Non-equidistant cuts are used in BonnRoute [42], and they are implemented in such a way that the distance

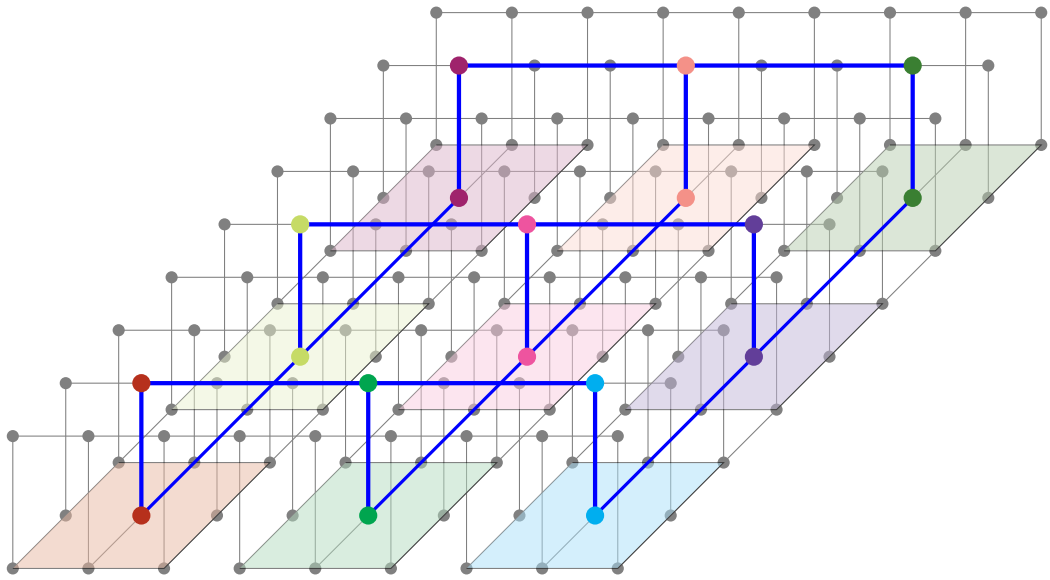


FIGURE 1.5. Construction of the global routing graph from the track graph from Figure 1.4: Vertices of the track graph (gray) that are located in the same global routing tile (colored rectangles) and on the same layer are contracted into a single vertex in the global routing graph (colored as its tile area). The vertices of the global routing graph are then connected by via edges and wiring edges in preference direction of the given layer (blue).

between neighboring cuts does not deviate too much from some predefined value. In our experiments, we set this value to 70 times the minimum track pitch across all routing layers. One can see that in such a setting the global routing graph is significantly smaller than the track graph, but on large designs it can still have millions of vertices. For instance, the global routing graph on our largest design from Figure 1.6 contains around two million vertices, and we have to route around 1.6 million nets in it. Therefore, running time is a critical factor for any global routing algorithm.

1.3.2.2. *Capacity and Routing Space Usage Estimations.* In contrast to the track graph, an edge of the global routing graph does not represent part of a single routing track or a single possible via position. Instead, we associate with each edge of the global routing graph an amount of free routing space, which is called *capacity* of the edge. If we regard an edge of the global routing graph as a cut in the track graph, we can just define the capacity of the edge as the size of the cut. If all wires were very long, this would possibly be an acceptable model. However, in practice, this is usually not the case: Vias might connect non-adjacent layers and require only small pieces of metal on the intermediate wiring layers; pin access, local congestion and design rules might require

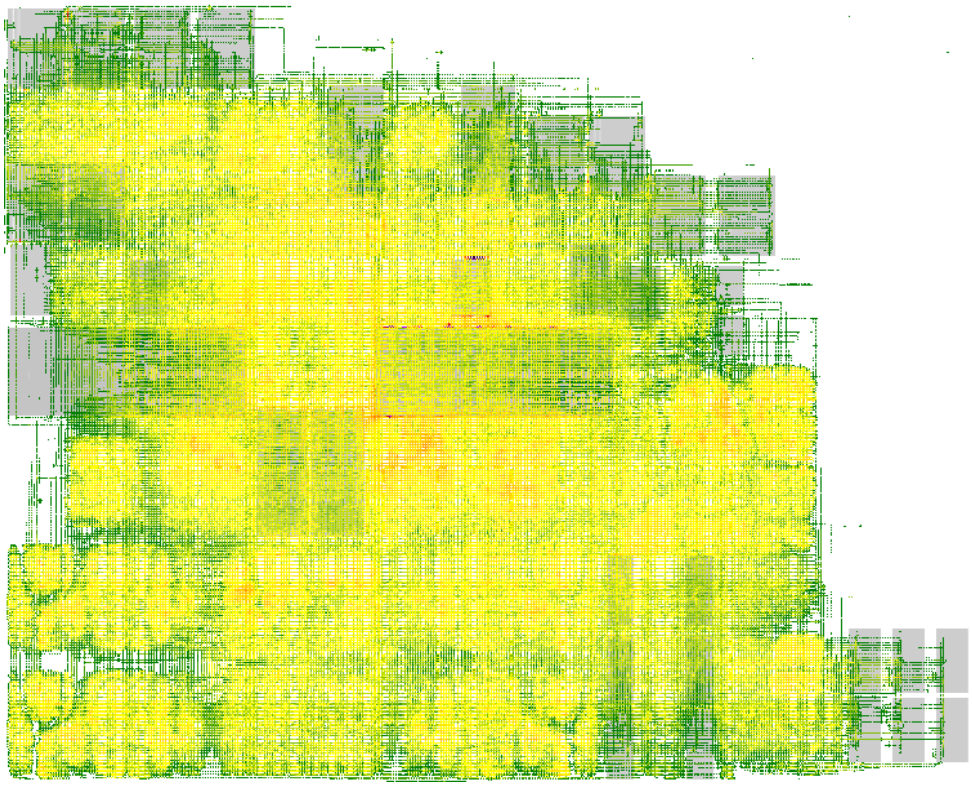


FIGURE 1.6. Congestion map of our largest unit U8. Every edge of this two-dimensional projection of the global routing graph is colored according to the maximum edge usage over all layers. The widespread yellow color indicates that congestion is a few percent below our congestion target of 90% for the most part, but some orange spots at 90% congestion exist. Around the borders of some large blockages there are a few dark-colored edges, which indicates routing overflow. Green and white regions are uncritical with respect to routing congestion.

the detailed router to make local detours; blockages and input wiring might locally block tracks; and in fact, on most designs many nets are so short that all their pins are in the same or neighboring global routing tiles.

All these things have to be considered during capacity estimation, making it a rather intricate problem in practice. Moreover, as the capacity estimation has to be tailored to the design rules and the detailed router that is used later, a fair amount of experimental studies and fine-tuning is needed to get good results.

Tightly connected to capacity estimation is the question of how to compute the routing space usage of a given wiring. In the standard model — the one that we are using

throughout this thesis — the routing space usage of a wire or via only depends on its dimensions and wire type, but not on other wires of the same or other nets.
In this thesis, we will not be particularly concerned with capacity estimation or routing space usage computations. Instead, we assume routing capacities and routing space usage functions as given. We refer the reader to Wei et al. [120] for an overview on the modeling of local congestion. They use a method where local congestion effects are estimated based on pin positions and estimated Steiner trees. On the other hand, an approach where local effects are estimated based on the local structure of the actual global routes is currently being implemented in BonnRouteGlobal, relying on the framework that we present in Chapter 6. At the time of this writing, this approach is in development for BonnRouteGlobal, but not yet usable for this thesis. An approach to compute global routing capacities based on maximum flows is given by Müller [84]. A comprehensive discussion of routing congestion is given by the book of Saxena et al. [105].
Any edge usage that exceeds its capacity is considered as *routing overflow* and should be minimized. In the simplest notion, a global routing solution is considered routable if there is no routing overflow. However, due to inaccuracies in the global routing model and capacity estimation and the inherent complexity in the subsequent detailed routing step, the question whether a given global routing will be sufficiently well routable by the detailed router is often not easy to answer in practice. Wei et al. [120] study the problem of routability prediction and establish global routing metrics that can be evaluated in order to predict the routability of a given global routing. They define the ACE metric, which basically represents the average congestion of the most congested edges (cf. Appendix A.4). Throughout this thesis, we use this metric in addition to the overflow metric to evaluate routability. Figure 1.6 shows a congestion map with a color scheme that represents edge usages.

1.3.2.3. *Global Routing Objectives.* The main objective during global routing is to output a global routing that represents routable input for track assignment and detailed routing. In that sense, producing a routable global routing, e.g. indicated by having zero routing overflow, can be regarded as a constraint.[1] When that need is satisfied, one usually turns towards the optimization of objectives such as timing, power consumption and manufacturing yield.
The traditional objective here is to minimize total wire length and possibly via count, as for example indicated by the evaluation metrics of the ISPD 2007 and 2008 global routing contests [66, 67]. Minimizing total wire length can be understood as an indirect attempt to optimize other objectives such as timing and power consumption. It is also

[1]In some applications for global routing, this might not be true. For the global routing preceding detailed routing, however, this is a reasonable conception.

pragmatical in the sense that minimization of wire length can be achieved by routing nets as minimum (rectilinear) Steiner trees, and the construction of approximately minimum Steiner trees is a classical optimization problem that has been studied long-since both from a theoretical and practical point of view (cf. Section 2.3).
The idea of modeling power minimization via wire length minimization during global routing stems from the fact that power consumption roughly correlates with total wire capacitance (assuming a fixed netlist), and this in turn correlates with total wire length. Here, improvements can be made by making use of the fact that wire capacitances differ (moderately) across the layer stack and by improving wire spreading. An approach to optimize power consumption during global routing is given by Vygen [**117**] and Müller, Radke and Vygen [**87**]. The important objective of satisfying timing constraints during global routing is covered in the next section.

1.3.2.4. *Global Routing and Timing.* When it comes to timing, then minimizing wire length correlates well with minimizing circuit delays, as these scale well with the capacitances of the nets that are driven by the output pins of the circuits. However, as the technology is advancing, wire delays are becoming increasingly important due to rapidly growing wire resistances [**28**]. Wire delay minimization, though, is not necessarily a byproduct of wire length minimization, as we see when discussing the choices of wire types and routing layers in Section 1.3.2.4.1 and routing topologies in Section 1.3.2.4.2. Moreover, balancing timing criticalities plays an important role in achieving good timing results, and, as Section 1.3.2.4.3 explains, it is not always covered well in traditional global routing frameworks. In this section we only provide a very short preview on the topic of considering timing during global routing. More details are then given in later chapters of this thesis that deal extensively with this topic.

1.3.2.4.1. *Wire Type and Layer Assignments.* In Section 1.3.1 we introduced wire types, which define the layer-dependent dimensions and spacing requirements of wires. The main reason to use width and spacing values that are larger than the minimum width and spacing imposed by manufacturing constraints is timing: Thicker wires have less resistance and more spacing means less (coupling) capacitance. This way, using wire types with larger metal widths or spacings can significantly improve signal delays, as will become evident when looking at the Elmore delay model from Section 3.4. To this end, the input to routing usually contains a *wire type assignment*, which assigns a wire type to every net.
For the same reason as a wire type assignment a so-called *layer assignment* is done: On modern metal stacks, the common wire widths, heights and spacings increase from lower to higher layers [**119**], which results in significant decreases of wire resistances on higher layers. Therefore, wire delay decreases when a net is routed on higher layers, unless the

wire length is too small and via delays outweigh the better electrical properties of the higher layers.

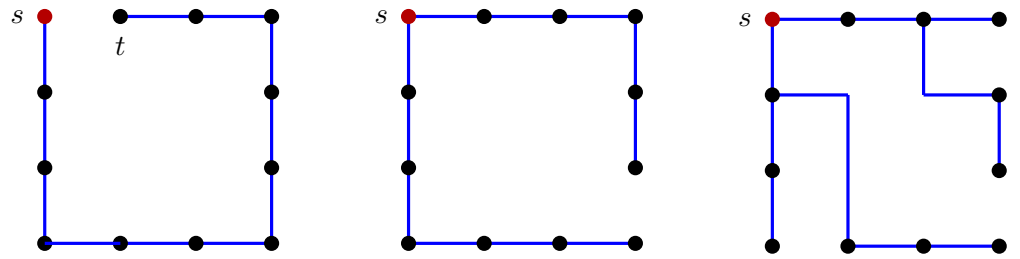
The task in layer assignment is to assign a contiguous layer range to each net with the intent that the router will predominantly use that layer range for routing the net, only deviating from it when it has to, e.g. for accessing pins. As wires on higher layers usually have better electrical properties, it is common to only specify a minimum allowed layer and set the maximum allowed layer to the maximum available layer of the chip.

Layer assignments are traditionally done by a timing optimization tool [**6**, **63**, **119**]. However, there are several difficulties and disadvantages inherent to this approach: Firstly, any layer assignment has to be congestion-aware, as assigning too many wires to the upper layers easily leads to overcongestion. This is usually achieved by having some sort of congestion estimation during the layer assignment step, but this congestion estimation can never be as accurate as the congestion information that the global router sees during the actual routing process. Secondly, the layer assignment step can usually only assign layers to a whole net, while it might actually be sufficient to make an assignment only for parts of the net (e.g. only wires close to the source, or only the connection to the critical sink). This can lead to a waste of routing resources. Thirdly, the layer assignment usually uses delay characteristics of estimated routing trees, and the actual tree computed by the router might diverge from the estimate.

We deal with this matter in Chapters 5 and 6: Our net router from Chapter 5 can inherently optimize Elmore delays and make good layer choices on its own. In Chapter 6 we also present a layer assignment algorithm that can assign individual wires of a fixed two-dimensional routing tree to individual layers, minimizing routing congestion and Elmore delays. As these approaches are directly integrated into our global routing algorithm, they do not suffer the disadvantanges of an external layer assignment step outlined above.

The above considerations also hold true for an externally created wire type assignment — it faces the same difficulties as an external layer assignment step. In this thesis, we are able to achieve better results by ignoring the input layer assignment during our net routing process, but we still adhere to the wire type assignment in the input. Generalizing the results from Chapters 5 and 6 to also include wire type assignments should be uncomplicated from a theoretical perspective. Working out a well-performing practical implementation is a possible task for the future.

1.3.2.4.2. *Routing Topologies*. The routing topology can have a large impact on wire delays. Here, an intuitive insight is that wire delays depend on the source-sink path length, which is not necessarily minimized by minimum Steiner trees. However, as we will see when introducing the Elmore delay model in Section 3.4, source-sink path




(a): A minimum Steiner tree with presumably bad delay properties due to long source-sink paths. The wire delay to $t$ might be very large.

(b): Another minimum Steiner tree where all source-sink paths are shortest paths. Wire delays are better than in (a), but may still not be optimal.

(c): Depending on the instance parameters, this might be a better tree than (b) with respect to the very popular Elmore delay model. The tree is longer, but wire delays may be smaller due to less capacitance in some of the subtrees.

FIGURE 1.7. Three Steiner trees for the same net with very different delay properties. The source pin $s$ is depicted by the red circle, while black circles depict sink pins.

length is not the only factor influencing wire delays. An illustration of this is given by Figure 1.7. More details on this topic and an algorithm to compute routing trees approximately minimizing Elmore delays are given in Chapter 5.

1.3.2.4.3. *Timing Criticalities.* Another reason why the approach of minimizing total wire length is not ideal for optimizing timing is that it does not distinguish between timing-critical and timing-uncritical nets. This can result in detours on timing-critical nets in favor of timing-uncritical nets. One attempt to alleviate this problem is to minimize the weighted sum of wire lengths over all nets, where more critical nets get higher weights. Another approach is to put constraints on the wire length of timing-critical nets, i.e. constraints that require the wire length to be within a certain factor of the length of an approximately minimum Steiner tree for the given net. Within the framework of wire length minimization, these measures are certainly useful and required for attaining good timing results. However, they still address the problem of optimizing timing during routing rather indirectly, and it is rather difficult to come up with a good set of such externally imposed constraints in advance. In this thesis, we present a better approach in Chapter 4, where we map out a global routing framework that can inherently model timing constraints and balance out timing criticalities during the course of global routing.

**1.3.3. Track Assignment.** Track assignment is an optional intermediate step between global and detailed routing, where the task is to assign long global wires to routing tracks. While the primary objective in track assignment is to assign as much wiring as possible, secondary objectives include optimization of wire length, timing and power consumption.
The wiring assigned by track assignment should obey most or all minimum distance rules, but it will probably not satisfy all same-net rules. However, if most of the long wires have been successfully assigned, the detailed router should subsequently be able to work mostly locally. This promises reduced detailed routing running times and an improved quality of results, as the track assignment step has a more global outlook on the routing problem than the detailed routing step.
In BonnRoute, the track assignment step is currently in development [88] and not included in the default routing flow. We therefore run all our experiments without track assignment. More details on the track assignment problem and heuristics to solve it are given by Batterywala et al. [9] and Chang and Cong [24].

**1.3.4. Detailed Routing.** The detailed routing step has to deal with most of the complexity of the routing step outlined in Section 1.3.1, as in this step the *detailed wires* that are later used for manufacturing are computed. However, this is done with the help of global routing and possibly track assignment: The usual approach here is to route one net after another and restrict the routing area for every net to an area encompassing the global routing for the net. This way only a small part of the track graph has to be considered when routing any given net, making the effective instance size for routing a single net manageable. At this point it becomes critical that the global router did not overuse routing capacities, and that the capacity estimation used by the global router was sound. Otherwise, detailed routing is likely to fail.
If the input from global routing is good, a well implemented ripup-and-reroute heuristic, i.e. an algorithm that routes nets sequentially with the possibility to rip out parts of previously routed nets (see e.g. Salowe [101]), is likely sufficient to produce a feasible detailed routing. If track assignment has successfully been run before detailed routing, then most long wires might already be in place, and the job of the detailed router might reduce to resolving local conflicts, accessing pins and fixing design rule violations.
Ripup-and-reroute is also implemented in BonnRouteDetailed, which is the detailed router that we use for our experiments throughout this thesis. BonnRouteDetailed uses an efficient implementation of Dijkstra's algorithm [34] as its path search algorithm and contains other techniques that improve running time and quality of results. In particular, BonnRouteDetailed does not only restrict the routing area for a net to a small area around the global wires, but also sticks to the topology of the global routing with only

local deviations [72]. This is particularly important in the context of timing-aware global routing, as global topology changes by a timing-unaware (or less timing-aware) detailed routing algorithm can negatively impact timing. For more on BonnRouteDetailed see Ahrens et al. [1], Gester et al. [41], Gester [40] and Schulte [111].

## 1.4. Thesis Overview

After presenting this introductory material, we use this section to give a short overview on the contents of this thesis. In Chapter 2, we start with basic concepts and definitions in the context of global routing that are used throughout this thesis. It also presents the traditional way of formulating the global routing problem, which uses wire length minimization as objective. Chapter 3 then covers basic concepts of VLSI timing in a level of detail that is sufficient for the subsequent chapters. In particular, it introduces the Elmore delay model, which is the delay model that is used throughout this thesis. Chapter 4 describes our timing-aware global routing framework, i.e. a global routing framework that improves upon the traditional global routing framework from Chapter 2 by inherently modeling timing constraints. As a subroutine, this framework requires a timing-aware net router, i.e. in our case a net router that is able to minimize a weighted sum of congestion prices and Elmore delays. This net router is presented in Chapter 5. A refinement of the global routing model is then presented in Chapter 6: It outlines a process to convert a global route that is basically a subgraph of the global routing graph to a global route that is connecting exact pin and prewire shapes. It can be used for a more accurate modeling of e.g. timing and routing space usages during global routing. Moreover, Chapter 6 contains a layer assignment algorithm for minimizing a weighted sum of congestion prices and Elmore delays and is therefore an extension of the results from Chapter 5 in that regard. Finally, Chapter 7 presents the aforementioned routing based optimization step, which allows timing optimization to be performed on a fixed global or detailed routing that is incrementally updated by our global router in response to any changes introduced during timing optimization.

CHAPTER 2

# Global Routing Basics

In this chapter we describe basic global routing concepts. The contents of this chapter are closely related to the introductory material from Chapter 1, in particular Section 1.3.2, which introduces global routing and its purpose in VLSI design. In contrast to Chapter 1, which presents its contents in a rather informal way, we use this chapter to establish formal definitions and concepts that are then used in later chapters. We start with a collection of basic concepts and definitions in Section 2.1, introduce the Traditional Global Routing Problem in Section 2.2, and conclude this chapter with a discussion of the Minimum Steiner Tree Problem in Section 2.3.

## 2.1. Basic Concepts and Definitions

This section provides a collection of basic concepts and definitions that are used throughout this thesis. It is primarily intended as a reference point when reading later chapters. We present a number of definitions regarding graphs. As a convention for this thesis, we usually give these definitions only for directed or undirected graphs. If the definition is trivially transferable to the other group of graphs, then it is also supposed to hold for them (unless explicitly stated otherwise). Moreover, we sometimes embed directed graphs into undirected graphs, or consider subgraphs of undirected graphs as directed based on some orientation of the edges. In the same manner, we sometimes laxly treat undirected graphs as if they were directed (also notation-wise), but the meaning should always be clear in the given context. We start our series of definitions with one regarding coordinates:

Definition 2.1. For $a \in \mathbb{R}^3$ and $d \in \{x, y, z\}$ let $a_d$ be the coordinate of $a$ in dimension $d$.

We shortly introduce our notation to refer to neighbors of vertices in a graph:

Definition 2.2. Let $G$ be a directed graph and $v \in V(G)$. Then we define $\Gamma_G(v) := \Gamma^+_G(v) \cup \Gamma^-_G(v)$ to be the *neighbors of* $v$, where $\Gamma^+_G(v) := \{w \in V(G) : (v, w) \in E(G)\}$ and $\Gamma^-_G(v) := \{u \in V(G) : (u, v) \in E(G)\}$.

In general, we may leave out the graph subscript if it is clear from the context which graph is meant. The next definition deals with our notion of graphs and their embedding into the rectilinear space:

DEFINITION 2.3. Let $S \subseteq \mathbb{R}^3$. A *graph $G$ embedded rectilinearly into $S$* is a graph $G$ associated with vertex positions $p\colon V(G) \to S$ such that for all $(v,w) \in E(G)$ we have $|\{d \in \{x,y,z\} : p(v)_d \neq p(w)_d\}| \leq 1$. Given a graph $G$, a *graph $H$ embedded into $G$* is a graph $H$ associated with vertex positions $p\colon V(H) \to V(G)$ such that $(v,w) \in E(H) \Rightarrow (p(v),p(w)) \in E(G) \vee p(v) = p(w)$. In that case, we define an extension $p\colon E(H) \to E(G) \cup \{(v,v) : v \in V(G)\}$ by setting $p(v,w) := (p(v),p(w))$ for $(v,w) \in E(H)$. A graph structure that is not associated with vertex positions is called a *topology*.

Allowing adjacent vertices of $H$ to be mapped to the same vertex in $G$ is sometimes required to enforce a special structure for $H$. In the sense of Definition 2.3, a graph will always implicitly be associated with vertex positions $p$ if this makes sense in the given context. When a graph $H$ is embedded into a graph $G$, functions on $V(G)$ and $E(G)$ are extended naturally to $H$:

DEFINITION 2.4. Let $H$ be a graph that is embedded into another graph $G$, and let $f\colon V(G) \to X$ and $g\colon E(G) \to Y$ for some sets $X, Y$ be arbitrary functions. If not explicitly stated otherwise, then we implicitly extend $f$ and $g$ to $H$ by setting $f(v) := f(p(v))$ for $v \in V(H)$, and $g(v,w) := g(p(v),p(w))$ for $(v,w) \in E(H)$ with $p(v) \neq p(w)$ and $g(v,w) := 0$ otherwise (assuming $Y$ contains a zero element).

Usually, graphs will be embedded into the layered chip area:

DEFINITION 2.5. The *chip area* is given by a non-empty rectangle $A := (x_{\min}, x_{\max}] \times (y_{\min}, y_{\max}] \subseteq \mathbb{R}^2$. Given a non-empty set of layers $Z := \{1, \dots, n_z\}$ for some $n_z \in \mathbb{N}$, the *layered chip area* is defined by $S := A \times Z$.

The chip area in Definition 2.5 is defined half-open as a matter of convenience for subsequent definitions. We categorize the edges of a graph that is embedded rectilinearly into the chip area into two different classes:

DEFINITION 2.6. Let $G$ be a graph that is embedded rectilinearly into the layered chip area. Then we partition $E(G) := E_{\text{wire}}(G) \cup E_{\text{via}}(G)$, where $E_{\text{wire}}(G) := \{(v,w) \in E(G) : p(v)_z = p(w)_z\}$ is the set of *wiring edges* and $E_{\text{via}}(G) := E(G) \setminus E_{\text{wire}}(G)$ is the set of *via edges* of $G$. If $H$ is a graph that is embedded into $G$, then we set $E_{\text{wire}}(H) := \{(v,w) \in E(H) : p(v,w) \in E_{\text{wire}}(G) \vee p(v) = p(w)\}$ and $E_{\text{via}}(H) := \{e \in E(H) : p(e) \in E_{\text{via}}(G)\}$.

We continue with pins and nets:

DEFINITION 2.7. Let $S$ be the layered chip area. The set of *pins* on the chip is denoted by $\Pi$ and characterized by a mapping $p_{\text{ex}}\colon \Pi \to S$, where $p_{\text{ex}}(\pi)$ denotes the *exact shape* of $\pi \in \Pi$. The *netlist* $\mathcal{N}$ consists of a set of *nets* such that $\sqcup_{N\in\mathcal{N}} N = \Pi$. Given a pin $\pi \in \Pi$ we let $N(\pi)$ denote the net containing $\pi$.

The netlist can be organized into a netlist graph as outlined in Section 1.1.3. In practice, pin shapes are not points, but rectilinear polygons. However, as they are usually small, modeling them as points as in Definition 2.7 is a reasonable simplification for our purposes. We now formally define the global routing graph, which has already been introduced in Section 1.3.2.1:

DEFINITION 2.8. Let $S = (x_{\min}, x_{\max}] \times (y_{\min}, y_{\max}] \times \{1, \dots, n_z\}$ be the layered chip area. Given sequences $x_{\min} = x_0 < x_1 < \dots < x_{n_x} = x_{\max}$ and $y_{\min} = y_0 < y_1 < \dots < y_{n_y} = y_{\max}$ of real numbers representing *$x$- and $y$-cuts* for some $n_x, n_y \in \mathbb{N}$, the *global routing tile* with *tile coordinate* $(i,j) \in \{1,\dots,n_x\} \times \{1,\dots,n_y\}$ is defined as $A(i,j) := (x_{i-1}, x_i] \times (y_{j-1}, y_j]$, and its *tile center* is the point $p(i,j) := \left(\frac{x_{i-1}+x_i}{2}, \frac{y_{j-1}+y_j}{2}\right) \in A(i,j)$. The *global routing graph* is a three-dimensional grid graph $G$ with

- $V(G) = \Big\{(i,j,k) : i = 1,\dots,n_x, j = 1,\dots,n_y, k = 1,\dots,n_z\Big\}$,
- $E(G) = \Big\{\{(i,j,k),(i',j',k')\} \in \binom{V(G)}{2} : |i-i'| + |j-j'| + |k-k'| = 1\Big\}$.

With each vertex $(i,j,k) \in V(G)$ we associate a *vertex coordinate* $p(i,j,k) := (p(i,j),k)$ and a *vertex area* $A(i,j,k) := A(i,j) \times \{k\}$.

The reasoning behind Definition 2.8 is explained in Section 1.3.2. Usually, the global routing graph is defined to only contain wiring edges expanding into the preference direction of the given layer. We do not make this distinction in Definition 2.8, but instead assume that there is no available routing space for wiring edges not expanding into preference direction of the layer. When not stated differently in the given context, then $G$ will refer to the global routing graph in this thesis. We note that when a graph $H$ is embedded into the global routing graph $G$ in the sense of Definition 2.3, then $H$ may contain many Steiner vertices of degree 2. In practice, one would not store $H$ in such a way, but allow a single long straight edge in $H$ to correspond to multiple consecutive edges in $G$, potentially reducing the number of edges in $H$ by a large amount. For simplicity of notation and presentation, however, we stick to the simpler way of embedding $H$ into $G$ given by Definition 2.3.
Traditionally, global routing is considered to output Steiner trees that are subgraphs of the global routing graph. Here, every pin on the chip is mapped to one vertex of the global routing graph based on the actual positions of its metal shapes. However, in the VLSI design flow where our global router is used, the global router must output Steiner trees that connect to the actual metal shapes of the pins, which are represented in a

simplified manner as points by the exact pin shapes from Definition 2.7. Complementary to exact shapes, we define *projected pin shapes* that will be used predominantly during global routing:

DEFINITION 2.9. Let $G$ be the global routing graph and $\Pi$ be the set of pins. *Projected pin shapes* are defined by a mapping $p_{\mathrm{pr}} \colon \Pi \to V(G)$. When it is clear from the context, $p(\pi)$ for $\pi \in \Pi$ either denotes $p_{\mathrm{ex}}(\pi)$ or $p_{\mathrm{pr}}(\pi)$.

Most of the time $p_{\mathrm{pr}}(\pi)$ is the vertex whose vertex area contains $p_{\mathrm{ex}}(\pi)$, but it is generally also possible to project pins to other vertices, e.g. neighboring vertices in cases where the capacity estimation is inadequate due to the blockage structure. Analogously to Definitions 2.7 and 2.9, exact and projected wire shapes can be defined. We refrain from a technical definition and rather use the term loosely when required. When referring to nets and pins in this thesis, we will implicitly assume exact and projected pin shapes to be given through the respective mappings. We can now distinguish between Steiner trees connecting exact and projected pin shapes:

DEFINITION 2.10. Let $S$ be the layered chip area, $G$ be the global routing graph and $N \subseteq N'$ for a net $N'$. A *Steiner tree connecting the exact shapes of* $N$ is a Steiner tree $Y$ with $N \subseteq V(Y)$ that is embedded rectilinearly into $S$ such that $p(\pi) = p_{\mathrm{ex}}(\pi)$ for $\pi \in N$. A *Steiner tree connecting the projected shapes of* $N$ is a Steiner tree $Y$ with $N \subseteq V(Y)$ that is embedded into $G$ such that $p(\pi) = p_{\mathrm{pr}}(\pi)$ for $\pi \in N$. When it is clear or indifferent in a given context, we just refer to a *Steiner tree for* $N$ to denote a Steiner tree connecting either the exact or projected shapes of $N$. Moreover, unless explicitly stated otherwise, we assume that $N$ is the set of leaves of $Y$. If applicable, $s$ refers to the source pin and $T$ to the set of sink pins of $N$, and $Y$ is implicitly regarded as an arborescence rooted at $s$.

The requirement that $N$ is the set of leaves of $Y$ is not a restriction, but merely a technicality that eases notation and description at some points — a Steiner tree not fulfilling this requirement can be transformed into one that does in linear time by introducing new vertices and edges of length zero and successively removing Steiner vertices of degree one. Next, we define distances of points in the three-dimensional space or entities that are embedded into the three-dimensional space:

DEFINITION 2.11. For $a, b \in \mathbb{R}^3$ we define $\mathrm{dist}(a, b) := |a_x - b_x| + |a_y - b_y|$ to be the rectilinear distance of the two-dimensional projections of $a$ and $b$. Let $v$ be either

- a pin with position $p(v) \in \mathbb{R}^3$ (either $p(v) = p_{\mathrm{ex}}(v)$ or $p(v)$ is the vertex position of $p_{\mathrm{pr}}(v)$), or
- a vertex of a graph that is embedded onto a position $p(v) \in \mathbb{R}^3$ (directly or by a chain of embeddings), or

- a point in $\mathbb{R}^3$, in which case we set $p(v) := v$.

Let the same hold true for $w$. Then we set $\operatorname{dist}(v, w) := \operatorname{dist}(p(v), p(w))$.

This definition of distances is rooted in our model of having a layered chip area, where the $z$-dimension has a different meaning than the $x$- and $y$-dimensions and is therefore treated differently. We also often use the term *length* in order to refer to distances, e.g. the length of an edge in a graph is the distance of its endpoints, and the length of a Steiner tree is the sum of its edge lengths. In the same manner, a *shortest* Steiner tree is one minimizing edge lengths. The meaning should be clear in the given context. Regarding Steiner trees as arborescences allows for the notion of subtrees rooted at certain vertices:

Definition 2.12. Let $Y$ be a Steiner tree for a net $N = \{s\} \sqcup T$, where $s$ is the source pin of $N$, and assume that $Y$ is directed in such a way that all vertices are reachable from $s$. Then $Y(v)$ for $v \in V(Y)$ denotes the subtree rooted at $v$, and $T(Y(v)) := T \cap V(Y(v))$ denotes the sinks in $Y(v)$. For $v, w \in V(Y)$, $P_Y(v, w)$ denotes the path from $v$ to $w$ in $\overleftrightarrow{Y}$, where $\overleftrightarrow{Y}$ is defined by $V(\overleftrightarrow{Y}) := V(Y)$ and $E(\overleftrightarrow{Y}) := E(Y) \cup \{(w, v) : (v, w) \in E(Y)\}$. Given that, we set $\operatorname{dist}_Y(v, w) := \sum_{(x,y) \in E(P_Y(v,w))} \operatorname{dist}(x, y)$ for $v, w \in V(Y)$.

An edge of a graph that is embedded into the chip area corresponds to a line segment:

Definition 2.13. Let $S$ be the layered chip area and $a, b \in S$. Then $L(a, b) := \{\mu a + (1 - \mu)b : \mu \in [0, 1]\} \cap S$ denotes the straight line segment between $a$ and $b$ in $S$. Additionally, we define $L^{\mathrm{2D}}(a, b)$ to be the two-dimensional projection of $L(a, b)$ (neglecting $z$-coordinates). If $G$ is a graph that is embedded into $S$, then we define $L(v, w) := L(p(v), p(w))$ and $L^{\mathrm{2D}}(v, w) := L^{\mathrm{2D}}(p(v), p(w))$ for $(v, w) \in E(G)$.

Edges of a graph embedded into the chip area may be subdivided by inserting a new vertex on the corresponding line segment:

Definition 2.14. Let $S$ be the layered chip area and $H$ be a graph that is embedded into $S$. A graph $H'$ is said to *originate from $H$ by subdivision of an edge* $(v, w) \in E(H)$ if $V(H') = V(H) \sqcup \{x\}$, $E(H') = (E(H) \setminus \{(v, w)\}) \cup \{(v, x), (x, w)\}$, and $p(x) \in L(v, w)$. A graph $H''$ is said to be a *subdivision of $H$* if $H''$ originates from $H$ through a sequence of edge subdivisions.

We continue with the *bounding box*:

Definition 2.15. Let $A \subseteq \mathbb{R}^2$ or $A \subseteq \mathbb{R}^3$. Then the *bounding box* $\mathrm{BB}(A) \subseteq \mathbb{R}^2$ of $A$ is the smallest axis-parallel rectangle containing the two-dimensional projection of $A$, i.e. $\mathrm{BB}(A) := [\min_{a \in A} a_x, \max_{a \in A} a_x] \times [\min_{a \in A} a_y, \max_{a \in A} a_y]$. Moreover, if $A'$, $A' \not\subseteq \mathbb{R}^2$

and $A' \not\subseteq \mathbb{R}^3$, is a set associated with positions $p\colon A' \to \mathbb{R}^2$ or $p\colon A' \to \mathbb{R}^3$, then we set $\mathrm{BB}(A') := \mathrm{BB}(p(A'))$.

We note that similarly to distances we define the bounding box always as two-dimensional, even for subsets of $\mathbb{R}^3$. The $z$-dimension will be treated separately in the given contexts. We conclude this section with a definition that allows for a more convenient notation:

Definition 2.16. Let $X$ be a set and $f\colon X \to \mathbb{R}^d$, $d \in \mathbb{N}$, be a function. If applicable and not explicitly stated otherwise, we define $f(X') := \sum_{x \in X'} f(x)$ for $X' \subseteq X$.

## 2.2. The Traditional Global Routing Problem

In this section we deal with the traditional way of formulating the global routing problem. We state the problem definition in Section 2.2.1 and give an overview on previous work in Section 2.2.2. Background information regarding the global routing problem, which naturally motivates our formulation of the Traditional Global Routing Problem, can be found in Section 1.3.2. This section is only meant to give an overview, but does not go into much detail. An algorithm by Müller, Radke and Vygen [87] for solving the Traditional Global Routing Problem is described in Section 4.2.

**2.2.1. The Problem Formulation.** We define the Traditional Global Routing Problem as follows:

Problem 2.17: Traditional Global Routing Problem

**Input:** The global routing graph $G$, edge lengths $l\colon E(G) \to \mathbb{R}_{\geq 0}$, a netlist $\mathcal{N}$, net weights $w\colon \mathcal{N} \to \mathbb{R}_{\geq 0}$, routing space usages $\mathrm{usg}\colon \mathcal{N} \times E(G) \to \mathbb{R}_{\geq 0}$.

**Task:** Find a Steiner tree $Y_N$ for all $N \in \mathcal{N}$ connecting the projected pin shapes of $N$ such that

$$\sum_{N \in \mathcal{N}} |\{e' \in E(Y_N) : p(e') = e\}| \cdot \mathrm{usg}(N, e) \leq 1 \quad \text{for all } e \in E(G),$$

and

$$\sum_{N \in \mathcal{N}} w(N) \cdot l(E(Y_N))$$

is minimized.

For wiring edges $(v, w) \in E(G)$ the standard choice is to set $l(v, w) := \mathrm{dist}(v, w)$. For via edges $(v, w) \in E(G)$, $l(v, w)$ can be used to express via costs, which can be adjusted from technology to technology. The edge usages $\mathrm{usg}(N, e)$ for $(N, e) \in \mathcal{N} \times E(G)$ are relative to the routing capacity of $e$, i.e. we assume unit capacities by adapting the

usage functions. The seemingly complicated formulation of routing capacity constraints is owed to the fact that we embed arbitrary Steiner trees into the global routing graph. In general, this allows for Steiner trees that are using the same edge in the global routing graph multiple times, which can make sense when computing RC-aware Steiner trees as in Chapter 5. However, it is easy to see that connecting a net $N$ by a Steiner tree $Y_N$ with $\left|\{e' \in E(Y_N) : p(e') = e\}\right| > 1$ for some $e \in E(G)$ is pointless in the Traditional Global Routing Problem. In that sense, the Traditional Global Routing Problem can equivalently be formulated in a model where the Steiner trees $Y_N$, $N \in \mathcal{N}$, are subgraphs of the global routing graph.

Even simple special cases of the Traditional Global Routing Problem are $NP$-hard: The special case with only one net contains the Rectilinear Minimum Steiner Tree Problem [37, 46] (cf. Section 2.3.2), and the special case where $\text{usg}(N, e) = 1$ for all $(N, e) \in \mathcal{N} \times E(G)$ and $|N| = 2$ for all $N \in \mathcal{N}$ corresponds to the Edge-Disjoint Paths Problem in Grid Graphs [112, 118]. The latter example also shows that it is $NP$-hard to find any feasible solution, as violating any routing capacity constraint makes a solution infeasible. In theory, this is reasonable. In practice, however, it is too rigid — here, one usually wants to minimize routing capacity constraint violations in cases where no solution obeying all constraints can be found. This is done for example by the modeling of the Traditional Global Routing Problem as a Min-Max Resource Sharing Problem by Müller, Radke and Vygen [87], which we further examine in Chapter 4.

This formulation of the global routing problem is not inherently timing-aware. To consider timing in an indirect manner, timing-critical nets can be given higher weights. Moreover, strict adherence to wire type and layer assignments (cf. Section 1.3.2.4.1) can easily be incorporated by always using the wire type assigned to the net, and setting $\text{usg}(N, e) = \infty$ if the layer assignment of $N \in \mathcal{N}$ forbids the use of $e \in E(G)$. An inherently timing-aware global routing framework is presented in Chapter 4.

**2.2.2. Previous Work.** Many of the recently published global routing approaches are based on heuristics which predominantly rely on a method called ripup-and-reroute. Roughly speaking, this method consists of starting with a reasonable initial solution and iteratively improving this solution by ripping out and rerouting nets until some abort criterion is met. The routing techniques used include *pattern routing*, meaning that only routes consisting of very simple patterns like L- and Z-shapes may be used, *monotonic routing*, meaning that only routes without detours are allowed, and *maze routing*, which is the term used for route construction that relies on actual shortest path computations. The edge costs for the route computations are usually chosen heuristically. Such ripup-and-reroute techniques are prevalently used among the contestants of the ISPD global

routing contests from 2007 and 2008 [66, 67], which we examine in Section 2.2.2.1. Another method to address global routing is via flow- and LP-based approaches, which we cover in Section 2.2.2.2. A survey on (early) global routing algorithms is given by Hu and Sapatnekar [61].

2.2.2.1. *The ISPD Global Routing Contests.* Noteworthy in the context of development of global routing algorithms are the ISPD global routing contests 2007 and 2008 [66, 67], where problem formulation and evaluation metrics result in a global routing problem that is very similar to the Traditional Global Routing Problem. Here, most top-ranked contestants use techniques that are similar (but not limited) to the ripup-and-reroute techniques explained above. The contestants include NTHU-Route [25], FastRoute [124], MaizeRouter [81], FGR [98], BoxRouter [26] and Archer [89]. A survey is given by Moffitt et al. in [83].

Compared to industrial global routing instances, the benchmarks used for the ISPD contests are significantly simplified. In particular, each net consumes the same amount of routing space across all layers, which is usually not given in practice. Moreover, via usages are ignored. This makes it natural to consider a simplified instance with only two layers and perform a layer assignment afterwards. An analysis of this matter and the global routing problem in general is given by Moffitt [82].

2.2.2.2. *Flow- and LP-Based Approaches.* Theoretically more profound approaches for solving the Traditional Global Routing Problem were first obtained through flow-based and integer linear programming formulations of global routing. An early work in this category is given by Shragowitz and Keel [113], who deal with a special case of the Traditional Global Routing Problem where $|N| = 2$ for all $N \in \mathcal{N}$, which is also a special case of the well known Minimum Cost Multi-Commodity Flow Problem. They start with a cost-optimal solution ignoring routing capacities and reduce overflow afterwards by an iterative method that bears resemblance to the ripup-and-reroute approaches mentioned earlier. A multi-commodity flow formulation of the global routing problem is also solved by Carden, Li and Cheng [22]. In contrast to [113], their formulation can also handle multi-terminal nets. Wu et al. [123] use an integer linear programming formulation of the global routing problem: They subdivide the global routing problem into smaller subproblems on rectangular regions on the chip and solve these subproblems via integer linear programming allowing only a predefined set of candidate routes.

Raghavan and Thompson [93] give a linear programming formulation of the global routing problem that is based on multi-commodity flows. They show how to use a randomized rounding technique [92] to get an integral solution whose deviation from the LP solution can be bounded. However, they do not outline a method to solve the underlying

linear program. This is addressed by Albrecht [2], who uses a modified version of the multi-commodity flow approximation algorithm by Garg and Könemann [38, 39] to solve a linear programming formulation of the global routing problem. Follow-ups of this approach are given by Vygen [117], who also incorporates delay bounds and power minimization, and Müller [85], who incorporates optimization of manufacturing yield. Finally, Müller, Radke and Vygen [87] formulate global routing as a more general Min-Max Resource Sharing Problem and present an approximation algorithm to solve it. We examine their approach in more detail in Section 4.2 and extend it in Section 4.3 to incorporate timing constraints.

## 2.3. The Minimum Steiner Tree Problem

As different variants of the well-known Minimum Steiner Tree Problem appear as subproblems in different parts of this thesis, we shortly introduce them in this section. Instead of pins that are embedded into the global routing graph or the layered chip area, we conventionally speak of *terminals* that are embedded into a general graph or $\mathbb{R}^2$ in this section. However, apart from that, the notation from Section 2.1 is used.

**2.3.1. The Minimum Steiner Tree Problem in Graphs.** We start with the Minimum Steiner Tree Problem in Graphs:

Problem 2.18: Minimum Steiner Tree Problem in Graphs

**Input:** A graph $G$, edge costs $c\colon E(G) \to \mathbb{R}_{\geq 0}$, a set of terminals $N$ that are embedded into $G$.

**Task:** Find a Steiner tree $Y$ for $N$ minimizing $c(E(Y))$.

This problem is known to be $NP$-hard [70], but efficient constant-factor approximation algorithms exist. The best known approximation guarantee of 1.39 is achieved by the algorithm of Byrka et al. [21]. The running time is polynomial, but for our purposes presumably too large in practice. A fast running time can be achieved by making use of the fact that a minimum terminal spanning tree in the metric closure of $G$ yields an approximation ratio of 2 for the Minimum Steiner Tree Problem in Graphs (see e.g. [75]). In our implementation we use an algorithm that is essentially the Standard Block Solver described by Müller [86] (Section 4.7.3), which iteratively connects connected components of wires and pins by shortest path computations until the whole net is connected. We will not elaborate on this any further in this thesis, but rather point to the description given in [86]. An overview on the Minimum Steiner Tree Problem in Graphs can be found in the book of Korte and Vygen [75].

**2.3.2. The Rectilinear Minimum Steiner Tree Problem.** The Rectilinear Minimum Steiner Tree Problem is of major importance in VLSI design due to the fact that wires are required to run parallel to the $x$- or $y$-axis. It can be formulated as follows:

Problem 2.19: Rectilinear Minimum Steiner Tree Problem

**Input:** A set $N$ of terminals that are embedded into $\mathbb{R}^2$.

**Task:** Find a Steiner tree $Y$ for $N$ minimizing $\sum_{(v,w)\in E(Y)} \text{dist}(v,w)$.

Here, dist refers to the rectilinear distance as in Definition 2.11. The Rectilinear Minimum Steiner Tree Problem can be reduced to the Minimum Steiner Tree Problem in Graphs through the use of the Hanan grid [46], and it is also $NP$-complete, as was shown by Garey and Johnson [37].
When it comes to solving the Rectilinear Minimum Steiner Tree Problem, polynomial time approximation schemes by Arora [7] and Rao and Smith [94] exist. However, as running time is critical in our scenario, we use a different approach: If the number of terminals does not exceed 8, then we use the optimal algorithm by Chu and Wong [27], which is based on a lookup table and runs very fast in practice for small terminal sets. In our application, this already covers the majority of the cases. For larger terminal sets, we use different approximation algorithms depending on the size of the terminal set. Here, one can make use of the fact that a minimum terminal spanning tree already provides a 1.5-approximation for the Rectilinear Minimum Steiner Tree Problem [65].

**2.3.3. The Rectilinear Minimum Steiner Tree with Prewires Problem.** We now describe a variant of the Rectilinear Minimum Steiner Tree Problem where we are already given prewires that are free to use. This will be important in Chapter 7, where we often start with a set of wires that almost connect our terminal set:

Problem 2.20: Rectilinear Minimum Steiner Tree with Prewires Problem

**Input:** A set of terminals $N$ that are embedded into $\mathbb{R}^2$ and a Steiner forest $Y_0$ connecting a subset $N_0 \subseteq N$ (possibly $N_0 = \emptyset$).

**Task:** Compute a subgraph $Y_0'$ of a suitable subdivision of $Y_0$ and a Steiner tree $Y$ for $N$ with $E(Y_0') \subseteq E(Y)$ such that $\sum_{(v,w)\in E(Y)\setminus E(Y_0')} \text{dist}(v,w)$ is minimized.

Here, the Steiner forest $Y_0$ may contain Steiner vertices of degree 1. In our application, $Y_0$ will usually be a Steiner tree. Clearly, the problem is $NP$-hard, as it contains the Rectilinear Minimum Steiner Tree Problem. We will therefore use an approximation algorithm to solve it. This algorithm has been developed and implemented by

Rodion Permin and works like this: It first computes a minimum spanning tree $X$ on the complete graph $G$ with cost function $c\colon E(G) \to \mathbb{R}_{\geq 0}$, where

- $V(G) = N \cup M$, where $M$ is the set of connected components of $Y_0$,
- $c(\{v, w\})$ is the minimum rectilinear distance between any two points in $A(v)$ and $A(w)$, where $A(u)$ for $u \in V(G)$ is defined by $A(u) := \{(p(u)_x, p(u)_y)\}$ if $u \in N$, and $A(u) = \bigcup_{e \in E(u)} L^{2\mathrm{D}}(e)$ if $u \in M$ (cf. Definition 2.13).

The algorithm then decomposes $X$ into maximum subgraphs $X_1, \ldots, X_k$ such that $|\delta_{X_i}(x)| = 1$ for every $x \in V(X_i) \cap M$, $i = 1, \ldots, k$. Thereafter, every $X_i$, $i = 1, \ldots, k$, is replaced by an approximately minimum rectilinear Steiner tree, where connection points at prewire components are chosen to minimize the distance to their neighbor in the respective subgraph $X_i$. The individual rectilinear Steiner trees are computed as outlined in Section 2.3.2.

This process results in a solution of the Rectilinear Minimum Steiner Tree with Prewires Problem. It can be shown that this algorithm is a 1.5-approximation algorithm for the Rectilinear Minimum Steiner Tree with Prewires Problem, which is basically due to the fact that a rectilinear minimum spanning tree is an 1.5-approximation for a minimum rectilinear Steiner tree [65].

CHAPTER 3

# VLSI Timing Basics

As one of the main focuses of this thesis is to provide a timing-aware global routing framework, we use this chapter to shortly explain a few basic concepts in the field of VLSI timing that are necessary for understanding our approach. We start with the basic notion of a signal and its associated parameters in Section 3.1 and then continue with our definition of the timing graph in Section 3.2. In Section 3.3 we formulate timing constraints and show how to check them using static timing analysis. Section 3.4 then introduces the Elmore delay model, which is the central delay model that is used throughout this thesis. Finally, Section 3.5 explains slew and capacitance limits. Generally, most of the material presented in this chapter is strongly simplified in order to match the purpose of this thesis without going into too much detail. A more detailed description of the topics presented here is given by Sapatnekar [**104**].

## 3.1. Signals

The logical state of a digital electronic circuit at any given point in time can be described by mapping every pin to a finite set of possible values. In our context, this set only contains the values 0 and 1, which are physically represented by two voltage levels $V_{\text{ss}}$ and $V_{\text{dd}}$. Here, $V_{\text{ss}}$, often also called *ground*, is the reference point with value 0, while $V_{\text{dd}}$ represents value 1. A *signal* can then be regarded as a voltage change at a given pin. However, this voltage change does not occur instantly, but gradually over a period of time. To deal with this, we identify a signal with the two parameters *arrival time* and *slew*: The arrival time is the point in time when the voltage reaches $0.5V_{\text{dd}}$, while the slew is the transition time between $0.1V_{\text{dd}}$ and $0.9V_{\text{dd}}$.[1] This concept is illustrated by Figure 3.1.

## 3.2. The Timing Graph

As we use a rather simple timing model throughout this thesis, we also use a rather simple timing graph, which is basically a condensed version of the netlist graph introduced in Section 1.1.3. Our timing graph $D$ is an acyclic digraph with vertex set $V(D) = V_{\text{in}} \sqcup V_{\text{out}} \sqcup V_{\text{gate}}$, where $V_{\text{in}}$ comprises the primary input and latch output pins on the

[1] Of course, other voltages might also be chosen for this definition.

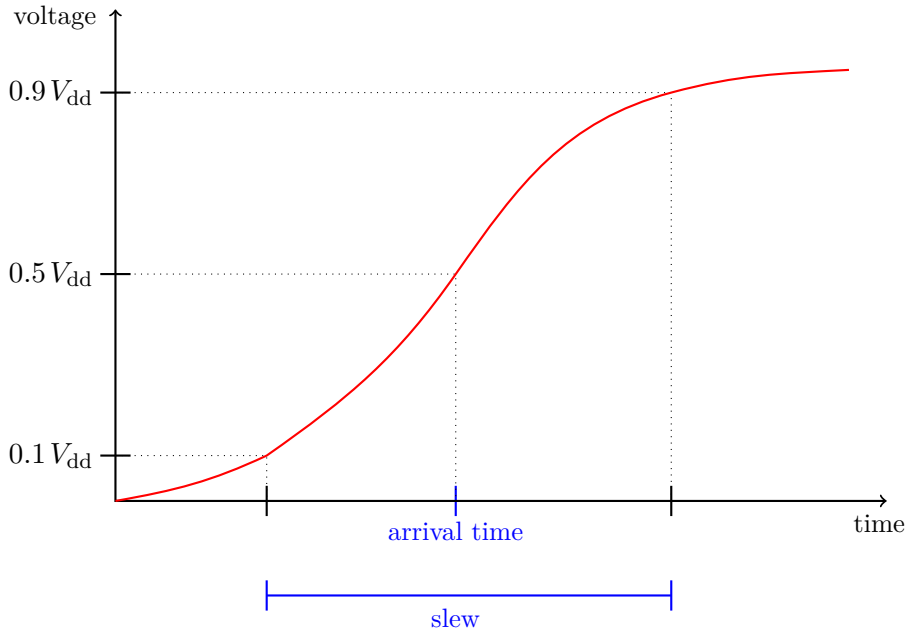



FIGURE 3.1. Illustration of a rising signal changing the voltage from $V_{ss}$ to $V_{dd}$. Arrival time and slew of the signal are marked in blue.

chip, $V_{out}$ the set of primary outputs and latch inputs and $V_{gate}$ the set of input pins of gates. The edges of $D$ correspond to signal propagation: $D$ contains an edge $(u, v)$ if and only if either $u \in V_{in}$ and $v$ is a sink pin of the net driven by $u$, or if $u \in V_{gate}$ and $v$ is a sink pin of a net driven by an output pin of the circuit that has $u$ as input pin. This construction is illustrated by Figure 3.2.

If the number of input pins of each gate is bounded by a small constant, which is usually given in practice, then we have $|E(D)| = \mathcal{O}(|V(D)|)$. In cases where gates with a large number of input pins exist, additional vertices can be added to the timing graph to achieve a linear number of edges again.

## 3.3. Timing Constraints

There are several methods to formulate timing constraints on a chip. In Section 3.3.1 we give a simple exponential model as an introduction, and then turn towards the better method of checking timing constraints via static timing analysis in Section 3.3.2. In this context it is important to note that we only address *late mode timing* in this thesis. To obtain a working chip, one must also consider *early mode timing*, i.e. making sure that certain signals do not arrive too early. However, as fulfilling late mode timing constraints is the main challenge during timing optimization, we neglect early mode timing.

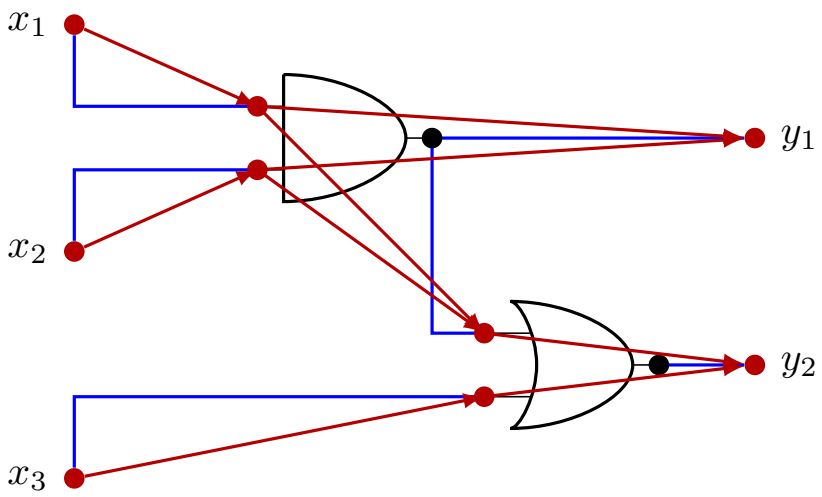



FIGURE 3.2. Illustration of the timing graph $D$ from Section 3.2: The picture shows the netlist from Figure 1.2. Signals flow from left to right, which means that we have $V_{\text{in}} = \{x_1, x_2, x_3\}$, $V_{\text{out}} = \{y_1, y_2\}$, and $V_{\text{gate}}$ consists of the input pins of the two gates. This results in the timing graph drawn in red.

**3.3.1. Path-Based Timing Constraints.** In this section we give a simple exponential size method to clarify the concept behind timing constraints. Assume we are given *arrival times* $\text{at}\colon V_{\text{in}} \to \mathbb{R}$, *required arrival times* $\text{rat}\colon V_{\text{out}} \to \mathbb{R}$ and signal delays $d\colon E(D) \to \mathbb{R}_{\geq 0}$. Let $\mathcal{P}$ contain all paths in $D$ that start in a vertex in $V_{\text{in}}$ and end in a vertex in $V_{\text{out}}$, and for $P \in \mathcal{P}$ let $s(P)$ and $t(P)$ denote the start and end point of $P$, respectively. Then timing constraints can be expressed by the inequalities

$$\text{at}(s(P)) + \sum_{e \in E(P)} d(e) \leq \text{rat}(t(P)) \qquad \text{for all } P \in \mathcal{P}. \tag{3.1}$$

Of course, $|\mathcal{P}|$ can be exponential in the size of $D$, which makes this method to express timing constraints impractical. We therefore introduce a better method in Section 3.3.2, which allows expressing and checking timing constraints in polynomial time.

**3.3.2. Static Timing Analysis.** As in Section 3.3.1 we assume here that we are given arrival times $\text{at}\colon V_{\text{in}} \to \mathbb{R}$, required arrival times $\text{rat}\colon V_{\text{out}} \to \mathbb{R}$ and signal delays $d\colon E(D) \to \mathbb{R}_{\geq 0}$. We extend the arrival times to $V(D)$ by propagating them in topological order through $D$: For $v \in V(D) \setminus V_{\text{in}}$ we set

$$\text{at}(v) := \max\{\text{at}(u) + d(u, v) : (u, v) \in \delta_D^-(v)\}.$$

In the same manner, we extend the required arrival times to $V(D)$ by propagating them in reverse topological order through $D$: For $v \in V(D) \setminus V_{\text{out}}$ we set

$$\text{rat}(v) := \min\{\text{rat}(w) - d(v,w) : (v,w) \in \delta_D^+(v)\}.$$

For every $v \in V(D)$ we can now define the *slack* at $v$ as

$$\text{slack}(v) := \text{rat}(v) - \text{at}(v).$$

It is now straightforward to check that the inequalities (3.1) from Section 3.3.1 are fulfilled if and only if $\text{slack}(v) \geq 0$ holds for all $v \in V_{\text{out}}$. In that sense, we define the *worst slack (ws)* as

$$\text{ws} := \min\{\text{slack}(v) : v \in V_{\text{out}}\}$$

and the *figure of merit (fom)* as

$$\text{fom} := \sum_{v \in V_{\text{out}}} \min\{\text{slack}(v) - \text{slacktgt}, 0\},$$

where the *slack target* slacktgt is a parameter that is usually in the range of 5 to 10 picoseconds for our test cases. The denotation "figure of merit" is quite generic but commonly used, and sometimes the more expressive term "sum of negative slacks" is used instead. We note that as a result of a positive slack target it might happen that the figure of merit is strictly negative, although the worst slack is non-negative. The worst slack and the figure of merit are the quantities that we use to measure the quality of timing results throughout this thesis. As already stated in the introduction to this section, this was only a very coarse and simplified overview on static timing analysis. A detailed description is given by Sapatnekar [104].

## 3.4. The Elmore Delay Model

In this section we are going to introduce the *Elmore delay model*, which is the delay model that we use for our internal delay computations. We start with a short general introduction to the Elmore delay model in RC trees in Section 3.4.1 and then describe how to apply it in our global routing context in Section 3.4.2. Major parts of this section are adopted from our paper [108].

**3.4.1. The Elmore Delay Model in RC Trees.** The Elmore delay model is a rather simple method to approximate the signal delay through what is called an *RC tree*. It was originally introduced by Elmore [35] in 1948 and later on extended by Rubinstein, Penfield and Horowitz [99], who also give a simple formula that can be used for fast computation. Their model works in a tree structured network consisting of a discrete number of resistors and capacitors, where each resistor has a fixed resistance and each capacitor has a fixed capacitance. We number the $k$ resistors and $n$ capacitors

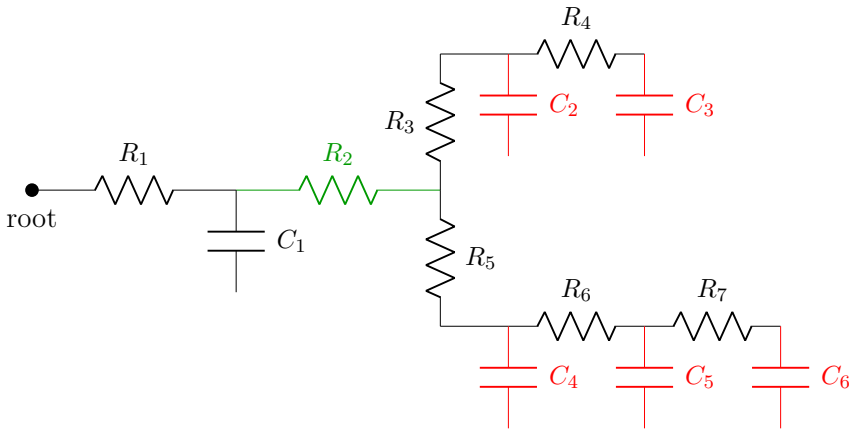



FIGURE 3.3. An RC tree with seven resistors and six capacitors: The subtree rooted at the green resistor 2 contains the five red capacitors $2, 3, 4, 5$ and 6. Therefore, resistor 2 imposes a delay of $R_2 \cdot C_2^{\text{down}} = R_2 \cdot (C_2 + C_3 + C_4 + C_5 + C_6)$. In the same way, the delay imposed by resistor 6 amounts to $R_6 \cdot (C_5 + C_6)$. One can see that resistors closer to the root have a higher downstream capacitance and therefore impose higher delays per resistance unit.

for some $k, n \in \mathbb{N}$ consecutively with resistances $R_1, ..., R_k$ and capacitances $C_1, ..., C_n$, respectively, and let $C_i^{\text{down}}$ for $i \in \{1, ..., k\}$ denote the sum of capacitances of the capacitors in the subtree rooted at resistor $i$. Rubinstein, Penfield and Horowitz show in [99] that the Elmore delay at capacitor $j$ is then given by $\sum_{i \in I_j} R_i \cdot C_i^{\text{down}}$, where $I_j \subseteq \{1, ..., k\}$ denotes the set of resistors on the path from the root to capacitor $j$. Figure 3.3 gives an illustration of this.

We omit their definition of an RC tree at this point but rather give a graph theoretical interpretation with emphasis on our application. In this regard, an RC tree can be modeled as a directed Steiner tree $Y$ with a source $s \in V(Y)$ and sinks $T \subseteq V(Y)$, where $s$ is the origin of the signal and the orientation of the edges corresponds to the direction in which the signal propagates. Here, the source $s$ is regarded as a resistor with resistance $R(s) \geq 0$ and the sinks are regarded as capacitors with capacitances $C(t) \geq 0, t \in T$. Each edge in the tree corresponds to a metal wire, which is simultaneously a resistor with resistance $R := R_{\text{wire}} \cdot l$ and a capacitor with capacitance $C := C_{\text{wire}} \cdot l$, where $l$ is the length of the wire and $R_{\text{wire}}, C_{\text{wire}} \geq 0$ are given constants. Steiner points do not have any resistance or capacitance.

