Dynamic Local Usage in BonnRouteGlobal

Dissertation

zur

Erlangung des Doktorgrades (Dr. rer. nat.)

der

Mathematisch-Naturwissenschaftlichen Fakultät

der

Rheinischen Friedrich-Wilhelms-Universität Bonn

vorgelegt von

Tilmann Bihler

aus

Bonn

Bonn, Mai 2023

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:	23. Juni 2023
Erscheinungsjahr:	2023

ACKNOWLEDGEMENTS

I would like to express my gratitude to all those who made this dissertation possible. First and foremost, I would like to thank my advisors Professor Dr. Jens Vygen and Professor Dr. Stephan Held, who were always supportive, brought up new ideas, and provided valuable feedback. I would also like to thank Professor Dr. Dr. h.c. Bernhard Korte for the great working environment at the Research Institute for Discrete Mathematics.

My thanks go to my current and former colleagues in the global routing team: to Dirk Müller, Daniel Rotter, and Rudolf Scheifele for introducing me to BonnRouteGlobal; Pietro Saccardi for many fruitful discussions on our related projects; and Daniel Blankenburg for handling most of the daily work on BonnRouteGlobal during the final periods of writing this thesis. I wish the best to my most recent colleague and successor, Antonia Ellerbrock. I would also like to thank the members of the other projects for their assistance and expertise. My thanks go to all the people of the institute and the Arithmeum for the great atmosphere and the good times we have had together.

Over the years, I have had the pleasure to supervise several students. I would like to thank Lukas Schembecker, Nicholas Schwab, Sebastian Lüderssen, Alexandra Niemann, Maximilian Kessler, and Adrian Glubrecht for the pleasant work times and for all that we learned from each other.

Furthermore, my thanks go to the people at IBM, especially Alex Suess and Christian Roth, for the close collaboration.

Finally, I'd like to thank family and friends, among whom are my friends in climbing and dancing, providing a fulfilling balance to the work on this thesis, and my parents, Ursula and Roland Bihler, for their support throughout my university studies and before.

CONTENTS

1 INTRODUCTION 1
2 PRELIMINARIES 5
- 2.1 Notation 5
- 2.2 The Global Routing Problem 11
- 2.3 The Resource Sharing Problem 12
- 2.4 Global Routing as a Resource Sharing Problem 13

3 DYNAMIC LOCAL USAGE 15
- 3.1 Usage Model 16
  - 3.1.1 Controlling Packing Density 20
  - 3.1.2 Estimating Signal Delay 28
- 3.2 Computing Coarse Routes 28
- 3.3 Computing Local Routes 34
  - 3.3.1 Optimizing x- and y-Coordinates 34
  - 3.3.2 Optimizing Layers and Wire Types 61
  - 3.3.3 Simultaneous Optimization 71
- 3.4 Incremental Routing 73

4 DYNAMIC LOCAL USAGE IN PRACTICE 79
- 4.1 Testbed 79
- 4.2 Metrics 79
- 4.3 Timing-Unaware 81
- 4.4 Large Global Routing Tiles 84
- 4.5 Timing-Aware Routing Flow 89

5 REACH- AND DIRECTION-RESTRICTED STEINER TREES 99
- 5.1 Preliminaries 99
- 5.2 Path Structure Theorem 103
  - 5.2.1 Direction-Restricted Obstacles 114
- 5.3 Algorithm 116
  - 5.3.1 Construction of a Visibility Graph 117
  - 5.3.2 Extraction of the Steiner Tree 122
  - 5.3.3 Running Time 123
  - 5.3.4 Post-Processing 124
  - 5.3.5 Multiple Nets 128
- 5.4 Experimental Results 129
  - 5.4.1 Standard Benchmarks 129
  - 5.4.2 Chip Instances 133
  - 5.4.3 Computing Lower Bounds in BonnRouteGlobal 134

6 SUMMARY 139

BIBLIOGRAPHY 141

1 INTRODUCTION

The modern world relies heavily on fast computer chips (Figure 1). The process of specifying and creating the blueprint of a chip before it goes into manufacturing is called chip design. A central task in chip design is routing. Sets of *pins* on the chip must be connected through wires without intersecting each other and meeting many design rules and optimization criteria. Each such set of pins is called a *net* and, because there can be millions of nets on one chip, routing is typically divided into two steps. Firstly, a rough global routing is computed with the ability to optimize global objectives. Secondly, a detailed routing algorithm uses this rough routing as guidance to compute the exact wiring, meeting all local design rules. The *BonnTools*, a collection of chip design software developed at the Research Institute for Discrete Mathematics in Bonn in cooperation with IBM, achieve this by *BonnRouteGlobal* and *BonnRouteDetail*.

This thesis is about improvements on BonnRouteGlobal. In Section 2, we explain the global routing problem in more detail and introduce basic notation. The primary focus of this thesis is on the dynamic local usage which is a more accurate way of measuring routing density in BonnRouteGlobal and is presented in Section 3. As a global router, BonnRouteGlobal computes longer wiring connections on a coarse grid graph that models a condensed version of the chip. Traditionally, the impact on space of shorter wires that connect to pins is estimated beforehand and independently of the actual routing topology. The dynamic local usage establishes a new optimization model in which all wiring can be optimized simultaneously, thereby improving the overall routing. The subject of this thesis is the introduction of this model, the development and implementation of algorithms necessary to compute an optimized routing within it, and enabling the dynamic local usage in the routing flow.

We discuss the impact of the dynamic local usage on the difficulty of finding good routing solutions on the coarse grid graph. Bonn-

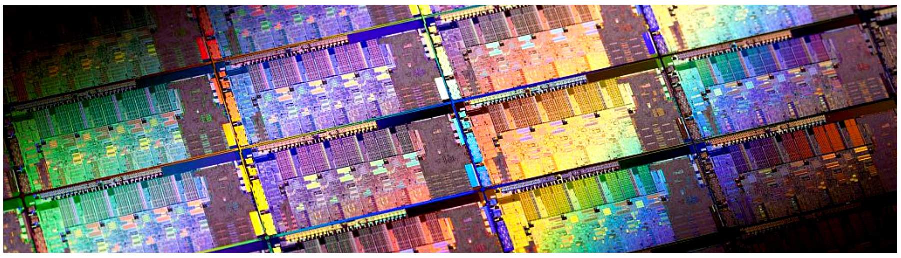


Figure 1: A silicon wafer containing several chips [17].

RouteGlobal employs the resource sharing framework from [49] to share the available routing space among the nets and to optimize signal delay through the wires. The resource sharing algorithm provides prices for the routing resources. Building on [52], using dynamic programming and arithmetics on piece-wise linear functions, we present an algorithm that computes optimum local wiring in a planar version of the dynamic local usage model with regard to these prices. To embed the wiring into the chip layers, we use a dynamic program from [53] whose running time is quadratically dependent on the number of chip layers. We present an improved version with a linear running time, that is able to optimize additional properties of the wires. We briefly outline how to combine the planar and the layer optimization into a full 3-dimensional optimization. This is followed by a discussion on how to adapt Incremental BonnRouteGlobal, which is a mode of the global router used intensively during the routing flow to improve timing, for the dynamic local usage.

In Section 4, we evaluate the resulting global routing quality in terms of routability in BonnRouteDetail and demonstrate the capability of the dynamic local usage to persistently improve the results in the routing flow of IBM, leading to improved signal delay, shorter wire length, and less design rule violations.

Section 5 presents a fast approximation algorithm for a variant of the rectilinear Steiner tree problem, that contains certain types of obstacles as additional constraints, expanding on the work of [6, 25]. Some obstacles may not be traversed at all, others may be traversed only horizontally, only vertically, or in both directions. The total length of each connected component in the intersection of the tree with the interior of the obstacles is bounded by a constant. This problem is motivated by the layout of repeater tree topologies, which is also a central task in chip design. Large obstacles might be crossed by wires on higher layers, but repeaters may not be placed within the obstacles and a long unbuffered piece of wiring would lead to timing violations. Due to special obstacle structures, the traversal can be restricted to one direction.

We present a 2-approximation algorithm with a worst-case running time of $\mathcal{O}\left((k \log k)^2\right)$, where k is the number of terminals plus the number of obstacle corner points. Under certain realistic assumptions on the obstacle structure, the running time is $\mathcal{O}\left(k(\log k)^2\right)$. The algorithm is fast in practice and it solves even large instances arising from global routing with almost 800000 terminals within 80 seconds, proving its practical applicability. Combined with very effective post-optimization we obtain better results than previous heuristics on large obstacle-avoiding DIMACS benchmarks. If the Steiner trees are allowed to reach over obstacles our algorithm finds significantly shorter solutions.

Compared to [6, 25] which this section is based on, we provide a more thorough case distinction in the proof of the main theorem, closing gaps in the previous proofs. This is joint work with the authors of [25], though the majority of the progress is due to the author of this thesis. Building on [6, 25], we also slightly improve the extraction of the Steiner tree and extend the post-optimization by an edge substitution method from [20]. The latter solves an online maximum cost on tree path problem which is also interesting for the Steiner tree problem without obstacles. For this problem we improve the preprocessing time of the algorithm by [31] from $\mathcal{O}(n \log n)$ to $\mathcal{O}(n)$, still allowing queries in $\mathcal{O}(\log n)$.

Finally, this thesis provides experimental results of a new application of our Steiner tree algorithm for the computation of lower bounds on wire length and via numbers in BonnRouteGlobal with the dynamic local usage. Results point out the algorithm's potential to generate strong lower bounds significantly faster than the currently used approach in BonnRouteGlobal that is based on the coarse global routing grid graph.

2 PRELIMINARIES

In the BonnTools [21, 34], BonnRouteGlobal computes a coarse global routing solution (Figure 2). Afterwards, BonnRouteDetail computes exact wiring that is free of overlaps and meets all local design rules (Figure 3). For this, it uses the coarse global routing solution of BonnRouteGlobal as a guidance and restricts its search area for wires around the wires of the global routing. Detailed routing is a very complex task [1] and is heavily dependent on the global routing input, which must be of good quality in terms of routability and other objectives such as total wiring length, signal delay properties and estimated power consumption.

It is possible to perform an intermediate step between global and detailed routing, called *track assignment*: Given a global routing solution, wires are moved in orthogonal routing dimension with the objective of minimizing total overlap. The detailed router tries to follow the almost legal result of track assignment [2] potentially resulting in a shorter running time and improved detailed routing quality. We will not consider track assignment in this thesis but it is compatible with the concepts that we describe here.

BonnRouteGlobal creates a coarse grid graph on the chip to simplify the instance and make global optimizations possible. To avoid congestion hotspots, i.e. high local packing densities of wires that would pose problems for BonnRouteDetail, BonnRouteGlobal employs the resource sharing framework described in [49]. Essentially, the edges of the grid graph are the resources representing the available space and the nets are the customers whose wires consume from the edge resources depending on their position, width and length. The main routing algorithm projects all pin positions to the nodes of the grid graph and only routes along its edges, trying not to overuse any edge resources. The resource sharing algorithm computes a convex combination of solutions for each of the nets on which randomized rounding is applied to obtain an integral solution, followed by a heuristic post-optimization. In the very end the routes on the coarse grid graph are connected to the exact pin shapes through local wiring.

2.1 NOTATION

We start with some basic definitions:

Definition 1 (Chip image)**.** By $\mathcal{C} = [x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$ we denote the rectangular area of the chip. Let $\mathcal{L} = [l_{\min}, ..., l_{\max}] \subset \mathbb{N}$ an interval that represents the set of chip layers. Let $\mathcal{L}_x \sqcup \mathcal{L}_y = \mathcal{L}$ the

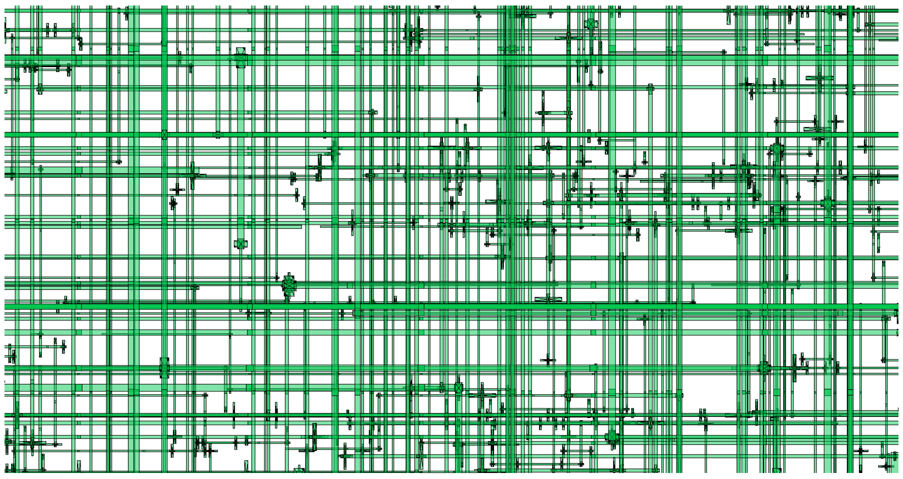


Figure 2: View from the top onto a small part of a global routed chip. The long green shapes are wires that can have different widths and are located on different layers on top of each other. The small rectangles are vias, which are metal shapes that connect neighboring layers.

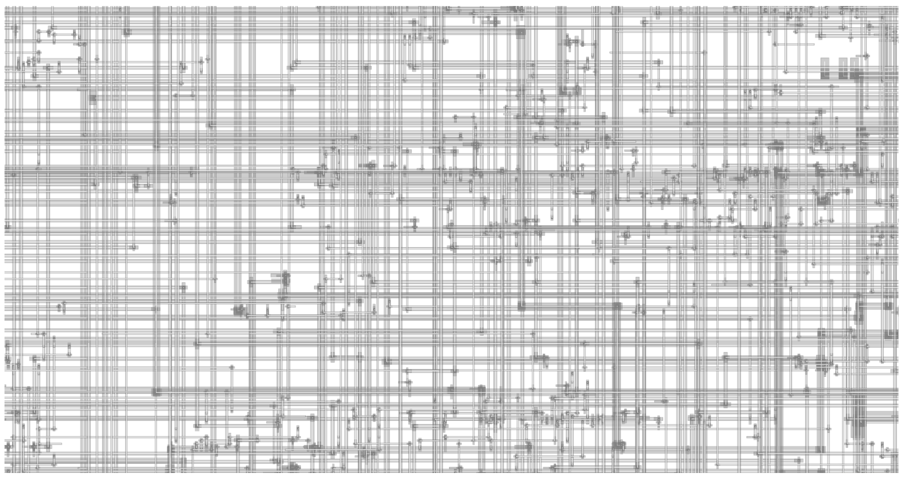


Figure 3: View from the top onto the same excerpt as in Figure 2 after detailed routing. In contrast to global routing the wires must not overlap each other on the same layer after detailed routing.

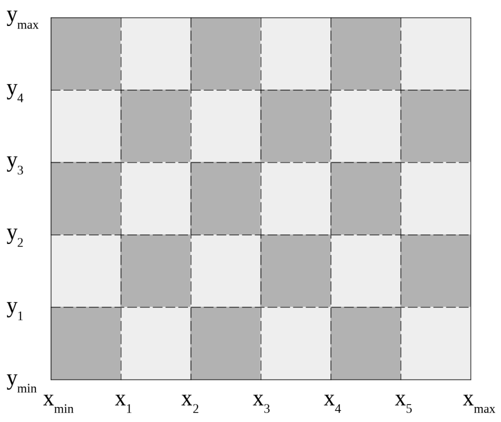


Figure 4: Division of the chip image $\mathcal{C}$ into a grid of rectangular global routing tiles (light and dark gray).

layers with preferred routing dimension x and y respectively, that is wires on x layers must only run in x-dimension and wires on y layers must only run in y-dimension.

Usually the routing dimension alternates over the layers. There may be blockages on the chip image on certain layers (Figure 10 on page 20). Blockages are rectangular areas in which BonnRouteDetail is not allowed to place wires, which has to be taken into account by BonnRouteGlobal.

BonnRouteGlobal creates the coarse global routing graph on the chip image :

Definition 2 (Global routing graph)**.** The chip image $\mathcal{C}$ is divided into a grid of n_x times n_y rectangular global routing tiles (Figure 4), whose widths and heights are defined by numbers $x_{\text{min}} < x_1 < \ldots < x_{n_x-1} < x_{\text{max}}$ and $y_{\text{min}} < y_1 < \ldots < y_{n_y-1} < y_{\text{max}}$. To each center of a global routing tile and for each layer we associate a node in the global routing graph $\mathcal{G}$ (Figure 5). That is, $\mathcal{G}$ is a three-dimensional grid graph with node set

$$V(\mathcal{G}) := \{1, \ldots, n_x\} \times \{1, \ldots, n_y\} \times \mathcal{L}$$

Nodes of neighboring global routing tiles on the same layer are connected through wire edges if those conform with the routing dimension of the layer the nodes belong to. Nodes that belong to adjacent layers and represent the same global routing tile are connected through via edges.

By $\square(v) \subset \mathcal{C} \times \mathcal{L}$ we denote the global routing tile area a node $v \in V(\mathcal{G})$ belongs to. In the same way each edge $e \in E(\mathcal{G})$ connecting two nodes on the same layer is associated with a rectangular edge area $\square(e) \subset \mathcal{C} \times \mathcal{L}$ spanning the area between the incident nodes (Figure 6).

Along with the chip image, BonnRouteGlobal is given pins and nets:

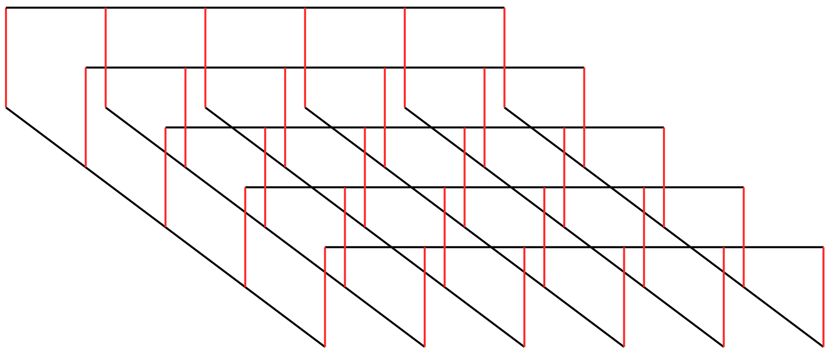


Figure 5: The global routing graph of a chip with two layers. The bottom layer has y-routing dimension and the top layer x-routing dimension. Wire edges are drawn in black and via edges in red.

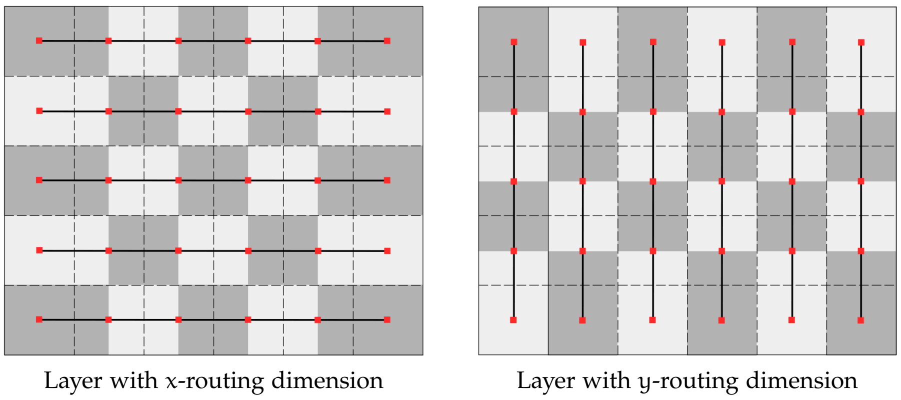

Layer with x-routing dimension
Layer with y-routing dimension

Figure 6: The edge areas (light and dark gray) associated to the wire edges (thick black), connecting the global routing graph nodes (red). Note that the edge areas are extended at the chip borders. The edge areas are offset from the global routing tiles (dashed black).

Definition 3 (Pin)**.** A pin p is a rectangular shape on the chip image on a certain layer, that is $p \subset \mathcal{C} \times \mathcal{L}$.

In practice, pins can consist of a set of rectangular shapes, but for simplicity here we assume that they only consist of one shape.

Definition 4 (Net)**.** A net n is a finite set of pins $\mathcal{P}_n$. By $\mathcal{N}$ we denote the set of all nets of the chip. Each net has a dedicated source pin. The other pins are referred to as sink pins.

The task of BonnRouteGlobal is to connect the pins in each net with a route, such that the electrical signal can flow from the source pin to the sink pins:

Definition 5 (Local route)**.** A local route for a net $n \in \mathcal{N}$ is an arborescence A with vertices $V(A) \subset \mathcal{C} \times \mathcal{L}$ connecting all pins $\mathcal{P}_n$ of n, that is for each $p \in \mathcal{P}_n$ there is $v \in V(A)$ with $v \in p$. The root of A connects the source pin of n. Nodes on different layers can be connected by so-called vias. Edges connecting nodes on the same layer must conform with the preferred routing dimension of that layer, that is for each edge $((x_1, y_1, z_1), (x_2, y_2, z_2)) \in E(A)$ it must either hold

- $y_1 = y_2$ and $z_1 = z_2 \in \mathcal{L}_x$ (wire in x-dimension),
- $x_1 = x_2$ and $z_1 = z_2 \in \mathcal{L}_y$ (wire in y-dimension) or
- $x_1 = x_2$ and $y_1 = y_2$ and $|z_1 - z_2| = 1$ (via).

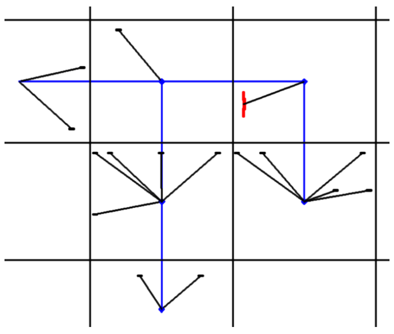
Note that there are cases where in practice we allow jogs that are wires not running in the preferred dimension of a layer. However, in this thesis we assume them to be forbidden. Multiple consecutive via edges without incident wire edges in the middle layers are called *stacked vias*.

To speed up the running time and make global optimizations possible, BonnRouteGlobal projects the pins onto the global routing graph nodes and computes *coarse routes* on the global routing graph:

Definition 6 (Coarse Route)**.** A coarse route A for a net $n \in \mathcal{N}$ has the same properties as a local route for n with the difference that

- the coarse route is a subset of the global routing graph, that is $V(A) \subset V(\mathcal{G})$ and $E(A) \subset E(\mathcal{G})$ and
- a pin $p \in \mathcal{P}_n$ is considered to be connected by A if there is a node $v \in V(A) \subset V(\mathcal{G})$ such that p intersects the global routing tile of v, that is $p \cap \square(v) \neq \emptyset$ (note that a pin can intersect several neighboring global routing tiles).

A coarse route does not need to connect to the exact pin shapes of a pin but only to the node in the global routing graph representing the global routing tile on which the pin is located. The coarse route


Coarse route on the global routing graph. Diagonal black arcs indicate projected pin positions to the centers of the global routing tiles.

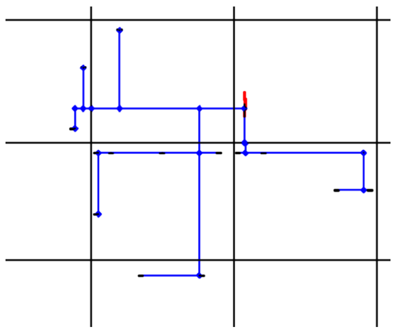

Local route connecting to the exact pin shapes.

Figure 7: A coarse route and a local route for a net, with the borders of the global routing tiles (black), edges of the routes (blue) and sink pins (black rectangles) and the source pin (red).

suffices for BonnRouteDetail to be used as a guidance to compute exact wiring for a net. For other purposes in BonnRouteGlobal like timing or wire length evaluation - and congestion estimation within the new dynamic local usage approach - it is necessary to have a local route connecting to the exact pin shapes as in Definition 5. BonnRouteGlobal computes local routes at a later time, sticking to the same global routing tiles as in the earlier computed coarse routes. Figure 7 shows a coarse route and a local route for a net.

During routing it has to be ensured that all signals arrive early enough at their destination. They must not arrive too early either, but since this can be satisfied much easier this will not be considered in this thesis. The information on how signals are propagated through the chip is given by the timing graph:

Definition 7 (Timing graph)**.** The timing graph $\mathcal{T}$ has node sets V_{in}, V_{out} and V_{gate}. V_{in} consists of the primary inputs of the chip and the output pins of latches which propagate a signal from a previous computation cycle. Analogously, V_{out} consists of the primary outputs of the chip and the input pins of latches, storing a signal for a later cycle. V_{gate} contains the input pins of gates. $\mathcal{T}$ contains an edge (v, w) iff $v \in V_{\text{in}}$ and w is a sink pin of the net driven by v, or $v \in V_{\text{gate}}$ and w is a sink pin of the net that is driven by an output pin of the gate that has v as input pin.

Given arrival times $\text{at}(v)$ for all $v \in V_{\text{in}}$ and required arrival times $\text{rat}(w)$ for $w \in V_{\text{out}}$ we want to find routes for the nets such that the following timing inequality is satisfied for all paths P in the timing graph $\mathcal{T}$ from some $v \in V_{\text{in}}$ to $w \in V_{\text{out}}$:

$$\text{at}(v) + \sum_{e \in E(P)} \text{delay}(e) \leqslant \text{rat}(w) \tag{1}$$

Here delay(e) denotes the time a signal has to travel through the edge e in the timing graph, and the inequality describes the assertion that a signal starting at v reaches w early enough.

[18] describes an easy way to approximate the signal delay through a tree network consisting of resistors and capacitors. In our application this yields the following definition:

Definition 8 (Elmore delay)**.** Given a local route A with source s and sinks $T \subset V(A)$, a source resistance $R(s) \geqslant 0$ and sink capacitances $C(w) \geqslant 0$ for all sink pins $w \in T$ and capacitances and resistances $C(e), R(e) \geqslant 0$ for all wires and vias $e \in E(A)$ we define the downstream capacitance of the sub-arborescence $A_v \subset A$ rooted at v as

$$C(A_v) := C(E(A_v)) + C(T \cap V(A_v))$$

and the Elmore delay from the source s to a sink $w \in T$ as

$$\text{delay}_A(s,w) := R(s) \cdot C(A) + \sum_{e \in P_{[s,w]}} R(e) \cdot \left(\frac{C(e)}{2} + C(A_w) \right)$$

where $P_{[s,w]}$ is the unique path from s to w in A.

The Elmore delay gives a reasonable estimate of the signal delay through a route while it is easily computable in linear time. For these reasons it is widely used in VLSI-Design and also employed by BonnRouteGlobal to estimate and optimize timing.

2.2 THE GLOBAL ROUTING PROBLEM

We formalize the task of BonnRouteGlobal: In its simplest version the global routing problem is a weighted Steiner tree packing problem. As input we are given the set of nets and the global routing graph along with capacities for the edges. The capacities are chosen such that they reflect an estimated amount of wiring that can be fit onto the edge area of a global routing edge, taking into account any preexisting wires and blockages on the chip (see e.g. [45, 48, 51, 53] for more details on capacity estimation). The simplified global routing problem asks to compute coarse routes for the nets such that the total number of coarse routes that contain an edge of the global routing graph does not exceed the capacity of that edge. The total length of all the coarse routes should be minimum [49].

Problem 9. SIMPLIFIED GLOBAL ROUTING

Input: Set of nets $\mathcal{N}$, global routing graph $\mathcal{G}$, edge capacities $u : E(\mathcal{G}) \to \mathbb{Z}_+$.

Output: A coarse route A^n for each net $n \in \mathcal{N}$ such that

$$|\{n \in \mathcal{N} : e \in E(A^n)\}| \leqslant u(e) \text{ for each edge } e \in E(G)$$

and $\sum_{n \in \mathcal{N}} \sum_{e \in E(A^n)} \text{length}(e)$ is minimum.

This simplified version assumes all wires to have the same width. In practice there can be wires with different widths and nets can be assigned weights. Moreover, further objectives like signal delay, signal integrity, power consumption and manufacturing yield can be incorporated into the problem formulation [49].

2.3 THE RESOURCE SHARING PROBLEM

BonnRouteGlobal uses the resource sharing framework to first solve the relaxed global routing problem, which asks for a convex combination of coarse routes for each net. For this it models the relaxation of the global routing problem as a resource sharing problem. An approximate solution to this problem can be computed efficiently with the resource sharing algorithm [49].

Problem 10. RESOURCE SHARING

Input: M *customers* and K *resources*, indexed by $1, \dots, M$ and $1, \dots, K$. For each customer m a finite set of possible *solutions* A_m, indexed by $1, \dots, |A_m|$. The solutions consume from the resources: Solution $j \in A_m$ for customer m consumes $w_{mjk} \in [0, \infty)$ from resource k.

Output: For each customer $m = 1, \dots, M$ a *convex combination* z_m of solutions in A_m, approximately minimizing the *maximum resource usage*

$$\lambda := \max_{k=1,\dots,K} \left(\sum_{m=1}^{M} \sum_{j \in A_m} z_{mj} \cdot w_{mjk} \right)$$

Moreover, for a fixed $\sigma \geqslant 1$ we require oracles $f_m : \mathbb{R}_+^K \to A_m$ for each customer $m = 1, \dots, M$. For a cost vector $y \in \mathbb{R}_+^K$ such an oracle function should return an element $f_m(y)$ with

$$\sum_{k=1}^{K} y_k \cdot w_{m f_m(y) k} \leqslant \sigma \cdot \text{opt}_m(y)$$

Here $\text{opt}_m(y) := \min_{j \in A_m} \sum_{k=1}^{K} y_k \cdot w_{mjk}$ denotes the cost of an optimum solution for customer m with respect to the cost vector y. The oracles are used by the resource sharing algorithm whose approximation guarantee linearly depends on σ:

Theorem 11. *[49] Consider an instance of the resource sharing problem with σ-approximate oracles and let θ denote the running time for one oracle call. Then, for every parameter $\omega > 0$, it is possible to compute a $\sigma(1+\omega)$-approximate solution to the resource sharing problem in time*

$$\mathcal{O}\left(\theta(|M|+|K|)\log|K|\left(\log\log|K|+\omega^{-2}\right)\right)$$

The running time was improved by the following result:

Theorem 12. *[8] In the same setting as in Theorem 11, one can compute a $\sigma(1+\omega)$-approximate solution to the resource sharing problem in time*

$$\mathcal{O}\left(\theta(|M|+|K|)\omega^{-2}\log|K|\right)$$

2.4 GLOBAL ROUTING AS A RESOURCE SHARING PROBLEM

Recall that we are given capacities $u : E(\mathcal{G}) \to \mathbb{Z}_+$ for the edges of the global routing graph. An instance of the relaxed global routing problem can be modeled as an instance of the resource sharing problem as follows:

For each edge $e \in E(\mathcal{G})$ in the global routing graph we add one resource r_e. For each net $n \in \mathcal{N}$ we add one customer c_n. The solution set for customer c_n corresponds to the possible coarse routes for net n. Let A be such a coarse route for net n. It consumes $\frac{1}{u(e)}$ from resource r_e if $e \in E(A)$. Otherwise, if $e \notin A$ it consumes nothing from r_e. By this definition a fractional assignment of solutions to customers with a maximum resource usage of at most 1 implies a fractional assignment of coarse routes to the nets such that no capacity constraint of the global routing graph is violated. The whole set of coarse routes is not computed explicitly. Only a polynomial number of coarse routes that are assigned a value greater zero in the output are actually computed. A regular Steiner tree approximation algorithm can be chosen as an oracle for the computation of the coarse routes.

Other objectives like total netlength can be modeled by additional resources. BonnRouteGlobal also optimizes arrival times in the timing graph during global routing within the resource sharing algorithm: To each edge of the timing graph $\mathcal{T}$ it adds a *sink delay* resource, and to each node of the timing graph it adds an *arrival time* customer representing the arrival time of the signal at a specific node in the timing graph. The arrival time customer of a node v consumes from the sink delay resources of the incident edges $\delta_{\mathcal{T}}(v)$. The later the arrival time the more it consumes from the outgoing sink delay resources but less from the incoming sink delay resource. For each source-sink path a net consumes from the corresponding sink delay resource an amount of usage that is proportional to the delay on that path from the source to the sink. Choosing appropriate weights for the usages, a resource sharing solution with a maximum resource usage of at most 1 implies a fractional global routing solution with delays and arrival times that satisfy the timing assertions (1). Such a solution usually is much better than working without arrival time customers and using static arrival times instead, because they do not adapt to the actual delays of the computed routes. For details we refer to [27]. Using more sophisticated oracles, one can compute coarse routes that are approximately optimum with regard to sink delay and edge resource usages [53] (see also Section 3.2).

Given a fractional solution to the relaxed global routing problem, BonnRouteGlobal applies randomized rounding to obtain an integral solution (see [7] for more details). [49] shows that the rounded solution is not much worse than the fractional solution. After rounding BonnRouteGlobal optimizes the rounded solution heuristically.

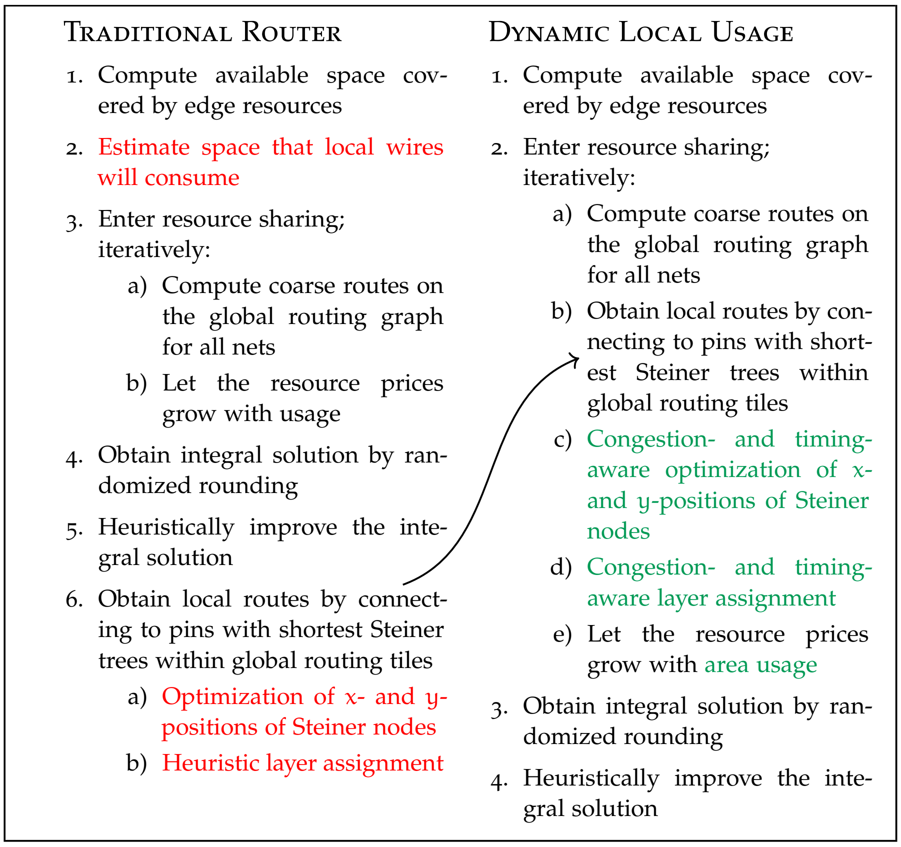
3 DYNAMIC LOCAL USAGE

The quality of a global routing is determined by the ability of the detailed router to connect the nets through overlap-free wires following the given global routing. BonnRouteGlobal computes coarse routes within the resource sharing framework, making sure that the precomputed capacity of each global routing graph edge is not exceeded. After applying randomized rounding and heuristic post-processing, BonnRouteGlobal obtains local routes by connecting the coarse route of each net to the exact pin shapes through shortest Steiner trees within the global routing tiles (Figure 7 on page 10). Hence, looking at the overall flow of BonnRouteGlobal (Figure 8), local routes are only computed at the very end.

The space used by wires connecting to the exact pin shapes has to be accounted for during the resource sharing algorithm: Heuristic algorithms pre-estimate the usage of this local wiring and additional space that BonnRouteDetail might need before the resource sharing starts. However, due to the static nature of the local wiring pre-estimates they can never represent the actual usage that the local wiring will eventually have, neither can it be optimized during resource sharing. During incremental routing, which is intensively used in the IBM routing flow to incorporate smaller changes on few nets after global routing, the pre-estimates cannot account for changes in the chip structure like moved pins. Moreover the pre-estimates make use of many parameters which partly have to be re-tuned for new chip technologies and lead to inefficient packing of wires.

The goal of the dynamic local usage is to overcome these drawbacks by modeling the local wiring usage and the packing ability of BonnRouteDetail in a simpler way. The computation of the local wiring is moved from after the randomized rounding into the resource sharing phases right after the computation of the coarse route on the global routing graph. This enables the resource sharing algorithm to consider the actual usages of the local wiring and the heuristic pre-estimates are thus not needed any longer. Using dynamic programming, the local wiring can adapt optimally to the current edge resource usages and also to other objectives like timing over the resource sharing phases. Altogether, a near-optimum routing also with regard to local wiring can be computed during the resource sharing, improving the overall quality of the output of BonnRouteGlobal. During incremental routing the local wiring usages can adapt to changing net topology.

To implement the dynamic local usage in BonnRouteGlobal, several key components and algorithms had to be modified. By the *traditional* *router* we denote BonnRouteGlobal without the dynamic local usage. Figure 8 shows the main steps of BonnRouteGlobal with and without the dynamic local usage.



Traditional Router

1. Compute available space covered by edge resources
2. Estimate space that local wires will consume
3. Enter resource sharing; iteratively:
   a) Compute coarse routes on the global routing graph for all nets
   b) Let the resource prices grow with usage
4. Obtain integral solution by randomized rounding
5. Heuristically improve the integral solution
6. Obtain local routes by connecting to pins with shortest Steiner trees within global routing tiles
   a) Optimization of x- and y-positions of Steiner nodes
   b) Heuristic layer assignment

Dynamic Local Usage

1. Compute available space covered by edge resources
2. Enter resource sharing; iteratively:
   a) Compute coarse routes on the global routing graph for all nets
   b) Obtain local routes by connecting to pins with shortest Steiner trees within global routing tiles
   c) Congestion- and timing-aware optimization of x- and y-positions of Steiner nodes
   d) Congestion- and timing-aware layer assignment
   e) Let the resource prices grow with area usage
3. Obtain integral solution by randomized rounding
4. Heuristically improve the integral solution

Figure 8: Main steps of the traditional router and of the dynamic local usage.

3.1 USAGE MODEL

To account for the space that wires consume, they add usage to the edge resources of the global routing graph within the resource sharing framework. With the traditional router, wires of a coarse route connecting nodes in the global routing graph add usage proportional to their width. The width is the sum of the metal width of the wire and the required spacing to neighboring wires.

To measure the edge resource usages for a local route, the traditional router filters all wires that cross global routing tile borders and also accounts usage proportional to the width of the wires, but regardless of their length. In this way, the edge resource usages do not change when transforming a coarse route to a local route sticking to the global routing tiles of the coarse route. The static pre-estimate usage covers for the wires inside the global routing tiles that connect to the exact pin shapes and does not depend on the actual routes.

With the dynamic local usage all wires of a local route consume space from the edge resources such that their effect on packing density can be measured. The usage of a wire is proportional to its area and it is distributed among the edge resources whose edge areas the wire intersects. This definition holds for all wires, both wires that cross global routing tile borders and wires that are located inside the global routing tiles. There is no pre-estimate usage with the dynamic local usage.

To prevent continuous price changes on global routing tile borders in orthogonal routing direction, instead of actual areas, we define sticks for wires and vias which are lines in preferred routing dimension from the minimum to the maximum expansion of the wire or via. We will intersect the sticks with the global edge areas and let them consume usage from the respective edge resources proportional to the length of the intersection times the width of the wire or via. We consider spacing requirements in preferred routing direction, the so-called tip-to-tip spacing, by which the sticks will be extended. For simplicity, we assume all wires to have unit width in orthogonal direction (including spacing constraints) in most parts of this thesis including this section. Only in Section 3.3.2 can wires have different widths, leading to different properties regarding signal propagation and space consumption. The implementation in BonnRouteGlobal is also capable of handling wires with different widths.

As BonnRouteGlobal computes routes in a two-step approach by first computing a coarse route for a net, which is then extended to a local route, we define the following definitions both for local and coarse routes.