To match the previous model, a wire with resistance $R$ and capacitance $C$ is divided into two capacitors with capacitance $C/2$ and one resistor with resistance $R$ in between, or alternatively (and equivalently) into two resistors with resistance $R/2$ and one capacitor with capacitance $C$ in between. It can be shown that in terms of Elmore delay, this is

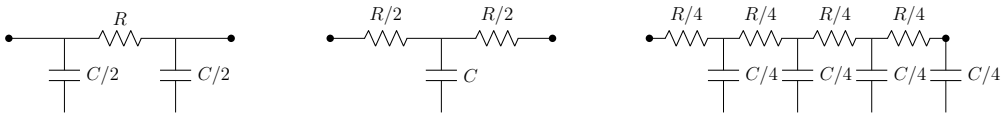



FIGURE 3.4. A wire with resistance $R$ and capacitance $C$ is modeled as two capacitors with a resistor in between (left). In terms of Elmore delay, it is equivalent to modeling it as two resistors with a capacitor in between (center). It is also the same as modeling it as $k$ alternating resistors and capacitors and taking the limit for $k \to \infty$ (right shows $k = 4$).

exactly the limit of dividing the wire into $k$ alternating resistors and capacitors with resistance $R \,/\, k$ and $C \,/\, k$, respectively, when $k$ goes to infinity. An illustration is given by Figure 3.4.

**3.4.2. Application in Global Routing.** Based on the preparatory work from Section 3.4.1 we can state a definition of Elmore delay that is geared to VLSI routing. Here, we use some of the notation from Definitions 2.12 and 2.16:

DEFINITION 3.1. Assume we are given a Steiner tree $Y$ rooted at $s \in V(Y)$, a set of sinks $T \subseteq V(Y)$, a source resistance $R(s) \in \mathbb{R}_{\geq 0}$, sink capacitances $C\colon T \to \mathbb{R}_{\geq 0}$ and edge resistances and capacitances $R, C\colon E(Y) \to \mathbb{R}_{\geq 0}$. Then we define the *downstream capacitance of* $v \in V(Y)$ as

$$C(Y(v)) := C(E(Y(v)) + C(T(Y(v))),$$

which allows us to define the *Elmore delay from* $s$ *to* $t \in T$ *in* $Y$ as

$$d_Y(t) := R(s) \cdot C(Y(s)) + \sum_{(v,w) \in E(P_Y(s,t))} R(v,w) \left( \frac{C(v,w)}{2} + C(Y(w)) \right).$$

We call the term $R(s) \cdot C(Y(s))$ the *source delay* and the term $\sum_{(v,w) \in E(P_Y(s,t))} R(v,w) \left( \frac{C(v,w)}{2} + C(Y(w)) \right)$ the *wire delay* from $s$ to $t$ in $Y$.

A great advantage of the Elmore delay model is that it can be computed in linear time by first computing the downstream capacitances $C(Y(v)), v \in V(Y)$, in reverse topological order, and then computing the delay to all vertices in topological order. This way it is fast to compute while being reasonably accurate in most cases. It has been shown by Boese et al. [16] that even in cases where it is not very accurate, it is still a high fidelity estimate, which means that improving Elmore delay will almost certainly improve delay simulated by very accurate tools that are too computationally expensive to be called more often than a very few times in the VLSI design flow. For these reasons, the Elmore

delay model has been the delay model of choice in VLSI design for the last decades. For more on it, see also Gupta et al. [44], Peyer [90] or the book of Celik et al. [23].
To apply Definition 3.1 in the context of global routing, we need to clarify how the edge resistances and capacitances used in the definition are obtained:

DEFINITION 3.2. Let $S = A \times Z$ with $Z = \{1, \dots, n_z\}$ be the layered chip area, $G$ be a graph that is embedded rectilinearly into $S$, and assume that we are given functions $R_{\text{wire}}, C_{\text{wire}}, \mathrm{R}_{\text{via}} \colon Z \to \mathbb{R}_{\geq 0}$. We first define an extension $\mathrm{R}_{\text{via}} \colon Z^2 \to \mathbb{R}_{\geq 0}$ by setting

$$\mathrm{R}_{\text{via}}(z_1, z_2) := \sum_{z=\min\{z_1,z_2\}}^{\max\{z_1,z_2\}-1} \mathrm{R}_{\text{via}}(z)$$

for $(z_1, z_2) \in Z^2$. Then for $(v, w) \in E(G)$ and $z_v := p(v)_z, z_w := p(w)_z$ we define

$$R(v,w) := \begin{cases} R_{\text{wire}}(z_v) \cdot \operatorname{dist}(v,w) & \text{if } z_v = z_w, \\ \mathrm{R}_{\text{via}}(z_v, z_w) & \text{if } z_v \neq z_w, \end{cases}$$

$$C(v,w) := \begin{cases} C_{\text{wire}}(z_v) \cdot \operatorname{dist}(v,w) & \text{if } z_v = z_w, \\ 0 & \text{if } z_v \neq z_w, \end{cases}$$

as the *resistance* and *capacitance of* $(v, w)$, respectively.

This definition of wire and via RC values is in sync with the modeling of wires in RC trees from Section 3.4.1. The functions $R_{\text{wire}}$, $C_{\text{wire}}$ and $\mathrm{R}_{\text{via}}$ are assumed to be given as part of the technology parameters. They are also dependent on the wire type that is used: Thicker wires have less resistance and more capacitance, and the (coupling) capacitance of a wire decreases as its spacing to neighboring wires increases. However, as we will always be using one fixed wire type when routing a certain net, we can assume that the functions $R_{\text{wire}}$, $C_{\text{wire}}$ and $\mathrm{R}_{\text{via}}$ are applicable to all wires as long as the net is fixed. Here, the value $\mathrm{R}_{\text{via}}(n_z)$ is actually irrelevant, but we still include it for ease of notation.
We note that when a graph $H$ is embedded into another graph $G$ (e.g. a Steiner tree into the global routing graph) and resistances and capacitances are defined for $E(G)$, then these resistances and capacitances are naturally extended to $E(H)$ through Definition 2.4. One more thing to remark is that we always assume zero via capacitances throughout this thesis. The reason is that via capacitances are small in current technologies and are therefore assumed to be zero in our data set. Making use of this will simplify some of our results. It is also worth mentioning that Definitions 3.1 and 3.2 are well-behaved with respect to insertion of Steiner points of degree 2: If an edge is subdivided by a Steiner point of degree 2 (in the sense of Definition 2.14), then the Elmore delay will not change. This can be proven by a straightforward computation.

Finally, we assume the source resistance and sink capacitances in Definition 3.1 to be given in the input. Depending on the timing engine that is used in the VLSI design flow, these values might not be retrievable in this simple form — for instance, driver delays are modeled by more complex functions in the IBM design flow where BonnRouteGlobal is used. In that case, approximations have to be used. We conclude this section with a simple identity:

LEMMA 3.3. *Let $Y$ be a Steiner tree and $c, d\colon E(Y) \to \mathbb{R}$. Then we have*

$$\sum_{(v,w)\in E(Y)} c(v,w) \sum_{(x,y)\in E(Y(w))} d(x,y) = \sum_{(x,y)\in E(Y)} d(x,y) \sum_{(v,w)\in E(P_Y(s,x))} c(v,w).$$

PROOF. For $(v,w),(x,y) \in E(Y)$ we have $(x,y) \in E(Y(w)) \Leftrightarrow (v,w) \in E(P_Y(s,x))$, so on both sides of the equation the same summands appear. □

Letting $c$ be edge resistances and $d$ be edge capacitances, this lemma implies an alternative way of expressing the sum of all wire delays (accumulated over all edges) in the tree. However, the functions $c$ and $d$ can be arbitrary. Lemma 3.3 will be helpful in subsequent chapters dealing with Elmore delays.

## 3.5. Slew and Capacitance Limits

As delay computations are only reliable within certain slew and downstream capacitance ranges, additional constraints called *slew limits* and *capacitance limits* have to be obeyed. This means that for every sink pin $t$ (i.e. primary output or circuit input pin) on the chip we are given an interval $[\text{slewmin}(t), \text{slewmax}(t)] \subseteq \mathbb{R}_{\geq 0}$, and the slew of any signal at $t$ must be contained in this interval. Similarly, the capacitance of the net driven by any source pin $s$ (i.e. primary input or circuit output pin) must be contained in an interval $[\text{capmin}(s), \text{capmax}(s)] \subseteq \mathbb{R}_{\geq 0}$. These intervals depend on the circuit of the given pin or are asserted for primay input and output pins. Here, the difficult task is to satisfy the maximum limits, i.e. letting slews and capacitances not become too large.

We will not be modeling these constraints directly within our framework, although at least modeling capacitance limits would be straightforward in the resource sharing framework presented in Chapter 4. However, we will often list *electrical violations*, i.e. the number of violations of slew and capacitance limits, as a metric in our various experimental results sections. As a rule of thumb, one can say that with respect to routing, improving slews correlates well with improving RC delays, and satisfying capacitance limits can be achieved by keeping Steiner trees short.

CHAPTER 4

# Global Routing with Timing Constraints

In this chapter we present a global routing framework that directly models timing constraints as formulated in Section 3.3. Before diving into this framework, we give an overview on previous work regarding timing-aware global routing in Section 4.1. Afterwards, we outline how to generally model global routing as a Min-Max Resource Sharing Problem in Section 4.2. Lastly, Section 4.3 constitutes the centerpiece of this chapter, as it describes how to incorporate timing constraints into the global routing framework from Section 4.2.

The modeling of global routing as a Min-Max Resource Sharing Problem and the algorithm used to solve it are due to Müller, Radke and Vygen [87]. The incorporation of timing constraints is joint work with Stephan Held, Dirk Müller, Daniel Rotter, Vera Traub and Jens Vygen [53]. The paper [53] is based on the two conference papers [54] and [107], but also contains extensions and improvements. Here, we present the essential points of [53], but refer the reader to the paper for some of the proofs. Major parts of [53] — and therefore also major parts of this chapter — are also already published in [97].

The framework presented in this chapter will use the RC-aware routing oracle from Chapter 5 as a central subroutine. We therefore do not present experiment results in this chapter. Instead, the experimental results presented in Section 5.5 cover our combined results from this chapter and Chapter 5.

## 4.1. Previous Work

There have been several approaches to consider timing constraints during global routing. Huang et al. [64] enforce net-based delay bounds by disregarding Steiner trees violating these bounds. This approach is extended by Hong et al. [57]: They use path-based delay bounds and accept subpar delays in individual nets as long as their path-based bounds are satisfied. Samanta et al. [102] restrict the choices for a tree for a net to a set of pre-computed timing-driven Steiner trees. They then compute a convex combination of those pre-computed trees for each net minimizing total quadratic overflow.

Delay bounds are also used by Vygen [117], who describes an approach where delay bounds can be imposed on certain subsets of nets, e.g. on the critical paths on the

chip. His approach is closely related to the modeling of global routing as a Min-Max Resource Sharing Problem that we describe in Section 4.2. As such, delay bounds are not hard constraints, but violations of delay bounds and violations of routing capacity constraints are minimized simultaneously.
Following a different approach, Hu and Sapatnekar [62], Yan and Lin [126] and Yan et al. [125] embed timing-driven Steiner trees congestion-aware into the global routing graph. They use various different techniques for reducing congestion, e.g. making use of the flexibility that is given by changing the embedding of diagonal edges.
Lastly, Albrecht et al. [3] combine buffering and global routing: For the special case where all nets have two terminals, they can give a fully polynomial time approximation scheme for minimizing a weighted sum of buffer and wire area given buffer congestion, wire congestion, and sink delay constraints.

## 4.2. Global Routing as Min-Max Resource Sharing Problem

In this section we introduce the Min-Max Resource Sharing Problem, present an algorithm by Müller, Radke and Vygen [87] to efficiently solve it, and show how the Traditional Global Routing Problem can be modeled as a Min-Max Resource Sharing Problem. We start with the formulation of the Min-Max Resource Sharing Problem:

Problem 4.1: Min-Max Resource Sharing Problem

**Input:** A finite set $\mathcal{R}$ of *resources*, a finite set $\mathcal{C}$ of *customers*, a convex set $B_c$ of possible solutions for each $c \in \mathcal{C}$, convex functions $\mathrm{usg}_{c,r} \colon B_c \to \mathbb{R}_{\geq 0}$ for all $(c,r) \in \mathcal{C} \times \mathcal{R}$, *oracle* functions $f_c \colon \mathbb{R}^{\mathcal{R}}_{\geq 0} \to B_c$ for all $c \in \mathcal{C}$.

**Task:** Find a solution $b(c) \in B_c$ for all $c \in \mathcal{C}$ approximately attaining

$$\lambda^* := \inf \left\{ \max_{r \in \mathcal{R}} \sum_{c \in \mathcal{C}} \mathrm{usg}_{c,r}(b(c)) : b(c) \in B_c \ (c \in \mathcal{C}) \right\}.$$

The value $\mathrm{usg}_{c,r}(b)$ for $(c,r) \in \mathcal{C} \times \mathcal{R}$ and $b \in B_c$ denotes the fraction of resource $r$ that is used by solution $b$ for customer $c$. The sets $B_c$, $c \in \mathcal{C}$, are called *blocks*, and they are usually not given as an explicit list. Instead, we assume that we are given oracle functions that can optimize linear functions over the blocks. To be precise, an oracle for a customer $c \in \mathcal{C}$ is a function $f_c \colon \mathbb{R}^{\mathcal{R}}_{\geq 0} \to B_c$ that gets resource prices $\mathrm{price}(r)$, $r \in \mathcal{R}$, as input, and computes a solution $b \in B_c$ approximately minimizing $\sum_{r \in \mathcal{R}} \mathrm{price}(r) \cdot \mathrm{usg}_{c,r}(b)$. In our application, all blocks will be compact. Therefore, the infimum in the objective function will always be attained.

If we model the TRADITIONAL GLOBAL ROUTING PROBLEM from Section 2.2 without the wire length objective function as a MIN-MAX RESOURCE SHARING PROBLEM, then $\mathcal{R}$ equals $E(G)$, $\mathcal{C}$ equals $\mathcal{N}$, $B_N$ for $N \in \mathcal{N}$ is the convex hull of the set of Steiner trees connecting $N$ (in some adequate representation), and $\text{usg}_{N,e}(Y)$ for a Steiner tree $Y$ for $N$ equals $\text{usg}(N, e)$ if $Y$ uses $e$, and 0 otherwise. The oracle function for each customer would be an algorithm for the MINIMUM STEINER TREE PROBLEM IN GRAPHS (cf. Section 2.3.1). $\lambda^*$ then is the best attainable maximum edge usage, and it exceeds 1 if and only if the given instance of the TRADITIONAL GLOBAL ROUTING PROBLEM is infeasible. Therefore, it follows that the MIN-MAX RESOURCE SHARING PROBLEM is $NP$-hard (cf. Section 2.2.1).

**Algorithm 1** Resource Sharing Algorithm

**Input:** An instance of the MIN-MAX RESOURCE SHARING PROBLEM, $\gamma > 0$, $p_{\max} \in \mathbb{N}$.

**Output:** A convex combination $\sum_{b \in B_c} x_{c,b}\, b$ for all $c \in \mathcal{C}$.

1: Set $\text{price}(r) := 1$ for all $r \in \mathcal{R}$.
2: Set $X_c := 0$ for all $c \in \mathcal{C}$ and $x_{c,b} := 0$ for all $c \in \mathcal{C}$ and $b \in B_c$.
3: **for** $p = 1$ **to** $p_{\max}$ **do**
4:  **while** there exists $c \in \mathcal{C}$ with $X_c < p$ **do**
5:  Let $c \in \mathcal{C}$ with $X_c < p$.
6:  Set $b := f_c(\text{price})$.
7:  Set $\xi := \min\left\{p - X_c,\ \min\{1 \,/\, \text{usg}_{c,r}(b) : \text{usg}_{c,r}(b) > 0, r \in \mathcal{R}\}\right\}$.
8:  Set $x_{c,b} := x_{c,b} + \xi$, $X_c := X_c + \xi$.
9:  **for all** $r \in \mathcal{R}$ **do**
10:  $\text{price}(r) := \text{price}(r) \cdot e^{\gamma \xi \text{usg}_{c,r}(b)}$.
11: Set $x_{c,b} := x_{c,b} \,/\, p_{\max}$ for all $c \in \mathcal{C}$ and $b \in B_c$.

In order to solve the MIN-MAX RESOURCE SHARING PROBLEM we can apply the approximation algorithm of Müller, Radke and Vygen [87], which is given as Algorithm 1. This yields the following result:

THEOREM 4.2 (Müller, Radke, Vygen [87]). *One can solve the* MIN-MAX RESOURCE SHARING PROBLEM *with approximation ratio* $\sigma(1+\omega)$ *for any* $\omega > 0$ *in* $\mathcal{O}(\theta(|\mathcal{C}| + |\mathcal{R}|) \log |\mathcal{R}| (\log\log |\mathcal{R}| + \omega^{-2}))$ *time. Here,* $\sigma \geq 1$ *is a constant bounding the approximation ratio of the oracle functions and* $\theta$ *is the time for an oracle call. If* $\frac{1}{2} \leq \lambda^* \leq 2$, *the running time reduces to* $\mathcal{O}(\theta(|\mathcal{C}| + |\mathcal{R}|)\omega^{-2} \log |\mathcal{R}|)$.

The parameters $\gamma$ and $p_{\max}$ are chosen depending on the given instance and the desired approximation guarantee. The iterations of the outer for-loop from line 3 to 10 are

called *phases*. In each phase of the algorithm the oracle is called at least once for every customer with the current prices as input (line 6).[1] If the solution $b$ returned by the oracle does not overuse any resource, then $b$ is taken with a coefficient of $p - X_c$, and the customer is not processed in the same phase again. If it overuses a resource, then the coefficient can be smaller, and it may be necessary to process the customer in the same phase again (line 7). The prices of used resources are then increased by a multiplicative update rule (line 10). In that sense, the price of a resource depends exponentially on its total usage. After the specified number of phases has been executed, the algorithm returns a convex combination of solutions for every customer (line 11). As the blocks $B_c$, $c \in \mathcal{C}$, are convex, this is a feasible solution for the Min-Max Resource Sharing Problem.

However, depending on the application, the convex combination returned by Algorithm 1 might not be a feasible solution for the original problem that was modeled as a Min-Max Resource Sharing Problem. For instance, in the case of the Traditional Global Routing Problem, Algorithm 1 would return a convex combination of Steiner trees for every net, which is not a feasible solution for the Traditional Global Routing Problem. In that case, *randomized rounding* can be applied:

Theorem 4.3 (Müller, Radke, Vygen [87]). *Consider an instance of the* Min-Max Resource Sharing Problem*, and for each $c \in \mathcal{C}$ let $(x_{c,b})_{b \in B_c}$ be non-negative numbers with $\sum_{b \in B_c} x_{c,b} = 1$. Let $\lambda := \max_{r \in \mathcal{R}} \sum_{c \in \mathcal{C}} \sum_{b \in B_c} x_{c,b} \cdot \mathrm{usg}_{c,r}(b)$.*

*Consider a randomly rounded solution that is obtained by picking a solution $b \in B_c$ for every $c \in \mathcal{C}$ with probability $x_{c,b}$. For each $c \in \mathcal{C}$ let $b(c)$ be the randomly picked solution and $\tilde{\lambda} := \max_{r \in \mathcal{R}} \sum_{c \in \mathcal{C}} \mathrm{usg}_{c,r}(b(c))$. For $r \in \mathcal{R}$ let $\rho_r := \max\{\mathrm{usg}_{c,r}(b) \,/\, \lambda : c \in \mathcal{C}, b \in B_c, x_{c,b} > 0\}$, and let $\delta > 0$. Then $\tilde{\lambda} \leq \lambda(1+\delta)$ with probability at least $1 - \sum_{r \in \mathcal{R}} e^{-g(\delta)/\rho_r}$, where $g(\delta) := (1+\delta)\ln(1+\delta) - \delta$.*

This result is taken from Müller, Radke and Vygen [87], and it uses results from Raghavan and Thompson [92].

In practice, the rounded result is usually significantly worse than the fractional solution. To overcome this problem, one can try to restore the quality of the fractional solution heuristically. In our implementation in the global routing context, this routine is called *Rechoose and Reroute*. It is called after randomized rounding and can pick other solutions from the pool of fractional solutions or reroute nets if no adequate solution is available. Moreover, other approaches for performing the rounding step exist. A comparison of different rounding techniques is given by Bihler [10].

[1] To save running time, we use a mechanism in our implementation to skip the computation of a solution for a customer if the solution from the previous phase is already "good enough". For simplicity, we neglect that in our description of the algorithm.

Due to its generality, the resource sharing framework can be used to model various constraints and objectives in the context of global routing. In the Traditional Global Routing Problem, the objective of minimizing weighted wire length can be modeled by adding an additional resource that represents total weighted wire length and is consumed by every net. Here, one would need to set up a reasonable capacity for this resource in advance (which is then scaled to 1 by adapting the usage functions). This capacity could be obtained by using binary search or taking a good guess. It is also possible to use heuristic methods that adapt the capacity of this resource during the course of the algorithm, as outlined by Müller [86] (Section 4.7.1), as running the global router several times during a binary search is not desirable in practice. Moreover, Müller [85], Vygen [117] and Müller, Radke and Vygen [87] show how to model power consumption and yield as objectives. A further advancement here is the incorporation of timing constraints, which is shown in Section 4.3.

## 4.3. Incorporating Timing Constraints

In this section we show how to incorporate timing constraints into the resource sharing framework from Section 4.2. This section is built upon the concepts introduced in Chapter 3. In particular, our algorithm is working on the timing graph $D$ from Section 3.2, and the timing constraints that are modeled here are equivalent to the ones introduced in Section 3.3. As already mentioned in the introduction of this chapter, the incorporation of timing constraints into the resource sharing framework is joint work with Stephan Held, Dirk Müller, Daniel Rotter, Vera Traub and Jens Vygen [53].

**4.3.1. The Timing Model.** Our new approach builds on the modeling of the Traditional Global Routing Problem as a Min-Max Resource Sharing Problem from Section 4.2, but adds new resources and customers to model timing constraints. Let $D$ be the timing graph from Section 3.2 and $b = (b(N))_{N \in \mathcal{N}}$ be a routing solution, i.e. $b(N) \in B_N$ for $N \in \mathcal{N}$. For $(v, w) \in E(D)$ let $d_b(v, w)$ denote the signal delay along $(v, w)$ determined by $b$. Here, we assume that $d_b(v, w)$ only depends on the net $N(w)$, and therefore also write $d_{b(N(w))}(v, w) := d_b(v, w)$. This is true for our definition of the Elmore delay model from Section 3.4, which we will use throughout this chapter. However, the timing model presented here also works for other delay models, e.g. the linear delay model from [53] that is useful for unbuffered netlists. In more elaborate delay models, e.g. ones that incorporate slew effects, the delay along an edge $(v, w) \in E(D)$ might depend on the routing of multiple nets. Depending on the model and the desired accuracy, this could be incorporated more or less easily.

More precisely, in order to fit into our model, every function $b \mapsto d_b(v, w)$ must be positive, separable (i.e. $d_b(v, w) = \sum_{N \in \mathcal{N}} g_N(b(N))$ for some non-negative functions

$(g_N)_{N\in\mathcal{N}}$) and convex. Convexity can be achieved by defining the blocks $B_N$, $N \in \mathcal{N}$, appropriately: For $N \in \mathcal{N}$, let $\mathcal{Y}_N$ be the set of all Steiner trees for $N$, and let $B_N := \{\beta \in [0,1]^{\mathcal{Y}_N} : \sum_{Y\in\mathcal{Y}_N} \beta_Y = 1\}$. For $(v,w) \in E(D)$ and $\beta \in B_{N(w)}$ we can now set

$$d_b(v,w) := \sum_{Y\in\mathcal{Y}_{N(w)}} \beta_Y d_Y(w),$$

where $d_Y(w)$ is the Elmore delay from the source pin of $N(w)$ to $w$ as in Definition 3.1. This way, the functions $b \mapsto d_b(v,w)$ are linear and therefore convex for all $(v,w) \in E(D)$. By the same reasoning, all usage functions $\mathrm{usg}_{N,r} : B_N \to \mathbb{R}_{\geq 0}$ for $(N,r) \in \mathcal{N} \times \mathcal{R}$ are convex. We note here that the running time of Algorithm 1 does not depend on the dimension or the size of the blocks $B_N$. Therefore, the above definition of the blocks does not have a negative impact on the running time of the algorithm.
Given that, it is easy to show that the timing constraints from Section 3.3 are fulfilled by $b$ if and only if there exists numbers $a(v), v \in V(D)$, such that

$$a(v) = \mathrm{at}(v) \qquad \text{for all } v \in V_{\mathrm{in}}, \tag{4.1}$$

$$a(v) = \mathrm{rat}(v) \qquad \text{for all } v \in V_{\mathrm{out}}, \tag{4.2}$$

$$a(v) + d_b(v,w) \leq a(w) \qquad \text{for all } (v,w) \in E(D). \tag{4.3}$$

Here, the numbers $a(v)$, $v \in V(D)$, are interpreted as *arrival times* and will be variables throughout our algorithm. We fix them to $a(v) = \mathrm{at}(v)$ for $v \in V_{\mathrm{in}}$, but let the resource sharing algorithm choose $a(v)$ for $v \in (V_{\mathrm{gate}} \cup V_{\mathrm{out}})$. Choosing $a(v) > \mathrm{rat}(v)$ for $v \in V_{\mathrm{out}}$ constitutes a violation of (4.2), but we still allow it at high costs and call it a *timing relaxation*. The reason for allowing this is that timing closure is often impossible to achieve, in particular when the design is in an early stage and timing assertions are immature. In that case, the goal is usually to produce a feasible global routing (i.e. one obeying routing capacity constraints) and optimize timing as best as possible. Our algorithm can achieve that by relaxing timing constraints where necessary.
The rest of this section is structured as follows: In Section 4.3.2 we deal with the computation of lower and upper bounds for the arrival times $a(v), v \in V(D)$, given delay lower and upper bounds for the edges in $E(D)$. These arrival time bounds are then used in Section 4.3.3, where we show how the timing constraints (4.1) – (4.3) can be integrated into the resource sharing framework by adding new resources and customers. Section 4.3.4 then presents an oracle for our new customers, and we conclude this section by defining our delay lower and upper bounds on $E(D)$ in Section 4.3.5.

**4.3.2. Lower and Upper Bounds on Arrival Times.** We start by showing how to compute reasonable arrival time intervals. As the arrival times are variables in our algorithm, restricting the choice to a reasonable interval will improve convergence. For

this, we need delay lower and upper bounds $d_{\text{lb}}\colon E(D) \to \mathbb{R}_{\geq 0}$ and $d_{\text{ub}}\colon E(D) \to \mathbb{R}_{\geq 0}$ such that $0 < d_{\text{lb}}(v,w) \leq d_{\text{ub}}(v,w)$ for all $(v,w) \in E(D)$. We will show in Section 4.3.5 how to define these delay lower and upper bounds.

We start by propagating arrival times with respect to $d_{\text{lb}}$ in topological order through $D$: Let $a^{\rightarrow}_{\text{lb}}(v) := \text{at}(v)$ for $v \in V_{\text{in}}$ and

$$a^{\rightarrow}_{\text{lb}}(v) := \max\left\{a^{\rightarrow}_{\text{lb}}(u) + d_{\text{lb}}(u,v) : (u,v) \in \delta^{-}_{D}(v)\right\}$$

for $v \in V_{\text{gate}} \cup V_{\text{out}}$. As timing closure is often not achievable, we will often not meet the required arrival times at $V_{\text{out}}$ even when lower bound delays are used. If that is the case, we compute a maximum timing relaxation value

$$\text{relax} := \max\left\{0, \max\{a^{\rightarrow}_{\text{lb}}(v) - \text{rat}(v) : v \in V_{\text{out}}\}\right\}$$

to allow relaxation of timing constraints at high costs (cf. Section 4.3.3.2). With this we can propagate relaxed required arrival times with respect to $d_{\text{lb}}$ in reverse topological order through $D$: We set $a^{\leftarrow}_{\text{lb}}(v) := \text{rat}(v) + \text{relax}$ for $v \in V_{\text{out}}$ and

$$a^{\leftarrow}_{\text{lb}}(v) := \min\left\{a^{\leftarrow}_{\text{lb}}(w) - d_{\text{lb}}(v,w) : (v,w) \in \delta^{+}_{D}(v)\right\}$$

for $v \in V_{\text{gate}} \cup V_{\text{in}}$.[2] Due to our choice of relax we get the following proposition:

PROPOSITION 4.4. *$a^{\rightarrow}_{\text{lb}}(v) \leq a^{\leftarrow}_{\text{lb}}(v)$ holds for all $v \in V(D)$.* □

The interval $[a^{\rightarrow}_{\text{lb}}(v), a^{\leftarrow}_{\text{lb}}(v)]$ can be large if $v$ is not part of a timing-critical path. In order to reduce the interval size in such cases, we also propagate arrival times and required arrival times with respect to our delay upper bounds $d_{\text{ub}}$ through $D$. Let $a^{\rightarrow}_{\text{ub}}(v) := \text{at}(v)$ for $v \in V_{\text{in}}$ and

$$a^{\rightarrow}_{\text{ub}}(v) := \max\left\{a^{\rightarrow}_{\text{ub}}(u) + d_{\text{ub}}(u,v) : (u,v) \in \delta^{-}_{D}(v)\right\}$$

for $v \in V_{\text{gate}} \cup V_{\text{out}}$. On the other hand, set $a^{\leftarrow}_{\text{ub}}(v) := \text{rat}(v)$ for $v \in V_{\text{out}}$ and

$$a^{\leftarrow}_{\text{ub}}(v) := \min\left\{a^{\leftarrow}_{\text{ub}}(w) - d_{\text{ub}}(v,w) : (v,w) \in \delta^{+}_{D}(v)\right\}$$

for $v \in V_{\text{gate}} \cup V_{\text{in}}$. We note here that $a^{\leftarrow}_{\text{lb}}$ and $a^{\leftarrow}_{\text{ub}}$ are initialized differently at $V_{\text{out}}$. The reason will become clear shortly. We can formulate the following proposition:

PROPOSITION 4.5. *$a^{\rightarrow}_{\text{lb}}(v) \leq a^{\rightarrow}_{\text{ub}}(v)$ and $a^{\leftarrow}_{\text{ub}}(v) \leq a^{\leftarrow}_{\text{lb}}(v)$ holds for all $v \in V(D)$.* □

We can make the following statements for any vertex $v \in V(D)$:

(i) It is pointless to choose $a(v)$ to be smaller than $a^{\rightarrow}_{\text{lb}}(v)$, as the signal arrival time at $v$ cannot be less than $a^{\rightarrow}_{\text{lb}}(v)$.

[2] As a memory aid: The arrow notation denotes the propagation order, i.e. the right arrow means "propagation in topological order of $D$", while the left arrow means "propagation in reverse topological order of $D$".

(ii) It is pointless to choose $a(v)$ to be smaller than $a^{\leftarrow}_{\mathrm{ub}}(v)$, as every signal from $v$ to $V_{\mathrm{out}}$ will arrive in time if $a(v)$ is set to $a^{\leftarrow}_{\mathrm{ub}}(v)$.

(iii) It is pointless to choose $a(v)$ to be larger than $a^{\rightarrow}_{\mathrm{ub}}(v)$, as the signal arrival time at $v$ will not exceed $a^{\rightarrow}_{\mathrm{ub}}(v)$.

(iv) If we have a signal arrival time at $v$ that is larger than $a^{\leftarrow}_{\mathrm{lb}}(v)$, then we will certainly violate a relaxed required arrival time at some timing endpoint in $V_{\mathrm{out}}$ that is reachable from $v$.

Using these observations we can now define our arrival time intervals $[a_{\min}(v), a_{\max}(v)]$ for $v \in V(D)$. First consider the case $a^{\leftarrow}_{\mathrm{ub}}(v) \leq a^{\rightarrow}_{\mathrm{ub}}(v)$. Then we set

$$a_{\min}(v) := \max\{a^{\rightarrow}_{\mathrm{lb}}(v), a^{\leftarrow}_{\mathrm{ub}}(v)\}, \tag{4.4}$$

$$a_{\max}(v) := \min\{a^{\leftarrow}_{\mathrm{lb}}(v), a^{\rightarrow}_{\mathrm{ub}}(v)\}, \tag{4.5}$$

where (4.4) is due to statements (i) and (ii) and (4.5) due to statements (iii) and (iv). Note that with the assumption $a^{\leftarrow}_{\mathrm{ub}}(v) \leq a^{\rightarrow}_{\mathrm{ub}}(v)$ and Propositions 4.4 and 4.5 we get that the interval $[a_{\min}(v), a_{\max}(v)]$ is non-empty. However, $a^{\rightarrow}_{\mathrm{ub}}(v) < a^{\leftarrow}_{\mathrm{ub}}(v)$ is also possible. In that case we call $v$ *uncritical*, as even with upper bound delays the arrival time is smaller than the required arrival time. In that case we fix $a(v)$ by setting

$$a_{\min}(v) := a_{\max}(v) := \frac{1}{2}(a^{\rightarrow}_{\mathrm{ub}}(v) + a^{\leftarrow}_{\mathrm{ub}}(v)).$$

Figure 4.1 gives an example of our arrival time interval computations. We can show that inequality (4.3) is fulfilled for any edge $(v, w) \in E(D)$ where $v$ or $w$ is uncritical:

PROPOSITION 4.6. *$a_{\max}(v) + d_{\mathrm{ub}}(v, w) \leq a_{\min}(w)$ holds for any $(v, w) \in E(D)$ where $v$ or $w$ is uncritical.*

PROOF. Case 1: Only $v$ is uncritical. Then $a_{\max}(v) \leq a^{\leftarrow}_{\mathrm{ub}}(v) \leq a^{\leftarrow}_{\mathrm{ub}}(w) - d_{\mathrm{ub}}(v, w) \leq a_{\min}(w) - d_{\mathrm{ub}}(v, w)$.
Case 2: Only $w$ is uncritical. Then $a_{\max}(v) + d_{\mathrm{ub}}(v, w) \leq a^{\rightarrow}_{\mathrm{ub}}(v) + d_{\mathrm{ub}}(v, w) \leq a^{\rightarrow}_{\mathrm{ub}}(w) \leq a_{\min}(w)$.
Case 3: $v$ and $w$ are uncritical. Then $a_{\max}(v) + d_{\mathrm{ub}}(v, w) = \frac{1}{2}(a^{\rightarrow}_{\mathrm{ub}}(v) + a^{\leftarrow}_{\mathrm{ub}}(v)) + d_{\mathrm{ub}}(v, w) \leq \frac{1}{2}(a^{\rightarrow}_{\mathrm{ub}}(w) + a^{\leftarrow}_{\mathrm{ub}}(w)) = a_{\min}(w)$. □

We state another simple proposition that we will later need in Section 4.3.3:

PROPOSITION 4.7. *$a_{\min}(v) < a_{\max}(w)$ holds for any $(v, w) \in E(D)$.*

PROOF. Let $(v, w) \in E(D)$. We note that we assumed $0 < d_{\mathrm{lb}}(v, w) \leq d_{\mathrm{ub}}(v, w)$ in the beginning of this section. If $v$ or $w$ is uncritical, then the claim follows directly from Proposition 4.6. Otherwise, we have $a^{\rightarrow}_{\mathrm{lb}}(v) < a^{\rightarrow}_{\mathrm{lb}}(v) + d_{\mathrm{lb}}(v, w) \leq a^{\rightarrow}_{\mathrm{lb}}(w)$ and $a^{\leftarrow}_{\mathrm{ub}}(v) \leq a^{\leftarrow}_{\mathrm{ub}}(w) - d_{\mathrm{ub}}(v, w) < a^{\leftarrow}_{\mathrm{ub}}(w)$, which results in $a_{\min}(v) < a_{\min}(w) \leq a_{\max}(w)$. □

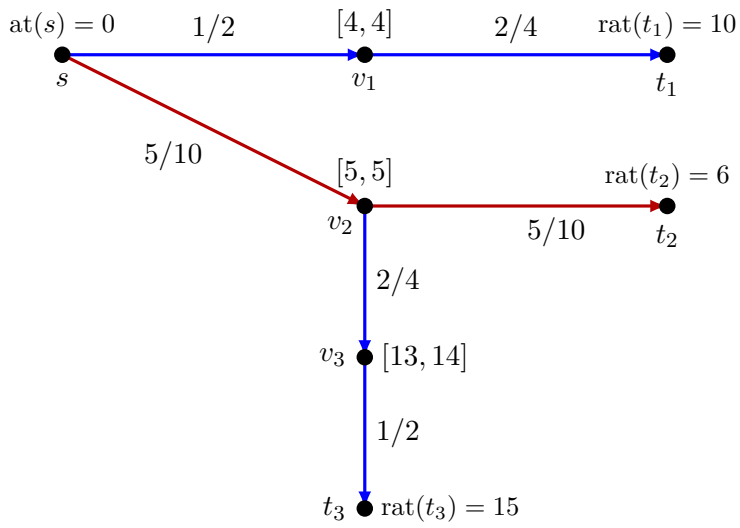



FIGURE 4.1. Illustration of the arrival time interval computations from Section 4.3.2: We have $V_{\text{in}} = \{s\}$ and $V_{\text{out}} = \{t_1, t_2, t_3\}$, and the edge labels denote lower and upper bound delays. $v_1$ is uncritical, and its arrival time is therefore fixed to 4. $v_2$ on the other hand is located on the critical path from $s$ to $t_2$ (in red), which will result in a timing violation even if lower bound delays are assumed: We have $a^{\rightarrow}_{\text{lb}}(t_2) = 10$ and $\text{rat}(t_2) = 6$, which results in relax $= 4$ and an arrival time interval of $[5, 5]$ at $v_2$ determined by $a^{\rightarrow}_{\text{lb}}(v_2)$ and $a^{\leftarrow}_{\text{lb}}(v_2)$ in (4.4) and (4.5). Lastly, $v_3$ is not uncritical in the sense of Section 4.3.2, but its arrival time bounds are still determined by $a^{\leftarrow}_{\text{ub}}(v_3)$ and $a^{\rightarrow}_{\text{ub}}(v_3)$.

**4.3.3. Modeling Timing Constraints with New Resources and Customers.** In this section we will extend the resource sharing model by adding new resources and customers that model timing constraints. We introduce delay resources and arrival time customers in Section 4.3.3.1, refine our timing model through timing relaxation resources in Section 4.3.3.2 and justify the model by stating a central result in Section 4.3.3.3.

4.3.3.1. *Delay Resources and Arrival Time Customers.* The timing constraints (4.3) are incorporated into the resource sharing framework in the following way: For every edge of $D$ we add a new *delay resource* to $\mathcal{R}$, which is used to express inequality (4.3). Moreover, for every $v \in V(D)$ we add a new *arrival time customer* to $\mathcal{C}$ with the purpose of choosing an arrival time $a(v)$. Here, we set

$$B_v := [a_{\min}(v), a_{\max}(v)],$$

where $a_{\min}(v)$ and $a_{\max}(v)$ are the arrival time bounds from Section 4.3.2. The arrival times will be fixed for some vertices in $V(D)$, but for notational purposes we still add

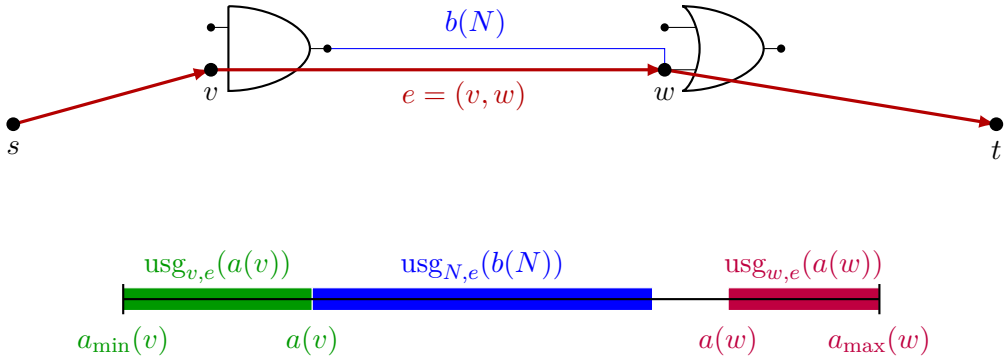



FIGURE 4.2. Illustration of the consumption of timing resources as declared in (4.6): In the upper half of the picture we see a small section of a timing graph $D$ consisting of a path from $s \in V_{\text{in}}$ to $t \in V_{\text{out}}$ with two inner vertices $v, w \in V_{\text{gate}}$. The lower half then shows how the delay capacity of $(v, w) \in E(D)$ is used up: $v$ and $w$ consume an amount that equals the difference between $a(v)$ and $a_{\min}(v)$ and $a(w)$ and $a_{\max}(w)$, and $N = N(w)$ consumes an amount of $d_{b(N)}(v, w)$. In this case, the relative usage of $(v, w)$ is 90%. The remaining 10% could be used for choosing a later arrival time at $v$, an earlier at $w$, or allowing a detour when routing $N$.

those vertices as customers with degenerate arrival time intervals containing only one point.

The delay resource $e = (v, w) \in E(D)$ is consumed by the customers $v$, $w$ and $N := N(w)$ in the following way: Let $a(v) \in B_v$, $a(w) \in B_w$ and $b(N) \in B_N$. Then usages are defined as

$$\begin{aligned} \text{usg}_{v,e}(a(v)) &:= \frac{a(v) - a_{\min}(v)}{a_{\max}(w) - a_{\min}(v)}, \\ \text{usg}_{w,e}(a(w)) &:= \frac{a_{\max}(w) - a(w)}{a_{\max}(w) - a_{\min}(v)}, \\ \text{usg}_{N,e}(b(N)) &:= \frac{d_{b(N)}(v, w)}{a_{\max}(w) - a_{\min}(v)}, \end{aligned} \tag{4.6}$$

with $\text{usg}_{c,e}$ being constantly zero for all $c \in \mathcal{C} \setminus \{N, v, w\}$. One can think of the delay resource $(v, w)$ as having a *delay capacity* of $a_{\max}(w) - a_{\min}(v)$ with the usage functions normalizing this to a unit capacity. By Proposition 4.7, the denominator in (4.6) is always positive. An illustration of these usage functions is given by Figure 4.2.

4.3.3.2. *Timing Relaxation Resources.* In Section 4.3.2 we introduced the relaxation value relax, which is used to relax required arrival times at vertices in $V_{\text{out}}$. To control this relaxation, we introduce a *timing relaxation resource* $r_v \in \mathcal{R}$ for every $v \in V_{\text{out}}$,

which is only consumed by the customer $v$. Given $v \in V_{\text{out}}$ and a parameter $\rho(v) > 0$, the usage function for $r_v$ is then given by

$$\text{usg}_{v,r_v} := \begin{cases} 1 + \rho(v)\dfrac{a(v) - a_{\min}(v)}{a_{\max}(v) - a_{\min}(v)} & \text{if } a_{\min}(v) \neq a_{\max}(v), \\ 1 & \text{otherwise,} \end{cases}$$

with $\text{usg}_{c,r_v}$ being constantly zero for all customers $c \in \mathcal{C} \setminus \{v\}$.
The interpretation here is as follows: If timing constraints are not relaxed at $v$, which corresponds to choosing $a(v) = a_{\min}(v)$, then the usage of $r_v$ is exactly 1. On the other hand, choosing arrival times $a(v) > a_{\min}(v)$ results in an increased usage of $r_v$ up to a maximum usage of $1 + \rho(v)$ when $a(v) = a_{\max}(v)$ is chosen. This way, any timing relaxation will result in a violation of a resource capacity constraint in the resource sharing algorithm, and the magnitude of such violations can be directly controlled by $\rho$. We note here that we take $a_{\min}(v)$ as a reference point for having no timing relaxation, which can already be larger than $\text{rat}(v)$ if $\text{rat}(v)$ cannot even be met with lower bound delays. However, in that case $a_{\min}(v)$ is the best arrival time that we can aim for. In our implementation we use the global setting $\rho(v) = 1.25$ for all $v \in V_{\text{out}}$.

4.3.3.3. *Justifying the Model.* We now state a theorem that forms the foundation of our approach:

THEOREM 4.8. *Let $(a(v))_{v \in V(D)}$ be an arrival time solution and $(b(N))_{N \in \mathcal{N}}$ be a routing. Then $(a, b)$ meets all timing constraints (4.1) – (4.3) if and only if $a^{\rightarrow}_{\text{lb}}(v) \leq \text{rat}(v)$ and $\text{usg}_{v,r_v}(a(v)) = 1$ for all $v \in V_{\text{out}}$ and*

$$\text{usg}_{v,e}(a(v)) + \text{usg}_{w,e}(a(w)) + \text{usg}_{N(w),e}(b(N(w)) \;\leq\; 1$$

*for all $e = (v, w) \in E(D)$.*

PROOF. (4.1) is trivially fulfilled since arrival times are fixed for all $v \in V_{\text{in}}$. $a^{\rightarrow}_{\text{lb}}(v) \leq \text{rat}(v)$ for all $v \in V_{\text{out}}$ is clearly necessary for meeting all required arrival times. If $a^{\rightarrow}_{\text{lb}}(v) \leq \text{rat}(v)$ for $v \in V_{\text{out}}$, then we have $a_{\min}(v) = \text{rat}(v)$, and hence $\text{usg}_{v,r_v}(a(v)) = 1$ is equivalent to $a(v) = \text{rat}(v)$, i.e. (4.2). Let $e = (v, w) \in E(D)$. Then

$$\begin{aligned} & \text{usg}_{v,e}(a(v)) + \text{usg}_{w,e}(a(w)) + \text{usg}_{N(w),e}(b(N(w)) \;\leq\; 1 \\ \iff \quad & \frac{a(v) - a_{\min}(v) + a_{\max}(w) - a(w) + d_{b(N(w))}(v, w)}{a_{\max}(w) - a_{\min}(v)} \;\leq\; 1, \end{aligned}$$

and multiplying both sides by $a_{\max}(w) - a_{\min}(v)$ shows that this is equivalent to $a(v) + d_{b(N(w))}(v, w) \leq a(w)$, i.e. (4.3). □

In particular this means that we can solve the corresponding instance of the Min-Max Resource Sharing Problem, and a solution where all delay and timing relaxation resources have a usage of at most 1 will fulfill all timing constraints.

**4.3.4. The Arrival Time Oracle.** As stated in Section 4.2, we need an oracle for our arrival time customers. For notational convenience, we will only describe the oracle for the vertices in $V_{\text{gate}}$. However, it is easy to see that all the results also apply to vertices in $V_{\text{out}}$: Given a vertex $v \in V_{\text{out}}$ with $a_{\min}(v) \neq a_{\max}(v)$ and a price $\text{price}(r_v)$, we have

$$\begin{aligned}\text{price}(r_v) \cdot \text{usg}_{v,r_v}(a(v)) &= \text{price}(r_v) \cdot \left(1 + \rho(v) \frac{a(v) - a_{\min}(v)}{a_{\max}(v) - a_{\min}(v)}\right) \\ &= \text{price}(r_v) + \text{price}(r_v) \cdot \rho(v) \cdot \frac{a(v) - a_{\min}(v)}{a_{\max}(v) - a_{\min}(v)}\end{aligned}$$

for any $a(v) \in B_v$. Therefore, the oracle can treat $r_v$ as if it were a delay resource corresponding to an edge $(v, w)$ with $a_{\max}(w) := a_{\max}(v)$ and $\text{price}(v, w) := \text{price}(r_v) \cdot \rho(v)$.
Providing an optimum oracle for arrival time customers is actually very simple:

Lemma 4.9. *Given $v \in V_{\text{gate}}$ and resource prices $\text{price}\colon E(D) \to \mathbb{R}_{\geq 0}$, one can compute an arrival time $a(v) \in [a_{\min}(v), a_{\max}(v)]$ minimizing $\sum_{e \in E(D)} \text{price}(e) \cdot \text{usg}_{v,e}(a(v))$ in $\mathcal{O}(|\delta_D(v)|)$ time.*

Proof. For any $t \in [a_{\min}(v), a_{\max}(v)]$ we have

$$\begin{aligned}\sum_{e \in E(D)} \text{price}(e) \cdot \text{usg}_{v,e}(t) = &\sum_{(u,v) \in \delta_D^-(v)} \text{price}(u, v) \frac{a_{\max}(v) - t}{a_{\max}(v) - a_{\min}(u)} \\ &+ \sum_{(v,w) \in \delta_D^+(v)} \text{price}(v, w) \frac{t - a_{\min}(v)}{a_{\max}(w) - a_{\min}(v)}.\end{aligned}$$

Since this is a linear function, its minimum is attained at one of the interval borders. As the evaluation of this function takes $\mathcal{O}(|\delta_D(v)|)$ time, we get the result. □

This oracle is optimal and fast, but it is not very stable in the sense that the chosen arrival times may bounce between the interval borders from iteration to iteration. An easy method to get a better behavior in practice is to call the above oracle multiple times in each resource sharing phase, which results in Algorithm 2. A deeper look at the proof from [87] reveals that the performance guarantee of Algorithm 1 still holds when Algorithm 2 is used as oracle. The following lemma shows that Algorithm 2 converges:

**Algorithm 2** Iterated Arrival Time Oracle

**Input:** $v \in V_{\text{gate}}$, $n \in \mathbb{N}$.

**Output:** An arrival time $a(v) \in B_v$.

1: **for** $i = 1$ **to** $n$ **do**
2: Compute $a_i(v) \in [a_{\min}(v), a_{\max}(v)]$ minimizing $\sum_{e \in E(D)} \text{price}(e) \cdot \text{usg}_{v,e}(a_i(v))$.
3: $\text{price}(e) := \text{price}(e) \cdot e^{(\gamma/n)\,\text{usg}_{v,e}(a_i(v))}$ for all $e \in E(D)$.
4: **return** $a(v) := \frac{1}{n} \sum_{i=1}^{n} a_i(v)$.

LEMMA 4.10. *Let $v \in V_{\text{gate}}$. Then for $n \to \infty$ the output of Algorithm 2 converges to* $\min\{\max\{a_{\min}(v), t^*\}, a_{\max}(v)\}$, *where $t^*$ is the unique root of the function*

$$g(t) = \sum_{(v,w) \in \delta_D^+(v)} \frac{\text{price}(v,w)}{a_{\max}(w) - a_{\min}(v)} \cdot e^{\gamma \frac{t - a_{\min}(v)}{a_{\max}(w) - a_{\min}(v)}} - \sum_{(u,v) \in \delta_D^-(v)} \frac{\text{price}(u,v)}{a_{\max}(v) - a_{\min}(u)} \cdot e^{\gamma \frac{a_{\max}(v) - t}{a_{\max}(v) - a_{\min}(u)}}.$$

For the proof the reader is referred to [53]. Basically, $t^*$ represents the arrival time with the property that the prices per time unit of incoming and outgoing edges are balanced at $v$ after the price update in line 10 of Algorithm 1.
We can now avoid running Algorithm 2 explicitly and instead approximate $t^*$ by Newton's method, which has a global quadratic convergence rate in our scenario:

THEOREM 4.11. *We can approximate the limit of the output of Algorithm 2 for $n \to \infty$ up to accuracy $\delta > 0$ in running time $\mathcal{O}(|\delta_D(v)| \cdot \log\log \frac{a_{\max}(v) - a_{\min}(v)}{\delta})$ for any $v \in V_{\text{gate}}$.*

Again, the proof can be found in [53]. In practice, we perform a small constant number of Newton steps, which usually gives good results.

**4.3.5. Lower and Upper Bounds for Delays.** In this section we will describe how to choose the delay bounds $d_{\text{lb}}$ and $d_{\text{ub}}$ that are needed for the arrival time interval computations from Section 4.3.2. As already stated, we use the Elmore delay model from Section 3.4 for measuring signal delays. Let $(v,w) \in E(D)$, $N := N(w)$, and $s \in N$ and $T \subseteq N$ be the source and sink pins of $N$, respectively. Note that $w \in T$.
For defining a delay lower bound $d_{\text{lb}}(v,w)$ we let $Y$ be an approximately shortest Steiner tree for $N(w)$, which can be computed using the methods from Section 2.3. Moreover, let $R_{\min}$ and $C_{\min}$ denote the minimum wire resistance and capacitance per unit length

over all allowed routing layers for $N(w)$, respectively. We define

$$d_{\text{lb}}(v,w) := R(s) \cdot \left( C_{\min} \sum_{(x,y)\in E(Y)} \text{dist}(x,y) + C(T) \right) + R_{\min} \cdot \text{dist}(s,w) \cdot \left( \tfrac{1}{2} C_{\min} \cdot \text{dist}(s,w) + C(w) \right).$$

This is a lower bound if $Y$ is a shortest Steiner tree for $N(w)$.
For defining a delay upper bound $d_{\text{ub}}(v,w)$ we again compute an approximately shortest Steiner tree $Y$ for $N(w)$, and we assume that all wiring edges of $Y$ are located on the lowest allowed wiring layers for $N(w)$, as these usually have the least favorable electrical properties when it comes to signal delay. We compute wire resistances and capacitances for $Y$ as described in Definition 3.2, but multiply them by a parameter $\zeta \geq 1$ in order to compute the *$\zeta$-scaled Elmore delay* $d_Y^\zeta(s,w)$ from $s$ to $w$ as in Definition 3.1. We then set $d_{\text{ub}}(v,w) := d_Y^\zeta(s,w) + \eta$, where $\eta$ is a second parameter. In our experiments, we set $\zeta$ to 1.5 and $\eta$ to 0.25 picoseconds.
The reasoning here is as follows: Multiplying all wire and via RC values by $\zeta$ is roughly the same as multiplying all edge lengths by $\zeta$, so $d_Y^\zeta(s,w)$ corresponds to the Elmore delay from $s$ to $w$ in $Y$ where we plan for a detour of factor $\zeta$ on every edge in $Y$ (and a few additional vias). $\eta$ is a constant that is added to give some margin for very short nets. We expect our router to make bigger detours only in very few cases, so $d_{\text{ub}}$ should indeed be a reasonable upper bound in practice. The detour histogram from Figure 5.3 clearly supports this claim.

**4.3.6. Summarizing the Model.** Lastly, we give a short summary of our model with respect to the formulation of the Min-Max Resource Sharing Problem from Section 4.2. Letting $G$ denote the global routing graph, we have $\mathcal{R} = E(G) \cup E(D) \cup \bigcup_{v\in V_{\text{out}}} \{r_v\}$ and $\mathcal{C} = \mathcal{N} \cup V(D)$. The blocks $B_N$, $N \in \mathcal{N}$, are defined in Section 4.3.1, while we use the arrival time bounds from Section 4.3.2 for defining the blocks $B_v$, $v \in V(D)$. The usage functions $\text{usg}_{N,e}$ for $(N,e) \in \mathcal{N} \times E(G)$ are defined as for the Traditional Global Routing Problem (cf. Section 4.2), while we define the usage functions for delay and timing relaxation resources in Section 4.3.3. It is easy to see that all usage functions are non-negative and convex and therefore meet the requirements of the Min-Max Resource Sharing Problem. Theorem 4.8 shows that the model is actually sound, and an oracle for arrival time customers is given in Section 4.3.4. Therefore, the only missing piece is the oracle for net customers, which must be able to minimize a weighted sum of congestion and timing prices. This oracle is described in Chapter 5. Given all that, we can run Algorithm 1 to obtain the result from Theorem 4.2. As we can provide an optimum oracle for arrival time customers, $\sigma$ from Theorem 4.2 is determined solely by the oracle for net customers. Applying randomized rounding as in

Theorem 4.3 and a heuristic called Rechoose and Reroute at the end, we can obtain a global routing from the fractional solution provided by Algorithm 1. Here, we note that we do not have to apply randomized rounding for arrival times, as a convex combination of arrival times for a customer is again a feasible arrival time for this customer.

CHAPTER 5

# The RC-Aware Routing Oracle

In this chapter we are going to present our RC-aware routing oracle, which fits into the resource sharing framework from Chapter 4 by functioning as oracle for net customers. To this end, the task for our RC-aware routing oracle consists of computing Steiner trees minimizing a weighted sum of congestion and timing prices, where the weights are generated by the resource sharing framework from Chapter 4. As for the rest of this thesis, signal delays are measured using the Elmore delay model from Section 3.4. This chapter is structured as follows: We start with a formal definition of the RC-Aware Steiner Tree Problem in Section 5.1 and continue with an overview on previous work in Section 5.2. We then first examine the RC-Aware Steiner Tree Problem for two-terminal nets in Section 5.3, before we approach the general version in Section 5.4. Finally, we conclude this chapter by presenting experimental results in Section 5.5. Parts of the materials presented in this chapter are already published in [**53**, **107**, **108**].

## 5.1. Problem Formulation

We start by giving a formal definition of the RC-Aware Steiner Tree Problem. In this chapter we are working exclusively with projected pin shapes (cf. Definition 2.9). Connecting to exact shapes is then being done using the techniques from Chapter 6. Moreover, as always throughout this thesis, we do not optimize wire types, but assume a fixed wire type to be given for every net. The RC-Aware Steiner Tree Problem can then be formulated as follows:

PROBLEM 5.1: RC-AWARE STEINER TREE PROBLEM

**Input:** A net $N$ with source $s \in N$ and sinks $T \subseteq N$, a source resistance $R(s) \in \mathbb{R}_{\geq 0}$, sink capacitances $C\colon T \to \mathbb{R}_{\geq 0}$, timing prices $\text{price}\colon T \to \mathbb{R}_{\geq 0}$, the global routing graph $G$ with layers $Z$, wire resistances and capacitances $R_{\text{wire}}, C_{\text{wire}}\colon Z \to \mathbb{R}_{\geq 0}$, via resistances $\text{R}_{\text{via}}\colon Z \to \mathbb{R}_{\geq 0}$, congestion prices $\text{price}\colon E(G) \to \mathbb{R}_{\geq 0}$.

**Task:** Find a Steiner tree $Y$ for $N$ minimizing

$$\text{price}(Y) := \text{price}(E(Y)) + \sum_{t \in T} \text{price}(t) \cdot d_Y(t),$$

where $d_Y(t)$ for $t \in T$ is the Elmore delay from $s$ to $t$ as defined by Definitions 3.1 and 3.2.

This problem formulation indeed meets the needs of the resource sharing framework from Chapter 4: There, the oracle for a net customer $N$ is required to find a Steiner tree $Y$ approximately minimizing

$$\begin{aligned} &\sum_{e \in E(G)} \text{price}(e) \cdot \text{usg}_{N,e}(Y) + \sum_{t \in T} \sum_{f=(u,t) \in \delta_D^-(t)} \text{price}(f) \cdot \text{usg}_{N,f}(Y) \\ = &\sum_{e \in E(Y)} \text{price}(e) \cdot \text{usg}(N,e) + \sum_{t \in T} \sum_{f=(u,t) \in \delta_D^-(t)} \frac{\text{price}(f)}{a_{\max}(t) - a_{\min}(u)} \cdot d_Y(t), \end{aligned} \tag{5.1}$$

where we can use the routing space usage $\text{usg}(N,e)$ for $e \in E(G)$ as in the TRADITIONAL GLOBAL ROUTING PROBLEM from Section 2.2 (also using Definition 2.4 assuming $N$ to be fixed). Clearly, minimizing (5.1) is equivalent to minimizing the objective function of the RC-AWARE STEINER TREE PROBLEM given an appropriate translation of the prices. As it contains the RECTILINEAR MINIMUM STEINER TREE PROBLEM (cf. Section 2.3.2), it follows that the RC-AWARE STEINER TREE PROBLEM is $NP$-hard. Further hardness results follow in Sections 5.3.1 and 5.4.1.

In our implementation, we use additional prices for wire length and vias, which can just be added to the prices on $E(G)$ without changing the semantics of the RC-AWARE STEINER TREE PROBLEM. In any case, for the rest of this chapter it will be sufficient to work directly with the formulation of the RC-AWARE STEINER TREE PROBLEM without caring about how the prices are generated.

## 5.2. Previous Work

Although it is a natural formulation for the problem of constructing Steiner trees minimizing Elmore delay during global routing, only few results regarding this problem are known: Hähnle and Rotter [97] show that the RC-AWARE STEINER TREE PROBLEM is

$NP$-hard even if $|T| = 1$. As we can show in [106], the general case with an arbitrary number of sinks is even $NP$-hard to approximate within $o(\log |T|)$. As both these works use problem formulations that are slightly different from ours, we transfer these results to our model in Sections 5.3.1 and 5.4.1. Moreover, Hähnle and Rotter [97] are also able to state fully polynomial time approximation schemes for the special case $|T| = 1$ and for the case where $|T|$ is arbitrary, but the topology of the Steiner tree is fixed. We elaborate more on their approximation schemes in Section 5.3.2.

On the other hand, the planar version of the RC-Aware Steiner Tree Problem with unit wire resistances and capacitances and without congestion prices has received quite some attention in the past. The rest of this section will be dealing with results concerning this problem: Boese et al. show in [18] that for the variant minimizing the weighted sum of source-sink delays there is always an optimum solution using only Steiner points on the Hanan grid.[1] Therefore, they can solve the problem in exponential time. They also give an example in [17] showing that the existence of optimum solutions on the Hanan grid is generally not given for the variant minimizing maximum source-sink delay. Kadodi [68] and Peyer [90] show how to solve the problem of minimizing maximum source-sink delay for instances with at most three sinks optimally in constant time. For larger terminal sets, various heuristics have been implemented and evaluated in practice, but no performance bounds are proven [17, 18, 19, 116]. Moreover, there is work by Cong et al. [29] on optimizing a simplification of the Elmore delay formula, which seems to be easier to optimize and yields an upper bound for the actual Elmore delay. Peyer et al. [91] give heuristics for improving the Elmore delay of a given rectilinear Steiner tree without increasing its length. The first constant-factor approximation algorithm is given by us in [108]. A more extensive summary of (early) results is presented by Kahng and Robins [69].

## 5.3. RC-Aware Paths

In this section we consider the RC-Aware Path Problem, which is the special case of the RC-Aware Steiner Tree Problem where $T$ consists of a single sink $t$. The difficulty here is that in contrast to the general Shortest Path Problem (see e.g. [75]), the price of the path in the RC-Aware Path Problem cannot be expressed as the sum of the prices of its edges. In fact, this leads to the RC-Aware Path Problem being $NP$-hard, as is shown by Hähnle and Rotter [97] (Theorem 4.6). They actually consider the RC-Aware Path Problem in graphs with arbitrary resistance and capacitance functions on the edges, but it is easy to extend their hardness result to our model. For

[1] The Hanan grid is the grid that is induced by the set of $x$- and $y$-coordinates of all terminals – see Hanan [46].

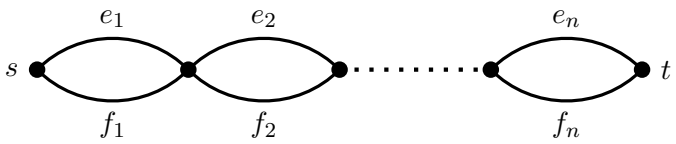



FIGURE 5.1. Illustration of the instance of the RC-AWARE PATH PROBLEM from the proof of Theorem 5.3. The edges $e_i$ and $f_i$, $i = 1, \ldots, n$, are on different layers of the global routing graph.

the sake of completeness, we restate their proof in Section 5.3.1. In Section 5.3.2 we then present our RC-aware path search, which gives strong approximation guarantees for the RC-AWARE PATH PROBLEM in practice. Parts of the results from Section 5.3.2 are published in [53, 107].

**5.3.1. NP-Hardness of the RC-Aware Path Problem.** To prove $NP$-hardness of the RC-AWARE PATH PROBLEM, Hähnle and Rotter [97] use a reduction from the PARTITION PROBLEM, which is known to be $NP$-complete [70]:

PROBLEM 5.2: PARTITION PROBLEM

**Input:** Numbers $a_1, \ldots, a_n \in \mathbb{Q}_{>0}$ with $\sum_{i=1}^{n} a_i = 2$.

**Task:** Decide whether there exists $I \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in I} a_i = 1$.

Demanding $\sum_{i=1}^{n} a_i = 2$ is unusual when formulating the PARTITION PROBLEM, but it does not change the problem. However, it will be convenient for the proof of Theorem 5.3:

THEOREM 5.3 (Hähnle and Rotter [97] (Theorem 4.6)). *The* RC-AWARE PATH PROBLEM *is $NP$-hard even if $n_y = 1$ in the definition of the global routing graph (cf. Definition 2.8).*

PROOF. Let $a_1, \ldots, a_n$ be an instance of the PARTITION PROBLEM. We construct an instance of the RC-AWARE PATH PROBLEM as illustrated in Figure 5.1: Let $n_x = n+1$, $n_y = 1$ and $n_z = 2$ in the definition of the global routing graph $G$, and let $s$ and $t$ be located in the tiles with minimum and maximum $x$-coordinate, respectively. The wiring edges on layers 1 and 2 are denoted by $e_1, \ldots, e_n$ and $f_1, \ldots, f_n$, respectively, where higher indices imply higher $x$-coordinates. Prices, resistances and capacitances are then set as follows:

- $R(s) = 0$, $C(t) = 0$, $\text{price}(t) = 1$,
- $\text{price}(e_i) = 2a_i$, $R(e_i) = 0$, $C(e_i) = 0$, $i = 1, \ldots, n$,
- $\text{price}(f_i) = 0$, $R(f_i) = a_i$, $C(f_i) = 2a_i$, $i = 1, \ldots, n$,
- $\text{price}(e) = R(e) = 0$ for all via edges $e \in E(G)$.

Clearly, these settings of $R$ and $C$ can be achieved by setting $R_{\text{wire}}(1) = C_{\text{wire}}(1) = 0$ and $R_{\text{wire}}(2) = 1$, $C_{\text{wire}}(2) = 2$, and choosing appropriate lengths for the edges in $G$. Given an $s$-$t$ path $Y$ embedded into $G$, we set $I := \{i \in \{1, \ldots, n\} : f_i \in p(E(Y))\} = \{i_1, \ldots, i_k\}$ for indices $i_1 < \ldots < i_k$. Moreover, define $I^c := \{1, \ldots, n\} \setminus I$ and $a(I') := \sum_{i \in I'} a_i$ for any $I' \subseteq \{1, \ldots, n\}$ for a more convenient notation. Then we can write

$$\begin{aligned}\text{price}(Y) &= \sum_{i \in I^c} 2a_i + \sum_{j=1}^{k} a_{i_j} \left( a_{i_j} + \sum_{l=j+1}^{k} 2a_{i_l} \right) \\ &= 2(2 - a(I)) + a(I)^2 \\ &= \Big(a(I) - 1\Big)^2 + 3,\end{aligned}$$

where we used $a(\{1, \ldots, n\}) = 2$ for the second equality. As the deduction of $I$ from $Y$ above defines a bijection between $s$-$t$ paths embedded into $G$ and subsets of $\{1, \ldots, n\}$, it follows that there exists an optimum $s$-$t$ path $Y$ in $G$ with $\text{price}(Y) = 3$ if and only if $a_1, \ldots, a_n$ define a yes-instance of the PARTITION PROBLEM. □

The restriction $n_y = 1$ in Theorem 5.3 seems to be of minor importance, but we use it later for proving $NP$-hardness of a different problem in Theorem 6.7.

**5.3.2. Approximating the RC-Aware Path Problem.** Due to the hardness result from Section 5.3.1, the best we can hope for is an approximation algorithm for the RC-AWARE PATH PROBLEM. In fact, as already mentioned in Section 5.2, Hähnle and Rotter [97] are able to devise a fully polynomial time approximation scheme for it. They start their path search at $t$ and, similarly to the well-known algorithm of Dijkstra [34], create labels for vertices in $G$ until a label for $s$ is created whose price is approximately minimum. In contrast to Dijkstra's algorithm, however, they can create multiple labels for any $v \in V(G)$: In their algorithm, a label is a pair $(v, i) \in V(G) \times (\mathbb{Z} \cup \{-\infty\})$ associated with a downstream capacitance $C(v, i)$. Here, $(v, i)$ corresponds to a $v$-$t$ path with price at most $(1 + \delta)^i$ for some $\delta > 0$ (with $(1 + \delta)^{-\infty} := 0$) and a capacitance that is at most $C(v, i)$. Given such a label $(v, i)$, it can then be propagated and create other labels $(u, j) \in \Gamma_G(v) \times (\mathbb{Z} \cup \{-\infty\})$ — as the downstream capacitance corresponding to $(v, i)$ is known, one can calculate the Elmore delay along $(u, v)$ in the path $P$ defined by adding $(u, v)$ to the path corresponding to $(v, i)$. If $P$ then defines an $u$-$t$ path for its price bucket $j$ that has a lower capacitance than the current value of $C(u, j)$, then $P$ is used to represent label $(u, j)$.

By choosing $\delta$ appropriately, this algorithm yields an $(1+\varepsilon)$-approximation for the RC-Aware Path Problem in

$$\mathcal{O}\left(n \frac{\log(\text{price}_{\text{ub}} / \text{price}_{\text{lb}})}{\varepsilon} \left(m + n \log\left(n \frac{\log(\text{price}_{\text{ub}} / \text{price}_{\text{lb}})}{\varepsilon}\right)\right)\right) \tag{5.2}$$

time, where $n := |V(G)|$, $m := |E(G)|$, $\text{price}_{\text{ub}} > 0$ is an upper bound for the price of any $s$-$t$ path in $G$, $\text{price}_{\text{lb}} > 0$ is a lower bound for the price of any $s$-$t$ path in $G$ with positive price, and we assume that we can compute logarithms to any base in constant time (Theorem 4.10 in [97]).

Using similar techniques, Hähnle and Rotter also state a fully polynomial time approximation scheme for embedding a fixed topology for a multi-sink net $N$ into $G$: Given $\varepsilon > 0$, their algorithm computes an $(1+\varepsilon)$-approximation for the problem of embedding the given fixed topology and runs in

$$\mathcal{O}\left(|N|^2 n \frac{\log(\text{price}_{\text{ub}} / \text{price}_{\text{lb}})}{\varepsilon} \left(\frac{|N|n^2}{\varepsilon} + m + n \log\left(n \frac{\log(\text{price}_{\text{ub}} / \text{price}_{\text{lb}})}{\varepsilon}\right)\right)\right) \tag{5.3}$$

time, where we use an analogous notation and assumption as for (5.2) (Theorem 4.14 in [97]).

Clearly, the results of Hähnle and Rotter answer the question of how well the RC-Aware Path Problem can be approximated in polynomial time. However, looking at the running times (5.2) and (5.3) of their algorithms, it becomes evident that in our application where millions of nets have to be routed in a graph that can contain millions of vertices, the fully polynomial time approximation schemes of Hähnle and Rotter are most likely too slow to be used for more than a small fraction of the nets, e.g. the most critical ones with respect to timing.

Therefore, we devise a different algorithm for the RC-Aware Path Problem in this section. It runs fast in theory and practice, and we will see that it offers strong approximation guarantees in our application. The idea is to define edge prices $\text{rcp}\colon E(G) \to \mathbb{R}_{\geq 0}$, called *RC prices*, and then find a shortest $s$-$t$ path with respect to these prices (e.g. using Dijkstra's algorithm [34]). If the RC prices are chosen properly to approximate congestion and timing prices that accrue from using the respective edges, then this results in strong approximation bounds in practice.

We will see our RC prices once more in this thesis in Section 6.3, where a layer assignment algorithm minimizing congestion and timing prices is presented. Therefore, there are strong similarities between this section and Section 6.3. However, there are also some differences, and in order to keep both descriptions as simple as possible, we give two standalone-descriptions for both sections, although this produces some amount of overlap. In Section 5.3.2.1 we present our RC-aware path search for approximating

the RC-Aware Path Problem, and we then analyze the given performance bounds in Section 5.3.2.2.

5.3.2.1. *RC-Aware Path Search.* We start with a definition:

Definition 5.4. Consider an instance of the RC-Aware Path Problem. We define

$$R^{\min}_{\text{wire}} := \min_{z \in Z} R_{\text{wire}}(z), \qquad C^{\min}_{\text{wire}} := \min_{z \in Z} C_{\text{wire}}(z),$$

to be the *minimum wire resistance* and *minimum wire capacitance* per unit length, respectively. Moreover, we define the *minimum upstream resistance* and *minimum downstream capacitance* of $v \in V(G)$ as

$$R^{\text{up}}_{\min}(v) := R(s) + \mathrm{R}_{\text{via}}(z_s, z_v) + R^{\min}_{\text{wire}} \cdot \operatorname{dist}(s, v),$$
$$C^{\text{down}}_{\min}(v) := C^{\min}_{\text{wire}} \cdot \operatorname{dist}(v, t) + C(t),$$

respectively, where $z_s$ and $z_v$ denote the layers of $s$ and $v$. Given an $s$-$t$ path $Y$, the actual *upstream resistance* of $v \in V(Y)$ is defined as

$$R^{\text{up}}_Y(v) := R(s) + R(E(P_Y(s, v))).$$

The downstream capacitance is already defined in Definition 3.1, as it is widely used throughout this thesis. It is easy to see that $R^{\text{up}}_{\min}(v)$ and $C^{\text{down}}_{\min}(v)$ are lower bounds for the upstream resistance and downstream capacitance of $v \in V(G)$. An alternative way to define $R^{\text{up}}_{\min}$ for $v \in V(G)$ would be to set

$$R^{\text{up}}_{\min}(v) := R(s) + \min_{z \in Z} \Big( R_{\text{wire}}(z) \cdot \operatorname{dist}(s, v) + \mathrm{R}_{\text{via}}(z_s, z) + \mathrm{R}_{\text{via}}(z, z_v) \Big), \tag{5.4}$$

noting that in order to use a layer $z \in Z$, we must first reach it from $z_s$ and later reach $z_v$. A better bound can be obtained by taking the preference directions of routing layers (cf. Section 1.3.1) into account, as during global routing, wires orthogonal to the preference direction of the given layer are usually not permitted: In this context, identifying our resistances with costs, Henke [56] shows that for any $v \in V(G)$ there always exists a minimum resistance $s$-$v$ path that for some $n_i \in \mathbb{Z}_{\geq 0}, i = 1, \ldots, 5$, is a sequence of

(a) $n_1$ via edges, $n_2$ wiring edges in $x$-direction, $n_3$ via edges, $n_4$ wiring edges in $y$-direction and $n_5$ via edges, or
(b) $n_1$ via edges, $n_2$ wiring edges in $y$-direction, $n_3$ via edges, $n_4$ wiring edges in $x$-direction and $n_5$ via edges.

Henke also states an algorithm with running time $\mathcal{O}(|Z|)$ that computes such an optimum sequence. Setting $R^{\text{up}}_{\min}$ as in (5.4) or as described in [56] gives a better estimate, but the advantage of setting it as in Definition 5.4 is that it can be evaluated in constant time. We can now continue to define other quantities that we need throughout this section:

DEFINITION 5.5. Consider an instance of the RC-AWARE PATH PROBLEM. For $(v, w) \in E(G)$ we set

$$\mathrm{cp}_{\mathrm{lb}}(v, w) := \mathrm{price}(t) \cdot R^{\mathrm{up}}_{\mathrm{min}}(v)$$

to be the *lower bound capacitance price* of $(v, w)$. Given an $s$-$t$ path $Y$ and $(v, w) \in E(Y)$, we define

$$\mathrm{cp}(v, w) := \mathrm{price}(t) \cdot R^{\mathrm{up}}_{Y}(v)$$

as the *capacitance price* of $(v, w)$, and

$$C_{\mathrm{corr}}(v, w) := C(v, w) + C^{\mathrm{min}}_{\mathrm{wire}} \cdot \Big(\mathrm{dist}(w, t) - \mathrm{dist}(v, t)\Big)$$

as the *corrective capacitance* of $(v, w)$.

We note that neither $\mathrm{cp}_{\mathrm{lb}}(v, w)$ nor $\mathrm{cp}(v, w)$ depend on $w$, but semantically we nevertheless treat them as edge properties. We can now define our RC prices:

DEFINITION 5.6. Consider an instance of the RC-AWARE PATH PROBLEM. For $(v, w) \in E(G)$ we set

$$\begin{aligned}
\mathrm{rcp}_{\mathrm{wire}}(v, w) &:= \mathrm{price}(t) \cdot R(v, w) \cdot \left(\frac{C(v, w)}{2} + C^{\mathrm{down}}_{\mathrm{min}}(w)\right), \\
\mathrm{rcp}_{\mathrm{corr}}(v, w) &:= \mathrm{cp}_{\mathrm{lb}}(v, w) \cdot C_{\mathrm{corr}}(v, w), \\
\mathrm{rcp}(v, w) &:= \mathrm{price}(v, w) + \mathrm{rcp}_{\mathrm{wire}}(v, w) + \mathrm{rcp}_{\mathrm{corr}}(v, w).
\end{aligned}$$

For an $s$-$t$ path $Y$ we set

$$\mathrm{rcp}(Y) := \mathrm{price}(t) \cdot R(s) \cdot C^{\mathrm{down}}_{\mathrm{min}}(s) + \mathrm{rcp}(E(Y)).$$

We will show that a path minimizing rcp is a good approximation for a path minimizing price. The idea is that when we are labeling an edge $(v, w) \in E(G)$ during our path search, we assume the minimum possible downstream capacitance for $w$ to estimate the delay along $(v, w)$. This introduces an error, but the error is mitigated by the correction term $\mathrm{rcp}_{\mathrm{corr}}$. This idea is illustrated by Figure 5.2. With that, we can define our main theorem of this section. To simplify notation, we define $\frac{0}{0} := 1$ for the rest of this section.

THEOREM 5.7. *Consider an instance of the* RC-AWARE PATH PROBLEM. *Let $Y$ be an $s$-$t$ path minimizing* rcp *and $Y^*$ be an $s$-$t$ path minimizing* price. *Then we have*

$$\mathrm{price}(Y) \leq \Big(1 + (\alpha - 1)(1 - \beta)\Big)\mathrm{price}(Y^*),$$

*where*

$$\alpha := \max_{v \in V(Y)} \Big(C(Y(v)) \,/\, C^{\mathrm{down}}_{\mathrm{min}}(v)\Big), \qquad \beta := \min_{v \in V(Y)} \Big(R^{\mathrm{up}}_{\mathrm{min}}(v) \,/\, R^{\mathrm{up}}_{Y}(v)\Big).$$

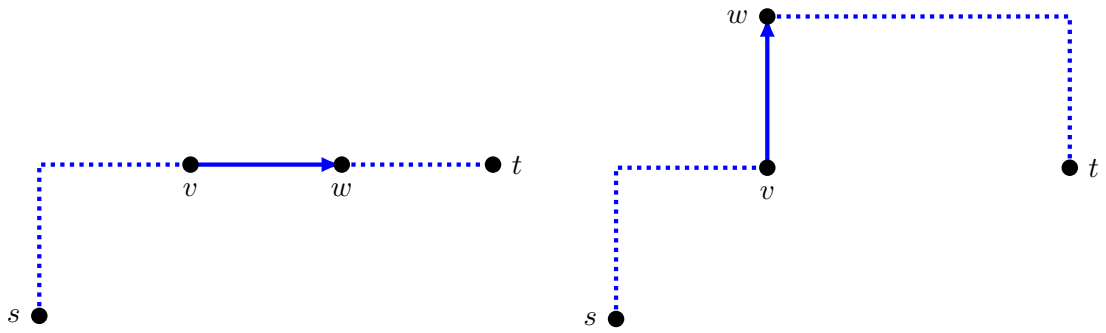



FIGURE 5.2. When labeling an edge $(v, w)$ during our RC-aware path search, we assume the rest of the path to be optimum, i.e. as short as possible (dotted lines) and using the best possible layers. Deviations from these assumptions are partially corrected by $\mathrm{rcp}_{\mathrm{corr}}(v, w)$: When labeling towards the target (left picture), we have $C_{\mathrm{corr}} = C(v, w) - C^{\mathrm{min}}_{\mathrm{wire}} \cdot \mathrm{dist}(v, w)$, i.e. we only correct the capacitance difference that stems from possibly not choosing the layer with minimum capacitance. When labeling away from the target (right picture), we have $C_{\mathrm{corr}} = C(v, w) + C^{\mathrm{min}}_{\mathrm{wire}} \cdot \mathrm{dist}(v, w)$, i.e. we additionally correct the detour of length $2\mathrm{dist}(v, w)$ that is introduced by using $(v, w)$. Of course, the accuracy of $\mathrm{rcp}_{\mathrm{corr}}(v, w)$ is determined by the accuracy of $\mathrm{cp}_{\mathrm{lb}}(v, w)$ compared to $\mathrm{cp}(v, w)$ (cf. Lemma 5.9).

Before giving the proof of Theorem 5.7, we state a few lemmas. We start with a simple identity involving corrective capacitances:

LEMMA 5.8. *Consider an instance of the* RC-AWARE PATH PROBLEM *and let $Y$ be an $s$-$t$ path. Then we have $C(Y(v)) = C^{\mathrm{down}}_{\mathrm{min}}(v) + C_{\mathrm{corr}}(E(Y(v)))$ for any $v \in V(Y)$.*

PROOF. We have

$$\begin{aligned} C(Y(v)) &= C(E(Y(v))) + C(t) + C^{\mathrm{min}}_{\mathrm{wire}} \cdot \mathrm{dist}(v, t) - C^{\mathrm{min}}_{\mathrm{wire}} \cdot \mathrm{dist}(v, t) \\ &= C^{\mathrm{down}}_{\mathrm{min}}(v) + C(E(Y(v))) - C^{\mathrm{min}}_{\mathrm{wire}} \cdot \mathrm{dist}(v, t) \\ &= C^{\mathrm{down}}_{\mathrm{min}}(v) + \sum_{(x,y) \in E(Y(v))} C(x, y) + C^{\mathrm{min}}_{\mathrm{wire}} \Big( \mathrm{dist}(y, t) - \mathrm{dist}(x, t) \Big) \\ &= C^{\mathrm{down}}_{\mathrm{min}}(v) + C_{\mathrm{corr}}(E(Y(v))). \end{aligned}$$

□

We continue with evaluating the difference between rcp and price:

LEMMA 5.9. *Consider an instance of the* RC-AWARE PATH PROBLEM *and let $Y$ be an $s$-$t$ path. Then we have*

$$\text{price}(Y) = \text{rcp}(Y) + \sum_{e \in E(Y)} \Big(\text{cp}(e) - \text{cp}_{\text{lb}}(e)\Big) \cdot C_{\text{corr}}(e).$$

PROOF. For the wire delay price of $Y$ we have

$$\begin{aligned}
&\text{price}(t) \sum_{(v,w)\in E(Y)} R(v,w) \cdot \left(\frac{C(v,w)}{2} + C(Y(w))\right) \\
&= \text{price}(t) \sum_{(v,w)\in E(Y)} R(v,w) \left(\frac{C(v,w)}{2} + C^{\text{down}}_{\text{min}}(w) + C_{\text{corr}}(E(Y(w)))\right) \\
&= \text{rcp}_{\text{wire}}(E(Y))) + \text{price}(t) \sum_{(v,w)\in E(Y)} R(E(P_Y(s,v))) \cdot C_{\text{corr}}(v,w) \\
&= \text{rcp}_{\text{wire}}(E(Y))) + \sum_{e \in E(Y)} \Big(\text{cp}(e) - \text{price}(t) \cdot R(s)\Big) \cdot C_{\text{corr}}(e),
\end{aligned}$$

where we used Lemma 5.8 to get the first and Lemma 3.3 to get the second equality. As a result we get

$$\begin{aligned}
\text{price}(Y) &= \text{price}(E(Y)) + \text{price}(t) \cdot R(s) \cdot C(Y(s)) + \text{rcp}_{\text{wire}}(E(Y)) \\
&\quad + \sum_{e \in E(Y)} \Big(\text{cp}(e) - \text{price}(t) \cdot R(s)\Big) \cdot C_{\text{corr}}(e) \\
&= \text{price}(E(Y)) + \text{price}(t) \cdot R(s) \cdot C^{\text{down}}_{\text{min}}(s) + \text{rcp}_{\text{wire}}(E(Y)) \\
&\quad + \sum_{e \in E(Y)} \text{cp}(e) \cdot C_{\text{corr}}(e) \\
&= \text{rcp}(Y) + \sum_{e \in E(Y)} \Big(\text{cp}(e) - \text{cp}_{\text{lb}}(e)\Big) \cdot C_{\text{corr}}(e),
\end{aligned}$$

where we again used Lemma 5.8 to decompose $C(Y(s)) = C^{\text{down}}_{\text{min}}(s) + C_{\text{corr}}(E(Y(s)))$ for the second equality. □

This gives us the following corollary:

COROLLARY 5.10. *Consider an instance of the* RC-AWARE PATH PROBLEM *and let $Y$ be an $s$-$t$ path. Then* $\text{rcp}(Y) \leq \text{price}(Y)$ *holds.*

PROOF. This follows directly from Lemma 5.9, as we have $\text{cp}_{\text{lb}}(e) \leq \text{cp}(e)$ and $C_{\text{corr}}(e) \geq 0$ for all $e \in E(Y)$. □

We can now prove Theorem 5.7:

PROOF OF THEOREM 5.7. We will prove

$$\text{price}(Y) \le \Big(1 + (\alpha - 1)(1 - \beta)\Big) \cdot \text{rcp}(Y) \le \Big(1 + (\alpha - 1)(1 - \beta)\Big) \cdot \text{price}(Y^*).$$

The second inequality follows directly from Corollary 5.10, as this gives us $\text{rcp}(Y) \le \text{rcp}(Y^*) \le \text{price}(Y^*)$. For proving the first inequality we have to show

$$\sum_{e \in E(Y)} \Big(\text{cp}(e) - \text{cp}_{\text{lb}}(e)\Big) \cdot C_{\text{corr}}(e) \le (\alpha - 1)(1 - \beta) \cdot \text{rcp}(Y) \tag{5.5}$$

according to Lemma 5.9. We note that we have

$$C(Y(v)) - C^{\text{down}}_{\text{min}}(v) \le (\alpha - 1)\, C^{\text{down}}_{\text{min}}(v) \qquad \forall\, v \in V(Y), \tag{5.6}$$

$$\text{cp}(e) - \text{cp}_{\text{lb}}(e) \le (1 - \beta)\, \text{cp}(e) \qquad \forall\, e \in E(Y), \tag{5.7}$$

by definition of $\alpha$ and $\beta$. Due to (5.7), showing (5.5) reduces to showing

$$\sum_{e \in E(Y)} \text{cp}(e) \cdot C_{\text{corr}}(e) \le (\alpha - 1) \cdot \text{rcp}(Y).$$

This is done by the following calculations with the help of (5.6) and Lemmas 3.3 and 5.8:

$$\begin{aligned}
&\sum_{e \in E(Y)} \text{cp}(e) \cdot C_{\text{corr}}(e) \\
&= \text{price}(t)\Big(R(s) \cdot C_{\text{corr}}(E(Y)) + \sum_{(v,w) \in E(Y)} R(E(P_Y(s,v))) \cdot C_{\text{corr}}(v,w)\Big) \\
&= \text{price}(t)\Big(R(s) \cdot \Big(C(Y(s)) - C^{\text{down}}_{\text{min}}(s)\Big) + \sum_{(v,w) \in E(Y)} R(v,w)\Big(C(Y(w)) - C^{\text{down}}_{\text{min}}(w)\Big)\Big) \\
&\le (\alpha - 1) \cdot \text{price}(t) \cdot \Big(R(s) \cdot C^{\text{down}}_{\text{min}}(s) + \sum_{(v,w) \in E(Y)} R(v,w) \cdot C^{\text{down}}_{\text{min}}(w)\Big) \\
&\le (\alpha - 1) \cdot \Big(\text{price}(t) \cdot R(s) \cdot C^{\text{down}}_{\text{min}}(s) + \text{rcp}_{\text{wire}}(E(Y))\Big) \\
&\le (\alpha - 1) \cdot \text{rcp}(Y).
\end{aligned}$$

□

5.3.2.2. *Analyzing the Performance Bounds.* The following proposition gives an easier to understand bound for $\alpha$:

PROPOSITION 5.11. *Consider an instance of the* RC-AWARE PATH PROBLEM, *and let $Y$ and $\alpha$ be as in Theorem 5.7. Then we have $\alpha \le \gamma\delta$, where*

$$\gamma := \frac{\max_{z \in Z} C_{\text{wire}}(z)}{\min_{z \in Z} C_{\text{wire}}(z)}, \qquad \delta := \max_{v \in V(Y)} \frac{\text{dist}_Y(v,t)}{\text{dist}(v,t)}.$$

| Unit | # Nets | Avg. Detour [%] | No Detour [% paths] | wACE4 [%] | OFtgt [100 pitch$^2$] |
|---|---|---|---|---|---|
| U1 | 77 528 | 0.40 | 99.35 | 88.9 | 6.0 |
| U2 | 79 119 | 0.47 | 99.08 | 87.6 | 0.0 |
| U3 | 100 827 | 0.24 | 99.61 | 86.4 | 0.0 |
| U4 | 111 140 | 0.66 | 98.80 | 89.5 | 7.5 |
| U5 | 119 228 | 0.07 | 99.91 | 81.7 | 0.0 |
| U6 | 254 208 | 1.39 | 97.52 | 88.7 | 29.4 |
| U7 | 276 799 | 0.14 | 99.42 | 83.5 | 0.0 |
| U8 | 1 681 671 | 0.45 | 98.92 | 86.1 | 37.4 |

TABLE 5.1. Path-based detour statistics on our testbed. A more detailed detour histogram for U6 is given by Figure 5.3. The wACE4 and OFtgt metrics are explained in Appendix A.

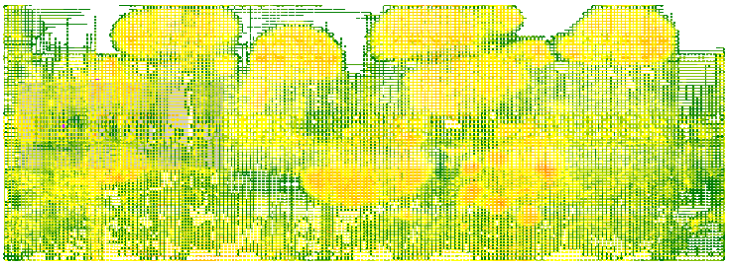
PROOF. We have

$$\begin{aligned}
\alpha &= \max_{v \in V(Y)} \frac{C(Y(v))}{C^{\text{down}}_{\min}(v)} \\
&\leq \max_{v \in V(Y)} \frac{\max_{z \in Z} C_{\text{wire}}(z) \cdot \text{dist}_Y(v,t) + C(t)}{\min_{z \in Z} C_{\text{wire}}(z) \cdot \text{dist}(v,t) + C(t)} \\
&\leq \max_{v \in V(Y)} \frac{\max_{z \in Z} C_{\text{wire}}(z) \cdot \text{dist}_Y(v,t)}{\min_{z \in Z} C_{\text{wire}}(z) \cdot \text{dist}(v,t)} \\
&= \gamma\delta.
\end{aligned}$$

□

$\gamma$ from Proposition 5.11 is a technology-dependent parameter, which is close to 1 in most cases: If the topmost layers with the largest track pitch are excluded, then the wire capacitance per unit length fluctuates by less than 15% for minimum width wires on our 14nm designs. If the topmost layers are included, then differences are still within a factor of two. However, if a net is routed mostly on the topmost layers, then $\beta$ is likely to become a strong bound, which compensates for the error in $\alpha$. Moreover, by their very nature, these layers contain only a relatively small fraction of the total wire length on the chip.

On the other hand, $\delta$ depends on the actual routing result, and therefore appears to be less predictable. However, Table 5.1 shows that on average, $\delta$ is very close to 1: For creating this table we traverse all proper paths in our final global routing result and compare the length of the path to the distance of its endpoints. In this context, a proper



| Detour % | % Paths |
|---|---|
| 0 | 97.52 |
| 0-10 | 0.15 |
| 10-20 | 0.31 |
| 20-30 | 0.37 |
| 30-50 | 0.56 |
| 50-100 | 0.76 |
| > 100 | 0.33 |

FIGURE 5.3. Congestion map and detour histogram of unit U6, which exhibits the most detours in our testbed according to Table 5.1. The congestion target for our run is at 90%, which corresponds to orange edges.

path is a maximum path whose internal vertices (i.e. all vertices excluding the endpoints) are Steiner vertices of degree 2 in the tree containing the path. Moreover, as our RC prices are used to connect projected pin shapes, we also inspect our global routes while they are connecting projected pin shapes, i.e. before using the methods from Chapter 6 to connect to exact shapes. As one can see in Table 5.1 and Figure 5.3, the large majority of paths are shortest paths, and large detours happen rarely. Consequently, $\alpha$ should be very close to 1 in most cases.

In contrast to $\alpha$, $\beta$ will often deviate significantly from 1: For shorter nets, $R^{\text{up}}_{\text{min}}$ might be a good approximation for the actual upstream resistance, as it includes the source resistance, which often dominates for short nets. However, for longer nets that are not routed on the topmost layers, $R^{\text{up}}_{\text{min}}$ becomes a weaker approximation, as the wire resistance per unit length varies significantly across the layer stack: Compared to the lowest layers, the wire resistance per unit length on the topmost layers decreases by more than a factor of 100, with the wire resistance on intermediate layers taking on intermediate values on this spectrum.

The effect of varying wire resistances could be mitigated by artificially restricting the layer range allowed for routing certain nets, but such restrictions are likely to introduce problems on their own. However, even if $\beta$ is close to zero, the approximation bound from Theorem 5.7 is still strong as long as $\alpha$ is not much larger than 1. An inversion of these considerations occurs for nets that are routed mostly on the topmost layers, as we already observed during the analysis of $\gamma$: In that case, $\beta$ should be close to 1, while $\alpha$ might deviate from it. However, the total error $(\alpha - 1)(1 - \beta)$ should still be small. As a result of this analysis, we conclude that Theorem 5.7 provides strong approximation bounds in practice given the current technology parameters.

## 5.4. RC-Aware Steiner Trees

Now that we have presented our approach to compute point-to-point connections in Section 5.3, the next question is how to solve the RC-Aware Steiner Tree Problem for multi-sink nets. In Section 5.4.1, we first recapitulate our result from [106] showing that it is $NP$-hard to approximate the RC-Aware Steiner Tree Problem within approximation guarantee $o(\log |T|)$. We then turn our attention towards our algorithm from [108] in Section 5.4.2, which achieves a constant-factor approximation guarantee for a simpler model without congestion prices and different layer characteristics. Finally, we extend this algorithm to our three-dimensional model in Section 5.4.3. The results from Section 5.4.3 are partly published in [53, 107].

**5.4.1. A Hardness Result.** In this section we are going to present our result from [106] regarding the hardness of the RC-Aware Steiner Tree Problem. As the problem formulation used in [106] is slightly different from the one used in this thesis, we adapt it to our model and restate the proof. We use a reduction from the Minimum Set Cover Problem:

Problem 5.12: Minimum Set Cover Problem

**Input:** A finite, non-empty set $U = \{u_1, \dots, u_n\}$ and a set system $\mathcal{S} = \{S_1, \dots, S_m\}$ with $\bigcup_{i=1}^{m} S_i = U$.

**Task:** Find $I \subseteq \{1, \dots, m\}$ with $\bigcup_{i \in I} S_i = U$ and $|I|$ minimum.

It is shown by Alon et al. [4] and Raz and Safra [96] that there exists a constant $\alpha > 0$ such that it is $NP$-hard to approximate the Minimum Set Cover Problem within approximation guarantee $\alpha \log(n)$. We use this result for the proof of Theorem 5.13:

Theorem 5.13. *There exists $\alpha > 0$ such that it is $NP$-hard to approximate the* RC-Aware Steiner Tree Problem *within approximation guarantee $\alpha \log |T|$.*

Proof. We use a reduction from the Minimum Set Cover Problem. Given an instance $SC = (U, \mathcal{S})$ of the Minimum Set Cover Problem, we construct an instance $RCS$ of the RC-Aware Steiner Tree Problem as illustrated in Figure 5.4: We set $n_x := m + 1$, $n_y := n + 1$ and $z = 3$ in the definition of the global routing graph $G$ (cf. Definition 2.8), and $p((i, j, k)) := (i, j, k)$ for $(i, j, k) \in V(G)$. Moreover, we set $p(s) := (0, 0, 1)$ and $T := \{t_1, \dots, t_n\}$ with $p(t_i) := (0, i, 3)$, $i = 1, \dots, n$, and distinguish the following edge sets in $E(G)$:

- $E_1 = \Big\{\{(i-1, 0, 1), (i, 0, 1)\} : i = 1, \dots, m\Big\}$,
- $E_2 = \Big\{\{(i, j-1, 2), (i, j, 2)\} : i = 1, \dots, m, j = 1, \dots, n\Big\}$,

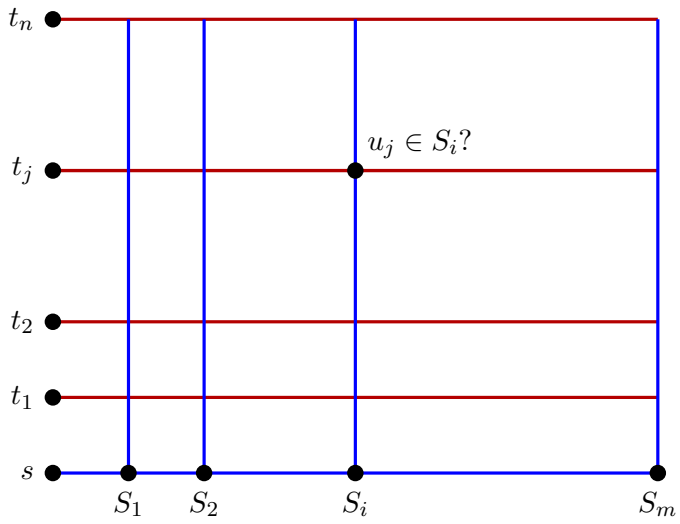



FIGURE 5.4. Instance of the RC-AWARE STEINER TREE PROBLEM used in the proof of Theorem 5.13: Blue edges are located on layers 1 and 2 and have a positive capacitance, but no resistance. Red edges are located on layer 3 and have a large resistance, but no capacitance. Using a via between $(i,0,1)$ and $(i,0,2)$ for $i \in \{1,\dots,m\}$ corresponds to picking set $S_i$, and we can use a via between $(i,j,2)$ and $(i,j,3)$ if and only if $u_j \in S_i$ $(i = 1,\dots,m,\ j = 1,\dots,n)$.

- $E_3 = \Big\{\{(i-1,j,3),(i,j,3)\} : i = 1,\dots,m, j = 1,\dots,n\Big\}$,
- $E_4 = \Big\{\{(i,0,1),(i,0,2)\} : i = 1,\dots,m\Big\}$,
- $E_5 = \Big\{\{(i,j,2),(i,j,3)\} : (i,j) \in \{1,\dots,m\} \times \{1,\dots,n\} \text{ with } u_j \in S_i\Big\}$.

We set $\text{price}(e) = 1$ for $e \in E_4$, $\text{price}(e) = 0$ for $e \in E_1 \cup E_2 \cup E_3 \cup E_5$, and $\text{price}(e) = m+1$ for all other edges in $G$. Moreover, we set $R(s) = 0$, and for $t \in T$ we set $C(t) = 0$ and $\text{price}(t) = 1$. Wire resistances and capacitances are defined by setting $R_{\text{wire}}(3) = m+1$, $C_{\text{wire}}(1) = C_{\text{wire}}(2) = 1$, and setting all resistances and capacitances of wires and vias to 0 otherwise.

We show $\text{OPT}(SC) = \text{OPT}(RCS)$, where OPT denotes the optimum objective function value of the respective instance. To this end, let $I \subseteq \{1,\dots,m\}$ be a solution of $SC$. Constructing a Steiner tree $Y$ for $RCS$ with $\text{price}(Y) \leq |I|$ is straightforward: For each $t_j$, $j = 1,\dots,n$, we choose an edge $e_j = \{(i,j,2),(i,j,3)\} \in E_5$ with $i \in I$ and connect $p(t_j)$ to $p(s)$ by a shortest path $P_j$ over $e_j$ and $\{(i,0,1),(i,0,2)\} \in E_4$. The edge set $\bigcup_{j=1}^{n} E(P_j)$ then defines our Steiner tree $Y$ with $\text{price}(Y) \leq |I|$. This shows $\text{OPT}(RCS) \leq \text{OPT}(SC) \leq m$.

Conversely, let $Y$ be a Steiner tree with $\text{price}(Y) \leq m$. Then for any wiring edge $(v, w) \in E(Y)$ located on layer 3, $Y(w)$ cannot contain any wiring edge that is located on layer 1 or 2, as in that case we would already have $d_Y(t) \geq m+1$ for any $t \in T(Y(w))$. That means that for any $j \in \{1, \ldots, n\}$, $t_j$ must be connected to $s$ in $Y$ by a path containing two vias $\{(i_j, j, 2), (i_j, j, 3)\} \in E_5$ and $\{(i_j, 0, 1), (i_j, 0, 2)\} \in E_4$ for some $i_j \in \{1, \ldots, m\}$ with $u_j \in S_{i_j}$. We can then derive a set cover solution $I := \bigcup_{j=1}^{n} \{i_j\}$ with $|I| \leq \text{price}(Y)$, proving $\text{OPT}(SC) \leq \text{OPT}(RCS)$, and therefore our claim. □

**5.4.2. An Approximation Algorithm for a Simpler Model.** We have seen in Section 5.4.1 that approximating the RC-Aware Steiner Tree Problem within a constant factor is $NP$-hard. In order to still be able to find good solutions for it, we take an indirect route, which involves projecting our problem into a simpler model from [108] as a first step:

Here, we deal with the construction of Steiner trees minimizing Elmore delay in general metric spaces where the resistance and the capacitance of a wire are proportional to its length, and congestion prices are not considered. Transferred to our scenario, this corresponds to the version of the RC-Aware Steiner Tree Problem where we have $\text{price}(e) = 0$ for all $e \in E(G)$, $\text{R}_{\text{via}}(z) = 0$ for all $z \in Z$ and $R_{\text{wire}}(z) = R^*_{\text{wire}}$ and $C_{\text{wire}}(z) = C^*_{\text{wire}}$ for all $z \in Z$, where $R^*_{\text{wire}}, C^*_{\text{wire}} \in \mathbb{R}_{\geq 0}$ are given constants. We call this special case the Simplified RC-Aware Steiner Tree Problem, and we are able to present a constant-factor approximation algorithm for it in [108].

In this section we recapitulate the mechanics behind this algorithm, as we use an extension of it for solving the RC-Aware Steiner Tree Problem for multi-sink nets in our implementation (cf. Section 5.4.3). Here, we do not give any proofs — all relevant proofs can be found in [108]. Our algorithm from [108] is given as Algorithm 3 and illustrated by Figure 5.5.

**Algorithm 3** RC Tree Topology Algorithm

**Input:** An instance of the Simplified RC-Aware Steiner Tree Problem, a Steiner tree $Y_0$ for $N$, $\varepsilon > 0$.
**Output:** A Steiner tree $Y$ for $N$.

1: $Y := Y_0$.
2: **for** all edges $(v, w)$ of $Y_0$ in reverse topological order **do**
3: $\quad C_{\text{bound}}(v, w) := C^*_{\text{wire}} \cdot \frac{\varepsilon}{2} \min\{\text{dist}(s, x) : x \in V(Y(w)) \cup \{v\}\}$.
4: $\quad$ **if** $C(Y(w)) + C(v, w) \geq C_{\text{bound}}(v, w)$ **then**
5: $\quad\quad$ delete $(v, w)$ and reconnect $Y(w)$ to $s$ by a shortest path.
6: **return** Y

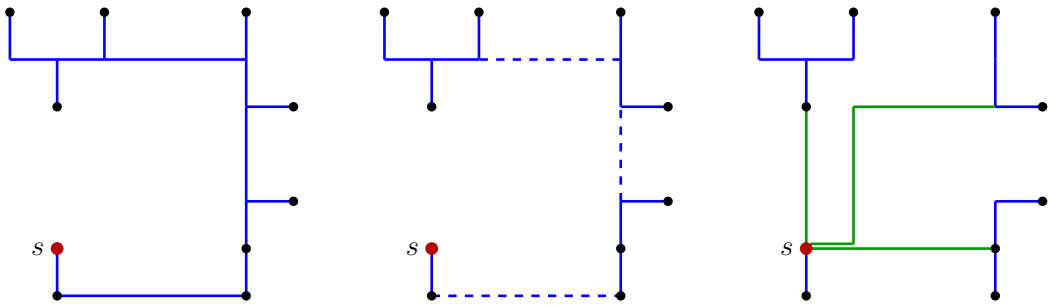



FIGURE 5.5. Schematic progression of Algorithm 3: We start with the initial Steiner tree $Y_0$ on the left. $Y_0$ is short, but the wire delay from $s$ (red dot) to some sinks (black dots) might be large. During the course of the algorithm, the dashed blue edges (middle picture) are deleted from the tree and replaced by the green paths (right picture) by performing reconnects in line 5. This increases the length of the tree, but reduces wire delays. By modulating the value $\varepsilon$, one might end up with fewer or more reconnects.

In addition to the usual instance specifications of the SIMPLIFIED RC-AWARE STEINER TREE PROBLEM, Algorithm 3 gets an initial Steiner tree $Y_0$ and a parameter $\varepsilon > 0$ as input. Here, $Y_0$ is supposed to be as short as possible, as the performance bounds achieved by Algorithm 3 depend on the length of $Y_0$. It is therefore natural to construct $Y_0$ by using an algorithm for the MINIMUM STEINER TREE PROBLEM IN GRAPHS or the RECTILINEAR MINIMUM STEINER TREE PROBLEM (cf. Section 2.3). $\varepsilon$ on the other hand determines a trade-off between minimizing source and wire delay: While minimizing source delay requires the tree to be as short as possible, wire delay is minimized by a star-like topology. In this regard, a large value of $\varepsilon$ will result in large values of $C_{\text{bound}}$ in line 3 of Algorithm 3 and therefore only few reconnects in line 5, keeping $Y$ short and the source delay small. In contrast to that, small values of $\varepsilon$ will result in many reconnects in line 5, which reduces wire delay, but increases source delay. This observation is formalized by the next theorem, which is adopted from [108]:

THEOREM 5.14. *Given an instance of the* SIMPLIFIED RC-AWARE STEINER TREE PROBLEM, *an initial Steiner tree $Y_0$ and $\varepsilon > 0$, Algorithm 3 outputs a tree $Y$ with*

*(1)* $C(Y(s)) \leq \left(1 + \frac{2}{\varepsilon}\right) \cdot C(Y_0(s))$,

*(2)* $\text{wd}_Y(t) \leq \max\left\{(1+\varepsilon)^2,\ 1 + \frac{1}{16}\varepsilon^3 + \frac{3}{4}\varepsilon^2 + 2\varepsilon\right\} \cdot \text{lb}_{\text{wd}}(t)$ *for all* $t \in T$,

*where* $\text{wd}_Y(t)$ *for* $t \in T$ *is the wire delay from* $s$ *to* $t$ *in* $Y$ *as in Definition 3.1, and* $\text{lb}_{\text{wd}}(t) := R^*_{\text{wire}} \cdot \text{dist}(s,t) \cdot \left(\frac{1}{2} C^*_{\text{wire}} \cdot \text{dist}(s,t) + C(t)\right)$ *is a lower bound for the wire delay from* $s$ *to* $t$ *in any Steiner tree for* $N$.

| $\alpha$ | $\beta$ | $\varepsilon$ |
|---|---|---|
| 1.00 | 3.39 | 0.839 |
| 1.39 | 4.11 | 1.025 |
| 1.50 | 4.31 | 1.073 |
| 2.00 | 5.16 | 1.270 |

TABLE 5.2. Approximation ratios $\beta$ of Algorithm 3 depending on the approximation ratio $\alpha$ of the algorithm for the MINIMUM STEINER TREE PROBLEM that is used to construct $Y_0$. The respective choices of $\varepsilon$ are also shown. The choices of $\alpha$ reflect bounds of different algorithms for the MINIMUM STEINER TREE PROBLEM— see Section 2.3 for an overview.

If the length of $Y_0$ is within a factor of $\alpha$ within the optimum length, then condition (1) from Theorem 5.14 tells us that the source delay of $Y$ is at most a factor of $\alpha(1 + 2 / \varepsilon)$ larger than the minimum possible source delay achieved by a shortest Steiner tree. Therefore, by choosing the right values of $\varepsilon$, one can achieve the approximation ratios listed in Table 5.2 depending on the length of $Y_0$. However, in practice, it might of course be better to choose other values of $\varepsilon$ depending on the instance parameters. For more details on the algorithm we refer the reader to [**108**].

**5.4.3. An Extension to Our Model.** As we have learned in Section 5.4.2, Algorithm 3 yields good approximate solutions for the SIMPLIFIED RC-AWARE STEINER TREE PROBLEM. However, these will in general not be good solutions for the RC-AWARE STEINER TREE PROBLEM: Firstly, congestion is ignored, which will almost certainly result in routing overflow. Secondly, uniform wire resistances and capacitances across all layers are assumed, which will often be reasonable with respect to wire capacitances, but very imprecise with respect to wire resistances. Therefore, we need to incorporate congestion prices and different layer characteristics into our model, which is the topic of this section: In Section 5.4.3.1 we explain how the initial tree $Y_0$ for running Algorithm 3 is constructed. Section 5.4.3.2 then describes our way to apply Algorithm 3, and Section 5.4.3.3 shortly describes a post-processing method that is applied before returning the resulting tree.

5.4.3.1. *Constructing the Initial Tree.* For constructing the initial tree $Y_0$ for Algorithm 3 it seems intuitive to compute a tree minimizing congestion prices and wire length or capacitance, as an initial tree minimizing wire length yields the best bounds for the SIMPLIFIED RC-AWARE STEINER TREE PROBLEM from Section 5.4.2. However, in practice, completely neglecting wire resistances during the construction of the initial tree results in insufficient use of the upper layers. We therefore use a heuristic variant

of our RC prices from Section 5.3: To do this, we regard all sinks in $T$ as one super-sink $t^*$ with

$$\text{price}(t^*) := \text{price}(T), \qquad C(t^*) := C(T) + C^{\min}_{\text{wire}} \cdot \sum_{(v,w)\in E(Y_T)} \text{dist}(v,w),$$

where $C^{\min}_{\text{wire}}$ is the minimum wire capacitance as in Definition 5.4, and $Y_T$ is an approximately minimum rectilinear Steiner tree for $T$ (cf. Section 2.3.2). Distance computations to $t^*$ during the computation of our RC prices are performed with regard to $\text{BB}_T := \text{BB}(T) \times Z$, where $\text{BB}(T)$ is the bounding box of $T$ as in Definition 2.15, and $Z$ is the set of layers on the chip. More precisely, this means that we set $\text{dist}(v,t^*) := \min_{a\in \text{BB}_T} \text{dist}(v,a)$ for $v \in V(G)$. For $(v,w) \in E(G)$ we then use modified RC prices $\text{rcp}^\mu(v,w)$ that are defined by

$$\text{rcp}^\mu(v,w) := \text{price}(v,w) + \mu(v,w) \cdot \text{rcp}_{\text{wire}}(v,w) + \text{rcp}^\mu_{\text{corr}}(v,w),$$

where

$$\mu(v,w) := \begin{cases} \lambda & \text{if } p(v) \in BB_T \text{ and } p(w) \in BB_T, \\ 1 & \text{otherwise,} \end{cases}$$

for some $\lambda \in [0,1]$, and

$$\begin{aligned} \text{rcp}^\mu_{\text{corr}}(v,w) &:= \text{cp}^\mu_{\text{lb}}(v,w) \cdot C_{\text{corr}}(v,w), \\ \text{cp}^\mu_{\text{lb}}(v,w) &:= \text{price}(t^*) \cdot R^{\text{up},\,\mu}_{\min}(v), \\ R^{\text{up},\,\mu}_{\min}(v) &:= R(s) + \mu(v,w) \cdot \Big(\text{R}_{\text{via}}(z_s, z_v) + R^{\min}_{\text{wire}} \cdot \text{dist}(s,v)\Big) \\ &\quad + (1-\mu(v,w)) \cdot R^{\min}_{\text{wire}} \cdot \text{dist}(s,t^*), \end{aligned}$$

using the notation and definitions from Section 5.3.2. In our implementation we set $\lambda = 0.1$.

The interpretation is as follows: When labeling an edge $(v,w) \in E(G)$ outside of $\text{BB}_T$, then we are often in the situation where $s$ is not contained in $\text{BB}_T$, and $(v,w)$ is likely to be part of every $s$-$t$ path, $t \in T$. This claim is supported by Table 5.1, which shows that detouring outside of $\text{BB}_T$ should only happen rarely. In that case, the model of considering $T$ as one super-sink $t^*$ is suitable, and we set $\mu(v,w) = 1$, as then $\text{rcp}^\mu$ coincides with rcp. If on the other hand $v$ is contained in $\text{BB}_T$, then it is hard to estimate in advance which sinks will be contained in $Y_0(w)$ (without consideration of the algorithm used to construct $Y_0$), which makes it difficult to come up with a good estimate for wire delay prices. In that case, we scale $\text{rcp}_{\text{wire}}$ by $\lambda$ to still give the router an incentive to use higher layers when $\text{price}(t^*)$ or $C(t^*)$ is high.

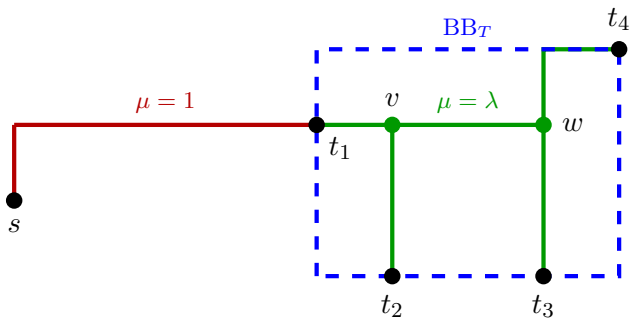



FIGURE 5.6. Illustration of $\mathrm{rcp}^{\mu}$ during the construction of $Y_0$: We see a net with source $s$ and sinks $T = \{t_1, t_2, t_3, t_4\}$ that is connected by the initial tree $Y_0$ consisting of red and green edges. Red edges outside of $\mathrm{BB}_T$ (dashed blue rectangle) drive all sinks in $T$. Therefore, setting $\mu = 1$ is an accurate estimate in this case. For green edges inside of $\mathrm{BB}_T$ we cannot make such a statement: The wire delay along the edge $(v, w)$ only affects $t_3$ and $t_4$, but not $t_1$ and $t_2$. Similarly, the corrective capacitance $C_{\mathrm{corr}}(v, w)$ only influences the path from $s$ to $v$. Here, its influence on the path from $s$ to $t_1$ affects all sinks, but its influence on the path from $t_1$ to $v$ does not affect $t_1$ any more. Before $Y_0$ is constructed, these effects are difficult to assess, as the routing topology inside of $\mathrm{BB}_T$ is unknown and can be hard to predict.

The correction term $\mathrm{rcp}^{\mu}_{\mathrm{corr}}$ is scaled in the same manner: Here, we first notice that for an edge $(v, w) \in E(G)$ outside of $\mathrm{BB}_T$ we have $\mu(v, w) = 1$, and so $\mathrm{rcp}^{\mu}_{\mathrm{corr}}$ coincides with $\mathrm{rcp}_{\mathrm{corr}}$. Otherwise, if $(v, w)$ is inside of $\mathrm{BB}_T$, we notice the simple identity

$$\mu(v,w)\mathrm{dist}(s,v) + (1-\mu(v,w))\mathrm{dist}(s,t^*) = \mu(v,w)(\mathrm{dist}(s,v) - \mathrm{dist}(s,t^*)) + \mathrm{dist}(s,t^*),$$

which allows for a more easily understood explanation of $R^{\mathrm{up},\mu}_{\mathrm{min}}(v)$: Here, we fully consider the upstream resistance $R(s) + R^{\mathrm{min}}_{\mathrm{wire}} \cdot \mathrm{dist}(s, t^*)$, as this resistance is likely to affect all sinks. Analogously to the estimation of wire delay prices, we scale the remaining upstream resistance $\mathrm{R}_{\mathrm{via}}(z_s, z_v) + R^{\mathrm{min}}_{\mathrm{wire}} \cdot (\mathrm{dist}(s, v) - \mathrm{dist}(s, t^*))$ by $\lambda$, as we cannot reliably estimate which sinks will be affected by this resistance. An illustration is given by Figure 5.6.

In our implementation we use the Standard Block Solver described by Müller [86] (Section 4.7.3) for the MINIMUM STEINER TREE PROBLEM IN GRAPHS (cf. Section 2.3.1) in order to construct $Y_0$. If edge costs for such an algorithm for the MINIMUM STEINER TREE PROBLEM IN GRAPHS are to be fixed a priori, a closer inspection of the location of the respective edge in relation to the positions of $s$ and the sinks in $T$ might give more accurate cost estimates. Better results might be obtained if cost estimates are adjusted during the course of the specific algorithm being used to construct $Y_0$.

Mentionable in this regard is an algorithm devised by Heeger [49], who uses a variant of the Dijkstra-Steiner algorithm by Hougardy, Silvanus and Vygen [59] and an extension of our techniques from Section 5.3 for the construction of RC-aware Steiner trees: Given an instance of the RC-Aware Steiner Tree Problem, his algorithm propagates labels $(v, I) \in V(G) \times 2^T$ through the graph, which correspond to Steiner trees connecting $v$ and $I$ with minimum estimated prices and an associated downstream capacitance. When a label $(v, I)$ is processed, it may generate labels $(w, I)$ for $w \in \Gamma_G(v)$ and labels $(v, I \cup J)$ in conjunction with other labels $(v, J)$ for $\emptyset \subsetneq J \subseteq T \setminus I$. Finally, the algorithm returns the Steiner tree corresponding to the label $(s, T)$ at a point where the estimated price of this label cannot be reduced.

Following our approach from Section 5.3, Heeger proposes a variant where he estimates the downstream capacitance of label $(v, I)$ by $C_{\text{smt}}(v, I)$, where $C_{\text{smt}}(v, I)$ is a lower bound for the capacitance of a Steiner tree connecting $v$ and $I$ (including sink capacitances). This way, he can compute a Steiner tree $Y$ achieving an $\alpha$-approximation for the RC-Aware Steiner Tree Problem in $\mathcal{O}(2^k(n \log n + \theta m) + 3^k n)$ time, where

$$\alpha := \max_{v \in V(Y)} \frac{C(Y(v))}{C_{\text{smt}}(v, T(Y(v)))} \qquad \text{(with } 0 \,/\, 0 := 1\text{)},$$

$k := |T|$, $n := |V(G)|$, $m := |E(G)|$, and $\theta$ is an upper bound for the running time of one computation of $C_{\text{smt}}$. Heeger also proposes a variant where the downstream capacitance of label $(v, I)$ equals the capacitance of the Steiner tree corresponding to $(v, I)$, and an optimal but slow variant, where the downstream capacitance is part of the label, and multiple labels are stored for any combination of $v \in V(G)$ and $I \subseteq T$. To save running time, Heeger's algorithm can also be restricted to only consider one routing topology, which is obtained by running our algorithm from this section (including Algorithm 3) first in Heeger's implementation. For more on the work of Heeger including experimental results, the reader is referred to [49].

As Algorithm 3 is unlikely to improve upon the initial tree for nets with only three pins by means of changing the two-dimensional outline of the tree and also entails some additional running time, we only call it for nets with at least four pins in our implementation. For nets with three pins we stop after the computation of $Y_0$, but set $\lambda = 0.25$ in the definition of $\mu$ to promote an increased use of upper layers, as we do not run Algorithm 3 afterwards to decrease wire delays. For nets with two pins, of course, we just run our RC-aware path search from Section 5.3.

5.4.3.2. *Application of Algorithm 3.* Having constructed the initial tree $Y_0$ as explained in Section 5.4.3.1, we are going to apply Algorithm 3 from Section 5.4.2: We run Algorithm 3 for every value of $\varepsilon \in [0.25, 25]$ that is a multiple of 0.25 without performing a path search for reconnecting components in line 5. Instead, we estimate the price of

the resulting solution by assuming that the newly found paths are shortest paths, and that their wire resistance and capacitance per unit length is given by

$$R^*_{\text{wire}} := \frac{\sum_{(v,w)\in E_{\text{wire}}(Y_0)} R(v,w)}{\sum_{(v,w)\in E_{\text{wire}}(Y_0)} \text{dist}(v,w)}, \qquad C^*_{\text{wire}} := \frac{\sum_{(v,w)\in E_{\text{wire}}(Y_0)} C(v,w)}{\sum_{(v,w)\in E_{\text{wire}}(Y_0)} \text{dist}(v,w)},$$

where $E_{\text{wire}}(Y_0)$ is the set of wiring edges of $Y_0$ (cf. Definition 2.6).[2] The idea is to be guided by $Y_0$ for estimating $R^*_{\text{wire}}$ and $C^*_{\text{wire}}$, as during the construction of $Y_0$ we already give incentive to use the upper layers for nets with a high capacitance or high timing prices, unless this is not possible due to congestion (cf. Section 5.4.3.1). Moreover, Table 5.1 shows that assuming shortest paths to be used for reconnects in line 5 is a reasonable choice. Congestion prices are neglected for our price estimates, i.e. at this point we only consider timing prices.

If we find a solution with lower estimated timing prices than $Y_0$, we use our RC-aware path search from Section 5.3 in order to perform the actual reconnect in line 5. Here, we again use the super-sink model from Section 5.4.3.1, but this time we model the component to be reconnected as super-sink: Consider the situation in line 5 of Algorithm 3 when $Y(w)$ needs to be reconnected to $s$. We model $Y(w)$ as super-sink $t^*$, where we set

$$\text{price}(t^*) := \text{price}(T(Y(w))), \qquad C(t^*) := C(E(Y(w))) + C(T(Y(w))).$$

We then use rcp as in Section 5.3 to reconnect $t^*$ to $s$, but make one simple modification: For $(v,w) \in E(G)$ we set

$$\text{cp}_{\text{lb}}(v,w) := \text{price}(T) \cdot R(s) + \text{price}(t^*) \cdot \Big(\text{R}_{\text{via}}(z_s, z_v) + R^{\min}_{\text{wire}} \cdot \text{dist}(s,v)\Big),$$

as the source delay affects all sinks in $T$, while the wire delay along the new path only affects $t^*$. Moreover, similar to Section 5.4.3.1, we perform distance computations to $t^*$ with respect to the bounding box of $V(Y(w))$, i.e. for any $v \in V(G)$ we set $\text{dist}(v,t^*) := \min_{a\in BB(V(Y(w)))\times Z} \text{dist}(v,a)$. This allows distance computations to be performed in constant time.

5.4.3.3. *Postprocessing and Returning the Tree.* For achieving its proclaimed approximation guarantee for the Simplified RC-Aware Steiner Tree Problem, Algorithm 3 requires all paths that are reconnecting components in line 5 to be disjoint. In our formulation of the Simplified RC-Aware Steiner Tree Problem and RC-Aware Steiner Tree Problem, this is always possible, as we do not define our Steiner trees to be subgraphs of the global routing graph. Instead, Definition 2.3 ensures that our output tree $Y$ can have an arbitrary graph structure, which is embedded into the global routing graph.

[2] We set $0 / 0 := 1$ for a valid (but meaningless) definition in degenerate cases.

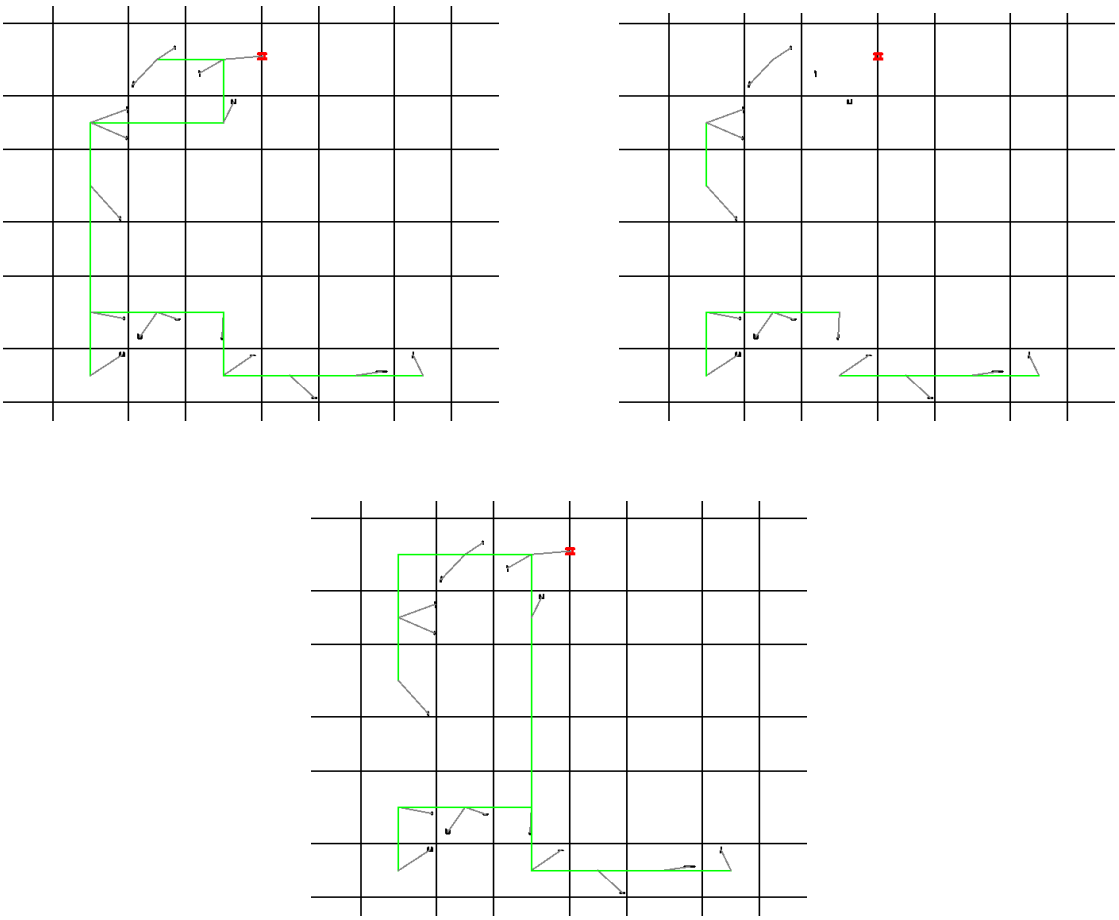


FIGURE 5.7. A sequence from practice illustrating the application of Algorithm 3 as described in Section 5.4.3. Black lines depict global routing tiles, and green lines illustrate global wires running from tile center to tile center connecting the pins (gray diagonal lines). The source pin is colored red while sink pins are colored black. We start with the initial tree $Y_0$ (upper left picture). $Y_0$ is short, but wire delays to the sinks in the lower right corner are presumably large. Algorithm 3 then deletes multiple wires (upper right picture), and the resulting components are reconnected to the source pin (lower picture). As described in Section 5.4.3.3, some wires (e.g. the ones reconnecting the components at the bottom to the source pin) are merged at the end.