Definition 13 (Wire and via sticks)**.** Let

$$\mathrm{dist}_{\mathrm{tip}}(z) \in \mathbb{R}_{\geqslant 0}$$

the required tip-to-tip spacing on layer $z \in \mathcal{L}$ given by the manufacturing rules and $k_{\mathrm{tip}} \geqslant 1$ constant. Let A be a local or coarse route and $v = (x, y, z) \in V(A)$. By $\mathbb{1}^+(v)$ we denote the indicator function that is true iff v is not incident to an edge in positive routing dimension. Analogously, we define $\mathbb{1}^-(v)$ to be true iff v is not incident to an edge in negative routing dimension. We define tip-to-tip lengths as

$$l_{\mathrm{tip}}(z) := k_{\mathrm{tip}} \cdot \frac{\mathrm{dist}_{\mathrm{tip}}(z)}{2}$$

$$l^-_{\mathrm{tip}}(v) := \mathbb{1}^-(v) \cdot l_{\mathrm{tip}}(z) \qquad l^+_{\mathrm{tip}}(v) := \mathbb{1}^+(v) \cdot l_{\mathrm{tip}}(z)$$

For a wire in x-dimension $e = (v, w) = ((x_1, y, z), (x_2, y, z)) \in E(A)$ where w.l.o.g. $x_1 \leqslant x_2$ we define

$$\mathrm{stick}(e) := [x_1 - l^-_{\mathrm{tip}}(v), x_2 + l^+_{\mathrm{tip}}(w)] \times \{y\} \times \{z\}$$

For a wire in y-dimension $e = (v, w) = ((x, y_1, z), (x, y_2, z)) \in E(A)$ where w.l.o.g. $y_1 \leqslant y_2$ we define

$$\text{stick}(e) := \{x\} \times [y_1 - l^-_{\text{tip}}(v), y_2 + l^+_{\text{tip}}(w)] \times \{z\}$$

For a wire in z-dimension $e = ((x, y, z_1), (x, y, z_2)) \in E(A)$ we define the stick as the empty set

$$\text{stick}(e) := \emptyset$$

For a node $v = (x, y, z) \in V(A)$ that is incident to a via edge we define

$$\text{stick}(v) := \begin{cases} [x - 0.5 - l^-_{\text{tip}}(v), x + 0.5 + l^+_{\text{tip}}(v)] \times \{y\} \times \{z\} & z \in \mathcal{L}_x \\ \{x\} \times [y - 0.5 - l^-_{\text{tip}}(v), y + 0.5 + l^+_{\text{tip}}(v)] \times \{z\} & z \in \mathcal{L}_y \end{cases}$$

For all other nodes $v = (x, y, z) \in V(A)$ we define their stick as the empty set

$$\text{stick}(v) := \emptyset$$

In other words, wire edges are geometrically embedded into $\mathcal{C} \times \mathcal{L}$ and extended by the tip-to-tip penalty. For vias we associate two lines of length 1 plus the tip-to-tip penalty in preferred routing dimension around the two end points of the via edge. The tip-to-tip penalty extension is only applied if a node is not incident to an edge that would overlap with the tip-to-tip extension (Figure 9). The tip-to-tip length is given by the tip-to-tip spacing divided by two and multiplied by k_{tip}. Setting $k_{\text{tip}} = 1$ ensures that any two wires or vias whose associated areas do not intersect have a distance of at least $\text{dist}_{\text{tip}}(z)$ in preferred routing dimension. In Section 3.1.1 we will elaborate more on the choice of k_{tip}.

The resource sharing framework provides prices for the edge resources, which translate into prices for the global routing graph edge areas:

Definition 14 (Congestion price function)**.** A function

$$\psi : \mathcal{C} \times \mathcal{L} \to \mathbb{R}_{\geqslant 0}$$

that is constant on each global edge area is called a congestion price function.

We use this general definition of ψ to make notation easier. However, keep in mind that for $e \in E(\mathcal{G})$ the restriction $\psi_{|\square(e)}$ of the congestion price function to one global edge area is a constant function. For $v \in V(\mathcal{G})$ the restriction $\psi_{|\square(v)}$ of the congestion price function to the area of a global routing tile on one layer yields a function assuming at most two different values because one global routing tile intersects exactly two global edge areas except it is located next to the chip boundary.

Now we can define the congestion costs of a route:

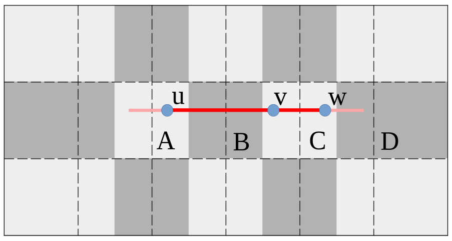


Figure 9: The wire sticks (dark red) from u to v and v to w intersect the edge areas A, B and C (light and dark gray). At node v no tip-to-tip penalty is added because it would overlap with the incident wires. At nodes u and w tip-to-tip penalty extensions are applied (light red) and they intersect the edge areas A, C and D.

Definition 15 (Congestion costs of a route)**.** Let A a local or coarse route and $\psi : \mathcal{C} \times \mathcal{L} \to \mathbb{R}_{\geqslant 0}$ a congestion price function. The congestion costs of A are defined as

$$\mathrm{cost}_{\mathrm{cong}}(A) := \int_{\bigcup_{e \in E(A)} \mathrm{stick}(e) \cup \bigcup_{v \in V(A)} \mathrm{stick}(v)} \psi \, ds$$

In other words, we consider the union of all wire and via sticks, intersect it with the global edge areas and multiply the length of each intersection by the respective price (see Figure 9).

At some places we use the following easier-to-handle definition which ignores overlaps of wire sticks (which happen at Steiner points) and thus can slightly deviate from Definition 15:

Definition 16 (Intersection-ignoring congestion costs of a route)**.** Let A a local or coarse route and $\psi : \mathcal{C} \times \mathcal{L} \to \mathbb{R}_{\geqslant 0}$ a congestion price function. The intersection-ignoring congestion costs of A are defined as

$$\mathrm{cost}_{\mathrm{cong}^+}(A) := \sum_{e \in E(A)} \int_{\mathrm{stick}(e)} \psi \, ds + \sum_{v \in V(A)} \int_{\mathrm{stick}(v)} \psi \, ds$$

The dynamic local usage uses the same timing usage model as the traditional router [27]. Like the congestion price function that defines prices for the edge resources, the resource sharing also provides prices for the sink delay resources. To optimize sink delay resource usage during resource sharing (see Section 2.4) we define the following delay costs using the Elmore delay from Definition 8:

Definition 17 (Elmore delay costs)**.** Given a local route A with source v_{source} and sink delay prices $\mathrm{cost}_{\mathrm{delay}}(t)$ for each sink t of A, the Elmore delay costs of A are defined as

$$\mathrm{cost}_{\mathrm{delay}}(A) := \sum_{\substack{t \in V(A) \\ t \text{ is a sink}}} \mathrm{delay}_A(v_{\mathrm{source}}, t) \cdot \mathrm{cost}_{\mathrm{delay}}(t)$$

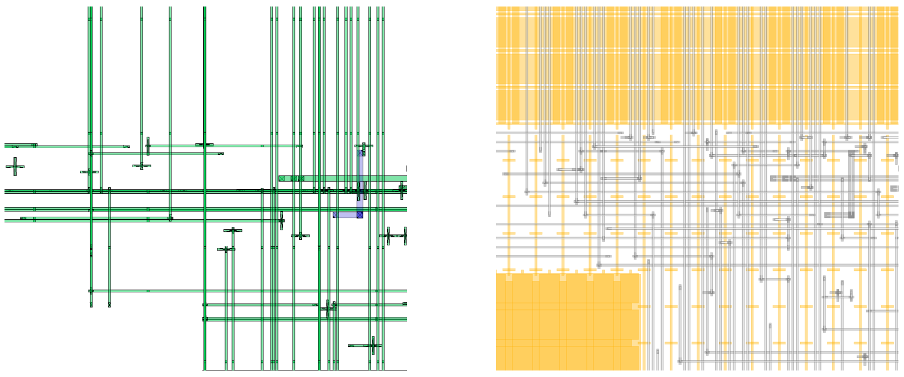

Wiring (green) of BonnRouteGlobal
Wiring (gray) of BonnRouteDetail

Figure 10: The same excerpt of a chip with complicated blockage structure (yellow): once with global wiring (left) and detailed wiring (right) on two neighboring layers. Note that many global wires lie on top of each other.

In addition to congestion and timing costs BonnRouteGlobal accounts costs per length of a wire and per via to favor routes with short wire length and few vias even if congestion and timing prices are low. Since these costs do not pose any additional difficulty for the optimization algorithms and can be modeled by a constant offset to the congestion costs we will ignore them in this thesis.

3.1.1 *Controlling Packing Density*

BonnRouteGlobal does not avoid overlaps between wires of different routes. If BonnRouteGlobal fully packed the chip with wires considering only their immediate area without additional measures, the packing would be much too dense for the detailed router which has to route overlap-free and respect additional design rule constraints. Figure 10 shows an excerpt of global wiring and the resulting detailed wiring output. BonnRouteDetail needs to put many more vias and short segments to be able to realize the global routes given the complicated blockage structure. There are several ways to make BonnRouteGlobal account for these difficulties, trying to model and estimate the additional space that BonnRouteDetail needs to realize all routes.

GLOBAL CAPACITY REDUCTION Usually, the capacity of all edge resources in BonnRouteGlobal is reduced by 10 percent. This is to account for additional space that BonnRouteDetail will require, but also to leave some space on the chip for other later local modifications. As an example via meshes, that are special via structures on top of some pins, have to be rebuilt several times during the routing flow. Such small changes are in need of some space left. For the dynamic local usage we reduce the global capacity by the same amount as for the traditional router.

layerwise capacity reduction BonnRouteGlobal can additionally reduce the capacity of specific layers. This is to account for layers that are specifically difficult to route for the detailed router, for example due to hard design rules. This depends on the chip technology BonnRouteGlobal is used for. The layerwise capacity reduction can also be used to account for space that BonnRouteDetail will need to access the pin shapes, which are usually located on the lower layers. Again, for the dynamic local usage we use the same paramter settings as for the traditional router which is to slightly reduce capacity on the lower layers that contain most of the pins.

local wiring pre-estimates The traditional router computes an initial usage for each edge resource (Figure 13, 14 and 15 on page 25). This should pre-estimate the usage arising from local wires and vias inside the global routing tiles. It is computed based on a shortest Steiner tree in the plane for each net. Depending on the location of tile-internal edges of this tree usage is added to the edge resources covering for tile-internal wires. Depending on the location of Steiner nodes and terminals of this tree usage is added covering for vias. Since the Steiner trees are two-dimensional only, all usage is collected for a certain area and then distributed among the layers: starting at the lowest layer the corresponding edge resources are filled up until some pre-defined threshold. This heuristic distribution of initial usage over the layers poses a severe inaccuracy of the pre-estimates which the dynamic local usage avoids. Furthermore, the pre-estimates are static and do not account for the actual routing of a net which might differ from a shortest Steiner tree due to congestion or critical timing. Moreover they do not account for changed pin positions or added or deleted pins during routing based optimization.

tip-to-tip penalty A wire placed by BonnRouteDetail partially blocking a routing track poses an obstacle for other nets, that cannot use this routing track anymore. They can still partly access the track by vias but it is unlikely that a partly blocked track can be completely filled up by wires of other nets. This is even impossible due to the requirement of so called tip-to-tip spacings that we already mentioned earlier: Manufacturing rules require the detailed router to leave a certain minimum spacing between wires in preferred routing dimension. In addition to the tip-to-tip spacing given by the design rules we also want to cover for space that BonnRouteDetail needs to pack the wires without overlaps (Figure 11) or that is consumed by additional detours that BonnRouteDetail has to take to route all nets conforming with the design rules. This motivates the extension of the

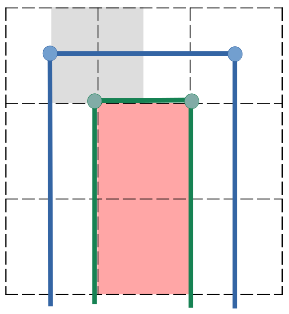


Figure 11: Assume that the red area is a blockage or are highly congested global routing tiles. To route around, the traditional router would account usage for the blue route through the centers of the global routing tiles (dashed). The dynamic local usage would account usage for the green route that is shortest possible around the blocked area. If there are several such routes on top of each other the dynamic local usage is too optimistic because the detailed router must spread the routes across the full area of the global routing tiles to pack them without overlaps. For example the gray edge area would not be accounted for enough usage. The tip-to-tip penalty partly compensates for this effect. On the other hand, the traditional router is too pessimistic if there are less than the maximum amount of routes that fit into the global routing tiles. Note that the route of the traditional router and of the dynamic local usage would look the same if the red blockage was slightly expanded in all directions. In that situation, the routes would be too optimistic because they were actually located inside the blockage.

area of all wires and vias by more than the tip-to-tip spacing: In Definition 13 we choose $k_{\text{tip}} > 1$ to approximately model all these effects. In the implementation in BonnRouteGlobal Definition 13 is slightly extended:

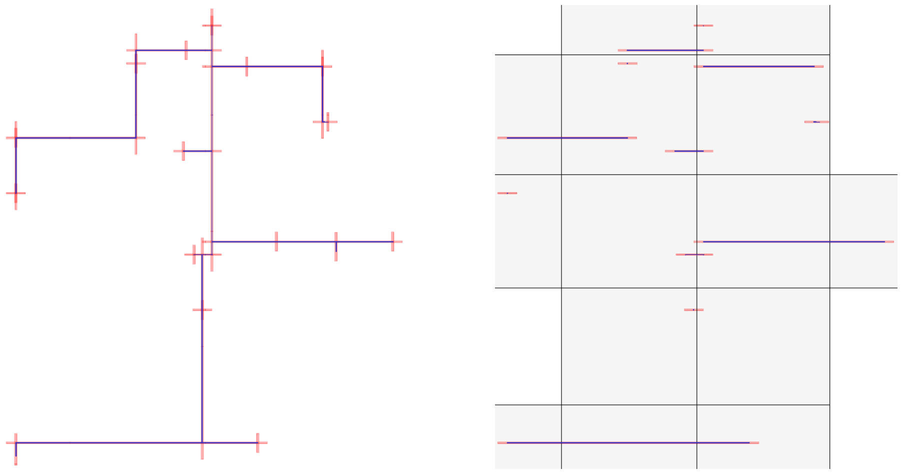
Definition 18 (Bounded tip-to-tip penalty)**.**

$$l_{\text{tip}}(z)^* := \min \begin{Bmatrix} k_{\text{tip}} \cdot \dfrac{\min \begin{Bmatrix} \text{dist}_{\text{tip}}(z) \\ 0.6 \cdot (\text{dist}_{\text{pitch}}(z+1) + \text{dist}_{\text{pitch}}(z-1)) \end{Bmatrix}}{2} \\ 0.4 \cdot \text{length}_{\text{tile}} \end{Bmatrix}$$

The tip-to-tip penalty is bounded at fourty percent of the average global routing tile length $\text{length}_{\text{tile}}$ to avoid over-congestion in case of large tip-to-tip spacings. Moreover, to make the definition more independent of the tip-to-tip spacing dist_{tip} we also take the minimum of roughly the average of the pitches on the layer above and below. The pitch of a layer is defined as the minimum distance of the center lines of two neighbouring wires such that they do not violate any spacing constraints. Since the detailed router has to route to the layer above or below for detours it makes sense to consider these numbers in the estimate of how much additional space BonnRouteDetail needs to realize the global routing. In practice this definition yields good results in terms of correlation with BonnRouteDetail, using a value of $k_{\text{tip}} \sim 12$. In current 7nm and 5nm technology there are up to fifteen layers and on the higher layers this results in tip-to-tip penalty lengths of up to fourty percent of the average global routing tile length. Without the bound it would be even bigger for wire types with particularly large tip-to-tip spacings. On lower layers the tip-to-tip penalty lengths are shorter starting at around eight percent of the average global routing tile length.

Note that the traditional router also extends all tile-crossing wires to the tile centers of the respective global routing tiles for usage computation, that is it extends the wires up to fifty percent of the global routing tile length. In comparison, the dynamic local usage is less pessimistic. Figure 12 shows the area covered by the tip-to-tip penalty for a route in a real-world chip instance.

In the implementation of the dynamic local usage we do not add tip-to-tip penalty usage at nodes that are incident to pins. The area of the pin shapes is already subtracted from the capacity of the belonging edge resources and pins can be packed very densely. Adding tip-to-tip penalty at them would lead to huge over-congestion. Another technical aspect is the presence of wires that were placed by the detailed router and which we call *detailed wires*. In particular during detail routing based optimization (see Section 4.5) there can be many detailed wires in the input to BonnRouteGlobal. BonnRoute-Global is supposed to close remaning opens in the routes for which



All layers

A low layer, including used global edge areas (gray)

Figure 12: Wires of a net (blue), area considered as usage including tip-to-tip spacing (red)

it may re-use the detailed wires in the input. The purpose of the tip-to-tip penalty is to account for additional space that BonnRouteDetail needs. Therefore we do not add it around detailed wires that are already valid detailed routing output.

Figure 13, 14 and 15 show the tip-to-tip penalty usage on three different layers along with congestion plots and plots of the pre-estimate usage of the traditional router on one medium-sized chip unit. One can see clear similarities among the tip-to-tip penalty and the pre-estimates. However, on the lower layers the pre-estimate usage is higher than the tip-to-tip penalty usage of the dynamic local usage (Figure 13). This is due to the fact that the dynamic local usage estimates the space BonnRouteDetail needs for local wiring not only by the tip-to-tip penalty but also by the actual area of the tile-internal wires themselves. Many of these tile-internal wires and also much of the pre-estimate usage within the traditional router is assigned to the lower layers where most of the pins are located. On the middle layers, that contain less pins and that hold the most of long interconnect wires, the tip-to-tip penalty consumes more usage than the traditional pre-estimates (Figure 14). For this keep in mind that the traditional router not only creates additional usage by the pre-estimates but also by extending all tile-crossing wires to the tile centers of the respective global routing tiles for usage computation. With the dynamic local usage the latter effect is partly replaced by the tip-to-tip penalty. On the highest layers the congestion is very similar among the dynamic local usage and the traditional router on that chip unit and there is not a significant difference between the tip-to-tip penalty and pre-estimate usage neither (Figure 15).

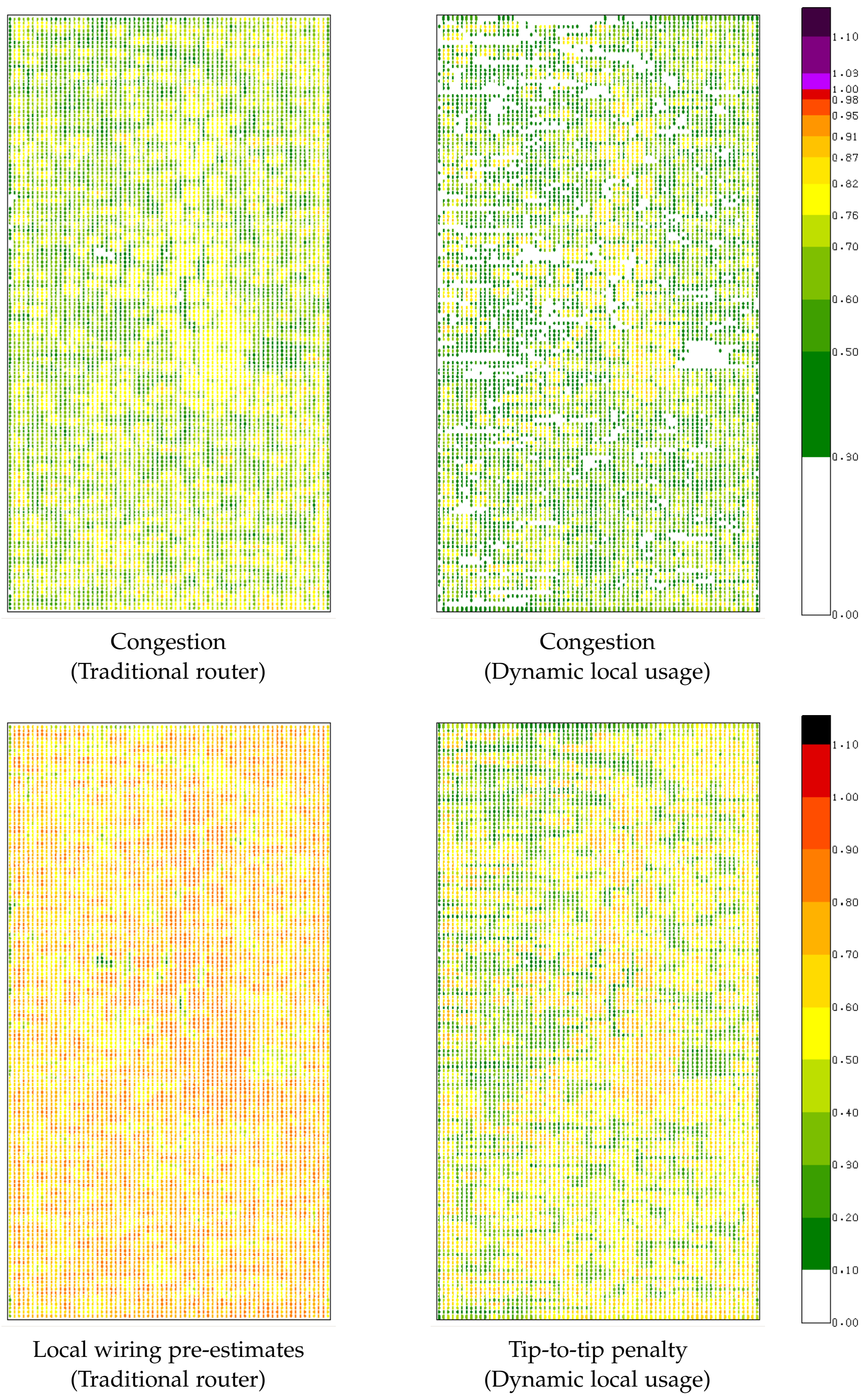
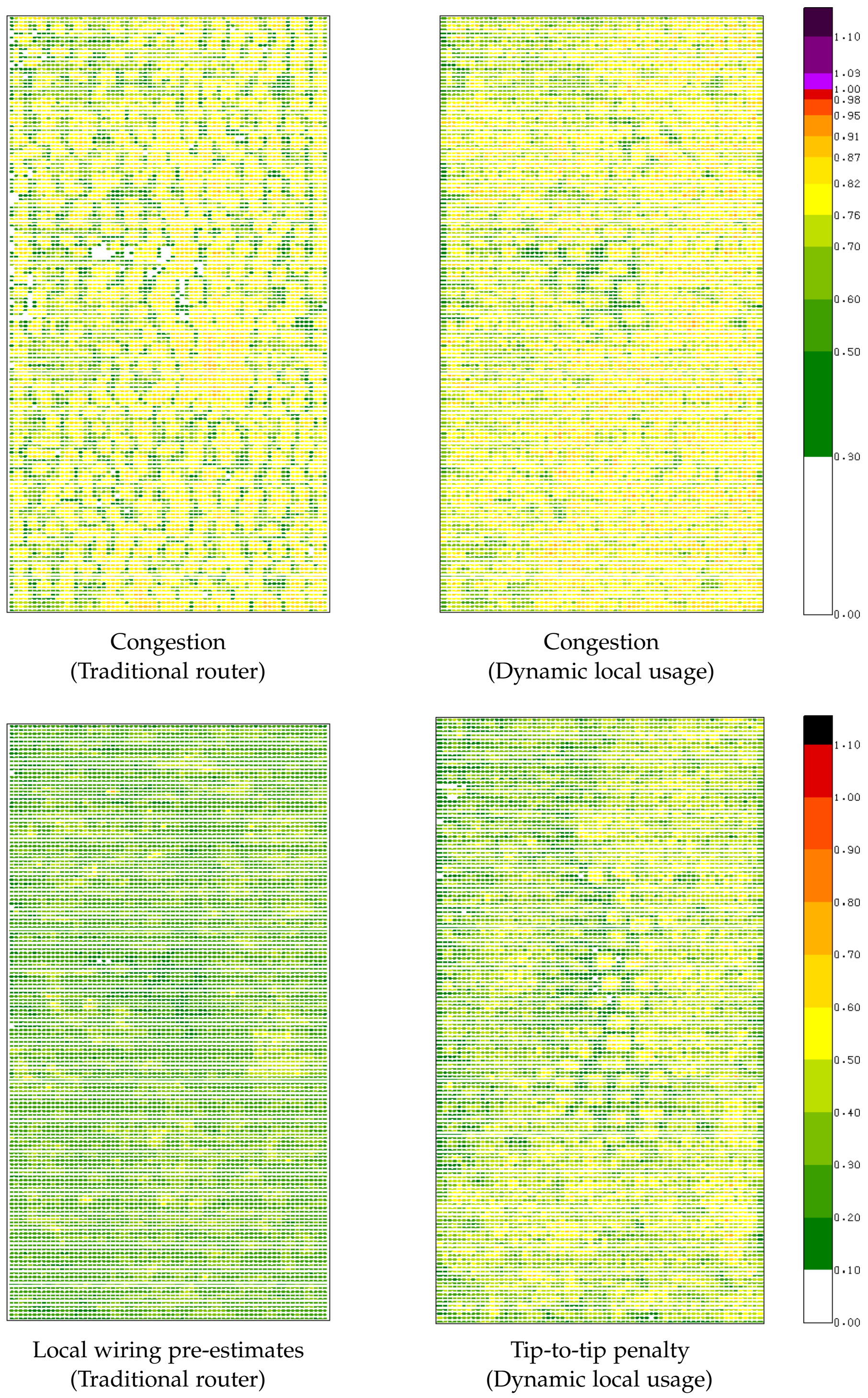


Figure 13: A low layer on chip C_1 (see Section 4.1 for more details on our testbed). The pre-estimate usage is higher than the tip-to-tip penalty usage due to the fact that the dynamic local usage adds separate usage for tile-internal wires based on their actual space consumption. In hotspots the traditional router and the dynamic local usage have the same congestion while overall congestion is lower with the dynamic local usage.



Congestion
(Traditional router)

Congestion
(Dynamic local usage)

Local wiring pre-estimates
(Traditional router)

Tip-to-tip penalty
(Dynamic local usage)

Figure 14: A medium layer on chip C_1. The tip-to-tip penalty requires more usage than the traditional pre-estimates. The tip-to-tip penalty also consists of usage from long tile-crossing wires which are not covered in the pre-estimates of the traditional router. However, the traditional router extends all global routing tile-crossing wires to the tile centers for usage computation, so there is less a difference in overall congestion between the traditional router and the dynamic local usage.

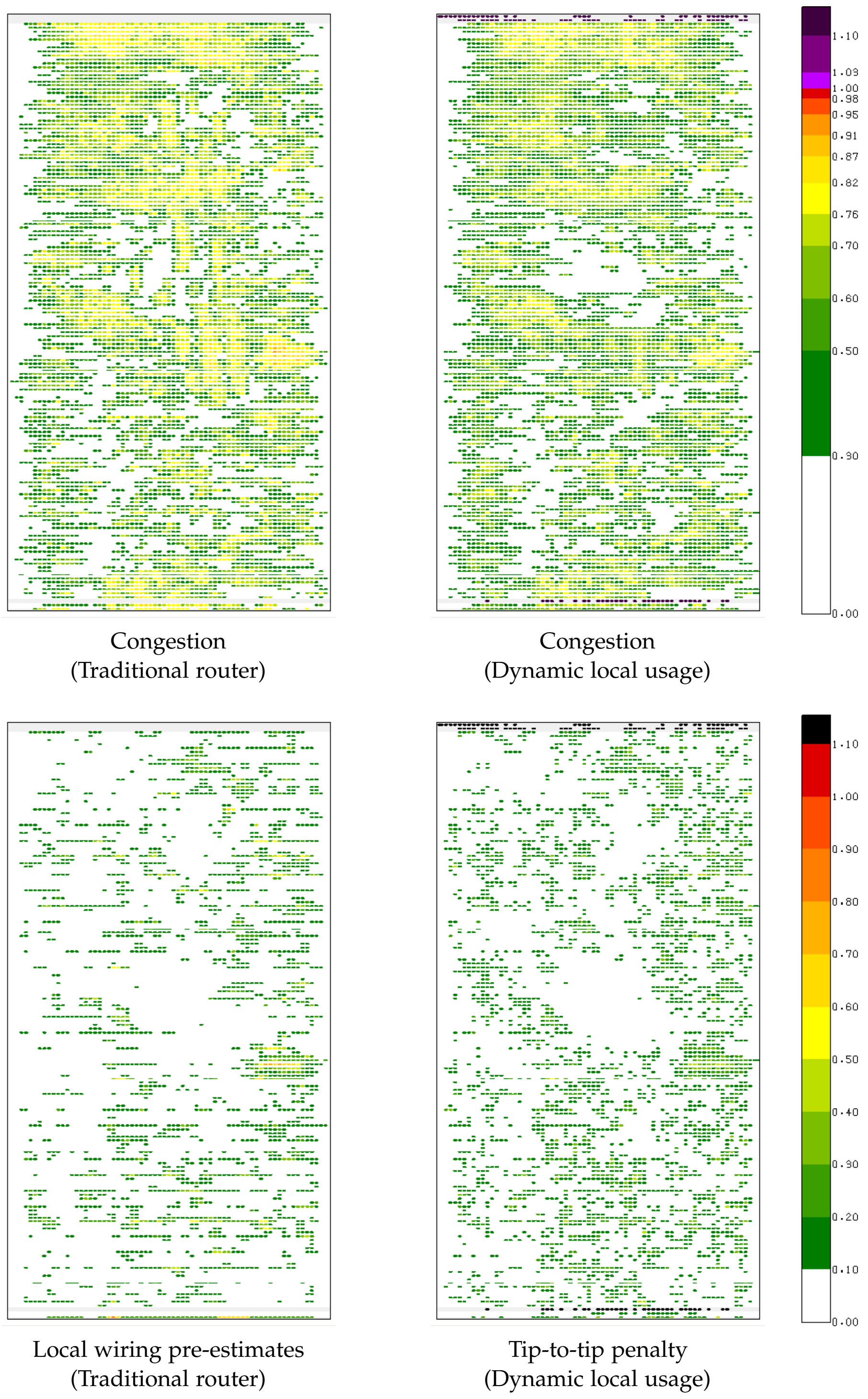


Figure 15: A high layer on chip C_1. Both the congestion plots and the tip-to-tip penalty and pre-estimate usage are very similar.

3.1.2 *Estimating Signal Delay*

BonnRouteGlobal estimates the signal delay through the routes using the Elmore delay [27]. As shortly outlined in Section 2.4, BonnRoute-Global computes timing-optimized routes and distributes the slack among the nets using the resource sharing framework. For this it is essential to have a good estimate on the signal delay through a route. The local routes computed by BonnRouteGlobal usually are shorter than the routes from BonnRouteDetail (see Section 4). With the dynamic local usage the tip-to-tip penalty covers for the additional wires of BonnRouteDetail in terms of congestion. A similar approach for timing showed improvements in practice: The resistance and capacitance of all wires created by BonnRouteGlobal is multiplied by a penalty factor. BonnRouteGlobal then optimizes all routes with regard to these adjusted timing values. Unlike the tip-to-tip penalty this is a multiplicative penalty. Different variants of an additive timing penalty did not show as good results. In fact, on more recent instances the timing after detailed routing was also good without this penalty factor during global routing. Therefore, depending on future technologies, this modification might not be needed in the future any longer.

Still it has to be kept in mind that the traditional router usually computes longer routes than the dynamic local usage (see Section 4). Moreover, for the embedding of tile-internal wires the traditional router does not consider timing. Altogether, the traditional router is more pessimistic with regard to timing than the dynamic local usage.

3.2 COMPUTING COARSE ROUTES

At the core of BonnRouteGlobal there is the computation of coarse routes on the global routing graph which can be described as the following problem:

Problem 19. MINIMUM COARSE ROUTE

Input: A net n.

Output: A coarse route A for n minimizing

$$\text{cost}_{\text{cong}}(A) + \text{cost}_{\text{delay}}(A)$$

If delay costs are ignored this is a weighted minimum Steiner tree problem in graphs which is well-known to be $\mathcal{NP}$-hard [32]. [28] is among the exact algorithms that are fast enough for instances of limited size. The best currently known approximation factor achievable in polynomial time is 1.39 by the LP-based approach from [12].

Only recently [54] showed that the same approximation factor can be achieved by a purely combinatorial algorithm. However, as the instances in global routing have enormous sizes only very fast approximation algorithms are suitable for BonnRouteGlobal. A 2-approximation to the minimum weighted Steiner tree problem can be obtained by successively connecting components of terminals in the graph with shortest paths [35]. For this, BonnRouteGlobal uses Dijkstra's algorithm [49], which is sufficiently fast and in practice the resulting Steiner trees are usually not much worse than the optimum solution. Furthermore, there exist various speedup techniques for this approach [26].

The edge costs for the path searches with Dijkstra's algorithm can be easily adapted to the area usage model of the dynamic local usage in Definition 15. The traditional router uses the same costs for edges like the dynamic local usage, but computes via congestion costs differently by considering the output of the pre-estimates (Section 3.1.1), causing another dependence on the pre-estimate computation. The dynamic local usage uses via congestion costs that purely depend on their area and the prices of the edge resources as in Definition 15.

To correctly model the tip-to-tip penalty costs we introduce node costs: a node has the cost of a stacked via, including tip-to-tip penalty costs. That is, for a node $v = (\overline{x}, \overline{y}, \overline{z}) \in V(\mathcal{G})$ that is located on an x-dimension layer $\overline{z} \in \mathcal{L}_x$ and a congestion price function ψ, we define

$$c(v) := \int_{\overline{x}-0.5-l_{\text{tip}}(\overline{z})}^{\overline{x}+0.5+l_{\text{tip}}(\overline{z})} \psi(\kappa, \overline{y}, \overline{z}) d\kappa$$

If a node $v \in \mathcal{G}$ has been already added to the tree during an earlier path search we set $c(v) := 0$ because the tip-to-tip penalty costs were already accounted for. This way, the costs of stacked vias are considered correctly during path search.

To compute correct via costs for nodes that are incident to x- or y-dimension edges, we have to modify the edge costs. For an edge in x-dimension $e = (v, w) = ((\overline{x_1}, \overline{y}, \overline{z}), (\overline{x_2}, \overline{y}, \overline{z})) \in E(\mathcal{G})$ with $\overline{x_1} < \overline{x_2}$ we reduce the cost induced by its wire stick by one tip-to-tip penalty extension and half the width of a via at both of its ends v and w, that is

$$c(e) := \int_{\overline{x_1}+0.5+l_{\text{tip}}(\overline{z})}^{\overline{x_2}-0.5-l_{\text{tip}}(\overline{z})} \psi(\kappa, \overline{y}, \overline{z}) d\kappa$$

Node and edge costs on y-dimension layers are defined analogously. This definition accounts for the fact that if an x- or y-dimension edge e is incident to a node v, its stick overlaps with the tip-to-tip penalty extension that already $c(v)$ accounts for. For a coarse route $A \subset \mathcal{G}$ we have

$$\sum_{v \in V(A)} c(v) + \sum_{e \in E(A)} c(e) = \text{cost}_{\text{cong}}(A)$$

with $\text{cost}_{\text{cong}}$ from Definition 15. If in practice there are global edges with a length shorter than two tip-to-tip penalty extensions, it has to be made sure that no edge is assigned a negative edge cost to be able to use Dijkstra's algorithm. We address this issue by taking the maximum of the previously defined costs $c(e)$ and zero, leading to slight inaccuracies only because only few global routing graph edges are substantially shorter than the average.

Given edge and node costs, Dijkstra's algorithm still finds a shortest path. However, the approximation ratio of the underlying Steiner tree algorithm is not given any longer. In fact [22] shows that it is $\mathcal{NP}$-hard to approximate the node-weighted Steiner tree problem within a factor of $(1-\epsilon)\ln k$ for any constant $\epsilon > 0$ and k being the number of terminals. In case of bounded node degrees like it is the case for the global routing graph there is still a constant-factor approximation:

Theorem 20. *Assume there is a graph* G *with edge and node costs* $c : V(G) \cup E(G) \to \mathbb{R}_{\geqslant 0}$, *a value* $\gamma > 0$ *with*

$$c(v) \leqslant \gamma \left(c(v) + \sum_{e \in \delta(v)} c(e)/2 \right)$$

for all $v \in V(G)$, *a set of terminals* $T \subset V(G)$ *and* $d \geqslant 3$ *with* $|\delta(v)| \leqslant d$ *for all* $v \in V(G)$. *Let* S *be a cheapest terminal spanning tree in the metric closure of* G *and* S^* *a cheapest Steiner tree for the terminals* T. *Then it holds that*

$$c(S) \leqslant \max(d-1, (d-2)\cdot\gamma + 2) \cdot c(S^*)$$

The global routing graph $\mathcal{G}$ has a maximum node degree of $d = 4$. Assuming uniform global routing graph edge lengths and neglecting via widths, for any node $v = (\overline{x_1}, \overline{y}, \overline{z}) \in \mathcal{G}$ and two incident edges $e = ((\overline{x_0}, \overline{y}, \overline{z}), v)$ and $f = (v, (\overline{x_2}, \overline{y}, \overline{z}))$ on an x-dimension layer we have

$$\begin{aligned} c(v) &= \int_{\overline{x_1}-l_{\text{tip}}(\overline{z})}^{\overline{x_1}+l_{\text{tip}}(\overline{z})} \psi(\kappa, \overline{y}, \overline{z}) d\kappa \\ &\leqslant 0.4 \cdot \int_{\overline{x_1}-\text{length}_{\text{tile}}(\overline{z})}^{\overline{x_1}+\text{length}_{\text{tile}}(\overline{z})} \psi(\kappa, \overline{y}, \overline{z}) d\kappa \\ &= 0.8 \cdot \int_{\overline{x_1}-\frac{1}{2}\text{length}_{\text{tile}}(\overline{z})}^{\overline{x_1}+\frac{1}{2}\text{length}_{\text{tile}}(\overline{z})} \psi(\kappa, \overline{y}, \overline{z}) d\kappa \\ &= 0.8 \cdot \left(c(v) + \sum_{e \in \delta(v)} c(e)/2 \right) \end{aligned}$$

The inequality holds by Definition 18 and the same holds for y-dimension layers. It follows that $\gamma = 0.8$ can be chosen in Theorem 20 which then states that a terminal spanning graph yields a 3.6-approximation for the minimum Steiner problem with tip-to-tip penalty costs.

Proof (of Theorem 20). For all $v \in V(G)$ with $|\delta_{S^*}(v)| = d$ let $\gamma_v \leqslant \gamma$ such that

$$c(v) = \gamma_v \left(c(v) + \sum_{e \in \delta(v)} c(e)/2 \right)$$

Rearranging, this is equivalent to

$$\begin{aligned} d \cdot c(v) + 2 \sum_{e \in \delta(v)} c(e)/2 \\ = ((d-2) \cdot \gamma_v + 2) \cdot \left(c(v) + \sum_{e \in \delta(v)} c(e)/2 \right) \end{aligned} \tag{2}$$

The proof extends the proof that a cheapest terminal spanning tree is a 2-approximation in the case that there are no node costs [35]. Consider $E' := E(S^*) \sqcup E(S^*)$ which is Eulerian and as such it contains an Eulerian path P through all terminals T. P traverses each edge of S^* at most twice and each node $v \in V(S)$ at most $|\delta(v)|$ times. Hence for the cost of the path P it holds that

$$\begin{aligned} c(P) \leqslant & \sum_{\substack{v \in V(S^*) \\ |\delta_{S^*}(v)| < d}} (d-1) \cdot c(v) + \sum_{\substack{v \in V(S^*) \\ |\delta_{S^*}(v)| = d}} d \cdot c(v) + 2 \cdot \sum_{e \in E(S^*)} c(e) \\ = & \sum_{\substack{v \in V(S^*) \\ |\delta_{S^*}(v)| < d}} \left((d-1) \cdot c(v) + 2 \cdot \sum_{e \in \delta_{S^*}(v)} c(e)/2 \right) \\ & + \sum_{\substack{v \in V(S^*) \\ |\delta_{S^*}(v)| = d}} \left(d \cdot c(v) + 2 \cdot \sum_{e \in \delta_{S^*}(v)} c(e)/2 \right) \\ \leqslant & (d-1) \cdot \sum_{\substack{v \in V(S^*) \\ |\delta_{S^*}(v)| < d}} \left(c(v) + \sum_{e \in \delta_{S^*}(v)} c(e)/2 \right) \\ & + ((d-2) \cdot \gamma + 2) \cdot \sum_{\substack{v \in V(S^*) \\ |\delta_{S^*}(v)| = d}} \left(c(v) + \sum_{e \in \delta_{S^*}(v)} c(e)/2 \right) \\ \leqslant & \max (d-1, (d-2) \cdot \gamma + 2) \, c(S^*) \end{aligned}$$

For the penultimate inequality we used $d - 1 \geqslant 2$, (2) and $\gamma_v \leqslant \gamma$. By short-cutting P one obtains a terminal spanning tree in the metric closure that has no higher cost than P proving the statement. □

Figure 16 shows that the approximation ratio is tight for $\gamma = 1$. Table 17 suggests that in practice the approximation ratio usually is much better.