In a geometric sense, however, $Y$ might contain parallel segments and loops. In practice, this can be undesirable, as it may complicate the representation and handling of Steiner trees in the data structures of all involved tools. For example, in the IBM physical design flow where BonnRouteGlobal is used, Steiner trees are represented as a set of wires in general interfaces, and connectivity is defined by geometric intersection. Therefore, we

merge parallel segments after application of Algorithm 3 as in Section 5.4.3.2, and remove possible loops in a geometric sense by computing a shortest-path tree in $Y$ rooted at $s$. In the end, we either return $Y$ or $Y_0$ depending on which one of the two achieves a lower price.

An example from practice is given by Figure 5.7. As we can see in this example, the merging of wires can result in topology changes, as now multiple components computed by Algorithm 3 may be driven by the same wire. In such cases, the layer assignment algorithm from Section 6.3 — possibly even enhanced by the capability to assign wire types — might be an advantage, as this algorithm could assign wires driving multiple components to higher layers or assign thicker wire types to them. Another possibility is to use the approach of Heeger [49], who fixes the topology of $Y$, but recomputes a tree approximately minimizing the price among all trees with this topology (cf. Section 5.4.3.1).

## 5.5. Experimental Results

In this section we are going to present experimental results that demonstrate the effectiveness of our RC-aware routing framework, which encompasses our general timing-aware routing framework from Chapter 4 as well as the RC-aware routing oracle presented in this chapter. As explained in Appendix A, we use recent 14nm microprocessor designs provided by IBM for our experimental results, which are optimized up to a point where they are ready for routing. In particular, the input contains a layer and wire type assignment (cf. Section 1.3.2.4.1) that has been generated by the IBM design flow. All experiments appearing in this section are conducted with 16 threads on an Intel Xeon E5-2667 v2 server running at 3.30 GHz. For the explanation of metrics appearing in subsequent tables we refer to Appendix A.4. Here, all metrics are measured after connecting to exact shapes using the methods from Chapter 6, more precisely Section 6.4.

In Table 5.3 we compare results for three different routing methods, where each one is represented by one row in the table: For the method titled *Estimates*, every net is assumed to be routed by an approximately shortest Steiner tree that is located completely on the two lowest layers of the layer assignment of the given net (apart from vias connecting to the pins), using the wire type assigned to the net. Here, the approximately shortest Steiner tree is computed using the methods outlined in Section 2.3.2. This is exactly the method that is used to estimate routing trees during timing optimization prior to global routing, as described in Section 1.2.2.

The run labeled *Traditional BonnRouteGlobal* (short: *Traditional BRG*) represents the traditional way of performing global routing. Here, ”traditional” also implies that basically the Traditional Global Routing Problem from Section 2.2 is solved, which is

| Unit (# nets) | Run | WS [ps] | FOM [ps] | EV | WL [m] | wACE4 [%] | OFtgt [100 pitch$^2$] | RT [h:mm:ss] |
|---|---|---|---|---|---|---|---|---|
| U1 | Estimates | -157 | -38463 | 621 | 0.939 | — | — | — |
| (77 528) | Traditional BRG | -179 | -41236 | 704 | 0.958 | 88.8 | 5.9 | 0:00:55 |
| | RC-Aware BRG | -129 | -29121 | 597 | 0.953 | 88.8 | 5.4 | 0:02:32 |
| U2 | Estimates | -104 | -81487 | 198 | 1.087 | — | — | — |
| (79 119) | Traditional BRG | -119 | -82537 | 347 | 1.111 | 87.7 | 0.0 | 0:00:32 |
| | RC-Aware BRG | -97 | -57441 | 311 | 1.102 | 87.5 | 0.0 | 0:02:00 |
| U3 | Estimates | -150 | -159305 | 406 | 1.236 | — | — | — |
| (100 827) | Traditional BRG | -150 | -157248 | 659 | 1.245 | 83.3 | 0.0 | 0:00:51 |
| | RC-Aware BRG | -149 | -118185 | 296 | 1.248 | 86.5 | 0.0 | 0:02:49 |
| U4 | Estimates | -245 | -214126 | 149 | 1.202 | — | — | — |
| (111 140) | Traditional BRG | -210 | -208458 | 260 | 1.235 | 88.7 | 3.3 | 0:00:58 |
| | RC-Aware BRG | -196 | -152791 | 197 | 1.230 | 89.6 | 8.7 | 0:03:11 |
| U5 | Estimates | -63 | -58267 | 52 | 1.367 | — | — | — |
| (119 228) | Traditional BRG | -63 | -56608 | 19 | 1.376 | 80.8 | 0.0 | 0:00:45 |
| | RC-Aware BRG | -62 | -27207 | 5 | 1.383 | 81.7 | 0.0 | 0:03:04 |
| U6 | Estimates | -122 | -325030 | 237 | 4.861 | — | — | — |
| (254 208) | Traditional BRG | -130 | -345512 | 1201 | 5.031 | 89.0 | 67.6 | 0:02:10 |
| | RC-Aware BRG | -121 | -226617 | 1406 | 5.013 | 88.6 | 26.8 | 0:08:48 |
| U7 | Estimates | -94 | -125151 | 375 | 4.714 | — | — | — |
| (276 799) | Traditional BRG | -109 | -156522 | 598 | 4.762 | 83.1 | 0.0 | 0:02:37 |
| | RC-Aware BRG | -55 | -64614 | 765 | 4.758 | 83.8 | 0.0 | 0:09:18 |
| U8 | Estimates | -109 | -1979601 | 3110 | 37.088 | — | — | — |
| (1 681 671) | Traditional BRG | -453 | -1980714 | 11309 | 37.508 | 85.9 | 358.9 | 0:17:01 |
| | RC-Aware BRG | -86 | -836609 | 9335 | 37.572 | 86.1 | 34.0 | 1:04:18 |

TABLE 5.3. Experimental results comparing our RC-aware routing framework to traditional BonnRouteGlobal and routing tree estimates.

the standard way of formulating the global routing problem. To achieve this, Traditional BonnRouteGlobal uses the resource sharing framework presented in Chapter 4 without the enhancements to incorporate timing constraints. Instead, timing is considered indirectly by means of a layer and wire type assignment and netlength bounds for timing

critical nets. Moreover, minimization of total wire length is regarded as objective function. Traditional BonnRouteGlobal strictly obeys the layer and wire type assignment given in the input: Layers below the minimum layer of the layer assignment are completely forbidden for routing (except for dropping vias on the pins), and the wire type assigned to the net is used for all wires. The aforementioned netlength bounds are also generated by the IBM design flow with the goal of preventing detours on timing-critical nets, and they are treated by Traditional BonnRouteGlobal as constraints similar to routing capacity constraints. Traditional BonnRouteGlobal has been the default global routing method in the IBM design flow for several years and is as such well established. An overview and comparison to another industrial router is given in [41] by Gester et al.

Finally, the last experiment denoted by *RC-Aware BonnRouteGlobal* (short: *RC-Aware BRG*) shows the results that we obtain using our new timing-aware routing framework from Chapter 4 in combination with the RC-aware routing oracle from this chapter. In this run, we ignore the layer assignment in the input when routing nets, i.e. we always allow all layers when routing any net. As already mentioned in the beginning of this chapter, we still obey the wire type assignment, as wire type optimization is not yet incorporated in our RC-aware routing framework. The netlength bounds for timing-critical nets that are used by Traditional BonnRouteGlobal are also ignored by RC-Aware BonnRouteGlobal, as it optimizes RC delay directly, and is therefore not reliant on netlength bounds.

The numbers in Table 5.3 demonstrate a strong performance of RC-Aware BonnRouteGlobal: While Traditional BonnRouteGlobal correlates fairly well with the routing estimates, RC-Aware BonnRouteGlobal achieves significantly better timing numbers on every design. On some designs, mainly U6 and U8, routing congestion can also be reduced by ignoring the layer assignment. Similarly, RC-Aware BonnRouteGlobal achieves a lower wire length than Traditional BonnRouteGlobal on some units, which is also mainly due to the fact that RC-Aware BonnRouteGlobal ignores the layer assignment, therefore making the global routing problem easier. Moreover, there can be stronger incentives to keep timing-critical nets short due to timing costs (cf. Chapter 4) in RC-Aware BonnRouteGlobal compared to Traditional BonnRouteGlobal, as netlength bounds are only set for a small fraction of all nets, and the objective function of minimizing wire length is subordinate to the goal of keeping congestion low. One can also see that the wire length achieved by RC-Aware BonnRouteGlobal is never significantly higher than the one achieved by Traditional BonnRouteGlobal, and always within a few percent of the wire length achieved by the routing estimates, which can be considered a lower bound

for practical purposes.[3] This invalidates possible worries about a noticeable wire length increase by using the RC tree topology algorithm from Section 5.4.

Naturally, running time is higher when using RC-Aware BonnRouteGlobal compared to Traditional BonnRouteGlobal, as the incorporation of timing constraints results in a more difficult variant of the global routing problem. However, running times are still fairly low an all units except U8, where RC-Aware BonnRouteGlobal takes roughly one hour. For a unit of this size, this is still adequate, in particular in light of the fact that results are significantly better. Moreover, as we will see later in Section 7.6, the complete routing flow including detailed routing takes several hours on this design, making the running time penalty of running RC-Aware BonnRouteGlobal easily bearable.

When it comes to the last metric that we tracked, namely the number of electrical violations, the picture is mixed: Compared to Traditional BonnRouteGlobal, the number of electrical violations decreases on most, but not all, units, when running RC-Aware BonnRouteGlobal. Overall, this can be considered as an improvement. Compared to the routing tree estimates, however, the number of electrical violations increases on multiple designs, especially the large ones. This can be explained by the fact that timing optimization running before global routing uses routing estimates, and therefore optimizes the netlist in such a way that electrical violations are minimized when routing estimates are used. RC-Aware BonnRouteGlobal, however, only optimizes timing, but does not model slew or capacitance limits directly. This can result in subpar routes for timing-uncritical nets that do not negatively impact timing, but result in electrical violations. Here, a reduction of electrical violations can be achieved afterwards by running the routing based optimization methods from Chapter 7, as is shown in Section 7.6. However, as a task for the future, it would be better to avoid electrical violations already in RC-Aware BonnRouteGlobal.

The second table, Table 5.4, then compares RC-Aware BonnRouteGlobal strictly obeying the layer assignment (short: RCA-BRG + LA) to RC-Aware BonnRouteGlobal ignoring the layer assignment (short: RCA-BRG - LA), where the latter mode is the one used for the runs depicted in Table 5.3. Here, one can see that timing and congestion are better on every design when the layer assignment is ignored, and wire length also decreases slightly on most designs. Regarding running times, the search space for path searches is larger when layer assignments are ignored, which should result in larger running times on the one hand, but on the other hand, the global routing problem becomes easier when layer assignments are ignored, which typically results in smaller running times. As a result, the running times of both runs are relatively even, with slightly larger running

[3]It is not strictly a lower bound as we use approximation algorithms for computing shortest Steiner trees for large terminal sets as outlined in Section 2.3.2, but on average, it should be close to an actual lower bound.

| Unit (# nets) | Run | WS [ps] | FOM [ps] | EV | WL [m] | wACE4 [%] | OFtgt [100 pitch$^2$] | RT [h:mm:ss] |
|---|---|---|---|---|---|---|---|---|
| U1 | RCA-BRG + LA | -129 | -30605 | 602 | 0.956 | 89.4 | 8.2 | 0:02:28 |
| (77 528) | RCA-BRG - LA | -129 | -29121 | 597 | 0.953 | 88.8 | 5.4 | 0:02:32 |
| U2 | RCA-BRG + LA | -97 | -58669 | 330 | 1.109 | 90.2 | 16.8 | 0:02:06 |
| (79 119) | RCA-BRG - LA | -97 | -57441 | 311 | 1.102 | 87.5 | 0.0 | 0:02:00 |
| U3 | RCA-BRG + LA | -149 | -120288 | 316 | 1.247 | 86.8 | 0.0 | 0:02:48 |
| (100 827) | RCA-BRG - LA | -149 | -118185 | 296 | 1.248 | 86.5 | 0.0 | 0:02:49 |
| U4 | RCA-BRG + LA | -196 | -156150 | 216 | 1.235 | 89.6 | 15.9 | 0:03:13 |
| (111 140) | RCA-BRG - LA | -196 | -152791 | 197 | 1.230 | 89.6 | 8.7 | 0:03:11 |
| U5 | RCA-BRG + LA | -62 | -27539 | 2 | 1.383 | 82.1 | 0.0 | 0:03:03 |
| (119 228) | RCA-BRG - LA | -62 | -27207 | 5 | 1.383 | 81.7 | 0.0 | 0:03:04 |
| U6 | RCA-BRG + LA | -121 | -238790 | 1117 | 5.030 | 88.9 | 38.8 | 0:07:47 |
| (254 208) | RCA-BRG - LA | -121 | -226617 | 1406 | 5.013 | 88.6 | 26.8 | 0:08:48 |
| U7 | RCA-BRG + LA | -55 | -72571 | 586 | 4.768 | 85.7 | 0.0 | 0:09:10 |
| (276 799) | RCA-BRG - LA | -55 | -64614 | 765 | 4.758 | 83.8 | 0.0 | 0:09:18 |
| U8 | RCA-BRG + LA | -90 | -861559 | 7792 | 37.584 | 86.9 | 340.0 | 1:01:35 |
| (1 681 671) | RCA-BRG - LA | -86 | -836609 | 9335 | 37.572 | 86.1 | 34.0 | 1:04:18 |

TABLE 5.4. Experimental results comparing RC-Aware BonnRoute-Global obeying the layer assignment in the input ('+ LA') to ignoring it ('- LA').

times on large units when the layer assignment is ignored. With respect to electrical violations, the picture is again mixed: On smaller units, ignoring the layer assignment results in a slight reduction of electrical violations. On our large units U6, U7 and U8 with a complete layer stack, ignoring the layer assignment rather results in more electrical violations. Here, violating the layer assignment on some timing-uncritical nets might result in congestion benefits, but also in electrical violations.

As a summary, it is safe to say that RC-Aware BonnRouteGlobal achieves significantly better results than Traditional BonnRouteGlobal, and can therefore be considered the superior routing method. Moreover, RC-Aware BonnRouteGlobal achieves better results when ignoring the layer assignment, which also demonstrates the validity of our methods from this chapter to construct Steiner trees minimizing Elmore delays and congestion not only in a two-, but directly in a three-dimensional model.

CHAPTER 6

# Connecting to Exact Shapes

In this chapter we illustrate the process of transforming a Steiner tree connecting the projected shapes of a net into one connecting exact shapes (cf. Definitions 2.7 and 2.9). Connecting to exact instead of projected shapes during global routing has several advantages: Firstly, it bears more resemblance to an actual detailed routing, and therefore gives much more accurate estimates for metrics like timing and power consumption than a global routing that only connects projected shapes. This is for instance very important for computing reasonable signal delays for the timing-aware global routing framework from Chapter 4, or for the routing based optimization flow from Chapter 7. A second important advantage of connecting to exact shapes is that it allows for a more precise modeling of local congestion effects. Thirdly, a locally well optimized global routing connecting to exact shapes might constitute better input for track assignment and detailed routing (cf. Sections 1.3.3 and 1.3.4) depending on whether these algorithms make use of the local structure of global routes. At this point it might be worth noting that our results from this chapter are not meant to replace track assignment, as we do not make any attempt to obey minimum distance rules (cf. Section 1.3.1).

In order to connect to exact shapes we apply a multi-stage approach when routing a net $N$: Firstly, we compute a Steiner tree $X$ connecting the projected pin and prewire shapes of $N$ using the routing algorithms from Chapter 5 and Chapter 7 in their respective scenarios. Afterwards, we transform $X$ into a two-dimensional tree $Y_0$ connecting the two-dimensional projections of the exact shapes of $N$ while still preserving the global topology of $X$. This step is described in Section 6.1. Here, and for the rest of this chapter, a *two-dimensional tree* is a tree embedded rectilinearly into $\mathbb{R}^2$ in the sense of Definition 2.3.

Thereafter, we apply a post-optimization that optimizes the $x$- and $y$-coordinates of the Steiner points of $Y_0$ on a local scale, yielding a two-dimensional tree $Y$ that is locally optimized. This post-optimization step is described in Section 6.2. In order to obtain a three-dimensional tree we then use a timing- and congestion-aware layer assignment step that assigns $z$-coordinates to the edges of $Y$ while preserving their $x$- and $y$-coordinates. This layer assignment step is described in Section 6.3. It is working as part of the global router and assigns individual wires in contrast to traditional layer assignment tools that

assign whole nets, and that are usually separate tools generating constraints for the router (cf. Section 1.3.2.4.1).

As a timing model for our layer assignment step we use the Elmore delay model from Section 3.4. However, adaptions of our results may also be used to optimize timing with other delay models, e.g. the linear delay model described by Held et al. [53]. The congestion model used during our layer assignment step is very simple and general. It can be used in traditional settings where only wires crossing tile borders are considered, but it can also optimize local congestion for congestion models that are reliant on that. As throughout the rest of this thesis, we do not incorporate a wire type assignment (cf. Section 1.3.2.4.1) in the layer assignment algorithm from Section 6.3, but assume a fixed wire type to be given for every net. However, as will become clear when reading Section 6.3, extending our algorithm to also assign wire types should be straightforward. We note that the layer assignment algorithm from Section 6.3 is not necessarily connected to the process of connecting to exact shapes. In fact, it could also be used in other contexts, e.g. as part of the net router from Chapter 5. However, as it is an essential part of the overall process of connecting to exact shapes outlined in this chapter, we present it in this context. The overall process from this chapter is currently being implemented in BonnRouteGlobal, and only partly finished at the time of this writing. Therefore, Section 6.4 concludes this chapter with a brief description of the current implementation that is being used to produce experimental results throughout this thesis.

The process outlined above may remind of global routing frameworks that first route in the two-dimensional projection of the global routing graph and assign layers afterwards, as e.g. described by Lee and Wang [78, 79]. However, our approach has the key advantage that during the actual routing step we already use our net router from Chapter 5 that optimizes timing and congestion in a three-dimensional model. This makes sure that subsequent steps — local optimization and layer assignment — are able to find a solution that is good with respect to timing and congestion.

A different approach to connect to exact shapes is taken by Hähnle and Saccardi [45, 100], who devise a global routing framework that neither uses the global routing graph as presented in Section 1.3.2.1 nor projected pin and prewire shapes. Instead, they consider exact shapes directly during the routing process and therefore do not need a separate routine that connects to exact shapes a posteriori. This approach is more precise, but it is also rather elaborate and requires significant changes in the global router, and is therefore considered out of scope for this thesis. In contrast to that, our approach from this chapter is simpler and easier to integrate into the current implementation of BonnRouteGlobal. However, it may be noted that our timing-aware layer assignment

algorithm from Section 6.3 might still be beneficial even in the scenario of Hähnle and Saccardi [45, 100].

## 6.1. From Projected to Exact Shapes

In this section we describe the process of converting a Steiner tree connecting the projected shapes of a net to the two-dimensional outline of one connecting exact shapes. We give a rather descriptive characterization of this process without going into every detail. Let $N$ be a net and $X$ be a Steiner tree connecting the projected shapes of $N$. Let $X_1, \dots, X_k$ be connected subgraphs of $X$ such that

- $V(X_i) \cap V(X_j) = \emptyset$, $i, j = 1, \dots, k$, $i \neq j$,
- $N \subseteq \bigcup_{i=1}^{k} V(X_i)$,
- $N \cap V(X_i) \neq \emptyset$, $i = 1, \dots, k$,
- $V(X_i) = \{v \in V(X) : \mathrm{dist}_X(\pi_i, v) = 0\}$, where $\pi_i \in N \cap V(X_i)$ is an arbitrary pin in $V(X_i)$, $i = 1, \dots, k$.

These subgraphs can be computed as follows: Assume that for some $i \in \mathbb{N}$, $X_1, \dots, X_{i-1}$ have already been computed, and $N \not\subseteq \bigcup_{j=1}^{i-1} V(X_j)$. Then pick $\pi_i \in N \setminus \bigcup_{j=1}^{i-1} V(X_j)$ and compute $V(X_i)$ by traversing $X$ starting at $\pi_i$ (e.g. with depth-first search), using only zero-length edges (including vias). Intuitively, each $X_i$ corresponds to a subgraph of $X$ in a global routing tile where connections to exact pin shapes have to be added, as $E(X_i)$ contains only vias and other zero-length edges. We note that the subgraphs $X_1, \dots, X_k$ are uniquely defined (up to the numbering).

We derive a two-dimensional projection $X'$ of $X$ by keeping the graph structure of $X$, but neglecting $z$-coordinates of all vertex positions.[1] In the same way we derive a subgraph $X_i'$ from $X_i$ for all $i = 1, \dots, k$. Moreover, we do not regard $X'$ as embedded into the global routing graph, but as embedded directly into the chip area.

For each $i = 1, \dots, k$ we apply the following procedure: We first subdivide all edges in $\delta_{X'}(V(X_i'))$ at the borders of the global routing tile corresponding to $X_i'$, and then remove the parts located inside this tile. Let $M_i$ be the set of endpoints of the previously subdivided edges located at the tile borders and $N_i := N \cap V(X_i)$. We compute an approximately shortest rectilinear Steiner tree (cf. Section 2.3.2) connecting $M_i$ and the two-dimensional projections of the exact shapes of $N_i$ and insert it into $X'$ in place of $X_i'$. When this process has been completed for all $i = 1, \dots, k$, we arrive at a two-dimensional Steiner tree $Y_0$ that connects the two-dimensional projections of exact pin shapes. By construction, $Y_0$ differs from $X'$ only locally in the tiles where local rectilinear Steiner trees connecting exact shapes were inserted. An illustration is given by Figure 6.1.

[1] We note that $X'$ might contain loops or parallel segments in a geometric sense even if $X$ does not, but this is not an issue in this context.

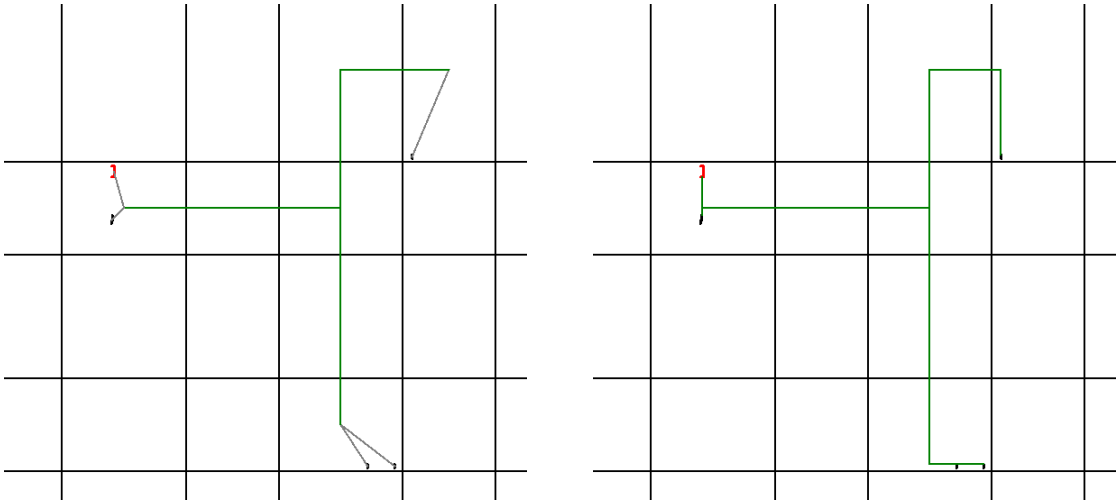


FIGURE 6.1. Illustration of connecting to exact shapes as outlined in Section 6.1: We start with a route connecting projected pin shapes (left picture), as indicated by the diagonal grey lines. In tiles containing pins the global wiring reaching to the tile center is then replaced by a Steiner tree connecting exact pin shapes, but tile borders are still crossed in their center (right picture). This can lead to local detours, which are resolved using the methods from Section 6.2.

The above description neglected the case where prewires are present, which occurs frequently when running the routing based optimization flow from Chapter 7. Analogously to pin shapes, these prewires, which are originally given by their exact shapes, can be projected to the global routing graph and used during the net routing process by mapping their endpoints to the vertex whose vertex area contains them. If connecting to exact shapes of prewires is required in certain global routing tiles, then we proceed as described above, but solve the RECTILINEAR MINIMUM STEINER TREE WITH PREWIRES PROBLEM as described in Section 2.3.3 instead of the RECTILINEAR MINIMUM STEINER TREE PROBLEM to also include the two-dimensional projections of these prewires in $Y_0$.

## 6.2. Optimizing x- and y-Coordinates

Consider the two-dimensional tree $Y_0$ resulting from the procedure described in Section 6.1. By construction, $Y_0$ crosses tile borders only in their centers. This can cause local detours, as can be seen in Figure 6.1. Therefore, we proceed with a post-optimization method by Kiefner [71], which fixes the topology of $Y_0$ but optimizes the positions of its Steiner points within their tiles. Before giving the problem statement, we define $R := \{[x_1, x_2] \times [y_1, y_2] \subseteq \mathbb{R}^2 : x_1 \leq x_2, y_1 \leq y_2\}$ to be the set of axis-parallel rectangles in $\mathbb{R}^2$. With that, the problem statement is as follows:

Problem 6.1: Minimum Steiner Tree with Fixed Topology Problem

**Input:** A Steiner tree topology $Y_0$ for a net $N$, rectangular move bounds $r\colon V(Y_0) \setminus N \to R$.

**Task:** Find Steiner point positions $p\colon V(Y_0) \setminus N \to \mathbb{R}^2$ with $p(v) \in r(v)$ for all $v \in V(Y_0) \setminus N$ minimizing

$$\sum_{(v,w)\in E(Y_0)} \operatorname{dist}(p(v), p(w)).$$

The move bounds $r$ are specific to our application in global routing. However, the case without move bounds can also be modeled in our formulation by setting all move bounds to the bounding box of $N$. Moreover, the problem could also be formulated in a more general fashion to consider Steiner trees in $\mathbb{R}^d$ ($d \in \mathbb{N}$), and the results mentioned in this section would still hold. However, we restrict ourselves to only cover the two-dimensional case, as this is the one that is relevant in our scenario.
Sankoff and Rousseau [103] show how to solve the Minimum Steiner Tree with Fixed Topology Problem in linear time:

Theorem 6.2 (Sankoff and Rousseau [103]). *One can solve the* Minimum Steiner Tree with Fixed Topology Problem *in* $\mathcal{O}(|V(Y_0)|)$ *time.*

An improvement of their algorithm by Kiefner [71] gives a stronger result by optimizing source-sink path lengths as secondary objective. In the following, we let $Y$ denote the Steiner tree defined by the topology of $Y_0$ and the Steiner point positions $p$ from the Minimum Steiner Tree with Fixed Topology Problem. Moreover, as usual, $s$ denotes the source and $T$ the sinks of $N$:

Theorem 6.3 (Kiefner [71]). *One can find a solution* $Y$ *of the* Minimum Steiner Tree with Fixed Topology Problem *such that*

*(1)* $\sum_{(v,w)\in E(Y)} \operatorname{dist}(v, w)$ *is minimized,*
*(2)* $\operatorname{dist}_Y(s, t)$ *is minimum for all* $t \in T$ *among all trees minimizing* (1),

*in* $\mathcal{O}(|V(Y_0)|)$ *time.*

It is not obvious that a solution fulfilling (2) actually exists, but this is proven in [71]. Compared to a tree only satisfying condition (1), a tree also satisfying condition (2) yields a better timing due to reduced source-sink path lengths.
In our implementation we apply the algorithm of Kiefner using the Steiner tree topology of $Y_0$ from Section 6.1. Before applying the algorithm, we clone every vertex $v \in V(Y_0)$ that is incident to both immovable prewires (e.g. detailed wires as in Chapter 7) and movable global wires into two connected vertices such that one of the two is incident

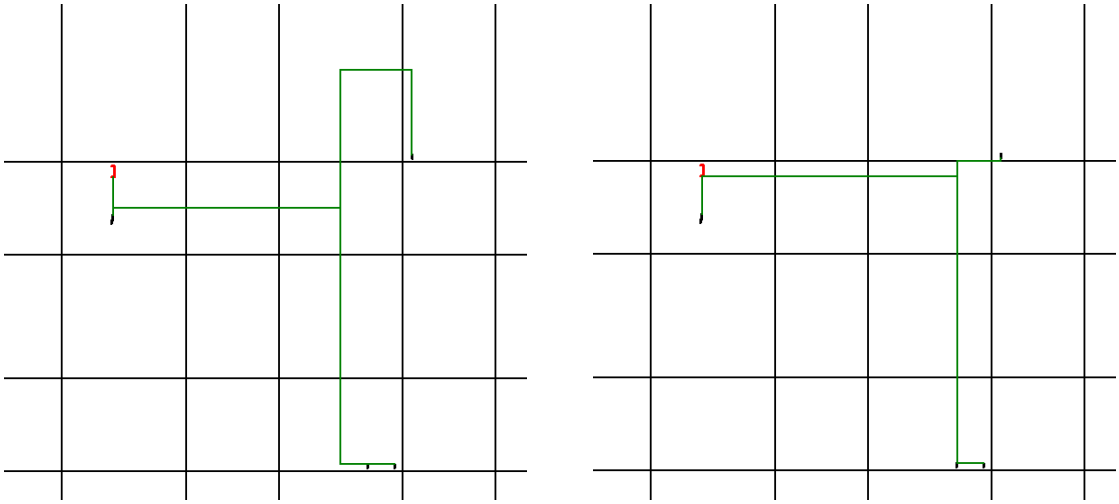


FIGURE 6.2. Continuation of the sequence started in Figure 6.1: After applying the procedure from Section 6.1, our route connects exact shapes, but crosses tile borders always in their center (left). This can result in local detours, as when connecting to the pin in the upper part of the picture. Applying the algorithm of Kiefner [71] as outlined in Section 6.2 solves this problem by moving Steiner points to optimum positions within their tiles (right). One can also observe that not only the length of the tree, but also the length of source-sink paths is minimized: Shifting the horizontal wire attached to the source pin (red) downwards as in the left picture yields a tree with the same length, but longer source-sink paths. In this case, however, Kiefner's algorithm yields a tree where all source-sink paths are shortest paths.

to all immovable prewires and the other to all movable global wires. As move bound for a Steiner vertex $v \in V(Y_0)$ we then use the area of the global routing tile that $v$ is located in if $v$ is not connected to immovable prewires, and we fix the positions of Steiner vertices that are connected to immovable prewires. This way, we ensure that this post-optimization only works locally and does not change the Steiner tree on a global scale. Letting this algorithm change the route on a global scale would be undesirable as it does not consider routing congestion. An illustration of the process from this section is given by Figure 6.2.

## 6.3. Assigning Layers

Let $Y$ be the two-dimensional Steiner tree that results from the optimization of $x$- and $y$-coordinates from Section 6.2. The last step to be run in our overall process consists of assigning layers to the edges of $Y$. In accordance with the resource sharing framework from Chapter 4 the objective will be to minimize a weighted sum of congestion and timing

prices, where we use the Elmore delay model from Section 3.4 to measure signal delays. As a technical prerequisite we assume throughout this section that the maximum vertex degree of $Y$ is bounded by a constant. Technically, this is not assured by our general framework from Chapter 2, as we embed arbitrary Steiner trees into the layered chip area or the global routing graph. In our implementation, however, every Steiner vertex in $Y$ will only be incident to at most four wires (two vias and two wires expanding in preference direction of the layer). This section is structured as follows: We start with explaining the difference between working locally and globally in Section 6.3.1 before we state the problem formulation in Section 6.3.2. In Section 6.3.3 we then introduce RC prices that are akin to the ones from Section 5.3.2, and give a layer assignment algorithm minimizing these RC prices in Section 6.3.4. Section 6.3.5 then concludes this section with a hardness result showing that requiring a bounded maximum vertex degree for $Y$ is necessary for deriving a layer assignment algorithm minimizing our RC prices.

**6.3.1. Working Locally or Globally.** As the first step in our process from Section 6.1 only works locally on tile-internal connections, and the second step from Section 6.2 does not move Steiner points outside of their global routing tiles, it is possible to map every tile border crossing segment in $Y$ back to an edge in the original global route $X$ (connecting projected shapes) and therefore preserve the wiring layers used by $X$ on a global scale.[2] In that case, the only thing that is left to do for the layer assignment algorithm is to assign layers to tile-internal segments, and therefore we refer to this method as *working locally*.
The opposite to that is *working globally*, which involves also assigning long wires that are crossing tile borders. This certainly results in longer running times, but when it comes to quality of results, it seems to be the more promising of both options. As our problem formulation from Section 6.3.2 allows fixing the layers of certain wires, working locally and globally both fit into the theoretical framework described here. However, in practice, both strategies will most likely give notably different results. We elaborate on the strategy used for experimental results for this thesis in Section 6.4.

**6.3.2. Problem Formulation.** Let $S := A \times Z$ be the layered chip area from Definition 2.5. Moreover, let $s \in V(Y)$ be the source and $T \subseteq V(Y)$ be the sink pins of the net $N$ connected by $Y$. As $Y$ is a two-dimensional tree connecting the two-dimensional projections of exact shapes of $N$, we assume $Y$ to be embedded into $A$. For each edge $e \in E(Y)$ we are given a non-empty set $Z_e \subseteq Z$ of allowed layers for $e$. Moreover, as the layers of pins are fixed, we are given a fixed layer $z(\pi)$ for every $\pi \in N$.

[2]This only holds if for all pins $\pi \in \Pi$ we have $p_{\mathrm{pr}}(\pi) = v$ for the vertex $v \in V(G)$ whose vertex area contains $p_{\mathrm{ex}}(\pi)$. If that is not the case, some tile border crossing segments are added by the method from Section 6.1. However, these segments should be short and only be used to access pins.

To simplify the presentation, we assume (as always) that $N$ is the set of leaves of $Y$, and that each $\pi \in N$ is reached by a zero-length edge $e_\pi$ with $Z_{e_\pi} = \{z(\pi)\}$.[3] Subsequently, the term *two-dimensional tree* will encompass this modeling of pin layers as zero-length edges as well as a constant maximum vertex degree.
We then are looking for a mapping $z\colon E(Y) \to Z$ with $z(e) \in Z_e$ for all $e \in E(Y)$. Such a mapping is called a *layer assignment*. In particular, when using the term layer assignment, we implicitly assume that the constraints $z(e) \in Z_e$ for all $e \in E(Y)$ are met (unless explicitly stated otherwise). For $v \in V(Y)$ a layer assignment $z$ defines values

$$z_{\min}(v) := \min\{z(e) : e \in \delta_Y(v)\}, \qquad z_{\max}(v) := \max\{z(e) : e \in \delta_Y(v)\},$$

that define the layer range that has to be spanned by a via at $v$. Clearly, these vias have to be incorporated into the objective function. The objective function will be dependent on the following additional input:

Definition 6.4. Given a two-dimensional Steiner tree $Y$ and a set of layers $Z = \{1, \dots, n_z\}$, *layer-dependent edge prices* are defined by a function price$\colon E(Y)\times Z \to \mathbb{R}_{\geq 0}$ and a function price$\colon V(Y) \times Z \to \mathbb{R}_{\geq 0}$. Here, price$(e, z)$ for $(e, z) \in E(Y) \times Z$ denotes the price of assigning $e$ to $z$, and price$(v, z)$ for $(v, z) \in V(Y) \times Z$ denotes the price of a via from $z$ to $z + 1$ at $v$. *Layer-dependent RC data* is defined by wire resistance and capacitance functions $R, C\colon E(Y) \times Z \to \mathbb{R}_{\geq 0}$ and a via resistance function $R\colon V(Y) \times Z \to \mathbb{R}_{\geq 0}$. Here, $R(e, z)$ and $C(e, z)$ for $(e, z) \in E(Y) \times Z$ denote the resistance and capacitance of the wire that results from assigning $e$ to $z$, and $R(v, z)$ for $(v, z) \in V(Y) \times Z$ denote the resistance of a via from $z$ to $z + 1$ at $v$. As a short form, we define an extension $f\colon V(Y) \times Z^2$ for any of the three via functions $f \in \{\text{price}, R, C\}$ by $f(v, z_1, z_2) := \sum_{z=\min\{z_1,z_2\}}^{\max\{z_1,z_2\}-1} f(v, z)$.

This definition is basically an adaption of Definition 3.2 to the scenario from this section. Here, the values price$(v, n_z)$, $R(v, n_z)$ and $C(v, n_z)$ for $v \in V(Y)$ are superfluous by definition, but we still keep them to simplify notation.
In our application, layer-dependent edge prices are composed of congestion and objective function prices, e.g. for the objective function of minimizing a weighted sum of wire length and via count. Subsequently, we laxly refer to them collectively as congestion prices, although in general any prices can be incorporated here as long as the total price of the route can be expressed as the sum of the prices of its wires and vias. However,

[3]Strictly speaking, in the degenerate case where many pins are located on the same position but on different layers (so they cannot be conflated), this construction can violate the assumption that the maximum vertex degree in $Y$ is bounded by a constant. This case can be incorporated into our algorithm from this section by making use of the fact that for any $v \in V(Y)$ all but a constant number of edges in $\delta_Y(v)$ are located on a fixed layer. However, to avoid overcomplicating the description, we neglect this degenerate case in this section.

this cannot be done for timing prices in most delay models, in particular not for the Elmore delay model from Section 3.4 that we are using. We address this issue later in Section 6.3.3.

We might also have prewires, in particular detailed wires, where the layer is fixed. This can be modeled by setting $Z_e = \{z_e\}$ for an edge $e$ corresponding to an input wire located on layer $z_e$, and defining $\text{price}(e, z_e)$, $R(e, z_e)$ and $C(e, z_e)$ to match the wire's properties. Input vias can be handled analogously. Moreover, also for global wires we might want to restrict the set of possible layers, e.g. in order to work only locally as described in Section 6.3.1.

The combination of a two-dimensional Steiner tree and a layer assignment naturally defines a three-dimensional Steiner tree:

Definition 6.5. Let $Y$ be a two-dimensional Steiner tree and $z\colon E(Y) \to Z$ be a layer assignment. Then the *three-dimensional tree according to* $z$ is denoted by $Y_z$. Given layer-dependent edge prices and layer-dependent RC data as in Definition 6.4, one can naturally define extensions $\text{price}, R, C\colon E(Y_z) \to \mathbb{R}_{\geq 0}$, e.g. in order to compute Elmore delays as in Definition 3.1.

Definition 6.5 leaves out the exact details on how to construct $Y_z$ and how to define the extensions of price, $R$ and $C$, but the process should be obvious. As the local topology at vias depends on the layer assignment, it is easier to express timing prices in terms of the resulting three-dimensional tree. This leads to the following problem formulation:

Problem 6.6: 2D Tree Layer Assignment Problem

**Input:** A two-dimensional Steiner tree $Y$ for a net $N$ with source $s \in N$ and sinks $T \subseteq N$, a source resistance $R(s) \in \mathbb{R}_{\geq 0}$, sink capacitances $C\colon T \to \mathbb{R}_{\geq 0}$, timing prices $\text{price}\colon T \to \mathbb{R}_{\geq 0}$, a set of layers $Z = \{1, \dots, n_z\}$, a non-empty set of allowed layers $Z_e \subseteq Z$ for every $e \in E(Y)$, layer-dependent edge prices and layer-dependent RC data as in Definition 6.4.

**Task:** Find a layer assignment $z\colon E(Y) \to Z$ with $z(e) \in Z_e$ for all $e \in E(Y)$ minimizing

$$\text{price}(z) := \text{price}(E(Y_z)) + \sum_{t \in T} \text{price}(t) \cdot d_{Y_z}(t),$$

where $Y_z$ is the three-dimensional tree according to $z$ as in Definition 6.5 and $d_{Y_z}(t)$ for $t \in T$ is the Elmore delay from $s$ to $t$ as in Definition 3.1.

$NP$-hardness of the 2D Tree Layer Assignment Problem follows directly from Theorem 5.3:

THEOREM 6.7. *The* 2D TREE LAYER ASSIGNMENT PROBLEM *is NP-hard even if* $|T| = 1$.

PROOF. This follows directly from Theorem 5.3, as when $n_y = 1$, the RC-AWARE PATH PROBLEM can be trivially reduced to the 2D TREE LAYER ASSIGNMENT PROBLEM. □

The next sections deal with finding an approximate solution for the 2D TREE LAYER ASSIGNMENT PROBLEM.

**6.3.3. Approximating Elmore Delay Prices.** To get a good approximate solution for the 2D TREE LAYER ASSIGNMENT PROBLEM we use an approximation of Elmore delay prices that is very similar to our RC prices from Section 5.3.2. As such, we use a similar line of thought as in Section 5.3.2 and derive similar results. However, as there are naturally some differences between the use of our RC prices in this section compared to Section 5.3.2, we give a standalone description here to keep the presentation simpler and self-contained.
Using a modified version of our RC prices from Section 5.3.2, we will be able to get modified layer-dependent edge prices that also include delay prices, which allows us to solve a simpler problem. Here, we make use of the following facts that are given in our situation:

(1) The two-dimensional outline of our Steiner tree is already fixed.
(2) Via capacitances are zero throughout our data set.
(3) Wire capacitances per unit length are usually not overly different across different layers for a given wire type.

Through item (1) the topology of the resulting three-dimensional tree is largely, but not entirely, independent of the layer assignment: There will be a one-to-one correspondence between $E(Y)$ and $E_{\text{wire}}(Y_z)$, but the number of vias and the local topologies at vias will be dependent on the actual layer assignment. (2) and (3) will help us to develop an algorithm that achieves a strong approximation guarantee for current technologies, as the approximation guarantee of our algorithm will depend on the error that we make when estimating downstream capacitances. We need the following definitions:

DEFINITION 6.8. Consider an instance of the 2D TREE LAYER ASSIGNMENT PROBLEM. *Minimum edge resistances and capacitances* $R_{\min}, C_{\min} \colon E(Y) \to \mathbb{R}_{\geq 0}$ are defined by

$$R_{\min}(e) := \min_{z_e \in Z_e} R(e, z_e), \qquad C_{\min}(e) := \min_{z_e \in Z_e} C(e, z_e),$$

for $e \in E(Y)$. This way, we can derive *minimum downstream capacitances*

$$C_{\min}(Y(v)) := C_{\min}(E(Y(v)) + C(T(Y(v))$$

for $v \in V(Y)$. For $e = (v, w) \in E(Y)$ and $z_e \in Z_e$ we define the *lower bound capacitance price* of $(e, z_e)$ by

$$\mathrm{cp}_{\mathrm{lb}}(e, z_e) := \mathrm{price}(T) \cdot R(s) + \mathrm{price}(T(Y(w))) \sum_{z'=\min\{z_s, z_e\}}^{\max\{z_s, z_e\}-1} \min_{x \in V(P_Y(s,v))} R(x, z')$$
$$+ \sum_{(x,y) \in E(P_Y(s,v))} \mathrm{price}(T(Y(y))) \cdot R_{\min}(x, y),$$

where $z_s \in Z$ is the fixed layer of the unique edge that is incident to $s$. Moreover, the *corrective capacitance* of $(e, z_e) \in E(Y) \times Z$ is defined as

$$C_{\mathrm{corr}}(e, z_e) := C(e, z_e) - C_{\min}(e).$$

Given a layer assignment $z \colon E(Y) \to Z$ and $e \in E_{\mathrm{wire}}(Y_z)$ located on layer $z_e$, we naturally extend $C_{\min}$, $\mathrm{cp}_{\mathrm{lb}}$ and $C_{\mathrm{corr}}$ by setting

$$C_{\min}(e) := C_{\min}(e'), \qquad \mathrm{cp}_{\mathrm{lb}}(e) := \mathrm{cp}_{\mathrm{lb}}(e', z_e), \qquad C_{\mathrm{corr}}(e) := C_{\mathrm{corr}}(e', z_e),$$

where $e' \in E(Y)$ is the two-dimensional edge corresponding to $e$. For $e \in E_{\mathrm{via}}(Y_z)$ we set $C_{\min}(e) = \mathrm{cp}_{\mathrm{lb}}(e) = C_{\mathrm{corr}}(e) = 0$. Moreover, for $(v, w) \in E_{\mathrm{wire}}(Y_z)$ we set its *capacitance price* to

$$\mathrm{cp}(v, w) := \mathrm{price}(T) \cdot R(s) + \sum_{(x,y) \in E(P_{Y_z}(s,v))} \mathrm{price}(T(Y_z(y))) \cdot R(x, y),$$

while we set $\mathrm{cp}(v, w) := 0$ for $(v, w) \in E_{\mathrm{via}}(Y_z)$.

It is straightforward to check that all values introduced by Definition 6.8 can be computed in a total of $\mathcal{O}(|V(Y)| \cdot |Z|)$ time. We note that it is not necessary to define meaningful minimum capacitances or capacitance prices for vias, as their capacitance is zero anyway. Compared to Section 5.3.2 the definition of corrective capacitances is simpler, as we cannot make detours when only computing a layer assignment. However, the lower bound capacitance prices are more complex, as we are also dealing with multi-sink nets in this chapter. Therefore, we need to multiply the minimum resistances in the definition of $\mathrm{cp}_{\mathrm{lb}}$ with varying prices, as varying sets of sinks are affected by the respective resistances. It is important to note that we have $\mathrm{cp}_{\mathrm{lb}}(v, w) \leq \mathrm{cp}(v, w)$ for all $(v, w) \in E_{\mathrm{wire}}(Y_z)$ (and trivially for all via edges), as we assume minimum resistances for edges in $Y$ and multiply the resistance of a possible via stack for reaching the layer of $(v, w)$ only by $\mathrm{price}(T(Y(w')))$ in the definition of $\mathrm{cp}_{\mathrm{lb}}$, where $(v', w') \in E(Y)$ is the two-dimensional edge corresponding to $(v, w)$. Again, as already elaborated on in Section 5.3.2.1, better lower bound capacitance prices $\mathrm{cp}_{\mathrm{lb}}(e, z_e)$, $e = (v, w) \in E(Y)$, $z_e \in Z$, could be obtained by making use of the fact that in order to use a certain layer $z \in Z$ for some edge in $E(P_Y(s, v))$, one would need a series of vias from $z_s$ to $z$ and from $z$ to $z_e$. With the help

of Definition 6.8 we can define modified edge prices for use during our layer assignment algorithm:

Definition 6.9. Consider an instance of the 2D Tree Layer Assignment Problem. For $e = (v, w) \in E(Y)$ and $z_e \in Z_e$ we set

$$\begin{aligned}\mathrm{rcp}_{\mathrm{wire}}(e, z_e) &:= \mathrm{price}(T(Y(w)) \cdot R(e, z_e) \cdot \left(\frac{C(e, z_e)}{2} + C_{\min}(Y(w))\right),\\ \mathrm{rcp}_{\mathrm{corr}}(e, z_e) &:= \mathrm{cp}_{\mathrm{lb}}(e, z_e) \cdot C_{\mathrm{corr}}(e, z_e),\\ \mathrm{rcp}(e, z_e) &:= \mathrm{price}(e, z_e) + \mathrm{rcp}_{\mathrm{wire}}(e, z_e) + \mathrm{rcp}_{\mathrm{corr}}(e, z_e).\end{aligned}$$

Given a layer assignment $z\colon E(Y) \to Z$ and $v \in V(Y)$, let $(u_1, w_1), \ldots, (u_k, w_k) \in E(Y_z)$ be the resulting via edges at $v$. Then we set

$$\mathrm{rcp}(v, z) := \mathrm{price}(v, z_{\min}(v), z_{\max}(v)) + \sum_{i=1}^{k} \mathrm{price}(T(Y_z(w_i))) \cdot R(u_i, w_i) \cdot C_{\min}(Y_z(w_i)),$$

and

$$\mathrm{rcp}(z) := \mathrm{price}(T) \cdot R(s) \cdot C_{\min}(Y(s)) + \sum_{v \in V(Y)} \mathrm{rcp}(v, z) + \sum_{e \in E(Y)} \mathrm{rcp}(e, z(e)).$$

We used $Y_z$ in order to simplify the definition of $\mathrm{rcp}(v, z)$ for $v \in V(Y)$, but $\mathrm{rcp}(v, z)$ actually only depends on the layer assignment of $\delta_Y(v)$. We will prove that a layer assignment minimizing rcp is a good approximation for a layer assignment minimizing price. Analogously to Section 5.3.2, the idea is that when we are dealing with an edge $(v, w) \in E(Y)$ in our layer assignment algorithm, we assume minimum downstream capacitances for $Y(w)$ in $\mathrm{rcp}_{\mathrm{wire}}$. This introduces an error, but the error is mitigated by the correction term $\mathrm{rcp}_{\mathrm{corr}}$, as quantified by Theorem 6.10. Here, and for the rest of this section, we define $0 \,/\, 0 := 1$ in order to simplify notation:

Theorem 6.10. *Consider an instance of the* 2D Tree Layer Assignment Problem. *Let $z$ be a layer assignment minimizing* rcp *and $z^*$ be a layer assignment minimizing* price. *Then we have*

$$\mathrm{price}(z) \leq \Big(1 + (\alpha - 1)(1 - \beta)\Big) \cdot \mathrm{price}(z^*),$$

*where*

$$\alpha := \max_{v \in V(Y_z)} \frac{C(Y_z(v))}{C_{\min}(Y_z(v))}, \qquad \beta := \min_{e \in E(Y_z)} \frac{\mathrm{cp}_{\mathrm{lb}}(e)}{\mathrm{cp}(e)}.$$

We can easily bound $\alpha$ by

$$\alpha \leq \max_{e \in E(Y_z)} \frac{C(e)}{C_{\min}(e)},$$

which results in strong approximation guarantees in practice due to the reasoning at the beginning of this section. An analogous bound for $\beta$ does not hold, as $Y_z$ may contain more vias than are accounted for in $\mathrm{cp}_{\mathrm{lb}}$. $\alpha$ and $\beta$ from Theorem 6.10 are similar, but not identical, to $\alpha$ and $\beta$ from Theorem 5.7. We therefore refer to Section 5.3.2.2 for an analysis of $\alpha$ and $\beta$, as the majority of this analysis also applies to our layer assignment scenario. We now continue with proving Theorem 6.10, but state a few lemmas first. We start with a simple identity:

LEMMA 6.11. *Consider an instance of the* 2D TREE LAYER ASSIGNMENT PROBLEM *and let $z$ be a layer assignment. Then for any $v \in V(Y_z)$ we have $C(Y_z(v)) = C_{\min}(Y_z(v)) + C_{\mathrm{corr}}(E(Y_z(v)))$.*

PROOF. We have

$$\begin{aligned} C(Y_z(v)) &= C(E(Y_z(v))) + C(T(Y_z(v))) + C_{\min}(E(Y_z(v))) - C_{\min}(E(Y_z(v))) \\ &= C_{\min}(Y_z(v)) + C_{\mathrm{corr}}(E(Y_z(v))). \end{aligned}$$

□

We continue with the relationship between rcp and price:

LEMMA 6.12. *Consider an instance of the* 2D TREE LAYER ASSIGNMENT PROBLEM *and let $z$ be a layer assignment. Then we have*

$$\mathrm{price}(z) = \mathrm{rcp}(z) + \sum_{e \in E(Y_z)} \Big(\mathrm{cp}(e) - \mathrm{cp}_{\mathrm{lb}}(e)\Big) \cdot C_{\mathrm{corr}}(e).$$

PROOF. Using Lemma 6.11 we get

$$\begin{aligned} &\sum_{t \in T} \mathrm{price}(t) \cdot d_{Y_z}(t) \\ &= \mathrm{price}(T) \cdot R(s) \cdot C(Y_z(s)) + \sum_{(v,w) \in E(Y_z)} \mathrm{price}(T(Y_z(w))) \cdot R(v,w) \left(\frac{C(v,w)}{2} + C(Y_z(w))\right) \\ &= \mathrm{price}(T) \cdot R(s) \cdot \Big(C_{\min}(Y_z(s)) + C_{\mathrm{corr}}(E(Y_z(s)))\Big) \\ &+ \sum_{(v,w) \in E(Y_z)} \mathrm{price}(T(Y_z(w))) \cdot R(v,w) \cdot \left(\frac{C(v,w)}{2} + C_{\min}(Y_z(w))\right) \\ &+ \sum_{(v,w) \in E(Y_z)} \mathrm{price}(T(Y_z(w))) \cdot R(v,w) \cdot C_{\mathrm{corr}}(E(Y_z(w))). \end{aligned}$$

Using Lemma 3.3 we can rearrange parts of the equation in the following way:

$$\begin{aligned}
&\mathrm{price}(T) \cdot R(s) \cdot C_{\mathrm{corr}}(E(Y_z(s))) + \sum_{(v,w)\in E(Y_z)} \mathrm{price}(T(Y_z(w))) \cdot R(v,w) \cdot C_{\mathrm{corr}}(E(Y_z(w))) \\
&= \sum_{(v,w)\in E(Y_z)} \Big(\mathrm{price}(T) \cdot R(s) + \sum_{(x,y)\in P_{Y_z}(s,v)} \mathrm{price}(T(Y_z(y))) \cdot R(x,y)\Big) \cdot C_{\mathrm{corr}}(v,w) \\
&= \sum_{e\in E(Y_z)} \mathrm{cp}(e) \cdot C_{\mathrm{corr}}(e) \\
&= \sum_{e\in E(Y_z)} \mathrm{cp}_{\mathrm{lb}}(e) \cdot C_{\mathrm{corr}}(e) + \sum_{e\in E(Y_z)} \Big(\mathrm{cp}(e) - \mathrm{cp}_{\mathrm{lb}}(e)\Big) \cdot C_{\mathrm{corr}}(e).
\end{aligned}$$

Combining this with our initial calculations we get

$$\begin{aligned}
&\sum_{t\in T} \mathrm{price}(t) \cdot d_{Y_z}(t) \\
&= \mathrm{price}(T) \cdot R(s) \cdot C_{\mathrm{min}}(Y_z(s)) \\
&+ \sum_{(v,w)\in E(Y_z)} \mathrm{price}(T(Y_z(w))) \cdot R(v,w) \cdot \left(\frac{C(v,w)}{2} + C_{\mathrm{min}}(Y_z(w))\right) \\
&+ \sum_{e\in E(Y_z)} \mathrm{cp}_{\mathrm{lb}}(e) \cdot C_{\mathrm{corr}}(e) + \sum_{e\in E(Y_z)} \Big(\mathrm{cp}(e) - \mathrm{cp}_{\mathrm{lb}}(e)\Big) \cdot C_{\mathrm{corr}}(e) \\
&= \mathrm{price}(T) \cdot R(s) \cdot C_{\mathrm{min}}(Y_z(s)) \\
&+ \sum_{e\in E(Y)} \mathrm{rcp}_{\mathrm{wire}}(e, z(e)) + \mathrm{rcp}_{\mathrm{corr}}(e, z(e)) \\
&+ \sum_{(v,w)\in E_{\mathrm{via}}(Y_z)} \mathrm{price}(T(Y_z(w))) \cdot R(v,w) \cdot C_{\mathrm{min}}(Y_z(w)) \\
&+ \sum_{e\in E(Y_z)} \Big(\mathrm{cp}(e) - \mathrm{cp}_{\mathrm{lb}}(e)\Big) \cdot C_{\mathrm{corr}}(e) \\
&= \mathrm{rcp}(z) - \mathrm{price}(E(Y_z)) + \sum_{e\in E(Y_z)} \Big(\mathrm{cp}(e) - \mathrm{cp}_{\mathrm{lb}}(e)\Big) \cdot C_{\mathrm{corr}}(e),
\end{aligned}$$

and this was to show. □

We get a simple corollary:

COROLLARY 6.13. *Consider an instance of the* 2D TREE LAYER ASSIGNMENT PROBLEM *and let $z$ be a layer assignment. Then* $\mathrm{rcp}(z) \leq \mathrm{price}(z)$ *holds.*

PROOF. This follows directly from Lemma 6.12, as we have $\mathrm{cp}_{\mathrm{lb}}(e) \leq \mathrm{cp}(e)$ and $C_{\mathrm{corr}}(e) \geq 0$ for all $e \in E(Y_z)$. □

We are now able to prove Theorem 6.10:

PROOF OF THEOREM 6.10. We prove

$$\text{price}(z) \leq \Big(1 + (\alpha - 1)(1 - \beta)\Big) \cdot \text{rcp}(z) \leq \Big(1 + (\alpha - 1)(1 - \beta)\Big) \cdot \text{price}(z^*).$$

The second inequality follows directly from Corollary 6.13, as this gives us $\text{rcp}(z) \leq \text{rcp}(z^*) \leq \text{price}(z^*)$. For proving the first inequality we have to show

$$\sum_{e \in E(Y_z)} \Big(\text{cp}(e) - \text{cp}_{\text{lb}}(e)\Big) \cdot C_{\text{corr}}(e) \leq (\alpha - 1)(1 - \beta) \cdot \text{rcp}(z) \tag{6.1}$$

according to Lemma 6.12. We note that we have

$$C(Y_z(v)) - C_{\min}(Y_z(v)) \leq (\alpha - 1)\, C_{\min}(Y_z(v)) \qquad \forall\, v \in V(Y_z), \tag{6.2}$$

$$\text{cp}(e) - \text{cp}_{\text{lb}}(e) \leq (1 - \beta)\, \text{cp}(e) \qquad \forall\, e \in E(Y_z), \tag{6.3}$$

by definition of $\alpha$ and $\beta$, and Lemma 6.11 immediately gives us

$$C_{\text{corr}}(E(Y_z(v))) \leq (\alpha - 1)\, C_{\min}(Y_z(v)) \qquad \forall\, v \in V(Y_z) \tag{6.4}$$

from (6.2). Due to (6.3), showing (6.1) reduces to showing

$$\sum_{e \in E(Y_z)} \text{cp}(e) \cdot C_{\text{corr}}(e) \leq (\alpha - 1) \cdot \text{rcp}(z).$$

This is done by the following calculations with the help of (6.4) and Lemma 3.3:

$$\begin{aligned}
&\sum_{e \in E(Y_z)} \text{cp}(e) \cdot C_{\text{corr}}(e) \\
&= \sum_{(v,w) \in E(Y_z)} \Big(\text{price}(T) \cdot R(s) + \sum_{(x,y) \in P_{Y_z}(s,v)} \text{price}(T(Y_z(y))) \cdot R(x,y)\Big) \cdot C_{\text{corr}}(v,w) \\
&= \text{price}(T) \cdot R(s) \cdot C_{\text{corr}}(E(Y_z(s))) + \sum_{(v,w) \in E(Y_z)} \text{price}(T(Y_z(w))) \cdot R(v,w) \cdot C_{\text{corr}}(E(Y_z(w))) \\
&\leq (\alpha - 1)\Big(\text{price}(T) \cdot R(s) \cdot C_{\min}(Y_z(s)) + \sum_{(v,w) \in E(Y_z)} \text{price}(T(Y_z(w))) \cdot R(v,w) \cdot C_{\min}(Y_z(w))\Big) \\
&\leq (\alpha - 1) \cdot \text{rcp}(z).
\end{aligned}$$

□

**6.3.4. Layer Assignment by Dynamic Programming.** We now show how to find a layer assignment minimizing our RC prices from Section 6.3.3. To achieve this we use a dynamic programming approach which is basically an improved and extended version of the one presented by Michaelis [80], who also considers congestion prices, but not RC delays. We introduce the notion of solution candidates in Section 6.3.4.1, present an outline of the overall layer assignment algorithm in Section 6.3.4.2 and answer the key question of how to compute solution candidates in Section 6.3.4.3.

6.3.4.1. *Solution Candidates.* In our dynamic program we compute *solution candidates* for the edges in $E(Y)$, which correspond to optimum partial layer assignments. We first define RC prices for the kinds of partial layer assignments that we are interested in:

Definition 6.14. Consider an instance of the 2D Tree Layer Assignment Problem, and let $v \in V(Y)$ with $\delta_Y^-(v) = \{e\}$, $e = (u, v)$. Then a *partial layer assignment* at $v$ is a layer assignment $z\colon E(Y(v)) \cup \{e\} \to Z$, and the *partial RC price* of $z$ is given as

$$\operatorname{rcp}(e, z(e)) + \sum_{w \in V(Y(v))} \operatorname{rcp}(w, z) + \sum_{e' \in E(Y(v))} \operatorname{rcp}(e', z(e')).$$

Partial RC prices are well-defined as they only depend on the layers of $E(Y(v)) \cup \{e\}$. We can now define solution candidates:

Definition 6.15. Consider an instance of the 2D Tree Layer Assignment Problem. A solution candidate $\operatorname{cand}(e, z_e)$ for $e = (u, v) \in E(Y)$ and $z_e \in Z_e$ is a partial layer assignment $z\colon E(Y(v)) \cup \{e\} \to Z$ with $z(e) = z_e$ that is minimal with regard to partial RC prices among all such layer assignments. As a notational convenience, we associate $\operatorname{cand}(e, z_e)$ not only with the layer assignment but also with its partial RC price.

6.3.4.2. *Layer Assignment Algorithm.* The notion of solution candidates allows us to formulate our layer assignment algorithm, which is given as Algorithm 4. In line 4

**Algorithm 4** 2D Tree Layer Assignment Algorithm

**Input:** An instance of the 2D Tree Layer Assignment Problem.
**Output:** A layer assignment $z\colon E(Y) \to Z$.
1: **for** all $e \in E(Y)$ in reverse topological order **do**
2: **for** all $z_e \in Z_e$ **do**
3: compute $\operatorname{cand}(e, z_e)$.
4: Let $e_s$ be the unique edge in $\delta_Y^+(s)$ and $z_s$ be its fixed layer.
5: **return** $\operatorname{cand}(e_s, z_s)$

we make use of the fact that $s$ is a leaf in $Y$ with $|Z_{e_s}| = 1$ for the unique outgoing edge $e_s$ of $s$. By definition of cand, the algorithm returns the correct solution. Of course, the crucial question is how to do the candidate computations in line 3.

6.3.4.3. *Computing Candidates.* Consider the computation of $\operatorname{cand}(e, z_e)$ for $e = (u, v) \in E(Y)$ and $z_e \in Z_e$. Let $\delta_Y^+(v) = \{e_1, \dots, e_d\}$ with $e_i = (v, w_i)$, $i = 1, \dots, d$. As discussed in the beginning of this section, $d$ can be considered as a constant. Since we are processing the edges of $Y$ in reverse topological order during the course of Algorithm 4, we can assume that $\operatorname{cand}(e_i, z_{e_i})$ has already been computed for every $e_i = (v, w_i) \in$

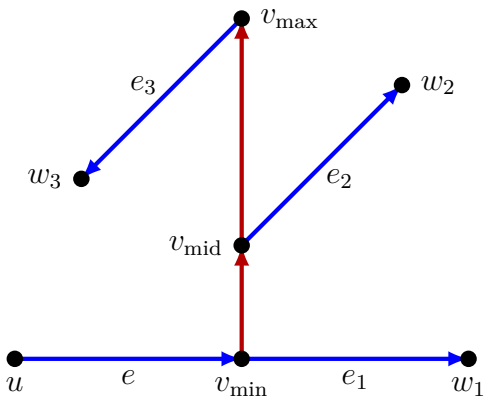



FIGURE 6.3. Illustration of the situation from Section 6.3.4.3: We are given an edge $e = (u, v) \in E(Y)$ with $\delta_Y^+(v) = \{e_1, e_2, e_3\}$, $e_i = (v, w_i), i = 1, \ldots, 3$, and a layer assignment $z\colon E(Y) \to Z$ obeying the given fixed layer order $e \preceq e_1 \preceq e_2 \preceq e_3$. In the three-dimensional tree $Y_z$ according to $z$, there are a via (red) and three vertices $v_{\min}$, $v_{\text{mid}}$ and $v_{\max}$ corresponding to $v$ (and possibly multiple vertices corresponding to $u$, $w_1$, $w_2$ and $w_3$ not shown in the picture). Arrows indicate edge directions in $Y_z$, and one can see that the local topology at the via depends on $z$. However, fixing the layer order to $e \preceq e_1 \preceq e_2 \preceq e_3$ already determines the local topology at the via to a large degree.

$\delta_Y^+(v)$ and $z_{e_i} \in Z_{e_i}$. According to Definition 6.14, $\text{cand}(e, z_e)$ is determined in the following way:

$$\text{cand}(e, z_e) = \min\{\text{rcp}(e, z_e) + \text{rcp}(v, z) + \sum_{i=1}^{d} \text{cand}(e_i, z(e_i)) : \\ z\colon \delta_Y(v) \to Z \text{ with } z(e) = z_e, z(e_i) \in Z_{e_i}, i = 1, \ldots, d\}.$$

To compute $\text{cand}(e, z_e)$ we compute an optimum partial layer assignment for all possible orders of $z_e, z(e_1), \ldots, z(e_d)$ — as $d$ is constant, there is only a constant number of those. To simplify the description, we only elaborate on the case $e \preceq e_1 \preceq \ldots \preceq e_d$, where this notation means that we restrict ourselves to layer assignments $z$ where $z_e \leq z(e_1) \leq \ldots \leq z(e_d)$. It will be easy to see that all other orders can be handled in the same manner. In particular, for any order where $e_i \preceq e \preceq e_j$ for some $i, j \in \{1, \ldots, d\}$, the parts below and above $z_e$ can be dealt with independently. Some orders may not yield a feasible solution due to the requirement that $z(e_i) \in Z_{e_i}$ for $i = 1, \ldots, d$. These cases will be identified by our algorithm. An illustration of the setting of this section is given by Figure 6.3.

As stated earlier, the delay prices through the via at $v$ depend on the local topology at the via, which in turn depends on the layer assignment of $\delta_Y(v)$. However, if the layer order is fixed, we can express the delay prices through the via in a separable fashion:

DEFINITION 6.16. Consider an instance of the 2D TREE LAYER ASSIGNMENT PROBLEM. Let $v \in V(Y)$ with $\delta_Y^-(v) = \{e\}$ and $\delta_Y^+(v) = \{e_1, \dots, e_d\}$, $e_i = (v, w_i)$, $i = 1, \dots, d$. Let $z_e \in Z_e$ and consider the fixed layer order $e \preceq e_1 \preceq \dots \preceq e_d$. For $i = 1, \dots, d$ let $\alpha_i := \text{price}(T(Y(w_i)))$ and $\beta_i := C_{\min}(e_i) + C_{\min}(Y(w_i))$. Then for $e_i \in \delta_Y^+(v)$ and $z_{e_i} \in Z_{e_i}$ we define the *via delay price* of assigning $e_i$ to $z_{e_i}$ as

$$\text{vdp}(e_i, z_{e_i}) := R(v, z_e, z_{e_i}) \cdot \Big(\alpha_i \cdot \beta_i + \sum_{j=i+1}^{d} (\alpha_i \cdot \beta_j + \beta_i \cdot \alpha_j)\Big).$$

As $d$ is considered to be constant, it is possible to compute $\text{vdp}(e_i, z_{e_i})$ for all pairs $(e_i, z_{e_i}) \in \delta_Y^+(v) \times Z$ in a total of $\mathcal{O}(|Z|)$ time. The meaning of vdp becomes clear through the following proposition:

PROPOSITION 6.17. *Consider an instance of the* 2D TREE LAYER ASSIGNMENT PROBLEM. *Let $v \in V(Y)$ with $\delta_Y^-(v) = \{e\}$ and $\delta_Y^+(v) = \{e_1, \dots, e_d\}$, $e_i = (v, w_i), i = 1, \dots, d$. Let $z_e \in Z_e$ and consider the fixed layer order $e \preceq e_1 \preceq \dots \preceq e_d$. Let $z\colon \delta_Y(v) \to Z$ be a partial layer assignment obeying this order with $z(e) = z_e$. Then*

$$\text{rcp}(v, z) = \text{price}(v, z_e, z(e_d)) + \sum_{i=1}^{d} \text{vdp}(e_i, z(e_i)).$$

PROOF. For $i = 1, \dots, d$ let $\alpha_i$ and $\beta_i$ as in Definition 6.16 and define

$$\gamma_i := \alpha_i \cdot \beta_i + \sum_{j=i+1}^{d} \Big(\alpha_i \cdot \beta_j + \beta_i \cdot \alpha_j\Big),$$

i.e.

$$\text{vdp}(e_i, z(e_i)) = R(v, z_e, z(e_i)) \cdot \gamma_i.$$

Moreover, for any $k \in \{1, \dots, d\}$ we have

$$\sum_{i=k}^{d} \gamma_i = \sum_{i=k}^{d} \alpha_i\beta_i + \sum_{i=k}^{d} \sum_{j=i+1}^{d} (\alpha_i\beta_j + \beta_i\alpha_j) = \sum_{i=k}^{d} \alpha_i \sum_{j=k}^{d} \beta_j.$$

For $z' \in \{z_e, \dots, z(e_d) - 1\}$ let $f(z') = (a(z'), b(z')) \in E(Y_z)$ be the via edge corresponding to $v$ between layers $z'$ and $z' + 1$ (after subdividing these via edges appropriately). Here, we can assume $z(e_d) > z_e$, as otherwise there is nothing to show. Moreover, let

$k(z') := \min\{i \in \{1,\dots,d\} : z(e_i) > z'\}$ for $z' \in Z$. Then we have

$$\begin{aligned}
\sum_{i=1}^{d} \mathrm{vdp}(e_i, z(e_i)) &= \sum_{i=1}^{d} R(v, z_e, z(e_i)) \cdot \gamma_i \\
&= \sum_{z'=z_e}^{z(e_d)-1} R(v, z') \cdot \sum_{i=k(z')}^{d} \gamma_i \\
&= \sum_{z'=z_e}^{z(e_d)-1} R(v, z') \cdot \sum_{i=k(z')}^{d} \alpha_i \sum_{j=k(z')}^{d} \beta_j \\
&= \sum_{z'=z_e}^{z(e_d)-1} R(f(z')) \cdot \mathrm{price}(T(Y_z(b(z')))) \cdot C_{\min}(Y_z(b(z'))) \\
&= \mathrm{rcp}(v, z) - \mathrm{price}(v, z_e, z(e_d)),
\end{aligned}$$

and this was to be proven. □

We can now state Algorithm 5, which computes an optimum partial layer assignment for our given layer order. As a notational convenience we set $\mathrm{cand}(e, z_e) := \infty$ for any $e \in E(Y)$ and $z_e \in Z \setminus Z_e$.

**Algorithm 5** Fixed Layer Order Partial Layer Assignment Algorithm

**Input:** An instance of the 2D Tree Layer Assignment Problem, $v \in V(Y)$ with $\delta_Y^-(v) = \{e\}$, $\delta_Y^+(v) = \{e_1, \dots, e_d\}$, $e_i = (v, w_i), i = 1, \dots, d$, a fixed layer order $e \preceq e_1 \preceq \dots \preceq e_d$, a fixed layer $z_e \in Z_e$ for $e$.

**Output:** A partial layer assignment $z\colon E(Y(v)) \cup \{e\} \to Z$ with $z(e) = z_e$.

1: **for** $k = 0, \dots, d$ **do**
2: $\quad \mathrm{OPT}(k, z_e) := \mathrm{rcp}(e, z_e) + \sum_{i=1}^{k} \mathrm{cand}(e_i, z_e)$.
3: **for** $z' = z_e + 1, \dots, n_z$ **do**
4: $\quad$ **for** $k = 0, \dots, d$ **do**
5: $$\mathrm{OPT}(k, z') = \min_{i=0,\dots,k} \Big(\mathrm{OPT}(i, z'-1) + \sum_{j=i+1}^{k} \mathrm{vdp}(e_j, z') + \mathrm{cand}(e_j, z')\Big) + \mathrm{price}(v, z'-1).$$
6: **return** $\min_{z'=z_e,\dots,n_z} \mathrm{OPT}(d, z')$

Theorem 6.18. *If Algorithm 5 returns a finite value, then it computes a partial layer assignment $z\colon E(Y(v)) \cup \{e\} \to Z$ minimizing partial RC prices among all such partial layer assignments with $z(e) = z_e$ and layer order $e \preceq e_1 \preceq \dots \preceq e_d$.*
*If Algorithm 5 returns an infinite value, then there exists no partial layer assignment $z\colon E(Y(v)) \cup \{e\} \to Z$ with $z(e) = z_e$ and layer order $e \preceq e_1 \preceq \dots \preceq e_d$.*

PROOF. For $k \in \{0, \dots, d\}$ and $z' \in Z$ let a *$k$-partial layer assignment with maximum layer $z'$* be a layer assignment $z\colon \left(\bigcup_{i=1}^{k} E(Y(w_i)) \cup \{e_i\}\right) \cup \{e\} \to Z$ with $z(e_i) \in Z_{e_i} \cap \{z_e, \dots, z'\}$, $i = 1, \dots, k$, and $z(e) = z_e$. Let the partial RC price of such a $k$-partial layer assignment be defined as

$$\operatorname{rcp}(e, z_e) + \operatorname{price}(v, z_e, z') + \sum_{i=1}^{k} \operatorname{vdp}(e_i, z(e_i)) + \operatorname{cand}(e_i, z(e_i)).$$

In particular, any $d$-partial layer assignment is a partial layer assignment in the sense of Definition 6.14. Moreover, for any optimum partial layer assignment $z^*\colon E(Y(v)) \cup \{e\} \to Z$ with $z^*(e) = z_e$ we have

$$\begin{aligned}
&\operatorname{rcp}(e, z^*(e)) + \sum_{w \in V(Y(v))} \operatorname{rcp}(w, z^*) + \sum_{e' \in E(Y(v))} \operatorname{rcp}(e', z^*(e')) \\
&= \operatorname{rcp}(e, z_e) + \operatorname{rcp}(v, z^*) + \sum_{i=1}^{d} \operatorname{cand}(e_i, z^*(e_i)) \\
&= \operatorname{rcp}(e, z_e) + \operatorname{price}(v, z_e, z^*(e_d)) + \sum_{i=1}^{d} \operatorname{vdp}(e_i, z^*(e_i)) + \operatorname{cand}(e_i, z^*(e_i)),
\end{aligned}$$

where we used Proposition 6.17 to get the last equality. Let $\operatorname{OPT}^*(k, z')$ for $(k, z') \in \{1, \dots, d\} \times Z$ be the minimum partial RC price of any $k$-partial layer assignment with maximum layer $z'$ if such a layer assignment exists, and $\operatorname{OPT}^*(k, z') = \infty$ otherwise. We claim that $\operatorname{OPT}^*$ equals OPT from Algorithm 5. In that case, Algorithm 5 returns the correct solution.

We certainly have $\operatorname{OPT}(k, z_e) = \operatorname{OPT}^*(k, z_e)$ for $k = 0, \dots, d$ after initialization in line 2. So let $z' > z_e$ and $k \in \{0, \dots, d\}$. We first note that $\operatorname{OPT}(k, z') = \infty$ if and only if at least one of the following conditions holds for all $i \in \{0, \dots, k\}$:

(1) $\operatorname{OPT}(i, z' - 1) = \infty$, i.e. no feasible layer assignment of $\{e_1, \dots, e_i\}$ in the layer range $\{z_e, \dots, z' - 1\}$ exists (by induction),
(2) $\operatorname{cand}(e_j, z') = \infty$ for some $j \in \{i + 1, \dots, k\}$, i.e. $z' \notin Z_{e_j}$.

Now assume that (1) and (2) do not hold. In that case $\operatorname{OPT}(k, z')$ can be associated with an index $i \in \{0, \dots, k\}$ such that $\{e_1, \dots, e_i\}$ are assigned optimally within the layer range $\{z_e, \dots, z' - 1\}$, and $\{e_{i+1}, \dots, e_k\}$ are assigned to $z'$. Therefore, using $\operatorname{OPT}^*(i, z' - 1) = \operatorname{OPT}(i, z' - 1)$ for all $i = 0, \dots, k$ by induction, line 5 correctly computes $\operatorname{OPT}^*(k, z') = \operatorname{OPT}(k, z')$ by taking the minimum over all such possible choices of $i \in \{0, \dots, k\}$. □

The running time analysis is simple:

PROPOSITION 6.19. *Algorithm 5 can be implemented in $\mathcal{O}(|Z|)$ time.*

PROOF. We first recall that by our assumptions from the beginning of this section, $d$ can be considered as a constant. The via delay prices $\mathrm{vdp}(e_i, z_{e_i})$, $i = 1, \ldots, d$, $z_{e_i} \in Z_{e_i}$, can be computed in a total of $\mathcal{O}(|Z|)$ time. Therefore, as lines 2 and 5 take constant time, the total running time of Algorithm 5 amounts to $\mathcal{O}(|Z|)$. □

We can now formulate the main results of this chapter:

COROLLARY 6.20. *Algorithm 4 can be implemented in $\mathcal{O}(|E(Y)| \cdot |Z|^2)$ time.*

PROOF. We show how to implement line 3 for $e = (u, v) \in E(Y)$ in $\mathcal{O}(|Z|)$ time: We apply Algorithm 5 to all possible layer orders of $\delta_Y(v)$. As $|\delta_Y(v)|$ is constant, there is only a constant number of such orders. Using Proposition 6.19, the total running time for one invocation of line 3 of Algorithm 4 then amounts to $\mathcal{O}(|Z|)$ time, and the claimed total running time for Algorithm 4 follows. □

THEOREM 6.21. *Algorithm 4 is an $\big(1 + (\alpha - 1)(1 - \beta)\big)$-approximation algorithm for the* 2D TREE LAYER ASSIGNMENT PROBLEM*, where $\alpha$ and $\beta$ are defined as in Theorem 6.10.*

PROOF. By definition of cand, Algorithm 4 returns a layer assignment minimizing rcp. The proclaimed approximation bound then follows from Theorem 6.10. □

At this point we also mention the fully polynomial time approximation scheme for embedding a fixed Steiner tree topology into a graph given by Hähnle and Rotter [97], which we shortly introduce in Section 5.3.2. This also yields a fully polynomial time approximation scheme for a slightly modified version of the 2D TREE LAYER ASSIGNMENT PROBLEM where $x$- and $y$-coordinates of Steiner vertices are not completely fixed (they are movable within the graph). However, as for the RC-AWARE PATH PROBLEM, the running time (5.3) of the approximation scheme is most likely too large for using it for more than a very small fraction of nets in our application. Therefore, our algorithm presented in this section seems more suitable for large-scale application in practice, as it should be sufficiently fast for practical purposes.

**6.3.5. A Hardness Result Regarding Unbounded Vertex Degrees.** During the development of Algorithm 4 we assumed that the maximum vertex degree in $Y$ is bounded by a constant. Without this requirement, Algorithm 4 would not be a polynomial time algorithm, as we consider every possible layer order of $\delta_Y(v)$ when computing $\mathrm{cand}(e, z_e)$ for $e = (u, v)$ and $z_e \in Z_e$ in line 3. We now show that assuming the maximum vertex degree to be bounded is indeed necessary for computing a layer assignment minimizing rcp:

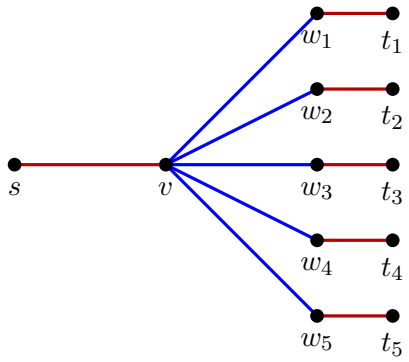



FIGURE 6.4. The Steiner tree $Y$ from the proof of Theorem 6.22. Red edges are fixed on layer 2, while blue edges may be assigned to layer 1 or 3. Layer-dependent edge prices and RC data are consistently zero except for vias at $v$, which have a positive resistance.

THEOREM 6.22. *Consider the version of the* 2D TREE LAYER ASSIGNMENT PROBLEM *where $Y$ is allowed to have unbounded vertex degrees. Then computing a layer assignment minimizing* rcp *is NP-hard.*

We use a reduction of the well-known $NP$-complete PARTITION PROBLEM [70], whose formulation is already given in Section 5.3.1. We need the following trivial lemma for the proof of Theorem 6.22:

LEMMA 6.23. *Consider the function $f\colon \mathbb{R} \to \mathbb{R}$ defined by $f(x) = x^2 + (S-x)^2$ for some $S \in \mathbb{R}$. Then $f$ attains its unique minimum for $x = S/2$.*

PROOF. We have

$$f(x) = x^2 + (S-x)^2 = 2x^2 - 2Sx + S^2 = 2(x - S/2)^2 + S/2.$$

□

PROOF OF THEOREM 6.22. Let $a_1, \ldots, a_n \in \mathbb{Q}_{>0}$ be an instance of the PARTITION PROBLEM. We construct an instance of the 2D TREE LAYER ASSIGNMENT PROBLEM as depicted by Figure 6.4:

- $V(Y) = \{s, v\} \cup \{w_1, \ldots, w_n\} \cup T$ with $T = \{t_1, \ldots, t_n\}$,
- $E(Y) = \{(s, v)\} \cup \{(v, w_i), (w_i, t_i) : i = 1, \ldots, n\}$,
- $Z = \{1, 2, 3\}$,
- $Z_e = \{1, 3\}$ if $e = (v, w_i)$ for $i \in \{1, \ldots, n\}$, $Z_e = \{2\}$ for all other $e \in E(Y)$,
- $R(s) = 0$, $\text{price}(t_i) = C(t_i) = a_i, i = 1, \ldots, n$,
- $\text{price}(e, z_e) = R(e, z_e) = C(e, z_e) = 0$ for all $(e, z_e) \in E(Y) \times Z$,
- $\text{price}(x, z_x) = 0$ for all $(x, z_x) \in V(Y) \times Z$,

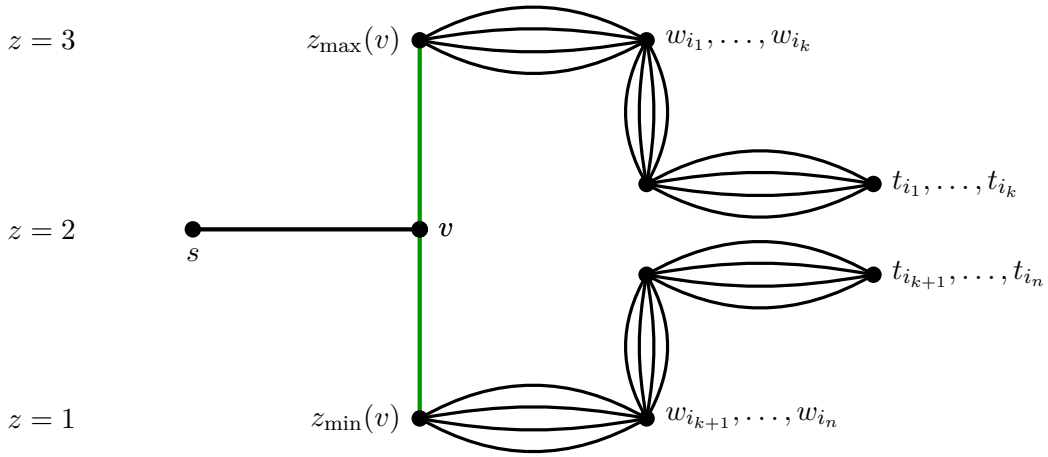



FIGURE 6.5. Illustration of a layer assignment corresponding to a partition $I = \{i_1, \ldots, i_k\}$ and $I^c = \{i_{k+1}, \ldots, i_n\}$ in the proof of Theorem 6.22. Vertical positions of vertices and edges indicate their $z$-coordinates. Edges incident to sinks in $T$ are located on layer 2, but their vertical positions are slightly displaced for illustrative purposes. Bundles of edges indicate that the paths starting in $z_{\min}(v)$ and $z_{\max}(v)$ and ending in $T$ are disjoint. The price of the layer assignment equals the delay price through the green via at $v$, and is therefore fully determined by $I$ and $I^c$.

- $R(v, z_v) = 1$ for all $z_v \in Z$, $R(x, z_x) = 0$ for $(x, z_x) \in (V(Y) \setminus \{v\}) \times Z$.

We note that the edges $(s, v)$ and $(w_i, t_i)$, $i = 1, \ldots, n$, exist due to our convention that pins are leaves in $Y$ that are incident to an edge on a fixed layer (representing the layer of the pin).

Every layer assignment for this instance of the 2D TREE LAYER ASSIGNMENT PROBLEM is fully determined by which of the edges $(v, w_i)$, $i = 1, \ldots, n$, are assigned to either layer 1 or 3. In that sense, let $z$ be a layer assignment, and let $I = \{i \in \{1, \ldots, n\} : z(v, w_i) = 3\}$ and $I^c = \{1, \ldots, n\} \setminus I$. An illustration is given by Figure 6.5. Then we have

$$\operatorname{rcp}(z) = \sum_{i \in I} \operatorname{price}(t_i) \sum_{i \in I} C(t_i) + \sum_{i \in I^c} \operatorname{price}(t_i) \sum_{i \in I^c} C(t_i) = \left( \sum_{i \in I} a_i \right)^2 + \left( \sum_{i \in I^c} a_i \right)^2 .$$

Letting $S := \sum_{i=1}^{n} a_i$, it follows from Lemma 6.23 that there exists an optimum layer assignment $z$ with $\operatorname{rcp}(z) = S^2 / 2$ if and only if $a_1, \ldots, a_n$ describe a yes-instance of the partition problem. □

## 6.4. Implementation in BonnRouteGlobal

As already mentioned, this chapter describes a three-step approach for transforming a route that is connecting projected shapes to one connecting exact shapes. At the time of this writing, only two of these three steps, namely the ones outlined in Sections 6.1 and 6.2, are implemented and fully functional in BonnRouteGlobal. The layer assignment step from Section 6.3 is currently being implemented.

However, as we still need to connect to exact shapes for our experimental results in Chapters 5 and 7, e.g. in order to be able to do timing computations, we replace the layer assignment step outlined in Section 6.3 by a heuristic method: We first restrict ourselves to working locally as described in Section 6.3.1, and then do a heuristic layer assignment for tile-internal segments. As this is only an intermediate solution, we omit the description of this heuristic layer assignment method at this point. Here, it is important to emphasize that as we are restricting ourselves to working locally, the heuristic layer assignment only has local effects — the global layer assignment of long wires is therefore still determined by the respective routing methods described in Chapters 5 and 7.

CHAPTER 7

# Routing Based Optimization

After the main global routing phase has finished and all nets are routed, one could directly proceed with detailed routing. Although this is a feasible approach that has been used successfully in the past, we present a new routing flow where we include an additional timing optimization step called *Global Routing Based Optimization (GRBO)* inbetween global and detailed routing. Moreover, we add a second routing based timing optimization step called *Detailed Routing Based Optimization (DRBO)* directly after detailed routing. This results in the routing flow depicted in Figure 7.1. This flow has been developed in collaboration with Michael Kazda and his team from IBM, who set up the optimization flow using BonnRouteGlobal.
The main focus of this chapter is our incremental global router, which we call Incremental BonnRouteGlobal. Its task is to incrementally change the existing routing in order to

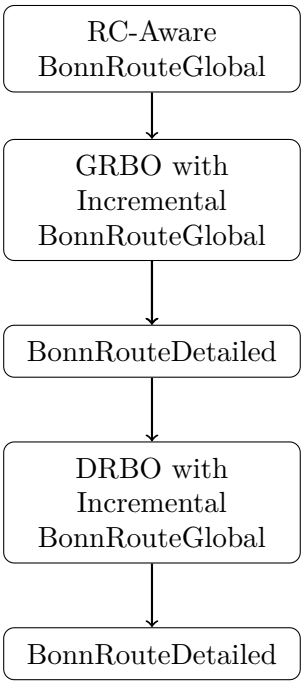



FIGURE 7.1. A simple sketch of our routing flow.

adapt to the changes that are induced by the timing optimization tools running in GRBO and DRBO. Due to this role, the incremental router is one of the central components of GRBO and DRBO and constitutes the main difference between RBO and the timing optimization steps running in the VLSI design flow prior to RBO. GRBO and DRBO are also incorporated and broadly used in the IBM production flow.

This chapter is structured as follows: Section 7.1 and 7.2 are overview chapters. The former gives an overview on GRBO and DRBO in general, and the latter describes the transaction framework that is used to control Incremental BonnRouteGlobal. The next three sections then describe features of Incremental BonnRouteGlobal: In Section 7.3 we describe the process that we use to complete the routing of partially wired nets, which goes under the name *minimal reroutes*. Subsequently, we turn our attention to the situation where buffers are inserted into an already routed net, and the problem of distributing the wiring into the resulting subnets. This problem is referred to as the *Copy Routes Problem*, and it is covered in Section 7.4. The last feature of Incremental BonnRouteGlobal that we describe in this thesis is its multi-threaded implementation, and the description is given in Section 7.5. Sections 7.3 – 7.5 contain experimental results tailored to the respective topic, and as a summary of that, we present experimental results showing the effectiveness of our whole new routing flow in Section 7.6.

The incremental routing framework utilized in this chapter including the code to run the various timing optimization steps is provided by IBM. Our main contributions to the routing flow depicted in Figure 7.1 are the parts related to BonnRouteGlobal, i.e. RC-Aware BonnRouteGlobal from Chapters 4 and 5 and Incremental BonnRouteGlobal from this chapter.

## 7.1. GRBO and DRBO

GRBO and DRBO constitute timing optimization flows (cf. Section 1.2.2) that are built around Incremental BonnRouteGlobal. This means that whenever pin positions are changed or nets are created during timing optimization, we use our incremental global router to complete the routing. Except for few detailed wires that are possibly in the input to global routing, we almost exclusively deal with global wires during GRBO. In contrast to that, we start with a fully detailed routed design in DRBO. Our incremental global router then adds global wires and removes no longer required detailed wires to connect nets that are modified during DRBO. Therefore, we work with a mix of global and detailed wires during DRBO.

In the remainder of this section we first motivate adding the GRBO and DRBO steps to our routing flow in Section 7.1.1. Afterwards, we give a cursory overview on the timing

optimization flows that we use in GRBO and DRBO in Section 7.1.2. At last, we turn our attention to the problem of estimating routing capacities for DRBO in Section 7.1.3.

**7.1.1. Motivation.** In the subsequent paragraphs we shortly reason about why we are adding GRBO and DRBO to our routing flow. Of course, the crucial point is that it gives good results in practice, as shown in Section 7.6. However, we want to elaborate more on the reasons behind this.

7.1.1.1. *Motivating GRBO.* The main reason for adding the GRBO step to our routing flow is as follows: While an extensive amount of timing optimization is already being done prior to global routing, this timing optimization has to use wiring estimates. More precisely, in our VLSI design flow every net is estimated to be wired as an approximately shortest Steiner tree that is embedded on the two lowest layers of the layer assignment of the given net (cf. Section 1.3.2.4.1).

This is optimistic in one way, namely that there are no detours introduced due to routing congestion. However, it can also be both optimistic and pessimistic with respect to the layer choice, as the router might be able to use higher layers for routing certain nets, or it might sometimes be forced to use lower layers than the layer assignment prescribes because it cannot close all connections otherwise. This inaccuracy is certainly strongly mitigated by a good layer assignment, but as the layer assignment algorithm most likely does not have an as accurate picture of routing congestion as the global router itself, some amount of inaccuracy is inherent to this approach. As an additional drawback, the current method of estimating Steiner trees before global routing does not incorporate our new techniques for computing RC-aware Steiner trees from Chapter 5, therefore introducing additional pessimism.

Since there is both optimism and pessimism in the estimation method, it is a priori not absolutely clear whether our global routes will yield a better or worse timing than the estimates. However, the data from Sections 5.5 and 7.6 clearly shows that our RC-aware global router consistently achieves significantly better timing results than the estimates. In any case, as the timing characteristics of the global routes differ significantly from the ones of the routing tree estimates, and the global routes are far better estimates for the detailed routes than the routing tree estimates, it certainly makes sense to run some amount of timing optimization after global routing in GRBO.

7.1.1.2. *Motivating DRBO.* Motivating the DRBO step is not difficult: Detailed routing is a hard problem, and, as will be shown in Section 7.6, we often see some amount of timing degradation after detailed routing compared to our global routes. This is expected to some extent, as the process from Chapter 6 applies local optimizations to the global routes, making them a rather optimistic estimate for the detailed routes. In this context, improving the interplay between the global and the detailed router is

of major importance. However, it is a hard problem in practice, and some deviation is expected.
Another reason for using a detailed routing based optimization step is that when the design is fully detailed routed, more precise timing models that take coupling capacitances into account can be used. As explained later in Section 7.1.2, we do not use such timing models for our experiments in this thesis, but it is certainly an advantage to incorporate them in a production flow, as it is done in the DRBO version running in the IBM production flow.

**7.1.2. GRBO and DRBO Flows.** The timing optimization done in RBO is less extensive than the one done before routing, but otherwise the same general principles apply. In our GRBO and DRBO flows we use buffering, gate sizing, Vt optimization and local placement changes, which are basic timing optimization steps introduced in Section 1.2.2. These operations are used in different variations in order to improve power consumption and timing, and to resolve electrical violations. The code to run these operations is taken from the IBM physical design environment, and the overall optimization flows are similar, but not identical, to the ones that are running in the IBM production flow.
With respect to incremental routing, gate sizing, Vt optimization and local placement changes are similar operations in the sense that they only modify nets by changing pin positions. The difference here is the extent to which pin positions change: Gate sizing and Vt optimization are often combined and usually lead to small changes of pin positions, or no changes at all, in which case the change is said to be *pin-compatible*. The circuit library is actually designed to encourage pin-compatible changes, as they provide a way to improve timing without requiring actual routing changes, which is especially helpful during DRBO. Bigger pin movements occur when circuits are moved with the intention of improving timing metrics — after all, very small movements will most likely only result in very small improvements. However, bigger movements can also occur during gate sizing when a larger gate is chosen that does not fit any more into its old spot on the placement layer. In that case placement legalization (cf. Section 1.2.1) must solve the problem and might cause bigger pin movements.
Contrary to that, buffering does not modify existing nets, but deletes them and creates new ones instead. Here, several modes are possible: In the simplest mode, an existing net is subdivided by insertion of buffers. This step may require bigger routing changes than for example gate sizing, but as the newly added buffers are usually placed relatively close to the routing tree, the routing can often be completed by adding a relatively small amount of wiring, as is shown in Section 7.4. On the other hand, buffers may also be removed, and in fact whole buffer trees that have been built earlier in the optimization

flow may be rebuilt. In that case it is likely that bigger routing changes are required. Our GRBO and DRBO flows both include pure buffer insertion routines (i.e. no buffer removal), while we use buffer tree restructuring only in GRBO due to the magnitude of routing changes required for that operation.
Throughout this chapter, timing extractions are done using RICE [95]. Moreover, we use a simple model where the capacitance of a wire only depends on the layer and wire type of the wire, but not on its neighbors. Incorporating coupling capacitances caused by neighboring wires is possible and more accurate when dealing with detailed wires, but some questions are raised when incremental routing changes and the addition of global wires come into play. In the IBM DRBO flow where our incremental global router is used, coupling capacitances are incorporated into the timing model. However, for simplicity and comparability of timing results, we use the simpler method of neglecting coupling capacitances in this thesis.

**7.1.3. Capacity Estimation for DRBO.** One important topic to be addressed in the context of DRBO is the capacity estimation: While there are already routing blockages present in the traditional global routing scenario, the amount of blocked space is generally small compared to the amount of free space. Moreover, at least large blockages often have a simple structure that makes them easier to handle. The contrary is true for DRBO: Here, the design is fully detailed routed, and therefore most of the space is blocked by detailed wiring that can be highly branched on a local scale. Therefore, the capacity estimation of BonnRouteGlobal that existed prior to the DRBO use case proved to be impractical for DRBO, and refinement was needed. In the remainder of this section we give a coarse description of the capacity estimation and usage update methods that we use for DRBO. This is joint work with Pietro Saccardi, who did the majority of the work by providing the implementation of the capacity estimation.
BonnRouteGlobal's new capacity estimation for DRBO works roughly like this: Consider the situation directly after detailed routing. Let $G$ be the global routing graph and $e = \{(i,j,k),(i',j',k)\} \in E(G)$ be a wiring edge, and assume $i < i'$ without loss of generality. Then we define the *edge area* of $e$ as $A(e) := \{a \in A(i,j,k) \cup A(i',j',k) : p(i,j)_x < a_x \leq p(i',j')_x\}$. Given this, we let

- $F(e) \subseteq A(e)$ be the free space that we could occupy by placing minimum width wires in $A(e)$ without violating minimum distance rules,
- $B(e) \subseteq (A(e) \setminus F(e))$ be the space that is blocked by non-removable routing blockages,
- $D(e) = A(e) \setminus (F(e) \cup B(e))$ be the space that is blocked by detailed wires.

We then set the capacity of $e$ to the area of $A(e) \setminus B(e)$ and assume $D(e)$ to be occupied by input detailed wires. $B(e)$ and $D(e)$ are both initially blocked, but in contrast to

$B(e)$, $D(e)$ can be freed during DRBO by deleting detailed wires that are no longer needed. In other words, the area of $B(e)$ is (permanently) subtracted from the edge capacity, while the area of $D(e)$ is counted as input usage that may be reduced later. To match our model, these capacities and input usages are divided by the length of $e$ and scaled down to unit capacities by adapting the usage functions.
Of course, this description is only a coarse outline. In practice, a considerable amount of fine-tuning has to be applied to compute reasonable estimates for $B(e)$, $D(e)$ and $F(e)$ in presence of complex spacing rules.
Moreover, computing these areas requires a global outlook on the shapes in the whole edge area $A(e)$, which makes it difficult to update $D(e)$ when detailed wires are removed during DRBO. We therefore use simple estimates that only look at individual wire shapes when updating $D(e)$ during DRBO. This is not exactly precise, but as the routing changes made during DRBO are typically only local, it is a reasonable method in practice. An improvement of this method might be to do an exact recomputation of $D(e)$ a few times in the DRBO flow, but we did not incorporate this yet.
Our way of computing capacities for DRBO will have some effects on the experimental results that we are going to present in the remainder of this chapter. We will elaborate on this in the respective sections where this is of consequence.

## 7.2. The Incremental Routing Framework

When the incremental routing process starts, our router is registered with a general incremental routing interface provided by IBM. Other tools (usually timing optimization tools) can then control the router by means of that interface. In the following we call the tool that is using the incremental router the *operating tool*. This section contains three subsections: In Section 7.2.1 we present the notion of a *transaction*, which is the basic underlying concept of the incremental routing interface mentioned above. After that, we outline in Section 7.2.2 how Incremental BonnRouteGlobal implements changes that occur during a transaction. At last, Section 7.2.3 describes the mechanics behind undoing a transaction.

**7.2.1. Transactions.** As already stated, transactions are the fundamental concept behind the incremental routing framework that we are working in. In fact, the incremental routing process can be regarded as a series of individual transactions. For the sake of simplicity we first describe the transaction framework with a single-threaded context in mind. In a multi-threaded context the process works in a similar manner, but some additional complexity is introduced. We elaborate on that in Section 7.5.
The transaction framework works like this: Before making any changes, the operating tool must start a transaction. When a transaction is started, a callback mechanism is

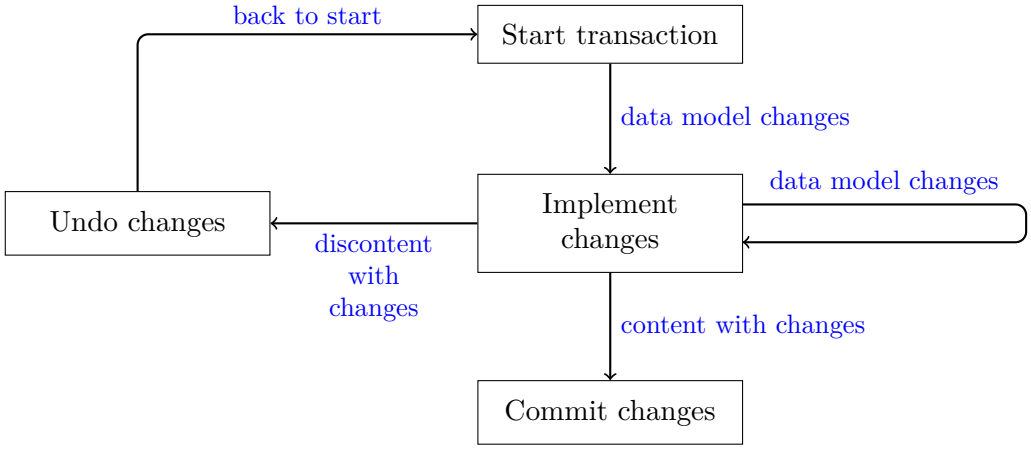



FIGURE 7.2. Transaction framework to operate the incremental router.

activated: Any relevant changes to the chip data model (e.g. circuit movements) are communicated to the router via callback functions that were previously installed. This way, the operating tool only needs to communicate its changes to the data model, but not directly to the router, as this is done automatically. It also makes the operating tool less dependent on the router, because for the most part the tool can act in the same way whether a router is installed or estimates are used.
When a transaction is active, the router only listens to the callbacks, but it does not implement them on-the-fly, as the callbacks can be very fine-grained. To have the changes implemented, the operating tool must issue a specific *implementation request* to the router. Afterwards, the operating tool can query various metrics like timing and routing congestion in order to decide whether to *commit* or *undo* the changes: When changes are committed, the state at the beginning of the transaction is forgotten, and the current state is established as a basis for further transactions. When *undo* is called, all changes made in this transaction (including routing changes) are undone. A more detailed description of how such an *undo* request is implemented in BonnRouteGlobal is given in Section 7.2.3. It is also possible for the operating tool to issue multiple implementation requests in a single transaction until the decision to commit or undo is made. A transaction is called *open* if it has been started but has neither been committed nor undone. Figure 7.2 illustrates the transaction framework.

**7.2.2. Implementing Changes.** During the incremental routing process, nets can be modified, deleted or newly created. An example for a modification is a placement change of a single circuit, which results in a pin position change for all the adjacent nets. Deletion and creation of nets occurs during buffering.

In our implementation we keep track of all nets that have been modified, deleted or newly created since the last implementation request or the start of the transaction. When implementation of these changes is requested, we remove all deleted nets from our data structures and complete the routing of modified and newly created nets. A crucial point about routing nets in this context is that we often can start with prewiring that already connects large parts of the net. This prewiring can be obtained in different ways depending on the history of the net.
In the first case, the net has been modified in the course of the transaction. In that case, we use the existing wiring of the net as a starting point and use the *minimal reroute* procedure from Section 7.3 to complete the routing.
In the second case, the net has been newly created by subdividing an already routed net into multiple nets through buffer insertion. In that case, we first run the *copy routes* procedure from Section 7.4 to distribute the wiring of the original net among the nets resulting from buffer insertion. After that, we again use the minimal reroute procedure from Section 7.3 to route those nets.
An exception to this rule occurs when there is a layer assignment or wire type change for a net: In that case, we reroute the affected net from scratch. Rerouting from scratch is also done in all cases that are not covered by the above rules. In our current optimization flows this only happens in GRBO when buffer trees are rebuilt, as our copy routes procedure from Section 7.4 only covers the case where a net is subdivided into multiple nets through buffer insertion, but not the reverse operation of merging multiple nets into one net through buffer removal. However, in general timing optimization flows there could also be other cases that are not covered by the above rules for preserving prewiring, which could for example occur when local logic restructuring is run (cf. Section 1.2.2).

**7.2.3. Undoing Transactions.** As stated in Section 7.2.1, the incremental routing framework offers the operating tool the possibility to undo transactions. When such a request to undo a transaction is issued, the router must revert all changes in its data model that were caused by this transaction. To be able to do this, we store the following extra information at any point in time during a currently open transaction:

- We store the set $\mathcal{N}_{\text{new}} \subseteq \mathcal{N}$ of *new net versions* (where $\mathcal{N}$ denotes the netlist, as usual): We have $N \in \mathcal{N}_{\text{new}}$ if and only if $N$ has been modified or newly created during the course of the current transaction. When a transaction is undone, we first remove all nets in $\mathcal{N}_{\text{new}}$ from the netlist, and all data associated with these nets from our other data structures.
- Complementary to $\mathcal{N}_{\text{new}}$, we store the set $\mathcal{N}_{\text{orig}}$ of *original net versions*: When a net $N \in (\mathcal{N} \setminus \mathcal{N}_{\text{new}})$ is modified or deleted, we add a snapshot of it to $\mathcal{N}_{\text{orig}}$. In addition to that, we also make a snapshot of the data associated with the net

that is stored in other data structures, including its route. In that sense, $\mathcal{N}_{\text{orig}}$ is not a subset of $\mathcal{N}$, but rather a collection of data associated with nets. When a transaction is undone, we add back all of this data to our data structures after having removed the data associated with $\mathcal{N}_{\text{new}}$.

- For every resource $r \in \mathcal{R}$ from our resource sharing framework (cf. Chapter 4), in particular edge resources, we store the usage difference $\Delta\text{usg}_r$ that was induced in this transaction (unless $\Delta\text{usg}_r = 0$). When a transaction is undone, we set $\text{usg}_r := \text{usg}_r - \Delta\text{usg}_r$ for all $r \in \mathcal{R}$, where $\text{usg}_r$ is the current usage of $r$.

In a single-threaded context, this method ensures that after undoing a transaction, our internal data model is restored back to its state at the point in time when the transaction was started. In a multi-threaded context (cf. Section 7.5), this cannot be guaranteed, as other threads may have made changes inbetween. However, what we can always guarantee is that we revert the changes that were made in the particular transaction being undone.

## 7.3. Minimal Reroutes

As outlined in Section 7.2.2, we are often facing the situation where we have to route a net $N$ and we are already given a wiring that almost connects $N$. This wiring may be the wiring of a previous version of $N$, or it may be the case that $N$ resulted from buffering a larger net, and the wiring has been assigned to $N$ during the copy routes process described in Section 7.4. This leads us to the notion of *minimal reroutes* and the corresponding Minimal Reroute Problem, which is described in this section.

**7.3.1. The Minimal Reroute Problem.** In the way our RBO framework is set up, most pin movements are relatively small, and buffers are usually also inserted close to the existing wiring. Therefore, we use a method that aims to reconnect disconnected pins to the existing prewiring with a minimum amount of added wire length. As a consequence, we treat the minimization of a weighted sum of wire length and via count as the objective and avoiding routing overflow as a constraint. This leads to the definition of the Minimal Reroute Problem and the Coarse Minimal Reroute Problem, which are introduced in Sections 7.3.1.1 and 7.3.1.2, respectively.

7.3.1.1. *Defining the Minimal Reroute Problem.* Before we can define the Minimal Reroute Problem, we need the following definitions:

Definition 7.1. Let $S$ be the layered chip area with layers $Z$, $Y$ be a Steiner tree connecting the exact shapes of a net $N$, and $\text{price}^{\text{wire}}_{\text{obj}} \colon Z \to \mathbb{R}_{\geq 0}$ and $\text{price}^{\text{via}}_{\text{obj}} \colon Z \to \mathbb{R}_{\geq 0}$

objective function prices for wires and vias. Then for $(v, w) \in E(Y)$ we define

$$\text{price}_{\text{obj}}(v, w) := \begin{cases} \text{price}^{\text{wire}}_{\text{obj}}(p(v)_z) \cdot \text{dist}(v, w) & \text{if } p(v)_z = p(w)_z, \\ \sum_{z'=\min\{p(v)_z, p(w)_z\}}^{\max\{p(v)_z, p(w)_z\}-1} \text{price}^{\text{via}}_{\text{obj}}(z') & \text{otherwise,} \end{cases} \tag{7.1}$$

to be the objective function price of $(v, w)$.

Definition 7.1 defines our objective function prices in a fairly standard way. We continue with a definition that is required to define our congestion prices:

Definition 7.2. Let $G$ be the global routing graph and $Y$ be a Steiner tree connecting the exact shapes of a net $N$. For $F \subseteq E(Y)$ we define

$$E_G(F) := \Big\{ \{(i, j, k), (i', j', k')\} \in E(G) : \exists\, (v, w) \in F \text{ with } p(v) \in A(i, j, k) \text{ and } p(w) \in A(i', j', k') \Big\}$$

to be the *edges of $G$ covered by $F$*.

Definition 7.2 will be used to determine congestion prices, which are always based on covered edges of the global routing graph due to our current congestion model. We first state our definition of the Minimal Reroute Problem and then explain the details and reasoning behind this problem formulation:

Problem 7.3: Minimal Reroute Problem

**Input:** A net $N$, a Steiner tree $Y_0$ connecting the exact shapes of a subset $N_0 \subseteq N$ (possibly $N_0 = \emptyset$), the global routing graph $G$ with layers $Z$, congestion prices $\text{price}_{\text{cong}} \colon E(G) \to \mathbb{R}_{\geq 0}$, wire and via objective function prices $\text{price}^{\text{wire}}_{\text{obj}} \colon Z \to \mathbb{R}_{\geq 0}$ and $\text{price}^{\text{via}}_{\text{obj}} \colon Z \to \mathbb{R}_{\geq 0}$.

**Task:** Compute a subgraph $Y_0'$ of a suitable subdivision of $Y_0$ and a Steiner tree $Y$ with $E(Y_0') \subseteq E(Y)$ connecting the exact shapes of $N$ such that $\text{price}_{\text{obj}}(E(Y) \setminus E(Y_0')) + \text{price}_{\text{cong}}(E_G(E(Y)) \setminus E_G(E(Y_0')))$ is minimized.

In contrast to most other parts of this thesis, we explicitly allow $Y_0$ to contain Steiner vertices of degree 1 — after all, $Y_0$ often almost connects the entire net, which includes wires that reach close to the disconnected pins. We require $Y_0$ to be a Steiner tree, but in principle, the problem could also be defined to work with a Steiner forest. However, defining $Y_0$ to be a Steiner tree will simplify the definition of our shape-based edge lengths from Section 7.3.3.2 a bit. In our application in practice, $Y_0$ is always a Steiner tree. Throughout this section, we often use the term *prewires* in order to refer to $Y_0$.

In the above model, $\text{price}_{\text{obj}}$ is measured based on the exact layout of $Y$, while $\text{price}_{\text{cong}}$ is measured based on covered edges of the global routing graph. The reason for the former is that it is more precise, and the reason for the latter is that it is in sync with the congestion model that we used during the main global routing phase before the incremental routing process. If the congestion model was based on exact shapes from the very beginning, then $\text{price}_{\text{cong}}$ could also be modeled based on exact shapes, which would yield a more consistent model. We will elaborate more on the definition of $\text{price}_{\text{obj}}$ and $\text{price}_{\text{cong}}$ in Section 7.3.3.

7.3.1.2. *Solving the Minimal Reroute Problem.* For solving the Minimal Reroute Problem we still mainly work on the global routing graph and only use a heuristic approach for minimizing wire length based on exact shapes. This leads to the definition of the Coarse Minimal Reroute Problem, which is basically a version of the Minimal Reroute Problem that is defined purely in terms of the global routing graph:

Problem 7.4: Coarse Minimal Reroute Problem

**Input:** A net $N$, a Steiner tree $X_0$ connecting the projected shapes of a subset $N_0 \subseteq N$ (possibly $N_0 = \emptyset$), the global routing graph $G$, edge prices $\text{price}\colon E(G) \to \mathbb{R}_{\geq 0}$.

**Task:** Compute a subgraph $X_0'$ of $X_0$ and a Steiner tree $Y$ with $E(X_0') \subseteq E(Y)$ connecting the projected shapes of $N$ such that $\text{price}(E(Y) \setminus E(X_0))$ is minimized.

Here, $X_0$ represents the edges of the global routing graph that are covered by the prewires of the underlying instance of the Minimal Reroute Problem in the sense of Definition 7.2. As such, $X_0$ might also contain Steiner vertices of degree 1. As we sometimes refer to an instance of the Coarse Minimal Reroute Problem and its underlying instance of the Minimal Reroute Problem at the same time, we always denote the prewires in the Minimal Reroute Problem as $Y_0$ and the ones in the Coarse Minimal Reroute Problem as $X_0$ in order to make them distinguishable.
Our process now works as follows: Given an instance of the Minimal Reroute Problem, we first turn it into the corresponding instance of the Coarse Minimal Reroute Problem. We then solve the Coarse Minimal Reroute Problem and finally use the methods from Chapter 6 — more precisely Section 6.4 — to convert the computed tree into one connecting exact shapes, providing a solution for the original instance of the Minimal Reroute Problem. Moreover, Steiner vertices of degree 1, which may be left over in the prewiring, are removed at the very end of the minimal reroute process. An image sequence from practice illustrating this process is given by Figure 7.3.

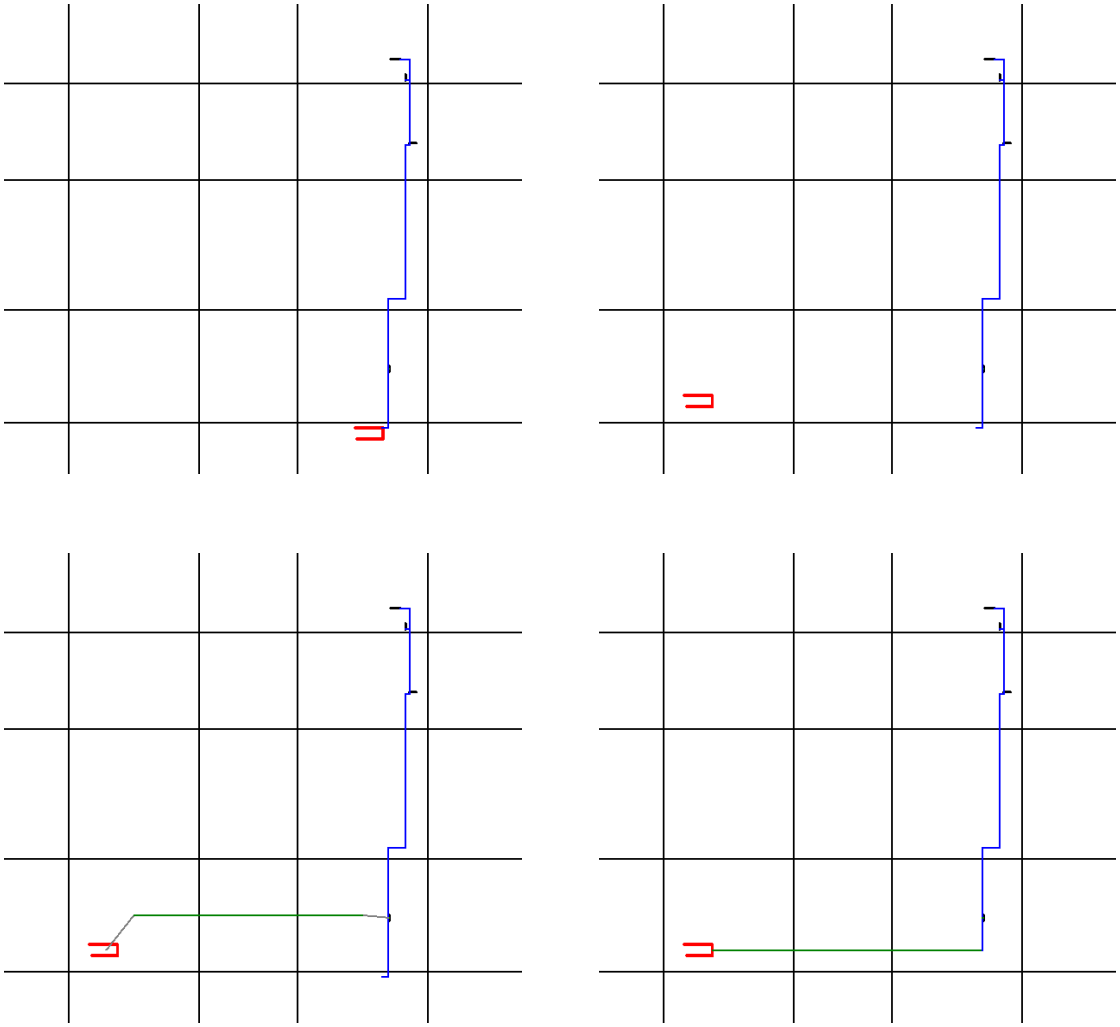


FIGURE 7.3. A minimal reroute sequence from practice: We start with a fully detailed routed net (blue), where the source pin is shown in red, while the four sink pins are colored black (upper left). The black horizontal and vertical lines illustrate the global routing tile grid. During DRBO, the source pin is moved to a different location (upper right). We then solve the corresponding instance of the COARSE MINIMAL REROUTE PROBLEM, which gives us a global routing (green) that reconnects the source pin to the prewiring (lower left). However, this global routing is embedded into the global routing graph, and therefore lacks connections to exact shapes, as indicated by the diagonal gray lines. Therefore, we use our methods from Chapter 6 in order to connect to exact shapes, and remove Steiner vertices of degree 1 at the very end (lower right).

It is easy to see that the Coarse Minimal Reroute Problem can be reduced to the Minimum Steiner Tree Problem in Graphs (cf. Section 2.3.1), as the edges corresponding to $X_0$ can just be contracted. We can therefore approximate it as outlined in Section 2.3.1. An open question is how to choose the edge prices for the Coarse Minimal Reroute Problem. We address this in Section 7.3.3. Before we do that, we discuss the reasoning behind our problem formulations in Section 7.3.2.

**7.3.2. Motivating the Minimal Reroute Problem.** In this section we motivate our formulation of the Minimal Reroute Problem from Section 7.3.1. We first present related work in Section 7.3.2.1 and then discuss our problem formulation in Section 7.3.2.2.

7.3.2.1. *Related Work.* In the context of the Minimal Reroute Problem, related work exists in the form of work dealing with variants of the Steiner Tree Reoptimization Problem. In this problem, we are given an instance $I$ of the Minimum Steiner Tree Problem in Graphs and an optimum solution of a slightly different instance $I_0$. Here, $I$ results from $I_0$ by a local modification, the most important ones being: increasing the cost of an edge, decreasing the cost of an edge, adding a vertex to the graph, deleting a vertex from the graph, adding a terminal to the terminal set, removing a terminal from the terminal set.
As it is the most applicable variant in our scenario, we will restrict ourselves to the case where a terminal is added to or removed from the terminal set. This yields the following problem formulation:

Problem 7.5: Steiner Tree Reoptimization Problem with Changed Terminal Set

**Input:** A complete graph $G$, metric edge costs $c\colon E(G) \to \mathbb{R}_{\geq 0}$, a terminal set $T_0$ embedded into $G$, a Steiner tree $Y_0$ for $T_0$ minimizing $c(E(Y_0))$, and a terminal set $T$ such that

(a) $T = T_0 \cup \{t\}$ for a new terminal $t$ embedded into $G$, or
(b) $T = T_0 \setminus \{t\}$ for $t \in T_0$.

**Task:** Compute a Steiner tree $Y$ for $T$ minimizing $c(E(Y))$.

In [15], Böckenhauer et al. show that both variants (a) and (b) of this problem are $NP$-hard: Any instance of the Minimum Steiner Tree Problem in Graphs can be solved by iteratively solving variant (a) starting with an empty terminal set, or by iteratively solving variant (b) starting with a terminal set that comprises all vertices of the graph, in which case the optimum solution is a minimum spanning tree. Moreover,

they give 1.5-approximation algorithms for both variants: For variant (a), their algorithm simply connects the new terminal $t$ to $Y_0$ by an edge of minimum cost. Therefore, their algorithm runs in $\mathcal{O}(|T|)$ time (given an adequate representation of the edge costs). On the other hand, their algorithm for variant (b) is not as simple and requires a running time of $\mathcal{O}(|V(G)|^{4.17})$. In addition to that, they devise a PTAS for the restricted case where edge costs are natural numbers in $\{1, \dots, k\}$ for a constant $k$. A related problem is studied by Escoffier et al. [36]: They study a variant of version (a) where the newly added terminal is also a new vertex in $V(G)$. For this case, they give an 1.5-approximation algorithm with running time $\mathcal{O}(|T|^2 \log |T|)$. Moreover, they present a $(2 - (1 / (k+2)))$-approximation algorithm for the case where $k$ such new terminal vertices are added.
These bounds are improved by Bilò et al. in [12]: They give an 1.344-approximation algorithm for the version where one terminal is added, and an 1.408-approximation algorithm for the variant where one terminal is removed. Further improvements are made by Bilò and Zych [13, 127]: They provide $((3\sigma - 2) / (2\sigma - 1)) + \varepsilon$ approximation algorithms for any $\varepsilon > 0$ for variants (a) and (b), where $\sigma$ is the approximation guarantee of an algorithm for the Minimum Steiner Tree Problem in Graphs that they use as a subroutine. Assuming the best-known algorithm of Byrka et al. [21] to be used, this results in an approximation guarantee of 1.218. With the same notation, this is improved by Goyal and Mömke [43] for both variants to $(10\sigma - 7) / (7\sigma - 4)$, which results in an approximation ratio of 1.204 using [21] again. Finally, Bilò gives polynomial time approximation schemes for both variants in [11].

7.3.2.2. *Discussing the Problem Formulation.* In our formulation of the Minimal Reroute Problem we seek to minimize the wire length that is added to the tree. When translated into the Coarse Minimal Reroute Problem, this corresponds to computing a minimum Steiner tree for $N$ while assuming zero prices for the edges of $X_0$. Generally, this can result in suboptimal solutions, as shown in Figure 7.4.
In order to avoid such situations, one could let the router use the prewires for a reduced, but not zero, price. This would be easy to model: Instead of making edges covered by prewires free to use as in the Coarse Minimal Reroute Problem, one could multiply the price of every edge $e \in E(G)$ that is covered by prewires by a factor $\alpha_e \in [0, 1]$. By a reasonable choice of these multipliers, the situation from Figure 7.4 could most likely be avoided (assuming moderate congestion prices). Another possibility might be to make use of the algorithms for the Steiner Tree Reoptimization Problem with Changed Terminal Set from Section 7.3.2.1, as most reroutes are required because only a few pins are moved — in many cases, it is actually only one pin in a net that is moved. This could be modeled as a sequence of both variants of the Steiner

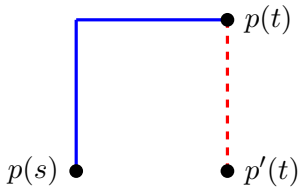



FIGURE 7.4. Sketch of an instance of the MINIMAL REROUTE PROBLEM: The sink $t$ is initially connected to $s$ by the blue path and then moved to a new position $p'(t)$ during RBO. In the corresponding instance of the MINIMAL REROUTE PROBLEM, the blue path corresponds to $Y_0$, and reconnecting $t$ to $Y_0$ by the dashed red line might very well be an optimum solution in our problem formulation. However, reconnecting $t$ directly to $s$ is more desirable in this case.

TREE REOPTIMIZATION PROBLEM WITH CHANGED TERMINAL SET, or it might even be interesting to derive algorithms for a third variant where a terminal is moved.
However, we refrain from these possibilities, as we want to stick to the objective of completing the existing wiring with a minimum amount of routing changes, even if there are cases where a smaller wire length can be obtained by making bigger changes. Firstly, this is naturally desirable in DRBO, as the goal in DRBO is to fix timing problems by introducing as few global wires as possible, as implementing newly added global wires later by the detailed router can be difficult and introduce new timing problems.
In GRBO the picture is less clear, but also there we seek to make as few routing changes as possible in the absence of an RC-aware incremental router, i.e. an extension of the methods from Chapters 4 and 5 to the incremental routing scenario. The reason for this is that our global routes that exist at the start of the incremental routing process are well optimized with respect to their timing characteristics. This is demonstrated clearly by our experimental results from Section 5.5, where we can show that our RC-aware routes achieve a much better timing than the baseline run, which basically optimizes wire length. Therefore, as our incremental routing framework is not yet RC-aware, we seek to modify our initial RC-aware global routes as little as possible in order to retain their timing characteristics.
Apart from that, it is natural that the more sophisticated routing methods from Chapter 4 used during the main global routing phase tend to find better routes than the simplistic method of routing one net at a time during incremental routing. Moreover, timing optimization tools might also consider electrical characteristics of the existing route for their optimization, which also supports making only small routing changes.
Another important benefit is running time: In many cases, the COARSE MINIMAL REROUTE PROBLEM can be solved very quickly, as only a few new short paths have to be

computed. Allowing bigger changes would almost certainly result in a larger running time — for example, multiplying the prices of all global routing graph edges covered by prewires by a certain factor as outlined above could require recomputations of long paths or the whole tree even in cases where one pin has been moved by only a small amount.
As a conclusion, there are several indications that making only minimal routing changes may yield better results in less running time than making larger changes with an incremental router that is not RC-aware. This is clearly demonstrated experimentally by Table 7.1 from Section 7.3.4.1. Therefore, we stick to our formulation of the Minimal Reroute Problem for both GRBO and DRBO, as it is a natural and simple problem formulation for minimizing the amount of routing changes that are performed to complete each route. A possible improvement might be to use a timing-aware incremental routing framework. In that case, bigger routing changes could be beneficial (at least in GRBO), and a different formulation for the Minimal Reroute Problem might be favorable. We leave this as a possible subject for future research.

**7.3.3. Defining Prices.** In this section we define the various price functions that are used in the definitions of the Minimal Reroute Problem and Coarse Minimal Reroute Problem in Section 7.3.1. We start with the price functions for the Minimal Reroute Problem in Section 7.3.3.1 and then use them to define our edge prices for the Coarse Minimal Reroute Problem in Section 7.3.3.2.