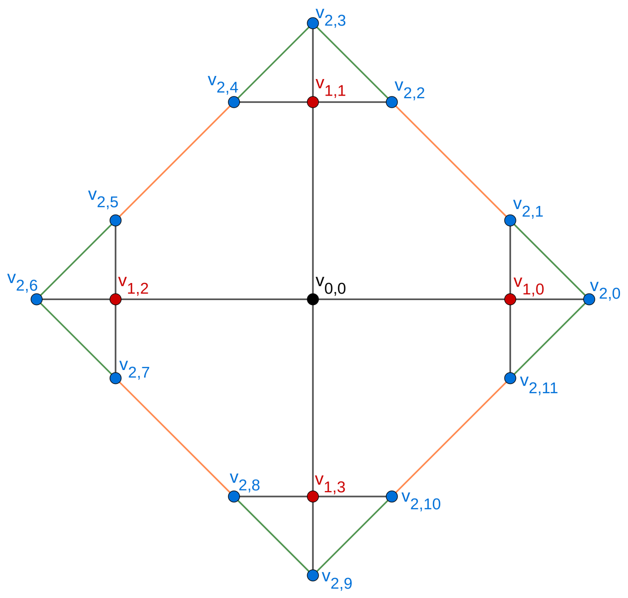


Figure 16: Let $d \geqslant 3$ and $k \geqslant 2$ (the image is for $d = 4$ and $k = 2$). Consider a graph $G_{d,k}$ that contains a central node $v_{0,0}$ on *level* zero. $v_{0,0}$ is adjacent to d nodes on level one, enumerated counterclockwise by $v_{1,0}, ..., v_{1,d-1}$. Each of these nodes is adjacent to $d - 1$ nodes on level two, all of them are again enumerated counterclockwise by $v_{2,0}, ..., v_{2,d\cdot(d-1)-1}$. We do this recursively until level k on which there are $d \cdot (d-1)^{k-1}$ nodes. Additionally, we add edges between the nodes on level k in counterclockwise order (green and orange). The nodes on level k are selected as terminals (blue). The nodes on level $k-1$ (red) have cost one, all other nodes have cost zero. The black edges also have cost zero. The green and orange edges have costs that equal the cost of the unique path between their end points using black edges only, that is green edges have cost one and orange edges have cost two. Consider the Steiner tree S^* that consists of all black edges. It connects all terminals and has cost $c(S^*) = d \cdot (d-1)^{k-2}$. The green and orange edges each connect a pair of terminals and have costs that equal the costs of a shortest path between them. Hence, the set of the green and all but one orange edge forms a minimum terminal spanning tree S in the metric closure of G. Its cost equals $c(S) = d \cdot (d-1)^{k-2} \cdot d - 2$, because for each node in level $k-1$ it contains $d-2$ edges with cost one and one edge with cost 2. One of the edges of cost 2 is not contained in S. All in all we have $\frac{c(S)}{c(S^*)} = \frac{d\cdot(d-1)^{k-2}\cdot d-2}{d\cdot(d-1)^{k-2}}$ which goes towards d for $k \to \infty$. For $d = 4$ the graph $G_{d,k}$ can arise as an instance of the minimum coarse route problem: The red nodes are embedded on a global routing layer with a tip-to-tip penalty length that equals half the length of the global routing graph edges, all other nodes are embedded on layers with zero tip-to-tip penalty. The edges are realized arbitrarily within the global routing graph while choosing the correct costs of either zero, one or two. For all other edges of the global routing graph we choose very large costs such that they will never be part of a solution.

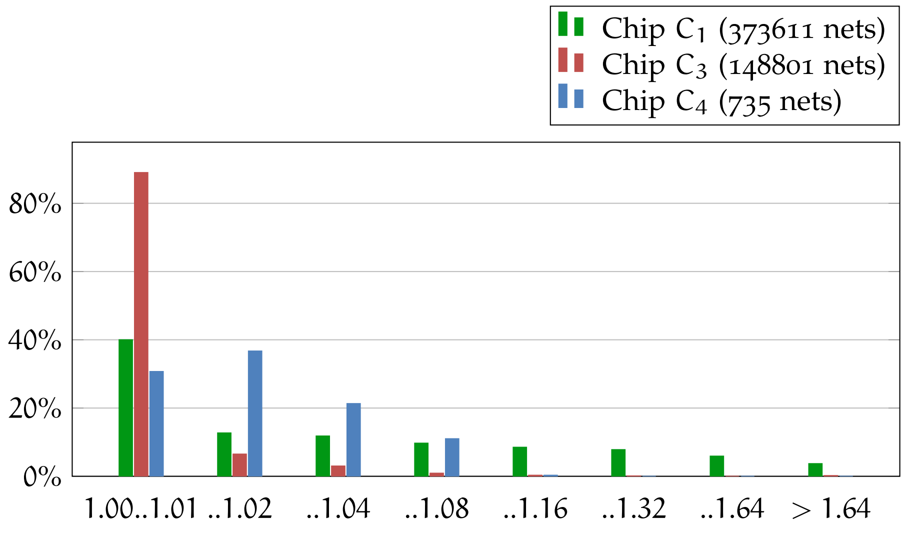


Figure 17: Let P_1 be the problem of finding an optimum solution for the minimum coarse route problem. Let P_2 be the problem of finding an optimum solution for the minimum coarse route problem with all tip-to-tip penalties being zero. Certainly the cost of an optimum solution for P_2 is a lower bound for the cost of an optimum solution for P_1. For this table we computed one approximately optimum solution for problems P_1 and P_2 each time a coarse route needed to be computed during global routing, using the approach described in this section. The table shows the percentages of instances for which the quotient of the costs of these two routes lies in a certain bucket. For the vast majority of instances this quotient is less than 1.08, meaning that the costs do not increase much with tip-to-tip penalties. Therefore, problem P_1 can be approximated almost as well as problem P_2 in practice, though the theoretic approximation guarantee is almost twice as bad. Note that a quotient larger than one does not imply that a solution for P_1 is sub-optimal: With tip-to-tip penalties the optimum solution can have arbitrary higher costs than the optimum solution for the case that all tip-to-tip penalties are zero.

Note that both for the dynamic local usage and for the traditional router stacked via costs, arising from the width of the via, have to be accounted per node. While these are much smaller than the additional tip-to-tip penalty costs they still have the effect that also the traditional router only achieves an approximation ratio of 3 in theory assuming that γ can be chosen small.

So far we only considered congestion costs. The computation of the coarse route is more elaborate if we want to optimize delay costs. [53] shows that it is $\mathcal{NP}$-hard to approximate a coarse route optimizing Elmore delay costs within a constant factor. BonnRouteGlobal implements a heuristic algorithm from [53] which takes an initial coarse route A as an input and reconnects sub-arborescences of A to the source if the delay from the source exceeds a certain threshold. The best of the initial and the reconnected route in terms of congestion and timing costs is taken. To be able to compute their costs accurately BonnRouteGlobal first connects the coarse route to the exact pin shapes and performs post-optimizations on the resulting local route as described in the following sections. This has to be kept in mind for the dynamic local usage, for which the post-optimization of the local routes is more elaborate and costs more running time.

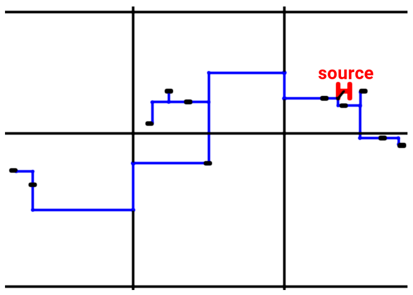
3.3 COMPUTING LOCAL ROUTES

BonnRouteGlobal connects the coarse route of a net through Steiner trees inside the global routing tiles to the exact shapes of the pins of the net (Figure 18 on the left). Unlike Definition 3, in practice pins can consist of several shapes which is why we are given a group Steiner tree problem. The local route only has to access one shape of each pin. As the group Steiner tree problem is even hard to approximate [23] BonnRouteGlobal heuristically picks an access point for each pin considering the locations of the other pins. Then we compute a rectilinear Steiner tree for the set of access points in that global routing tile and the intersection points of the coarse route with the global routing tile borders. Replacing the (planar) edges of the coarse route inside each global routing tile by these Steiner trees yields an arborescence that connects to the two-dimensional projections of the exact pin shapes. However, as seen in Figure 18, this local route usually is far from optimum in terms of wire length. The following sections describe a post-optimization routine that is applied to reduce wire length, congestion and timing costs. Moreover the arborescence has to be embedded into layers in order to obtain a local route for the net, which will be dealt with in Section 3.3.2.

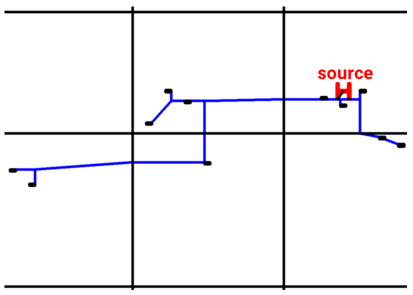
3.3.1 *Optimizing x- and y-Coordinates*

Having computed a Steiner tree connecting to the exact pin shapes, BonnRouteGlobal optimizes the x- and y-positions of the Steiner nodes to minimize its length (see Figure 18). This is in particular important to establish short source-sink-paths for a good timing. In the traditional router this is achieved by a dynamic program solving a minimum Steiner tree with fixed topology problem [33], in which all Steiner nodes are constrained to stay in their respective global routing tile, such that the global layout of the route does not change. As in the traditional router only wires crossing global routing tile borders contribute usage, this optimization does not change the edge resource usages because the route is modified within the global routing tiles only.

However, with the dynamic local usage edge resource usage can change through this optimization, since all wires contribute usage and might be moved to different price regions (see Figure 19 for an



Local route connecting to the exact pin shapes.



Local route with optimized x-y-coordinates.

Figure 18: The local route on the left connects to the exact pin shapes through shortest Steiner trees inside the global routing tiles (blue). To optimize the length of the resulting local route, the wires are moved within the global routing tiles (black) in x- and y-direction (local route on the right). Resulting diagonal edges are later replaced by rectilinear wires.

example). Optimizing the x- and y-coordinates without considering congestion can lead to congestion issues (see Figure 20). Hence, with the dynamic local usage we need to optimize the x- and y-coordinates also with regard to congestion costs. In this section we modify the dynamic program from [33] to minimize not only the length but also the cost with regard to congestion. Moreover, we incorporate timing costs for linear delay or for a lower bound of the Elmore delay.

3.3.1.1 *Problem Definition*

Problem 21. MIN. COST ARBORESCENCE WITH FIXED TOPOLOGY

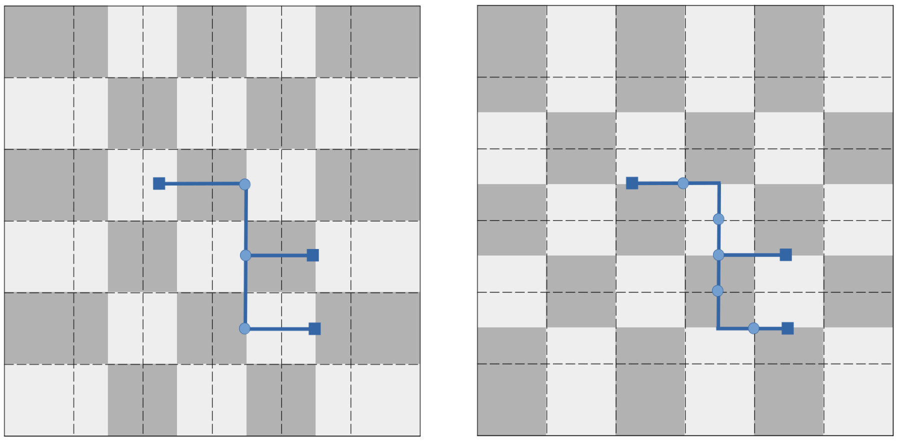
Input: An arborescence G with fixed topology, rectangular move bounds $\square(v) \subset \mathcal{C}$ for all nodes $v \in V(G)$, edge costs $c_e : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}_{\geqslant 0}$ for each $e \in E(G)$ and node costs $c_v : \mathbb{R}^2 \to \mathbb{R}_{\geqslant 0}$ for each $v \in V(G)$.

Output: For each node $v \in V(G)$ find a position $f(v) \in \square(v)$ minimizing the total cost

$$\sum_{e=(v,w)\in E(G)} c_e(f(v), f(w)) + \sum_{v\in V(G)} c_v(f(v))$$

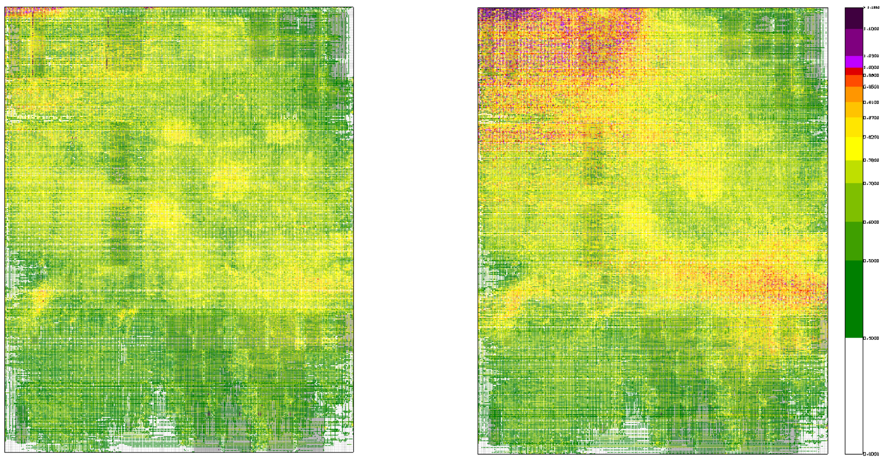
For a Steiner node we will choose the movebound $\square$ as the area of the global routing tile the node belongs to. For a terminal it will be set as the area of the rectangular pin shape it connects to. For the congestion-unaware optimization problem, we choose

$$c_e(p_1, p_2) := \|p_1 - p_2\|$$



x-dimension layers   y-dimension layers

Figure 19: Borders of global routing tiles (dashed black), different price regions (light and dark gray). Moving the trunk in the center of the blue Steiner tree to the left or right changes the amount of wiring that is contained in some of the price regions on the x-dimension layers. The y-dimension layers are not affected by this movement.



No x-y-optimization.   Congestion-unaware x-y-optimization.

Figure 20: With the dynamic local usage there can be a severe increase in congestion with the congestion unaware x-y-optimization. Note that in these runs the tip-to-tip penalty usage was accounted pointwise at nodes and not distributed along a stick as in Definition 15 and in the current implementation. Distributing it over all edge resources that are intersected by the tip-to-tip penalty extension stick already slightly reduces the increase in congestion.

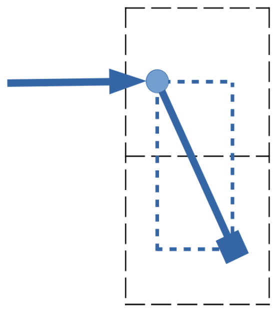


Figure 21: There are two rectilinear embeddings of the diagonal arc which might have different congestion costs on the x-layers.

for all $e \in E(G)$ and $p_1, p_2 \in \mathbb{R}^2$. The via cost functions are set to $c_v \equiv 0$ for all nodes $v \in V(G)$. To optimize congestion costs, we will define the edge costs as the sum of the component-wise wire length times the congestion price of the underlying global edge areas. Note that each global routing tile contains two different price regions (Figure 19). The via costs will reflect the congestion costs of the via pads at a certain location. For terminals $t \in T$ we will choose $c_t \equiv 0$. See Section 3.3.1.4 for detailed definitions. How to additionally approximate Elmore delay costs is described in Section 3.3.1.5. Linear delay costs can be easily modeled by an offset to the wire and via prices.

Note that we do not require that the resulting optimized arborescence is rectilinear. This is because the layer embedding algorithm that follows the optimization of the x- and y-positions in BonnRoute-Global will optimally choose one of the two possible embeddings of a diagonal edge (Figure 21). Moreover, the aim is to be able to solve the minimum cost arborescence with fixed topology problem in x- and y-dimension independently, which would not be possible if the result would be required to be rectilinear.

Definition 22. A function $c : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}_{\geqslant 0}$ is additively separable if for some $c^x, c^y : \mathbb{R} \times \mathbb{R} \to \mathbb{R}_{\geqslant 0}$ it can be written as

$$c(p_1, p_2) = c^x(p_1^x, p_2^x) + c^y(p_1^y, p_2^y)$$

for all $(p_1^x, p_1^y) = p_1 \in \mathbb{R}^2$ and $(p_2^x, p_2^y) = p_2 \in \mathbb{R}^2$.

All cost functions c_e and c_v will be additively separable in the x- and y-coordinates. In other words, the costs incurred by the length and position in x-dimension do not depend on the length and position in y-dimension and vice versa. As the task is to minimize over a sum of these functions, the whole problem can be solved independently in x- and y-dimension and thereby gets much simpler. Hence from now on we consider the one-dimensional problem of optimizing x-coordinates only. Furthermore we consider cost functions that arise from an integral so we can use the one-dimensional projection of a given congestion price function.

Problem 23. ONE-DIMENSIONAL MINIMUM COST ARBORESCENCE WITH FIXED TOPOLOGY

Input: An arborescence G with fixed topology, intervals $\square(v) \subset \mathbb{R}$ for all nodes $v \in V(G)$, for each edge $e = (v, w) \in E(G)$ prices $p_{e,v}$ and $p_{e,w} : \mathbb{R} \to \mathbb{R}_{\geqslant 0}$ and node costs $c_v : \mathbb{R} \to \mathbb{R}_{\geqslant 0}$ for all nodes $v \in V(G)$.

Output: For each node $v \in V(G)$ find a position $f(v) \in \square(v)$ minimizing the total cost

$$\sum_{e=(v,w)\in E(G)} \min_{\alpha\in\{v,w\}} \left| \int_{f(v)}^{f(w)} p_{e,\alpha}(\kappa) d\kappa \right| + \sum_{v\in V(G)} c_v(f(v)) \quad (3)$$

An instance of the minimum cost arborescence with fixed topology problem can be projected to two instances of the one-dimensional problem if the cost functions are additively separable with $c_e = c_e^x + c_e^y$ and $c_v = c_v^x + c_v^y$ for all $e \in E(G)$ and $v \in V(G)$ and the cost functions c_e^x and c_e^y have the form

$$(x_1, x_2) \mapsto \min_{\alpha\in\{v,w\}} \left| \int_{x_1}^{x_2} p_{e,\alpha}(\kappa) d\kappa \right|$$

for functions $p_{e,v}$ and $p_{e,w}$. In our global routing context, the separability of the edge cost functions particularly requires that any two adjacent nodes in the instance route belong to the same global routing tile column or row. This is always the case because the coarse routes are computed on the global routing graph and movement thereafter is restricted to global routing tile areas.

The cost functions that we will define for the congestion-aware x-y-optimization will be different for each edge or node, because nodes and edges whose projections are equal in one dimension may have different coordinates in the other dimension and hence may belong to different global routing tiles with different prices. The integral yields the congestion cost of a wire with unit width as defined in Definition 15. The intent behind the minimum in (3) is to allow two different price functions for all edges, one for each of their end points and depending on the prices in the global routing tiles of the end points. If the end points of an edge whose end points are located in different global routing tiles are moved such that the edge is diagonal, a later post-processing can choose the cheaper of the two possible rectilinear embeddings (Figure 21) which we want to model in the problem definition.

3.3.1.2 *Dynamic Programming*

Considering a node $v \in V(G)$, the costs incurred by this node and incident edges only depend on its position and the positions of adjacent nodes. Hence there is a straight-forward dynamic program to solve the one-dimensional minimum cost arborescence with fixed topology problem (see also [33]). First we give some definitions:

Definition 24. Let v_{source} the root of the arborescence G. Define for each node $u \in V(G)$:

- p_u as the parent node ($\emptyset$ if $u = v_{\text{source}}$),
- Y_u as the sub-arborescence rooted at u and
- $s_u(x) : \square(u) \to \mathbb{R}_{\geqslant 0}$ as the cheapest total cost of Y_u if $f(u) = x$, that is

$$s_u(x) = \min_{\substack{f:V(G)\to\mathbb{R} \\ f(v)\in\square(v)\forall v\in V(G) \\ f(u)=x}} \left(\sum_{\substack{e=(v,w) \\ \in E(Y_u)}} \min_{\alpha\in\{v,w\}} \left| \int_{f(v)}^{f(w)} p_{e,\alpha}(\kappa)d\kappa \right| + \sum_{v\in V(Y_u)} c_v(f(v)) \right)$$

Algorithm 1 computes the functions s recursively in reversed topological order. Then, in topological order it assigns the positions to the nodes for which s is minimum.

A straight-forward induction shows that Algorithm 1 computes an optimum solution to the one-dimensional minimum cost arborescence with fixed topology problem. The following two sections describe how to efficiently implement Algorithm 1 with cost functions defined for the congestion-unaware and congestion-aware minimum cost arborescence with fixed topology problem.

3.3.1.3 *Congestion-Unaware x- and y-Optimization*

Assume we are given an instance of the one-dimensional minimum cost arborescence with fixed topology problem. If we only want to optimize total wire length one can choose $p_{e,v} \equiv p_{e,w} \equiv 1$ for all edges $e = (v, w) \in E(G)$ and $c_v \equiv 0$ for all $v \in V(G)$. Choosing these cost functions the cost of an edge equals its length:

$$\min_{\alpha\in\{v,w\}} \left| \int_{x_1}^{x_2} p_{e,\alpha}(\kappa)d\kappa \right| = |x_1 - x_2|$$

We refer to [52] where it is shown that in this setting all the functions s from Definition 24 are piece-wise linear and have an integral slope that is monotonically increasing (see Figure 22 for an example).

Algorithm 1: One-dimensional dynamic embedding

```
Input: An instance $(G, \Box, p, c)$ of the one-dimensional minimum cost arborescence with fixed topology problem.
Output: Positions for the nodes minimizing total cost and obeying the movebounds.
1 for $u \in V(G)$ in reverse topological order do
2     Compute the function $s_u : \Box(u) \to \mathbb{R}_{\geqslant 0}$
      $$s_u(x) = c_u(x) + \sum_{\substack{e=(u,v)\in\\ \delta^+_G(u)}} \min_{\substack{x'\in\Box(v)\\ \alpha\in\{u,v\}}} \left( \left| \int_x^{x'} p_{e,\alpha}(\kappa)d\kappa \right| + s_v(x') \right)$$
3 $f(v_{\text{source}}) \in \arg\min_{x\in\Box(v_{\text{source}})} s_{v_{\text{source}}}(x)$
4 for $u \in V(G) \setminus v_{source}$ in topological order do
5     $$f(u) \in \arg\min_{x\in\Box(u)} \left( s_u(x) + \min_{\alpha\in\{p_u,u\}} \left| \int_{f(p_u)}^{x} p_{(p_u,u),\alpha}(\kappa)d\kappa \right| \right)$$
6 return f
```

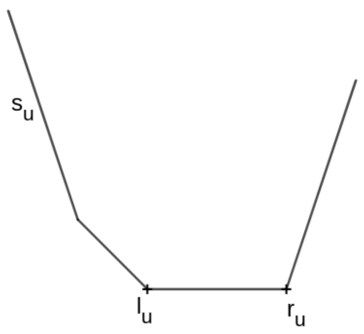


Figure 22: An example of what the function s_u from Definition 24 for some $u \in V(G)$ might look like with unit edge price functions and no via costs. In $[l_u, r_u]$ the function attains its minimum.

For $v \in V(G)$ let $[l_v, r_v]$ be the interval where s_v is minimal. It is proven in [52] that in order to compute $[l_v, r_v]$ it suffices to know the values l_u, r_u for all children $u \in W_v$ and to compute their median. Computing the median of n numbers can be done in time $\mathcal{O}(n)$ [11], leading to the following result:

Theorem 25. *[52] The one-dimensional minimum cost arborescence with fixed topology problem with unit edge price functions and no via costs can be solved in time $\mathcal{O}(|V(G)|)$.*

In [33] this result is extended by modifying the dynamic program to obtain an optimum solution for which, among all optimum solutions, all paths from the source to the sinks of the arborescence are shortest possible. This aids in achieving good timing.

3.3.1.4 *Congestion-Aware x- and y-Optimization*

We present two variants of cost functions that optimize congestion costs with the dynamic local usage.

First, assume that we are given an arborescence G along with a preliminary embedding into the chip image and layers, that is we have some $g : V(G) \to \mathcal{C} \times \mathcal{L}$. We want to optimize the x- and y-coordinates within the global routing tiles respecting the given layers.

Definition 26 (Layer-embedded congestion costs)**.** Let G an arborescence embedded into $\mathcal{C} \times \mathcal{L}$ and $\psi : \mathcal{C} \times \mathcal{L} \to \mathbb{R}_{\geqslant 0}$ a congestion price function. We define two instances of the one-dimensional minimum cost arborescence with fixed topology problem, one for x- and one for y-dimension.

For $v \in V(G)$ let $t(v) \in V(\mathcal{G})$ be the global routing graph node whose tile area contains v. For each $v \in V(G)$ define movebound intervals

$$\begin{aligned}\square^x(v) &:= \text{proj}^x(\square(t(v)))\\ \square^y(v) &:= \text{proj}^y(\square(t(v)))\end{aligned}$$

as the projection of the belonging tile areas into x- and y-dimension.

For each edge $e = (v, w) \in E(G)$ we define edge price functions $p^x_{e,v}, p^x_{e,w} : \text{proj}^x(\mathcal{C}) \to \mathbb{R}_{\geqslant 0}$ and $p^y_{e,v}, p^y_{e,w} : \text{proj}^y(\mathcal{C}) \to \mathbb{R}_{\geqslant 0}$. Here the cost functions $p_{e,v}$ and $p_{e,w}$ will only depend on e, hence for simpler notation we only define functions p^x_e and p^y_e. For each node $v \in V(G)$ we define via price functions $c^x_v : \square^x(v) \to \mathbb{R}_{\geqslant 0}$ and $c^y_v : \square^y(v) \to \mathbb{R}_{\geqslant 0}$.

For an edge in x-dimension $e = (v, w) = ((\overline{x_1}, \overline{y}, \overline{z}), (\overline{x_2}, \overline{y}, \overline{z}))$ define

$$\begin{aligned}p^x_e(\_) &:= \psi(\_, \overline{y}, \overline{z})\\ p^y_e(\_) &:= MAX\end{aligned}$$

Note that $\psi(x, \overline{y}, \overline{z}) = \psi(x, y', \overline{z})$ for any x and $y' \in \square^y(v)$ by definition of the congestion price function ψ. Setting $p^y_e(\_) := MAX$ for a sufficiently large $MAX \in \mathbb{R}$ ensures that both endpoints of the edge e are always embedded at the same y-coordinate, hence the edge keeps its x-dimension. Analogously, for an edge in y-dimension $e = ((\overline{x}, \overline{y_1}, \overline{z}), (\overline{x}, \overline{y_2}, \overline{z}))$ define

$$\begin{aligned} p_e^y(\_) &:= \psi(\overline{x}, \_, \overline{z}) \\ p_e^x(\_) &:= MAX \end{aligned}$$

For a via edge $e = (v, w) = ((\overline{x}, \overline{y}, \overline{z_1}), (\overline{x}, \overline{y}, \overline{z_2}))$ with $|\overline{z_1} - \overline{z_2}| = 1$ define

$$\begin{aligned} p_e^x(\_) &:= MAX \\ p_e^y(\_) &:= MAX \end{aligned}$$

For all nodes $v = (\overline{x}, \overline{y}, \overline{z}) \in V(G)$ that are incident to a via edge, we set

$$c_v^x(\_) := \begin{cases} \int_{-0.5-l_{tip}(\overline{z})}^{0.5+l_{tip}(\overline{z})} \psi(\_ + \kappa, \overline{y}, \overline{z}) d\kappa & \overline{z} \in \mathcal{L}_x \\ 0 & \text{otherwise} \end{cases}$$

$$c_v^y(\_) := \begin{cases} \int_{-0.5-l_{tip}(\overline{z})}^{0.5+l_{tip}(\overline{z})} \psi(\overline{x}, \_ + \kappa, \overline{z}) d\kappa & \overline{z} \in \mathcal{L}_y \\ 0 & \text{otherwise} \end{cases}$$

For all nodes $v \in V(G)$ that are not incident to a via edge, we set $c_v^x \equiv c_v^y \equiv 0$.

In other words, the layer-embedded congestion functions allow the nodes of G to be moved within the global routing tiles so that each edge keeps its orientation and has cost according to the congestion prices on the layer it was embedded into. The x-instance only considers costs of wires in x-dimension and of vias on x-dimension layers, while the y-instance does the same for y-dimension and y-layers. Recall that we assume wires and vias to have width one; otherwise, we would have to multiply ψ by the width. Each via is expanded by the tip-to-tip penalty, which is slightly pessimistic, as we do not check here if a via is incident to a wire. It is not possible to consider tip-to-tip penalties exactly at this point because then the x-instance and the y-instance would not be independent from each other (Figure 23).

If tip-to-tip penalties are zero this model induces the same costs as in Definition 15:

Remark 27. Let A a local route embedded into $\mathcal{C} \times \mathcal{L}$ and $\text{dist}_{tip}(z) = 0$ for all layers $z \in \mathcal{L}$. Then the sum of the layer-embedded congestion costs of all edges and nodes of the two instances of the one-dimensional minimum cost arborescence with fixed topology problem arising from the underlying arborescence equals the intersection-ignoring congestion costs of A:

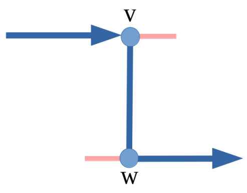


Figure 23: This example shows that the tip-to-tip penalties in x-dimension depend on the positions of the nodes in y-dimension. If and only if nodes v and w do not have the same y-coordinates, tip-to-tip penalties (light red) have to be accounted for at v and w.

$$\begin{aligned}&\sum_{\substack{e=((\overline{x_1},\overline{y},\overline{z}),(\overline{x_2},\overline{y},\overline{z}))\in E(A)\\ \overline{x_1}\neq\overline{x_2}}} \left|\int_{\overline{x_1}}^{\overline{x_2}} p_e^x(\kappa)d\kappa\right|\\ +&\sum_{\substack{e=((\overline{x},\overline{y_1},\overline{z}),(\overline{x},\overline{y_2},\overline{z}))\in E(A)\\ \overline{y_1}\neq\overline{y_2}}} \left|\int_{\overline{y_1}}^{\overline{y_2}} p_e^y(\kappa)d\kappa\right|\\ &+\sum_{v=(\overline{x},\overline{y},\overline{z})\in V(A)} c_v^x(\overline{x})+c_v^y(\overline{y})\\ &=\mathrm{cost}_{\mathrm{cong}^+}(A)\end{aligned}$$

The layer-embedded congestion costs require an arborescence that is already embedded into the layers of the chip. Right after connecting the coarse route to the exact pin shapes the layout of the resulting arborescence can be poor as can be seen on Figure 18. Hence it can be beneficial to first optimize the x- and y-coordinates of the arborescence before embedding it into layers. We will now propose congestion costs that do not assume the arborescence to be embedded into layers but rather approximate a lower bound on the congestion costs.

Definition 28 (Planar congestion costs)**.** Let G an arborescence embedded into the chip image $\mathcal{C}$ and $\psi : \mathcal{C} \times \mathcal{L} \to \mathbb{R}_{\geqslant 0}$ a congestion price function. Again we define two instances of the one-dimensional minimum cost arborescence with fixed topology problem, one for x- and one for y-dimension.

For $v \in V(G)$ let $t(v) \in V(\mathcal{G})$ a global routing graph node on any layer whose projected tile area contains v. For each $v \in V(G)$ define the same movebound intervals as in Definition 26:

$$\begin{aligned}\square^x(v) :=& \mathrm{proj}^x(\square(t(v)))\\ \square^y(v) :=& \mathrm{proj}^y(\square(t(v)))\end{aligned}$$

For each edge $e = (v, w) \in E(G)$ we define edge price functions $p^x_{e,v}, p^x_{e,w} : \mathrm{proj}^x(\mathcal{C}) \to \mathbb{R}_{\geqslant 0}$ and $p^y_{e,v}, p^y_{e,w} : \mathrm{proj}^y(\mathcal{C}) \to \mathbb{R}_{\geqslant 0}$. Here the cost functions $p_{e,v}$ and $p_{e,w}$ will only depend on v and w, hence for simpler notation we only define functions p^x_v and p^y_v for each

$v \in V(G)$. Moreover, for each node $v \in V(G)$ we define via price functions $c_v^x : \Box^x(v) \to \mathbb{R}_{\geqslant 0}$ and $c_v^y : \Box^y(v) \to \mathbb{R}_{\geqslant 0}$.

For a node $v = (\bar{x}, \bar{y}) \in V(G)$ define edge price functions

$$p_v^x(\_) := \min_{z \in \mathcal{L}_x} \psi(\_, \bar{y}, z)$$
$$p_v^y(\_) := \min_{z \in \mathcal{L}_y} \psi(\bar{x}, \_, z)$$

Let p_1, p_2 be the two prices that p_v^x assumes in the global routing tile $\Box(t(v))$ and l_1, l_2 the layers at which the congestion price function assumed these values. Let x_{cut} be the coordinate (that is the global routing edge border) at which p_v^x switches from p_1 to p_2. $\text{via}_1 := (1 + 2l_{\text{tip}}(l_1)) \cdot p_1$ is the cost of a via pad including tip-to-tip penalty on the edge area with price p_1. Likewise, define $\text{via}_2 := (1 + 2l_{\text{tip}}(l_2)) \cdot p_2$. The via price function is now defined as

$$c_v^x(\kappa) := \text{via}_1 \qquad \text{for } \kappa \leqslant x_{\text{cut}} - l_{\text{tip}}(l_1)$$
$$c_v^x(\kappa) := \text{via}_2 \qquad \text{for } \kappa \geqslant x_{\text{cut}} + l_{\text{tip}}(l_2)$$

From $x_{\text{cut}} - l_{\text{tip}}(l_1)$ to $x_{\text{cut}} + l_{\text{tip}}(l_2)$ we let $c_v^x(\_)$ transition linearly between the two values. $c_v^y(\_)$ is defined analogously.

The planar congestion costs assume that the wires will be embedded on the cheapest possible layers. On those cheapest layers, the via cost functions add for every Steiner node the cost of one via pad including tip-to-tip penalty. The wire cost functions p alone constitute a lower bound on the congestion costs of the resulting route. Together with the via cost functions the planar cost functions are not a lower bound though: The arborescence could be embedded into layers such that no vias are used on nodes whose incident edges have the same routing dimension. However, one would need to know both the x- and y-coordinates of the nodes to determine those nodes. Not considering any via costs would lead to a real lower bound of the congestion costs, however that did not yield good results in practice.

In contrast to the setting with unit edge price functions and no via costs in Section 3.3.1.3, both with the planar and the layer-embedded cost functions the resulting cost functions s in the dynamic program can have several local minima (Figure 24). We will show that they are still piece-wise linear, using the fact that the edge price functions are piece-wise constant and the via cost functions are piece-wise linear because the congestion price function is piece-wise constant.

Definition 29 (Piece-wise linear function)**.** Let $[a, b] \subset \mathbb{R}$. A function $f : [a, b] \to \mathbb{R}$ is piece-wise linear if there are intervals

$$[a, x_1), [x_1, x_2), ..., [x_n, b]$$

such that the restriction of f to any of these intervals is a linear function. $a, x_1, x_2, ..., x_n, b$ are referred to as break points.

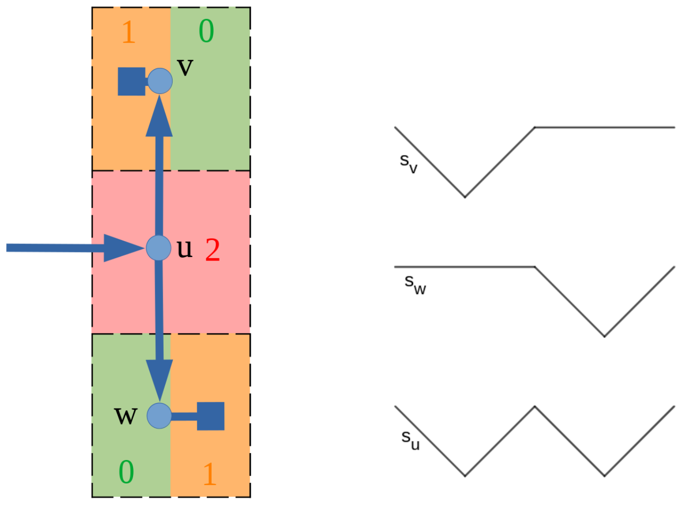


Figure 24: Consider the x-dimension instance resulting from this arborescence either with planar or layer-embedded congestion costs. The colors indicate the prices of the edge areas. The Steiner nodes u, v and w can be moved in x-direction within the global routing tiles (dashed). On the right the cost functions s_u, s_v and s_w are drawn in x-dimension. Because the red edge area has costs two, and v and w have only one child, it is $s_u = s_v + s_w$. The sum s_u has two local minima. As u is accessed from the left the optimum solution puts all Steiner nodes u, v and w on one vertical line below the top terminal (blue square).

Definition 30 (Piece-wise constant function)**.** Let $[a, b] \subset \mathbb{R}$. A function $g : [a, b] \to \mathbb{R}$ is piece-wise constant if it is piece-wise linear and has constant slope except at its discontinuities.

A piece-wise linear function f with m break points can be represented by $3m - 2$ numbers: the coordinates $x_1, ..., x_m$ of the break points, the function values $f(x_1), ..., f(x_{m-1})$ and the slopes of f between the break points. A piece-wise constant function g with m break points can be represented by $2m - 1$ numbers: the coordinates $x_1, ..., x_m$ of the break points and the function values $f(x_1), ..., f(x_{m-1})$. By $\frac{d}{d\kappa\downarrow} f$ we denote the derivative from above which is defined at the full support of a piece-wise linear function f, because the intervals $[a, x_1), [x_1, x_2), ..., [x_n, b]$ in Definition 29 are assumed to be closed at their lower ends.

The via cost functions defined in Definition 26 and 28 are piece-wise linear, the edge price functions are piece-wise constant. Given a piece-wise constant edge price function $p : [a, b] \to \mathbb{R}_{\geqslant 0}$ with break points $a, x_1, ..., x_m, b$, the integral

$$\int_l^u p(x)\, d\kappa$$

is both piece-wise linear in u for any fixed $l \in [a, b]$ and in l for any fixed $u \in [a, b]$ with the break points being $a, x_1, ..., x_m, b$, the same as of p, plus the fixed value of l or u. From now on we will assume that all edge price functions are piece-wise constant and all via cost functions are piece-wise linear.

Adding, subtracting or computing the pointwise minimum of two piece-wise linear functions again results in a piece-wise linear function. To show that the function computed in line 2 of the dynamic program 1 is piece-wise linear is the subject of the following two lemmas:

Lemma 31. *Let* $[a,b] \subset \mathbb{R}$ *and* $p : [a,b] \to \mathbb{R}_{\geqslant 0}$ *piece-wise constant and* $s : [a,b] \to \mathbb{R}_{\geqslant 0}$ *piece-wise linear and* $x \in [a,b]$ *fixed. Moreover fix an operator* $\circ \in \{\leqslant, \geqslant, \vee\}$ *(we say that* $r \vee s$ *is true for all* $r, s \in \mathbb{R}$*). Then*

$$\begin{aligned}&\min_{\substack{x' \in [a,b] \\ x' \circ x}} \left(\left| \int_x^{x'} p(\kappa) d\kappa \right| + s(x') \right) \\ = &\min_{\substack{x' \text{ break point of } p \text{ or } s, \text{ or } x'=x \\ x' \circ x}} \left(\left| \int_x^{x'} p(\kappa) d\kappa \right| + s(x') \right)\end{aligned}$$

Proof. Define

$$c(x,x') := \left| \int_x^{x'} p(\kappa) d\kappa \right|$$

$c(x,\_)$ is piece-wise linear and has the same break points as p apart from potentially x.

The direction $\leqslant$ is clear. For $\geqslant$ let

$$x'' := \arg\min_{\substack{x' \in [a,b] \\ x' \circ x}} \left(c(x,x') + s(x') \right)$$

If x'' is a break point of s or $c(x,\_)$ or $x'' = x$ we are done. Otherwise let x_l be the nearest break point left of x'' of s or $c(x,\_)$, or $x_l = x$ if x comes first. Analogously let x_r the nearest break point right of x'' or $x_r = x$. Since $x'' \circ x$ it must also hold that $x_l \circ x$ and $x_r \circ x$. The restrictions of $s(\_)$ and $c(x,\_)$ to $[x_l, x_r)$ must be linear functions. It follows that also $c(x,\_) + s(\_)$ is linear on $[x_l, x_r)$ and thus must be even constant, as otherwise x'' would not be the minimum. Therefore

$$\min_{x' \in [a,b]} \left(c(x,x') + s(x') \right) = c(x,x'') + s(x'') = c(x,x_l) + s(x_l)$$

while x_l is a break point of s or $c(x,\_)$ and $x_l \circ x$.

□

Lemma 32. *Let* $[a,b] \subset \mathbb{R}$ *and* $p : [a,b] \to \mathbb{R}_{\geqslant 0}$ *piece-wise constant and* $s : [a,b] \to \mathbb{R}_{\geqslant 0}$ *piece-wise linear. Let* m *the number of distinct break points of* p *and* s*. Then*

$$t(x) := \min_{x' \in [a,b]} \left(\left| \int_x^{x'} p(\kappa) d\kappa \right| + s(x') \right)$$

is piece-wise linear and can be computed in time $\mathcal{O}(m)$*.*

Algorithm 2: Right-Min-Propagation

```
Input: $[a, b] \subset \mathbb{R}$, $p : [a, b] \to \mathbb{R}_{\geqslant 0}$ piece-wise constant,
       $s : [a, b] \to \mathbb{R}_{\geqslant 0}$ piece-wise linear.
Output: A piece-wise linear function as in Claim 32.1.
1  $x_0 := a$
2  if $\frac{d}{d\kappa\downarrow} s(x_0) > p(x_0)$ then
3      $x_1 := \min\left(\left\{x \in (x_0, b) : s(x) = s(x_0) + \left|\int_{x_0}^{x} p(\kappa) d\kappa\right|\right\} \cup \{b\}\right)$
4      $g(x) := s(x_0) + \left|\int_{x_0}^{x} p(\kappa) d\kappa\right|$   for all $x \in [x_0, x_1)$
5  else
6      $x_1 :=$ next break point of $p$ or $s$ after $x_0$, or $b$
7      $g(x) := s(x)$   for all $x \in [x_0, x_1)$
8  $x_0 := x_1$
9  if $x_0 < b$ then
10     goto 2
11 $g(b) := \lim_{x \uparrow b} g(x)$ // Extend $g$ to the right border of $[a, b]$
12 return $g$
```

Proof. Again we define

$$c(x, x') := \left|\int_{x}^{x'} p(\kappa) d\kappa\right|$$

By Lemma 31 (with $\circ = \vee$) and using $c(x', x') = 0$ for all x' we know

$$t(x) = \min_{x' \text{ break point of } p \text{ or } s, \text{ or } x'=x} \left(c(x, x') + s(x')\right) \tag{4}$$

For each break point x', $c(\_, x') + s(x')$ is piece-wise linear as a sum of two piece-wise linear functions. Hence t is piece-wise linear as it is the minimum of finitely many piece-wise linear functions.

t could be computed as the pointwise minimum of $m + 1$ piece-wise linear functions with m distinct break points in total. By [10] this can be done in time $\mathcal{O}(m \log m)$. In general, this is also a natural lower bound for computing the pointwise minimum of piece-wise linear functions with m break points in total because it is necessary to sort the break points of all functions given as input.