7.3.3.1. *Prices for the Minimal Reroute Problem.* To define $\text{price}_{\text{cong}}$ and $\text{price}_{\text{obj}}$ for the Minimal Reroute Problem, we first define *load prices* $\text{lp}\colon \mathbb{R} \to \mathbb{R}$ by

$$\text{lp}(x) := e^{\alpha x+\beta\left(\min\left\{\max\{\gamma_{\min},x\},\gamma_{\max}\right\}-\gamma_{\min}\right)}.$$

In our implementation we choose $\alpha = 40$, $\beta = 160$, $\gamma_{\min} = 0.975$ and $\gamma_{\max} = 1.025$. Given a net $N$ and $e \in E(G)$ we set

$$\text{price}_{\text{cong}}(e) := \text{lp}(\text{usg}_e + \text{usg}(N, e)) - \text{lp}(\text{usg}_e),$$

where $\text{usg}_e$ is the current relative routing space usage of $e$, and $\text{usg}(N, e)$ is the fraction of routing space that $N$ consumes on $e$. The intention here is as follows: We express congestion prices in terms of our load prices, and the congestion price for using an edge $e$ when routing a net $N$ is exactly the load price difference that is caused when $N$ uses $e$. These load prices depend exponentially on the load with a base of $e^{\alpha}$, and when the load is in the critical interval $[\gamma_{\min}, \gamma_{\max}]$, prices grow rapidly with a base of $e^{\alpha+\beta}$.
For defining $\text{price}_{\text{obj}}$ we set

$$\text{price}^{\text{wire}}_{\text{obj}}(z) := \rho(z) \cdot \frac{\text{lp}(\delta_{\max}) - \text{lp}(\delta_{\min})}{l_{\text{avg}}},$$

for $z \in Z$, where $\delta_{\max} = \gamma_{\min}$ and $\delta_{\min} = \delta_{\max} - 0.05$ are constants, $l_{\text{avg}}$ is the average length of wiring edges in the global routing graph, and $\rho(z)$ is a layer-dependent constant: Letting $z_{\min}$ and $z_{\max}$ denote the minimum and maximum layer of the layer assignment of the net $N$ to be routed, and $w(z)$ for $z \in Z$ denote the metal width of the wire type of $N$ on $z$, we set

$$\rho(z) := \begin{cases} 1 & \text{if } z_{\min} \leq z \leq z_{\max}, \\ 1.05 + 0.05 \cdot \max\left\{0, \left(w(z_{\min}) \,/\, w(z)\right) - 1\right\} & \text{otherwise.} \end{cases}$$

The reasoning here is as follows: The objective function price of a wire of length $l_{\text{avg}}$ on a layer $z \in [z_{\min}, z_{\max}]$ is set to equal the load price difference of $\delta_{\min}$ and $\delta_{\max}$ (cf. Equation 7.1), i.e. the congestion price of adding a wire on an edge that increases usage from $\delta_{\min}$ to $\delta_{\max}$. The intention behind our setting of $\delta_{\min}$ and $\delta_{\max}$ is to let objective function prices dominate congestion prices for edges safely below 100% usage, but let congestion prices dominate when approaching 100% usage. Moreover, as, unlike the routing oracle from Chapter 5, our incremental router is currently not timing-aware, we penalize violating the layer assignment through $\rho$, where the magnitude of the penalization depends on the quotient of the wire widths on $z$ and $z_{\min}$. The via prices $\text{price}^{\text{via}}_{\text{obj}}(z)$, $z \in Z$, are expressed in multiples of $\text{price}^{\text{wire}}_{\text{obj}}(z)$, where the multipliers are technology-dependent parameters that are also used for determining via prices during the main global routing phase.

Our definitions of $\text{price}_{\text{cong}}$ and $\text{price}_{\text{obj}}$ are tailored to the way we do incremental routing: We route one net at a time without ripping up and rerouting other nets, and there is currently no step in our optimization flow that aims to resolve congestion. Therefore, it is natural to let congestion prices grow rapidly in an interval around 100% routing space usage, as creating detours is more desirable than creating routing overflow in our scenario. This is especially true for our current optimization flow, as not all operating tools make their decision to commit or undo a transaction based on congestion increases, but they do check timing metrics.

Adding the possibility to rip out and reroute bystander nets during incremental routing would probably be an improvement, but in particular in a multi-threaded context (cf. Section 7.5), some issues would need to be addressed. Another possible improvement might be to add steps to the optimization flow whose goal is to resolve congestion created during the incremental routing process. The challenge here would be to preserve most of the timing improvements that were achieved during the previous optimization steps.

7.3.3.2. *Edge Prices for the Coarse Minimal Reroute Problem.* In this section we define the edge prices $\text{price}\colon E(G) \to \mathbb{R}_{\geq 0}$ for the Coarse Minimal Reroute Problem. As we solve the Coarse Minimal Reroute Problem as an approximation of the

Minimal Reroute Problem, these edge prices are defined based on the underlying instance of the Minimal Reroute Problem. In that sense, assume for the remainder of this section that we are given an instance of the Minimal Reroute Problem which we use as a basis for defining edge prices for the corresponding instance of the Coarse Minimal Reroute Problem.

Let $e \in E(G)$. If $e$ is a via edge between layer $z$ and $z+1$, then we set $\text{price}(e) := \text{price}^{\text{via}}_{\text{obj}}(z) + \text{price}_{\text{cong}}(e)$. If $e$ is a wiring edge on layer $z$, then we set

$$\text{price}(e) := \text{price}^{\text{wire}}_{\text{obj}}(z) \cdot \text{sbl}(e) + \text{price}_{\text{cong}}(e),$$

where $\text{sbl}(e)$ is the *shape-based length* of $e$. This means that in contrast to the classical model, where we would set $\text{sbl}(v,w) := \text{dist}(v,w)$ for $(v,w) \in E(G)$, we take the exact shapes of the pins and prewires in the tiles that we are connecting into account. As already stated in Section 7.3.1, making $\text{price}_{\text{cong}}(e)$ dependent on $\text{sbl}(e)$ would also be possible in a congestion model that is based on exact pin and wire shapes.

The lengths $\text{sbl}\colon E(G) \to \mathbb{R}_{\geq 0}$ are determined in the following way: Consider the input of the underlying instance of the Minimal Reroute Problem, and let $N' := N \setminus N_0$. For every global routing tile $(i,j) \in \{1,\dots,n_x\} \times \{1,\dots,n_y\}$ (cf. Definition 2.8) we define

$$\begin{aligned}\text{pins}(i,j) &:= A(i,j) \cap \left\{(p(\pi)_x, p(\pi)_y) : \pi \in N'\right\},\\ \text{wires}(i,j) &:= A(i,j) \cap \bigcup_{e \in E(Y_0)} L^{\text{2D}}(e),\end{aligned}$$

to be the areas that are covered by the two-dimensional projections of exact shapes of disconnected pins and prewires (see Definition 2.13 for the definition of $L^{\text{2D}}$). We then set

$$\text{bb}(i,j) := \begin{cases}\text{BB}(\text{pins}(i,j) \cup \text{wires}(i,j)) & \text{if } \text{pins}(i,j) \cup \text{wires}(i,j) \neq \emptyset,\\ \{p(i,j)\} & \text{otherwise,}\end{cases}$$

where BB is the bounding box as in Definition 2.15, and $p(i,j)$ is the center of tile $(i,j)$. This allows us to define $\text{sbl}(v,w)$ for a wiring edge $\{v,w\} = \{(i_v,j_v,k),(i_w,j_w,k)\} \in E(G)$ as

$$\text{sbl}(v,w) := \begin{cases}\text{dist}(v,w) & \text{if } \text{wires}(i_v,j_v) \neq \emptyset \text{ and } \text{wires}(i_w,j_w) \neq \emptyset,\\ \text{dist}(\text{bb}(i_v,j_v), \text{bb}(i_w,j_w)) & \text{otherwise,}\end{cases}$$

where $\text{dist}(\text{bb}(i_v,j_v), \text{bb}(i_w,j_w))$ is the minimum rectilinear distance between any two points in $\text{bb}(i_v,j_v)$ and $\text{bb}(i_w,j_w)$.

An illustration is given by Figure 7.5. The reasoning behind this definition of sbl is as follows: If a tile $(i,j)$ does not contain any prewires or pins from $N'$, then we assume

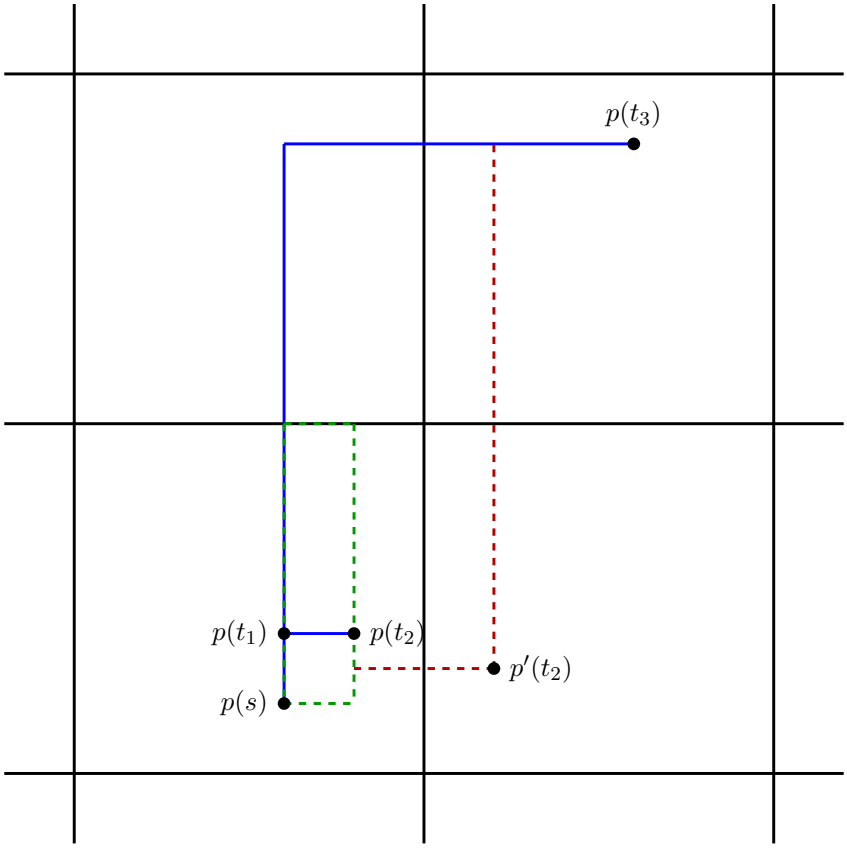



FIGURE 7.5. Illustration of our shape-based edge lengths from Section 7.3.3.2: We see four global routing tiles delimited by black lines, and we identify them by their tile coordinates $(0,0)$, $(0,1)$, $(1,1)$ and $(1,0)$ (clockwise starting from the lower left). Moreover, we see a net with four pins $s$, $t_1$, $t_2$ and $t_3$, and a route connecting them (in blue). During RBO, $t_2$ is moved from $p(t_2)$ to $p'(t_2)$, making a reroute necessary. For computing shape-based lengths for edges between tile $(1,0)$ and tiles $(0,0)$ and $(1,1)$, we compute the bounding boxes of the pin and wire shapes in each tile: For tile $(0,0)$, this bounding box is illustrated by the dashed green lines; for tile $(1,0)$, the bounding box just comprises $p'(t_2)$; for tile $(1,1)$, it coincides with the area covered by the blue wire located in that tile.
The dashed red lines then illustrate the shortest connections between these bounding boxes. Their lengths define the shape-based edge lengths from Section 7.3.3.2. This way, the objective function prices for connecting $p'(t_2)$ to tile $(0,0)$ are much smaller than for connecting it to tile $(1,1)$. In the traditional model, where edge lengths are always measured from tile center to tile center, connecting $p'(t_2)$ to tile $(1,1)$ would be associated with the same objective function prices as connecting it to tile $(0,0)$, which could result in a locally bad routing.

that we are routing from and to the center of the tile, and so $\text{bb}(i,j) = \{p(i,j)\}$. On the other hand, if $(i,j)$ contains prewires or pins from $N'$, then we assume that a wire entering the tile is used to access these pins or prewires, and so $\text{bb}(i,j)$ is the bounding box of their shapes.

An exception occurs when both adjacent tiles already contain prewires, which must already belong to the same connected component of $Y_0$. This is irrelevant in a two-dimensional model, but in our three-dimensional model, the router might in some rare cases actually use an edge between two tiles that are already covered by prewires on a different layer if via prices in one of the two tiles are very high. In that case, we use the standard model that assumes the distance of the respective tile centers as the length of the wire.

As just mentioned, our definition of sbl is based on the two-dimensional projections of pin and prewire shapes, but the actual net routing is done on the three-dimensional global routing graph. However, in the majority of cases, using a two-dimensional model for the definition of sbl seems to be more reasonable, as it is common to first enter a tile with a wiring edge and then use vias to connect to pins or prewires.

The following propositions show that the computation time for sbl including preprocessing is fast. For these propositions we make some realistic assumptions about data structures in our global router implementation:

PROPOSITION 7.6. *Computing* $\text{wires}(i,j)$, $\text{pins}(i,j)$ *and* $\text{bb}(i,j)$ *for all tiles* $(i,j)$ *can be done in* $\mathcal{O}(n \log k + m)$ *time, where* $n := |N'|$, $k := n_x \cdot n_y$ *is the number of global routing tiles, and* $m := \sum_{e \in E(Y_0)} \left|\{(i,j) \in \{1,\dots,n_x\} \times \{1,\dots,n_y\} : A(i,j) \cap L^{\text{2D}}(e) \neq \emptyset\}\right|$.

PROOF. We assume that for any point $(x,y)$ in the two-dimensional chip area we can find the tile $(i,j)$ with $(x,y) \in A(i,j)$ in $\mathcal{O}(\log k)$ time, as this can be achieved by using binary search on a data structure that contains all tile borders sorted by their coordinate. Therefore, we can compute $\text{pins}(i,j)$ for all tiles $(i,j)$ in $\mathcal{O}(n \log k)$ time. For computing wires, we start at an arbitrary vertex $v \in V(Y_0)$ and query its tile. We then traverse $Y_0$ starting from $v$, for instance by using depth first search, and assign every wire part to the tile it is located in. This can be done in $\mathcal{O}(\log k + m)$ time, and so the total running time follows. □

Querying $\text{sbl}(e)$ for $e \in E(G)$ can then be done in constant time:

PROPOSITION 7.7. *Given* $\text{wires}(i,j)$ *and* $\text{bb}(i,j)$ *for all tiles* $(i,j)$, *we can compute* $\text{sbl}(e)$ *for any* $e \in E(G)$ *in* $\mathcal{O}(1)$ *time.*

PROOF. We store a $n_x \times n_y$ matrix containing wires, bb and an integer time stamp for every tile $(i,j)$. Each time we route a net, we increase a counter and use this counter

as time stamp to distinguish current and outdated entries $(i, j)$ (which are interpreted as $\text{wires}(i, j) = \text{bb}(i, j) = \emptyset$) in our matrix. Initializing the matrix requires $\mathcal{O}(n_x \cdot n_y)$ time, but has to be done only once when the incremental routing process begins. Afterwards, sbl can be queried in constant time. □

In practice, there are usually only relatively few tiles $(i, j)$ with $\text{bb}(i, j) \neq \emptyset$. Therefore, one can also use a fast search data structure like a hash table in order to save memory and keep the implementation simple.
Clearly, this approach is far from being accurate. An accurate approach for handling exact pin and prewire positions is given by Hähnle and Saccardi [45, 100]. However, as already stated in Chapter 6, this approach is rather complex and beyond the scope of this thesis. We therefore stick to this simpler method of using shape-based edge lengths, and our experimental results from Section 7.3.4 show that this method already helps to reduce the amount of wire length that is added when solving the Minimal Reroute Problem.

**7.3.4. Experimental Results.** In this section we analyze the practical performance of our minimal reroute framework. To do this, we present two different sets of experimental results: In Section 7.3.4.1 we compare GRBO results with minimal reroutes to GRBO results in a setting where every modified net is rerouted from scratch. Afterwards, we examine the effects of our shape-based edge lengths in practice in Section 7.3.4.2.
All experiments presented in this section are carried out on an Intel Xeon E5-2667 v2 server running at 3.30 GHz using 16 threads. As usual, our testbed and metrics appearing in subsequent tables are explained in Appendix A.

7.3.4.1. *Minimal Reroutes versus Rerouting from Scratch.* We start with comparing our default GRBO flow that uses minimal reroutes against a hypothetical GRBO flow where every net that is changed during GRBO is rerouted from scratch, ignoring all prewires. This rerouting is done using the same edge prices and minimum Steiner tree algorithm (cf. Section 2.3.1) as in the case with minimal reroutes, just without using the prewires. We choose GRBO for this experiment as rerouting all nets from scratch is not really acceptable in DRBO, and a comparison of metrics like timing and wire length between the two runs would be difficult to interpret due to the fact that the run using minimal reroutes would preserve much more detailed wiring while rerouting from scratch would add much more global wiring. The purpose of this experiment is to validate claims from Section 7.3.2 stating that retaining large portions of our RC-aware routes is superior to reconstructing large parts of the routing tree by a minimum Steiner tree algorithm, therefore also confirming the validity of our formulation of the Minimal Reroute Problem.

| Unit (# nets) | Run | WS [ps] | FOM [ps] | EV | WL [m] | wACE4 [%] | OFtgt [100 pitch$^2$] | RT [h:mm:ss] |
|---|---|---|---|---|---|---|---|---|
| U1 | Start | -129 | -29328 | 598 | 0.953 | 88.8 | 4.6 | — |
| (77 528) | MR | -72 | -21851 | 406 | 0.954 | 88.9 | 5.3 | 0:06:10 |
| | No MR | -77 | -24539 | 438 | 0.957 | 89.7 | 10.9 | 0:06:12 |
| U2 | Start | -97 | -57448 | 313 | 1.102 | 87.6 | 0.0 | — |
| (79 119) | MR | -85 | -48097 | 230 | 1.102 | 87.9 | 0.0 | 0:05:32 |
| | No MR | -94 | -50430 | 211 | 1.100 | 88.0 | 0.0 | 0:06:00 |
| U3 | Start | -149 | -118322 | 299 | 1.246 | 86.4 | 0.0 | — |
| (100 827) | MR | -74 | -84562 | 259 | 1.249 | 86.8 | 0.0 | 0:07:13 |
| | No MR | -81 | -88906 | 233 | 1.248 | 87.4 | 0.0 | 0:07:13 |
| U4 | Start | -196 | -152236 | 200 | 1.230 | 89.4 | 8.2 | — |
| (111 140) | MR | -116 | -84031 | 150 | 1.242 | 89.3 | 9.1 | 0:10:53 |
| | No MR | -148 | -102543 | 129 | 1.247 | 90.3 | 34.9 | 0:11:18 |
| U5 | Start | -62 | -27501 | 3 | 1.383 | 82.1 | 0.0 | — |
| (119 228) | MR | -59 | -10847 | 0 | 1.382 | 82.7 | 0.0 | 0:08:47 |
| | No MR | -59 | -20220 | 14 | 1.381 | 85.8 | 0.0 | 0:09:27 |
| U6 | Start | -120 | -226777 | 1385 | 5.007 | 88.5 | 19.1 | — |
| (254 208) | MR | -102 | -193967 | 1193 | 5.012 | 88.9 | 30.7 | 0:22:29 |
| | No MR | -108 | -217576 | 972 | 5.022 | 89.3 | 37.7 | 0:27:03 |
| U7 | Start | -59 | -64554 | 771 | 4.757 | 83.6 | 0.0 | — |
| (276 799) | MR | -55 | -22667 | 606 | 4.757 | 83.8 | 0.0 | 0:17:12 |
| | No MR | -55 | -30704 | 464 | 4.755 | 85.4 | 0.1 | 0:19:15 |
| U8 | Start | -86 | -834010 | 9321 | 37.560 | 86.1 | 43.1 | — |
| (1 681 671) | MR | -75 | -356411 | 2946 | 37.523 | 86.4 | 29.9 | 2:12:02 |
| | No MR | -75 | -490186 | 1614 | 37.497 | 87.3 | 28.0 | 2:45:53 |

TABLE 7.1. Experimental results comparing minimal reroutes against rerouting from scratch.

Our results are illustrated in Table 7.1. The ”Start” row depicts the results after running RC-aware BonnRouteGlobal from Chapters 4 and 5, directly before starting GRBO. The row labeled ”MR” represents the run where minimal reroutes are used (which is the default in our RBO flows), while the ”No MR” row represents the run where we always

reroute from scratch. Confirming our conjectures from Section 7.3.2, the first thing to notice is that timing results are always superior when minimal reroutes are used, and often by large amounts. This is not surprising, as our RC-aware routes computed during the main global routing phase should exhibit a better timing behavior than minimum Steiner trees computed during incremental routing. Moreover, keeping routing changes minimal might also be favorable for timing optimization tools.

The second observation is that the number of electrical violations often decreases when rerouting from scratch. This is in accord with our findings from Section 5.5, where we observed that our RC-aware router sometimes introduces electrical violations by making detours on timing-uncritical nets. If fixing such electrical violations is attempted during GRBO, then rerouting from scratch using a timing-agnostic minimum Steiner tree algorithm can actually solve the problem, as such an algorithm does not introduce generous detours due to the timing-uncriticality of the net. However, this effect is likely to diminish once electrical violations are better handled during RC-aware routing.

Looking at congestion, we can see that it does not increase to a noticeable extent when minimal reroutes are used, but increases slightly on some units when rerouting from scratch. In principle, this does not have to be the case, as when rerouting from scratch the previous route is still a possible option, unless some of the used edges of the previous route have been blocked by a different thread. However, in practice, deviations are likely to occur. Moreover, as already pointed out above, in some cases such as electrical violation fix up and power recovery, detouring routes for timing-uncritical nets might be replaced by short routes, as the minimum Steiner tree algorithm used by Incremental BonnRouteGlobal is not timing-aware.

Wire length decreases slightly on the less congested units when rerouting from scratch, which is expected due to the objective of minimizing wire length during the net routing process. On the more congested units, e.g. on U6, wire length can also increase. However, the variations in wire length are very small throughout the whole testbed. In general, it is also intuitive that the simple method of routing one net at a time does not achieve the same quality of results as the more elaborate global routing algorithm from Chapter 4.

At last, we see that using minimal reroutes results in decreased running times, which is expected. Here, it is important to note that with 16 threads — the number of threads used for our experiment — only a small fraction of the GRBO running time is actually spent in BonnRouteGlobal — as we see later in Table 7.6 from Section 7.5.3.1, it is less than 15% on our largest unit U8. In fact, the running time spent in BonnRouteGlobal increases by 73% on U8 when rerouting from scratch, which can be considered a substantial increase.

As a summary, we conclude that using minimal reroutes results in superior results in less running time. The only exception here is the number of electrical violations, but as explained above, the minimal reroute framework is unlikely to be the root cause of this effect. These results confirm claims from Section 7.3.2 about the motivation behind our formulation of the Minimal Reroute Problem, and prove the practical viability of our approach. Of course, further improvements can certainly be made, as pointed out in Section 7.3.2.

7.3.4.2. *Shape-Based Edge Lengths.* In this section we evaluate our shape-based edge lengths from Section 7.3.3.2 experimentally. To this end, we provide two tables — Table 7.2 for GRBO and Table 7.3 for DRBO. The rows of our tables can be explained as follows: In the row labeled "Start" we list the metrics directly before we start our incremental routing process. In GRBO, this is directly after RC-Aware BonnRouteGlobal from Chapters 4 and 5 has been run. For the starting point for our DRBO tables, we run RC-Aware BonnRouteGlobal without GRBO followed by BonnRouteDetailed. The second and third row then display the results after the incremental routing process. Here, the run labeled "IBRG + sbl" uses the shape-based edge lengths presented in Section 7.3.3.2 during the reroute process. In contrast to that, the run labeled "IBRG - sbl" represents the traditional setting without shape-based edge lengths, i.e. $\mathrm{sbl}(v,w) := \mathrm{dist}(v,w)$ for any wiring edge $(v,w) \in E(G)$ in the notation of Section 7.3.3.2.
In addition to general metrics, our tables contain one column labeled ΔWL and one column labeled ΔAWL. The former contains the number $(\mathrm{wl} - \mathrm{wl}_0) \,/\, \mathrm{wl}_0$, where wl is the total wire length after the step in the respective row, and $\mathrm{wl}_0$ is the wire length at the start of the incremental routing process, i.e. the wire length corresponding to the "Start" row. In the row labeled ΔAWL we display the relative difference of the total added wire length in all reroutes and an approximate lower bound for it. More precisely, we have

$$\Delta\mathrm{AWL} = \frac{\sum_{I\in\mathcal{I}} \mathrm{awl}(I, Y(I)) - \mathrm{awl}(I, Y^*(I))}{\sum_{I\in\mathcal{I}} \mathrm{awl}(I, Y^*(I))},$$

where

- $\mathcal{I}$ is the set of instances of the Minimal Reroute Problem that were solved during the whole incremental routing process,
- $\mathrm{awl}(I,Y) := \sum_{(v,w)\in E(Y)\setminus E(Y_0(I))} \mathrm{dist}(v,w)$, where $Y$ is a solution for $I$ and $Y_0(I)$ represents the prewires for $I$ in the Minimal Reroute Problem,
- $Y(I)$ for $I \in \mathcal{I}$ denotes the solution that we found for $I$,
- $Y^*(I)$ for $I \in \mathcal{I}$ is a solution for $I$ that approximately minimizes awl.

Here, $Y^*$ is found by the approximation algorithm for the Rectilinear Minimum Steiner Tree with Prewires Problem that is described in Section 2.3.3.

| Unit (# nets) | After... | WS [ps] | FOM [ps] | PWR [mW] | ΔWL [%] | ΔAWL [%] | wACE4 [%] | OFtgt [100 pitch$^2$] | RT [h:mm:ss] |
|---|---|---|---|---|---|---|---|---|---|
| U1 | Start | -129 | -29665 | 54.79 | — | — | 88.8 | 3.9 | — |
| (77 528) | IBRG - sbl | -72 | -22250 | 55.28 | 0.18 | 7.52 | 89.0 | 6.5 | 0:05:49 |
| | IBRG + sbl | -72 | -22023 | 55.39 | 0.16 | 6.98 | 89.0 | 5.2 | 0:06:10 |
| U2 | Start | -98 | -57419 | 22.41 | — | — | 87.6 | 0.0 | — |
| (79 119) | IBRG - sbl | -90 | -48920 | 22.47 | -0.06 | 2.97 | 88.0 | 2.0 | 0:05:27 |
| | IBRG + sbl | -88 | -48449 | 22.51 | -0.06 | 2.40 | 87.8 | 0.0 | 0:05:29 |
| U3 | Start | -149 | -118255 | 70.71 | — | — | 86.3 | 0.0 | — |
| (100 827) | IBRG - sbl | -76 | -85909 | 71.04 | 0.24 | 1.71 | 86.8 | 0.1 | 0:07:02 |
| | IBRG + sbl | -75 | -83985 | 71.01 | 0.21 | 1.19 | 86.8 | 0.0 | 0:07:13 |
| U4 | Start | -196 | -152137 | 37.05 | — | — | 89.4 | 6.3 | — |
| (111 140) | IBRG - sbl | -116 | -83667 | 39.03 | 1.09 | 15.66 | 89.2 | 4.1 | 0:10:34 |
| | IBRG + sbl | -118 | -85564 | 39.04 | 1.02 | 15.58 | 89.4 | 5.4 | 0:11:18 |
| U5 | Start | -62 | -27489 | 34.08 | — | — | 82.0 | 0.0 | — |
| (119 228) | IBRG - sbl | -59 | -9333 | 34.34 | -0.03 | 1.72 | 82.5 | 0.0 | 0:08:20 |
| | IBRG + sbl | -59 | -9526 | 34.39 | -0.05 | 1.03 | 82.9 | 0.0 | 0:08:55 |
| U6 | Start | -122 | -225276 | 198.86 | — | — | 88.6 | 24.0 | — |
| (254 208) | IBRG - sbl | -108 | -188177 | 193.66 | 0.14 | 13.19 | 89.0 | 42.7 | 0:21:01 |
| | IBRG + sbl | -109 | -188048 | 194.64 | 0.13 | 12.80 | 88.9 | 34.1 | 0:22:53 |
| U7 | Start | -55 | -63792 | 100.10 | — | — | 83.5 | 0.0 | — |
| (276 799) | IBRG - sbl | -55 | -27269 | 101.90 | -0.01 | 1.06 | 84.1 | 0.4 | 0:16:29 |
| | IBRG + sbl | -55 | -24928 | 101.96 | -0.01 | 0.90 | 83.7 | 0.0 | 0:18:01 |
| U8 | Start | -87 | -834668 | 729.16 | — | — | 86.1 | 44.1 | — |
| (1 681 671) | IBRG - sbl | -75 | -354967 | 739.78 | -0.10 | 2.79 | 86.4 | 29.7 | 2:07:58 |
| | IBRG + sbl | -75 | -359301 | 739.30 | -0.10 | 2.33 | 86.5 | 27.0 | 2:11:26 |

TABLE 7.2. Experimental results for GRBO comparing a run with usage of shape-based edge lengths ("IBRG + sbl") to one without ("IBRG - sbl").

When comparing "IBRG + sbl" to "IBRG - sbl", the ΔWL and ΔAWL columns display the differences with regard to our objective of minimizing the amount of added wire length most directly. However, we are of course also interested in the effects that these differences have on general metrics, which are illustrated by the numbers in the other columns.

| Unit (# nets) | After... | WS [ps] | FOM [ps] | PWR [mW] | ΔWL [%] | ΔAWL [%] | wACE4 [%] | OFtgt [100 pitch$^2$] | RT [h:mm:ss] |
|---|---|---|---|---|---|---|---|---|---|
| U1 | Start | -132 | -46373 | 55.05 | — | — | 91.0 | 60.4 | — |
| (77 528) | IBRG - sbl | -45 | -33013 | 59.02 | -0.01 | 3.57 | 90.5 | 56.4 | 0:05:36 |
| | IBRG + sbl | -44 | -32892 | 59.04 | -0.03 | 2.48 | 90.7 | 51.3 | 0:05:38 |
| U2 | Start | -112 | -81335 | 22.73 | — | — | 88.1 | 7.8 | — |
| (79 119) | IBRG - sbl | -86 | -75395 | 23.04 | 0.06 | 3.26 | 88.1 | 10.4 | 0:06:47 |
| | IBRG + sbl | -84 | -76904 | 23.09 | 0.05 | 1.79 | 87.7 | 7.8 | 0:06:01 |
| U3 | Start | -153 | -159431 | 71.15 | — | — | 92.1 | 98.4 | — |
| (100 827) | IBRG - sbl | -78 | -114716 | 74.67 | 0.0 | 2.74 | 90.9 | 43.2 | 0:07:12 |
| | IBRG + sbl | -79 | -113313 | 74.69 | 0.01 | 1.65 | 91.5 | 75.0 | 0:07:01 |
| U4 | Start | -200 | -215175 | 37.45 | — | — | 95.6 | 208.4 | — |
| (111 140) | IBRG - sbl | -104 | -181698 | 39.40 | 0.07 | 3.05 | 96.0 | 224.1 | 0:08:14 |
| | IBRG + sbl | -101 | -183683 | 39.33 | 0.08 | 1.80 | 95.5 | 199.0 | 0:08:01 |
| U5 | Start | -64 | -57007 | 34.65 | — | — | 79.0 | 0.0 | — |
| (119 228) | IBRG - sbl | -59 | -26671 | 35.26 | 0.0 | 3.46 | 79.9 | 0.2 | 0:08:28 |
| | IBRG + sbl | -61 | -26191 | 35.16 | 0.0 | 1.99 | 79.0 | 0.0 | 0:08:28 |
| U6 | Start | -134 | -315633 | 199.99 | — | — | 98.3 | 2985.0 | — |
| (254 208) | IBRG - sbl | -119 | -275496 | 197.19 | 0.03 | 3.29 | 98.4 | 3056.4 | 0:18:31 |
| | IBRG + sbl | -118 | -273741 | 197.37 | 0.02 | 2.52 | 98.3 | 2965.5 | 0:17:58 |
| U7 | Start | -94 | -180073 | 101.69 | — | — | 95.8 | 1257.4 | — |
| (276 799) | IBRG - sbl | -58 | -79064 | 104.08 | -0.03 | 2.42 | 95.4 | 1110.6 | 0:16:21 |
| | IBRG + sbl | -55 | -82463 | 103.21 | -0.03 | 1.63 | 95.7 | 1192.1 | 0:16:05 |
| U8 | Start | -105 | -1752478 | 739.41 | — | — | 95.0 | 13456.3 | — |
| (1 681 671) | IBRG - sbl | -100 | -1469722 | 756.94 | -0.05 | 2.64 | 95.1 | 13737.9 | 2:21:52 |
| | IBRG + sbl | -101 | -1462652 | 756.23 | -0.05 | 1.98 | 95.0 | 13347.0 | 2:25:16 |

TABLE 7.3. Experimental results for DRBO comparing a run with usage of shape-based edge lengths ("IBRG + sbl") to one without ("IBRG - sbl").

The first thing to notice is that our shape-based edge lengths have a clear positive impact on the amount of added wire length. This can be seen by looking at the columns labeled ΔWL and ΔAWL, which generally show smaller numbers when shape-based edge lengths are used. In this regard, ΔAWL is more expressive, as all reroutes are tracked in this number, while ΔWL basically only tracks changes in nets that persist until the end of

the incremental routing process. In particular, if a transaction is undone due to a large detour, then this detour is tracked by ΔAWL, but not by ΔWL. Moreover, ΔWL is often close to zero, and can also be subject to some variation in the incremental routing process, e.g. the number of transactions being performed.

Looking only at GRBO first, the ΔAWL numbers show that if congestion is low, then the amount of wire length that we add during our reroutes is already close to our approximate lower bound. We can also see that on uncongested test cases, using our shape-based edge lengths closes the gap between our solution and our lower bound by a relatively large amount. As congestion gets higher, the deviation from the lower bound gets significantly larger, as can be observed in the GRBO runs on U4 and U6. On these units, the relative ΔAWL differences between our two runs also become much smaller, as there the ΔAWL number is largely determined by the amount of detours caused by congestion, which affects both runs in the same way.

However, as already explained in Section 7.3.3.1, this is expected due to our aggressive congestion prices, which encourage detours instead of creating routing overflow. In particular, if a pin is moved into a neighboring tile during an operation such as gate sizing in a congested region, then comparatively large detours might be necessary to close the connection.

Surprisingly at first, high wACE4 numbers do not seem to imply high ΔAWL numbers in DRBO. This seems to contradict our observation that high congestion causes higher ΔAWL numbers. However, a look at the congestion plots from Figure 7.6 helps to understand this effect: The congestion plots are taken from the DRBO runs on U4. On the left side we see the congestion plot directly after global routing, and on the right side we see the congestion plot that we get after initializing BonnRouteGlobal after detailed routing. This means that the left congestion map would be the starting point for running GRBO, and the right the one for DRBO. One can see that the left plot contains large areas that are predominantly dark yellow and orange, which corresponds to around 90% edge usage. As our congestion target is 90%, BonnRouteGlobal regards these regions as very critical. In contrast to that, the right plot is largely green, but contains a few heavily congested edges due to local detailed routing congestion. By definition of the wACE4 metric [**120**], these few heavily congested edges cause a high wACE4 number of over 95% on the right side, while the wACE4 is below 90% on the left side. However, it is easy to imagine that working around the congestion hotspots is much harder given the left congestion map than given the right one. This explains the ΔAWL differences between GRBO and DRBO on designs with a large wACE4 number.

Getting a better match between the congestion maps after global and detailed routing is an ongoing project in BonnRouteGlobal. However, to some extent, these differences

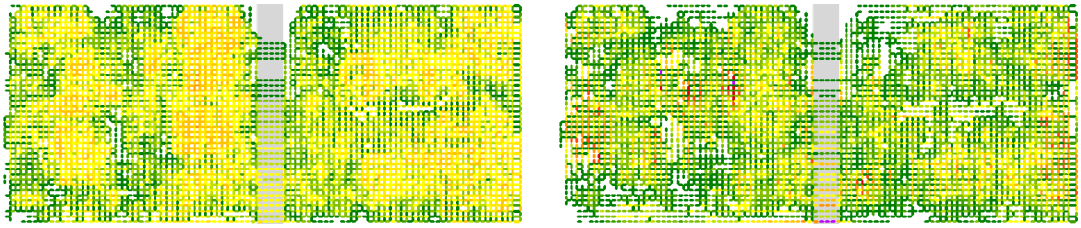


FIGURE 7.6. Congestion maps on U4 at different points in the routing flow: Left directly after global routing, and right directly after initializing BonnRouteGlobal after detailed routing.

are inherent in our ways of measuring congestion before and after detailed routing (cf. Section 7.1.3), as we also explain later in Section 7.6.

Unfortunately, the wire length improvements gained through the use of shape-based edge lengths do not translate into a consistent improvement of timing metrics — all in all, possible improvements obtained by using shape-based edge lengths seem to be getting largely lost in the usual fluctuations that occur from run to run.

On the flip side, one can observe a small running time increase in GRBO when shape-based edge lengths are used. The reason for this is that the "IBRG - sbl" run uses future costs [47] for the objective function, while we disable them for the "IBRG + sbl" run, as future costs are harder to obtain in this scenario. Looking at the results, it is not clear whether it is worthwhile to invest the effort to obtain good future costs for the scenario where shape-based edge lengths are used. Disabling objective function future costs in the "IBRG - sbl" run adjusts the running times to roughly match the ones from the "IBRG + sbl" run. These running time differences cannot be seen in DRBO, as here we rarely encounter larger spots of high congestion and pin movements are generally more local than in GRBO. This lessens the impact of a missing future cost on the total running time.

Our conclusion is that when it comes to the objective of minimizing the amount of added wire length, then using our shape-based edge lengths brings consistent and noticeable improvements. However, the improvements seem to be too local to make a noticeable difference in the general metrics that we are tracking in our runs. Therefore, to make further improvements, it seems more worthwhile to improve the incremental routing process to be able to work better on congested designs, e.g. by allowing to rip out bystander nets when required. Another possible way to improve results could be to do incremental routing in a timing-aware fashion similar to what is described in Chapters 4 and 5, as this should result in better routing topologies and layer choices, which should positively impact timing.

## 7.4. Copy Routes

In this section we discuss the Copy Routes Problem, which occurs when a routed net $N$ is subdivided into multiple nets $N_1, \dots, N_k$ by buffer insertion (cf. Section 1.2.2) during GRBO or DRBO. As already outlined in Section 7.2.2, we must then first decide how to distribute the wiring of $N$ among $N_1, \dots, N_k$ before we can use the minimal reroute procedure from Section 7.3 to route every net $N_i, i = 1, \dots, k$. The Copy Routes Problem presented in this section only covers the case where buffers are inserted, but not where buffers are removed. Dealing with the removal of buffers would be a possible extension for the future.

This section consists of four subsections: In Section 7.4.1 we state some basic definitions and outline the process of *buffering along the route*, which is used by the buffering tools in our GRBO and DRBO flows. Afterwards, we formally introduce the Copy Routes Problem in Section 7.4.2, which is then simplified to the Simplified Copy Routes Problem in Section 7.4.3. Section 7.4.3 also describes our algorithm for solving the Simplified Copy Routes Problem, which is then validated in practice by our experimental results given in Section 7.4.4.

### 7.4.1. Buffering Routes.

The buffer tree is given as a graph:

Definition 7.8. Let $N$ be a net with source $s$ and sinks $T$. Then a *buffer tree netlist graph for* $N$ is an arborescence $X$ rooted at $s$ such that $T = \{x \in V(X) : \delta_X^+(x) = \emptyset\}$. The vertices in $V(X) \setminus N$ are called *buffers*, and we associate $X$ with buffer positions $p\colon V(X) \setminus N \to S$, where $S$ is the layered chip area. $X$ defines a set of *target nets* $\mathcal{N}_X := \{N_x : x \in V(X) \setminus T\}$, where $N_x := \{x\} \cup \Gamma_X^+(x)$ for $x \in V(X) \setminus T$ is the net driven by $x$.

Basically, the buffer tree netlist graph is a condensed section of the netlist graph from Section 1.1.3, where pins belonging to the same buffer are contracted into one vertex. A buffer tree netlist graph is also a graph that is embedded into the layered chip area in the sense of Definition 2.3, but in this case the embedding is in general not rectilinear.

A common strategy for buffering a given Steiner tree $Y$ for a net $N = \{s\} \cup T$ is to *buffer along the route*, which corresponds to the following procedure:

(1) Clone vertices $v \in V(Y)$ with $|\delta_Y(v)| \geq 3$ appropriately by introducing new vertices and zero-length edges.
(2) Subdivide the edges of $Y$ appropriately by inserting Steiner points of degree 2.
(3) Pick a subset $B \subseteq V(Y)$ where buffers should be placed. The maximum subtrees of $Y$ rooted at some vertex in $\{s\} \cup B$ and with leaves in $T \cup B$ define the buffer tree netlist graph $X$ with $V(X) = N \cup B$: Each such maximum subtree

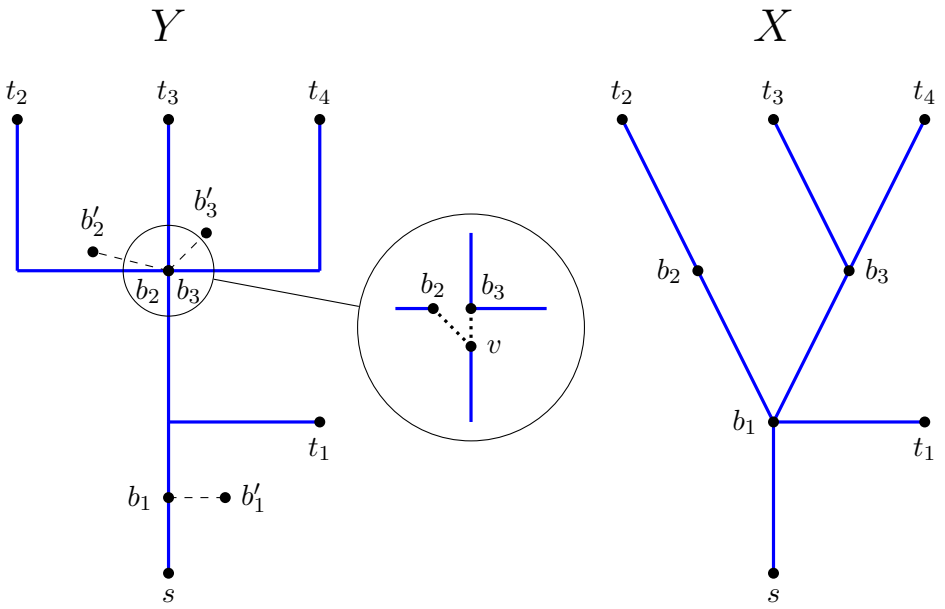



FIGURE 7.7. Illustration of buffering along the route: The buffering tool inserts three buffers $b_1, b_2$ and $b_3$ on $Y$ (left), which results in the buffer tree netlist graph $X$ (right). Here, $b_2$ and $b_3$ are located on different copies of the same vertex. This situation is shown "under the magnifying glass" in the middle: $v$, $b_2$, and $b_3$ are located on the same position, and the dotted edges have zero length. After placement legalization, each buffer $b_i$ is moved to a modified location $b'_i$, $i = 1, 2, 3$.

of $Y$ corresponds to one net in $X$, and the driver of the net is the vertex in $\{s\} \cup B$ that is closest to $s$ in $Y$.

An illustration is given by Figure 7.7. If such a strategy was applied and every buffer was placed exactly at its position on the route, distributing the wires into the target nets would be a trivial task. However, in practice, this is not the case: The space on the placement layer below $Y$ might be largely blocked, and the buffers might end up at positions that do not coincide with their originally intended positions. At this point, distributing the wires from the root net into the target nets becomes non-trivial. Based on this buffering strategy, we only deal with certain kinds of buffer tree netlist graphs for the rest of this section:

DEFINITION 7.9. Let $Y$ be a Steiner tree for a net $N$ with source $s$ and sinks $T$, and $X$ be a buffer tree netlist graph for $N$. Then $X$ is said to *match the topology of* $Y$ if for all $x \in V(X)$ there exists $v \in V(Y)$ such that $T(X(x)) = T(Y(v))$.

A buffer tree netlist graph resulting from buffering along the route naturally matches the topology of the corresponding routing tree.

**7.4.2. The Copy Routes Problem.** Before stating our problem formulation, we first need to define what it means to distribute the wires of the root net among the target nets:

Definition 7.10. Given a Steiner tree $Y$ and a buffer tree netlist graph $X$ for a net $N$, a *wire distribution* is a function $f\colon E(Y) \to \mathcal{N}_X$ describing the distribution of the wires corresponding to $E(Y)$ among $\mathcal{N}_X$. Given a target net $N_x \in \mathcal{N}_X$, we let $\mathrm{smt}(N_x, f)$ denote the optimum objective function value of the Rectilinear Minimum Steiner Tree with Prewires Problem (cf. Section 2.3.3) given $N_x$ and the Steiner forest defined by $f^{-1}(N_x)$ as input. Moreover, the rectilinear Steiner length of $N_x$ (without prewires) is denoted by $\mathrm{smt}(N_x)$, and we set $\mathrm{smt}(\mathcal{N}_X) := \sum_{N_x \in \mathcal{N}_X} \mathrm{smt}(N_x)$ and $\mathrm{smt}(\mathcal{N}_X, f) := \sum_{N_x \in \mathcal{N}_X} \mathrm{smt}(N_x, f)$.

We would like to compute a wire distribution such that the total additional wire length required to route each target net is minimum. This leads to the following problem formulation:

Problem 7.11: Copy Routes Problem

**Input:** A net $N$, a Steiner tree $Y$ connecting the exact shapes of $N$, a buffer tree netlist graph $X$ for $N$ that matches the topology of $Y$.

**Task:** Compute a subdivision $Y'$ of $Y$ and a wire distribution $f\colon E(Y') \to \mathcal{N}_X$ maximizing $\mathrm{gain}(f) := \mathrm{smt}(\mathcal{N}_X) - \mathrm{smt}(\mathcal{N}_X, f)$.

Maximizing $\mathrm{gain}(f)$ is equivalent to minimizing $\mathrm{smt}(\mathcal{N}_X, f)$, as $\mathrm{smt}(\mathcal{N}_X)$ is a constant. The reason for maximizing $\mathrm{gain}(f)$ as objective function instead of minimizing $\mathrm{smt}(\mathcal{N}_X, f)$ is as follows: As we will later see in Section 7.4.4, $\min\{\mathrm{smt}(\mathcal{N}_X, f) : f \text{ is a wire distribution}\}$ is relatively small compared to $\mathrm{smt}(\mathcal{N}_X)$ on most instances. Therefore, minimizing $\mathrm{smt}(\mathcal{N}_X, f)$ is generally not well-behaved with respect to approximation in practice, as local imperfections can result in significant multiplicative deviations from the optimum, even though the majority of the wiring may have been optimally distributed. This is particularly true when comparing experimental results to lower bounds instead of the optimum, which introduces additional errors. On the other hand, $\mathrm{gain}(f)$ expresses how much newly added wire length we saved by preserving the old wiring, and from a practical perspective, a high gain that is relatively close to $\mathrm{smt}(\mathcal{N}_X)$ should indicate that the instance has been sucessfully solved. An example from practice is given by Figure 7.8.

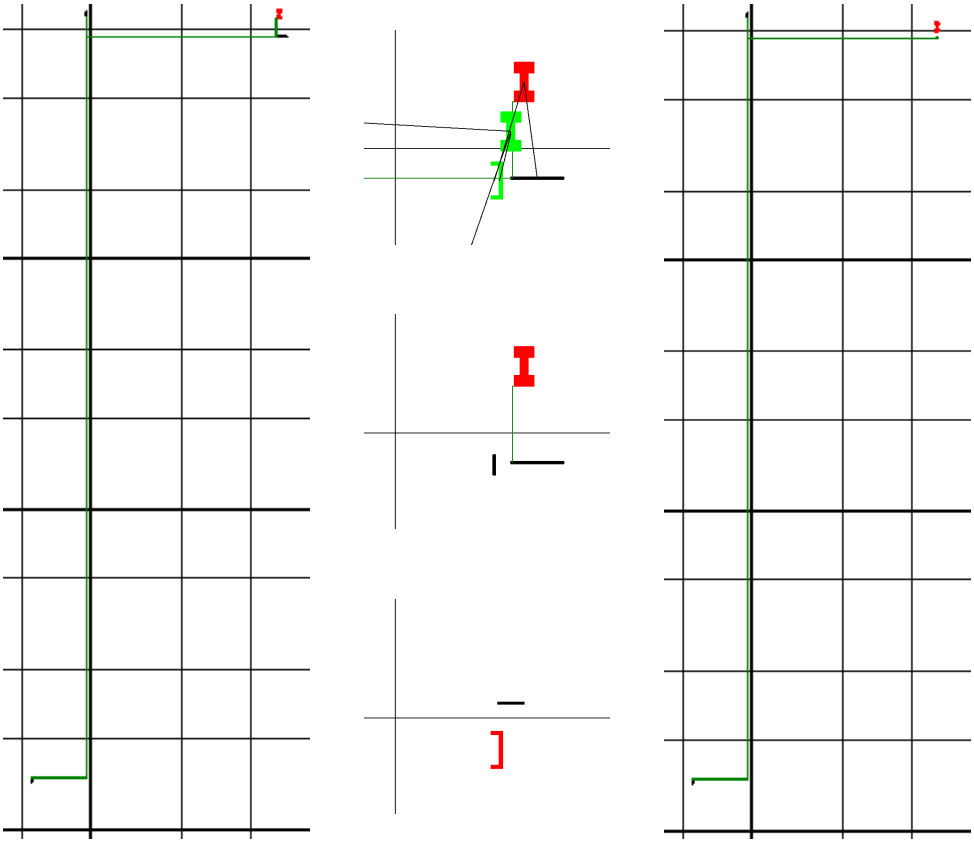


FIGURE 7.8. Example of a wire distribution $f$ from practice where $\text{gain}(f)$ is good, but where we cannot prove reasonable bounds for $\text{smt}(\mathcal{N}_X, f)$: We start with a routed net with four pins, where the source pin is colored red, the sink pins are colored black, and the route is colored dark green (left picture). The buffering tool then inserts two buffers (light green) close to the source pin (upper central picture). Here, the diagonal black lines indicate the buffer tree netlist graph: The original source pin connects to the lower buffer and the black sink pin, the lower buffer connects to the upper buffer, and the upper buffer connects to the remaining two sink pins of the root net. This results in three target nets (center, lower center, and right), and most of the wiring is assigned to the long target net in the right picture. With our bounds from Section 7.4.4 we can prove that $\text{gain}(f)$ is less than 1% away from the optimum, but for $\text{smt}(\mathcal{N}_X, f)$ we can only prove that it is within a factor of 2 within the optimum. In this example, the wire distribution is very close to optimum for both metrics, and the problem is to compute bounds that prove it. However, it is easy to see that in similar situations local suboptimalities can lead to $\text{smt}(\mathcal{N}_X, f)$ being a large factor away from optimum, although the majority of the wiring has been successfully assigned, which is what matters for practical purposes.

A wire distribution assigns every wire to some target net — after all, it would not be useful anyway to omit assigning certain wires in our theoretical formulation of the Copy Routes Problem. In practice, however, not assigning certain wires can be beneficial: When $Y$ consists of detailed wires, as it is predominantly the case in DRBO, distributing wires among the target nets in an unrestricted manner is likely to cause violations of minimum distance rules between detailed wires assigned to different target nets. Therefore, we apply a post-optimization routine that cuts away minor parts of the assigned detailed wiring where it is necessary to avoid violations of distance rules. As the minimum distance rule violations that are caused during the copy routes process can generally be resolved by this routine by working only locally, we leave out this aspect in our formulation of the Copy Routes Problem in order to keep the problem formulation more concise.

**7.4.3. The Simplified Copy Routes Problem.** Instead of trying to approximate the Copy Routes Problem, we will instead turn to a simpler version. Our experimental results from Section 7.4.4 will verify that solving the simpler version gives good approximate solutions for the Copy Routes Problem in practice. The first simplification is to restrict the set of possible wire distributions:

Definition 7.12. Let $N$ be a net, $Y$ be a Steiner tree for $N$ and $X$ be a buffer tree netlist graph for $N$ that matches the topology of $Y$. A *buffer mapping* is a function $q\colon V(X) \to V(Y)$ such that

(1) $q(\pi) = \pi$ for $\pi \in N$,
(2) $T(Y(q(x))) = T(X(x))$ for all $x \in V(X)$,
(3) $y \in V(X(x)) \Rightarrow q(y) \in V(Y(q(x)))$ for all $(x, y) \in V(X)^2$.

Given a buffer mapping $q$, the *wire distribution* $f_q$ *associated with* $q$ is defined by

$$f_q^{-1}(N_x) = E(Y(q(x))) \setminus \bigcup_{y \in \Gamma_X^+(x)} E(Y(q(y))),$$

where $N_x \in \mathcal{N}_X$ is the target net rooted at $x \in V(X) \setminus T$.

The assumption that $X$ matches the topology of $Y$ makes sure that a buffer mapping fulfilling (2) and (3) always exists. We note that $f_q^{-1}(N_x)$ for $N_x \in \mathcal{N}_X$ always defines a Steiner tree (as opposed to a more general Steiner forest).
We are now able to define a simplified version of the Copy Routes Problem, where we ask for a buffer mapping instead of a more general wire distribution, and we also simplify the objective function:

PROBLEM 7.13: SIMPLIFIED COPY ROUTES PROBLEM

**Input:** A net $N$, a Steiner tree $Y$ connecting the exact shapes of $N$, a buffer tree netlist graph $X$ for $N$ that matches the topology of $Y$.

**Task:** Compute a subdivision $Y'$ of $Y$ and a buffer mapping $q\colon V(X) \to V(Y')$ such that $\text{dist}(X, q) := \sum_{x \in V(X)} \text{dist}(p(x), q(x))$ is minimized.

In the context of the SIMPLIFIED COPY ROUTES PROBLEM we say that *a subdivision $Y'$ admits an optimum buffer mapping* if there exists a buffer mapping $q\colon V(X) \to V(Y')$ attaining the optimum objective function value of the given instance of the SIMPLIFIED COPY ROUTES PROBLEM.

The justification for this formulation of the SIMPLIFIED COPY ROUTES PROBLEM goes back to the strategy of buffering along the route (cf. Section 7.4.1): We know that the idealized buffer positions before placement legalization (cf. Section 1.2.1) were fulfilling the requirements of Definition 7.12. Therefore, our buffer mapping can be seen as a reconstruction of these idealized buffer positions, although local deviations are possible and required to adapt to the distortion caused by placement legalization. The objective function is based on the assumption that given a buffer mapping $q$, the routing of each target net $N_x \in \mathcal{N}_X$ will often not deviate very much from using $f_q^{-1}(N_x)$ as trunk and connecting each $y \in N_x$ to this trunk in the shortest possible way. Our experimental results from Section 7.4.4 will prove the validity of our assumptions and show that the SIMPLIFIED COPY ROUTES PROBLEM is a good approximation of the COPY ROUTES PROBLEM in practice. An illustration of the SIMPLIFIED COPY ROUTES PROBLEM and the concept behind buffer mappings is given as Figure 7.9.

Before proceeding with the description of our algorithm, we require a technical property of $Y$:

DEFINITION 7.14. Let $Y$ be a Steiner tree for a net with source $s$ and sinks $T$. Then $Y$ is said to have a *sufficiently cloned vertex set* if the implication $\text{dist}(v, w) > 0 \Rightarrow T(Y(v)) = T(Y(w))$ holds for all $(v, w) \in E(Y)$.

The reasoning behind Definition 7.14 is as follows: As we are not only computing a buffer mapping in the SIMPLIFIED COPY ROUTES PROBLEM, but also a subdivision of $Y$, we are basically mapping buffers to edges in $Y$ — mapping buffers to vertices of a subdivision just allows for a more comprehensible description. Requiring that $Y$ has a sufficiently cloned vertex set then goes back to requirement (2) of Definition 7.12: Given $x \in V(X)$ and $(v, w) \in E(Y)$ with $\text{dist}(v, w) > 0$, we can either map $x$ to $v$ and $w$ or to none of them. This property will simplify our subsequent descriptions, as we will see later.

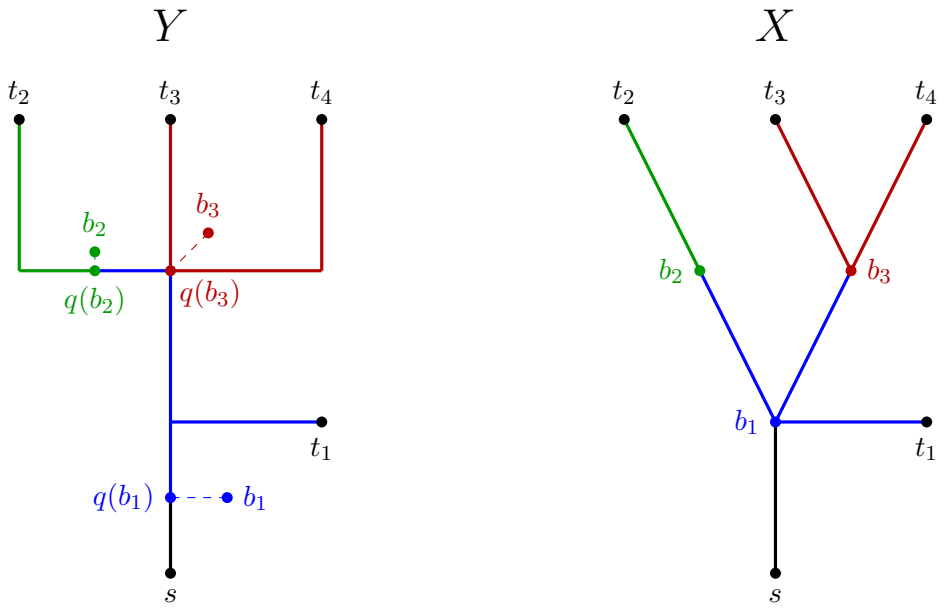



FIGURE 7.9. The instance of the SIMPLIFIED COPY ROUTES PROBLEM that originates from buffering along the route as in Figure 7.7. The buffer mapping $q$ defines a wire distribution, which is illustrated by the use of different colors. Note that there are multiple vertices at the location of $q(b_3)$, which are interconnected by edges of length zero.

Any Steiner tree can easily be transformed into one with a sufficiently cloned vertex set in linear time. Therefore, we assume from here on implicitly that our given routing tree has a sufficiently cloned vertex set and only mention it where it is required for correctness.

7.4.3.1. *The Simplified Copy Routes Problem on Paths.* As it will emerge as a subproblem when solving the SIMPLIFIED COPY ROUTES PROBLEM, we first consider the special case where the buffer tree netlist graph is a path. Here, we first show how to compute a sufficiently fine-grained subdivision of $Y$. For this we basically use the well-known Hanan grid [46] induced by the buffer coordinates:

DEFINITION 7.15. Let $N$ be a net and $X$ be a buffer tree netlist graph for $N$. Then we call the tuple $H(X) := (H_x(X), H_y(X))$ with $H_d(X) := \{p(v)_d : v \in V(X)\}$ for $d \in \{x, y\}$ the *Hanan grid* induced by $X$. Given a Steiner tree $Y$, the *subdivision of $Y$ through $H(X)$* is the unique subdivision $Y'$ of $Y$ such that

(1) $(\min\{p(v)_d, p(w)_d\}, \max\{p(v)_d, p(w)_d\}) \cap H_d(X) = \emptyset$ for all $(v, w) \in E(Y')$ and $d \in \{x, y\}$,
(2) $|E(Y')|$ is minimum among all subdivisions of $Y$ fulfilling (1).

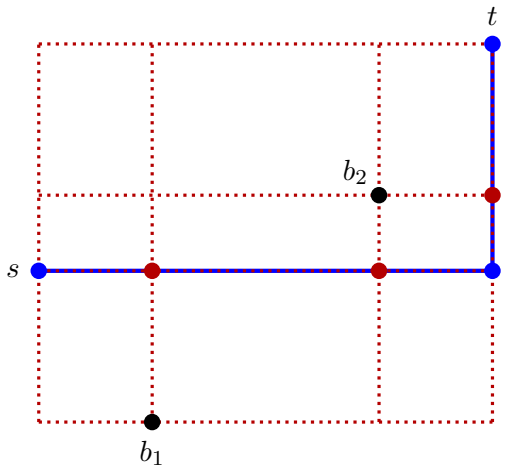



FIGURE 7.10. Illustration of Definition 7.15: $Y$ is an $s$-$t$ path with three vertices (blue). $X$ is a path with $V(X) = \{s, b_1, b_2, t\}$, which induces the Hanan grid depicted by the dotted red lines. At the intersection points of the Hanan grid and $Y$ we insert new vertices (red), which yields the subdivision of $Y$ through $H(X)$.

An illustration is given by Figure 7.10. It is easy to compute the subdivision through the Hanan grid:

PROPOSITION 7.16. *Let $Y$ be a Steiner tree and $X$ be a buffer tree netlist graph for a net $N$, and let $Y'$ be the subdivision of $Y$ through $H(X)$. Moreover, let $n := |V(Y)|$, $n' := |V(Y')|$ and $k := |V(X)|$. Then $n' = \mathcal{O}(nk)$ and $Y'$ can be computed in $\mathcal{O}(n' + (n+k)\log k) = \mathcal{O}(nk + k\log k)$ time.*

PROOF. We sort $H_x(X)$ and $H_y(X)$ in ascending order in $\mathcal{O}(k \log k)$ time. Consider an edge $(v, w) \in E(Y)$ and assume $p(v)_x < p(w)_x$ without loss of generality. Using binary search we can determine $H(v, w) := (p(v)_x, p(w)_x) \cap H_x(X)$ in $\mathcal{O}(\log k + |H(v, w)|)$ time and (recursively) subdivide $(v, w)$ appropriately. Applying this procedure for all $e \in E(Y)$ yields the claimed running time. □

The subdivision through the Hanan grid is important as it admits an optimum buffer mapping:

PROPOSITION 7.17. *Consider an instance of the* SIMPLIFIED COPY ROUTES PROBLEM *where $X$ is a path, and let $Y'$ be the subdivision of $Y$ through $H(X)$. Then $Y'$ admits an optimum buffer mapping.*

PROOF. Assume the contrary and let $Y^*$ be a subdivision of $Y'$ with minimum number of edges that admits an optimum buffer mapping $q^* \colon V(X) \to V(Y^*)$. Let

$(v,w) \in E(Y')$ such that the set $B := \{b \in V(X) : q^*(b) \in V(P_{Y^*}(v,w)) \setminus \{v,w\}\}$ is non-empty — such an edge must exist as otherwise the image of $q^*$ would be $V(Y')$. Without loss of generality we assume $p(v)_x = p(w)_x$ and $p(v)_y \leq p(w)_y$.
For $b \in B$ we define $f_b \colon [0, \operatorname{dist}(v,w)] \to \mathbb{R}_{\geq 0}$ by $f_b(l) := \operatorname{dist}(p(b), (p(v)_x, p(v)_y + l))$. By definition of $Y'$ we have $p(b)_y \notin (p(v)_y, p(w)_y)$ for $b \in B$. Therefore, $f_b$ for $b \in B$ is a linear function, and so is the sum $\sum_{b \in B} f_b$. This means that we can either set $q^*(b) = v$ or $q^*(b) = w$ for all $b \in B$ without increasing the objective function value: If $\operatorname{dist}(v,w) = 0$, then setting $q^*(b) = w$ suffices. Otherwise, if $\operatorname{dist}(v,w) > 0$, we make this choice dependent on $\sum_{b \in B} f_b$.
This operation will yield a valid buffer mapping according to Definition 7.12: Property (1) is still fulfilled as we clearly have $B \cap N = \emptyset$. Property (2) is fulfilled as we have $T(Y^*(v)) = T(Y^*(w))$ if $\operatorname{dist}(v,w) > 0$ (cf. Definition 7.14), so mapping all $b \in B$ to $v$ does not violate property (2) (for $w$ this is naturally the case). Finally, property (3) is fulfilled as we move all $b \in B$ at once to the same vertex. This leads to a contradiction to the choice of $Y^*$, as we could replace $P_{Y^*}(v,w)$ by a single edge and obtain a subdivision of $Y'$ with fewer edges that still admits an optimum buffer mapping. □

We now show how to compute an optimum buffer mapping for a given subdivision. Together with Proposition 7.17 this will yield an optimal algorithm:

THEOREM 7.18. *Consider an instance of the* SIMPLIFIED COPY ROUTES PROBLEM *where $X$ is a path, and assume that $Y$ already admits an optimum buffer mapping. Then we can compute an optimum buffer mapping $q \colon V(X) \to V(Y)$ in $\mathcal{O}(nk)$ time, where $n := |V(Y)|$ and $k := |V(X)|$.*

PROOF. We show how the above problem can be reduced to finding a shortest path in an acyclic digraph. As $X$ and $Y$ are paths, we can describe $X$ by a sequence $(x_1, \ldots, x_k)$ and $Y$ by a sequence $(v_1, \ldots, v_n)$ of their vertices, where $s = x_1 = v_1$ and $t = x_k = v_n$. Consider the digraph $D$ with $V(D) := V(X) \times V(Y)$ and $E(D) := E_X \cup E_Y$, where

- $E_X := \{((x_l, v_i), (x_{l+1}, v_i)) : l = 1, \ldots, k-1, i = 1, \ldots, n\}$,
- $E_Y := \{((x_l, v_i), (x_l, v_{i+1})) : l = 1, \ldots, k, i = 1, \ldots, n-1\}$.

An illustration of $D$ is given as Figure 7.11. We define costs $c \colon E(D) \to \mathbb{R}_{\geq 0}$ by setting

$$c((x_l, v_i), (x_{l+1}, v_i)) := \begin{cases} \operatorname{dist}(x_l, v_i) & 2 \leq l \leq k-1, i = 1, \ldots, n, \\ 0 & l = 1, i = 1, \ldots, n, \end{cases}$$

for $((x_l, v_i), (x_{l+1}, v_i)) \in E_X$ and $c(e) = 0$ for $e \in E_Y$. We note that the edges $((s, v_i), (x_2, v_i))$, $i = 2, \ldots, n$, and $((s, v_i), (s, v_{i+1}))$, $i = 1, \ldots, n-1$, are superfluous, but we added them for notational convenience.

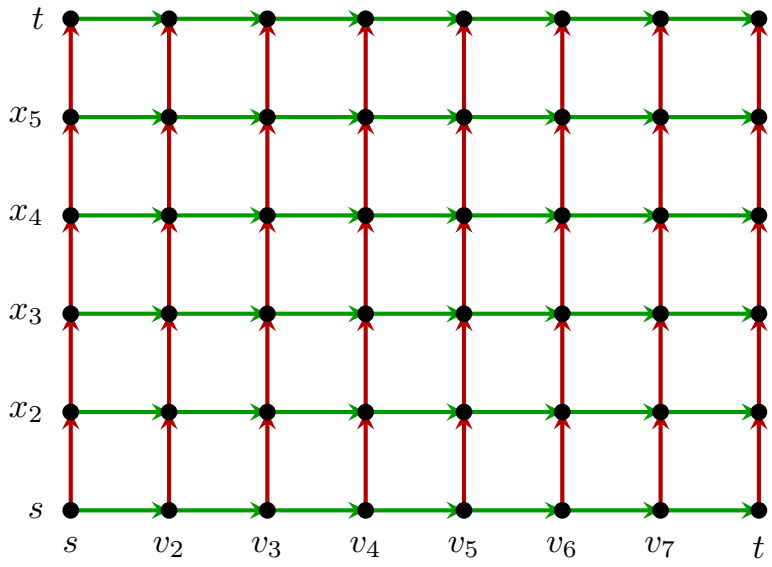



FIGURE 7.11. Illustration of the digraph $D$ from the proof of Theorem 7.18 with $k = 6$ and $n = 8$. Horizontal green edges belong to $E_Y$, while vertical red edges belong to $E_X$.

Given an $(s, s)$-$(t, t)$ path $P$ in $D$ we can derive a buffer mapping $q\colon V(X) \to V(Y)$ by setting $q(x_l) = v$ for every edge $((x_l, v), (x_{l+1}, v)) \in E(P) \cap E_X$ with $l \geq 2$. Conversely, given a buffer mapping $q\colon V(X) \to V(Y)$, we can derive an $(s, s)$-$(t, t)$ path $P$ in $D$ with $E(P) \cap E_X = \{((s, s), (x_2, s))\} \cup \{((x_l, q(x_l)), (x_{l+1}, q(x_l))) : l = 2, \ldots, k-1\}$ — by the structure of $D$, this already defines $P$. In both cases we have $c(P) = \operatorname{dist}(X, q)$, and so we can find an optimum buffer mapping by searching for a shortest $(s, s)$-$(t, t)$ path in $D$. As $D$ is acyclic, the running time can be bounded by $\mathcal{O}(|E(D)|) = \mathcal{O}(nk)$ time. □

This yields the following theorem:

THEOREM 7.19. *The* SIMPLIFIED COPY ROUTES PROBLEM *where $X$ is a path can be solved optimally in $\mathcal{O}(nk^2)$ time with $n := |V(Y)|$ and $k := |V(X)|$.*

PROOF. We compute the subdivision $Y'$ through $H(X)$ and an optimum buffer mapping $q\colon V(X) \to V(Y')$ as described in Theorem 7.18. As we have $|V(Y')| = \mathcal{O}(nk)$ according to Proposition 7.16, the total running time amounts to $\mathcal{O}(|V(Y')|k) = \mathcal{O}(nk^2)$. □

7.4.3.2. *The General Simplified Copy Routes Problem.* We now turn to the general version of the SIMPLIFIED COPY ROUTES PROBLEM, where $X$ is not restricted to be a path. We first show how to identify the set of vertices in $Y$ that a buffer may be mapped to according to (2) of Definition 7.12:

Definition 7.20. Consider an instance of the Simplified Copy Routes Problem. We define the *allowed path* $Q(x)$ of $x \in V(X) \setminus N$ as the path in $Y$ with $V(Q(x)) = \{v \in V(Y) : T(Y(v)) = T(X(x))\}$.

It is easy to see that $Q(x)$ is a path: We just consider the vertex $w \in \{v \in V(Y) : T(Y(v)) = T(X(x))\}$ with maximum distance to $s$ and successively traverse its incoming edges until $s$ or a vertex $u$ with $|\delta_Y^+(u)| \geq 2$ is reached. Note that $V(Q(x))$ for $x \in X$ is never empty if $X$ matches the topology of $Y$. We can restrict ourselves to mapping every buffer to a subdivision of its allowed path:

Lemma 7.21. *Consider an instance of the* Simplified Copy Routes Problem*, and for $x \in V(X) \setminus N$ let $Q(x) = P_Y(q_1(x), q_2(x))$ for some $q_1(x), q_2(x) \in V(Y)$. Then there exists a subdivision $Y'$ of $Y$ and an optimum buffer mapping $q \colon V(X) \to V(Y')$ such that $q(x) \in P_{Y'}(q_1(x), q_2(x))$ for all $x \in V(X) \setminus N$.*

Proof. This follows directly from the fact that we require $Y$ to have a sufficiently cloned vertex set (cf. Definition 7.14): For $x \in V(X) \setminus N$ we either have $q_1(x) = s$ or the incoming edge of $q_1(x)$ has length zero, and can therefore be assumed to exist in $Y'$ without being subdivided. The claim then follows. □

We continue with another lemma:

Lemma 7.22. *Consider an instance of the* Simplified Copy Routes Problem *and let $(x, y) \in E(X)$ with $x, y \in V(X) \setminus N$. Then if $|\delta_X^+(x)| = 1$, we have $Q(x) = Q(y)$. Otherwise we have $V(Q(x)) \cap V(Q(y)) = \emptyset$ and $Q(x)$ is a subpath of the $s$-$v$ path in $Y$ for any $v \in V(Q(y))$.*

Proof. The claim follows directly from the fact that we have $T(X(y)) \subseteq T(X(x))$ with equality if and only if $|\delta_X^+(x)| = 1$. □

We will now partition $V(X)$ into subsets representing paths in $X$. This will allow us to solve independent subproblems on these paths:

Definition 7.23. Let $X$ be a buffer tree netlist graph for a net $N$. A *path partitioning* of $X$ is a decomposition of $X$ into vertex-disjoint subgraphs $X_1, \ldots, X_l$ such that $V(X) = \bigcup_{i=1}^{l} V(X_i)$ and each $X_i$ is a (directed) path in $X$ such that $\Big(x \in T \text{ or } |\delta_X^+(x)| \geq 2\Big) \iff \delta_{X_i}^+(x) = \emptyset$ holds for $x \in X_i$.

Clearly, $\delta_{X_i}^+(x) = \emptyset$ holds for exactly one vertex in $X_i$, as $X_i$ is a directed path. We note that it will naturally occur that some of the subgraphs in our path partitioning will only have one vertex. The path partitioning is unique and easy to compute:

Proposition 7.24. *Given a buffer tree netlist graph $X$ for a net $N$, the path partitioning of $X$ can be computed in $\mathcal{O}(|V(X)|)$ time.*

PROOF. Let $R = T \cup \{x \in V(X) : |\delta_X^+(x)| \geq 2\}$. For $x \in R$ we construct a path $X_x$ by traversing the $x$-$s$ path in $X$ until another vertex in $R$ (which is excluded from $X_x$) or $s$ (which is included in $X_x$ if and only if $s \notin R$) is reached. Performing this process for all $x \in R$ will yield the path partitioning. □

The path partitioning partitions $V(X) \setminus N$ into equivalence classes with regard to allowed paths:

LEMMA 7.25. *Consider an instance of the* SIMPLIFIED COPY ROUTES PROBLEM *and let* $X_1, \ldots, X_l$ *be the path partitioning of* $X$. *Let* $x \in V(X_i) \setminus N$ *and* $y \in V(X_j) \setminus N$. *Then we have* $\Big(i = j\Big) \iff \Big(T(X(x)) = T(Y(y))\Big) \iff \Big(Q(x) = Q(y)\Big)$.

PROOF. The equivalence $\Big(T(X(x)) = T(X(y))\Big) \iff \Big(Q(x) = Q(y)\Big)$ follows directly from Definition 7.20. If $i = j$, then $T(X(x)) = T(X(y))$ follows directly from the structure of the path partitioning. For the other direction let $T(X(x)) = T(X(y))$, and without loss of generality we can assume $y \in V(X(x))$. Then $|\delta_X^+(z)| = 1$ must hold for all $z \in V(P_X(x, y)) \setminus \{y\}$, as otherwise we would not have $T(X(x)) = T(X(y))$. This implies $i = j$ by definition of the path partitioning. □

We continue with another lemma. Its statement is illustrated by Figure 7.12:

LEMMA 7.26. *Consider an instance of the* SIMPLIFIED COPY ROUTES PROBLEM *and let* $X_1, \ldots, X_l$ *be the path partitioning of* $X$. *Let* $x \in V(X_i) \setminus N$ *and* $y \in V(X_j) \setminus N$, $i \neq j$, *and let* $q(x) \in V(Q(x))$ *and* $q(y) \in V(Q(y))$. *Then we have* $y \in V(X(x)) \Rightarrow q(y) \in V(Y(q(x)))$.

PROOF. We first note that we cannot have $T(X(x)) = T(X(y))$, as this would imply $i = j$ according to Lemma 7.25. Therefore we get

$$\begin{aligned} y \in V(X(x)) &\Rightarrow T(X(y)) \subsetneq T(X(x)) \\ &\Rightarrow T(Y(v_y)) \subsetneq T(Y(v_x)) && \forall\, (v_x, v_y) \in V(Q(x)) \times V(Q(y)) \\ &\Rightarrow v_y \in V(Y(v_x)) && \forall\, (v_x, v_y) \in V(Q(x)) \times V(Q(y)), \end{aligned}$$

which proves the claim. □

The above results imply that given the path partitioning $X_1, \ldots, X_l$ of $X$, one can solve the SIMPLIFIED COPY ROUTES PROBLEM independently on every path $X_i$, $i = 1, \ldots, l$, using the algorithm outlined in Theorem 7.19. Requirements (2) and (3) from Definition 7.12 are then automatically fulfilled due to Lemma 7.25 and 7.26. In addition to the path partitioning we need the allowed paths $Q(x)$, $x \in V(X) \setminus N$. We use Algorithm 6 to compute them:

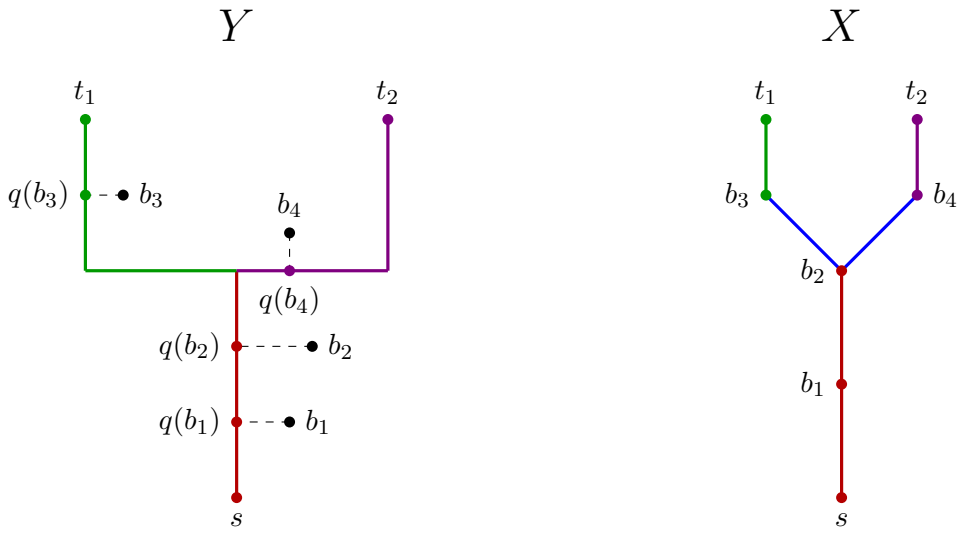



FIGURE 7.12. Example illustrating Lemma 7.26: We have a route $Y$ (left) and a corresponding buffer tree netlist graph $X$ (right). The path partitioning of $X$ encompasses three paths that are highlighted by the colors red, green and violet, and the corresponding allowed paths in $Y$ are also colored respectively (blue edges do not belong to any path in the path partitioning). Hence, each buffer $b_i$, $i = 1, \ldots, 4$, is mapped onto its allowed path.
This automatically results in $q(b_j) \in V(Y(q(b_i)))$ for $j \in \{3, 4\}$ and $i \in \{1, 2\}$, which is exactly what is claimed by Lemma 7.26. However, it does not automatically result in $q(b_2) \in V(Y(q(b_1)))$, as $b_1$ and $b_2$ belong to the same path of the path partitioning: If we reversed the mapped positions of $b_1$ and $b_2$, then all buffers would still be mapped onto their allowed paths, but $b_1$ and $b_2$ would not be mapped correctly according to Definition 7.12.
Thus, the statement of Lemma 7.26 only holds for pairs of buffers that belong to different paths in the path partitioning.

THEOREM 7.27. *Algorithm 6 computes $Q(x)$ for all $x \in V(X) \setminus N$ correctly.*

PROOF. We prove it by induction in reverse topological order of $X$: The statement can be assumed to be true for all $t \in T$, as the output only consists of $Q(x)$ for $x \in V(X) \setminus N$. Therefore, consider $(x, y) \in E(X)$ as chosen in line 5, and assume that $Q(y)$ has already been correctly computed. If $|\delta_X^+(x)| = 1$ and $y \notin T$, then Lemma 7.22 tells us that $Q(x)$ is set correctly in line 7.
Now consider the case $|\delta_X^+(x)| \geq 2$ or $y \in T$. Here we know that $Q(x)$ is a subpath of $P$ from line 9: If $|\delta_X^+(x)| \geq 2$, then this follows from Lemma 7.22, and for $y \in T$ this follows from the definition of $Q(t)$, $t \in T$, in line 3. Let $P$ be defined by the sequence

**Algorithm 6** Allowed Path Computation Algorithm

**Input:** An instance of the Simplified Copy Routes Problem.
**Output:** $Q(x), x \in V(X) \setminus N$.

1: Compute $|T(X(x))|$ for all $x \in V(X)$ and $|T(Y(v))|$ for all $v \in V(Y)$.
2: **for** $t \in T$ **do**
3: $V(Q(t)) := \{t\}$, $E(Q(t)) := \emptyset$, $q_1(t) := q_2(t) := t$.
4: **for** $x \in V(X) \setminus N$ in reverse topological order of $X$ **do**
5: Let $(x, y) \in E(X)$.
6: **if** $|\delta_X^+(x)| = 1$ **and** $y \notin T$ **then**
7: $Q(x) := Q(y)$.
8: **else**
9: Traverse the $q_1(y)$-$s$ path $P$ in $Y$ until $q_2(x)$ and $q_1(x)$ are found, which are the first and last vertices in $P$ with $|T(Y(q_i(x)))| = |T(X(x))|$, $i = 1, 2$.
10: Set $Q(x)$ to be the $q_1(x)$-$q_2(x)$ path in $Y$.

$q_1(y) = v_1, v_2, \ldots, v_l = s$. Then we have $T(Y(v_i)) \subseteq T(Y(v_j))$ for $i, j \in \{1, \ldots, l\}$ with $i \leq j$, and so $\Big(|T(Y(v))| = |T(X(x))|\Big) \iff \Big(T(Y(v)) = T(X(x))\Big)$ holds for $v \in V(P)$. It follows that $q_2(x)$ and $q_1(x)$ are set correctly in line 10. □

Theorem 7.28. *Algorithm 6 can be implemented in $\mathcal{O}(n+k)$ time with $n := |V(Y)|$ and $k := |V(X)|$.*

Proof. We show that every edge in $E(Y)$ is traversed at most once as part of the $q_1(x)$-$q_1(y)$ path in line 9 during the course of the whole algorithm. The result then follows, as the rest of the algorithm can obviously be implemented in $\mathcal{O}(n + k)$ time. Consider $e = (v, w) \in E(Y)$ that is traversed for the first time as part of the $q_1(x)$-$q_1(y)$ path in line 9 when $(x, y) \in E(X)$ is considered in line 5, and assume that $e$ is traversed a second time as part of the $q_1(x')$-$q_1(y')$ path in line 9 for $(x', y') \neq (x, y)$. Then we know

$$T(X(\bar{y})) = T(Y(q_1(\bar{y}))) \subseteq T(Y(v)) \subseteq T(Y(q_1(\bar{x}))) = T(X(\bar{x})) \tag{7.2}$$

for $(\bar{x}, \bar{y}) \in \{(x, y), (x', y')\}$. In particular this gives us $T(X(y')) \subseteq T(X(x))$, and as we cannot have $y' \in V(X(x)) \setminus \{x\}$ since $x$ was traversed before $x'$ in line 4, $y' \in V(P_X(s, x))$ and $T(X(x)) = T(X(y'))$ follows. This implies $Q(x) = Q(y')$ due to Lemma 7.25, and so $q_1(x) = q_1(y')$. But then we have $(v, w) \in E(P_Y(q_1(x), q_1(y)))$ and $(v, w) \in E(P_Y(q_1(x'), q_1(x)))$ by our assumption, which is a contradiction due to $E(P_Y(q_1(x), q_1(y))) \cap E(P_Y(q_1(x'), q_1(x))) = \emptyset$. □

We combine our results to derive an algorithm for the Simplified Copy Routes Problem:

THEOREM 7.29. *Consider an instance of the* SIMPLIFIED COPY ROUTES PROBLEM *with a path partitioning where $X_1, \dots, X_l$ are the paths with $V(X_i) \setminus N \neq \emptyset$, $i = 1, \dots, l$. For $i = 1, \dots, l$ let $Q_i := Q(x_i)$ for some $x_i \in V(X_i) \setminus N$, $n_i := |V(Q_i)|$ and $k_i := |V(X_i)|$. Then we can solve the given instance of the* SIMPLIFIED COPY ROUTES PROBLEM *in a total of $\mathcal{O}(n + k + \sum_{i=1}^{l} n_i k_i^2) = \mathcal{O}(nk^2)$ time.*

PROOF. We can compute the path partitioning of $X$ and all allowed paths $Q(x), x \in V(X) \setminus N$, in $\mathcal{O}(n+k)$ time, as stated in Proposition 7.24 and Theorems 7.27 and 7.28. As every path $X_i$, $i = 1, \dots, l$, defines an independent instance of the SIMPLIFIED COPY ROUTES PROBLEM, we can run the algorithm outlined in Theorem 7.19 on each $Q_i$ and $X_i$ individually and achieve the desired running time. Combining the buffer mappings on these individual instances yields a buffer mapping fulfilling the requirements from Definition 7.12 due to the definition of the allowed paths and Lemma 7.26. Moreover, Lemma 7.21 tells us that only considering $Q_i$ for mapping $V(X_i)$ is sufficient, and we indeed arrive at an optimum solution. □

7.4.3.3. *A Greedy Heuristic.* Theorem 7.29 gives a recipe for solving the SIMPLIFIED COPY ROUTES PROBLEM, but the running time can theoretically be large if the individual $X_i$ define large instances with many buffers. In practice, this is usually not the case, in particular not at the stage in the design flow where our tool is used. Still, for running time reasons and simplicity we use a different approach in our practical implementation: We proceed as in Theorem 7.29, but instead of solving the arising subproblems on the paths $X_i$, $i = 1, \dots, l$, optimally as in Theorem 7.19, we use the heuristic depicted as Algorithm 7. We use some terminology that is explained in the following definition, which uses line segments as in Definition 2.13:

DEFINITION 7.30. Let $Y$ be a Steiner tree that is embedded rectilinearly into the layered chip area $S$. Then given $e \in E(Y)$ and $a \in S$ we set $\operatorname{dist}(e, a) := \min_{b \in L(e)} \operatorname{dist}(a, b)$. Moreover, *subdividing $e$ at the closest point $v$ to $a$* means subdividing $e$ as in Definition 2.14 by inserting a new vertex $v$ with $\operatorname{dist}(v, a) = \operatorname{dist}(e, a)$.

**Algorithm 7** Copy Routes on Paths Greedy Algorithm

**Input:** An instance of the SIMPLIFIED COPY ROUTES PROBLEM where $X$ is a path.
**Output:** A subdivision $Y'$ of $Y$ and a buffer mapping $q\colon V(X) \to V(Y')$.

1: $Y' := Y$.
2: Set $q(\pi) := \pi$ for $\pi \in N$.
3: **for** $x \in V(X) \setminus N$ in reverse topological order of $X$ **do**
4: Let $\Gamma_X^+(x) = \{y\}$ and $e^* := \arg\min_{e \in E(P_{Y'}(s, q(y)))} \operatorname{dist}(e, p(x))$.
5: Subdivide $e^*$ at the closest point $v$ to $p(x)$ and set $q(x) := v$.

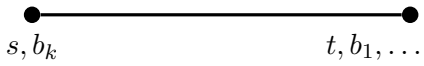



FIGURE 7.13. Illustration of the instance of the SIMPLIFIED COPY ROUTES PROBLEM used in the proof of Proposition 7.31.

The reasoning behind Algorithm 7 is that most of the time when a path is buffered, the buffers are just chained up along that path with only local deviations occuring. However, it is easy to prove that Algorithm 7 does not achieve any satisfactory theoretical approximation guarantee:

PROPOSITION 7.31. *For any $k \in \mathbb{N}$ there exists an instance of the* SIMPLIFIED COPY ROUTES PROBLEM *where $X$ is a path and $|V(X) \setminus N| = k$, such that Algorithm 7 returns a buffer mapping $q$ with $\mathrm{dist}(X, q) \geq (k-1)\mathrm{OPT}$, where $\mathrm{OPT}$ is the optimum objective function value.*

PROOF. Consider the instance of the SIMPLIFIED COPY ROUTES PROBLEM illustrated in Figure 7.13. $X$ is a path defined by the sequence $s, b_1, \ldots, b_k, t$ and $Y$ consists of a straight wire from $s$ to $t$. Algorithm 7 will start by setting $q(b_k) = s$ and consequently must set $q(b_i) = s$ for $i = 1, \ldots, k-1$ (or achieve the same by creating zero length edges). However, an optimum buffer mapping $q^*$ sets $q^*(b_i) = t$ for $i = 1, \ldots, k$. We then have $\mathrm{dist}(X, q) = (k-1)\mathrm{dist}(s, t)$ and $\mathrm{dist}(X, q^*) = \mathrm{dist}(s, t)$, proving the claim. □

However, the running time is faster than the one outlined in Theorem 7.19:

PROPOSITION 7.32. *Algorithm 7 can be implemented in $\mathcal{O}(nk)$ time.* □

Using Algorithm 7 as a subroutine for the algorithm outlined in Theorem 7.29 yields a total running time of $\mathcal{O}(n + k + \sum_{i=1}^{l} n_i k_i) = \mathcal{O}(nk)$ in the notation of the theorem. In practice it rarely happens that a single path in the route has very many vertices, and the number of buffers inserted at this stage in the design flow is rather small. Therefore, Algorithm 7 is sufficiently fast for our purposes. This claim is supported by our experimental results from Section 7.4.4. Moreover, Section 7.4.4 shows that Algorithm 7 also produces good results in practice.

**7.4.4. Experimental Results.** We implemented Algorithm 7 from Section 7.4.3.3 and evaluated the results. The purpose of our experiment is to show that the SIMPLIFIED COPY ROUTES PROBLEM is a good approximation of the COPY ROUTES PROBLEM in practice, and that Algorithm 7 provides good approximate solutions for it. As achieving a high gain in the sense of the objective function of the COPY ROUTES PROBLEM is more important in DRBO than in GRBO, we run our experiment in a setting that

resembles DRBO. More precisely, we start with fully detailed routed designs and only run buffering.[1] For our experiment we disabled the post-optimization routine that cuts away detailed wires in order to prevent minimum distance rule violations (cf. Section 7.4.2), as it would artificially reduce the performance of Algorithm 7 with respect to the objective function of the Copy Routes Problem. The experiment is conducted on an Intel Xeon E5-2667 v2 server running at 3.30 GHz using 16 threads. More information on our setup and testbed is provided in Appendix A.

Table 7.4 displays various metrics, whose meanings are as follows: We let $\mathcal{I}$ denote the set of instances of the Copy Routes Problem that are processed in our run. In this experiment we process all instances that are handed to us by the buffering tool except the ones where the wire length of the routing tree of the root net is less than three times the average edge length of the global routing graph. The reason for sorting out these instances is that on such small instances local issues become the dominating factor in determining the quality of results in the sense of our metrics, although they are not very important for practical performance.

We then display $|\mathcal{I}|$ in the column labeled "# Inst", while the column labeled "# Nets" contains the number of nets before buffering on the given unit. For each $I \in \mathcal{I}$ we let $Y(I)$, $X(I)$ and $f(I)$ denote the routing tree, buffer tree netlist graph, and wire distribution computed by application of Algorithm 7 for $I$, respectively. To evaluate the quality of $f(I)$, we compare $\text{gain}(f(I))$ against an approximate upper bound $\text{gain}_{\text{ub}}(I) := \text{smt}(\mathcal{N}_{X(I)}) - \text{smt}_{\text{lb}}(I)$, where $\text{smt}_{\text{lb}}(I)$ is an approximate lower bound for $\min\{\text{smt}(\mathcal{N}_{X(I)}, f) : f \text{ is wire distribution for } I\}$. For computing $\text{smt}_{\text{lb}}(I)$ we compute two different approximate lower bounds $\text{smt}^1_{\text{lb}}(I)$ and $\text{smt}^2_{\text{lb}}(I)$ and set $\text{smt}_{\text{lb}}(I) := \max\{\text{smt}^1_{\text{lb}}(I), \text{smt}^2_{\text{lb}}(I)\}$. These two lower bounds are defined as follows: For defining $\text{smt}^1_{\text{lb}}(I)$ let $f_{N_x}$ for $N_x \in \mathcal{N}_{X(I)}$ denote the wire distribution that assigns all wires in $Y(I)$ to $N_x$, and we set $\text{smt}^1_{\text{lb}}(I) := \sum_{N_x \in \mathcal{N}_{X(I)}} \text{smt}(N_x, f_{N_x})$. For defining $\text{smt}^2_{\text{lb}}(I)$ we note that $\text{smt}(N_x) \leq \text{smt}(N_x, f) + \sum_{(v,w) \in f^{-1}(N_x)} \text{dist}(v, w)$ holds for any wire distribution $f$ for $I$ and any $N_x \in \mathcal{N}_{X(I)}$, and so $\text{smt}^2_{\text{lb}}(I) := \text{smt}(\mathcal{N}_{X(I)}) - \sum_{(v,w) \in E(Y(I))} \text{dist}(v, w) \leq \text{smt}(\mathcal{N}_{X(I)}, f)$ follows. Here, all Steiner trees for determining these bounds are computed by solving the Rectilinear Minimum Steiner Tree Problem and Rectilinear Minimum Steiner Tree with Prewires Problem as outlined in Sections 2.3.2 and 2.3.3 (depending on whether prewires are present or not).

We then set $r_{\text{gain}}(I) := \text{gain}(I) \,/\, \text{gain}_{\text{ub}}(I)$ for $I \in \mathcal{I}$ and denote $\min_{I \in \mathcal{I}} r_{\text{gain}}(I)$ as "Min. $r_{\text{gain}}$" and $|\mathcal{I}|^{-1} \sum_{I \in \mathcal{I}} r_{\text{gain}}(I)$ as "Avg. $r_{\text{gain}}$" in Table 7.4. Moreover, the column labeled "Avg. $r_{\text{smt}}$" shows $|\mathcal{I}|^{-1} \sum_{I \in \mathcal{I}} r_{\text{smt}}(I)$, where $r_{\text{smt}}(I) := \text{smt}(\mathcal{N}_{X(I)}, f(I)) \,/\, \text{smt}(\mathcal{N}_{X(I)})$.

[1] Buffering is actually used in multiple optimization steps during DRBO, e.g. also for correcting electrical violations. We use the buffer insertion procedure for improving (late mode) timing for our experiment.

| Unit | #Nets | #Inst | Min. $r_{\text{gain}}$ [%] | Avg. $r_{\text{gain}}$ [%] | Avg. $r_{\text{smt}}$ [%] | RT [sec.] |
|---|---|---|---|---|---|---|
| U1 | 77 528 | 119 | 80.97 | 97.83 | 8.65 | 0.04 |
| U2 | 79 119 | 80 | 88.67 | 98.70 | 5.36 | 0.04 |
| U3 | 100 827 | 131 | 85.06 | 98.39 | 8.39 | 0.04 |
| U4 | 111 140 | 205 | 80.62 | 98.58 | 7.19 | 0.07 |
| U5 | 119 228 | 224 | 77.83 | 98.30 | 7.11 | 0.09 |
| U6 | 254 208 | 699 | 80.98 | 98.01 | 10.40 | 0.18 |
| U7 | 276 799 | 299 | 78.10 | 99.03 | 7.07 | 0.21 |
| U8 | 1 681 671 | 718 | 78.35 | 99.79 | 1.63 | 0.22 |

TABLE 7.4. Experimental results obtained by running Algorithm 7.

Note that these metrics are listed in percent. Finally, the last column shows the accumulated running time of Algorithm 7 on all $I \in \mathcal{I}$.

Table 7.4 demonstrates a strong performance of Algorithm 7 in practice: On average, the gain of our computed solution is only very few percent away from the upper bound, and even in the worst cases the gain is still tolerable.

The column labeled "Avg. $r_{\text{smt}}$" shows that on average only a relatively small amount of wire length needs to be added by the global router to complete the routing of the target nets. This confirms a claim that we made in Section 7.4.2 when defining the COPY ROUTES PROBLEM, which led us to defining the objective function of this problem as maximizing the gain instead of minimizing the required additional wire length. Lastly, the running time column shows that our implementation of Algorithm 7 runs very fast: We did not have to solve a great number of instances in each run, which is in the nature of post-routing optimization, but extrapolating the numbers also shows that a larger number of instances could be solved within seconds.

All in all, our data shows that Algorithm 7 achieves good results while being very fast. This in turn validates our approach of first approximating the COPY ROUTES PROBLEM by the SIMPLIFIED COPY ROUTES PROBLEM and then solving the SIMPLIFIED COPY ROUTES PROBLEM with the use of Algorithm 7. To further improve the results at the cost of a larger running time, one could switch to the algorithm outlined in Theorem 7.29 whenever necessary. However, in consideration of the good results outlined in Table 7.4, we refrain from this additional effort and complexity for the time being.

## 7.5. Multi-Threaded Incremental Routing

We also provide a multi-threaded implementation of our incremental router, which is described in this section. An introduction to multi-threading in C++ is given by Williams [122]. We first describe the basic mechanics of our multi-threaded incremental routing framework in Section 7.5.1, then turn our attention to the problem of updating usages during multi-threaded incremental routing in Section 7.5.2, and present experimental results in Section 7.5.3.

**7.5.1. The Multi-Threaded Incremental Routing Framework.** When running multi-threaded we use the same transaction framework as described in Section 7.2, but extend it to a multi-threaded scenario. Here, every thread starts and performs its own transaction, which is largely independent of the transactions of other threads. This means that when the incremental router receives a callback for a data model change, it first checks which thread made the change, and records the change in the transaction log of this thread. In the same manner, when one of the incremental routing control functions to start a transaction, implement changes, commit changes, or undo changes (cf. Figure 7.2), is called, then this function call is ascribed to the calling thread, and only affects the transaction of this particular thread.
To make this process work, some concurrency issues have to be addressed. To avoid synchronization problems for data model changes in different threads, the netlist is divided into *netlist sections*. For our purposes, a netlist section is a set of nets, and every thread only works on one netlist section at a time. Moreover, the following conditions hold:

- If $\tau_1$ and $\tau_2$ are threads working on netlist sections $S_1$ and $S_2$, respectively, then $S_1 \cap S_2 = \emptyset$.
- If a thread $\tau$ is working on a netlist section $S$ and makes a data model change, then this change must not affect any nets that are not in $S$.

This way it is ensured that different threads never make simultaneous changes that affect the same net. Here, the division of the netlist into sections is not fixed, but may be changed perpetually. It is also possible (and maybe more natural) to define netlist sections in terms of circuits, but for our purposes, the definition in terms of nets is sufficient. The concept behind netlist sections is illustrated by Figure 7.14.
Of course there are also intricacies in dividing the netlist into such sections, in particular changes in one section might very well have effects on timing and slew outside of the section. In this thesis, we do not elaborate on this any further, as the multi-threaded incremental routing framework including code to compute and manage netlist sections is provided by IBM, and we only integrate our incremental router into this framework.

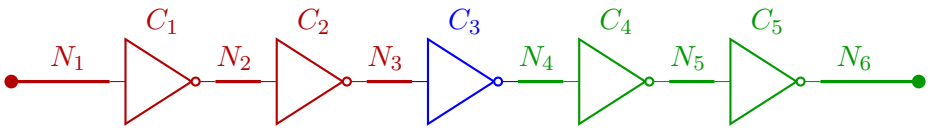



FIGURE 7.14. A very simple netlist used to illustrate the concept behind netlist sections from Section 7.5.1: The netlist is divided into two sections $S_1 = \{N_1, N_2, N_3\}$ (red) and $S_2 = \{N_4, N_5, N_6\}$ (green). A thread working on $S_1$ might modify (e.g. move or resize) $C_i$, $i \in \{1, 2\}$, at the same time where a thread working on $S_2$ modifies $C_j$, $j \in \{4, 5\}$. $C_3$ cannot be modified by any thread, as modifying $C_3$ would affect $N_3 \in S_1$ and $N_4 \in S_2$. In order to modify $C_3$, one would need to repartition the netlist into different sections.

Despite the separation of the netlist into sections, there are concurrency issues to be addressed. Firstly, our router must support making incremental changes in most of its core data structures in a multi-threaded context in an efficient way. Secondly, although netlist sections are disjoint with regard to the nets they contain, they are not disjoint in a geometric sense. This can be problematic with regard to congestion if multiple threads are concurrently routing nets in the same area. We address this issue in Section 7.5.2, where we elaborate on different methods to update edge usages during multi-threaded incremental routing with the goal of preventing routing overflow caused by concurrency.

**7.5.2. Updating Usages During Multi-Threaded Incremental Routing.** As outlined above, the division into netlist sections ensures that different threads never work simultaneously on the same net, but it is not ensured that different threads are not routing nets in the same area. Therefore, threads may be competing for edge resources. For this reason we develop a method for updating edge usages with the goal of minimizing congestion problems caused by concurrency.

In the following we will use the notion of *threads* in a theoretical context. For our purposes, a thread is a sequence of elementary machine operations. We assume a total order of all machine operations of all threads, but do not impose any particular requirements on the exact order of operations of different threads.[2] This is of course neither a formal definition nor does it represent the behavior of modern machines exactly, but it should be sufficient for our purposes.

In this section we only consider edge resources, although the results of this section can be applied to any resource type. Here, the notion of edge resources and usages is adopted from Chapters 2 and 4. Throughout this section, $G$ will always denote the global routing

[2]In practice, this is not exactly true, as we use synchronization methods like mutexes in order to ensure an orderly execution of the code. However, we neglect this for the moment in our model.

graph and $\mathcal{T}$ the set of threads. Moreover, we denote the currently open transaction of any thread $\tau \in \mathcal{T}$ by $\phi(\tau)$. For convenience of notation we always assume that every thread has an open transaction, but the set of changes in this transaction may be empty. When referring to a (possibly open) transaction, we refer to all data model changes in this transaction and the routing changes required to implement these changes (cf. Section 7.2).

The rest of this section is structured as follows: In Section 7.5.2.1 we introduce the concepts of global and thread-local usages. This allows us to define three different usage update schemes in Section 7.5.2.2. A theoretical analysis of these update schemes is given in Section 7.5.2.3, and we evaluate them in practice later in Section 7.5.3.2.

7.5.2.1. *Global and Thread-Local Usages.* Let $e \in E(G)$ and $\tau \in \mathcal{T}$. Then we let $\Delta\text{usg}_e(\phi(\tau))$ be the *usage change of $e$ caused by all operations in $\phi(\tau)$*. In our routing space usage model, $\Delta\text{usg}_e(\phi(\tau))$ only depends on $\phi(\tau)$, but not on the initial state when the transaction was started or on concurrent transactions in other threads. We partition $\Delta\text{usg}_e(\phi(\tau))$ into

$$\Delta\text{usg}_e(\phi(\tau)) = \Delta\text{usg}_e^{\text{g}}(\phi(\tau)) + \Delta\text{usg}_e^{\text{t}}(\phi(\tau)), \tag{7.3}$$

where $\Delta\text{usg}_e^{\text{g}}(\phi(\tau))$ is the *global usage difference of $\phi(\tau)$* that is seen by all threads, while $\Delta\text{usg}_e^{\text{t}}(\phi(\tau))$ is the *thread-local usage difference of $\phi(\tau)$* that is only seen by $\tau$. Different methods to do this partitioning are presented in Section 7.5.2.2.

Moreover, let $\text{usg}_e$ denote the *base usage of $e$*, which is the usage of $e$ without considering usage changes in any open transactions. Then a thread $\tau$ sees a *thread-local usage* of $\text{usg}_e^\tau$ of $e$ at any given time, which is defined as

$$\text{usg}_e^\tau := \text{usg}_e + \Delta\text{usg}_e^{\text{t}}(\phi(\tau)) + \sum_{\tau' \in \mathcal{T}} \Delta\text{usg}_e^{\text{g}}(\phi(\tau')).$$

Here, with "see" we mean that whenever the usage of $e$ is queried from $\tau$ (e.g. when computing edge prices as in Section 7.3.3), $\text{usg}_e^\tau$ is returned.

7.5.2.2. *Usage Update Schemes.* We examine three different *usage update schemes*, i.e. methods to do the partitioning in (7.3):

*Fully global update scheme:*

- $\Delta\text{usg}_e(\phi(\tau)) = \Delta\text{usg}_e^{\text{g}}(\phi(\tau))$,

*Fully thread-local update scheme:*

- $\Delta\text{usg}_e(\phi(\tau)) = \Delta\text{usg}_e^{\text{t}}(\phi(\tau))$,

*Semi thread-local update scheme:*

- $\Delta\text{usg}_e(\phi(\tau)) \geq 0 \Rightarrow \Delta\text{usg}_e(\phi(\tau)) = \Delta\text{usg}_e^{\text{g}}(\phi(\tau))$,
- $\Delta\text{usg}_e(\phi(\tau)) < 0 \Rightarrow \Delta\text{usg}_e(\phi(\tau)) = \Delta\text{usg}_e^{\text{t}}(\phi(\tau))$.

In our implementation we use the semi thread-local update scheme, which is the most conservative of the three.

7.5.2.3. *Analyzing the Usage Update Schemes.* In this section we state propositions that point out conditions under which certain update schemes are guaranteed to not cause routing overflow, and give examples that illustrate what can go wrong with other update schemes. We start with the following proposition:

PROPOSITION 7.33. *Assume that every thread $\tau$ queries routing congestion at the end of its transaction $\phi(\tau)$, and undoes the transaction if there is an edge $e \in E(G)$ with $\Delta\text{usg}_e(\phi(\tau)) > 0$ and $\text{usg}_e^\tau > 1$. Moreover, assume that when one thread $\tau$ has started querying congestion at the end of its transaction, then no other thread can do the same until $\tau$ has completely committed or undone its transaction. Then routing overflow does not increase during incremental routing for any edge $e \in E(G)$ if the fully thread-local or the semi thread-local update scheme is used.*

PROOF. Assume the contrary and let $e \in E(G)$ be an edge where overflow increases during incremental routing. Consider the first point in time when this happens. Then there must be a thread $\tau$ committing a transaction $\phi(\tau)$ such that $\Delta\text{usg}_e(\phi(\tau)) > 0$ and $\text{usg}_e + \Delta\text{usg}_e(\phi(\tau)) > 1$ when the commit is being done. We know that $\text{usg}_e$ cannot change between the time when $\tau$ commits its transaction and the time when $\tau$ queries the usage of $e$, as no other thread is allowed to commit a transaction while $\tau$ is querying congestion. Therefore, when $\tau$ queries the usage of $e$, it sees a usage of

$$\begin{aligned}
\text{usg}_e^\tau &= \text{usg}_e + \Delta\text{usg}_e^{\text{t}}(\phi(\tau)) + \sum_{\tau' \in \mathcal{T}} \Delta\text{usg}_e^{\text{g}}(\phi(\tau')) \\
&\geq \text{usg}_e + \Delta\text{usg}_e^{\text{t}}(\phi(\tau)) + \Delta\text{usg}_e^{\text{g}}(\phi(\tau)) \\
&= \text{usg}_e + \Delta\text{usg}_e(\phi(\tau)) \\
&> 1.
\end{aligned}$$

This is a contradiction to our assumptions, as $\tau$ would have to undo its transaction. □

The assumptions from Proposition 7.33 are reasonable in practice, but not all operating tools query congestion at the end of their transaction. However, as the router offers this functionality, it can be incorporated if deemed necessary. Moreover, the congestion query can be serialized by a mutex to ensure proper synchronization.
Next we want to give an example that shows that Proposition 7.33 does not hold when the fully global update scheme is used:

EXAMPLE 7.34. Assume we are using the fully global update scheme. Consider the instance depicted in Figure 7.15: There are two nets $N_i = \{s_i, t_i\}$, $i = 1, 2$, with original pin positions $p\colon N_1 \cup N_2 \to V(G)$, and two threads $\tau_i$, $i = 1, 2$, each working on $N_i$,

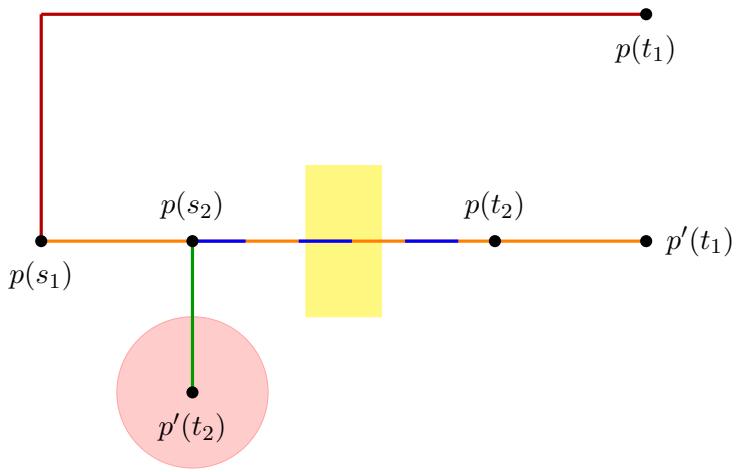



FIGURE 7.15. Illustration of Example 7.34.

respectively. The initial routes for $N_1$ and $N_2$ are depicted in red and blue, respectively. We assume that $N_1$ and $N_2$ use the same amount of routing space on every edge in the global routing graph. The light red area around $p'(t_2)$ is overcongested. The light yellow area between $p(s_2)$ and $p(t_2)$ is at exactly 100% congestion when either $N_1$ or $N_2$ are routed through it, but overcongested when both are. Consider the following sequence of operations:

(1) $\tau_2$ moves $t_2$ to $p'(t_2)$ and replaces the blue by the green route.
(2) $\tau_1$ moves $t_1$ to $p'(t_1)$ and replaces the red by the orange route.
(3) $\tau_1$ queries congestion and commits its transaction.
(4) $\tau_2$ undoes its transaction (e.g. because the green route creates additional overflow).

$N_1$ is now routed by the orange and $N_2$ by the blue route, and routing overflow is created in the light yellow area. This constellation is possible because the blue route is subtracted from the congestion map when $\tau_1$ implements and commits its transaction in (2) and (3).

This example is not possible with the fully or the semi thread-local update scheme, as in that case the blue route would not be subtracted from the congestion map during (2) and (3). This would give $\tau_1$ a chance to work around the congestion or undo its transaction.

The next proposition presents a way to prevent overflow increase during multi-threaded incremental routing without relying on the operating tool querying congestion. It only works with the global or the semi thread-local update scheme:

PROPOSITION 7.35. *Assume that when routing a net $N$ in a thread $\tau$ during incremental routing, $\tau$ proceeds as follows: After computing a route $Y$ for $N$, $\tau$ updates $\Delta\mathrm{usg}_e(\phi(\tau))$ for all edges $e \in E(G)$ with $\mathrm{usg}_{N,e}(Y) > 0$. In particular, $\tau$ makes any update of $\Delta\mathrm{usg}^{\mathrm{g}}_e(\phi(\tau))$ visible to all other threads. After each such update, $\tau$ checks whether $\mathrm{usg}^{\tau}_e > 1$, and if so, then $\tau$ rejects $Y$ and reroutes $N$. Assume that updating $\Delta\mathrm{usg}_e(\phi(\tau))$ and checking $\mathrm{usg}^{\tau}_e$ for a given edge $e \in E(G)$ happens in one single operation. Moreover, assume that this net routing process always terminates, i.e. it never goes on indefinitely. Then overflow will not increase during incremental routing for any edge $e \in E(G)$ if the global or the semi thread-local update scheme is used.*

PROOF. Assume the contrary and let $e \in E(G)$ be an edge where overflow increases during incremental routing. Let $x_c$ be the first point in time when this happens, i.e. when a thread $\tau^*$ commits a transaction $\phi(\tau^*)$ such that $\Delta\mathrm{usg}_e(\phi(\tau^*)) > 0$ and $\mathrm{usg}_e > 1$ directly after the commit. Let $U := \mathrm{usg}_e$ at $x_c$. Consider the last point in time $x_r$ before $x_c$ immediately after a thread $\tau$ increases $\Delta\mathrm{usg}^{\mathrm{g}}_e(\phi(\tau))$ without rejecting the corresponding route afterwards (possibly $\tau = \tau^*$).
For the rest of this proof, we let all notations except $U$ refer to the situation at time $x_r$ ($U$ will still denote $\mathrm{usg}_e$ at $x_c$). We note that

$$U \leq \mathrm{usg}_e + \sum_{\tau' \in \mathcal{T}} \Delta\mathrm{usg}^{\mathrm{g}}_e(\phi(\tau')),$$

which is due to the choice of $x_r$ and the choice of our update scheme: By the choice of $x_r$, no thread $\tau'$ can increase $\Delta\mathrm{usg}^{\mathrm{g}}_e(\phi(\tau'))$ between $x_r$ and $x_c$. Therefore, the amount of usage committed between $x_r$ and $x_c$ can be at most $\sum_{\tau' \in \mathcal{T}} \Delta\mathrm{usg}^{\mathrm{g}}_e(\phi(\tau'))$, as thread-local usages are never positive with the global or the semi thread-local update scheme. Moreover, note that $\Delta\mathrm{usg}^{\mathrm{t}}_e(\phi(\tau)) = 0$ also holds with the semi thread-local update scheme due to $\Delta\mathrm{usg}^{\mathrm{g}}_e(\phi(\tau)) > 0$ by our choice of $x_r$.
Therefore, when $\tau$ checks the usage of $e$ at $x_r$, it sees a usage of

$$\mathrm{usg}^{\tau}_e = \mathrm{usg}_e + \Delta\mathrm{usg}^{\mathrm{t}}_e(\phi(\tau)) + \sum_{\tau' \in \mathcal{T}} \Delta\mathrm{usg}^{\mathrm{g}}_e(\phi(\tau')) = \mathrm{usg}_e + \sum_{\tau' \in \mathcal{T}} \Delta\mathrm{usg}^{\mathrm{g}}_e(\phi(\tau')) \geq U > 1.$$

This is a contradiction to our assumptions, as $\tau$ would need to reject its route. □

The assumption that the net routing process from Proposition 7.35 always terminates is hypothetical, as it is generally not given in practice:

(1) It might happen that a thread cannot find a route for a net without using overcongested edges even without interference of other threads.
(2) It might happen that a thread $\tau$ needs to reject its route $Y$ multiple times because another thread $\tau'$ increases $\mathrm{usg}_e$ or $\Delta\mathrm{usg}^{\mathrm{g}}_e(\phi(\tau'))$ between the computation of $Y$ and the corresponding usage update.

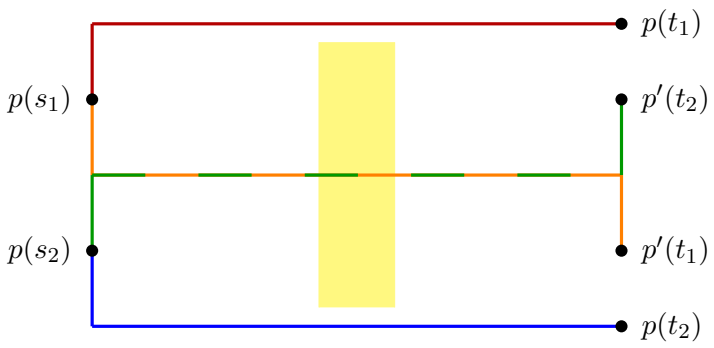



FIGURE 7.16. Illustration of Example 7.36.

(1) certainly happens in practice, but it is rather not a multi-threading problem, but a general one. (2) might also happen, but it seems unlikely that $\tau$ does not find a route after several tries, unless (1) occurs at some point.
Therefore, as a recipe derived from Proposition 7.35, one may try a constant number of reroutes if (2) occurs to reduce the amount of overflow created due to multi-threading.
It is easy to construct an example that shows that Proposition 7.35 does not hold for the fully thread-local update scheme. The basic idea is that multiple times threads may use the same edge because usage changes are kept thread-local until they are committed:

EXAMPLE 7.36. Assume we are using the fully thread-local update scheme. Consider the instance depicted in Figure 7.16: Let $N_i = \{s_i, t_i\}$, $i = 1, 2$, be two nets with original pin positions $p\colon N_1 \cup N_2 \to V(G)$. The initial routes for $N_1$ and $N_2$ are shown in red and blue, respectively. Assume that routing resources in the light yellow area are almost completely used. More precisely, they are used to an extent where either $N_1$ or $N_2$ may use an edge without creating routing overflow, but not both. Let thread $\tau_i$ be working on $N_i$, $i = 1, 2$, and consider the following sequence of operations:

(1) $\tau_1$ moves $t_1$ to $p'(t_1)$ and replaces the red by the orange route.
(2) $\tau_2$ moves $t_2$ to $p'(t_2)$ and replaces the blue by the green route.
(3) $\tau_1$ commits its transaction.
(4) $\tau_2$ commits its transaction.

This sequence of operations will create overflow even when the requirements of Proposition 7.35 are fulfilled: When $\tau_2$ checks usages in (2), it cannot see the added usage of the orange route, as this usage is kept thread-local by $\tau_1$. Consequently, $\tau_2$ does not reject its route, which results in overflow later.

Example 7.36 cannot occur with the global or the semi thread-local update scheme, as with these update schemes $\tau_1$ would increase $\Delta\mathrm{usg}_e^{\mathrm{g}}(\phi(\tau_1))$ for any newly used edge

$e \in E(G)$ in (1). This would give $\tau_2$ a chance to work around the congestion or reject its route in (2).
In our implementation we neither use the techniques from Proposition 7.33 nor the ones from Proposition 7.35. The reason is that our experimental results from Section 7.5.3 suggest that when the semi thread-local update scheme is used, routing congestion due to multi-threading is not an issue in our current application. Therefore, we keep our implementation fast and simple. However, should the need arise, Propositions 7.33 and 7.35 demonstrate ways to increase the protection against routing overflow due to multi-threading during incremental routing.

**7.5.3. Experimental Results.** In this section we analyze the practical performance of our multi-threaded incremental routing implementation. We do this by examining two different aspects of it separately: In Section 7.5.3.1 we analyze the general performance of our multi-threading framework by comparing results and running times with different numbers of threads. Here, we use the semi thread-local update scheme from Section 7.5.2 as our default choice for updating usages. An evaluation of usage update schemes is then given in Section 7.5.3.2, where we compare the three usage update schemes from Section 7.5.2 using 64 threads.
We only run GRBO for our experiments, as there are no fundamental differences between GRBO and DRBO with respect to multi-threading, and optimization is done on a larger scale in our GRBO flow than in our DRBO flow. The machine used for all experiments from this section runs two AMD EPYC 7601 32-core processors at 2.2GHz. Our runs with 64 threads can therefore be considered to run concurrently on the 64 available cores. As usual, an explanation of our setup and general metrics displayed in subsequent tables is given in Appendix A.

7.5.3.1. *Multi-Threading Performance.* We begin with analyzing the general performance of our multi-threaded implementation of Incremental BonnRouteGlobal. To do this, we run GRBO with multiple different thread counts and track general metrics like timing and running time. These results are given in Table 7.5. As a starting point we use the output of RC-Aware BonnRouteGlobal (cf. Chapters 4 and 5) on the given designs, and the "Start" row displays the respective metrics obtained in the single-threaded run. In addition to general metrics we list the total number of transactions (accumulated over all threads) in the column labeled "# TA". As one can see, the number of transactions can vary between different runs, resulting in some variations in the results. On all units except U8, these variations can be considered small. On the biggest unit U8, however, one can see that significantly more transactions are performed when more threads are used, which leads to better timing results but also smaller running time reductions.

| Unit (# nets) | After... | WS [ps] | FOM [ps] | PWR [mW] | wACE4 [%] | OFtgt [100 pitch$^2$] | # TA [k] | RT [h:mm:ss] |
|---|---|---|---|---|---|---|---|---|
| U3 | Start | -149 | -118244 | 70.71 | 86.3 | 0.0 | — | — |
| (100 827) | 1 thread | -75 | -85724 | 71.00 | 86.8 | 0.0 | 497 | 0:34:18 |
| | 8 threads | -74 | -84577 | 70.98 | 86.8 | 0.0 | 517 | 0:09:16 |
| | 16 threads | -74 | -85636 | 71.03 | 86.8 | 0.0 | 507 | 0:07:04 |
| | 64 threads | -74 | -86241 | 71.27 | 86.8 | 0.0 | 493 | 0:06:08 |
| U4 | Start | -195 | -152110 | 37.05 | 89.3 | 6.6 | — | — |
| (111 140) | 1 thread | -116 | -85581 | 38.85 | 88.9 | 1.9 | 769 | 1:05:04 |
| | 8 threads | -116 | -84171 | 39.02 | 89.3 | 5.2 | 828 | 0:16:39 |
| | 16 threads | -117 | -85688 | 39.01 | 89.3 | 6.3 | 830 | 0:11:36 |
| | 64 threads | -115 | -83890 | 39.03 | 89.6 | 10.1 | 820 | 0:09:11 |
| U5 | Start | -62 | -27540 | 34.08 | 82.0 | 0.0 | — | — |
| (119 228) | 1 thread | -59 | -9116 | 34.23 | 82.8 | 0.0 | 209 | 0:24:01 |
| | 8 threads | -59 | -8928 | 34.39 | 82.9 | 0.0 | 221 | 0:09:20 |
| | 16 threads | -59 | -9540 | 34.35 | 82.7 | 0.0 | 222 | 0:08:11 |
| | 64 threads | -59 | -9686 | 34.49 | 82.9 | 0.0 | 221 | 0:07:12 |
| U6 | Start | -121 | -226390 | 198.86 | 88.6 | 25.8 | — | — |
| (254 208) | 1 thread | -104 | -192108 | 189.19 | 88.9 | 40.0 | 1218 | 2:26:33 |
| | 8 threads | -101 | -189738 | 193.27 | 88.8 | 36.6 | 1229 | 0:29:01 |
| | 16 threads | -105 | -191376 | 194.15 | 88.9 | 35.4 | 1407 | 0:21:24 |
| | 64 threads | -106 | -192347 | 195.69 | 88.8 | 32.3 | 1448 | 0:18:37 |
| U7 | Start | -58 | -66021 | 100.08 | 83.7 | 0.0 | — | — |
| (276 799) | 1 thread | -55 | -24599 | 101.87 | 84.0 | 0.0 | 413 | 0:55:17 |
| | 8 threads | -55 | -22745 | 102.07 | 83.9 | 0.4 | 410 | 0:20:00 |
| | 16 threads | -55 | -24389 | 102.19 | 83.7 | 0.0 | 396 | 0:16:38 |
| | 64 threads | -55 | -25114 | 101.95 | 83.9 | 0.0 | 398 | 0:15:42 |
| U8 | Start | -85 | -840433 | 729.13 | 86.2 | 36.5 | — | — |
| (1 681 671) | 1 thread | -70 | -445934 | 734.46 | 86.5 | 19.4 | 3334 | 9:36:29 |
| | 8 threads | -73 | -446434 | 735.74 | 86.5 | 21.5 | 3371 | 2:28:43 |
| | 16 threads | -78 | -367789 | 740.23 | 86.4 | 27.7 | 3490 | 2:02:20 |
| | 64 threads | -75 | -281910 | 746.92 | 86.4 | 24.1 | 4015 | 1:57:26 |

TABLE 7.5. GRBO results with different numbers of threads. # TA denotes the total number of transactions that were performed.

| | Time per $10^6$ Transactions [h:mm::ss] | Wall Time in BRG [%] | BRG Waiting [%] |
|---|---|---|---|
| 1 thread | 2:52:52 | 61.30 | 0.00 |
| 4 threads | 0:57:40 | 35.83 | 0.08 |
| 8 threads | 0:44:06 | 25.74 | 0.18 |
| 16 threads | 0:35:03 | 13.67 | 0.44 |
| 24 threads | 0:32:19 | 10.02 | 0.73 |
| 32 threads | 0:31:37 | 8.06 | 0.97 |
| 48 threads | 0:30:03 | 5.81 | 1.87 |
| 64 threads | 0:29:14 | 4.90 | 3.96 |

TABLE 7.6. Multi-threaded performance of BonnRouteGlobal (BRG) in GRBO on U8 (1 681 671 nets) with different numbers of threads.

A more detailed analysis of the running time performance of multi-threaded Incremental BonnRouteGlobal on U8 is given by Table 7.6 and Figure 7.17: The first column in Table 7.6 measures the total running time of our GRBO flow per one million transactions — as one can see in Table 7.5, the total number of transactions can vary between different runs with different thread counts, so dividing by it allows for a better comparison of running times.

The second column measures the fraction of the running time that is spent in BonnRouteGlobal: We take the total running time spent in BonnRouteGlobal during GRBO (accumulated over all threads) and divide it by the number of threads to get the average running time per thread that is spent in BonnRouteGlobal. This average running time per thread in turn is divided by the GRBO running time and displayed in the table. Lastly, we display the fraction of the time that BonnRouteGlobal spends waiting on mutexes in BonnTools code in the third column — mutexes in third-party code that is used by BonnRouteGlobal (e.g. the memory manager) are not tracked.

The first thing to notice is that using more threads does not negatively impact the quality of results. In many cases, the opposite is true — as already pointed out earlier, using more threads often results in more transactions being performed, which can lead to better results. Except for the unit U8, this effect is not very large. Moreover, the transaction count is not directly controlled by Incremental BonnRouteGlobal, but by the timing optimization environment Incremental BonnRouteGlobal is running in. However, apart from cases where transaction counts differ significantly (e.g. on U8), similar results are obtained by different thread counts.