However, here we are given the break points of p and s in sorted order and the functions $c(\_, x') + s(x')$ share many similarities. To achieve linear running time for this problem we make use of Algorithm 2. This algorithm iterates over all break points of p and s from left to right and hence has running time $\mathcal{O}(m)$. Let g the returned function. Define

$$r(x) := \min_{\substack{x' \text{ break point of } p \text{ or } s, \text{ or } x'=x \\ x \geqslant x'}} \left(c(x, x') + s(x')\right)$$

Claim 32.1. *r is piece-wise linear, has at most $2m$ break points and it is $g = r \leqslant s$.*

Proof. Lines 2 and 3 of Algorithm 2 ensure $g \leqslant s$. As for all $x \in [a, b]$ it is $g(x) = c(x, x') + s(x')$ for some break point $x' \leqslant x$ of p or s, or $g(x) = c(x, x) + s(x)$ (using $c(r, r) = 0$ for all r) we have $g \geqslant r$. Together it is $r \leqslant g \leqslant s$. Now fix any $x \in [a, b]$. Let

$$x'' = \max \left\{ \operatorname*{arg\,min}_{\substack{x' \text{ break point of } p, \text{ or } x'=x \\ x \geqslant x'}} \big(c(x, x') + s(x')\big) \right\} \tag{5}$$

that is $r(x) = c(x, x'') + s(x'')$. If $x'' = x$ we have $r(x) = s(x)$ and hence $g(x) = r(x)$ as desired.

So now assume that $x'' < x$ and x'' is a break point of p or s. We know $g(x'') \leqslant s(x'')$. Let $x' \leqslant x''$ be such that $g(x'') = c(x'', x') + s(x')$ and $x' = x''$ or x' is a break point of p or s. We want to show $g(x'') = s(x'')$: If $x' = x''$, then this follows by $c(x'', x'') = 0$. Otherwise, x' must be a break point of p or s. If $g(x'') < s(x'')$ it would be

$$\begin{aligned} c(x, x') + s(x') &= c(x, x'') + c(x'', x') + s(x') \\ &= c(x, x'') + g(x'') < c(x, x'') + s(x'') = r(x) \end{aligned}$$

which is a contradiction to the definition of r. Hence $g(x'') = s(x'')$.

Because $g(x'') = s(x'')$ and x'' is a break point of p or s the algorithm once enters line 2 with $x_0 = x''$. By the definition of x'' in (5) the algorithm must enter the first condition. Then $x_1 > x$ in line 3 as otherwise x'' would not be biggest possible in (5). Hence the algorithm sets $g(\kappa) := s(x'') + c(\kappa, x'')$ for all $x'' \leqslant \kappa < x_1$ including $g(x) := s(x'') + c(x, x'')$.

g is piece-wise linear by construction. All break points of g arising from line 7 are also break points of s. Break points arising from line 4 are either break points of p or equal x_1 defined in line 3. Assume $x_1 \neq b$. If s has a break point in the interval (x_0, x_1) that is not a break point of g, the new break point x_1 is balanced out. If s has no break points in the interval (x_0, x_1), p must have a break point in this interval because otherwise it would be impossible that $g(x_0) = s(x_0)$ and $g(x_1) = s(x_1)$ and $g < s$ on (x_0, x_1). That is why g can have at most m more break points than s has, because for each such new break point x_1 there must be another one of p in (x_0, x_1). □

Claim 32.2. *For* $x \in [a, b]$ *we have*

$$t(x) = \min_{\substack{x' \textit{ break point of } p \textit{ or } r, \textit{ or } x'=x \\ x' \geqslant x}} \big(c(x, x') + r(x')\big) \tag{6}$$

Proof. Let $x \in [a, b]$ fixed.

"$\geqslant$": By (4) there is x' which is a break point of p or s or $x' = x$ with $t(x) = c(x, x') + s(x')$. If $x' \leqslant x$ we have

$$t(x) = c(x, x) + (c(x, x') + s(x')) \geqslant c(x, x) + r(x)$$

If $x' > x$, let $x_0 \leqslant x' \leqslant x_1$ the nearest break points of p or r left and right of x' or x whatever comes first. We have

$$\begin{aligned} t(x) &= c(x,x') + (c(x',x') + s(x')) \\ &\geqslant c(x,x') + r(x') \\ &\geqslant \min\{c(x,x_0) + r(x_0), c(x,x_1) + r(x_1)\} \end{aligned}$$

The last inequality follows because $c(x,\_) + r(\_)$ is linear on $[x_0, x_1]$.

"$\leqslant$": Let x' a break point of p or r or $x' = x$ attaining the minimum in (6). Let $x'' \in [a,b]$ with $x'' = x'$ or x'' a break point of p or s and $x'' \leqslant x'$ such that $r(x') = c(x',x'') + s(x'')$. Therefore, the right-hand side of (6) equals $c(x,x') + c(x',x'') + s(x'') = c(x,x'') + s(x'')$. If x'' is a break point of p or s or $x'' = x$ we are done by (4). Otherwise let $x_0 \leqslant x'' \leqslant x_1$ the nearest break points of p or s left and right of x'' or x whatever comes first. By (4) it is

$$t(x) \leqslant \min\{c(x,x_0) + s(x_0), c(x,x_1) + s(x_1)\} \leqslant c(x,x'') + s(x'')$$

The second inequality is true because $c(x,\_) + s(\_)$ is linear on $[x_0, x_1]$.

$\square$

We first run Algorithm 2 on p and s obtaining r. Then we run a reversed version of Algorithm 2 that iterates over the break points from right to left on p and r obtaining the desired function t by Claim 32.2. As r has at most $2m$ break points, the second call of Algorithm 2 has running time $\mathcal{O}(m)$. Hence, this is also the total running time proving the lemma.

$\square$

Theorem 33. *An instance $(G, \square, p, c)$ of the one-dimensional minimum cost arborescence with fixed topology problem with piece-wise constant edge price functions and piece-wise linear via cost functions can be solved by Algorithm 1 in time $\mathcal{O}(|V(G)| \cdot m)$, where m is the maximum number of break points occurring within the computed piece-wise linear functions.*

Proof. Correctness can be shown by a straight-forward induction.

Adding two piece-wise linear functions or computing their point-wise minimum yields another piece-wise linear function. By Lemma 32 we know that

$$x \to \min_{x' \in \square(v)} \left\{ \left| \int_x^{x'} p_{e,\alpha}(\kappa) d\kappa \right| + s_v(x') \right\} \tag{7}$$

in line 2 of Algorithm 1 is a piece-wise linear function given that $p_{e,\alpha}$ is piece-wise constant and s_v is piece-wise linear. As all the via cost functions are piece-wise linear as well it follows by induction in reverse topological order that s_u is piece-wise linear for all $u \in V(G)$. Adding two piece-wise linear functions with n_1 and n_2 break points or computing their pointwise minimum can be done straightforward in time $\mathcal{O}(n_1 + n_2)$. Finding the minimum of a piece-wise linear function with n break points can be done in time $\mathcal{O}(n)$. Hence, line 3 of Algorithm 1 can be implemented in time $\mathcal{O}(m)$ and line 4 and 5 in $\mathcal{O}(|V(G)| \cdot m)$. In line 2, for each node $v \in V(G) \setminus v_{\text{source}}$ (when iterating over its parent node p_v) we have to compute two functions (7) and add their pointwise minimum to another piece-wise linear function to compute s_{p_v}. This can be done in time $\mathcal{O}(m)$ using Lemma 32. In total, this amounts to a running time of $\mathcal{O}(|V(G)| \cdot m)$ for Algorithm 1. $\square$

We will now show how to bound the number of break points that occur in Algorithm 1.

Lemma 34. *Assume we are given an instance $(G, \square, p, c)$ of the one-dimensional minimum cost arborescence with fixed topology problem. Let*

$$G' := (V(G), \{(v,w) \in E(G) : \square(v) \cap \square(w) \neq \emptyset\}$$

that is G' is a branching containing all nodes of G and only those edges whose endpoints have intersecting movebound intervals. Now let C be a connected component of G' and $[a,b] \subset \mathbb{R}$ such that $\bigcup_{v \in C} \square(v) \subset [a,b]$. Define

$$\begin{aligned} T :=& \{v \in C : \square(v) \neq [a,b]\} \\ B^p :=& \bigcup_{\substack{e=(v,w)\in E(C) \\ \alpha \in \{v,w\}}} \{\textit{break points of } p_{e,\alpha|(a,b)}\} \\ B^c :=& \bigcup_{v \in V(C)} \{\textit{break points of } c_{v|(a,b)}\} \end{aligned}$$

For all $v \in C$, the number of break points of the function s_v computed in Algorithm 1 can be bounded by

$$|B^c| + |B^p| + 2 + 2|T| + 4|B^p| \cdot (|C| - 1)$$

Proof. Given a piece-wise linear function $g : [a,b] \to \mathbb{R}^2_{\geqslant 0}$ and piece-wise constant $p : [a,b] \to \mathbb{R}^2_{\geqslant 0}$, define for $x \in [a,b]$

$$\begin{aligned} p^+[g](x) :=& \min_{a \leqslant y \leqslant x} \left(g(y) + \int_y^x p(\kappa) d\kappa \right) \\ p^-[g](x) :=& \min_{x \leqslant y \leqslant b} \left(g(y) + \int_x^y p(\kappa) d\kappa \right) \\ p[g](x) :=& p^-[p^+[g]](x) \end{aligned}$$

By Lemma 31 $p^+[g]$ and $p^-[g]$ are piece-wise linear and therefore also $p[g]$ is piece-wise linear. Using Lemma 31 and Claim 32.2, it is $p[g](x) = \min_{a \leqslant y \leqslant b} \left(g(y) + \left| \int_x^y p(\kappa) d\kappa \right| \right)$. See Figure 25 for an example.

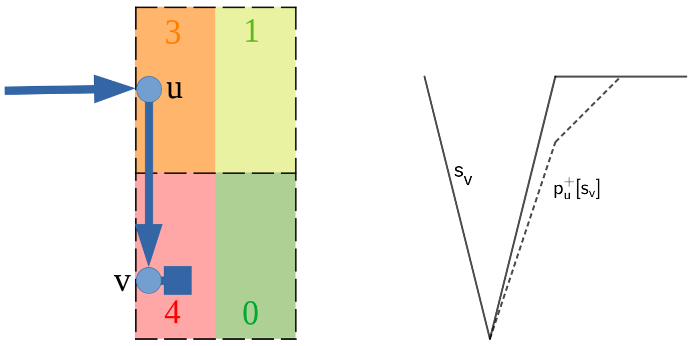


Figure 25: Consider the x-dimension instance resulting from this arborescence with planar congestion costs. The colors indicate the prices of the edge areas. The Steiner nodes u and v can be moved in x-direction within the global routing tiles (dashed). Let p_u the edge price function for incident edges of u. On the right there are the cost functions s_v and $p_u^+[s_v]$ (dashed), which has one additional break point compared to s_v.

Remark 35. Let $g, h : [a, b] \to \mathbb{R}_{\geqslant 0}$ piece-wise linear and $p : [a, b] \to \mathbb{R}_{\geqslant 0}$ piece-wise constant. If $g \leqslant h$ then

$$p^+[g] \leqslant p^+[h] \text{ and } p^-[g] \leqslant p^-[h]$$

In line 2, Algorithm 1 computes a function of the form $\min(p[g], q[g])$. We want to bound the number of its break points. For this, we first rewrite $\min(p[g], q[g])$:

Claim 35.1. *Let* $g : [a, b] \to \mathbb{R}_{\geqslant 0}$ *piece-wise linear and* $p, q : [a, b] \to \mathbb{R}_{\geqslant 0}$ *piece-wise constant. For the pointwise minimum it holds that*

$$\min(p[g], q[g]) = \min(p^-[\min(p^+[g], q^+[g])], q^-[\min(p^+[g], q^+[g])])$$

Proof. "$\geqslant$" follows by the fact that $p^+[g] \geqslant \min(p^+[g], q^+[g])$ and $q^+[g] \geqslant \min(p^+[g], q^+[g])$ and hence Remark 35 implies

$$\begin{aligned}\min(p[g], q[g]) &= \min(p^-[p^+[g]], q^-[q^+[g]]) \\ &\geqslant \min(p^-[\min(p^+[g], q^+[g])], q^-[\min(p^+[g], q^+[g])])\end{aligned}$$

For "$\leqslant$" fix any $x \in [a, b]$. Without loss of generality, assume

$$\begin{aligned}\min(p^-[\min(p^+[g], q^+[g])](x), q^-[\min(p^+[g], q^+[g])](x)) \\ = p^-[\min(p^+[g], q^+[g])](x)\end{aligned}$$

Now distinguish two cases:

1. Assume $p^-[\min(p^+[g], q^+[g])](x) = p^+[g](y) + \int_x^y p(\kappa)d\kappa$ for some $x \leqslant y \leqslant b$. The latter is greater or equal $p^-[p^+[g]](x)$, proving the claim.

2. Assume $p^-[\min(p^+[g], q^+[g])](x) = q^+[g](y) + \int_x^y p(\kappa)d\kappa$ for some $x \leqslant y \leqslant b$ and $q^+[g](y) = g(z) + \int_z^y q(\kappa)d\kappa$ for some $a \leqslant z \leqslant y$, that is we have

$$p^-[\min(p^+[g], q^+[g])](x) = g(z) + \int_z^y q(\kappa)d\kappa + \int_x^y p(\kappa)d\kappa \quad (8)$$

If $z > x$ then (8) is greater or equal than

$$g(z) + \int_x^z p(\kappa)d\kappa \geqslant p^-[g](x) \geqslant p^-[p^+[g]](x)$$

Otherwise, if $z \leqslant x$ then (8) is greater or equal than

$$g(z) + \int_z^x q(\kappa)d\kappa \geqslant q^+[g](x) \geqslant q^-[q^+[g]](x)$$

showing the claim in both cases.

□

Now we bound the number of break points of $\min(p[g], q[g])$:

Claim 35.2. *Let $g : [a, b] \to \mathbb{R}_{\geqslant 0}$ piece-wise linear and $p, q : [a, b] \to \mathbb{R}_{\geqslant 0}$ piece-wise constant. Let P the set of break points of p and q. Define $g' := \min(p^+[g], q^+[g])$. g' has at most $2|P|$ more break points that are not contained in P than g has.*

By symmetry, the same holds for $g'' := \min(p^-[g], q^-[g])$.

Proof. It is $g' \leqslant g$. Partitioning the interval $[a, b]$ at the intersections of g and g' yields intervals

$$[a, b] = [x_1, x_2] \sqcup [x_2, x_3] \sqcup \ldots \sqcup [x_{n-1}, x_n]$$

such that for each interval $[x_i, x_{i+1}]$ it holds that either

$$g'_{|[x_i, x_{i+1}]} = g_{|[x_i, x_{i+1}]}$$

or

$$g'_{|(x_i, x_{i+1})} < g_{|(x_i, x_{i+1})} \text{ and } g'(x_i) = g(x_i) \text{ and } g'(x_{i+1}) = g(x_{i+1})$$

To bound the number of additional break points of g' that are not contained in P we only have to consider the intervals where g' is strictly smaller than g, hence let $[a', b']$ be such an interval. g' must be continuous on this interval.

First consider the set X of break points of g' in $(a', b') \setminus P$. We will consider a', b' later. Let $x \in X$ arbitrary and x' the next smaller break point in X or a' if there is none. As $a' < x < b'$ it is $g(x) > g'(x)$. Because x is not a break point of p or q and by definition of g' it must be $g'(x) = p^+[g](x) = q^+[g](x)$, otherwise x could not be a break point of g'. It follows that $p^+[g](x) = q^+[g](x) < g(x)$ and x is not a break point of $p^+[g](x)$ or $q^+[g](x)$ as well. Moreover, it is $g'(x') = p^+[g](x') = q^+[g](x')$: If $a' \neq x'$ this follows by the same argumentation as above, otherwise if $a' = x'$ this holds because always $g' \leqslant p^+[g], q^+[g] \leqslant g$ and $g'(a') = g(a')$.

Now we want to show that there must be some $y \in P$ with $y \in (x', x)$. Assume that there is no such y. Hence g' has no break points in (x', x). Because x is a break point of g' but neither of $p^+[g]$ nor $q^+[g]$ the slopes of $p^+[g]$ and $q^+[g]$ must be different at x. W.l.o.g assume $0 \leqslant p(x) = \frac{d}{d\kappa\downarrow}p^+[g](x) < \frac{d}{d\kappa\downarrow}q^+[g](x) = q(x)$. Hence $\frac{d}{d\kappa\uparrow}g'(x) = \frac{d}{d\kappa\downarrow}q^+[g](x)$ and since g' has no break points in (x', x) it is

$$\frac{d}{d\kappa\downarrow}g'(x') = \frac{d}{d\kappa\uparrow}g'(x) = \frac{d}{d\kappa\downarrow}q^+[g](x)$$

Because $g'(x') = p^+[g](x')$ it is $\frac{d}{d\kappa\downarrow}g'(x') <= \frac{d}{d\kappa\downarrow}p^+[g](x') = p(x')$ by definition of g'. Altogether, we get $p(x) < q(x) \leqslant p(x')$ which is a contradiction because p has no break points in $(x', x]$. Hence there is $y \in P$ with $y \in (x', x)$ and it follows that to each $x \in X$ we can assign a unique $p \in P$. Therefore, g' has at most $|P|$ break points that are not break points of g, p and q and at which g is strictly larger than g'.

Finally look at a' and b'. a' must be a break point of g or p or q because $g'(a') = g(a')$ and $g'_{|(a',b')} < g_{|(a',b')}$. If a' was not a break point we would obtain a contradiction to the definition of $p^+[g]$ and $q^+[g]$. Hence we do not have to account for a'. Consider the case that b' is a break point of g' but not of g and $b' \notin P$.

First consider the case that $(a', b') \cap P = \emptyset$. We want to show that g has a break point x in (a', b'): Assume g has no break point in (a', b'). We saw earlier that then g' cannot have a break point in (a', b') neither as we assumed $(a', b') \cap P = \emptyset$. This is a contradiction to the fact that $g(a') = g'(a')$ and $g(b') = g'(b')$ and g is strictly larger than g' in (a', b'). So let $x \in (a', b')$ a break point of g. We can now assign x to b' as a break point of g that is not a break point of g', canceling out the break point b' of g'.

Now consider the case $(a', b') \cap P \neq \emptyset$. We take any $x \in (a', b') \cap P$ and assign it to b'. We might have already assigned x to a break point of g' in (a', b'), so we bound the total number of such break points b' by an additional summand of $|P|$.

As all break points of g' that are not break points of g and not contained in P can be covered by an element in P such that each $x \in P$ is assigned to at most two break points, the number of such break points of g' can be bounded by $2|P|$. □

Claim 35.3. *Let* $g : [a, b] \to \mathbb{R}_{\geqslant 0}$ *piece-wise linear and* $p, q : [a, b] \to \mathbb{R}_{\geqslant 0}$ *piece-wise constant. Let* P *the set of break points of* p *and* q*. Then* $\min(p[g], q[g])$ *has at most* $4|P|$ *more break points that are not contained in* P *than* g *has.*

Proof. By Claim 35.1 it is

$$\min(p[g], q[g]) = \min\left(p^-[\min(p^+[g], q^+[g])], q^-[\min(p^+[g], q^+[g])]\right)$$

The statement follows by applying Claim 35.2 on

$$g' := \min(p^+[g], q^+[g])$$

and then on $\min(p^-[g'], q^-[g'])$. □

Now we conclude the proof of Lemma 34. For this we ignore break points that are contained in $B^c \cup B^p \cup \{a, b\}$ but instead bound their number by $|B^c| + |B^p| + 2$ beforehand. The sum of two piece-wise linear functions has no break points additional to those of the two summands. We bound the number of occurences a function s_v has a break point that none of its children's functions had, subtracting the number of break points a children's function had but which is no longer a break point of s_v. If we bound this number by $2|T| + 4|B^p| \cdot (|C| - 1)$ this proves the statement, as all the functions s_v are computed in reverse topological order from each other.

Clearly, any interval border of a node in T can introduce a new break point, whose number we bound by $2|T|$. Adding the via cost function in line 2 of Algorithm 1 only adds break points in $B^c \cup \{a, b\}$, which we ignore at this point. Now look at the following function in line 2 of Algorithm 1

$$\min_{\substack{x' \in \square(v) \\ \alpha \in \{v, w\}}} \left(\left| \int_x^{x'} p_{e,\alpha}(\kappa) d\kappa \right| + s_v(x') \right) \tag{9}$$

which is computed by running Algorithm 2 twice as described in Lemma 32, once for $\alpha = v$ and $\alpha = w$ and computing the pointwise minimum. First note that for a child v of u in G that is not contained in C, i.e. $\square(u) \cap \square(v) = \emptyset$, function (9) has no break points other than possibly B^p on the whole domain $\square(u)$. Hence no break points outside of B^p are introduced and we now restrict ourselves to children that are also contained in C.

By Claim 35.3 at most $4|B^p|$ additional break points can arise in (9) during the computation of $s_{p(u)}$ for any node $u \in C$. C is an arborescence, thus there are $|C| - 1$ nodes in C that have a parent in C and we can bound the total number of break points added in this way by $4|B^p| \cdot (|C| - 1)$ proving the statement.

□

Corollary 36. *Assume we are given an instance $(G, \square, p, c)$ of the one-dimensional minimum cost arborescence with fixed topology problem with layer-embedded or planar congestion price functions. Let*

$$G' := (V(G), \{(v, w) \in E(G) : \square(v) \cap \square(w) \neq \emptyset\}$$

Let C a connected component of G' and $[a, b] \subset \mathbb{R}$ and $T \subset C$ the number of terminals within C. For all $v \in C$ and with layer-embedded congestion *price functions, the number of break points of the function s_v computed in Algorithm 1 can be bounded by*

$$2|\mathcal{L}| + 2|T| + 4|C| - 2$$

With planar congestion price functions, it can be bounded by

$$2|T| + 4|C|$$

Proof. With the layer-embedded or planar congestion costs, the move-bound intervals $\Box$ are defined as the x- or y-projections of the global routing tiles. Therefore, all nodes in the component C must be located in the same global routing tile column or row, depending on whether the given instance is the x- or y-projection of the original two-dimensional instance. Hence, in the setting of Lemma 34, B^p consists of the unique break point of the congestion price function within the global routing tile column or row. The via cost functions from Definition 26 or 28 can have two break points with a specific tip-to-tip penalty. With layer-embedded congestion costs, these tip-to-tip penalty lengths can be different depending on the layer of the node in the input, whereas the planar congestion costs always assume the cheapest layer. Hence we have $|B^c| \leqslant 2|\mathcal{L}|$ for the layer-embedded congestion costs and $|B^c| \leqslant 2$ for the planar congestion price functions. Only terminals can have intervals that are not equal to the whole range of the respective global routing tile, hence we bound $|T|$ by the number of terminals. Plugging this into the statement of Lemma 34 yields the claim. $\Box$

In the worst case the number of break points can be linear in the number of nodes resulting in total quadratic running time. However, in practice the number of break points is small as Figure 26 shows. Most instances in global routing attain a maximum of ten to twenty break points, without any extreme outliers. By modeling the cost functions of the sub-arborescences as piece-wise continuous functions bad solution candidates for the nodes are pruned away automatically. This is in contrast to an approach in which solution candidates are computed on a Hanan grid (Section 3.3.3). If running time should become an issue it would also be possible to only store approximations of the piece-wise linear functions with a bounded number of break points.

3.3.1.5 *Delay Costs*

In a timing-driven routing context it is important to have a routing that is not only short and optimized with regard to congestion but also has good timing properties.

Definition 37 (Downstream capacitance lower bound)**.** Let $C_{\min}$ the minimum unit capacity over the layers. Given an instance $(A, \Box, p, c)$ of the minimum cost arborescence with fixed topology problem we

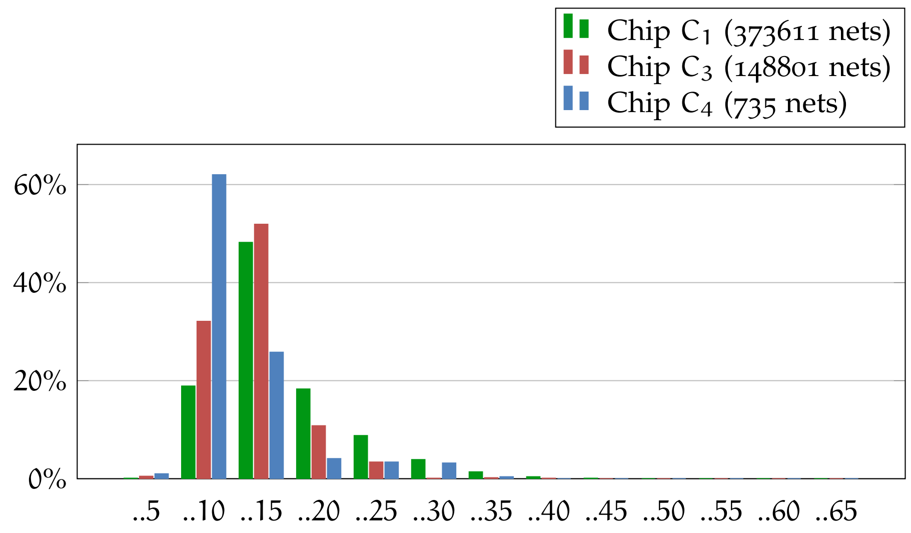


Figure 26: Percentages of instances with certain maximum numbers of break points that occur in Algorithm 1 during global routing on three chip instances. On these instances, there never occurred more than 65 break points.

define a lower bound on the downstream capacitance for all $w \in V(A)$:

$$C_{\min}(Y_w) := \min_{\substack{f:V(Y_w)\to\mathbb{R}^2 \\ f(v)\in\Box(v)\forall v\in V(Y_w)}} \sum_{e=(u,v)\in E(Y_w)} C_{\min}\cdot|f(u),f(v)|$$

Since $C_{\min}(Y_w)$ is defined as the minimum possible length of the sub-arborescence Y_w respecting the move bounds $\Box$ multiplied by the minimum unit capacity it is a lower bound on the downstream capacity $C_{\text{down}}(w)$ for any feasible positioning $f : V(Y_w) \to \mathbb{R}^2$. It can be computed efficiently using the dynamic program for the congestion-unaware x-y-optimization from Section 3.3.1.3.

Having projected an instance of the minimum cost arborescence with fixed topology problem to two instances of the one-dimensional minimum cost arborescence with fixed topology problem, we can add a lower bound delay cost to the edge prices:

Definition 38 (Layer-embedded congestion costs with lower-bound delay costs)**.** Let G be an arborescence embedded into $\mathcal{C} \times \mathcal{L}$ and p^x, p^y and c^x, c^y the layer-embedded congestion cost functions for two instances of the one-dimensional minimum cost arborescence with fixed topology problem as in Definition 26. For $w \in V(G)$ let $C_{\min}(Y_w)$ the lower bound on the downstream capacitance at w from Definition 37. Let $R(z)$ the unit resistance on layer $z \in \mathcal{L}$. For an edge in x-dimension $e = (v, w) = ((\overline{x_1}, \overline{y}, \overline{z}), (\overline{x_2}, \overline{y}, \overline{z})) \in E(G)$ define

$$p_e^{x,\text{delay}}(\_, \overline{y}, \overline{z}) := p_e^x(\_, \overline{y}, \overline{z}) + R(\overline{z}) \cdot C_{\min}(Y_w)$$

Analogously, for an edge in y-dimension $e = (v, w) = ((\overline{x}, \overline{y_1}, \overline{z}), (\overline{x}, \overline{y_2}, \overline{z}))$ define

$$p_e^{y,\text{delay}}(\overline{x}, \_, \overline{z}) := p_e^y(\overline{x}, \_, \overline{z}) + R(\overline{z}) \cdot C_{\min}(Y_w)$$

For $v \in V(G)$ define

$$c_v^{x,\text{delay}} := c_v^x \quad \text{and} \quad c_v^{y,\text{delay}} := c_v^y$$

Remark 39. Let A be a local route embedded into $\mathcal{C} \times \mathcal{L}$ and assume $\text{dist}_{\text{tip}}(z) = 0$ for all layers $z \in \mathcal{L}$. Then the sum of the layer-embedded congestion costs with lower-bound delay costs of all edges and nodes of the two instances of the one-dimensional minimum cost arborescence with fixed topology problem for the underlying arborescence is at most the sum of the intersection-ignoring congestion cost and delay cost of A:

$$\begin{aligned}
&\sum_{\substack{e=((\overline{x_1},\overline{y},\overline{z}),(\overline{x_2},\overline{y},\overline{z}))\in E(A)\\ \overline{x_1}\neq\overline{x_2}}} \left| \int_{\overline{x_1}}^{\overline{x_2}} p_e^{x,\text{delay}}(\kappa)d\kappa \right| \\
+ &\sum_{\substack{e=((\overline{x},\overline{y_1},\overline{z}),(\overline{x},\overline{y_2},\overline{z}))\in E(A)\\ \overline{y_1}\neq\overline{y_2}}} \left| \int_{\overline{y_1}}^{\overline{y_2}} p_e^{y,\text{delay}}(\kappa)d\kappa \right| \\
&+ \sum_{v=(\overline{x},\overline{y},\overline{z})\in V(A)} c_v^{x,\text{delay}}(\overline{x}) + c_v^{y,\text{delay}}(\overline{y}) \\
&\leqslant \text{cost}_{\text{cong}^+}(A) + \text{cost}_{\text{delay}}(A)
\end{aligned}$$

The ratio between these two terms can be arbitrary large, because the delay cost can be arbitrarily larger than the lower-bound delay cost. In case there is only one terminal in the sub-arborescence Y_v of a node v and that terminal is located inside the movebound of v, there is not even a lower bound greater than zero, because v could be placed on top of the terminal. Hence an optimum solution with regard to the layer-embedded congestion costs with lower-bound delay costs, as it can be computed by Algorithm 1, is not a constant factor approximation with regard to congestion and delay costs.

Analogously to Definition 38 we define lower-bound delay costs for the planar congestion costs:

Definition 40 (Planar congestion costs with lower-bound delay costs)**.** Let G be an arborescence embedded into $\mathcal{C}$ and p^x, p^y and c^x, c^y the planar congestion cost functions for two instances of the one-dimensional minimum cost arborescence with fixed topology problem as in Definition 28. For $w \in V(G)$ let $C_{\min}(Y_w)$ a lower bound on the downstream capacitance at w. Let $R_{\min}$ the minimum unit resistance over the layers. For a node $w = (\overline{x}, \overline{y}, \overline{z}) \in V(G)$ define

$$p_w^{x,\text{delay}}(\_, \overline{y}, \overline{z}) := p_w^x(\_, \overline{y}, \overline{z}) + R_{\min} \cdot C_{\min}(Y_w)$$

$$p_w^{y,\mathrm{delay}}(\bar{x},\_,\bar{z}) := p_w^y(\bar{x},\_,\bar{z}) + R_{\min} \cdot C_{\min}(Y_w)$$
$$c_w^{x,\mathrm{delay}} := c_w^x \quad \text{and} \quad c_w^{y,\mathrm{delay}} := c_w^y$$

Though there are no strong theoretical guarantees, the planar congestion costs with lower-bound delay costs yield slightly better results in practice with regard to timing.

3.3.1.6 *Experimental Results*

TIMING-UNAWARE Table 2 shows results on several chip instances for the congestion-aware x-y-optimization with planar congestion costs (see Sections 4.1 and 4.2 for details on the testbed and the metrics). It was run within BonnRouteGlobal after connecting the coarse route to the exact pin shapes within the global routing tiles. The x-y-optimization was followed by a layer embedding algorithm that we will describe later in Section 3.3.2. For comparison, there are runs of BonnRouteGlobal without any x-y-optimization and with the congestion-unaware x-y-optimization. One can see that already the congestion-unaware x-y-optimization achieves to reduce the congestion (wACE4), wire length and via number dramatically. That is because shorter routes lead to less edge resource usage and hence less detours requiring fewer vias.

The congestion-aware x-y-optimization manages to obtain an even slightly better wire length with additional benefits in congestion and via count. In particular the number of vias decreases substantially, which is partly due to the fact that the implementation favors solutions that contain less corner points (Steiner points) if the costs are identical otherwise. Low numbers of vias during global routing also benefit the number of vias in detailed routing as can be seen in the experiments in Section 4.

Cost function	wACE4		GR wire length		GR vias	
Planar cong. costs	83.2%		33.2		22723566	
Layer-embedded cong. costs	84.6%	+1.4%	33.4	+0.6%	23177438	+2.0%

Table 1: Average results over five units (B_2, B_3, C_1, C_2, C_3) comparing the x-y-optimization with planar congestion costs and layer-embedded costs. In particular the number of vias is much better with planar congestion costs.

Table 1 shows results with the layer-embedded congestion costs. Before running the x-y-optimization with layer-embedded congestion costs, the route that already connects to the pin shapes is embedded into the layers using the algorithm from Section 3.3.2. That algorithm is called a second time on the x-y-optimized route. The layer-embedded congestion costs have the major disadvantage that the orientation of the wires remains fixed. Recall that their orientation originates from the planar arborescences connecting to the pin shapes

Chip	Nets	Router	RS time		RS reroutes		wACE4		Wire length		Vias	
B_1	1732566	DLU (no x-y-opt.)	00:45:01		16575780		86.79		28.7738		16482039	
		DLU (cong. unaware x-y-opt.)	00:30:29	-32.30 %	13061217	-21.20 %	84.69	-2.10 %	26.8726	-6.61 %	15258846	-7.42 %
		DLU (cong. aware x-y-opt.)	00:30:48	-31.58 %	12269469	-25.98 %	83.16	-3.63 %	26.7896	-6.90 %	14868741	-9.79 %
B_2	1202357	DLU (no x-y-opt.)	00:12:05		4458765		86.37		20.7273		11790427	
		DLU (cong. unaware x-y-opt.)	00:13:17	+9.93 %	4511960	+1.19 %	85.21	-1.16 %	19.6423	-5.23 %	11104898	-5.81 %
		DLU (cong. aware x-y-opt.)	00:22:36	+86.89 %	6536968	+46.61 %	82.59	-3.78 %	19.5849	-5.51 %	10815995	-8.26 %
B_3	372283	DLU (no x-y-opt.)	00:05:16		2130416		83.67		6.4290		3577123	
		DLU (cong. unaware x-y-opt.)	00:05:04	-3.65 %	1833810	-13.92 %	81.87	-1.80 %	6.0898	-5.28 %	3345867	-6.46 %
		DLU (cong. aware x-y-opt.)	00:05:17	+0.37 %	1706589	-19.89 %	80.85	-2.82 %	6.0809	-5.41 %	3256918	-8.95 %
C_1	373556	DLU (no x-y-opt.)	00:04:40		6663570		86.62		2.4139		3342100	
		DLU (cong. unaware x-y-opt.)	00:03:48	-18.48 %	4719780	-29.17 %	83.41	-3.21 %	2.1340	-11.60 %	2940100	-12.03 %
		DLU (cong. aware x-y-opt.)	00:04:37	-1.31 %	4339666	-34.87 %	82.61	-4.01 %	2.1321	-11.67 %	2860501	-14.41 %
C_2	245235	DLU (no x-y-opt.)	00:01:10		810072		79.68		3.0571		2052029	
		DLU (cong. unaware x-y-opt.)	00:01:29	+28.48 %	890484	+9.93 %	77.17	-2.51 %	2.8825	-5.71 %	1929154	-5.99 %
		DLU (cong. aware x-y-opt.)	00:01:53	+62.21 %	864423	+6.71 %	75.70	-3.98 %	2.8802	-5.79 %	1883992	-8.19 %
C_3	147877	DLU (no x-y-opt.)	00:00:57		486993		78.53		2.1324		1234789	
		DLU (cong. unaware x-y-opt.)	00:00:54	-6.07 %	420697	-13.61 %	76.68	-1.85 %	2.0401	-4.33 %	1178705	-4.54 %
		DLU (cong. aware x-y-opt.)	00:01:00	+4.24 %	414044	-14.98 %	76.52	-2.01 %	2.0392	-4.37 %	1155236	-6.44 %
Summary		DLU (no x-y-opt.)	01:09:12		31125596		83.61		63.5335		38478507	
		DLU (cong. unaware x-y-opt.)	00:55:04	-20.41 %	25437948	-18.27 %	81.51	-2.10 %	59.6613	-6.09 %	35757570	-7.07 %
		DLU (cong. aware x-y-opt.)	01:06:13	-4.31 %	26131159	-16.05 %	80.24	-3.37 %	59.5069	-6.34 %	34841383	-9.45 %

Table 2: The dynamic local usage without x-y-optimization, congestion-unaware x-y-optimization and the congestion-aware x-y-optimization with planar congestion costs on 7nm instances.

Chip	Nets	Router	RS time		RS reroutes		wACE4		Wire length		Vias		SNS		WS	
D_1	215089	DLU (timing-unaware x-y-opt)	00:06:46		2004232		86.35		2.0998		1647039		-663156		-202	
		DLU (timing-aware x-y-opt.)	00:06:55	+2.22 %	1997283	-0.35 %	86.36	0.01 %	2.0995	-0.01 %	1646600	-0.03 %	-660693	-0.37 %	-202	0
D_2	184853	DLU (timing-unaware x-y-opt)	00:05:02		1555380		82.15		2.1011		1361490		-58153		-62	
		DLU (timing-aware x-y-opt.)	00:04:39	-7.73 %	1551557	-0.25 %	82.30	0.15 %	2.1012	+0.00 %	1361437	+0.00 %	-57759	-0.68 %	-62	0
D_3	146458	DLU (timing-unaware x-y-opt)	00:04:03		1382280		84.48		1.3389		1109144		-77952		-74	
		DLU (timing-aware x-y-opt.)	00:04:06	+0.98 %	1377187	-0.37 %	84.60	0.12 %	1.3388	-0.01 %	1109560	+0.04 %	-77805	-0.19 %	-74	0
D_4	87766	DLU (timing-unaware x-y-opt)	00:03:19		682245		84.56		0.8593		759305		-14186		-66	
		DLU (timing-aware x-y-opt.)	00:03:16	-1.61 %	678487	-0.55 %	84.45	-0.11 %	0.8600	+0.08 %	759051	-0.03 %	-14168	-0.12 %	-66	0
D_5	79065	DLU (timing-unaware x-y-opt)	00:02:19		550981		82.31		0.8489		594660		-259161		-320	
		DLU (timing-aware x-y-opt.)	00:02:28	+6.51 %	547745	-0.59 %	82.25	-0.06 %	0.8488	-0.01 %	594491	-0.03 %	-259127	-0.01 %	-320	0
D_6	43477	DLU (timing-unaware x-y-opt)	00:00:59		335739		82.67		0.3303		296172		-9540		-200	
		DLU (timing-aware x-y-opt.)	00:01:03	+5.88 %	335524	-0.06 %	82.48	-0.19 %	0.3303	+0.00 %	295979	-0.07 %	-9515	-0.26 %	-200	0
D_7	13764	DLU (timing-unaware x-y-opt)	00:00:33		109410		81.27		0.0918		102501		-1505		-42	
		DLU (timing-aware x-y-opt.)	00:00:35	+6.24 %	109220	-0.17 %	81.21	-0.06 %	0.0918	+0.00 %	102400	-0.10 %	-1473	-2.16 %	-42	1
Summary		DLU (timing-unaware x-y-opt)	00:23:05		6620267		83.40		7.6701		5870311		-1083652		-320	
		DLU (timing-aware x-y-opt.)	00:23:05	-0.04 %	6597003	-0.35 %	83.38	-0.02 %	7.6704	+0.00 %	5869518	-0.01 %	-1080540	-0.29 %	-320	0

Table 3: The dynamic local usage with planar congestion costs within the x-y-optimization on 5nm instances; once without timing costs and once with the lower-bound delay costs.

within the global routing tiles, not considering congestion and layers of the pin shapes. In contrast to the layer-embedded congestion prices the planar congestion prices allow and favor diagonal edges where possible. This enables the layer embedding algorithm (Section 3.3.2) to choose the optimum layers with regard to congestion and via costs, greatly reducing the number of vias. Pin shapes can be accessed much more efficiently. This is why in all experiments of Section 4 the planar congestion costs are used.

timing-aware Table 3 shows results for the congestion-aware x-y-optimization with planar congestion costs and lower-bound delay costs. As for the planar congestion costs without lower-bound delay costs the route is embedded into the layers after its x-y-coordinates have been optimized. One can see a slight improvement regarding the sum of negative slacks. Moreover, the number of resource sharing reroutes drops throughout the whole testbed supporting the fact that the routes are better optimized with regard to timing and hence behave more harmonically with the other timing-aware algorithms of BonnRouteGlobal.

3.3.2 *Optimizing Layers and Wire Types*

The last step during the computation of a local route is to embed the x- and y-optimized arborescence into layers. The traditional router keeps the layers of wires that cross global routing tile borders fixed as they were chosen in the coarse route by the path search. All tile-internal wires that were added to connect to the exact pin shapes are heuristically assigned to the layers, taking into account the layers of the pin shapes and the allowed layer ranges of the nets. As tile-internal wires do not count towards congestion with the traditional router, their assigned layers do not affect congestion. However they do affect timing. With the dynamic local usage also tile-internal wires affect congestion, making it necessary to embed them into layers more carefully. [53] describes a dynamic programming approach that finds an almost optimum layer embedding with regard to congestion and timing, improving on a result from [45]. Here we present an improved faster version of [53] that can additionally optimize wire types.

So far we assumed all wires to have the same width and the same properties with regard to timing propagation. In fact, wires can be assigned different so-called wire types. A wire type defines the width of a wire, spacing constraints and timing properties like resistance and capacitance. Usually a thicker wire type has less resistance, but more capacitance and space consumption than a thinner wire type. For a timing critical net it might be beneficial to choose a thicker wire type in order to achieve a better signal delay due to the smaller resistance. The path search algorithm of BonnRouteGlobal (Section 3.2) is not able to choose between different wire types but only uses the default wire type of a net, which is given as input to BonnRouteGlobal. However, it can still be beneficial to choose among different wire types during the optimization of the local route.

Definition 41. We say that a rectilinear two-dimensional arborescence $Y = (V, E)$ with $V \subset \mathcal{C} \subset \mathbb{R}^2$ connects a net n, if Y connects the two-dimensional projections of all pins of n, that is for each pin $p \in n$ there is $v \in V$ with $v \in \text{proj}^{xy}(p)$, and Y is oriented from the source to the sinks of n.

Using the previous definition we define the problem as follows, extending the problem defined in [53] for layer optimization only.

Problem 42. LAYER AND WIRE TYPE ASSIGNMENT

Input: A rectilinear two-dimensional arborescence Y for a net n with source v_{source} and sinks $T \subset n$, a source resistance $R(v_{\text{source}}) \in \mathbb{R}_{\geqslant 0}$, sink capacitances $C : T \to \mathbb{R}_{\geqslant 0}$, timing prices $\text{cost}_{\text{delay}} : T \to \mathbb{R}_{\geqslant 0}$, a set of layers $\mathcal{L} = \{l_{\min}, \dots, l_{\max}\}$ and a set of wire types $\mathcal{W}$, a non-empty set of allowed layers $L_e \subset \mathcal{L}$ and allowed wire types $\mathcal{W}_e \subset \mathcal{W}$ for each $e \in E(Y)$, layer- and wire type-dependent wire and via congestion costs $\text{cost}_{\text{wire}} : E(Y) \times \mathcal{L} \times \mathcal{W} \to \mathbb{R}_{\geqslant 0}$, $\text{cost}_{\text{via}} : V(Y) \times \mathcal{L} \times \mathcal{W} \to \mathbb{R}_{\geqslant 0}$, and wire resistance and capacitance values $R, C : E(Y) \times \mathcal{L} \times \mathcal{W} \to \mathbb{R}_{\geqslant 0}$ and via resistances $R : \mathcal{L} \times \mathcal{L} \to \mathbb{R}_{\geqslant 0}$.

Output: A layer and wire type assignment $z : E(Y) \to \mathcal{L}$, $\omega : E(Y) \to \mathcal{W}$ with $z(e) \in L_e$ and $\omega(e) \in \mathcal{W}_e$ for all $e \in E(Y)$ such that incident edges on the same layer are assigned the same wire type, minimizing

$$\text{cost}(Y^{z,\omega}) := \text{cost}_{\text{delay}}(Y^{z,\omega}) + \sum_{e \in E(Y^{z,\omega})} \text{cost}_{\text{wire}}(e) + \sum_{v \in V(Y^{z,\omega})} \text{cost}_{\text{via}}(v)$$

where $Y^{z,\omega}$ is the route arising from the arborescence Y and the layer assignment z and the wire types ω by lifting the edges of Y to the chosen layers and adding vias to keep connectivity. Wire types for vias are determined implicitly by ω so that they match the wire type of incident wires. For stacked vias we choose the thinnest (that is cheapest) wire type.

We are looking for a layer assignment such that the sum of the wire, via and timing costs (Definition 17) of the resulting route is minimum. Wire types can only be changed at vias, which is why incident wires on the same layer have to share the same wire type. Vias have very small capacitances which is why we ignore their capacitance in this section, and mostly also in the implementation, for simplicity. Moreover, we currently do not consider wire type-dependent resistances for vias but instead always assume the resistance of the smallest default wire type which is slightly pessimistic. To make this more precise one would have to check for compatibility of via wire types with the wire types chosen on the layer above and below, which we also want to avoid for simplicity.

Tip-to-tip penalty costs as defined in Definition 15 can be easily incorporated by adjusting the edge and via congestion costs like it was also done for path search in Section 3.2. For each node $v \in V(Y)$, layer $z \in \mathcal{L}$ and wire type $w \in \mathcal{W}$ we let $\text{cost}_{\text{via}}(v, z, w)$ the congestion cost of a stacked via pad at node v on layer z with wire type w including tip-to-tip penalty extension costs in both directions. For each edge $e \in E(Y)$, layer $z \in \mathcal{L}$ and wire type $w \in \mathcal{W}$ we let $\text{cost}_{\text{wire}}(e, z, w)$ the congestion cost of the wire area of e on layer z with wire type w, subtracted by the cost of one tip-to-tip penalty extension at each end of the wire. For details, see Section 3.2.

[53] showed that even with $|\mathcal{W}| = 1$ the layer and wire type assignment problem is $\mathcal{NP}$-hard using a reduction from [50] from the $\mathcal{NP}$-hard partition problem. The difficulty of the layer and wire type assignment problem lies in the fact that the delay through an edge $e = (u, v)$ depends on the capacitances and layers of all edges in the sub-arborescence Y_v. Here we describe a dynamic program that approximately solves the layer and wire type assignment problem by finding an optimum solution with regard to an approximation of $\text{cost}_{\text{delay}}$ based on lower bounds for the capacitances of sub-arborescences. The following definitions are based on [53] and extended by wire types. We always assume an instance of the layer and wire type assignment problem to be given.

Definition 43. For $e \in E(Y)$ define minimum edge resistances and capacitances

$$R_{\min}(e) := \min_{\substack{z \in L_e \\ w \in W_e}} R(e, z, w) \qquad C_{\min}(e) := \min_{\substack{z \in L_e \\ w \in W_e}} C(e, z, w)$$

For $v \in V(Y)$ derive a lower bound on the downstream capacitance of the sub-arborescence Y_v as

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

We define approximate delay costs based on the capacitance lower bounds, which constitute a lower bound on the Elmore delay costs in Definition 8:

Definition 44. For $e = (u, v) \in E(Y)$, $z \in L_e$ and $w \in W_e$ define

$$\text{rcp}_{\text{wire}}(e, z, w) := \text{cost}_{\text{delay}}(T(Y_v)) \cdot R(e, z, w) \cdot \left(\frac{C(e, z, w)}{2} + C_{\min}(Y_v)) \right)$$

$\mathrm{rcp}_{\mathrm{wire}}(e, z, w)$ is smaller than the actual Elmore delay price induced by this edge if edges in the sub-arborescence Y_v are assigned to layers and wire types on which they have larger capacitance than the minimum capacitance. To partially compensate for this error, we consider lower bounds on the resistance in a path from the source to another node:

Definition 45. Let $l_{\mathrm{source}} \in \mathcal{L}$ the predefined layer of the source terminal. For $e = (u, v) \in E(Y)$ and $z \in L_e$ define the capacitance price lower bound as

$$\begin{aligned} \mathrm{cp}_{\mathrm{lb}}(e, z) := \mathrm{cost}_{\mathrm{delay}}(T) \cdot R(v_{\mathrm{source}}) \qquad & (10)\\ + \mathrm{cost}_{\mathrm{delay}}(T(Y_v)) \sum_{z'=\min\{l_{\mathrm{source}}, z\}}^{\max\{l_{\mathrm{source}}, z\}-1} R(z', z'+1) \qquad & (11)\\ + \sum_{(x,y) \in P_{[v_{\mathrm{source}}, u]}} \mathrm{cost}_{\mathrm{delay}}(T(Y_y)) \cdot R_{\min}((x, y)) \qquad & (12) \end{aligned}$$

Here $P_{[v_{\mathrm{source}}, u]}$ denotes the unique path from the source v_{source} to u in the arborescence Y.

In other words, if in any layer assignment $z : E(Y) \to \mathcal{L}$ an edge e is moved from layer $z(e)$ to layer z' increasing its capacitance by c, the total cost of the route would increase by at least $c \cdot \mathrm{cp}_{\mathrm{lb}}(e, z')$: The additional capacitance c increases the delay through the root (10), through a minimum number of vias to reach the source layer (11) and through the upstream path edges from v to v_{source} (12). The same happens if e is assigned another wire type increasing its capacitance by c.

Definition 46. The additional capacitance that an edge $e \in E(Y)$ has on layer $z \in L_e$ with wire type $w \in W_e$ compared to the lower bound is defined as

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

Multiplying C_{corr} by $\mathrm{cp}_{\mathrm{lb}}$ yields a lower bound on the delay costs that are missing in $\mathrm{rcp}_{\mathrm{wire}}$:

Definition 47. For $e \in E(Y)$, $z \in L_e$ and $w \in W_e$ define

$$\mathrm{rcp}_{\mathrm{corr}}(e, z, w) := \mathrm{cp}_{\mathrm{lb}}(e, z) \cdot C_{\mathrm{corr}}(e, z, w)$$

Altogether we obtain the following cost functions:

Definition 48. For $e \in E(Y)$, $z \in L_e$ and $w \in W_e$ define

$$\mathrm{rcp}(e, z, w) := \mathrm{cost}_{\mathrm{wire}}(e, z, w) + \mathrm{rcp}_{\mathrm{wire}}(e, z, w) + \mathrm{rcp}_{\mathrm{corr}}(e, z, w)$$

Given a layer and wire type assignment $z : E(Y) \to \mathcal{L}$ and $\omega : E(Y) \to \mathcal{W}$ and $v \in V(Y)$, let (u_1, w_1), ..., $(u_k, w_k) \in E(Y^{z,\omega})$ the via edges at v in the local route $Y^{z,\omega}$ and $[l_1, l_2]$ their layer range. Define via costs

$$\mathrm{rcp}(v, z, \omega) := \sum_{l=l_1}^{l_2} \mathrm{cost}_{\mathrm{via}}(v, l, \omega)$$
$$+ \sum_{i=1}^{k} R\left(\mathrm{proj}^z(u_i), \mathrm{proj}^z(w_i)\right) \cdot \mathrm{cost}_{\mathrm{delay}}\left(T(Y^{z,\omega}_{w_i})\right) \cdot C_{\min}(Y^{z,\omega}_{w_i})$$

Finally we define

$$\begin{aligned}\mathrm{rcp}(z, \omega) :=& \mathrm{cost}_{\mathrm{delay}}(T) \cdot R(v_{\mathrm{source}}) \cdot C_{\min}(Y)\\ &+ \sum_{v \in V(Y)} \mathrm{rcp}(v, z, \omega) + \sum_{e \in E(Y)} \mathrm{rcp}(e, z(e), \omega(e))\end{aligned}$$

We want to compute a layer and wire type assignment z, ω minimizing $\mathrm{rcp}(z, \omega)$. To quantify its quality we need the following analog from Definition 45:

Definition 49. Given a layer and wire type assignment z, ω, define the capacitance price for a non-via edge $e = (u, v) \in E(Y^{z,\omega})$ as

$$\begin{aligned}\mathrm{cp}(e) :=& \mathrm{cost}_{\mathrm{delay}}(T) \cdot R(v_{\mathrm{source}})\\ &+ \sum_{\substack{(x,y)\in\\ P_{Y^{z,\omega}}[v_{\mathrm{source}},u]}} \mathrm{cost}_{\mathrm{delay}}(T(Y^{z,\omega}_y)) \cdot R\left((x,y), z(x,y), \omega(x,y)\right)\end{aligned}$$

The following theorem extends a result from [53] who showed an analogous statement for the case $|\mathcal{W}| = 1$ with a slightly slower running time of $\mathcal{O}(|E(Y)| \cdot |\mathcal{L}|^2$.

Theorem 50. *Given an instance Y of the layer and wire type assignment problem, let z, ω be a layer and wire type assignment minimizing $rcp(z, \omega)$ and $z*, \omega*$ minimizing $cost(Y^{z*,\omega*})$. Then it holds that*

$$cost(Y^{z,\omega}) \leqslant (1 + (\alpha - 1)(1 - \beta)) \cdot cost(Y^{z*,\omega*})$$

where

$$\alpha := \max_{v \in V(Y^{z,\omega})} \frac{C(Y^{z,\omega}_v)}{C_{min}(Y^{z,\omega}_v)} \qquad \beta := \min_{e \in E(Y^{z,\omega})} \frac{cp_{lb}(e)}{cp(e)}$$

Given that Y has bounded vertex degrees such a layer and wire type assignment z, ω can be computed in time $\mathcal{O}(|E(Y)| \cdot |\mathcal{L}| \cdot |\mathcal{W}|)$ using dynamic programming.

The approximation factor is good if the wire capacitances do not vary too much over the layers and wire types, or the capacitance prices of $Y^{z,\omega}$ are not much bigger than the lower bound capacitance prices from Definition 45. The latter is the case if $Y^{z,\omega}$ does not contain too many vias and the wire resistances do not vary much over the layers and wire types.

In current chip technologies, resistances can vary by up to a factor of several thousands over all layers and wire types making it difficult to find a useful bound on β. However, capacitances vary much less, roughly by a factor of 2. Because it holds that

$$\alpha \leqslant \max_{e \in E(Y), z \in \mathcal{L}, w \in \mathcal{W}} \frac{C(e,z,w)}{C_{\min}(e)}$$

Theorem 50 implies a 2-approximation in practice. We obtain an even better bound if layers only are optimized and the default wire type is chosen for all wires. On many smaller chip instances the highest layers are not available and even if they are, only a part of the nets is allowed to use them. Excluding the highest layers yields a bound of α near 1, implying an almost exact algorithm.

The huge differences of the resistance values over the layers point out the importance of considering timing when assigning the wires to layers, as the choice of layers has a huge impact on the signal delay. If all timing costs are zero the dynamic program finds an optimum solution with regard to wire and via congestion costs.

We now proof Theorem 50:

Proof. The approximation ratio can be shown by proving

$$\begin{aligned}
\mathrm{cost}(Y^{z,\omega}) &\leqslant (1 + (\alpha - 1)(1 - \beta)) \cdot \mathrm{rcp}(z, \omega) \\
&\leqslant (1 + (\alpha - 1)(1 - \beta)) \cdot \mathrm{rcp}(z*, \omega*) \\
&\leqslant (1 + (\alpha - 1)(1 - \beta)) \cdot \mathrm{cost}(Y^{z*,\omega*})
\end{aligned}$$

The second inequality is satisfied by the choice of z, ω and the third because rcp is a lower bound on cost. We omit the details of the proof for the first inequality because it works exactly the same as the proof in [53] for the case that $|\mathcal{W}| = 1$ except that different wire types have to be considered in all calculations.

To compute a solution minimizing $\mathrm{rcp}(z, \omega)$ we use Algorithm 3 which is a modification of the dynamic program in [53]. It computes partial candidates $\mathrm{cand}(e, l, w)$ for all edges $e \in E(Y)$, layers $l \in L_e$ and wire types $w \in W_e$.

Definition 51. Given $e = (u, v) \in E(Y), l \in L_e, w \in W_e$, $\mathrm{cand}(e, l, w)$ denotes the partial costs of a layer and wire type assignment $z : E(Y_v) \cup \{e\} \to \mathcal{L}$ and $\omega : E(Y_v) \cup \{e\} \to \mathcal{W}$ for which $z(e) = l$ and $\omega(e) = w$, minimizing

$$\sum_{v' \in V(Y_v)} \mathrm{rcp}(v', z, \omega) + \sum_{e' \in E(Y_v) \cup \{e\}} \mathrm{rcp}(e', z(e'), \omega(e'))$$

If additionally given an ordering $e_1 \geqslant ... \geqslant e_k$ of $\delta^+_Y(v)$, $\mathrm{cand}^{\geqslant}(e, l, w)$ denotes the same minimum partial costs with the additional constraint that $z(e_1) \geqslant ... \geqslant z(e_k)$.

Algorithm 3: Layer and wire type assignment dynamic program

```
   Input: Instance of the layer and wire type assignment problem.
   Output: An approximately optimal layer and wire type assignment.
1  $Y := (V(Y) \sqcup \{v_{\text{source}}\}, E(Y) \cup \{(v'_{\text{source}}, v_{\text{source}})\})$ where $v'_{\text{source}}$ is the copy of $v_{\text{source}}$
2  for $e = (u,v) \in E(Y)$ in reverse topological order do
3      foreach ordering $e_1 = (u_1, v_1) \geqslant \dots \geqslant e_k = (u_k, v_k)$ of $\delta^+_Y(v)$ do
           // Compute upper partial candidates
4          foreach $i \in \{0, \dots, k\}$ do
5              $\mathrm{OPT}^{\downarrow}(i, l_{\max}, w) := \mathrm{cost}_{\text{via}}(l_{\max}, w) + \sum_{j=1}^{i} \mathrm{cand}(e_j, l_{\max}, w) \quad \forall w \in \mathcal{W}$
6              $\alpha_i := \sum_{j=1}^{i} \left(C_{\min}(e_j) + C_{\min}(Y_{v_j})\right) \cdot \sum_{j=1}^{i} \mathrm{cost}_{\text{delay}}(Y_{v_j})$
7          for $l = l_{max} - 1, \dots, l_{min}$ do
8              $\mathrm{OPT}^{\downarrow}(0, l, w) := \mathrm{cost}_{\text{via}}(l, w) \quad \forall w \in \mathcal{W}$
9              for $i = 1, \dots, k$ do
10                 $\mathrm{OPT}^{\downarrow}(i, l, w) :=$
                   $\min \begin{Bmatrix} \min\limits_{w' \in \mathcal{W}} \left(\mathrm{OPT}^{\downarrow}(i, l+1, w')\right) + \mathrm{cost}_{\text{via}}(l, w) + R(l, l+1) \cdot \alpha_i \\ \mathrm{OPT}^{\downarrow}(i-1, l, w) + \mathrm{cand}(e_i, l, w) \end{Bmatrix} \quad \forall w \in \mathcal{W}$
           // Compute lower partial candidates
11         foreach $i \in \{0, \dots, k\}$ do
12             $\mathrm{OPT}_{\uparrow}(i, l_{\min}, w) := \mathrm{cost}_{\text{via}}(l_{\min}, w) + \sum_{j=i}^{k} \mathrm{cand}(e_j, l_{\min}, w) \quad \forall w \in \mathcal{W}$
13             $\beta_i := \sum_{j=i}^{k} \left(C_{\min}(e_j) + C_{\min}(Y_{v_j})\right) \cdot \sum_{j=i}^{k} \mathrm{cost}_{\text{delay}}(Y_{v_j})$
14         for $l = l_{min} + 1, \dots, l_{max}$ do
15             $\mathrm{OPT}_{\uparrow}(k+1, l, w) := \mathrm{cost}_{\text{via}}(l, w) \quad \forall w \in \mathcal{W}$
16             for $i = k, \dots, 1$ do
17                 $\mathrm{OPT}_{\uparrow}(i, l, w) :=$
                   $\min \begin{Bmatrix} \min\limits_{w' \in \mathcal{W}} \left(\mathrm{OPT}_{\uparrow}(i, l-1, w')\right) + \mathrm{cost}_{\text{via}}(l, w) + R(l-1, l) \cdot \beta_i \\ \mathrm{OPT}_{\uparrow}(i+1, l, w) + \mathrm{cand}(e_i, l, w) \end{Bmatrix} \quad \forall w \in \mathcal{W}$
           // Compose upper and lower partial candidates
18         foreach $l \in L_e$ do
19             $\mathrm{cand}^{\geqslant}(e, l, w) := \mathrm{rcp}(e, l, w)$
                 $+ \min\limits_{i=0,\dots,k} \left(\mathrm{OPT}^{\downarrow}(i, l, w) + \mathrm{OPT}_{\uparrow}(i+1, l, w) - \mathrm{cost}_{\text{via}}(l, w)\right) \quad \forall w \in W_e$
20     foreach $l \in L_e$ do
21         $\mathrm{cand}(e, l, w) := \min_{\text{all orderings}\geqslant} \left(\mathrm{cand}^{\geqslant}(e, l, w)\right) \quad \forall w \in W_e$
22 return $\min_{w \in \mathcal{W}} \mathit{cand}((v'_{source}, v_{source}), l_{source}, w)$
```

For $l \in \mathcal{L} \setminus L_e$ or $w \in \mathcal{W} \setminus W_e$ we set $\mathrm{cand}(e, l, w) := \infty$.

By the following remark, line 21 in Algorithm 3 is correct:

Remark 52. For all $e = (u, v) \in E(Y), l \in L_e, w \in W_e$ it is

$$\mathrm{cand}(e, l, w) = \min_{\text{all orderings} \geqslant \text{ of } \delta_Y^+(v)} (\mathrm{cand}^{\geqslant}(e, l, w))$$

To verify the computation of $\mathrm{cand}^{\geqslant}(e, l, w)$ we fix $e = (u, v) \in E(Y)$ and an ordering $\geqslant$ of the edges in $\delta_Y^+(v)$, that is $e_1 = (u_1, v_1) \geqslant \dots \geqslant e_k = (u_k, v_k)$. We define upper partial candidates that optimally embed all child edges $e_1, \dots, e_i$ using some layer range $\{l, \dots, l_{\max}\}$. They will be computed from the top layer to the bottom layer.

Definition 53 (Upper partial candidates)**.** For $l \in \mathcal{L}$ and $w \in \mathcal{W}$, $\mathrm{OPT}^{\downarrow}(i, l, w)$ denotes the partial costs of a layer and wire type assignment

$$z : \bigcup_{j=1,\dots,i} \left(E(Y_{v_j}) \cup \{e_j\}\right) \to \mathcal{L} \quad \text{and} \quad \omega : \bigcup_{j=1,\dots,i} \left(E(Y_{v_j}) \cup \{e_j\}\right) \to \mathcal{W}$$

for which $z(e_1) \geqslant \dots \geqslant z(e_i) \geqslant l$ and which contains vias at v from l to $z(e_1)$ and uses wire type w on layer l, minimizing

$$\sum_{v' \in \bigcup_{j=1,\dots,i} V(Y_{v_j})} \mathrm{rcp}(v', z) + \sum_{e' \in \bigcup_{j=1,\dots,i} E(Y_{v_j})} \mathrm{rcp}(e', z(e'), \omega(e')) + \sum_{l'=l}^{z(e_1)} \mathrm{cost}_{\mathrm{via}}(v, l', \omega) + \sum_{l'=l+1}^{z(e_1)} R\left(l'-1, l'\right) \cdot \alpha_{l'} \tag{13}$$

where for $l' \in \mathcal{L}$

$$\alpha_{l'} := \sum_{j : z(e_j) \geqslant l'} \left(C_{\min}(e_j) + C_{\min}(Y_{v_j})\right) \cdot \sum_{j : z(e_j) \geqslant l'} \mathrm{cost}_{\mathrm{delay}}\left(T(Y_{v_j})\right)$$

The first two summands in (13) cover the costs incurred by rcp except for the vias at v which are covered by the last two summands.

Analogously, lower partial candidates are defined that optimally embed all child edges $e_i, \dots, e_k$ using some layer range $\{l_{\min}, \dots, l\}$. They will be computed from the bottom layer to the top layer.

Definition 54 (Lower partial candidates)**.** For $l \in \mathcal{L}$ and $w \in \mathcal{W}$, $\mathrm{OPT}_{\uparrow}(i, l, w)$ denotes the partial costs of a layer and wire type assignment

$$z : \bigcup_{j=i,\dots,k} \left(E(Y_{v_j}) \cup \{e_j\}\right) \to \mathcal{L} \quad \text{and} \quad \omega : \bigcup_{j=i,\dots,k} \left(E(Y_{v_j}) \cup \{e_j\}\right) \to \mathcal{W}$$

for which $l \geqslant z(e_i) \geqslant \ldots \geqslant z(e_k)$ and which contains vias at v from $z(e_k)$ to l and uses wire type w on layer l minimizing its partial costs

$$\sum_{v' \in \bigcup_{j=i,\ldots,k} V(Y_{v_j})} \mathrm{rcp}(v', z) + \sum_{e' \in \bigcup_{j=i,\ldots,k} E(Y_{v_j})} \mathrm{rcp}(e', z(e'), \omega(e'))$$
$$+ \sum_{l'=z(e_k)}^{l} \mathrm{cost}_{\mathrm{via}}(v, l', \omega) + \sum_{l'=z(e_k)}^{l-1} R\left(l', l'+1\right)\right) \cdot \alpha_{l'}$$

where for $l' \in \mathcal{L}$

$$\alpha_{l'} := \sum_{j: z(e_j) \leqslant l'} \left(C_{\min}(e_j) + C_{\min}(Y_{v_j})\right) \cdot \sum_{j: z(e_j) \leqslant l'} \mathrm{cost}_{\mathrm{delay}}\left(T(Y_{v_j})\right)$$

The following claim shows correctness of line 19 in the dynamic program:

Claim 54.1. *For $l \in L_e$ and $w \in W_e$ it holds that*

$$\mathit{cand}^{\geqslant}(e, l, w) = \mathit{rcp}(e, l, w)$$
$$+ \min_{i=0,\ldots,k} \left(\mathit{OPT}^{\downarrow}(i, l, w) + \mathit{OPT}_{\uparrow}(i+1, l, w) - \mathit{cost}_{\mathit{via}}(l, w) \right)$$

Proof. Let $z : E(Y_v) \cup \{e\} \to \mathcal{L}$ and $\omega : E(Y_v) \cup \{e\} \to \mathcal{W}$ for which $z(e) = l$, $\omega(e) = w$ and which respects the ordering $\geqslant$. Let

$$i := \max\left\{j = 1, \ldots, k : z(e_j) \geqslant l\right\}$$

Then the cost of the partial layer and wire type assignment $\mathrm{cand}^{\geqslant}(e, l, w)$ equals the sum of the costs of an upper and lower partial assignment at v by splitting z, ω at child edge i, adding the cost of the edge e and subtracting the cost of a via at layer l at v because both the upper and lower partial assignment contain such a via at layer l. □

Finally, we want to show the correct computation of the partial upper candidates in line 10. An analogous statement can be proven for the partial lower candidates in line 17.

Claim 54.2. *For $l \in \mathcal{L}$ and $w \in \mathcal{W}$ and $i = 1, \ldots, k$ it holds that*

$$\mathit{OPT}^{\downarrow}(i, l, w) =$$
$$\min \begin{Bmatrix} \min_{w' \in \mathcal{W}} \left(\mathit{OPT}^{\downarrow}(i, l+1, w') \right) + \mathit{cost}_{\mathit{via}}(l, w) + R(l, l+1) \cdot \alpha_i \\ \mathit{OPT}^{\downarrow}(i-1, l, w) + \mathit{cand}(e_i, l, w) \end{Bmatrix}$$

where $\alpha_i := \sum_{j=1}^{i} \left(C_{\mathit{min}}(e_j) + C_{\mathit{min}}(Y_{v_j})\right) \cdot \sum_{j=1}^{i} \mathit{cost}_{\mathit{delay}}(Y_{v_j})$.

Proof. Consider an optimum layer and wire type assignment z, ω for $\mathrm{OPT}^{\downarrow}(i, l, w)$:

1. If $z(e_i) > l$ then z, ω contain an optimum wire and layer assignment for $\mathrm{OPT}^{\downarrow}(i, l+1, w')$ for some wire type w' to which we have to add the cost of a via at layer l and the delay costs from layer l to layer $l+1$ to obtain the costs of the optimum layer and wire type assignment z, ω for $\mathrm{OPT}^{\downarrow}(i, l, w)$. The delay costs through the via from layer l to $l+1$ equal $R(l, l+1) \cdot \alpha_i$ because exactly the child edges $e_1, ..., e_i$ are embedded above layer l.

2. If $z(e_i) = l$ then z, ω contain an optimum wire and layer assignment for $\mathrm{OPT}^{\downarrow}(i-1, l, w')$ to which we have to add the cost $\mathrm{cand}(e_i, l, w)$ of an optimum embedding of child edge e_i and its sub-arborescence to obtain the costs of an optimum layer and wire type assignment z, ω for $\mathrm{OPT}^{\downarrow}(i, l, w)$.

$\square$

To compute $\mathrm{OPT}^{\downarrow}(i, l_{\max}, w)$ we have to sum up the costs of one via pad at layer $l_{\max}$ and the costs of optimum partial candidates for the child edges $e_1, ..., e_i$ at layer $l_{\max}$ as it is done in line 5. $\mathrm{OPT}^{\downarrow}(0, l, w)$ only consists of the cost of a via pad at layer l showing the correctness of line 8.

$\mathrm{cand}\left((v'_{\mathrm{source}}, v_{\mathrm{source}}), l_{\mathrm{source}}, w\right)$ denotes the complete cost of an optimum layer and wire type assignment, assuming that we use wire type w on the source terminal. Note that the artificial pointwise edge $(v'_{\mathrm{source}}, v_{\mathrm{source}})$ added at the beginning has zero costs. Minimizing over the wire types, Algorithm 3 correctly returns the cost of a solution minimizing rcp on line 22. The actual solution can be rebuilt by storing back pointers to the partial candidates from which a candidate was computed.

The running time is dominated by the nested loops iterating over all edges, all orderings of outgoing edges, all layers, all outgoing edges and all wire types. Note that $\min_{w' \in \mathcal{W}} \left(\mathrm{OPT}^{\downarrow}(i, l+1, w')\right)$ in line 10 does not depend on w and hence can be computed separately. By the assumption that all node degrees of Y are bounded by a constant, we obtain the desired running time of $\mathcal{O}(|E(G)| \cdot |\mathcal{L}| \cdot |\mathcal{W}|)$. $\square$

Algorithm 3 is implemented in BonnRouteGlobal and used with the dynamic local usage. The assumption on the constant vertex degree is important. Even for $|\mathcal{W}| = 1$ [53] shows that it is $\mathcal{NP}$-hard to find a layer and wire type assignment z, ω minimizing $\mathrm{rcp}(z, \omega)$ if the vertices of the given arborescence have arbitrary degrees. In our situation we can always assume Y to have bounded degrees: since Y is required to be rectilinear there are only four directions from which a node in Y can be accessed. If parallel edges are removed beforehand the degrees can hence be bounded by five, taking into account that a node might additionally connect to a pin.

3.3.3 *Simultaneous Optimization*

Sections 3.3.1 and 3.3.2 described how to optimize x- and y-coordinates first and layers and wire types afterwards. Here we briefly outline how to optimize them simultaneously in order to achieve an approximation guarantee for the overall problem of embedding an arborescence into the chip image and the layers and assigning wire types obeying the given move bounds. It is possible to generalize the approaches from Sections 3.3.1 and 3.3.2: For an edge $e = (u, v)$ of the given arborescence, a candidate $\text{cand}(e, l, w)$ for a layer l and wire type w would not consist of a single optimum value but of a function $s : \square(u) \to \mathbb{R}_{\geqslant 0}$ that returns the optimum cost of an embedding of the sub-arborescence $Y_v \cup \{e\}$ given that u is located on a specific position in its move bound. These functions s could still be represented by a finite number of anchor points and the sum or the pointwise minimum of such functions would be well-defined. However, optimizing x- and y-coordinates together with the layers, the cost functions s are no longer separable in the x- and y-coordinates:

Theorem 55. *Assume that we are given an arborescence A that connects the pin shapes of a net n. The minimum costs $s_v(x, y, z)$ for an embedding of the sub-arborescence rooted at $v \in V(A)$, under the condition that v is embedded at (x, y, z), are not additively separable in the x- and y-coordinates. This holds even if there are no tip-to-tip penalties, and wire costs and via costs are constant on each layer.*

Proof. Consider the sample instance in Figure 27. Assume that w is a terminal fixed at coordinates $(0, 0, l_1)$ and that there are four layers $l_1, ..., l_4$ with $\mathcal{L}_x = \{l_1, l_3\}$ and $\mathcal{L}_y = \{l_2, l_4\}$. Unit wire length costs amount to 2 on layers l_1 and l_2 and to 1 on layers l_3 and l_4. Assume that a via from l_1 to l_2 has zero costs, from l_2 to l_3 it costs 5 and from l_3 to l_4 its costs are 0.

Consider three different locations for a node v that has w as its only child node. It is $s_v(0, 0, l_1) = 0$. A straight connection on layer l_1 suffices to connect w to $(10, 0, l_1)$, hence $s_v(10, 0, l_1) = 20$. To connect to $(0, 10, l_1)$ one has to use a via to layer l_2 and again down to l_1, whose costs are zero. Therefore, $s_v(0, 10, l_1) = 20$ as well. For a connection to $(10, 10, l_1)$ it is beneficial to pay for a via up to and down from layers l_3 and l_4 in order to profit from the lower wire costs on l_3 and l_4. This results in minimum costs of $s_v(10, 10, l_1) = 30$.

If s_v were additively separable in the x- and y-coordinates it could be written as $s_v(x, y, z) = s_1(x, z) + s_2(y, z)$ for some functions s_1 and s_2. This leads to the following contradiction:

$$\begin{aligned}
30 =& s_v(10, 10, l_1) + s_v(0, 0, l_1) \\
=& s_1(10, l_1) + s_2(10, l_1) + s_1(0, l_1) + s_2(0, l_1) \\
=& s_v(10, 0, l_1) + s_v(0, 10, l_1) = 40
\end{aligned}$$

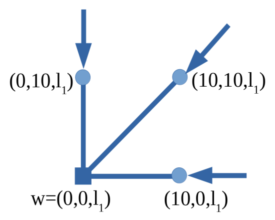


Figure 27: This example shows that the route costs are not additively separable in the x- and y-coordinates if layers are optimized simultaneously. This holds even without tip-to-tip penalty costs and constant wire and via costs on each layer.

□

In order to optimize x-, y- and z-coordinates simultaneously, one would have to deal with two-dimensional piece-wise linear functions, i.e. polygons, for each layer. This might be cumbersome and it is also questionable whether running times would be fast enough in practice.

In the context of continuous routing, that uses rhomboidal global routing tiles instead of the rectilinear tiles on our grid graph $\mathcal{G}$, [51] uses a Hanan grid to embed a planar arborescence into three dimensions optimally with regard to linear prices including congestion costs. This translates into our global routing setting with rectilinear tiles. Essentially, using a Hanan grid is the same as the approach from Section 3.3.1.4, but instead of modeling the cost functions on continuous intervals only values of the cost functions at certain grid positions are kept. Using a Hanan grid one can directly apply the dynamic program from Section 3.3.2: A candidate $\text{cand}(e, l, w, x, y)$ additionally contains the x- and y-position of the end node v of $e = (u, v)$. The computation of such a candidate works similarly as in Algorithm 3.

To avoid a quadratic dependence of the running time on the number of possible x-coordinates (analogously y-coordinates) one can perform an analogous technique as in [51] or in Algorithm 32. Having computed optimum candidates $\text{cand}(e', l, w, x', y)$ for a child edge e' of e and fixed $l \in \mathcal{L}_x, w, y$ and for all possible values of x' one can propagate the minimum costs that the sub-arborescence rooted at e' induces to the candidates of the parent edge $\text{cand}(e, l, w, x', y)$ for all possible values of x' by only sweeping once over all candidates of e' from left to right and right to left.

Timing can be optimized as well: A lower bound on the downstream capacitance can be computed similarly as in Section 3.3.1.5 with the difference that the lower bounds are stronger, because for $\text{cand}(e = (u, v), l, w, x, y)$ we know the fixed position $v = (x, y)$ in contrast to the setting in 3.3.1.5 where v can be moved within its move bound. Similarly, one can compute a lower bound on the upstream resistance. The capacitance correction factor for a certain embedding of an edge $e = (u, v)$ (analogously to Section 3.3.2) would have to consider the difference in the downstream capacitance lower bounds from u and v and the capacitance of e. This approach generalizes Section 3.3.2 and the approximation algorithm for timing-aware path search from [53] and the achievable approximation ratio would be a combination of these two results. While the approach from [53] only works for paths, here we can deal with arbitrary arborescences because we assume their topology to be fixed making it possible to compute reasonable lower bounds on the downstream capacitance.

The drawback of this approach is that its running time depends on the size of the Hanan grid which might be large in case of many pins.

A completely different solution would be to skip the computation of the coarse route and to try to immediately compute an optimum local route. This would require a denser global routing graph and is similar to what the continuous router in [51] is doing. In this way it would be possible to establish an approximation ratio for the whole problem of computing a local route which is not possible if this task is subdivided into the computation of a coarse and local route. However, also [51] only uses a coarser version of the Hanan grid for the computation of the local route and performs post-processing afterwards including an optimization of the wires that access the exact pin shapes. Therefore, the approach for the dynamic local usage that we described in this thesis and which is implemented in BonnRouteGlobal is already very similar with the difference that our global routing graph still is much coarser than the graph used in [51].

Finally, instead of running the x-y-optimization followed by the z-optimization once one could run them in an alternating sequence. Besides a longer running time, this did not prove useful in practice with the layer-embedded congestion costs as demonstrated in Section 3.3.1.6. The same holds true for a modification of the planar congestion costs to use the layers of a previously-run layer optimization. In particular, it is not clear how to best use given layers within the planar congestion costs, as the routing dimension of the wires can still change, making it necessary to embed them into other layers than might have been considered in the costs during the x-y-optimization.

3.4 INCREMENTAL ROUTING

Once BonnRouteGlobal has completed all the steps outlined in Figure 8 on page 16 it is possible to immediately proceed with the detailed routing. However, IBM performs an additional step between global and detailed routing called routing based optimization [53]. External tools make small changes on few nets to improve the overall timing of the chip. Among these changes there can be the insertion of buffers (buffering), moving and resizing logical gates (gate sizing) and the modification of net properties like its wire type or assignable layer range. All these optimizations are also performed before global routing, but at that point optimizations have to rely on estimates for the wire length and timing of the nets. Only after global routing it becomes clear which nets are timing-critical with worse delays than estimated beforehand. This can happen in case of large detours that were necessary to avoid high congested areas. During global routing based optimization many small changes are performed on the nets based on the timing properties of the given global routing. Analogously, there is detailed routing based optimization after detailed routing. In Section 4.5 we describe this flow in more detail.

Having moved, added or deleted pins of a net or having changed its properties makes it necessary to reroute the net (Figure 28). This should be done with as few disruptions as possible so that the timing properties of the net do not change too much and no other nets have to be rerouted due to rising congestion. Incremental BonnRouteGlobal has been developed to address this task [53]. Using the methods from Sections 3.2 and 3.3 the incremental router works together with the dynamic local usage with one difference which we will explain in the following.

To reroute the nets with as little disruption as possible, Incremental BonnRouteGlobal solves the following problem [53]:

Problem 56. COARSE MINIMAL REROUTE

Input: A net n, an initial arborescence Y_0 that is part of the global routing graph $\mathcal{G}$ and connects the global routing tiles of a subset of the pins of n.

Output: A coarse route A for n minimizing

$$\text{cost}_{\text{cong}}(A \setminus Y_0)$$

that is A can use wires and vias in Y_0 for free.

Up to now Incremental BonnRouteGlobal is not timing-aware which is why we ignore timing in the problem formulation. This is partly due to the fact that we usually reroute only small parts of an existing local route during incremental routing reducing the need to optimize timing. Moreover, employing the techniques of the timing-aware computation of coarse routes described in 3.2 and reconnecting sinks to the source might lead to too much disruption. Nonetheless, with the dynamic local usage we use the timing-aware algorithms described in Section 3.3 to optimize local routes. In the future it might be desirable to consider timing in some way already in the path search during incremental routing.

The incremental router considers the coarse minimal reroutes problem since tile-internal wires do not affect congestion within the traditional router. It can be reduced to a minimum Steiner tree problem in the global routing graph by contracting the edges of the initial arborescence. Hence the incremental router can employ the techniques from Section 3.2. In addition, it slightly modifies the edge costs of the global routing graph for the path search depending on the position of the wires of the initial arborescence to heuristically improve wire length of the local routes: If a large part of a global routing graph edge is covered by an input wire its length objective cost is reduced proportionally during path search (see [53] for details). With the traditional router this approach is limited to objective costs because congestion costs are not influenced by tile-internal input wires. However, using area usages with the dynamic local usage, one could apply the same modification on the congestion costs of an edge, that is, if a larger portion of a global routing graph edge is covered by an input wire one could reduce the congestion costs of that edge proportionally. If the path search decides to use this route it is likely that the resulting post-processed local route will use the input wire, justifying the reduction in congestion costs. In fact, using this modification, congestion got slightly worse in practice. Possibly this is due to the fact that even if a congested edge is largely covered by an input wire it might still be necessary to put additional vias or wires to be able to actually connect to the input wires in the local route, causing additional congestion.

Having computed the coarse route as a solution to the coarse minimal reroutes problem, the traditional router computes a local route as described in Section 3.3 and embeds the tile-internal wires heuristically into layers. In this step it can optimize all tile-internal wires including those of the initial arborescence (Figure 28).

With the dynamic local usage local movements of wires inside the global routing tiles affect congestion. Using the same approach as for the traditional router that allows movements of wires inside the global routing tiles would lead to a significant increase of congestion during incremental routing on some instances, similar as in Figure 20. This is why we want to solve the following more restrictive problem with the dynamic local usage:

Problem 57. LOCAL MINIMAL REROUTE

Input: A net n, an initial local route A_0 for n with the exception that A_0 does not need to connect all pins of n.