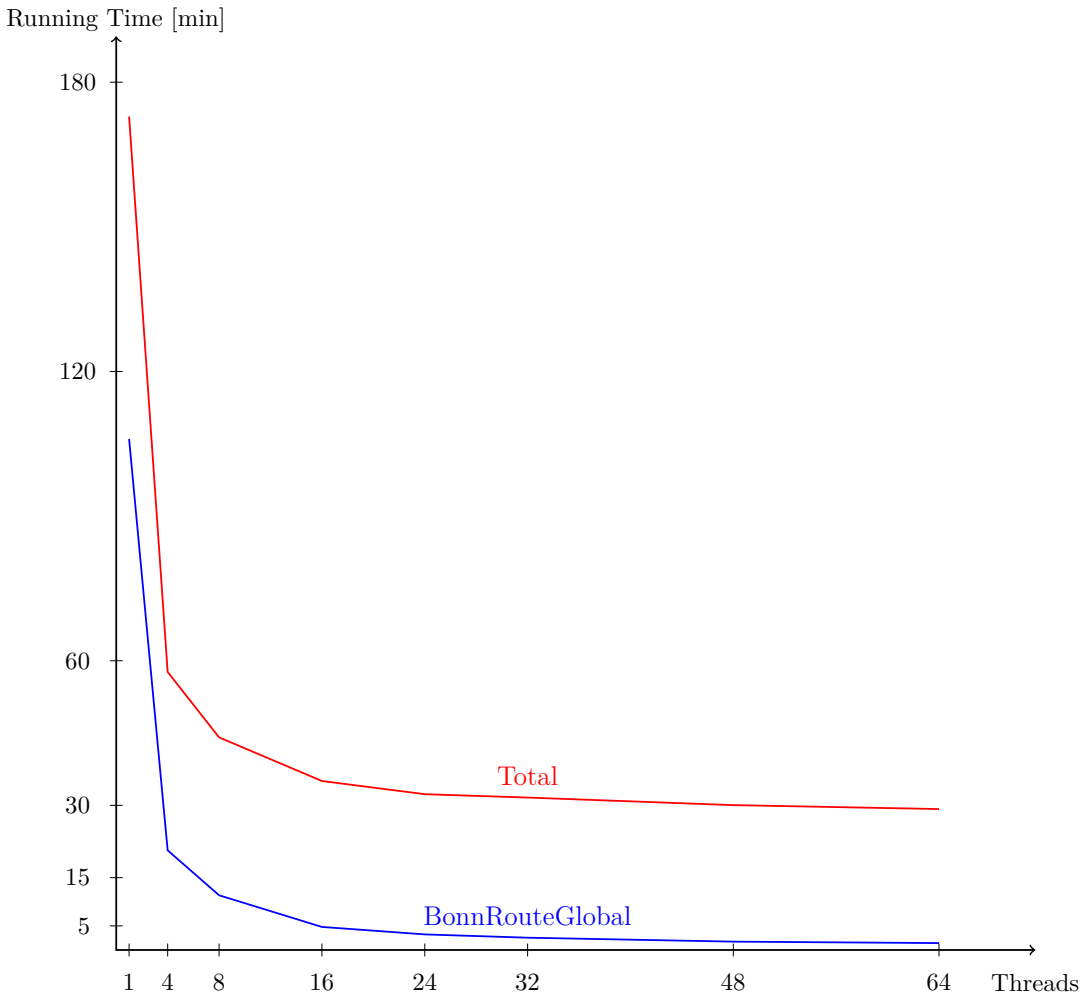



FIGURE 7.17. Visualization of the data from Table 7.6: We run GRBO on U8 (1 681 671 nets) and measure the total running time of our GRBO flow and the average running time per thread spent in BonnRouteGlobal. As in Table 7.6, the running times shown are per million transactions.

When it comes to running time, the data from Table 7.6 and Figure 7.17 shows that scaling is reasonable when the number of threads is low, but the gain in running time performance flattens out starting at 16 threads, and almost stalls for thread counts above 32. However, the fraction of running time spent in BonnRouteGlobal decreases as the number of threads increases, which indicates that BonnRouteGlobal exhibits a better scaling than the total GRBO flow. With more than 24 threads, the running time spent in BonnRouteGlobal even drops below ten percent. Therefore, it seems that there is not much to gain by speeding up the multi-threaded performance of Incremental

BonnRouteGlobal at high thread counts. This observation is confirmed by the small amount of time that BonnRouteGlobal actually spends waiting: It is less than one percent for up to 32 threads and less than four percent for up to 64 threads, which makes it negligible for reasonable thread counts.
As a summary, we conclude that running Incremental BonnRouteGlobal with multiple threads does not negatively impact the quality of results, and its running time performance can be considered very satisfactory given its small wait times and significantly decreasing fraction of the total running time when higher thread counts are used.

7.5.3.2. *Comparing Usage Update Schemes.* In this section we compare experimental results for the three usage update schemes from Section 7.5.2. We present two tables: In Table 7.7 we compare the three update schemes in our default flow that uses minimal reroutes as presented in Section 7.3. In order to exaggerate the effects of our usage update schemes, we run a second experiment where we disable the minimal reroute feature from Section 7.3, i.e. we reroute every modified net from scratch. The results for this second experiment are presented in Table 7.8. This is a hypothetical scenario, but it amplifies the effects of our usage update schemes, and is therefore worth investigating. Moreover, we use 64 threads for all experiments from this section, which can be considered a comparatively large number of threads for our application.
Again, the "Start" row in our tables represents the start of GRBO, directly after timing-aware BonnRouteGlobal (cf. Chapters 4 and 5) has been run. Here, we take the metrics of the run with the semi thread-local update scheme, which is the default in our flow. In the other experiments, the code used for creating the starting point for GRBO is the same, so only minor variations should occur. Each of the other rows then shows the results after GRBO with the respective usage update scheme being used. In addition to columns displaying general metrics that are explained in Appendix A.4, Tables 7.7 and 7.8 contain a column labeled "Undo %", which displays the percentage of transactions that are undone (cf. Section 7.2). As usual, the congestion target is at 90% in our runs. Therefore, a wACE4 that is fairly below 90% implies that the unit can be considered uncongested, while a wACE4 close to or above 90% implies high congestion.
Looking at the results from both tables, one can make the following central statement: While the semi thread-local and fully thread-local update schemes both perform roughly equally well and do not increase overflow by significant amounts, using the fully global update scheme can result in substantial increases in routing overflow. By the very nature of minimal reroutes, this effect is weak when minimal reroutes are used, but it can still be observed on U6 and U8. However, when minimal reroutes are turned off and every net is rerouted from scratch, then large overflow increases occur on most units with the fully global update scheme. Moreover, overflow increases are more drastic on larger units

| Unit (# nets) | After... | WS [ps] | FOM [ps] | Undo [%] | wACE4 [%] | OFtgt [100 pitch$^2$] | RT [h:mm:ss] |
|---|---|---|---|---|---|---|---|
| U1 | Start | -129 | -29226 | — | 88.7 | 5.5 | — |
| (77 528) | Fully global | -72 | -22237 | 95.51 | 89.1 | 6.8 | 0:05:10 |
| | Fully thread-local | -72 | -22579 | 95.52 | 89.1 | 6.0 | 0:04:59 |
| | Semi thread-local | -72 | -22038 | 95.49 | 89.0 | 7.2 | 0:05:05 |
| U2 | Start | -97 | -58077 | — | 87.5 | 0.0 | — |
| (79 119) | Fully global | -86 | -48229 | 97.11 | 87.8 | 0.0 | 0:04:50 |
| | Fully thread-local | -88 | -48597 | 97.13 | 88.0 | 0.3 | 0:05:18 |
| | Semi thread-local | -85 | -48420 | 97.11 | 87.8 | 0.3 | 0:05:08 |
| U3 | Start | -149 | -118530 | — | 86.5 | 0.0 | — |
| (100 827) | Fully global | -74 | -85553 | 95.27 | 86.7 | 0.0 | 0:06:26 |
| | Fully thread-local | -74 | -86098 | 95.27 | 86.9 | 0.0 | 0:06:11 |
| | Semi thread-local | -75 | -86014 | 95.27 | 86.9 | 0.0 | 0:06:04 |
| U4 | Start | -196 | -151908 | — | 89.5 | 8.7 | — |
| (111 140) | Fully global | -117 | -84846 | 94.64 | 89.7 | 9.6 | 0:08:59 |
| | Fully thread-local | -117 | -85079 | 94.71 | 89.5 | 7.1 | 0:08:54 |
| | Semi thread-local | -117 | -85725 | 94.63 | 89.5 | 6.4 | 0:08:48 |
| U5 | Start | -62 | -27955 | — | 81.9 | 0.0 | — |
| (119 228) | Fully global | -59 | -9920 | 96.08 | 82.8 | 0.0 | 0:07:24 |
| | Fully thread-local | -59 | -9703 | 96.09 | 82.6 | 0.0 | 0:07:42 |
| | Semi thread-local | -59 | -10089 | 96.08 | 83.0 | 0.0 | 0:07:54 |
| U6 | Start | -121 | -225122 | — | 88.6 | 28.0 | — |
| (254 208) | Fully global | -105 | -193569 | 96.38 | 89.2 | 48.7 | 0:18:13 |
| | Fully thread-local | -105 | -190746 | 96.41 | 88.9 | 32.8 | 0:19:17 |
| | Semi thread-local | -107 | -187005 | 96.37 | 88.9 | 36.6 | 0:19:01 |
| U7 | Start | -62 | -65828 | — | 83.6 | 0.0 | — |
| (276 799) | Fully global | -55 | -25463 | 93.55 | 83.8 | 1.1 | 0:16:05 |
| | Fully thread-local | -55 | -24040 | 93.60 | 83.8 | 0.0 | 0:15:17 |
| | Semi thread-local | -55 | -28096 | 93.54 | 83.8 | 0.0 | 0:16:16 |
| U8 | Start | -86 | -840934 | — | 86.1 | 42.0 | — |
| (1 681 671) | Fully global | -75 | -279623 | 94.70 | 86.5 | 81.1 | 2:06:38 |
| | Fully thread-local | -86 | -281313 | 94.70 | 86.4 | 25.7 | 2:04:35 |
| | Semi thread-local | -74 | -284244 | 94.73 | 86.4 | 25.1 | 2:00:18 |

TABLE 7.7. Comparison of the different usage update schemes from Section 7.5.2 running GRBO with 64 threads.

| Unit (# nets) | After... | WS [ps] | FOM [ps] | Undo [%] | wACE4 [%] | OFtgt [100 pitch$^2$] | RT [h:mm:ss] |
|---|---|---|---|---|---|---|---|
| U1 | Start | -129 | -29700 | — | 89.0 | 7.4 | — |
| (77 528) | Fully global | -77 | -25832 | 96.16 | 91.2 | 81.1 | 0:05:39 |
| | Fully thread-local | -77 | -25054 | 96.07 | 89.8 | 16.3 | 0:05:15 |
| | Semi thread-local | -77 | -24150 | 96.11 | 89.5 | 8.4 | 0:05:20 |
| U2 | Start | -97 | -57375 | — | 87.6 | 0.0 | — |
| (79 119) | Fully global | -103 | -52803 | 97.19 | 88.9 | 3.1 | 0:05:16 |
| | Fully thread-local | -93 | -49153 | 97.54 | 88.0 | 0.2 | 0:05:59 |
| | Semi thread-local | -94 | -51004 | 97.70 | 88.0 | 0.0 | 0:06:05 |
| U3 | Start | -149 | -118438 | — | 86.3 | 0.0 | — |
| (100 827) | Fully global | -83 | -91254 | 95.57 | 87.8 | 0.7 | 0:06:01 |
| | Fully thread-local | -81 | -90848 | 95.59 | 87.4 | 0.0 | 0:06:08 |
| | Semi thread-local | -82 | -91276 | 95.47 | 87.4 | 0.1 | 0:05:56 |
| U4 | Start | -195 | -152136 | — | 89.4 | 7.4 | — |
| (111 140) | Fully global | -143 | -112105 | 95.27 | 92.3 | 195.7 | 0:09:48 |
| | Fully thread-local | -143 | -99436 | 95.27 | 90.0 | 18.2 | 0:09:07 |
| | Semi thread-local | -138 | -103014 | 95.29 | 90.4 | 37.7 | 0:09:34 |
| U5 | Start | -62 | -27949 | — | 81.8 | 0.0 | — |
| (119 228) | Fully global | -59 | -20109 | 96.90 | 85.9 | 0.0 | 0:08:30 |
| | Fully thread-local | -59 | -20531 | 96.91 | 85.9 | 0.0 | 0:08:18 |
| | Semi thread-local | -59 | -20408 | 96.93 | 85.9 | 0.0 | 0:08:15 |
| U6 | Start | -121 | -225830 | — | 88.6 | 26.8 | — |
| (254 208) | Fully global | -115 | -225075 | 97.07 | 95.7 | 1363.5 | 0:23:09 |
| | Fully thread-local | -106 | -227839 | 97.18 | 89.3 | 46.8 | 0:23:07 |
| | Semi thread-local | -114 | -220931 | 97.13 | 89.2 | 41.4 | 0:20:45 |
| U7 | Start | -60 | -66052 | — | 83.7 | 0.0 | — |
| (276 799) | Fully global | -55 | -32178 | 94.55 | 91.2 | 432.0 | 0:18:25 |
| | Fully thread-local | -55 | -30329 | 94.47 | 85.4 | 2.8 | 0:17:55 |
| | Semi thread-local | -55 | -31867 | 94.41 | 85.4 | 2.7 | 0:17:42 |
| U8 | Start | -86 | -840612 | — | 86.1 | 37.5 | — |
| (1 681 671) | Fully global | -73 | -417696 | 96.05 | 88.6 | 971.5 | 2:43:01 |
| | Fully thread-local | -74 | -424260 | 96.09 | 87.4 | 32.9 | 2:40:28 |
| | Semi thread-local | -74 | -429837 | 96.07 | 87.4 | 18.1 | 2:45:55 |

TABLE 7.8. Comparison of the different usage update schemes from Section 7.5.2 running GRBO with 64 threads. The minimal reroute feature from Section 7.3 has been disabled for this run, i.e. all nets are rerouted from scratch.

with a full layer stack: For example, U7 with a starting wACE4 of roughly 84% can be considered quite uncongested, but Table 7.8 still shows large overflow increases with the fully global update scheme. This can be explained by the fact that U7 has an almost complete layer stack, containing two of the possible four thickest layers, where very thick wires can be used. Therefore, congestion problems caused by concurrency can be more profound, as individual wires on the topmost layers may use up a significant fraction of the available routing space on a global routing graph edge. In fact, basically all the overflow on U7 is on the topmost layers.
Our results can be explained as follows: Firstly, the semi-thread local update scheme is the most conservative of the three, which makes it seem like the natural option for producing the least amount of overflow. The differences in performance between the fully global and fully thread-local update schemes may be surprising at first, but can be explained by looking at the "Undo" columns: As one can see, the vast majority of transactions is not committed, but undone. This implies a high likelihood of the situation outlined in Example 7.34, where overflow is created using the fully global update scheme because one thread $\tau$ undoes its transaction while another thread has already occupied some of the resources that were freed temporarily in $\tau$'s transaction. On the other hand, the situation from Example 7.36 is less likely to occur: Using the fully thread-local update scheme, two threads might use the same resource temporarily such that overflow is created when both commit their transaction, but looking at the percentages of transactions that are undone, it is very unlikely that both threads will actually commit their transactions. As a result, the fully thread-local update scheme does not produce noticeable more overflow than the more conservative semi thread-local update scheme. This leads to the conclusion that in our current application, there is no noticeable difference between the semi thread-local and fully thread-local update scheme, but the fully global update scheme is prone to producing significant overflow on congested designs, in particular when rerouting is done on a large scale. We therefore choose to use the conservative semi-thread local update scheme as a default, although the fully thread-local update scheme also seems to be a valid choice based on our experiments.

## 7.6. Routing Flow Results

After providing experimental results measuring the practical performance of the features presented in Sections 7.3, 7.4 and 7.5 individually, we now provide results that measure the performance of our overall routing flow. To this end, we run our complete routing flow on the five largest designs in our testbed and measure general metrics after each major step. For information regarding our testbed and an explanation of the measured metrics we refer to Appendix A. Our runs from this section are performed with 16 threads

on an Intel Xeon E5-2667 v2 server running at 3.30 GHz, and metrics on global routes are computed after connecting to exact shapes as outlined in Section 6.4.

Our results are given in Table 7.9. Here, the first row labeled *Estimates* shows the results that are attained by the routing tree estimates that are used during timing optimization before routing, which are approximately shortest Steiner trees embedded on the two lowest layers of the layer assignment of the net, using the wire type assigned to the net for all segments. Each of the subsequent rows then contains the results after one of the major steps in our routing flow: We start with global routing using RC-Aware BonnRouteGlobal from Chapters 4 and 5 (*RC-Aware BRG*) and then run *GRBO* from this chapter on the resulting global routes. This is followed by detailed routing using BonnRouteDetailed (*BRD*), our second routing based optimization step *DRBO*, and a final detailed routing (*BRD after DRBO*) that implements the global wires added during DRBO.

At this point it should be noted that in our flow, BonnRouteDetailed completely routes all nets and does not leave open connections — if BonnRouteDetailed cannot close a connection without violating minimum distance rules, then it violates them by using (usually short) *guide wires* to close the connection. In our runs, the number of nets containing guide wires is below 50 in both invocations of BonnRouteDetailed on all designs except U8, where around 2000 nets need to use guide wires to access pins at large blockages. All in all, however, the detailed routing on these designs can be considered essentially complete.

Looking at the results, the first thing to notice is that RC-Aware BonnRouteGlobal achieves significantly better timing results than the pre-routing estimates, and it does so in a reasonable running time without creating overcongestion. As the details of this comparison are already discussed in Section 5.5, we do not elaborate on this any further in this section. Further large timing improvements can be achieved by running GRBO after RC-Aware BonnRouteGlobal: Worst slack, FOM and the number of electrical violations are improved on every design, and often by large amounts. These improvements come at the expense of a small power increase on most units, but in relation to the improvements in timing metrics, this power increase can be considered a worthwhile expenditure. Congestion is also held in check, and does not change significantly during GRBO. Running times can be considered small on all but the biggest design U8, where our GRBO flow takes roughly two hours. In light of the large improvements in quality of results and the detailed routing running time of roughly 18 hours, the GRBO running time can still be considered tolerable. Moreover, as our data from Table 7.6 from Section 7.5.3.1 shows, less than 15% of the GRBO running time on U8 is spent

| Unit (# nets) | After... | WS [ps] | FOM [ps] | EV | PWR [mW] | wACE4 [%] | OFtgt [100 pitch$^2$] | RT [hh:mm:ss] |
|---|---|---|---|---|---|---|---|---|
| U4 | Estimates | -245 | -214126 | 149 | 37.07 | — | — | — |
| (111 140) | RC-Aware BRG | -196 | -152541 | 204 | 37.05 | 89.4 | 7.3 | 00:03:10 |
| | GRBO | -116 | -84530 | 144 | 38.94 | 89.2 | 2.7 | 00:10:45 |
| | BRD | -130 | -138307 | 364 | 39.33 | 95.6 | 190.0 | 00:53:55 |
| | DRBO | -115 | -123131 | 224 | 41.20 | 95.3 | 173.1 | 00:07:39 |
| | BRD after DRBO | -116 | -136931 | 271 | 41.23 | 94.9 | 165.5 | 00:37:30 |
| U5 | Estimates | -63 | -58267 | 52 | 33.88 | — | — | — |
| (119 228) | RC-Aware BRG | -62 | -27486 | 5 | 34.08 | 81.7 | 0.0 | 00:03:03 |
| | GRBO | -59 | -10016 | 1 | 34.39 | 82.6 | 0.0 | 00:08:45 |
| | BRD | -62 | -33828 | 21 | 34.96 | 79.1 | 0.0 | 00:19:21 |
| | DRBO | -60 | -15024 | 1 | 35.50 | 79.1 | 0.0 | 00:08:58 |
| | BRD after DRBO | -60 | -20632 | 2 | 35.51 | 79.7 | 0.7 | 00:07:42 |
| U6 | Estimates | -122 | -325030 | 237 | 197.18 | — | — | — |
| (254 208) | RC-Aware BRG | -121 | -224967 | 1410 | 198.87 | 88.6 | 24.2 | 00:08:15 |
| | GRBO | -105 | -190008 | 1204 | 194.19 | 88.9 | 41.1 | 00:23:04 |
| | BRD | -128 | -277386 | 1619 | 195.28 | 98.3 | 3051.1 | 01:38:07 |
| | DRBO | -122 | -246809 | 781 | 198.88 | 98.3 | 3037.1 | 00:17:56 |
| | BRD after DRBO | -133 | -270978 | 808 | 198.99 | 99.2 | 4937.4 | 01:06:39 |
| U7 | Estimates | -94 | -125151 | 375 | 98.98 | — | — | — |
| (276 799) | RC-Aware BRG | -61 | -64116 | 781 | 100.10 | 83.6 | 0.0 | 00:09:15 |
| | GRBO | -55 | -23936 | 603 | 101.65 | 83.8 | 0.0 | 00:17:04 |
| | BRD | -60 | -86038 | 737 | 103.22 | 95.7 | 1295.3 | 03:13:08 |
| | DRBO | -60 | -36008 | 670 | 104.77 | 95.7 | 1279.7 | 00:16:40 |
| | BRD after DRBO | -55 | -48310 | 673 | 104.97 | 98.6 | 3375.6 | 01:00:11 |
| U8 | Estimates | -109 | -1979601 | 3110 | 705.87 | — | — | — |
| (1 681 671) | RC-Aware BRG | -87 | -838015 | 9341 | 729.19 | 86.2 | 34.6 | 01:01:06 |
| | GRBO | -82 | -358955 | 2924 | 740.27 | 86.5 | 27.3 | 02:12:14 |
| | BRD | -79 | -1186585 | 7436 | 750.47 | 95.2 | 13966.6 | 18:01:37 |
| | DRBO | -74 | -991936 | 2271 | 765.64 | 95.2 | 13896.8 | 02:24:51 |
| | BRD after DRBO | -81 | -1119024 | 2553 | 766.54 | 96.7 | 23008.7 | 06:39:36 |

TABLE 7.9. Experimental results of our new routing flow consisting of multiple invocations of BonnRouteGlobal (BRG) and BonnRouteDetailed (BRD) and timing optimization with Incremental BonnRouteGlobal (GRBO and DRBO). For comparison, pre-routing estimates are also listed.

in BonnRouteGlobal when using 16 threads, and the number drops further when more threads are used.

After detailed routing, timing metrics take a hit. To some extent, this is expected, as detailed routing is a hard problem in practice, and detailed routes are expected to contain more (mainly local) detours than our global routes due to the nature of the detailed routing problem. This is particularly true as we apply local optimizations to our global routes, as described in Chapter 6. However, improving the interplay between the global and the detailed router and possibly using track assignment (cf. Section 1.3.3) can certainly help reducing the gap.

When comparing congestion metrics, one can observe a discontinuity between the GRBO and BRD rows, which is due to the way congestion is measured in these two steps: When running RC-Aware BonnRouteGlobal and GRBO, we are global routing a basically empty design, and the global router distributes the global wiring in order to obey routing capacity constraints. These routing capacity constraints are always expressed with respect to our congestion target of 90%. Therefore, BonnRouteGlobal attempts to keep congestion below 90%, which can also be observed by looking at the wACE4 numbers.

After detailed routing, however, routing space usages are measured based on space that is left empty by the detailed wires, as described in Section 7.1.3. This can naturally result in large OFtgt values, as the congestion target is 90%, and the detailed router may easily use more than 90% of the available routing space in certain regions. This is also the reason for increases in the wACE4 metric, which can easily be dominated by a relatively small fraction of heavily used global routing graph edges after detailed routing. Therefore, based on the methods used to measure congestion, the discontinuity in congestion metrics between the GRBO and BRD step is expected. An illustration of this is given by Figure 7.18. Based on these observations, it is fair to say that the wACE4 and OFtgt metrics, which are useful for predicting routability before detailed routing, do not provide reliable congestion assessments after detailed routing with the current methods of measuring congestion after detailed routing.

The results in the DRBO row can be described in a similar way as the results in the GRBO row: Timing numbers and electrical violations are improved at the expense of a higher power consumption, the running time is reasonable, and congestion does not increase. However, the improvements are smaller compared to GRBO, which has two main reasons: Firstly, global routes have already been optimized during GRBO, which leaves less room for improvement in DRBO. Secondly, changes during DRBO are generally more restricted than during GRBO (e.g. only small pin movements are allowed in DRBO), as large changes after detailed routing are generally tried to be avoided. We note

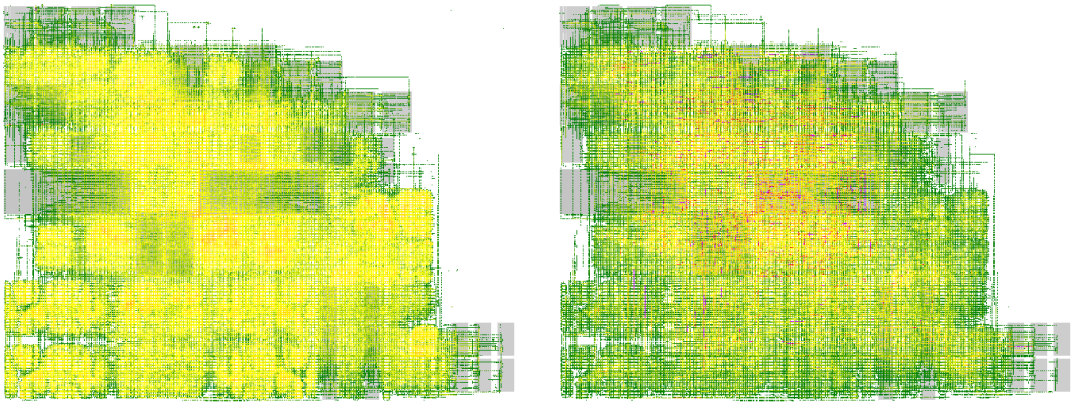


FIGURE 7.18. Congestion plots on U8 directly after RC-Aware BonnRouteGlobal (left) and BonnRouteDetailed (right): Naturally, BonnRouteGlobal distributes the wiring in order to meet the congestion target of 90%, which results in widespread yellow and orange areas where congestion is just below the congestion target. After BonnRouteDetailed, the plot shows less congestion for the most part, but some heavily congested edges scattered around the chip area result in large wACE4 and OFtgt values (cf. Table 7.9).

that as already described in Section 7.1.2, we use the same RICE [95] timing extraction method after all steps in our routing flow, which makes timing results comparable. In VLSI design production flows, however, things can be different: Here, one can use more precise timing models after detailed routing which take coupling capacitances based on the actual locations of detailed wires into account. In that case, detailed routing based optimization can lead to much bigger timing improvements, as the timing model change can cause the netlist to be inferior from a timing perspective.

Finally, we run BonnRouteDetailed a second time after DRBO. Compared to the results after the first invocation of BonnRouteDetailed, timing numbers and electrical violations are generally noticeably better. Here, one can again observe some degree of timing degradation compared to the DRBO numbers, although it is much smaller than the one occuring between GRBO and BRD. Naturally, this is due to the fact that DRBO changes are typically local, and most nets are not touched during DRBO. However, there is still room for improvement, as speaking of today, BonnRouteDetailed cannot properly handle nets that contain a mix of global and detailed wires, which occur naturally after running DRBO. Therefore, we convert all wires in nets containing a mix of global and detailed wires to global wires, which allows BonnRouteDetailed to route those nets. This can result in unnecessarily large routing changes, as whole nets are rerouted even in cases where closing only very short connections would suffice. Here, a proper support for

handling such mixed nets in BonnRouteDetailed is likely to result in a better quality of results and smaller running times in the second invocation of BonnRouteDetailed.
All in all, the results achieved by our new routing flow are convincing: RC-Aware BonnRouteGlobal achieves significantly better timing results than the pre-routing estimates, and these results are further improved by our routing based optimization steps. This allows us to generate global and detailed wires whose timing properties outperform the ones of the pre-routing estimates by a substantial amount, which gives practical validation to the theoretical approaches and results devised in this thesis.

APPENDIX A

# Experimental Results

This appendix contains additional information regarding our experimental results presented throughout this thesis. It can be used as a reference when looking at the various experimental results sections, in particular if the meaning of some of the evaluated metrics is unclear (cf. Section A.4).

## A.1. Our Testbed

Our testbed consists of state-of-the-art microprocessor units from the 14nm technology node. They are part of recent high-performance processor units that are designed and brought to market by IBM. As such, they should reflect the current state of technology when it comes to the design of high-performance integrated circuits. For our runs we use snapshots from the IBM design flow that are taken at a point directly before routing.

## A.2. Our Platform

Apart from their Tcl interface, our tools are implemented in C++ using a Linux environment. Our experiments are then conducted under Linux on x86 server workstations. The exact workstation used varies from experiment to experiment, and we give the processor specifications in the various experimental results sections.

## A.3. Metric Evaluation

The metrics in our tables are either evaluated directly by our router, or by IBM tools that are used for the evaluation of the respective metric in the IBM design flow. In particular, timing results are reported by IBM EinsTimer, which is the timing engine used throughout the IBM design flow. Although we optimize Elmore delays throughout this thesis, we use more accurate RICE [95] extractions for timing computations in our experimental results, the reason being that this is the timing extraction method used at the stage in the IBM design flow where our global router is used.

## A.4. Metrics

Our tables in this thesis contain various metrics, and we explain the most general ones shortly in this section. They are identified and sorted alphabetically by the abbreviation

that we use for them throughout this thesis. Some result tables contain metrics that are specific to the section containing the results. These metrics are then explained in the respective section and not listed here. The metrics are:

- **EV:** The number of *electrical violations*, i.e. the sum of the numbers of slew and capacitance limit violations, as described in Section 3.5.
- **FOM:** The *figure of merit* as defined in Section 3.3.2. In our runs, the slack target is a design specific parameter between five and ten picoseconds.
- **OFtgt:** The *routing overflow with respect to the congestion target*. In our runs, this is the same as OF90 (see definition of OF$x$ below), as we use a congestion target of 90% in our runs. Here, a congestion target of 90% means that the estimated routing capacities are scaled by 90%, and the actual global routing algorithm regards these reduced capacities as routing capacity constraints.
- **OF$x$:** The *routing overflow with respect to edge capacities scaled by $x$%*, expressed as a wire area with a unit of $100\rho^2$, where $\rho$ is the minimum track pitch (cf. Section 1.3) across all routing layers (often just called *tracks*). The formula for OF$x$ is

$$\begin{aligned}\mathrm{OF}x &= \sum_{e\in E(G)} \max\left\{\mathrm{usg}'(e) - 0.01x \cdot u(e), 0\right\} \cdot l(e) \\ &= 100\rho^2 \sum_{e\in E(G)} \frac{\max\left\{\mathrm{usg}'(e) - 0.01x \cdot u(e), 0\right\}}{\rho} \cdot \frac{l(e)}{100\rho},\end{aligned}$$

  where $G$ is the global routing graph, $\mathrm{usg}'(e)$ and $u(e)$ are the (not-normalized) usage and capacity of $e \in E(G)$ expressed in length units, and $l(e)$ is the length of $e$. Here, the traditional one-dimensional routing overflow is scaled by the length of the corresponding edge, which for example makes the metric more stable when changing tile sizes.
  With uniform tile sizes of $100\rho$, this definition of routing overflow coincides with the traditional one-dimensional definition, where the unit is tracks. In our experiments we use non-uniform tile sizes (cf. Section 1.3.2), which are oriented towards a mean tile size of $70\rho$.
- **PWR:** The *power consumption* of the chip.
- **RT:** The *running time* for the experiment. We use the *wall time* as a measurement for the running time.
- **wACE4:** The average of the four ACE metrics ACE(0.5), ACE(1), ACE(2) and ACE(5) defined by Wei et al. [120]. Basically, ACE($x$) for $x \in (0, 100]$ is the average congestion of the $x$% most congested edges. As we use a congestion

target of 90% in our runs, the boundary between easy-to-route and hard-to-route designs can be marked by a wACE4 of around 90%.

- **WL:** The *total wire length* on the chip (accumulated over all nets).
- **WS:** The *worst slack* as defined in Section 3.3.2.

# Summary

In this thesis we consider the global routing problem, which arises as one of the major subproblems during the physical design step in VLSI design. In this problem, a coarse layout of the wires on the chip has to be computed, which is then given to the detailed router as a guideline for computing the actual wiring on the chip. As such, during global routing, we are given a three-dimensional grid graph $G$ with edge capacities, called the *global routing graph*, and a set of *nets*, where each net consists of a set of pins associated with vertices in $G$. In each net, one pin is the sender of signals, while all other pins are receivers. The task in global routing is then to connect all nets without overusing the routing capacity on any of the edges in $G$.

The global routing problem has been studied extensively in the past, both from a theoretical and practical standpoint. Traditionally, next to obeying all routing capacity constraints, the objective has been to minimize wire length and possibly via (edges in $z$-direction) count. However, for a chip to function properly at its designated speed, timing constraints have to be fulfilled. In the traditional approach, this was attempted indirectly within the limits of the available means, e.g. by forcing timing-critical nets to be routed within a certain layer range (range of $z$-coordinates in $G$) or imposing upper bounds on the wire length of such nets. In Chapter 4 we present a new approach, where timing constraints are modeled directly during global routing: In joint work with Stephan Held, Dirk Müller, Daniel Rotter, Vera Traub and Jens Vygen [53], we extend the modeling of global routing as a Min-Max Resource Sharing Problem to also incorporate timing constraints. For measuring signal delays, we use the well-established Elmore delay model [35, 99] throughout this thesis.

In our timing-aware global routing framework, the key subproblem to be solved is the following: Given a net $N$, compute a Steiner tree connecting $N$ that minimizes a weighted sum of prices for routing space usages and signal delays. For $k$ pins, this problem is $NP$-hard to approximate within $o(\log k)$ [106], and even the special case $k = 2$ is $NP$-hard, as was shown by Hähnle and Rotter [97]. We therefore first present a fast approximation algorithm for the case $k = 2$, and the approximation bounds turn out to be very strong in practice. For nets with more than two pins we use a multi-stage approach that consists of solving the well-known Minimum Steiner Tree Problem, modifying the topology

of the computed tree as in [108], and using a slightly modified version of our algorithm for the two-pin case for computing new connections.

We evaluate the practical performance of our timing-aware global routing framework by running it as part of BonnRouteGlobal from the BonnTools program suite, which is used extensively by IBM for the development of state-of-the-art microprocessor units. We can show that compared to the traditional global routing method presented above, our new timing-aware global router achieves significantly better results on recent microprocessor units from the 14nm technology node.

A refinement of the global routing model is then presented in Chapter 6: Instead of only working on the global routing graph $G$, where each vertex represents a rectangular area on a given layer on the chip, we present an additional step that connects to the actual metal shapes of the pins (without obeying manufacturing rules such as minimum distance rules between wires). This allows for more precise timing computations and a better modeling of local routing space usages. The key part here is a layer assignment algorithm again minimizing a weighted sum of prices for routing space usages and signal delays, i.e. an algorithm that assigns $z$-coordinates — which have a profound impact on timing properties — to the edges of a two-dimensional input Steiner tree. In our layer assignment algorithm we use similar techniques as in Chapter 5 and derive similar bounds. Although this algorithm is designed for application in the context of Chapter 6, it can be used in a general global routing context as an addition to the timing-aware Steiner tree algorithm from Chapter 5.

At last, we discuss the topic of routing based optimization in Chapter 7. Here, the starting point is a complete global or detailed routing, and timing optimization tools make changes like moving circuits, adding repeaters and resizing circuits in order to improve timing. The task for the global router is then to adapt to these changes by incrementally updating the underlying routing. We investigate several aspects of this problem: We consider the problem of completing an almost complete routing for a net, which can for example arise when a circuit is moved or resized; we deal with the situation where an already routed net is subdivided by insertion of repeaters, and the wiring has to be distributed to the resulting subnets; and we deal with problems arising in multi-threaded contexts and evaluate the performance of the multi-threaded implementation of our incremental router. This enables a new routing flow, which consists of the sequence timing-aware global routing $\rightarrow$ routing based optimization $\rightarrow$ detailed routing $\rightarrow$ routing based optimization $\rightarrow$ detailed routing. On our 14nm microprocessor test cases from IBM we can demonstrate a strong performance of individual features of our incremental global router, and a strong performance of the resulting overall routing flow.

# Bibliography


[1] M. Ahrens, M. Gester, N. Klewinghaus, D. Müller, S. Peyer, C. Schulte, and G. Tellez, *Detailed Routing Algorithms for Advanced Technology Nodes*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 34 (2015), pp. 563–576. (Cited on page 25.)

[2] C. Albrecht, *Global Routing by New Approximation Algorithms for Multicommodity Flow*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 20 (2001), pp. 622–632. (Cited on page 35.)

[3] C. Albrecht, A. B. Kahng, I. Mandoiu, and A. Zelikovsky, *Floorplan Evaluation with Timing-Driven Global Wireplanning, Pin Assignment, and Buffer/Wire Sizing*, in Proceedings of ASP-DAC/VLSI Design 2002. 7th Asia and South Pacific Design Automation Conference and 15th International Conference on VLSI Design, 2002, pp. 580–587. (Cited on page 48.)

[4] N. Alon, D. Moshkovitz, and S. Safra, *Algorithmic Construction of Sets for K-Restrictions*, ACM Trans. Algorithms, 2 (2006), pp. 153–177. (Cited on page 76.)

[5] C. J. Alpert, D. P. Mehta, and S. S. Sapatnekar, eds., *Handbook of Algorithms for Physical Design Automation*, Auerbach Publications, Boston, USA, 2008. (Cited on pages 10 and 11.)

[6] J. Ao, S. Dong, S. Chen, and S. Goto, *Delay-Driven Layer Assignment in Global Routing Under Multi-Tier Interconnect Structure*, in Proceedings of the ACM International Symposium on Physical Design, 2013, pp. 101–107. (Cited on page 22.)

[7] S. Arora, *Polynomial Time Approximation Schemes for Euclidean Traveling Salesman and Other Geometric Problems*, Journal of the ACM, 45 (1998), pp. 753–782. (Cited on page 36.)

[8] C. Bartoschek, *Fast Repeater Tree Construction*, PhD thesis, University of Bonn, 2014. (Cited on page 12.)

[9] S. Batterywala, N. Shenoy, W. Nicholls, and H. Zhou, *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, ACM, 2002, pp. 59–66. (Cited on page 24.)

[10] T. Bihler, *Rounding Fractional Global Routings*, Master's thesis, University of Bonn, 2017. (Cited on page 50.)

[11] D. Bilò, *New Algorithms for Steiner Tree Reoptimization*, in 45th International Colloquium on Automata, Languages, and Programming (ICALP 2018), I. Chatzigiannakis, C. Kaklamanis, D. Marx, and D. Sannella, eds., Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2018, pp. 19:1–19:14. (Cited on page 128.)

[12] D. Bilò, H.-J. Böckenhauer, J. Hromkovič, R. Královič, T. Mömke, P. Widmayer, and A. Zych, *Reoptimization of Steiner Trees*, in Algorithm Theory – SWAT 2008, J. Gudmundsson, ed., Berlin, Heidelberg, 2008, Springer Berlin Heidelberg, pp. 258–269. (Cited on page 128.)

[13] D. Bilò and A. Zych, *New Advances in Reoptimizing the Minimum Steiner Tree Problem*, in Mathematical Foundations of Computer Science 2012: 37th International Symposium, MFCS 2012, Bratislava, Slovakia, August 27 – 31, B. Rovan, V. Sassone, and P. Widmayer, eds., Springer, 2012, pp. 184–197. (Cited on page 128.)

[14] A. Bock, S. Held, N. Kämmerling, and U. Schorr, *Local Search Algorithms for Timing-Driven Placement Under Arbitrary Delay Models*, in Proceedings of the 52nd Annual Design Automation Conference, DAC '15, ACM, 2015, pp. 29:1–29:6. (Cited on page 13.)

[15] H.-J. Böckenhauer, J. Hromkovič, R. Královič, T. Mömke, and P. Rossmanith, *Reoptimization of Steiner Trees: Changing the Terminal Set*, Theoretical Computer Science, 410 (2009), pp. 3428 – 3435. (Cited on page 127.)

[16] K. D. Boese, A. B. Kahng, B. A. McCoy, and G. Robins, *Fidelity and Near-Optimality of Elmore-Based Routing Constructions*, in IEEE International Conference on Computer Design, 1993, pp. 81–84. (Cited on page 44.)

[17] ———, *Rectilinear Steiner Trees with Minimum Elmore Delay*, in Proceedings of the 31st Annual Design Automation Conference, New York, 1994, ACM, pp. 381–386. (Cited on page 65.)

[18] ———, *Near-Optimal Critical Sink Routing Tree Constructions*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 14 (1995), pp. 1417–1436. (Cited on page 65.)

[19] K. D. Boese, A. B. Kahng, and G. Robins, *High-Performance Routing Trees with Identified Critical Sinks*, in Proceedings of the 30th International Design Automation Conference, 1993, pp. 182–187. (Cited on page 65.)

[20] U. Brenner, M. Struzyna, and J. Vygen, *BonnPlace: Placement of Leading-Edge Chips by Advanced Combinatorial Algorithms*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 27 (2008), pp. 1607–1620. (Cited on page 11.)

[21] J. Byrka, F. Grandoni, T. Rothvoss, and L. Sanità, *Steiner Tree Approximation via Iterative Randomized Rounding*, J. ACM, 60 (2013), pp. 6:1–6:33. (Cited on pages 35 and 128.)

[22] R. C. Carden, J. Li, and C.-K. Cheng, *A Global Router with a Theoretical Bound on the Optimal Solution*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 15 (1996), pp. 208–216. (Cited on page 34.)

[23] M. Celik, L. Pileggi, and A. Odabasioglu, *IC Interconnect Analysis*, Kluwer Academic Publishers, Boston, 2002. (Cited on page 45.)

[24] C.-C. Chang and J. Cong, *Pseudo Pin Assignment with Crosstalk Noise Control*, in Proceedings of the 2000 International Symposium on Physical Design, ACM, 2000, pp. 41–47. (Cited on page 24.)

[25] Y.-J. Chang, Y.-T. Lee, J.-R. Gao, P.-C. Wu, and T.-C. Wang, *NTHU-Route 2.0: A Robust Global Router for Modern Designs*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 29 (2010), pp. 1931–1944. (Cited on page 34.)

[26] M. Cho, K. Lu, K. Yuan, and D. Z. Pan, *BoxRouter 2.0: A Hybrid and Robust Global Router with Layer Assignment for Routability*, ACM Transactions on Design Automation of Electronic Systems, 14 (2009), pp. 32:1–32:21. (Cited on page 34.)

[27] C. Chu and Y. C. Wong, *FLUTE: Fast Lookup Table Based Rectilinear Steiner Minimal Tree Algorithm for VLSI Design*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 27 (2008), pp. 70–83. (Cited on page 36.)

[28] J. Cong, *An Interconnect-Centric Design Flow for Nanometer Technologies*, Proceedings of the IEEE, 89 (2001), pp. 505–528. (Cited on page 21.)

[29] J. Cong, K.-S. Leung, and D. Zhou, *Performance-Driven Interconnect Design Based on Distributed RC Delay Model*, in Proceedings of the 30th International Design Automation Conference, ACM, 1993, pp. 606–611. (Cited on page 65.)

[30] P. Coussy and A. Morawiec, *High-Level Synthesis*, Springer, 2008. (Cited on page 8.)

[31] P. Cremer, *Algorithms for Cell Layout*, PhD thesis, University of Bonn, 2019. (Cited on page 9.)

[32] S. Daboul, S. Held, J. Vygen, and S. Wittke, *An Approximation Algorithm for Threshold Voltage Optimization*, ACM Transactions on Design Automation of Electronic Systems, 23 (2018), pp. 68:1–68:16. (Cited on page 13.)

[33] S. Devadas, A. Ghosh, and K. Keutzer, *Logic Synthesis*, McGraw-Hill, Inc., New York, 1994. (Cited on page 8.)

[34] E. W. Dijkstra, *A Note on Two Problems in Connexion with Graphs*, Numerische Mathematik, 1 (1959), pp. 269–271. (Cited on pages 16, 24, 67, and 68.)

[35] W. Elmore, *The Transient Response of Damped Linear Networks with Particular Regard to Wideband Amplifiers*, Journal of Applied Physics, 19 (1948), pp. 55–63. (Cited on pages 11, 42, and 185.)

[36] B. Escoffier, M. Milanič, and V. Paschos, *Simple and Fast Reoptimizations for the Steiner Tree Problem*, Algorithmic Operations Research, 4 (2009), pp. 86–94. (Cited on page 128.)

[37] M. R. Garey and D. S. Johnson, *The Rectilinear Steiner Tree Problem is NP-Complete*, SIAM Journal on Applied Mathematics, 32 (1977), pp. 826–834. (Cited on pages 33 and 36.)

[38] N. Garg and J. Könemann, *Faster and Simpler Algorithms for Multicommodity Flow and Other Fractional Packing Problems*, in Proceedings of the 39th Annual Symposium on Foundations of Computer Science, FOCS '98, IEEE Computer Society, 1998, pp. 300–309. (Cited on page 35.)

[39] ———, *Faster and Simpler Algorithms for Multicommodity Flow and Other Fractional Packing Problems*, SIAM Journal on Computing, 37 (2007), pp. 630–652. (Cited on page 35.)

[40] M. Gester, *VLSI Routing for Advanced Technology*, PhD thesis, University of Bonn, 2015. (Cited on pages 16 and 25.)

[41] M. Gester, D. Müller, T. Nieberg, C. Panten, C. Schulte, and J. Vygen, *BonnRoute: Algorithms and Data Structures for Fast and Good VLSI Routing*, ACM Transactions on Design Automation of Electronic Systems, 18 (2013), pp. 32:1–32:24. (Cited on pages 7, 25, and 88.)

[42] C. Gottschalk, *Berechnung Optimaler Global Routing Graphen*, Bachelor's thesis (in German), University of Bonn, 2010. (Cited on page 17.)

[43] K. Goyal and T. Mömke, *Robust Reoptimization of Steiner Trees*, in LIPIcs 45, H. Prahladh and G. Ramalingam, eds., Schloss Dagstuhl — Leibniz-Zentrum für Informatik, 2015, pp. 10–24. (Cited on page 128.)

[44] R. Gupta, B. Tutuianu, and L. Pileggi, *The Elmore Delay as a Bound for RC Trees with Generalized Input Signals*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 16 (1997), pp. 95–104. (Cited on page 45.)

[45] N. Hähnle and P. Saccardi, *Global Routing with Exact Pin Positions*, tech. rep., University of Bonn, 2016. (Cited on pages 92, 93, and 135.)

[46] M. Hanan, *On Steiner's Problem with Rectilinear Distance*, SIAM Journal on Applied Mathematics, 14 (1966), pp. 255–265. (Cited on pages 33, 36, 65, and 149.)

[47] P. E. Hart, N. J. Nilsson, and B. Raphael, *A Formal Basis for the Heuristic Determination of Minimum Cost Paths*, IEEE Transactions on Systems Science and Cybernetics, 4 (1968), pp. 100–107. (Cited on page 142.)

[48] S. Hassoun and T. Sasao, eds., *Logic Synthesis and Verification*, Kluwer Academic Publishers, Norwell, USA, 2002. (Cited on page 8.)

[49] K. Heeger, *Congestion-Aware Steiner Trees with Small Elmore Delay*, Master's thesis, University of Bonn, 2018. (Cited on pages 83 and 86.)

[50] S. Held, *Timing Closure in Chip Design*, PhD thesis, University of Bonn, 2008. (Cited on page 13.)

[51] S. Held and J. Hu, *Gate Sizing*, in Electronic Design Automation for IC Implementation, Circuit Design, and Process Technology, L. Lavagno, I. Markov, G. Martin, and L. Scheffer, eds., CRC Press, 2016, pp. 245–260. (Cited on page 12.)

[52] S. Held, B. Korte, D. Rautenbach, and J. Vygen, *Combinatorial Optimization in VLSI Design*, in Combinatorial Optimization: Methods and Applications, V. Chvátal, ed., IOS Press, Amsterdam, 2011, pp. 33–96. (Cited on page 7.)

[53] S. Held, D. Müller, D. Rotter, R. Scheifele, V. Traub, and J. Vygen, *Global Routing with Timing Constraints*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 37 (2018), pp. 406–419. (Cited on pages 47, 51, 59, 63, 66, 76, 92, and 185.)

[54] S. Held, D. Müller, D. Rotter, V. Traub, and J. Vygen, *Global Routing with Inherent Static Timing Constraints*, in Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, ICCAD '15, IEEE Press, 2015, pp. 102–109. (Cited on page 47.)

[55] S. Held and S. T. Spirkl, *Binary Adder Circuits of Asymptotically Minimum Depth, Linear Size, and Fan-Out Two*, ACM Transactions on Algorithms, 14 (2018), pp. 4:1–4:18. (Cited on page 13.)

[56] D. Henke, *Pfadsuche im Detailed Routing*, Bachelor's thesis (in German), University of Bonn, 2016. (Cited on page 69.)

[57] X. Hong, T. Xue, J. Huang, C.-K. Cheng, and E. S. Kuh, *TIGER: An Efficient Timing-Driven Global Router for Gate Array and Standard Cell Layout Design*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 16 (1997), pp. 1323–1331. (Cited on page 47.)

[58] S. Hougardy, T. Nieberg, and J. Schneider, *BonnCell: Automatic Layout of Leaf Cells*, in 18th Asia and South Pacific Design Automation Conference (ASP-DAC), 2013, pp. 453–460. (Cited on page 9.)

[59] S. Hougardy, J. Silvanus, and J. Vygen, *Dijkstra Meets Steiner: A Fast Exact Goal-Oriented Steiner Tree Algorithm*, Mathematical Programming Computation, 9 (2017), pp. 135–202. (Cited on page 83.)

[60] J. Hu, Z. Li, and S. Hu, *Buffer Insertion Basics*, in Handbook of Algorithms for Physical Design Automation, C. J. Alpert, D. P. Mehta, and S. S. Sapatnekar, eds., Auerbach Publications, Boston, USA, 2008, pp. 535–556. (Cited on page 12.)

[61] J. Hu and S. S. Sapatnekar, *A Survey on Multi-Net Global Routing for Integrated Circuits*, Integration, the VLSI Journal, 31 (2001), pp. 1–49. (Cited on page 34.)

[62] ———, *A Timing-Constrained Simultaneous Global Routing Algorithm*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 21 (2002), pp. 1025–1036. (Cited on page 48.)

[63] S. Hu, Z. Li, and C. J. Alpert, *A Polynomial Time Approximation Scheme for Timing Constrained Minimum Cost Layer Assignment*, in 2008 IEEE/ACM International Conference on Computer-Aided Design, 2008, pp. 112–115. (Cited on page 22.)

[64] J. Huang, X.-L. Hong, C.-K. Cheng, and E. S. Kuh, *An Efficient Timing-Driven Global Routing Algorithm*, in 30th ACM/IEEE Design Automation Conference, 1993, pp. 596–600. (Cited on page 47.)

[65] F. Hwang, *On Steiner Minimal Trees with Rectilinear Distance*, SIAM Journal on Applied Mathematics, 30 (1976), pp. 104–114. (Cited on pages 36 and 37.)

[66] *ISPD 2007 Global Routing Contest*. http://www.ispd.cc/contests/07/contest.html, 2007. (Cited on pages 20 and 34.)

[67] *ISPD 2008 Global Routing Contest*. http://www.ispd.cc/contests/08/ispd08rc.html, 2008. (Cited on pages 20 and 34.)

[68] T. Kadodi, *Steiner Routing Based on Elmore Delay Model for Minimizing Maximum Propagation Delay*, Master's thesis, Japan Advanced Institute of Science and Technology, 1999. (Cited on page 65.)

[69] A. Kahng and G. Robins, *On Optimal Interconnections for VLSI*, Kluwer Academic Publishers, Boston, 1995. (Cited on page 65.)

[70] R. Karp, *Reducibility Among Combinatorial Problems*, in Complexity of Computer Computations, R. Miller and J. Thatcher, eds., Plenum Press, New York, 1972, pp. 85–103. (Cited on pages 35, 66, and 112.)

[71] A. K. Kiefner, *Minimizing Path Lengths in Rectilinear Steiner Minimum Trees with Fixed Topology*, Operations Research Letters, 44 (2016), pp. 835 – 838. (Cited on pages 94, 95, and 96.)

[72] N. Klewinghaus, *Efficient Detailed Routing*, Diploma's thesis, University of Bonn, 2013. (Cited on page 25.)

[73] B. Korte, D. Rautenbach, and J. Vygen, *BonnTools: Mathematical Innovation for Layout and Timing Closure of Systems on a Chip*, in Proceedings of the IEEE 95, 2007, pp. 555–572. (Cited on page 7.)

[74] B. Korte and J. Vygen, *Combinatorial Problems in Chip Design*, in Building Bridges: Between Mathematics and Computer Science, M. Grötschel, G. O. H. Katona, and G. Sági, eds., Springer Berlin Heidelberg, 2008, pp. 333–368. (Cited on page 7.)

[75] ———, *Combinatorial Optimization: Theory and Algorithms*, Springer, 6th ed., 2018. (Cited on pages 7, 35, and 65.)

[76] M. Kramer and J. van Leeuwen, *Wire-Routing is NP-Complete*, tech. rep., University of Utrecht, 1982. (Cited on page 15.)

[77] ———, *The Complexity of Wirerouting and Finding Minimum Area Layouts for Arbitrary VLSI Circuits*, Advances in Computing Research, 2 (1984), pp. 129–146. (Cited on page 15.)

[78] T. Lee and T. Wang, *Congestion-Constrained Layer Assignment for Via Minimization in Global Routing*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 27 (2008), pp. 1643–1656. (Cited on page 92.)

[79] T.-H. Lee and T.-C. Wang, *Robust Layer Assignment for Via Optimization in Multi-Layer Global Routing*, in Proceedings of the 2009 International Symposium on Physical Design, ISPD '09, ACM, 2009, pp. 159–166. (Cited on page 92.)

[80] F. Michaelis, *Capacity Estimation in Global Routing*, Master's thesis, University of Bonn, 2017. (Cited on page 105.)

[81] M. D. Moffitt, *MaizeRouter: Engineering an Effective Global Router*, in Proceedings of the 2008 Asia and South Pacific Design Automation Conference, IEEE Computer Society Press, 2008, pp. 226–231. (Cited on page 34.)

[82] ———, *Global Routing Revisited*, in Proceedings of the 2009 International Conference on Computer-Aided Design, ACM, 2009, pp. 805–808. (Cited on page 34.)

[83] M. D. Moffitt, J. A. Roy, and I. L. Markov, *The Coming of Age of (Academic) Global Routing*, in Proceedings of the 2008 International Symposium on Physical Design, ACM, 2008, pp. 148–155. (Cited on page 34.)

[84] D. Müller, *Bestimmung der Verdrahtungskapazitäten im Global Routing von VLSI-Chips*, Diploma's thesis (in German), University of Bonn, 2002. (Cited on page 20.)

[85] D. Müller, *Optimizing Yield in Global Routing*, in 2006 IEEE/ACM International Conference on Computer-Aided Design, 2006, pp. 480–486. (Cited on pages 35 and 51.)

[86] D. Müller, *Fast Resource Sharing in VLSI Design*, PhD thesis, University of Bonn, 2009. (Cited on pages 35, 51, and 82.)

[87] D. Müller, K. Radke, and J. Vygen, *Faster Min-Max Resource Sharing in Theory and Practice*, Mathematical Programming Computation, 3 (2011), pp. 1–35. (Cited on pages 21, 32, 33, 35, 47, 48, 49, 50, 51, and 58.)

[88] M. Neuwohner, *Trackless TrackAssignment*, Bachelor's thesis (in German), University of Bonn, 2018. (Cited on page 24.)

[89] M. M. Ozdal and M. D. F. Wong, *Archer: A History-Based Global Routing Algorithm*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 28 (2009), pp. 528–540. (Cited on page 34.)

[90] S. Peyer, *Elmore-Delay-Optimale Steinerbäume im VLSI-Design*, Diploma's thesis (in German), University of Bonn, 2000. (Cited on pages 45 and 65.)

[91] S. Peyer, M. Zachariasen, and D. G. Jørgensen, *Delay-Related Secondary Objectives for Rectilinear Steiner Minimum Trees*, Discrete Applied Mathematics, 136 (2004), pp. 271–298. (Cited on page 65.)

[92] P. Raghavan and C. D. Thompson, *Randomized Rounding: A Technique for Provably Good Algorithms and Algorithmic Proofs*, Combinatorica, 7 (1987), pp. 365–374. (Cited on pages 34 and 50.)

[93] ———, *Multiterminal Global Routing: A Deterministic Approximation Scheme*, Algorithmica, 6 (1991), pp. 73–82. (Cited on page 34.)

[94] S. B. Rao and W. D. Smith, *Approximating Geometrical Graphs via "Spanners" and "Banyans"*, in Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing, New York, 1998, ACM, pp. 540–550. (Cited on page 36.)

[95] C. L. Ratzlaff and L. T. Pillage, *RICE: Rapid Interconnect Circuit Evaluation Using AWE*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 13 (1994), pp. 763–776. (Cited on pages 119, 179, and 181.)

[96] R. Raz and S. Safra, *A Sub-Constant Error-Probability Low-Degree Test, and a Sub-Constant Error-Probability PCP Characterization of NP*, in Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing, STOC '97, ACM, 1997, pp. 475–484. (Cited on page 76.)

[97] D. Rotter, *Timing-Constrained Global Routing with Buffered Steiner Trees*, PhD thesis, University of Bonn, 2017. (Cited on pages 12, 47, 64, 65, 66, 67, 68, 111, and 185.)

[98] J. A. Roy and I. L. Markov, *High-Performance Routing at the Nanometer Scale*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 27 (2008), pp. 1066–1077. (Cited on page 34.)

[99] J. Rubinstein, P. Penfield, and M. A. Horowitz, *Signal Delay in RC Tree Networks*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2 (1983), pp. 202–211. (Cited on pages 11, 42, 43, and 185.)

[100] P. Saccardi, *Global Routing with Exact Pin Positions*, Master's thesis, University of Bonn, 2015. (Cited on pages 92, 93, and 135.)

[101] J. S. Salowe, *Rip-Up and Reroute*, in Handbook of Algorithms for Physical Design Automation, C. J. Alpert, D. P. Mehta, and S. S. Sapatnekar, eds., Auerbach Publications, Boston, USA, 2008, pp. 615–626. (Cited on page 24.)

[102] R. Samanta, A. I. Erzin, S. Raha, Y. V. Shamardin, I. I. Takhonov, and V. V. Zalyubovskiy, *A Provably Tight Delay-Driven Concurrently Congestion Mitigating Global Routing Algorithm*, Applied Mathematics and Computation, 255 (2015), pp. 92–104. (Cited on page 47.)

[103] D. Sankoff and P. Rousseau, *Locating the Vertices of a Steiner Tree in an Arbitrary Metric Space*, Mathematical Programming, 9 (1975), pp. 240–246. (Cited on page 95.)

[104] S. Sapatnekar, *Timing*, Springer, Berlin, Heidelberg, 2004. (Cited on pages 39 and 42.)

[105] P. Saxena, R. S. Shelar, and S. Sapatnekar, *Routing Congestion in VLSI Circuits: Estimation and Optimization*, Springer, 2007. (Cited on page 20.)

[106] R. Scheifele, *Steiner Trees with Bounded Elmore Delay*, Master's thesis, University of Bonn, 2013. (Cited on pages 65, 76, and 185.)

[107] ———, *RC-Aware Global Routing*, in Proceedings of the 35th International Conference on Computer-Aided Design, ICCAD '16, ACM, 2016, pp. 21:1–21:8. (Cited on pages 47, 63, 66, and 76.)

[108] ———, *Steiner Trees with Bounded RC-Delay*, Algorithmica, 78 (2017), pp. 86–109. (Cited on pages 42, 63, 65, 76, 78, 79, 80, and 186.)

[109] J. Schneider, *Transistor-Level Layout of Integrated Circuits*, PhD thesis, University of Bonn, 2014. (Cited on page 9.)

[110] U. Schorr, *Algorithms for Circuit Sizing in VLSI Design*, PhD thesis, University of Bonn, 2015. (Cited on pages 12 and 13.)

[111] C. Schulte, *Design Rules in VLSI Routing*, PhD thesis, University of Bonn, 2012. (Cited on pages 16 and 25.)

[112] W. Schwärzler, *On the Complexity of the Planar Edge-Disjoint Paths Problem with Terminals on the Outer Boundary*, Combinatorica, 29 (2009), pp. 121–126. (Cited on page 33.)

[113] E. Shragowitz and S. Keel, *A Global Router Based on a Multicommodity Flow Model*, Integr. VLSI J., 5 (1987), pp. 3–16. (Cited on page 34.)

[114] M. Struzyna, *Flow-Based Partitioning and Fast Global Placement in Chip Design*, PhD thesis, University of Bonn, 2010. (Cited on page 11.)

[115] J. Uyemura, *Introduction to VLSI Circuits and Systems*, Wiley, 2002. (Cited on page 7.)

[116] A. Vittal and M. Marek-Sadowska, *Minimal Delay Interconnect Design Using Alphabetic Trees*, in Proceedings of the 31st Annual Design Automation Conference, ACM, 1994, pp. 392–396. (Cited on page 65.)

[117] J. Vygen, *Near-Optimum Global Routing with Coupling, Delay Bounds, and Power Consumption*, in Integer Programming and Combinatorial Optimization: 10th International IPCO Conference, New York, USA, June 7-11, 2004. Proceedings, D. Bienstock and G. Nemhauser, eds., Springer, Berlin, Heidelberg, 2004, pp. 308–324. (Cited on pages 21, 35, 47, and 51.)

[118] J. Vygen, *Chip Design*. Lecture Notes, 2016. (Cited on page 33.)

[119] Y. Wei, Z. Li, C. Sze, S. Hu, C. J. Alpert, and S. S. Sapatnekar, *CATALYST: Planning Layer Directives for Effective Design Closure*, in Proceedings of the Conference on Design, Automation and Test in Europe, 2013, pp. 1873–1878. (Cited on pages 21 and 22.)

[120] Y. Wei, C. Sze, N. Viswanathan, Z. Li, C. J. Alpert, L. Reddy, A. D. Huber, G. E. Tellez, D. Keller, and S. S. Sapatnekar, *Techniques for Scalable and Effective Routability Evaluation*, ACM Trans. Des. Autom. Electron. Syst., 19 (2014), pp. 17:1–17:37. (Cited on pages 20, 141, and 182.)

[121] J. Werber, *Logic Restructuring for Timing Optimization in VLSI Design*, PhD thesis, University of Bonn, 2007. (Cited on page 13.)

[122] A. Williams, *C++ Concurrency in Action: Practical Multithreading*, Manning, 2012. (Cited on page 161.)

[123] T. Wu, A. Davoodi, and J. T. Linderoth, *GRIP: Global Routing via Integer Programming*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 30 (2011), pp. 72–84. (Cited on page 34.)

[124] Y. Xu, Y. Zhang, and C. Chu, *FastRoute 4.0: Global Router with Efficient Via Minimization*, in 2009 Asia and South Pacific Design Automation Conference, 2009, pp. 576–581. (Cited on page 34.)

[125] J.-T. Yan, Y.-H. Chen, C.-F. Lee, and M.-C. Huang, *Multilevel Timing-Constrained Full-Chip Routing in Hierarchical Quad-Grid Model*, in 2006 IEEE International Symposium on Circuits and Systems, 2006, pp. 5439 – 5442. (Cited on page 48.)

[126] J.-T. Yan and S.-H. Lin, *Timing-Constrained Congestion-Driven Global Routing*, in Proceedings of the 2004 Asia and South Pacific Design Automation Conference, 2004, pp. 683–686. (Cited on page 48.)

[127] A. Zych and D. Bilò, *New Reoptimization Techniques Applied to Steiner Tree Problem*, Electronic Notes in Discrete Mathematics, 37 (2011), pp. 387 – 392. (Cited on page 128.)