Output: A local route A for n minimizing

$$\text{cost}_{\text{cong}}(A \setminus A_0)$$

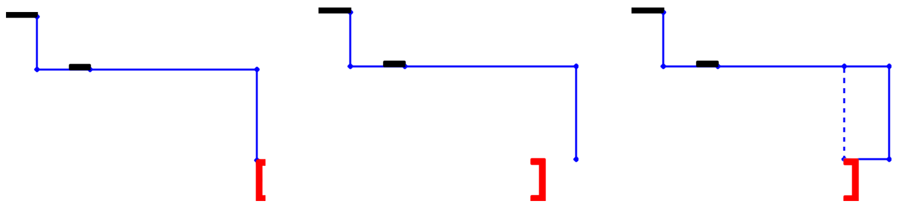
that is A can use wires and vias in A_0 for free.

The difference is that we optimize the congestion costs of the local route and keep the wires of the initial local route fixed. If Incremental

BonnRouteGlobal was extended such that it reroutes not only the modified nets but, at certain points of time, a larger number of nets that consume from congested edge resources, one might allow some re-optimization of tile-internal wires of the initial local route.

We solve the local minimal reroute problem heuristically with the same two-step approach as used during the resource sharing algorithm: We take a solution of the coarse minimal reroutes problem and connect it to the exact pin shapes as described in Section 3.3 taking the existing wires of the initial local route into account. Afterwards, we optimize the resulting local route as in sections 3.3.1 and 3.3.2 for which we fix the positions and layers of the wires that are taken from the initial route.

Although the increase in congestion on any edge resource is kept as small as possible with this approach, this comes with the disadvantage that rerouted nets might not have optimum wire length inside the global routing tiles (Figure 28). However, already the coarse route often does not have optimum total wire length because that is not the objective of the coarse minimal reroutes problem. Practical experiments show that wire length increases only slightly during routing based optimization (Section 4.5). On the contrary, congestion sometimes increases noticeable with the dynamic local usage showing that the major effort should be on keeping congestion low. For this reason we also weigh congestion costs stronger with the dynamic local usage than with the traditional router during incremental routing. Future work might put efforts on trying to optimize existing wiring in areas that are not highly congested. Independently of the dynamic local usage, Incremental BonnRouteGlobal might be made more robust with regard to congestion by adding bulk reroute phases, in which all nets in highly congested areas are rerouted. Still, the number of rerouted nets would have to be kept as low as possible in order not to destroy the efforts of the routing based optimization tools.



Initial local route.

The source pin (red) has been flipped and moved to another position.

The local route is reconnected to the new pin position.

Figure 28: Incremental BonnRouteGlobal reconnects a route after a pin has moved. Assuming that all wires are contained in one global routing tile the traditional router re-optimizes the wires removing the U-shape on the right and obtaining optimum wire length (dashed). The dynamic local usage never modifies pre-existing wires which is why for this instance the rerouted local route does not have optimum wire length. Re-optimizing wires inside global routing tiles would disturb congestion with the dynamic local usage, even if done congestion-aware. This is even the case if re-optimization is restricted to global routing tiles in which the net has changed (a pin was added, moved or deleted) and fixing wires in tiles where there were no changes except for potential price changes of the edge resources.

4 DYNAMIC LOCAL USAGE IN PRACTICE

The dynamic local usage was implemented in BonnRouteGlobal and tested on chip units provided by IBM, with whom the Research Institute for Discrete Mathematics collaborates with. Moreover, it was tested within the routing flow at IBM with the goal of using the dynamic local usage in the IBM chip design process.

4.1 TESTBED

Our testbed consists of chips with 5nm and 7nm technology.

For 7nm, there is chip A_1 that by far has the largest chip image in our testbed but most nets have already been routed, chips B_1 to B_3 that are also large with a high number of nets, and chips C_1 to C_4 that have medium to small sizes. C_4 is so small that we exclude it in most of the experiments.

For 5nm, we have chips D_1 to D_7 that have medium to small sizes, chips E_1, E_2 and F_1 that are larger with a large number of nets, and chips G_1 to G_5 that are also medium to small and which are the most recent instances. Not all of the IBM routing flow was running for chips G_1 to G_5 at the time of writing.

It should be noted that the instances and the whole code environment were constantly slightly changing. This is why results on the same chip but from different experiments are not necessarily comparable.

4.2 METRICS

The following metrics are used to evaluate the experimental results:

RS TIME The running time of the resource sharing algorithm in BonnRouteGlobal in *hours* : *minutes* : *seconds*. All runs that are compared to each other were executed on the same machine, while runs of different test setups are not necessarily comparable in terms of running time.

RS REROUTES The number of reroutes during the resource sharing algorithm. Nets are only rerouted if the cost of the previously computed route has increased to a certain minimum amount (see [49] and Section 5.4.3 for more details). With the dynamic local usage all nets contribute to congestion including those whose pins are all located inside one global routing. Therefore, the set of nets that have to be rerouted during resource sharing generally is larger with the dynamic local usage than with the traditional router.

WACE4 The *wACE4*-metric measures critical congestion of a chip [56] and is computed as follows: Let $c_{\delta\%}$ denote the average congestion of the δ% most congested area on the chip. The wACE4 value is defined as the average of the four numbers $c_{0.5\%}, c_{1\%}, c_{2\%}$ and $c_{5\%}$. This way the average congestion of the 0.5% most congested area is counted 46.25%, the area in the range of 0.5% to 1% 21.25%. The area in the range of 1% to 1.5% and 1.5% to 2% is counted 8.75% each and that in the range of 2% to 5% 2.5% per 0.5% step. Hence there is an increasing dependence on the congestion of the 5% most congested area of the chip.

As it depends on the usage model, the wACE4 is not comparable between the traditional router and the dynamic local usage. The wACE4-metric is used to predict detailed routability of a chip and hence is required to be stable and correlated with how difficult it is for BonnRouteDetail to route the chip without too many design rule violations, shorts and opens.

WIRELENGTH The total length of the wires in all routes in *meters*. This includes wires that were already present in the input.

VIAS The total number of vias in all routes, including vias in the input.

FAILED SEARCHES The number of unsuccessful path searches in BonnRouteDetail, making it necessary to enlarge the search area. A higher number hints at a higher difficulty for BonnRouteDetail to route the chip following the given routes by BonnRouteGlobal.

DR TIME The total running time of BonnRouteDetail in *hours* : *minutes* : *seconds*.

SCENIC 125 / 150 / 200 The number of nets whose route computed by BonnRouteDetail is at least 25% / 50% / 100% longer than the route computed by BonnRouteGlobal. A higher value indicates problems of BonnRouteDetail to follow the global routes.

STEINER SCENIC 125 / 150 / 200 The number of nets whose route computed by BonnRouteDetail is at least 25% / 50% / 100% longer than a planar shortest Steiner tree for the pins.

DRCS The total number of design rule violations (DRCs) in the routes computed by BonnRouteDetail. Usually it is easier for Bonn-

RouteDetail to compute routes that do not have too many DRCs if the routes computed by BonnRouteGlobal are not packed too densely.

WS The *worst slack* over all routes in *pico seconds*. At a sink pin the *slack* is the difference between the required arrival time given by the design specifications and the actual arrival time resulting from the computed routes by BonnRouteGlobal / BonnRouteDetail. The worst slack is the minimum among these.

SNS The sum of all negative slacks in *pico seconds*.

SLEW VIOLATIONS The number of slew violations. The slew describes how fast the voltage of a signal changes. It is defined as the length of the time interval in which the voltage of a signal lies between 10 and 90% of the maximum voltage. The slew must not be too large.

SHORTS The total number of shorts after detailed routing. A short is an overlap of wires that belong to different nets and must be avoided, as it would cause signal errors.

4.3 TIMING-UNAWARE

Tables 4 and 5 show results on seven chip units of the 7nm technology node and with different numbers of nets of a global to detail routing flow: First, BonnRouteGlobal is run, whose output is given to BonnRouteDetail. Once BonnRouteGlobal is run with the traditional router and once with the dynamic local usage.

Looking at Table 4, the congestion metric *wACE4* drops significantly with the dynamic local usage across the entire testbed by more than 4 percent on average. This means that it is necessary to adjust the congestion thresholds when assessing routability of a chip with the dynamic local usage. The smaller pessimism of the dynamic local usage with regard to congestion also reflects in shorter wire length and in particular in much fewer vias. Due to the different usage models, changes in these metrics are expected and do not necessarily mean that metrics will improve in the detailed router. However, as Table 5 shows, BonnRouteDetail is still able to pack the wires despite the smaller pessimism. Moreover, wire length and via numbers after detailed routing improve as well, suggesting that decreased wire length and via numbers in global routing are desirable in order to achieve good detailed routing results as long as the routes are not too densely packed. With the dynamic local usage, the number of resource sharing reroutes drops on almost all instances, which can be explained by the lower congestion that makes resource sharing prices grow slower and less reroutes necessary. Moreover, the dynamic computation of

Chip	Nets	Router	RS time		RS reroutes		wACE4		Wire length		Vias	
B_1	1735036	Traditional	00:12:01		7088771		86.51		27.2609		16155665	
		DLU	00:15:43	+30.81 %	5214158	-26.44 %	82.29	-4.22 %	26.7957	-1.71 %	14617797	-9.52 %
B_2	1206790	Traditional	00:10:25		5145961		86.23		19.7871		11711717	
		DLU	00:16:13	+55.80 %	3877853	-24.64 %	82.76	-3.47 %	19.6026	-0.93 %	10571000	-9.74 %
B_3	373431	Traditional	00:02:03		1092355		84.84		6.1440		3505605	
		DLU	00:02:35	+26.55 %	742870	-31.99 %	80.48	-4.36 %	6.0811	-1.02 %	3177167	-9.37 %
C_1	373611	Traditional	00:00:42		1515062		86.73		2.1339		3184455	
		DLU	00:02:26	+243.30 %	1534078	+1.26 %	81.34	-5.39 %	2.1324	-0.07 %	2859398	-10.21 %
C_2	246194	Traditional	00:00:27		668912		81.88		2.8710		2034164	
		DLU	00:00:56	+109.29 %	435933	-34.83 %	75.59	-6.29 %	2.8779	+0.24 %	1842539	-9.42 %
C_3	148801	Traditional	00:00:17		297160		80.70		2.0419		1215662	
		DLU	00:00:52	+200.49 %	299924	+0.93 %	76.60	-4.10 %	2.0398	-0.10 %	1124175	-7.53 %
Summary		Traditional	00:25:57		15808221		84.48		60.2388		37807268	
		DLU	00:38:50	+49.63 %	12104816	-23.43 %	79.84	-4.64 %	59.5295	-1.18 %	34192076	-9.56 %

Table 4: Results of BonnRouteGlobal on 7nm instances; once with the traditional router and once with the dynamic local usage. The resource sharing algorithm takes only a fraction of the running time of BonnRouteGlobal which on B_1 took one hour and 14 minutes in total.

Chip	Router	DR time		Failed searches		Wire length		Vias		Scenic 125		Scenic 150		Scenic 200		DRCs	
B_1	Traditional	10:20:33		94		28.7246		18508522		23948		3303		416		1064	
	DLU	12:03:23	+16.57 %	66	-29.79 %	28.2433	-1.68 %	18270841	-1.28 %	25406	+6.09 %	3728	+12.87 %	487	+17.07 %	1006	-5 %
B_2	Traditional	07:02:13		47		20.8139		13473228		9175		1055		126		1402	
	DLU	06:57:17	-1.17 %	45	-4.26 %	20.6289	-0.89 %	13365593	-0.80 %	10893	+18.72 %	1528	+44.83 %	212	+68.25 %	1438	+3 %
B_3	Traditional	02:01:26		12		6.4396		3957512		2397		319		46		123	
	DLU	01:43:06	-15.10 %	5	-58.33 %	6.3817	-0.90 %	3949313	-0.21 %	3201	+33.54 %	539	+68.97 %	85	+84.78 %	124	+1 %
C_1	Traditional	01:04:37		3		2.3177		3455421		4067		246		12		79	
	DLU	01:16:19	+18.11 %	9	+200.00 %	2.3196	+0.08 %	3420389	-1.01 %	5017	+23.36 %	448	+82.11 %	24	+100.00 %	126	+59 %
C_2	Traditional	00:46:56		2		3.0081		2327123		1527		104		8		370	
	DLU	00:52:43	+12.32 %	1	-50.00 %	3.0201	+0.40 %	2327579	+0.02 %	1962	+28.49 %	232	+123.08 %	26	+225.00 %	365	-1 %
C_3	Traditional	00:38:02		3		2.1275		1366605		670		114		7		153	
	DLU	00:32:46	-13.85 %	7	+133.33 %	2.1270	-0.02 %	1364513	-0.15 %	916	+36.72 %	183	+60.53 %	19	+171.43 %	144	-6 %
Summary	Traditional	21:53:47		161		63.4314		43088411		41784		5141		615		3191	
	DLU	23:25:34	+6.99 %	133	-17.39 %	62.7206	-1.12 %	42698228	-0.91 %	47395	+13.43 %	6658	+29.51 %	853	+38.70 %	3203	+0 %

Table 5: Results of BonnRouteDetail on 7nm instances; once following the traditional router and once the dynamic local usage.

local wiring usage might make it easier to compute good local routes. Only on C_1 and C_3, that have particularly many large obstacles, the number of reroutes slightly increases.

The running time of the resource sharing algorithm increases with the dynamic local usage because each time a net is rerouted, an optimized local route has to be computed while the traditional router does so only after the resource sharing. Furthermore, the usage computation takes more time as all wires of the computed local routes have to be accounted for.

Unlike the results after global routing, the results after detailed routing in Table 5 are directly comparable between the traditional router and the dynamic local usage. The running times of the detailed router are similar on average. The number of failed searches during detailed routing decreases on average proving the ability of the dynamic local usage to successfully match the packing density for the detailed router on most chip instances. The wire length after detailed routing decreases by more than one percent over the whole testbed, which is a substantial improvement. The via number also improves by almost one percent, which is significant as well. This does not come at the cost of more DRCs, whose numbers do not change on average, again underlining the ability of the dynamic local usage to reasonably pack the global wires letting enough space for the detailed router to realize the wiring meeting all design constraints. It should be noted that on one instance the number of DRCs increases by almost sixty percent indicating that on some special input the traditional router might still estimate the packing density better than the dynamic local usage.

The only metric that is getting worse on average is the number of scenic routes. Because the dynamic local usage creates shorter and therefore denser routes, BonnRouteDetail potentially struggles on some of the routes to exactly follow the global routing corridors. Large scenic numbers can pose problems when routing with timing: If BonnRouteDetail cannot stick to the routing corridor that BonnRouteGlobal chose and computed the timing for, the timing can degrade after detailed routing. Currently, BonnRouteDetail is not timing-aware itself and relies on good timing properties of the routes by BonnRouteGlobal. However, Section 4.5 shows that throughout the whole IBM routing flow the dynamic local usage can achieve very good timing also.

4.4 LARGE GLOBAL ROUTING TILES

The granularity of a global routing can be controlled by the density of the global routing graph. If this graph is chosen to be less dense, that is each global routing tile covers a larger area of the chip, the global routing problem becomes easier as there are fewer nodes and edges and more pins are mapped to the same global routing graph node. This immediately reduces the running time of BonnRouteGlobal and, in particular, of the resource sharing algorithm. For large instances long running times can pose a difficulty in early design stages during which the congestion of an instance has to be evaluated very often, and choosing large global routing tiles can solve this problem. The question is whether the results are still meaningful. With a coarser global routing graph BonnRouteGlobal loses some ability to detect and estimate local packing issues, since all the congestion is spread and flattened out over a larger area. Moreover, blockage structures that are smaller than the global routing tiles cannot be captured accurately. Therefore, it is important to verify the performance of the dynamic local usage on large global routing tile sizes. With larger global routing tiles more wires are located inside single global routing tiles. This poses an advantage for the dynamic local usage which measures the congestion of the tile-internal wires dynamically during the resource sharing algorithm in contrast to the traditional router, which relies on the static local wiring pre-estimates. These have an even greater impact in combination with large global routing tiles.

Tables 6 and 7 show results of BonnRouteGlobal on the largest 7nm instances of our testbed with default global routing tile size and four times the default size. The running time of the resource sharing algorithm reduces significantly with both the traditional router and with the dynamic local usage. The dynamic local usage is very stable in terms of wire length which is almost the same with four times the global routing tile size as with the default size. The number of vias reduces by a few percent. The congestion reduces substantially with the dynamic local usage (Figures 29 and 30). With the traditional router there is less a reduction in congestion but at the cost of a largely increased wire length. Moreover the number of vias decreases significantly with the traditional router with large tile sizes. Using large tile sizes the coarse routes have to make larger detours in case of congestion or blockages. This explains the larger wire length of the traditional router, because its congestion map adapts to the longer coarse routes during resource sharing. On the contrary, the dynamic local usage computes local routes already during the resource sharing and optimizes their wire length inside the global routing tiles. It accounts usage for the optimized local routes resulting in much less congestion.

Like the traditional router extends all tile-crossing wires to tile-center to tile-center wires for usage computation, the dynamic local usage extends the wires by the tip-to-tip penalty. Both strategies help to face the problem that the optimized local routes of the global router might be much shorter than the routes by the detailed router because the latter has to pack them without overlaps (Figure 11 on page 22). This inconsistency grows proportionally with the tile size. The tra-

Chip	Nets	Router	RS time		RS reroutes		wACE4		Wire length		Vias	
A_1	1783142	Traditional 1x	00:08:20		3648215		85.63		299.8292		19296867	
		Traditional 4x	00:01:05	-86.82 %	2481836	-31.97 %	76.50	-9.13 %	304.6716	+1.62 %	17864486	-7.42 %
B_1	1732566	Traditional 1x	00:07:19		8389754		87.31		29.2103		19868611	
		Traditional 4x	00:03:58	-45.79 %	7485771	-10.77 %	86.68	-0.63 %	30.7097	+5.13 %	18172221	-8.54 %
B_2	1202357	Traditional 1x	00:07:54		7146521		87.78		21.1050		14284172	
		Traditional 4x	00:02:47	-64.79 %	5168919	-27.67 %	86.98	-0.80 %	22.4745	+6.49 %	13137176	-8.03 %
B_3	372283	Traditional 1x	00:01:09		1290122		85.85		6.5241		4277007	
		Traditional 4x	00:00:37	-45.91 %	1046003	-18.92 %	86.47	+0.62 %	7.0094	+7.44 %	3939969	-7.88 %
Summary		Traditional 1x	00:24:44		20474612		86.64		356.6686		57726657	
		Traditional 4x	00:08:28	-65.72 %	16182529	-20.96 %	84.16	-2.49 %	364.8652	+2.30 %	53113852	-7.99 %

Table 6: Results of BonnRouteGlobal with the traditional router on the largest 7nm instances; once with default global routing tile size and once with four times the default size.

Chip	Nets	Router	RS time		RS reroutes		wACE4		Wire length		Vias	
A_1	1783142	DLU 1x	00:13:44		3908967		89.43		297.2586		17340224	
		DLU 4x	00:02:42	-80.23 %	2440354	-37.57 %	69.79	-19.64 %	296.3549	-0.30 %	16611509	-4.20 %
		DLU 4x *)	00:02:44	-80.10 %	2575937	-34.10 %	71.28	-18.15 %	296.5431	-0.24 %	16596235	-4.29 %
B_1	1732566	DLU 1x	00:06:00		4991799		82.52		26.8382		15786132	
		DLU 4x	00:02:54	-51.63 %	3078731	-38.32 %	78.46	-4.06 %	26.7088	-0.48 %	15374236	-2.61 %
		DLU 4x *)	00:04:49	-19.90 %	7141757	+43.07 %	81.72	-0.80 %	26.8727	+0.13 %	15532210	-1.61 %
B_2	1202357	DLU 1x	00:09:47		4257787		83.69		19.7074		11391032	
		DLU 4x	00:02:47	-71.44 %	2486680	-41.60 %	80.18	-3.51 %	19.6190	-0.45 %	11172824	-1.92 %
		DLU 4x *)	00:03:27	-64.62 %	2991347	-29.74 %	83.08	-0.61 %	19.7502	+0.22 %	11247779	-1.26 %
B_3	372283	DLU 1x	00:01:09		890822		81.27		6.0859		3438506	
		DLU 4x	00:00:38	-43.90 %	668365	-24.97 %	78.87	-2.40 %	6.0979	+0.20 %	3358891	-2.32 %
		DLU 4x *)	00:00:38	-44.80 %	811679	-8.88 %	81.29	+0.02 %	6.1224	+0.60 %	3394769	-1.27 %
Summary		DLU 1x	00:30:42		14049375		84.23		349.8901		47955894	
		DLU 4x	00:09:04	-70.45 %	8674130	-38.26 %	76.83	-7.40 %	348.7806	-0.32 %	46517460	-3.00 %
		DLU 4x *)	00:11:39	-62.04 %	13520720	-3.76 %	79.34	-4.88 %	349.2884	-0.17 %	46770993	-2.47 %

Table 7: Results of BonnRouteGlobal with the dynamic local usage on the largest 7nm instances; once with default global routing tile size and once with four times the default size. *) In the third run the tip-to-tip penalty was increased by fourty percent.

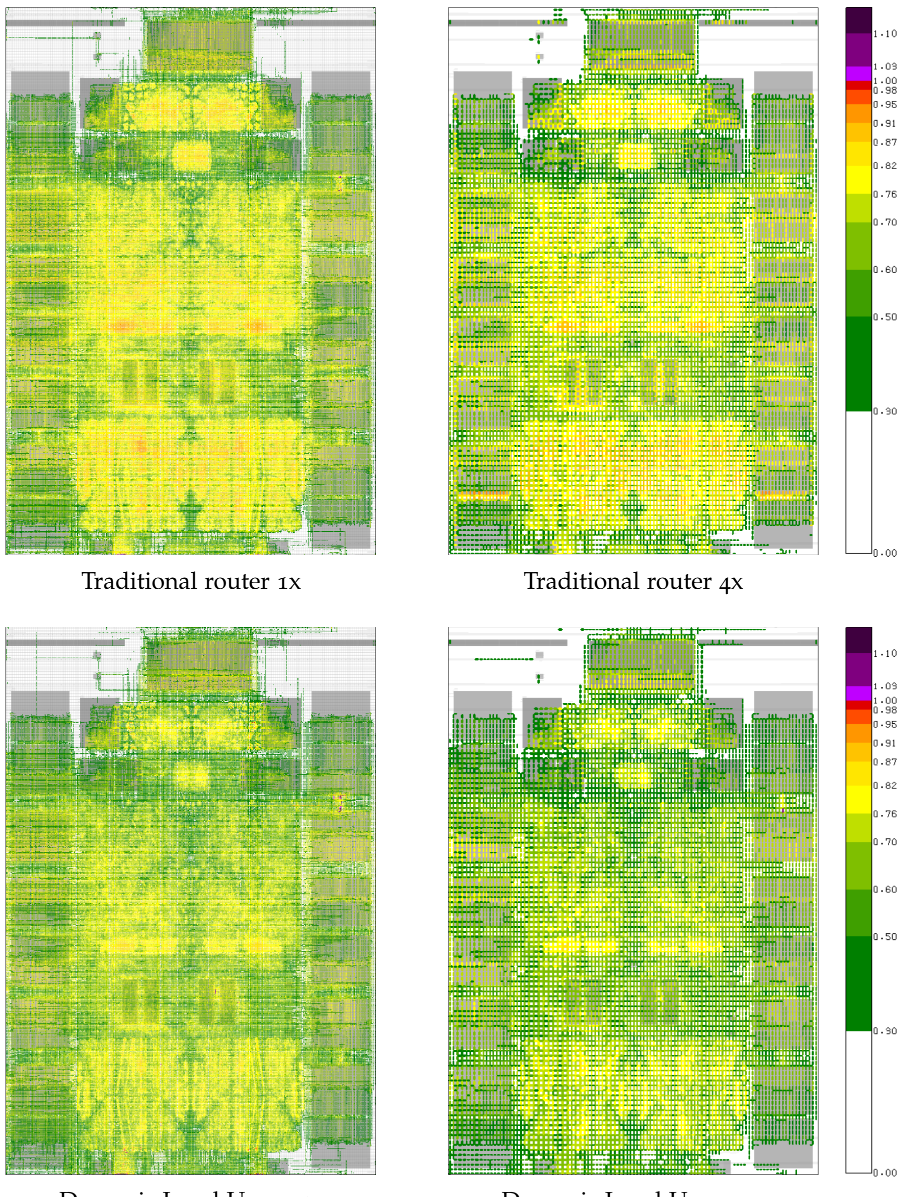


Figure 29: Congestion over all layers on chip B_1 with the traditional router and the dynamic local usage, once using default global routing tile size and once four times the default size. Congestion hotspots are clearly visible also with large tile sizes and both with the traditional router and with the dynamic local usage.

ditional router compensates for this by extending the wires proportionally to the global routing graph tile size. In fact this might be overly pessimistic since with larger tile sizes global routing edge resources rarely are completely packed by wires. In these experiments the tip-to-tip penalty of the dynamic local usage does not adapt to the tile size, explaining the drop in congestion with larger tiles. It is possible to make the tip-to-tip penalty depend on the tile size: The third line in Table 7 shows results of the dynamic local usage with large tile sizes and a tip-to-tip penalty that is increased by fourty percent. In this setting the congestion drops much less on three of the instances while wire length remains almost unchanged and the number of vias matches better with that of the default tile size. Depending on future technologies, it might be beneficial to increase the tip-to-tip penalty even further for large tile sizes. Likewise, first experiments have shown that for very small tile sizes one should use smaller tip-to-tip penalties.

In general it is expected that both with the traditional router and the dynamic local usage the congestion decreases with larger tile sizes, as local packing issues and blockage structures cannot be considered accurately. It would be possible to adjust parameters that control the packing density (see Section 3.1.1) to keep congestion stable with larger tile sizes. However that would be artifical and not match the granularity of the congestion with smaller tile sizes.

Overall the dynamic local usage is much more stable in terms of wire length with large tile sizes than the traditional router. The congestion reduces only slightly and can be made more stable by increasing the tip-to-tip penalty, proving that the dynamic local usage is a valid tool for congestion assessment.

4.5 timing-aware routing flow

BonnRouteGlobal and BonnRouteDetail are used by IBM to design chips with state-of-the-art technology. Routing at IBM comprises of a complex flow in which BonnRouteGlobal and BonnRouteDetail play a major role. One key motivation for the dynamic local usage was to improve overall results within this routing flow, having an immediate impact on chip development at IBM.

Figure 31 shows the part of the flow that is affected by BonnRouteGlobal. At this point, all components have already been placed on the chip and need to be connected through wires. BonnRouteGlobal computes a global routing, followed by the global routing based optimization. In this step, external tools perform local changes of the chip topology to improve timing, based on the computed global routing. These changes involve movements, deletions or additions of pins, deletions or additions of nets and changing the properties of specific

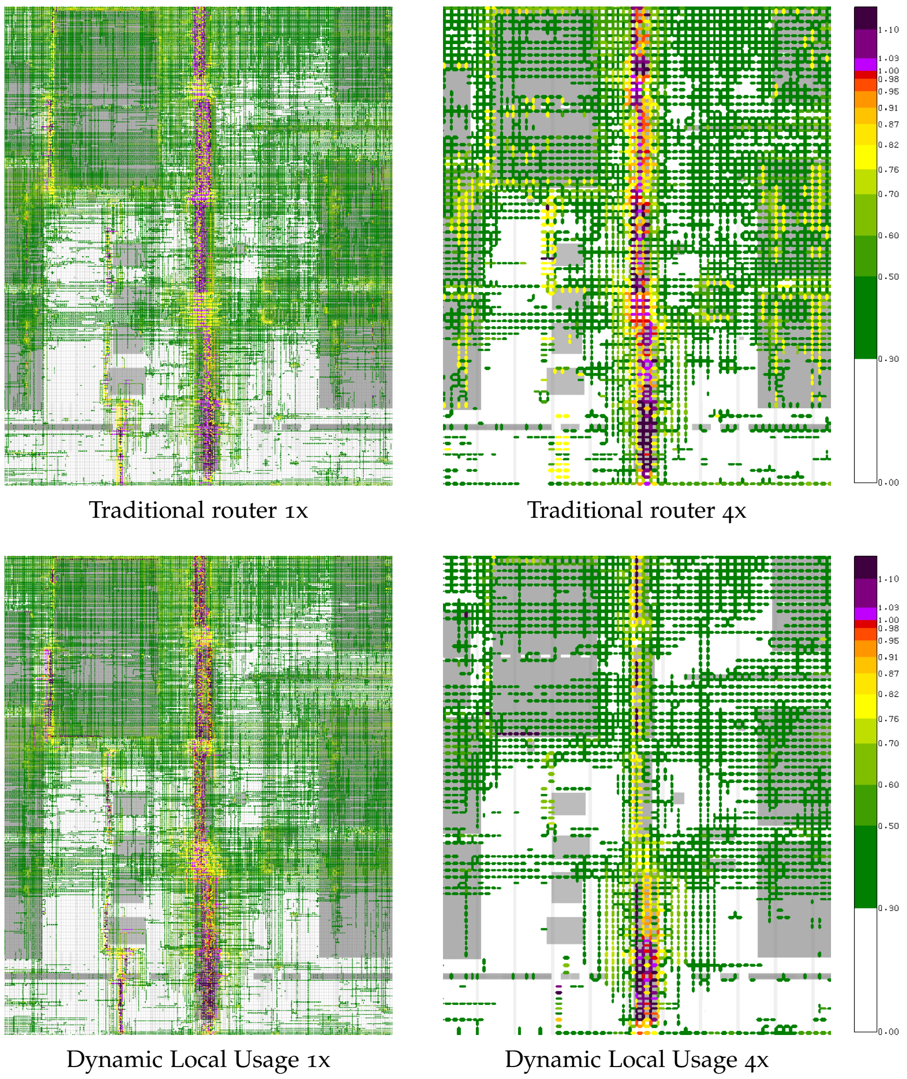


Figure 30: Congestion over all layers on an excerpt of chip A_1 with the traditional router and the dynamic local usage, once using default global routing tile size and once four times the default size. The chip has a vertical bottleneck caused by blockages through which BonnRouteGlobal struggles to put all wires through, leading to huge over-congestion with both the traditional router and the dynamic local usage.

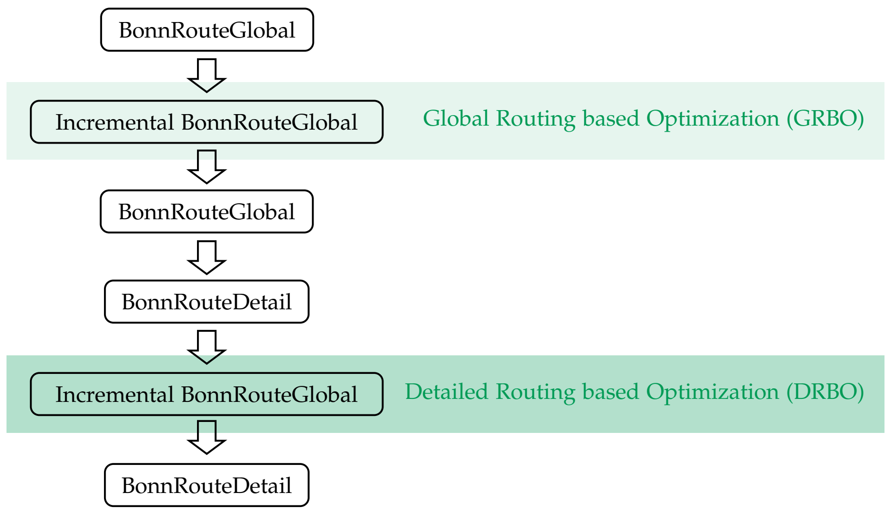

BonnRouteGlobal
Incremental BonnRouteGlobal
Global Routing based Optimization (GRBO)
BonnRouteGlobal
BonnRouteDetail
Incremental BonnRouteGlobal
Detailed Routing based Optimization (DRBO)
BonnRouteDetail

Figure 31: The routing flow consists of several calls to BonnRouteGlobal, BonnRouteDetail and Incremental BonnRouteGlobal. During global and detail routing based optimization external tools perform optimizations on nets that are going to be rerouted by Incremental BonnRouteGlobal.

nets. All affected nets are rerouted by Incremental BonnRouteGlobal (see Section 3.4).

Since Incremental BonnRouteGlobal reroutes single nets only and does not reroute a larger number of nets in case of developing congestion hotspots, the quality of the global routing might degrade during the global routing based optimization. Moreover, the external tools currently do not assess congestion but only timing during global routing based optimization. That is why BonnRouteGlobal is called again afterwards to route all nets from scratch considering the optimized chip topology. At the time of writing, this is the only call of BonnRouteGlobal that is run with arrival time customers (Section 2.4). The first call to BonnRouteGlobal is timing-aware but uses static timing budgets. Incremental BonnRouteGlobal is not timing-aware but only optimizes timing costs during the computation of the local routes with the dynamic local usage (see Section 3.4).

Thereafter, BonnRouteDetail computes a detailed routing based on the second global routing. This is followed by detailed routing based optimization, during which again external tools perform modifications to single nets, but based on the detailed routing. Incremental BonnRouteGlobal reroutes changed nets by adding global wires, that is the wires are not necessarily overlap-free with other wires. In a final call to BonnRouteDetail all nets that still contain global wires are rerouted such that all wires conform to local design rules and there are no overlaps.

Afterwards, IBM computes the timing with more refined models to detect persisting signal delay issues. Furthermore, the routing is checked for design rule violations.

Tables 9, 10, 11 and 12 show results at different points in the IBM routing flow for the 5nm instances in our testbed with the traditional router and with the dynamic local usage. After the first run of BonnRouteGlobal the dynamic local usage achieves a shorter wire length and better timing while there are both instances with higher and with lower congestion (Table 9). After global routing based optimization congestion and timing look worse on many instances with the dynamic local usage. Unlike the traditional router, the dynamic local usage sees local usage changes during incremental routing. Therefore, it is more prone to an increase of congestion than the traditional router, in particular as the tools used during routing based optimization do not monitor congestion but only timing.

Moreover, there is a significant degradation of timing during the last step of global routing based optimization (Table 8). In this step, gates are flipped if that reduces the wire length. For a gate with two output pins this would mean that the two pins swap positions which might reduce the wire length of the pin's routes. This affects all nets of the chip and not all routes are becoming shorter, making it necessary to perform many smaller local reroutes that affect timing and congestion in particular with the dynamic local usage. The traditional router also sees a degradation in timing, suggesting that this is not a specific problem of the dynamic local usage. The degradation is only temporary as after the second call of BonnRouteGlobal, which reroutes everything from scratch, congestion and timing improve again (Table 10).

		Before Flip Gate Opt.		After Flip Gate Opt.	
Chip	Router	SNS	wACE4	SNS	wACE4
E1	Traditional	-2282	92.4	-8395	92.4
	DLU	-403	97.2	-27191	97.5
E2	Traditional	-3883	92.3	-9573	92.2
	DLU	-2213	92.5	-19861	92.7
F1	Traditional	-1859	87.7	-1874	87.9
	DLU	-1711	90.4	-2121	90.5

Table 8: Results in the IBM Routing Flow before and after the flip gate optimization during global routing based optimization. Timing degrades both with the traditional router and with the dynamic local usage.

The wire length after the first run of BonnRouteDetail is better with the dynamic local usage on all of the instances except for one, with some instances experiencing a substantial improvement (Table 10).

Along with the shorter wire length there come substantial benefits in timing, namely the sum of negative slacks decreases significantly on most instances. The number of slew violations decreases on half of the instances. The improved numbers of shorts on all instances except for one indicate that BonnRouteDetail can handle the input of the dynamic local usage without difficulty.

The good results of the dynamic local usage persist through detailed routing based optimization (Table 11). At the time of writing, not all of the routing flow was enabled for the most recent instances G1 to G5. For those instances that did go through the entire flow, the dynamic local usage resulted in much better timing, wire length, and via numbers, as well as significantly fewer design rule violations at the end of the routing flow (Table 12). These results demonstrate that the dynamic local usage is capable of making persistent and significant improvements on routing. Currently, IBM is testing the dynamic local usage with the aim of making it the default and replacing the traditional router.

Chip	Nets	Router	After 1st BRG				After GRBO			
			WS	SNS	Wire length	wACE4	WS	SNS	Wire length	wACE4
E_1	820000	Traditional	-44	-17747	12.4204	92.6	-28	-8395	12.4953	92.4
		DLU	-27	-3959	12.3887	93.7	-38	-27191	12.4807	97.5
E_2	332000	Traditional	-41	-22481	6.5533	92.3	-24	-9573	6.5979	92.2
		DLU	-38	-8362	6.4401	91.7	-28	-19861	6.4982	92.7
F_1	418000	Traditional	-24	-4065	8.1812	87.7	-19	-1874	8.2236	87.9
		DLU	-20	-3709	8.1678	88.5	-22	-2121	8.2163	90.5
G_1	99599	Traditional	-151	-129042	0.7934	109.1	-43	-925	0.7953	109.7
		DLU	-119	-70960	0.7699	98.6	-46	-1361	0.7776	104.4
G_2	53925	Traditional	-30136	-30633	0.8232	78.9	-30117	-30613	0.8238	78.4
		DLU	-30136	-30600	0.8220	79.9	-30117	-30580	0.8233	80.2
G_3	27911	Traditional	-127	-18572	0.2632	103.9	-17	-136	0.2632	105.9
		DLU	-149	-15424	0.2538	97.4	-13	-47	0.2563	100.7
G_4	12547	Traditional	-11	-6	0.0884	85.0	5	0	0.0898	85.6
		DLU	-22	-17	0.0881	87.1	5	0	0.0890	88.0
G_5	12165	Traditional	-59	-890	0.0826	81.8	-3	0	0.0831	81.0
		DLU	-50	-741	0.0824	86.5	-3	0	0.0832	86.5

Table 9: Results in the IBM Routing Flow after first BonnRouteGlobal and after Global Routing Based Optimization.

		After 2nd BRG	After 1st BRD							
Chip	Router	wACE4	WS	SNS	Slew violations	Shorts	Wire length	Vias	Steiner Scenic 150	Steiner Scenic 200
E_1	Traditional	89.1	-34	-12869	1539	101	12.3616	9028705	3664	229
	DLU	88.4	-37	-8189	799	34	12.3204	9017757	1949	33
E_2	Traditional	90.0	-46	-6439	2695	9	6.8934	3672522	4585	715
	DLU	88.9	-35	-4768	1336	0	6.7343	3646841	1473	144
F_1	Traditional	87.7	-19	-2283	1139	33	8.5267	3941481	2003	120
	DLU	87.8	-26	-2282	1229	22	8.5324	3946702	1781	110
G_1	Traditional	102.5	-296	-88546	760	59455	0.9288	1150479	2189	699
	DLU	96.4	-250	-63968	594	42201	0.8822	1108086	1384	471
G_2	Traditional	77.4	-30114	-30593	463	287	0.8588	449789	15	5
	DLU	75.9	-30115	-30613	475	298	0.8584	445549	6	1
G_3	Traditional	99.2	-153	-6752	145	2450	0.2966	380373	605	130
	DLU	96.6	-104	-4832	86	2204	0.2857	373482	395	92
G_4	Traditional	83.2	-59	0	423	72	0.0963	105605	4	0
	DLU	82.6	-46	0	447	65	0.0955	105226	1	0
G_5	Traditional	75.5	37	0	1139	33	0.0885	86271	3	0
	DLU	71.4	38	0	1229	22	0.0883	85478	1	0

Table 10: Results in the IBM Routing Flow after the second call to BonnRouteGlobal and the first call to BonnRouteDetail.

		After DRBO			After 2nd BRD				
Chip	Router	WS	SNS	Wire length	WS	SNS	Slew violations	Shorts	Wire length
E_1	Traditional	-116	-45408	13.1826	-117	-24479	4050	68	13.1826
	DLU	-96	-43383	13.1170	-43	-19129	3099	24	13.1171
E_2	Traditional	-32	-16742	6.8941	-41	-9063	1600	8	6.8941
	DLU	-24	-14867	6.7338	-39	-6186	807	0	6.7338
F_1	Traditional	-29	-13425	8.5310	-22	-3449	613	18	8.5310
	DLU	-28	-13709	8.5374	-22	-3018	528	21	8.5374
G_1	Traditional	-494	-168773	0.9308	*)	*)	*)	*)	*)
	DLU	-495	-164569	0.8862	*)	*)	*)	*)	*)
G_2	Traditional	-30114	-30600	0.8589	-30114	-30600	457	272	0.8589
	DLU	-30114	-30624	0.8584	-30115	-30624	457	269	0.8584
G_3	Traditional	-5	0	0.2975	-218	-2654	69	5324	0.2975
	DLU	-4	0	0.2868	-184	-2445	68	3759	0.2868
G_4	Traditional	34	0	0.0969	25	0	354	63	0.0969
	DLU	32	0	0.0962	31	0	381	55	0.0962
G_5	Traditional	-36	0	0.0885	35	0	0	21	0.0885
	DLU	-39	0	0.0884	37	0	0	21	0.0884

Table 11: Results in the IBM Routing Flow after Detailed Routing Based Optimization and the second call to BonnRouteDetail. *) At the time of writing, not the whole flow was running through on instance G_1 due to congestion issues.

		End of routing flow						
Chip	Router	WS	SNS	Slew violations	Shorts	Wire length	Vias	DRCs
E_1	Traditional	-96	-16805	2941	68	13.2286	9096609	839
	DLU	-37	-12528	2255	24	13.1629	9088511	305
E_2	Traditional	-37	-6247	1051	8	6.9141	3698352	145
	DLU	-36	-4357	500	0	6.7539	3666115	101
F_1	Traditional	-22	-3273	519	18	8.5575	3969742	467
	DLU	-21	-2949	485	21	8.5617	3964583	465
G_1	Traditional	*)	*)	*)	*)	*)	*)	*)
	DLU	*)	*)	*)	*)	*)	*)	*)
G_2	Traditional	-30114	-30628	456	240	0.8589	450363	*)
	DLU	-30114	-30652	454	239	0.8585	446203	*)
G_3	Traditional	-131	-272	32	5993	0.3022	397932	*)
	DLU	-43	-74	29	4301	0.2914	388368	*)
G_4	Traditional	19	0	354	64	0.0969	106085	*)
	DLU	22	0	381	56	0.0962	105776	*)
G_5	Traditional	36	0	0	21	0.0885	86331	*)
	DLU	37	0	0	21	0.0884	84541	*)

Table 12: Results at the end of the IBM routing flow. *) At the time of writing, not the whole flow was running through on instance G_1 due to congestion issues. Moreover, for all the most recent instances $G_1, \ldots, G_5$ the final check for design rule violations was not working yet.

5 REACH- AND DIRECTION-RESTRICTED STEINER TREES

This section is about the minimum rectilinear Steiner tree problem in the presence of obstacles. Some obstacles may not be traversed at all; others may be traversed only horizontally, only vertically, or in both directions. In any case, the total length of each connected component in the intersection of the tree with the interior of the obstacles is bounded by a constant. For this problem we present a fast 2-approximation algorithm. Compared to [6, 25], which this work is based on, we provide a more thorough case distinction in the proof of the main theorem, closing gaps in the previous proofs. This is joint work with the authors of [25], though the majority of the progress is due to the author of this thesis. Building on [6, 25], we also slightly improve the extraction of the Steiner tree and extend the post-optimization by an edge substitution method from [20]. The latter solves an online maximum cost on tree path problem for which we improve the pre-processing time of the algorithm by [31] from $\mathcal{O}(n \log n)$ to $\mathcal{O}(n)$. Finally, we present results of a new application of our Steiner tree algorithm in BonnRouteGlobal with dynamic local usage to compute lower bounds on the total wire length and the number of vias.

5.1 PRELIMINARIES

Besides routing, another important step in chip design is buffering [5]: repeaters are inserted into the routes, repropagating and strengthening incoming signals. This becomes necessary if a net must power a large downstream capacitance, resulting in long delays. IBM mainly performs buffering before the routing flow that is described in Section 4.5. Nonetheless, some buffers are also inserted during global and detailed routing based optimization. The computation of rectilinear Steiner trees is a central task during buffering [4, 24]. Buffering can be restricted by macro cells, which correspond to areas where no repeaters can be placed. Often, the top-level layout is dominated by big macros, leaving only small gaps for repeaters. Wires, on the other hand, may well reach over some of the macros using higher routing layers. However, unbuffered components of the wires must not become too long to avoid capacitance, slew, and delay violations. This motivates bounding the length of tree components sitting on top of macros. In addition, some macros may reach up to the chip ceiling, preventing any routing on top of them, and others may leave only

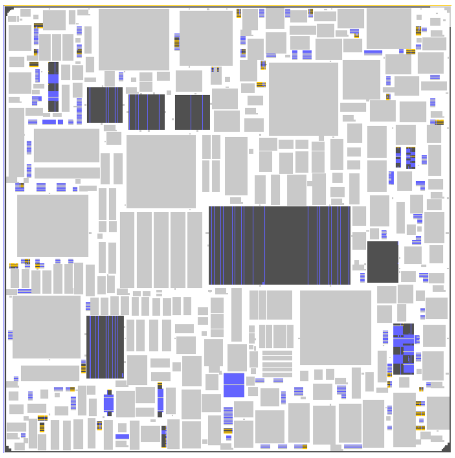


Figure 32: A chip instance with fully blocked area (dark gray), vertically-restricted (blue), horizontally-restricted (orange) and length-restricted-only area (light gray).

a single layer of the chip free, restricting routing to the dimension of that layer. In general, macros can be perforated by routing tracks on single layers to provide routing space for the outer entity, resulting in arbitrary unidirectional routing restrictions. Figure 32 shows a real-world example of the obstacle structure on a chip.

We use the following definitions:

Definition 58. For $X \subseteq \mathbb{R}^2$, X° denotes the *interior* of X, and ∂X denotes the *boundary* of X. The *set of obstacles* consists of three finite sets of rectangles $\mathcal{R}, \mathcal{R}_h, \mathcal{R}_v$. We further define the *horizontally-restricted area* $A_h := \bigcup_{r \in \mathcal{R}_h} r$, the *vertically-restricted area* $A_v := \bigcup_{r \in \mathcal{R}_v} r$ and the *length-restricted (or blocked) area* $A := \bigcup_{r \in \mathcal{R} \cup \mathcal{R}_h \cup \mathcal{R}_v} r$. The complement $\mathbb{R}^2 \setminus A$ is the *unblocked area*. $A_h \cap A_v$ is the *fully blocked* area.

Definition 59. A length-/vertically-/horizontally-restricted obstacle is a maximal closed area $O \subseteq A/A_v/A_h$ whose interior is connected.

Note that all the area of a vertically- or horizontally-restricted obstacle O is also length-restricted, though O itself is not necessarily a length-restricted obstacle in terms of this definition. A direction-restricted obstacle O needs not to be maximal with regard to A and can be a proper subset of a length-restricted obstacle.

While this definition is simple, it still allows for nested holes and obstacles within holes. We can now define the reach- and direction-restricted Steiner tree (RDRST) problem:

Problem 60. REACH- AND DIRECTION-RESTRICTED STEINER TREE

Input: A 5-tuple (S, A, A_h, A_v, L), where A, A_h, and A_v are length-restricted, horizontally-restricted and vertically-restricted

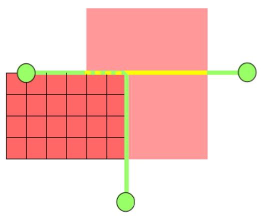


Figure 33: A length-restricted obstacle containing a fully blocked obstacle on the left. For $L =$ "6 squares" the unique optimal solution contains (geometrically) parallel edges.

areas, $S \subset \mathbb{R}^2 \setminus A^\circ$ is a finite set of terminals, and $L \geqslant 0$ the *reach length*.

Output: A shortest *feasible RDRST*, that is a rectilinear Steiner tree T for S that satisfies that

1. each segment in $E(T) \cap A_h^\circ$ is horizontal,
2. each segment in $E(T) \cap A_v^\circ$ is vertical and
3. the length of each connected (geometric) component in $E(T) \cap A^\circ$ is at most L.

A rectilinear Steiner tree for S is a tree T with $S \subseteq V(T) \subset \mathbb{R}^2$, such that each edge $\{v, w\} \in E(T)$ is horizontal or vertical (v and w have the same y-coordinates or x-coordinates). Its *length* $l(T)$ is the sum of the lengths of line segments represented by its edges. For easier notation, we denote by $E' \subseteq E(T)$ a set of graph edges and also a set of (geometric) line segments in the plane. For $r, s \in \mathbb{R}^2$, an r-s-*path* is an embedded Steiner tree for $\{r, s\}$.

Note that by definition we always have $A_h, A_v \subseteq A$. The edges of a feasible RDRST are not allowed to intersect $A_h^\circ \cap A_v^\circ$. The feasibility of an embedded Steiner tree depends only on the restricted areas A, A_h, and A_v, and not on the structure of the underlying rectangle sets $\mathcal{R}, \mathcal{R}_h$, and $\mathcal{R}_v$. We say that a Steiner tree is *reach-aware* if it is feasible with regard to the length-restricted obstacles, analogously *direction-aware* if it is feasible with regard to the direction-restricted obstacles. There is no restriction for edges on the boundary ∂A of the blocked area. Moreover, parallel edges are allowed and may be necessary on length-restricted obstacles; see Figure 33.

We make the simplifying assumption that no terminal is located within the interior of an obstacle, that is, $S \subset \mathbb{R}^2 \setminus A^\circ$. If terminals are allowed to be in the interior of the blocked area A, it becomes $\mathcal{NP}$-complete to decide whether a feasible solution exists, as a reduction from the rectilinear Steiner tree problem shows: For $L > 0$, an instance of the latter can be placed inside a large length-restricted obstacle, such that all terminals are further than L from unblocked area.

Hence no terminal can be connected to the unblocked area in a feasible RDRST. Therefore, a solution to the RDRST problem exists if and only if there is a rectilinear Steiner tree for the terminals of length at most L.

Under the reasonable restriction that the distance of each terminal to unblocked area is at most L, it is still $\mathcal{NP}$-hard to compute a 2-approximation for the RDRST problem as can be seen in [6]. This result is extended in [9] to show that it is even $\mathcal{NP}$-hard to compute an $f(k)$-approximation where k is the input size of the instance and $f : \mathbb{N} \to \mathbb{R}$ an arbitrary function.

With the assumption that no terminal is located within the interior of an obstacle, the RDRST problem remains $\mathcal{NP}$-hard, since it generalizes the $\mathcal{NP}$-hard rectilinear Steiner tree problem.

The special case where $L = 0$ (or $A = A_h \cap A_v$) is known as the *Obstacle-Avoiding Rectilinear Steiner Tree (OARST) problem*, which has been studied extensively. Several heuristics and approximation algorithms were proposed by [3, 13, 14, 30, 37, 43]. A near-linear-time PTAS for the analog of the obstacle-avoiding Steiner tree problem in any uniform orientation metric was presented in [47]. ObSteiner, an obstacle-avoiding extension of the exact GeoSteiner algorithm [55], was developed in [29]. Finally, algorithms for multilayer obstacle-avoiding Steiner trees were proposed in [36, 38, 42].

The special case in which all obstacles can be traversed in both directions was introduced by [46], who also developed approximation algorithms based on an extended Hanan grid. Typically, the extended Hanan grid has a quadratic size. The algorithm of [25] builds on the sub-quadratic visibility graph of [16] to obtain a smaller graph size and faster running time. [6] extends this algorithm to also account for direction-restrictions on obstacles. In this thesis, we extend and improve upon [6, 25].

Heuristics for similar models were proposed by [57, 58], obeying a slew limit instead of a reach length. It was extended to perform an explicit buffer insertion [57].

Like many algorithms for similar variants [37, 43, 46], our algorithm has three main phases: First, we construct a *visibility graph* for the union of the terminals S and obstacle corners that contains a shortest path between every pair of vertices. During the second phase, we use a Dijkstra-Kruskal approach [40] to compute a Steiner tree for the terminal set S in this graph. Finally, the Steiner tree is post-optimized by local search heuristics. Since the visibility graph is shortest-path preserving, our algorithm achieves an approximation factor of two, given by the Steiner ratio in graphs.

5.2 PATH STRUCTURE THEOREM

The aim of this section is to prove Theorem 64 about the structure of feasible paths among length- and direction-restricted obstacles.

Our algorithm computes a visibility graph extending a construction of Clarkson et al. [16] for the special case $L = 0$. Clarkson's version used to compute shortest paths avoiding polygonal (not necesserily rectilinear) obstacles. We define the *bounding box* $BB(x, y)$ of two points $x = (x_1, x_2)$ and $y = (y_1, y_2)$ as the closed axis-aligned rectangle

$$BB(x, y) = \{(z_1, z_2) \in \mathbb{R}^2 : \min(x_i, y_i) \leqslant z_i \leqslant \max(x_i, y_i) \ (i \in \{1, 2\})\}$$

Clarkson's visibility graph is based on the observation that every obstacle-avoiding shortest path can be modified (while preserving the length) in such a way that the new path can be decomposed into shortest sub-paths between terminals and blockage corners such that the interior of the bounding boxes spanned by the sub-path endpoints neither intersect blockages nor terminals and its length equals the distance between its endpoints.

The construction in [16] ensures that between each pair of vertices, there is a so-called median line, a vertical line to which both vertices are connected by a horizontal segment (if possible). If the bounding box of two consecutive points does not intersect any obstacle, this is always possible. This construction is indeed shortest-path preserving. Note that horizontal instead of vertical median lines could be used equivalently.

For our length- and direction-restricted obstacles shortest paths may reach over obstacles. Therefore, we cannot require that the interior of the bounding box spanned by two consecutive sub-path endpoints does not intersect obstacles. However, we can at least require that for a suitable set of points, a shortest path is decomposable into subpaths such that the bounding box spanned by the endpoints of a subpath does not contain any other such point, and the length of each subpath equals the distance of its endpoints. This special set of path endpoints, which we will call base points in the following, is defined as follows.

Let $\mathcal{K} := \{\text{corners of } A\} \cup \{\text{corners of } A_h\} \cup \{\text{corners of } A_v\}$ denote the set of all obstacle corners.

Furthermore, for any $x \in \mathbb{R}^2$, we define a set $\mathcal{P}_x$ of *projection points* and a set of *escape points* $\mathcal{E}_x$ as follows (see also Figure 34). For each of the four directions north, east, south and west, if an arbitrary small step from x in that direction ends in the interior of an obstacle, we add the nearest point in that direction on the boundary of unblocked area and different from x as a projection point to $\mathcal{P}_x$. If an arbitrary small step from x to east or west x ends in A_h we add the nearest point in ∂A_h different from x as *horizontal escape point* to $\mathcal{E}_x$. Analogously, we add *vertical escape points* with respect to A_v to $\mathcal{E}_x$.

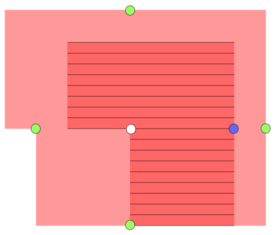


Figure 34: A length-restricted obstacle containing another horizontally-restricted obstacle. The (white) corner of A_h in the middle has four projection points (green) and one escape point (blue).

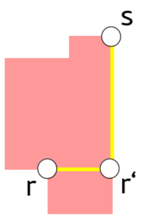


Figure 35: If L is sufficiently small, the shortest path from r to s is unique. $BB(r, s)$ is not free, and hence the projection point r' of r is required.

Finally, we define the set of *base points* as

$$\mathcal{B} := S \cup \mathcal{K} \cup \bigcup_{x \in \mathcal{K}} (\mathcal{P}_x \cup \mathcal{E}_x)$$

We will construct a visibility graph that contains a shortest path for each pair of base points.

Since each obstacle corner has at most four projection points and four (two horizontal and two vertical) escape points, $|\mathcal{B}| \leqslant |S| + 9|\mathcal{K}|$.

Remark 61. $\mathcal{B}$ contains the projection and escape points of all base points in $\mathcal{B} \cap A^\circ$.

Proof. All base points in $\mathcal{B} \cap A^\circ$ are either corners of direction-restricted obstacles, for which $\mathcal{B}$ contains their projection and escape points by definition, or escape points. Let $p \in \mathcal{B} \cap A^\circ$ be an escape point of some $q \in \mathcal{K}$. Then, p is located on the boundary e of some direction-restricted obstacle. If p is an obstacle corner, $\mathcal{B}$ contains its projection and escape points by definition. Otherwise, the projection points along the boundary e are either the corner points of that boundary or their projection points. The projection and escape points orthogonal to e are either q or its projection or escape points. □

In the remainder of the paper, we will often use the notion of free bounding boxes with a constant obstacle structure.

Definition 62. (Free Bounding Box) Given a set of base points $\mathcal{B}$ and $s, t \in \mathbb{R}^2$, the bounding box $BB(s, t)$ of s and t is *free* if it does not contain any base points except for (potentially) s or t.

Note that free bounding boxes may intersect obstacles whose corners are located outside the box. Being free also depends on the boundary of the bounding box, and thus it depends on the choice of its spanning points s and t.

The concept of base points and free bounding boxes allows us to generalize Clarkson's approach to our setting.

Definition 63. A rectilinear s-t-path P is called an *ℓ_1-path* if

$$\sum_{\{x,y\}\in E(P)} \|x-y\|_1 = \|s-t\|_1$$

Given obstacle areas A, A_h, A_v and a reach length L, P is called a *feasible ℓ_1-path* if it is feasible for the instance $(\{s,t\}, A, A_h, A_v, L)$.

Let P be an s-t-path, and let $X \subseteq \mathbb{R}^2$. Let $\{p_1,\dots,p_k\} = X \cap V(P)$ such that $p_1,\dots,p_k$ appear along P from s to t in this order. We say that P is X-*simple* if for $i \in \{0,\dots,k\}$

- $BB(p_i, p_{i+1})$ is free; and
- the subpath of P from p_i to p_{i+1} is an ℓ_1-path.

where $p_0 = s$ and $p_{k+1} = t$.

The statement of the main theorem of this section is that any feasible shortest path can be transformed into a $\mathcal{B}$-simple path:

Theorem 64. *Let (S, A, A_h, A_v, L) be an RDRST instance and $s, t \in \mathcal{B}$. Every s-t-path P for $(\{s,t\}, A, A_h, A_v, L)$ of minimum length can be transformed into a $\mathcal{B}$-simple feasible s-t-path $\bar{P}$ of the same length.*

Figure 35 shows that projection points are needed as base points, to enable the transformation into $\mathcal{B}$-simple paths. An analogous example with direction-restricted obstacles shows the necessity of escape points.

To prove Theorem 64, we will first prove a specialized statement for ℓ_1-paths in Lemma 66. Then, in Lemma 67 and Lemma 68, we will show that a path P that is not an ℓ_1-path can be subdivided into ℓ_1-subpaths that start and end at base points. For each of these subpaths, we will apply Lemma 66, yielding a path that satisfies the conditions of Theorem 64.

Aside from some trivial cases, in the proof of Lemma 66 we will transform the given path in three different ways, depending on the obstacle structure. We use the following notion of *tightly-covered* boxes (Figure 36):

Definition 65. Let $s, t \in \mathbb{R}^2$. We say that their bounding box $BB(s,t)$ is tightly-covered, if there is an obstacle O (recall Definition 59) that is either length-restricted or vertically-restricted (inside a larger length-restricted obstacle) that is aligned with the bottom of $BB(s,t)$, extends past the upper and right boundary of $BB(s,t)$ and either aligns with or extends past the left boundary of $BB(s,t)$. The corners of $BB(s,t)$ may be located on the boundary ∂O or in its interior O°.

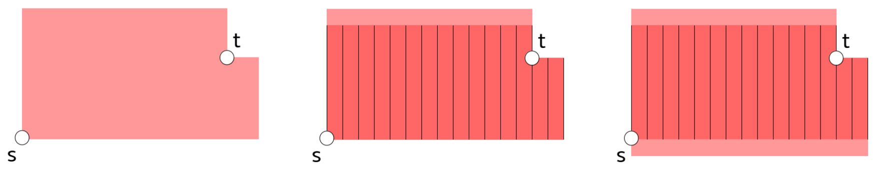


Figure 36: On the left, the bounding box of s and t is tightly-covered by the length-restricted obstacle. In the middle, it is both tightly-covered by the vertically-restricted and by the containing length-restricted obstacle. On the right, it is tightly-covered by the vertically-restricted obstacle but not by the containing length-restricted obstacle, that extends past the lower boundary of $BB(s,t)$.

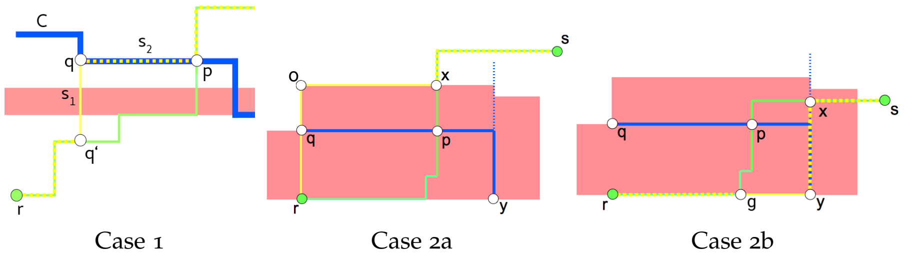


Figure 37: The staircase C (blue), the path P (green), the modified path P' (yellow).

In contrast to the notion of tightly-covered boxes that requires an obstacle to align with the lower boundary of the box, we say that the bounding box $BB(s,t)$ of $s,t \in \mathbb{R}^2$ is *completely covered* by an obstacle O if $BB(s,t) \setminus \{s,t\} \subset O^\circ$. Similarly, a path P from s to t is completely covered by O if $P \setminus \{s,t\} \subset O^\circ$. Note that in both cases, s and t may be located in the interior or on the boundary of O.

Lemma 66. *Let* (S, A, A_h, A_v, L) *be an RDRST instance, and let* $r, s \in \mathcal{B} \cup (\mathbb{R}^2 \setminus A^\circ)$ *be two points, connected by a feasible* ℓ_1*-path* P*. Then* P *can be transformed into a feasible* $\mathcal{B}$*-simple* r-s-ℓ_1*-path* P'.

Proof. Let P be an r-s-path as in the statement of the lemma. We proceed by induction on $|(BB(r,s) \setminus \{r,s\}) \cap \mathcal{B}|$. If $|(BB(r,s) \setminus \{r,s\}) \cap \mathcal{B}| = 0$, then $BB(r,s)$ is free and since P is an ℓ_1-path, it follows that P is $\mathcal{B}$-simple. If r and s can be joined by an axis-parallel line, we subdivide P at each base point located along the line to obtain P'.

Therefore, we may assume (after translation, rotation, and mirroring) that $r = (0,0)$ and $s = (x,y)$ with $x, y > 0$, and moreover, $BB(r,s)$ is not free. We define a "staircase" C as the upper-right boundary of $\{u \in \mathbb{R}^2_{\geqslant 0} : BB(r,u) \text{ is free}\}$ (see Figure 37). Let p be the first intersection point of P with C starting from r. Since $BB(r,s)$ is not free, it follows that p exists ($p = s$ is possible). By subdividing the edge of $E(P)$ containing p, we may assume that $p \in V(P)$.

From the definition of C, it follows that $BB(r,p)^\circ \cap \mathcal{B} = \emptyset$. Assume that $(\partial BB(r,p)) \cap (\mathcal{B} \setminus \{r,p\}) = \emptyset$, that is, there are no other base points on the boundary of $BB(r, p)$ other than possibly r and p. It follows that $p \in \mathcal{B}$, as otherwise $p \in C$ was impossible. Hence $BB(r, p)$ is free and we can apply induction to the remaining subpath of P from p to s.

Now assume that $(\partial BB(r, p)) \cap (\mathcal{B} \setminus \{r, p\}) \neq \emptyset$. Since p is at the boundary of the points $u \in \mathbb{R}^2_{\geqslant 0}$ for which $BB(r, u)$ is free, r and p cannot be located on an axis-parallel line; i.e. $BB(r, p)^\circ \neq \emptyset$. By symmetry, we may assume that the first edge of P containing p is vertical, and thus, it meets C from below (otherwise flip x- and y-coordinates of the instance). By the definitions of C and p, there cannot be any base points on the right, lower and left boundary of $\partial BB(r, p)$, except for possibly r and p and the upper left corner of $BB(r, p)$. Therefore, the upper boundary of $\partial BB(r, p)$ contains a base point different from p; let q be the left-most such base point. Note that q is either the left end of C or a corner of the staircase.

The idea is to reroute P to P' through some base point b for which $BB(r, b)$ is free, such that we can apply induction on the remaining subpath $P'_{[b,s]}$. The reroute is conducted such that the length over blockages does not increase so that the final path satisfies the reach condition inductively. We will re-route P in three different ways, depending on the structure of the obstacles and P:

1. P leaves r upwards or $BB(r, p)$ is not tightly-covered (Definition 65):

   Let q' be the first intersection point of P with the vertical line through q. We obtain P' from P by replacing the q'-p-subpath of P by a vertical q'-q-segment s_1 and a horizontal q-p-segment s_2 (case 1 in Figure 37).

2. P leaves r to the right and $BB(r, p)$ is tightly-covered by a length-restricted obstacle O:

   The projection point of q is located on the lower boundary of $BB(r, p)$. By the definition of C, there cannot be any base points on the lower boundary of $BB(r, p)$ apart from r, hence q must be located directly above r. Let y be the intersection of the lower boundary of O with the staircase C (see cases 2a and 2b in Figure 37). Such a y must exist since, by definition, the staircase C is restricted by the lower right corner point of O. Starting at r, let x be the first intersection of P after p with the boundary of O or with the vertical line through y. Such an intersection must exist, as by the assumption of the lemma s would be a base point if it were located inside the same obstacle O. But then, s could not be inside O and left to the vertical line through y, as this would imply a projection point on the line from r to y.

   a) If x is not on the vertical line through y, let o be the intersection of a vertical line through q and a horizontal line through x. We obtain P' by replacing the $r - x$-subpath by the rectilinear path from r through q and o to x.

b) Otherwise, we replace the $r - x$-subpath of P by the rectilinear path from r through y to x to obtain P'.

3. P leaves r to the right and $BB(r, p)$ is not tightly-covered by a length-restricted obstacle but by a vertically-restricted obstacle:

   We define P' analogously to case 2a with regard to the vertically-restricted obstacle. For details, see Claim 72 in Section 5.2.1.

In all cases, the length of the path is preserved. We have to show feasibility of P' with regard to the obstacles. Note that Definition 65 is bound to length- and vertically-restricted obstacles only, because there cannot be a horizontally-restricted obstacle that tightly-covers $BB(r, p)$. Otherwise, P would be infeasible as we assumed P to meet p from below.

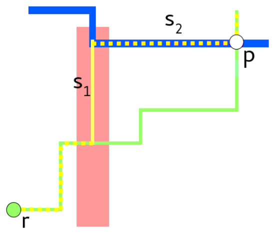
In cases 2a and 2b, the path P might be partially unblocked as shown in Figure 37. However, the path P' from case 1 could be covered by O to a larger extent making it potentially infeasible for cases 2a and 2b if L is small enough. This is why P has to be rerouted differently in these cases. On the other hand we will see that the fact that O aligns with the lower boundary of $BB(r, p)$ implies simple obstacle structures in cases 2a and 2b.

To show the feasibility of P' in all cases, the length restrictions and the direction restrictions can be handled independently of each other. For the sake of simplicity, here we will only show the feasibility with regard to length-restricted obstacles. Feasibility with regard to direction-restricted obstacles can be shown similarly using the escape points instead of the projection points, as may be seen in Claim 70 and Claim 71 in Section 5.2.1.

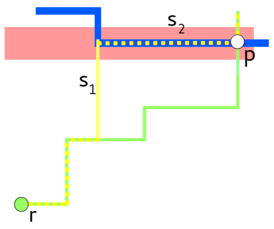
Claim 66.1. *In case 1, where* $BB(r, p)$ *is not tightly-covered (Definition 65) or* P *leaves* r *upwards, the path* P' *is feasible with regard to length-restricted obstacles.*

Proof. Consider a length-restricted obstacle O that intersects $BB(r, p)^\circ$. Without loss of generality assume $BB(r, p)^\circ \cap O$ contains only one connected component (if not consider each connected component separately). By the definition of the staircase C it follows that $BB(r, p)^\circ \cap \mathcal{B} = \emptyset$, therefore O cannot have corners inside $BB(r, p)^\circ$. Moreover, since $BB(r, p) \cap \mathcal{B} \setminus \{r\}$ is contained in the upper boundary of $\partial BB(r, p)$, O° must intersect the lower or right boundary of $BB(r, p)$, that is, O must be at least one of

- *horizontally spanning*, which means that O° intersects the right boundary of $\partial BB(r, p)$ (that is, O extends past the right boundary) and O intersects the left boundary of $\partial BB(r, p)$, or



Type α: vertically spanning and length-restricted



Type β: horizontally spanning and length-restricted

Figure 38: Possibly problematic obstacles.

- *vertically spanning*, which means O° intersects the lower boundary of $\partial BB(r, p)$ (that is O extends past the lower boundary) and O intersects the upper boundary of $\partial BB(r, p)$.

See case 1 in Figure 37 for a horizontally spanning obstacle. Note that O can be both horizontally and vertically spanning at the same time. To prove that P' is feasible with regard to length restrictions, we will show that P' is feasible with respect to each length-restricted obstacle O intersecting $BB(r, p)^\circ$ separately.

Assume O is horizontally spanning and $s_2 \cap O^\circ = \emptyset$. Then we have $l(E(P') \cap O^\circ) \leqslant l(E(P) \cap O^\circ)$, and thus, P' is feasible with respect to O since P is feasible. If O is vertically spanning and $s_1 \cap O^\circ = \emptyset$, analogously P' is feasible due to $l(E(P') \cap O^\circ) \leqslant l(E(P) \cap O^\circ)$.

Now suppose that there is a length-restricted obstacle O which is either vertically spanning with $s_1 \cap O^\circ \neq \emptyset$ (*type* α), or horizontally spanning with $s_2 \cap O^\circ \neq \emptyset$ (*type* β, Figure 38). Recall that obstacles can be of both types at the same time, if they cover the entire bounding box $BB(r, p)$. Using the projection points we will show that in both cases the obstacles must contain $BB(r, p)$, completely covering $P_{[r,p]}$. This shows feasibility of P' since P is feasible and P' has the same length as P.

Type α: Suppose that O is a length-restricted and vertically spanning obstacle and $s_1 \cap O^\circ \neq \emptyset$. We want to show that O must reach beyond the left boundary of $BB(r, p)$: If q is on the upper left corner of $BB(r, p)$, this follows by the fact that $s_1 \cap O^\circ \neq \emptyset$. Otherwise, there are four possibilities for the structure of O around q (see Figure 39). If the left boundary of the obstacle did not reach beyond r, there would be a base point (a projection point of q in cases Q1 and Q2, using Remark 61, or an obstacle corner in cases Q3 and Q4) on the upper boundary of $\partial BB(r, p)$ left of q, contradicting the choice of q.

Now that we have proven that O must reach beyond the left boundary of $BB(r, p)$, there are two cases to consider at r (Figure 40). In both cases the right boundary of O must reach beyond p, as otherwise the projection point of r would be on the lower boundary of $\partial BB(r, p)$. In case R1 r has a projection point because it is a corner point, in case

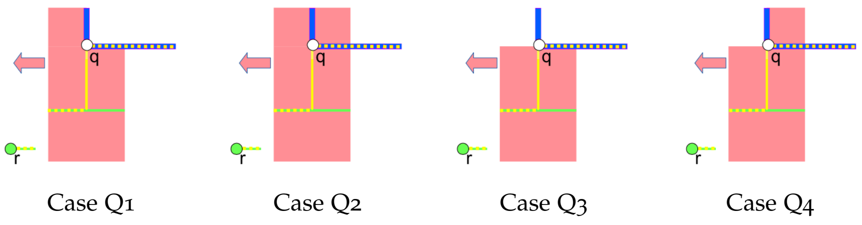


Figure 39: Possible cases around q.

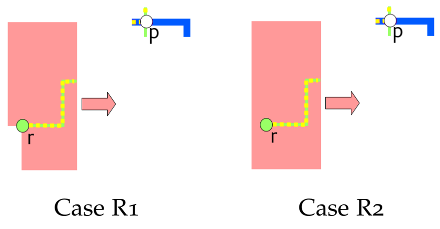


Figure 40: Possible cases around r.

R2 due to Remark 61 and by the lemma's assumption that r is a base point if it is located in the interior of A.

Hence we know that O spans the entire bounding box of r and p and extends past the left, lower and right boundary of $BB(r, p)$. Also, the lower right corner of $BB(r, p)$ must be inside O, as otherwise it would be a corner of O and hence a base point located on the right boundary of $BB(r, p)$. It follows that $P_{[r,p]}$ is completely covered by O, which is why $l(E(P') \cap O^{\circ}) \leqslant l(E(P) \cap O^{\circ})$. Thus, P' is feasible with respect to O.

Type β*:* Suppose that O is a length-restricted and horizontally spanning obstacle and $s_2 \cap O^{\circ} \neq \emptyset$, hence O extends past the upper boundary of $BB(r, p)$. O must at least reach until exactly r to the bottom, because otherwise the projection point of q would be located in $BB(r, p)^{\circ}$ or on the left boundary of $BB(r, p)$ between r and q. As O is horizontally spanning it must reach beyond p to the right.

If P leaves r upwards, let g be the point where P first goes to the right. $P_{[g,p]}$ is completely covered by O. Otherwise, if P leaves r to the right, $BB(r, p)$ is not tightly-covered by assumption and hence O must also extend past the lower boundary. Thus $P_{[r,p]}$ is completely covered by the length-restricted obstacle O. In both cases $l(E(P') \cap O^{\circ}) \leqslant l(E(P) \cap O^{\circ})$ and therefore P' must be feasible with respect to O.

It follows that P' is feasible with regard to length restrictions proving the statement of the claim. □

Claim 66.2. *In case 2, where* P *leaves* r *to the right and* $BB(r, p)$ *is tightly-covered by a length-restricted obstacle* O*, the modified path* P' *is feasible with respect to length-restricted obstacles.*

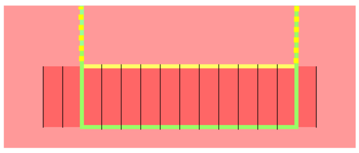


Figure 41: The vertically-restricted obstacle O, part of path P (green), the rerouted path (yellow).

Proof. In case 2a, O cannot have any corners on the upper boundary from o to x, as otherwise their projection points were located on the line from r to y. Thus, the segment from o to x runs at the boundary of O. The segment from r to o possibly intersects O, but $l(E(P) \cap O^{\circ}) \geqslant l(E(P'_{[r,o]}) \cap O^{\circ})$ because O tightly-covers $BB(r,p)$ and thus P' is feasible.

In case 2b, let g be the most right intersection point of P with the horizontal line from r to y. g is located to the right of r as, by assumption, P leaves r to the right. P and P' only differ between g and x, while $P_{[g,x]}$ is completely covered by O which tightly-covers $BB(r,p)$. Therefore, also P' must be feasible with regard to length restrictions. □

We now conclude the proof of Lemma 66: In case 1 and 2a $BB(r,q)$ is free, the same holds for $BB(r,y)$ in case 2b. q is a base point by definition. y must be a base point, since otherwise the staircase C would not end at y. Applying induction to the subpath of P' from q to s (case 1 and 2a) and from y to s (case 2b) yields the statement of the lemma. □

Given a path P as in Theorem 64, that is not necessarily an ℓ_1-path, we want to apply Lemma 66 to ℓ_1-subpaths of P. This will finally prove the theorem if the subpaths start and end at base points. Splitting a path into ℓ_1-subpaths that start and end at base points is the content of the following two lemmas. The first one shows this for paths that are fully contained in length-restricted area. The second one deals with general paths, but has stronger preconditions.

Lemma 67. *Let obstacles A, A_h, A_v, a reach length L, $a, b \in \mathcal{B} \cup (\mathbb{R}^2 \setminus A^{\circ})$ be given. Let P be a shortest feasible path for the instance $(\{a,b\}, A, A_h, A_v, L)$ that is completely covered by length-restricted obstacles. Then P can be subdivided into ℓ_1-subpaths $P = P_{[a,r_1]}, P_{[r_1,r_2]}, \dots, P_{[r_{m-1},r_m]}, P_{[r_m,b]}$ such that $r_i \in \mathcal{B}$ for $i = 1, \dots, m$.*

Proof. Assume that it is not possible to divide P into such ℓ_1-subpaths. We will show that P can be shortened while remaining feasible, contradicting its optimality. P must contain a non-ℓ_1-subpath that cannot be decomposed. Without loss of generality, after rotation and reflection, there must exist a U-shaped subpath as depicted in Figure 41, with no base points on the lower side. The only reason that prevents us from moving up the lower segment infinitesimally and thereby shortening the path could be some vertically-restricted obstacle O right above. There cannot be any corners at the upper boundary of O, since otherwise there were escape points at the lower boundary at which we could split up the path. For the same reason, a, b cannot be inside O. Hence, we can reroute P along the upper boundary, reducing its length also in this case. □

The following lemma assumes that we have a subdivision of a path into ℓ_1-paths that start and end at base points inside blockages but at arbitrary points outside of blockages. If additionally consecutive base points on the path have free bounding boxes, one can show that the subdivision can be modified to start and end at base points also outside of blockages.

Lemma 68. *Let obstacles* A, A_h, A_v*, a reach length* L*,* $a, b \in \mathcal{B} \cup \left(\mathbb{R}^2 \setminus A^\circ\right)$ *be given. Let* $P = P_1, \ldots, P_n$ *be a shortest feasible path for the instance* $(\{a, b\}, A, A_h, A_v, L)$*, such that for the subpath* P_i *(*$i = 1, \ldots, n$*), starting at* r_i *and ending at* r_{i+1}*, it holds that*

- P_i *is an* ℓ_1*-path with* $r_i, r_{i+1} \in \mathcal{B} \cup \left(\mathbb{R}^2 \setminus A^\circ\right)$*,*
- *for* $i < n$*,* P_i *combined with* P_{i+1} *is not an* ℓ_1*-path and*
- *the bounding box of successive points* $p, q \in \mathcal{B} \cup \{r_i, r_{i+1}\}$ *on* P_i *is free.*

Then P *can be subdivided into* ℓ_1*-subpaths* $P = P'_1, \ldots, P'_m$ *such that for all* P'_i *from* r'_i *to* r'_{i+1} *it holds that* $r'_i, r'_{i+1} \in \mathcal{B} \cup \{a, b\}$*.*

Proof. Assume that this is not possible. We will show that P can be made shorter, contradicting its optimality. Without loss of generality, there must exist a U-shaped subpath with no base points on its lower side, enclosed by two ℓ_1-subpaths P_i and P_{i+1}, as depicted in Figure 42. Starting at r_i, let q the last base point on P_i before r_{i+1} (or r_i if there is no such base point). Let s be the first base point on P_{i+1} after r_{i+1} (or r_{i+2} if there is no such base point). If the lower side of the U-shape is unblocked, the only reason which could prevent us from shortening P by moving it up could be a length-restricted (possibly also vertically-restricted) obstacle O aligning with the U-shape as shown in case W1 in Figure 42. If the lower side of the U-shape is blocked, the only such reason could be a vertically-restricted obstacle as in case W2 in Figure 42. In case W1 we might violate the reach length by moving it up, in case W2 we would violate the direction restriction of O.

In both cases, by assumption, the bounding boxes $BB(q, r_{i+1})$ and $BB(r_{i+1}, s)$ are free, hence O cannot have corners inside the bounding boxes and must extend past the right boundary of $BB(r_{i+1}, s)$ and past the left boundary of $BB(q, r_{i+1})$. It follows that O cannot extend

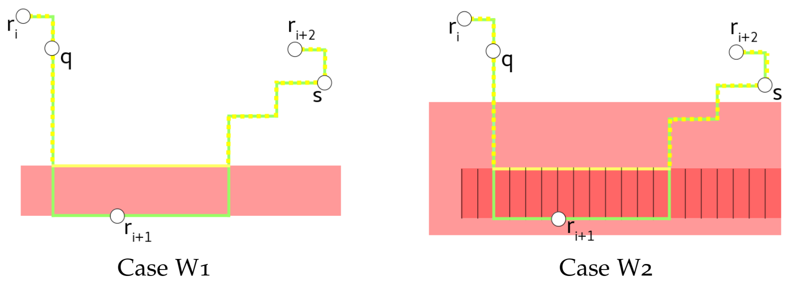


Figure 42: Possible cases in Theorem 64. The modified path P' in yellow.

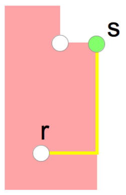


Figure 43: This instance shows that Lemma 66 does not hold without the assumptions on r, s. L is chosen small enough such that the shortest feasible path is unique.

past q or s to the top, as otherwise their projection points (case W1) or escape points (case W2) were located inside $BB(q, r_{i+1})$ or $BB(r_{i+1}, s)$. Hence, we can shorten P as depicted in Figure 42.

□

We are now ready to prove Theorem 64 on page 105, that states that for $s, t \in \mathcal{B}$ any shortest feasible s-t-path P for the instance $(\{s, t\}, A, A_h, A_v, L)$ can be transformed into a $\mathcal{B}$-simple feasible s-t-path of the same length.

Proof. (Theorem 64) We want to subdivide P at base points into ℓ_1-subpaths and then apply Lemma 66 on these subpaths. For this, let $P = P_1, ..., P_m$ an arbitrary subdivision of P into ℓ_1-subpaths which is always possible. By Lemma 67 and looking at the parts of P covered by lenght-restricted area, we can at least assume that all division points are located in $\mathbb{R}^2 \setminus A^\circ$ or are base points. Hence we can apply Lemma 66 on each of these subpaths, resulting in $P'_1, ..., P'_m$. $P' := P'_1, ..., P'_m$ satisfies the stronger preconditions of Lemma 68. Thus, we can subdivide the whole path P' at base points into ℓ_1-subpaths. Finally apply Lemma 66 on these ℓ_1-subpaths, and the resulting path $\bar{P}$ has the desired properties. □

Corollary 69. *Let $s, t \in \mathcal{B} \setminus A^\circ$ be two base points not in the interior of length-restricted obstacles. Let P be a shortest feasible path from s to t, modified according to Theorem 64. If P crosses a base point in the interior of a length-restricted obstacle, then P both enters and leaves this obstacle*

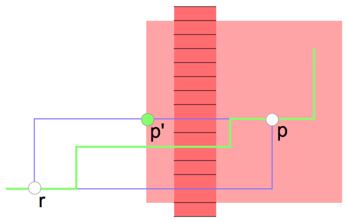


Figure 44: A priori situation in Corollary 69 (the proof shows $r = p'$).

on non-terminal base points (i.e. base points which are not contained in the terminal set S*).*

Proof. Let p be the last base point on P in the interior of the length-restricted obstacle and r be the following one. r is located outside the obstacle or on its boundary and the bounding box of p and r must be free by assumption (Figure 44). The projection point p' of p lies in the bounding box of p and r. Thus r must be equal p', which is a non-terminal base point. □

5.2.1 *Direction-Restricted Obstacles*

For completeness, here we prove feasibility of the modified paths in Lemma 66 with regard to direction-restricted obstacles. This can be accomplished in a similar manner as the proof for length-restricted obstacles. The following claims use the notation of the proof of Lemma 66.

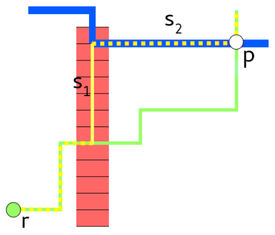
Claim 70. *Given that* P *leaves* r *upwards or* $BB(r,p)$ *is not tightly-covered, the path* P' *in case 1 is feasible with regard to direction-restricted obstacles.*

Proof. Let O a direction-restricted obstacle intersecting $BB(r,p)^\circ$. Without loss of generality assume $BB(r,p)^\circ \cap O$ contains only one connected component (if not consider each connected component separately). As seen in the proof of Lemma 66 O must be horizontally or vertically spanning.

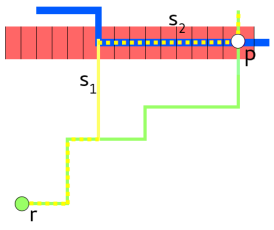
Assume O is horizontally spanning and $s_2 \cap O^\circ = \emptyset$. If P' intersects O°, P does the same. As P is feasible and crosses O in vertical direction O cannot be horizontally-restricted and thus P' is feasible with respect to O as well. Now assume O is vertically spanning and $s_1 \cap O^\circ = \emptyset$. Analogously, if P' intersects O°, P does the same. As P is feasible O cannot be vertically-restricted and thus P' is feasible with respect to O.

Now suppose that there is a direction-restricted obstacle O which is either vertically spanning with $s_1 \cap O^\circ \neq \emptyset$ (*type* γ), or horizontally spanning with $s_2 \cap O^\circ \neq \emptyset$ (*type* δ, Figure 45).

Using the escape points we will show that in both cases the obstacles must contain $BB(r,p)$, completely covering $P_{[r,p]}$. This leads to a contradiction as P is feasible. Hence, such obstacles cannot exist.



Type γ: vertically spanning and horizontally-restricted



Type δ: horizontally spanning and vertically-restricted

Figure 45: Possibly problematic obstacles.

Type γ: Suppose that O is a direction-restricted and vertically spanning obstacle and $s_1 \cap O^\circ \neq \emptyset$. O must be horizontally-restricted as otherwise P would be infeasible. We want to show that O must reach beyond the left boundary of $BB(r, p)$: If q is on the upper left corner of $BB(r, p)$, this follows by the fact that $s_1 \cap O^\circ \neq \emptyset$. Otherwise, there are four possibilities for the structure of O around q (see Figure 39). If the left boundary of the obstacle did not reach beyond r, there would be a base point (an escape point of q in cases Q1 and Q2, using Remark 61, or an obstacle corner in cases Q3 and Q4) on the upper boundary of $\partial BB(r, p)$ left of q, contradicting the choice of q.

Now that we have proven that O must reach beyond the left boundary of $BB(r, p)$, there are two cases to consider at r (Figure 40). In both cases the right boundary of O must reach beyond p, as otherwise the escape point of r would be on the lower boundary of $\partial BB(r, p)$. In case R1 r has an escape point because it is a corner point, in case R2 due to Remark 61 and by the assumption of the lemma that r is a base point if it is located in the interior of A.

Hence we know that the horizontally-restricted obstacle O spans the entire bounding box of r and p and extends past the left, lower and right boundary of $BB(r, p)$. This contradicts the fact that P is feasible, hence such an obstacle O cannot exist.

Type δ: Suppose that O is a direction-restricted and horizontally spanning obstacle and $s_2 \cap O^\circ \neq \emptyset$, hence O extends past the right and upper boundary of $BB(r, p)$. As P crosses O in vertical direction O must be vertically-restricted. It follows that O must at least reach until exactly r to the bottom, because else the southern escape point of q would be located in $BB(r, p)^\circ$ or on the left boundary of $BB(r, p)$ between r and q. If P leaves r upwards P later has to cross O in horizontal direction constituting a contradiction to its feasibility. If P leaves r to the right, by assumption O does not tightly-cover $BB(r, p)$ and hence must also extend past the lower boundary of $BB(r, p)$. Again this contradicts the fact that P is feasible, hence such an obstacle O cannot exist.

Therefore, P' is feasible with regard to direction restrictions. $\square$

Claim 71. *Given that* P *leaves* r *to the right and* $BB(r,p)$ *is tightly-covered by a length-restricted obstacle* O*, the modified paths* P' *in cases 2a and 2b are feasible with regard to direction-restricted obstacles.*

Proof. Note that any direction-restricted obstacle must be contained in O. First consider case 2a: As the path segment $o-x$ of P' runs on the boundary of O it cannot interfere with any direction-restricted obstacle. If the path segment $r-q$ or $q-o$ intersected with the interior of a horizontally-restricted obstacle, the very same obstacle would also have to intersect the path P, because otherwise its corners would imply projection points on the line from r to y which is impossible. Hence P would be infeasible which is a contradiction.

Analogously feasibility with regard to direction-restricted obstacles can be shown in case 2b: The path segment $r-y$ is feasible because it runs on the boundary of O and a horizontally-restricted obstacle intersecting the path segment $y-x$ would also intersect the feasible path P. □

The following claim shows how to modify the path in case 3 of the proof of Lemma 66:

Claim 72. *Given that* P *leaves* r *to the right and* $BB(r,p)$ *is not tightly-covered by a length-restricted obstacle but by a vertically-restricted obstacle* O*, we can modify* P *to* P' *such that it fulfills the desired properties of Lemma 66.*

Proof. We modify P to P' as in case 2a of Lemma 66 with respect to the vertically-restricted obstacle O. Case 2b cannot happen as then P would be infeasible. Whenever we used projection points during the construction of P' in case 2a we now have to use escape points.

Let O' the length-restricted obstacle containing O. Since $BB(r,p)$ is not tightly-covered by a length-restricted obstacle, O' must not only extend past the right and upper boundary of $BB(r,p)$ but also past the lower boundary. Since in addition P leaves r to the right, $P_{[r,x]}$ is completely covered by the length-restricted obstacle. It follows that P' must also be feasible with regard to length-restricted obstacles as it has the same length as P. The feasibility with respect to direction-restricted obstacles can be shown in the very same way as in Claim 71, but using escape points instead of projection points at the boundary of O.

Induction can be applied on the remaining subpath $P'_{[q,s]}$ to obtain a path fulfilling the desired properties of Lemma 66. □

5.3 ALGORITHM

In this section we describe an algorithm that computes a 2-approximation for the reach- and direction-restricted Steiner tree problem. We use Theorem 64 to prove its approximation ratio.

5.3.1 *Construction of a Visibility Graph*

We want to construct a graph that contains feasible shortest paths between all pairs of vertices. From this graph we extract a terminal spanning tree yielding a 2-approximation. By Theorem 64 it suffices to connect pairs of base points with free bounding box by l_1-paths (where this is feasible) to obtain such a graph. To ensure feasibility of the extracted Steiner tree, all these connecting paths should be direction-aware. Moreover, we have to make sure that the extracted Steiner tree is reach-aware. This can be achieved by forbidding Steiner points on obstacles and only inserting edges into the visibility graph which are reach-aware themselves. This means that we cannot add base points to the graph that are located in the interior of length-restricted area (corners of direction-restricted obstacles). Still, they might be part of shortest paths between some terminals. The solution to this problem is to find reach-aware shortest paths over length-restricted area with separate calls to a shortest-path algorithm. The following graphs will be computed:

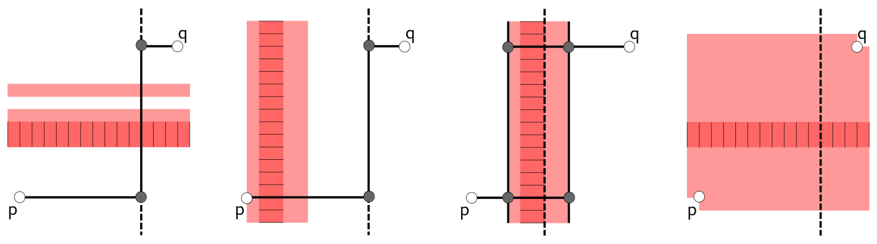
1. **Inter-Blockage graph G_{inter}:** A graph containing shortest paths between pairs of base points, crossing obstacles with at most one single edge. All computed paths are reach- and direction-aware.

2. **Intra-Blockage graphs:** For each connected component of length-restricted obstacles, computation of a graph G_{intra} containing shortest paths between pairs of base points on this component, running over this component only. These paths also respect the direction-restricted obstacles, but are not necessarily reach-aware.

3. **Visibility graph G_{vis}:** This graph contains all nodes and edges of the Inter-Blockage graph G_{inter}. Moreover, for each pair of base points on the boundary of a connected component of length-restricted obstacles a shortest path is computed on the intra-blockage graph G_{intra} of that component. If the length does not exceed L, an edge representing this path is inserted into G_{vis}. If this edge is later chosen to be in the Steiner tree, the exact path over the length-restricted area will be reconstructed.

To efficiently connect pairs of base points with free bounding box by l_1-paths we use the same idea as in [16], which is to insert so-called vertical median lines recursively, making sure that there is inserted at least one between every pair of base points. Then the base points are connected to the median lines via horizontal edges if possible.

For the computation of the inter-blockage graph G_{inter}, let p, q be two base points not in the interior of length-restricted area and with free bounding box, connectable by a feasible ℓ_1-path. If there are no



M1. Horizontally spanning obstacle

M2. Vertically spanning obstacle

M3. Vertically spanning obstacle on the median line

M4. $BB(p, q)$ is completely covered by an obstacle

Figure 46: Cases to consider when connecting to the median line (dashed), added nodes (dark grey) and edges (black).

obstacles intersecting $BB(p, q)$, p and q can simply be connected by edges to a vertical median line between them. The same holds if p, q share a common x or y-coordinate. Otherwise, $BB(p, q)^\circ \neq \emptyset$ and obstacles intersect the bounding box. Using the notation from the proof of Lemma 66, the obstacles must be horizontally or vertically spanning because no other base points may lie in the bounding box. End-points other than p, q may not even lie on the boundary $\partial BB(p, q)$ as opposed to the situation in the proof of Lemma 66, because $BB(p, q)$ is free. If an obstacle encloses one of the points as depicted in case M4 in Figure 46, the obstacle must already completely cover the bounding box due to the projection points of its corners. Thus, the following four cases for the inserted median line can occur (Figure 46):

M1. In the presence of horizontally spanning obstacles, not completely covering $BB(p, q)$, p, q can be directly connected by horizontal edges to the vertical median line.

M2. If the median runs between vertically spanning obstacles, not completely covering $BB(p, q)$, p, q can be connected in the same way.

M3. If the median runs over a vertically spanning obstacle, not completely covering $BB(p, q)$, p, q can be connected to the contour of the obstacle. Steiner points on the contour will be connected in a post-processing step.

M4. If $BB(p, q)$ is completely covered by an obstacle, p, q will not be connected in the inter-blockage graph.

Having selected one component of length-restricted obstacles, we ignore reach constraints during the computation of the intra-blockage graph on that component, because at the end we will only add paths from the intra-blockage graph to the inter-blockage graph that are not longer than the reach length L. Let p, q be two base points with free bounding box, connectable by a feasible ℓ_1-path. If no direction-restricted obstacles intersect $BB(p,q)$ or p, q share a common coordinate, they can be directly connected by edges to the vertical median line. Otherwise, case M1 to M3 in Figure 46 can occur with direction-restricted obstacles and are handled in the same way as in the inter-blockage graph. Case M4 cannot occur, since then $BB(p,q)$ would be completely covered by a direction-restricted obstacle making any ℓ_1-path from p to q infeasible.

To make the insertion of the median lines efficient, they are inserted recursively. First, a line is inserted on the median x-coordinate of all base points, then the function is called for the set of base points left of this coordinate and for the base points right of this coordinate. The median lines are inserted from left to right, so we can keep track of the obstacles on the median in a sweepline.

Algorithm 4 and Algorithm 5 implement the outlined approach. In contrast to Algorithm 4, Algorithm 5 connects points on the median line only if the resulting segment is contained in the length-restricted component that is currently considered to avoid unnecessary computations. In the code, a point is blocked by a length-restricted or by a direction-restricted obstacle, if it lies in A°, or $A_h^\circ \cup A_v^\circ$ respectively. A point p is visible from a point q if p, q are located on the same horizontal or vertical line and their direct connection is feasible with regard to the obstacles, which are subject to the input of the algorithm. We define the *visible interval* of a base point $p \in \mathcal{B}$ as the union of the maximum horizontal feasible lines starting at p to the left and to the right. This makes it possible to check if a base point can see another point in constant time. The visible intervals can be precomputed with a sweepline algorithm.

Inter-Blockage-Insert-Median is called with the set of unblocked base points and those located on the boundary of a length-restricted obstacle. For each connected component of length-restricted obstacles, *Intra-Blockage-Insert-Median* is called with all non-terminal base points of this component, making sure the running time to compute the intra-blockage graph of a connected component only depends on the complexity of that obstacle component.

Algorithm 6 shows the overall code for the computation of the visibility graph. Figure 47 illustrates the graphs computed.

Proposition 73. *For an RDRST instance (S, A, A_h, A_v, L), the graph G_{vis} computed by Algorithm 6 contains shortest feasible paths for all pairs of terminals.*

Proof. Let $p, q \in S$ two terminals and P a shortest feasible $\mathcal{B}$-simple path from p to q according to Theorem 64, i.e. the bounding box of any two successive base points $x, y \in E(P)$ is free and $P_{[x,y]}$ is an ℓ_1-path. Define $r_0 := p$ and $r_n := q$ and let $r_1, \ldots, r_{n-1}$ be the remaining base points on P, on which P transitions from unblocked to blocked area. Let P_i be the subpath from r_i to r_{i+1}. Now we show that all

Algorithm 4: Inter-Blockage-Insert-Median($\mathcal{B}', A, A_h, A_v, L$)

```
1  Let $x_m$ be the median of the $x$-coordinates of $\mathcal{B}'$
2  Inter-Blockage-Insert-Median($\{(x,y) \in \mathcal{B}' : x < x_m\}, A, A_h, A_v, L$)
3  foreach $p = (x,y) \in \mathcal{B}'$ increasing in $y$ do
4      if $(x_m, y)$ is visible from $p$ then
5          if $(x_m, y) \notin A^\circ$ then
6              Add node $(x_m, y)$ and edge $\{(x_m, y), p\}$
7              if $\exists$ previous node $(x_m, y')$ on the median visible from $(x_m, y)$ then
8                  Connect $(x_m, y)$ with $(x_m, y')$
           else
9              Let $(x_l, x_r) \ni x_m$ be max. interval s.t. $(x', y) \in A^\circ \; \forall\, x' \in (x_l, x_r)$
10             $s := \arg\max\{\|p - p'\| : p' \in \{(x_l, y), (x_r, y)\}\}$
11             $r := \arg\min\{\|p - p'\| : p' \in \{(x_l, y), (x_r, y)\}\}$
12             if $r$ and $s$ are visible from $p$ then
13                 Add $r$ and connect with $p$
14                 Add $s$ and connect with $r$
15 Inter-Blockage-Insert-Median($\{(x,y) \in \mathcal{B}' : x > x_m\}, A, A_h, A_v, L$)
```

Algorithm 5: Intra-Blockage-Insert-Median($\mathcal{B}', A, A_h, A_v, L$)

```
1  Let $x_m$ be the median of the $x$-coordinates of $\mathcal{B}'$
2  Intra-Blockage-Insert-Median($\{(x,y) \in \mathcal{B}' : x < x_m\}, A, A_h, A_v, L$)
3  foreach $p = (x,y) \in \mathcal{B}'$ increasing in $y$ do
4      if $(x_m, y) \in A$ and $(x_m, y)$ is visible from $p$ then
5          if $p \in A_h^\circ \cup A_v^\circ$ or $(x_m, y) \notin A_h^\circ \cup A_v^\circ$ then
6              Add node $(x_m, y)$ and edge $\{(x_m, y), p\}$
7              if $\exists$ previous node $(x_m, y')$ on the median visible from $(x_m, y)$ and
                 $[(x_m, y), (x_m, y')]$ is contained in $A$ then
8                  Connect $(x_m, y)$ with $(x_m, y')$
9          else if $(x_m, y) \in A_h^\circ$ then
10             Let $(x_l, x_r) \ni x_m$ be max. interval s.t. $(x', y)$ is blocked by
                 horizontally-restricted obstacles for all $x' \in (x_l, x_r)$
11             $s := \arg\max\{\|p - p'\| : p' \in \{(x_l, y), (x_r, y)\}\}$
12             $r := \arg\min\{\|p - p'\| : p' \in \{(x_l, y), (x_r, y)\}\}$
13             if $r$ and $s$ are visible from $p$ then
14                 Add $r$ and connect with $p$
15                 Add $s$ and connect with $r$
16 Intra-Blockage-Insert-Median($\{(x,y) \in \mathcal{B}' : x > x_m\}, A, A_h, A_v, L$)
```

```
Algorithm 6: Computation of a visibility graph
   Input: An RDRST instance $(S, A, A_h, A_v, L)$.
   Output: A visibility graph that contains a shortest feasible path
           between every pair of terminals.
   // Preprocessing
 1 Compute connected components of $A$
 2 $\mathcal{K} := \{$corners of $A, A_h, A_v\}$
 3 Compute projection and escape points for all $x \in \mathcal{K}$
 4 $\mathcal{B} := S \cup \mathcal{K} \cup \bigcup_{x \in \mathcal{K}}\{$projection and hor./ver. escape points for $x\}$
 5 Compute horizontal visible intervals for each $p \in \mathcal{B}$.

   // Inter-blockage graph computation
 6 $\mathcal{B}' := \{p \in \mathcal{B} \setminus A^\circ\}$.
 7 $G_{inter} := (\mathcal{B}', \emptyset)$
 8 Run Inter-Blockage-Insert-Median$(\mathcal{B}', A, A_h, A_v, L)$ on $G_{inter}$
 9 Connect points on the contour of $A$ in $G$

   // Intra-blockage graphs and visibility graph computation
10 $G_{vis} := G_{inter}$
11 foreach $Z$ connected component of $A$ do
12     $\mathcal{B}'' := (\mathcal{B} \setminus S) \cap Z$
13     $G_{intra} := (\mathcal{B}'', \emptyset)$
14     Run Intra-Blockage-Insert-Median$(\mathcal{B}'', A, A_h, A_v, L)$ on $G_{intra}$
15     Connect points on the contour of $A_h \cap Z$ in $G_{intra}$
16     foreach $p \in \mathcal{B}''$ on the boundary of $A$ do
17         Compute a shortest-path tree in $G_{intra}$ with root $p$
18         foreach $q \in \mathcal{B}''$ on the boundary of $A$ do
19             if dist$(p, q) \leqslant L$ then
20                 Add edge $\{p, q\}$ to $G_{vis}$ of weight $dist(p, q)$
21 return $G_{vis}$
```

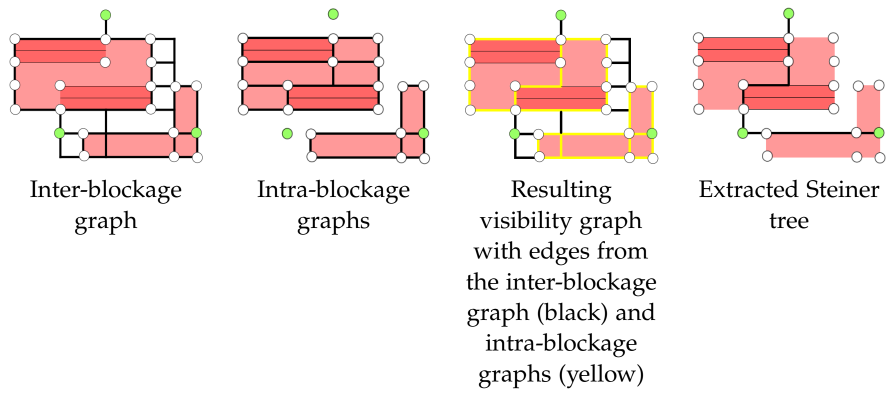

Inter-blockage graph
Intra-blockage graphs
Resulting visibility graph with edges from the inter-blockage graph (black) and intra-blockage graphs (yellow)
Extracted Steiner tree

Figure 47: Result of the algorithm with terminals (green) and other base points (white).

these subpaths are contained (in an equivalent form) in the graph G_{vis} computed by Algorithm 6.

For $i \in \{0, \dots, n-1\}$ let $s_0 = r_i, s_1, \dots, s_{m-1}, s_m = r_{i+1}$ be the base points on P_i. Consider three cases (Figure 48):

1. P_i is not completely covered by a length-restricted obstacle:

   By definition of r_i and r_{i+1} none of the $s_1, \dots, s_{m-1}$ can be a transition point of P_i from unblocked to blocked area. Furthermore, none of these points can be located in the interior of length-restricted obstacles, because otherwise P_i would be completely covered by Corollary 69. Hence none of the subpaths $P_{[s_j, s_{j+1}]}$ ($j = 0, \dots, m-1$) is completely covered. Because in addition all the $P_{[s_j, s_{j+1}]}$ are ℓ_1-paths and s_j, s_{j+1} have free bounding box, s_j and s_{j+1} ($j = 0, \dots, m-1$) are connected by the Inter-Blockage-Insert-Median function on line 8 and through the obstacle contours computed in line 9.

2. P_i is completely covered by a length-restricted obstacle and passes through a base point in the interior of that obstacle:

   We can assume r_i and r_{i+1} to be non-terminal base points by Corollary 69. For $j = 0, \dots, m-1$ the points s_j, s_{j+1} can be connected by a feasible ℓ_1-path and their bounding box is free. Hence, the graph G_{intra} computed in lines 14 and 15 of the algorithm contains ℓ_1-paths between successive base points $s_0, \dots, s_m$. Thus the shortest-path tree with root r_i computed in the loop in line 17 contains a path from r_i to r_{i+1} of length at most L. Hence an edge from r_i to r_{i+1} is added to G_{vis}.

3. P_i is completely covered by a length-restricted obstacle but does not pass any base point in the interior of that obstacle:

   The bounding box of r_i and r_{i+1} must be free. If the points are on the same horizontal or vertical line they are connected by the Inter-Blockage-Insert-Median function on line 8. Otherwise r_i and r_{i+1} must be located on corner points which, as non-terminal base points on the boundary of A, are going to be connected through the intra-blockage graph of that obstacle component (like in the previous case).

□

5.3.2 *Extraction of the Steiner Tree*

The extraction of the Steiner tree is based on the same approach as in [25]. Given a visibility graph $G_{vis} = (V, E)$ for the terminal set S, we are now interested in extracting a Steiner tree. Since G_{vis} is shortest-path preserving, any minimum terminal spanning tree routine will provide an approximation guarantee of two. In our implementation,

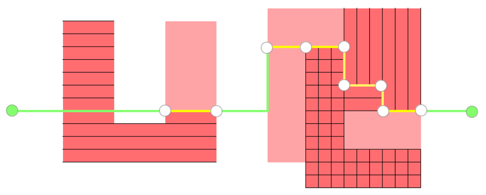


Figure 48: Completely covered subpaths (yellow), remaining subpaths (green).

Steiner trees are found in $\mathcal{O}(|E| \log |E|)$ time using the Dijkstra-Kruskal approach from [40]. We also applied Mehlhorn's algorithm [44], with a faster running time of $\mathcal{O}(|E| + |V| \log |V|)$, but found it slower in practice. This Steiner tree is a 2-approximation of an optimal RDRST, because it is at most as long as a reach- and direction-aware minimum spanning tree. In fact, since Steiner points on obstacles are forbidden, it is a 2-approximation algorithm for a less restrictive version of the problem wherein the longest path across blocked area has length $\leqslant L$, since both problems coincide for minimum spanning trees and our solution is feasible for both.

Extending [6, 25], we improve the computation of the Steiner tree as follows. The minimum terminal spanning tree routine from [40] starts with a forest consisting of the terminals. It uses two heaps: one for propagating the distances from the terminals and one for connecting terminals of different components by shortest paths. This is sufficient for a minimum terminal spanning tree, but a shorter length might be achievable by adding shortest paths between tree components and not only between terminals. This can be done only if Steiner nodes that were added to the tree are relabeled and propagated with distance zero. Doing this can result in a runtime of $\Theta(|V||E| \log |V|)$, since $\Theta(|V|)$ times components are connected, each resulting in an effort of $\Theta(|E| \log |V|)$ for repropagating distances in the worst case. However, we found that with this approach in practice the runtime only slightly increases while the extracted Steiner tree gets significantly shorter (see Table 13).

5.3.3 *Running Time*

Let $k := |\mathcal{B}|$ denote the number of base points. If we assume the interior of all rectangles in $\mathcal{R} \cup \mathcal{R}_h \cup \mathcal{R}_v$ are pairwise disjunct (apart from fully blocked obstacles that are modeled by two identical rectangles in $\mathcal{R}_h$ and $\mathcal{R}_v$), then k is linear in the input size $|\mathcal{R}| + |\mathcal{R}_h| + |\mathcal{R}_v| + |S|$.

Pre-processing can be done with sweeplines in time $\mathcal{O}(k \cdot \log k)$. During the computation of the inter-blockage graph each base point is in the input of at most $\mathcal{O}(\log k)$ calls of the Inter-Blockage-Insert-Median function. Hence, the loop in the Inter-Blockage-Insert-Median function is executed $\mathcal{O}(k \cdot \log k)$ times, each of which needs time

$\mathcal{O}(\log k)$ through the use of the sweeplines. In total, the running time is $\mathcal{O}\left(k \cdot (\log k)^2\right)$. In each loop pass a constant number of edges and nodes is added to G_{inter}, thus G_{inter} has a size of $\mathcal{O}(k \cdot \log k)$.

Let Z be a connected component of A (that is, Z is a length-restricted obstacle) and $|Z|$ the number of non-terminal base points on its boundary and its interior. Analogously to the computation of the inter-blockage graph the computation of the intra-blockage graph for Z needs time $\mathcal{O}\left(|Z| \cdot (\log |Z|)^2\right)$, since only obstacles and base points of the current component have to be considered. The constructed graph G_{intra} has $\mathcal{O}(|Z| \cdot \log |Z|)$ nodes and edges, which is why a call of Dijkstra consumes time

$$\mathcal{O}\left(|Z| \cdot \log |Z| \cdot \log(|Z| \cdot \log |Z|)\right) = \mathcal{O}\left(|Z| \cdot (\log |Z|)^2\right)$$

All $\mathcal{O}(|Z|)$ calls of Dijkstra need $\mathcal{O}\left(|Z|^2 \cdot (\log |Z|)^2\right)$ time in total. At most $\mathcal{O}\left(|Z|^2\right)$ edges, and no additional nodes are added to the graph G_{vis}.

Denote by l the maximum size of a connected component, then the computation of all intra-blockage graphs takes time

$$\mathcal{O}\left(\sum_{\substack{Z \text{ connected} \\ \text{component}}} |Z| \cdot l \cdot (\log l)^2\right) = \mathcal{O}\left(kl \cdot (\log l)^2\right)$$

and $\mathcal{O}(kl)$ edges are added to G_{vis}. Moreover, G_{vis} contains $\mathcal{O}(k \cdot \log k)$ nodes and edges from G_{inter}, amounting to a total number of $\mathcal{O}(k \cdot \log k)$ nodes and $\mathcal{O}(k \cdot \log k + kl)$ edges.

The total running time can be estimated by

$$\mathcal{O}\left(k \cdot (\log k)^2 + kl \cdot (\log l)^2\right) = \mathcal{O}\left(kl \cdot (\log k)^2\right).$$

The time of the final extraction of the Steiner tree, without repropagating labels from added Steiner points as described in Section 5.3.2, also amounts to $\mathcal{O}\left(kl \cdot (\log k)^2\right)$. If k is the only upper bound on l, the final runtime is $\mathcal{O}\left(k^2 \cdot (\log k)^2\right)$. Otherwise, if l is assumed to be constant the final runtime is $\mathcal{O}\left(k \cdot (\log k)^2\right)$, which is the same as in the obstacle-avoiding case [16].

5.3.4 *Post-Processing*

The length of the extracted Steiner can be reduced significantly through heuristic post-processing (Table 13). We apply an edge substitution method followed by geometric local recomputations of the Steiner tree.

5.3.4.1 *Edge Substitution*

Given the visibility graph G_{vis} and the extracted Steiner tree T, divide T at nodes of degree at least three and at the terminals into *path*

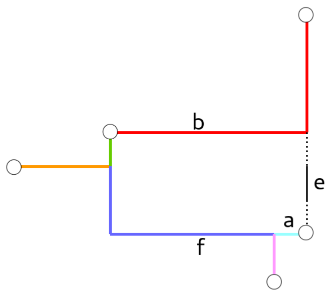


Figure 49: Adding edge e and the shortest paths from its end points to the tree allows the deletion of the longest path segment f on the path from path segment a to b.

segments (in Figure 49 each path segment of T is colored differently). Consider some edge $e \in E(G_{\text{vis}})$ that is not part of T and for which the nearest path segments a, b to each of its end points are not identical. Removing the longest path segment f on the tree path from a to b and adding the edge e and the shortest paths from e to a and e to b, T remains a tree (see Figure 49). The weight decreases by

$$\text{gain}_{(e,a,b,f)} := \text{length}(f) - \text{length}(e) - \text{dist}(e,a) - \text{dist}(e,b)$$

(e, a, b, f) is called a substitution candidate. We look for all substitution candidates with positive gain and sort them in descending order by gain. Then we employ all substitution candidates (e, a, b, f) one by one if the path segments a, b and f have not been removed during an earlier substitution. If a, b and f are still part of the tree, it can be shown that f must still be on the unique path from a to b, assuming a fixed order of path segments of equal length. Therefore, applying the substitution candidate (e, a, b, f) leaves the tree connected and achieves the gain that was computed before sequentially applying the substitution candidates. The whole procedure can be iterated, though in our experiments more than two iterations were not worth the additional runtime. This heuristic was described in [20] (p. 57) as a generalization of the edge substitution heuristic in [43]. Two iterations of this heuristic substantially improve the length of the Steiner tree as can be seen in Table 13.

Checking whether a path segment has been removed during an earlier substitution can be done in constant time. Finding the longest edge on the tree path between two nodes can be described as the following problem:

Problem 74. Online Maximum Cost Edge on Tree Path

Input: A weighted Steiner tree T.

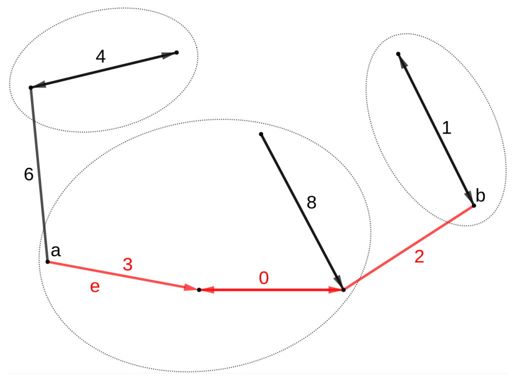


Figure 50: Edge-weighted tree as sample instance for the maximum cost on tree path problem. The edge with weight three is the most expensive edge on the path from a to b, and it is contained in K_a (ellipses denote components of directed edges).

Output: An oracle that, for any pair of nodes $a, b \in V(T)$, returns the most expensive edge on the unique path from a to b in T.

Using a pre-processing time of $\mathcal{O}(|V(T)| \cdot \log |V(T)|)$, [31] shows how to obtain such an oracle that answers queries in time $\mathcal{O}(\log |V(T)|)$. Together with an initial shortest path search in $V(G_{vis})$ starting at all nodes of T to find the nearest path segments for each edge $e \in V(G_{vis})$ we obtain a total running time of $\mathcal{O}(|E(G_{vis})| \cdot \log |V(G_{vis})|)$ for one edge substitution iteration.

As a side node, the pre-processing time in the result from [31] for the maximum cost edge on tree path problem can be improved as follows:

Theorem 75. *Let T a weighted Steiner tree. After a pre-processing time of $\mathcal{O}(|V(T)|)$, for any two nodes $a, b \in V(T)$ the most expensive edge on the path from a to b in T can be found in time $\mathcal{O}(\log |V(T)|)$.*

Proof. Unlike [31] we do not initially sort the edges of T by weight. Instead, we perform the algorithm that is described in [31] for intuition only:

For each node $u \in V(T)$ we direct its cheapest incident edge e away from u and set $\text{edge}(u) := e$. This way, some edges remain undirected, some become unidirected and some bidirected (Figure 50). Now consider the subgraph of T that contains all uni- and bidirected edges. Each connected component K in this subgraph consists of one bidirected edge e and two, possibly empty, anti-arborescences attached to its endpoints. Let T' the tree that arises from T by collapsing each such connected component K to a single node q and set $\text{parent}(u) := q$ for each node $u \in K$. Because each connected component contains one bidirectional edge, it is $|V(T')| \leqslant |V(T)|/2$.

This procedure is repeated iteratively. One iteration requires linear running time, and because after each iteration the tree size halves at least, the total running time amounts to $\mathcal{O}(|V(T)|)$.

Queries are performed the same way as in [31]: Let $a, b \in V(T)$. We return the more expensive edge of $\text{edge}(a)$, $\text{edge}(b)$, and the most expensive edge on the path from $\text{parent}(a)$ to $\text{parent}(b)$ in the contracted tree T'. The latter can be found recursively with the same procedure and since each of the contracted trees has at most half the size of its predecessor, the depth of this recursion is $\mathcal{O}(\log |V(T)|)$.

To prove correctness, let K_a, K_b be the connected components in the subgraph of T consisting of unidirectional and bidirectional edges with $a \in K_a$ and $b \in K_b$. Let $e = (v, w)$ the most expensive edge on the path from a to b.

First assume that $e \in K_a \cup K_b$. That means that e is directed and it is the cheapest incident edge for one of its endpoints. Because e is the most expensive edge on the path from a to b, e must be incident to a or b and it is the cheapest incident edge of that node. Without loss of generality assume that e is incident to a (Figure 50). The incident edge to b that is on the path from b to a is either e or a cheaper edge. Therefore the more expensive edge of $\text{edge}(a)$ and $\text{edge}(b)$ equals the most expensive edge on the path from a to b.

Now consider the case that $e \notin K_a \cup K_b$. Thus, e is located on the path from $\text{parent}(a)$ to $\text{parent}(b)$ in the contracted tree T' on which we find it recursively. a and b must both be incident to cheaper edges which is why $\text{edge}(a)$ and $\text{edge}(b)$ are both cheaper than e, proving correctness also in this case.

$\square$

5.3.4.2 *Geometric Optimizations*

In this section we describe the way in which we recompute local unblocked components of the Steiner tree, taken from [25]. Since the visibility graph only contains certain shortest paths, we found that a combination of simple local search heuristics can significantly improve the result for most instances. Especially in practical instances, we often encountered large clusters of terminals in an unblocked area. For those, any (obstacle-unaware) Steiner tree algorithm could be used instead. Therefore, during post-processing, we collect maximal components of the constructed tree with unblocked bounding box and reconnect them using the exact FLUTE algorithm [15] for up to 9 terminals and a Prim heuristic in the Delaunay triangulation for larger terminal sets.

Furthermore, there are some nonoptimal local configurations, such as trunks with more branches on one side and L-shapes that can be mirrored to decrease the length. They can be found and processed efficiently for the entire tree by changing the edge structure locally if the resulting tree is feasible.

This procedure can be iterated for better results; in our experience, a good trade-off of running time and solution quality is achieved by one iteration for chip instances and two iterations for benchmark instances (Table 13).

	plain		+ repropagation		+ 2x edge substitution		+ 2x postopt	
Instance	length	time	length	time	length	time	length	time
IND1	665	0,00	642	0,00	632	0,00	632	0,00
IND2	10700	0,00	10700	0,00	10600	0,00	10500	0,00
IND3	696	0,00	688	0,00	680	0,00	670	0,00
IND4	1196	0,00	1163	0,00	1163	0,00	1145	0,00
IND5	inf.	0,00	inf.	0,00	inf.	0,00	inf.	0,00
RC01	29040	0,00	28230	0,00	27970	0,00	26660	0,00
RC02	44810	0,00	44000	0,00	42880	0,00	42270	0,00
RC03	58510	0,00	57250	0,00	56060	0,00	54470	0,00
RC04	62790	0,00	62300	0,00	61000	0,00	60100	0,00
RC05	81660	0,00	79580	0,00	78050	0,00	75300	0,01
RC06	89072	0,03	84835	0,03	82565	0,06	80719	0,05
RC07	119002	0,02	116043	0,03	114272	0,05	111419	0,05
RC08	126517	0,04	120691	0,05	118387	0,08	115358	0,08
RC09	121381	0,05	118242	0,05	116081	0,11	113118	0,11
RC10	181820	0,02	178160	0,01	173600	0,03	167840	0,03
RC11	254111	0,02	250244	0,02	243734	0,06	234542	0,07
RC12	834103	0,76	802694	0,97	786445	1,76	765277	1,85
RL01	520700	0,43	507575	0,54	497957	1,22	482599	1,13
RL02	691479	0,21	682818	0,26	665927	0,86	636527	0,89
RL03	694918	0,28	687170	0,25	669350	0,82	638864	0,94
RL04	756458	0,30	748095	0,25	728209	0,91	694676	1,02
RL05	729978	0,04	729978	0,04	729978	0,04	723191	0,12
RT01	2342	0,04	2272	0,03	2236	0,04	2211	0,04
RT02	51510	0,04	49269	0,04	48354	0,04	47142	0,05
RT03	8973	0,04	8620	0,03	8405	0,04	8190	0,05
RT04	10944	0,06	10470	0,06	10221	0,09	9992	0,09
RT05	57595	0,14	55109	0,16	54046	0,24	52957	0,25
Total	5540970	2,54	5436838	2,84	5328802	6,48	5156369	6,86
	100,00%	100,00%	98,12%	111,70%	96,17%	254,48%	93,06%	269,55%

Table 13: Increase of running time and decrease of length by the repropagation technique, the edge substitution heuristic and the geometric postopt routines, performed consecutively. See Section 5.4 for more details on the test setup.

5.3.5 *Multiple Nets*

This section also originates from [25]. As already seen, there can be millions of nets on one chip. Many nets consist of only two or three terminals and it would be too time-consuming to compute the visibility graph from scratch for each net. Because the set of obstacles usually is persistent, we proceed as follows when processing all nets of a chip: Some pre-processing steps are independent of the terminal set and can be pre-computed for all nets. Using this information, we construct an obstacle visibility graph, i.e. a visibility graph for an empty set of terminals. This obstacle graph can be extended to a visibility graph for a given set of terminals by inserting a new median line through every terminal and connecting all unblocked obstacle base points and other terminals to this line (if possible). This preserves the visibility graph invariant of having a median line between each pair of base points with free bounding box containing an ℓ_1-path between them. Note that we do not have to connect terminals to base points on length-restricted obstacles, because by Corollary 69 for any pair of terminals we can find a shortest path which, if it crosses a blocked base point, enters that obstacle on another base point. In particular the terminals themselves must not lie on obstacles by our definition of the problem.

This construction is most useful if a net has few terminals and L is small. In our experience, the number of terminals for which this construction is applied should be less than logarithmic in the number of obstacles and linear (with a very small slope) in L. For sufficiently large L, constructing an l_1-Steiner tree built by an obstacle-unaware heuristic initially and checking feasibility leads to better results and running times of our algorithm.

5.4 EXPERIMENTAL RESULTS

We performed experiments on standard benchmarks from the literature for the obstacle-avoiding Steiner tree problem, along with some very big practical chip instances. Those tests were carried out on an Intel® Xeon® E5-2667 v2 CPU @ 3.3GHz, 384 GB RAM with 16 cores. Moreover we applied our algorithm in BonnRouteGlobal to compute lower bounds on the wire length and via number of nets. Those experiments were carried out on an AMD®EPYC®7601 CPU @ 2.7GHz, 512 GB RAM with 32 cores.

5.4.1 *Standard Benchmarks*

The standard benchmarks include the following instances: IND1-IND5 are industrial test cases by Synopsys, RC01-RC12 are randomly generated instances by Feng et al. [19], RL01-RL05 are large random instances by Long et al. [43] and RT01-RT05 have a fixed ratio of terminals to obstacles of 5, 10 and 50 and were introduced by Lin et al. [37].

The instances are given as sets of rectangles whose union represents the blocked area. In most of them, no obstacle has both width and height exceeding 10% of the size of the bounding box of the whole instance. To obtain instances with direction-restrictions we once turn all the obstacles into into horizontally-, and once into vertically-restricted obstacles.

			Opt [29]		[39]		[3]		[41]		[14] (seq.)		[25]		Ours (L = 0)	
Instance	|S|	|O|	length	time	length	time	length	time	length	time	length	time	length	time	length	time
IND1	10	32	604	0,11	626	1	604	0	604	0	619	0,01	629	0,01	632	0,00
IND2	10	43	9500	0,25	9700	1	9500	0	9600	0	9500	0,03	10600	0,00	10500	0,00
IND3	10	50	600	0,19	600	1	600	0	600	0	600	0,01	678	0,00	670	0,00
IND4	25	79	1086	0,87	1095	2	1129	0	1092	0	1096	0,01	1160	0,01	1145	0,00
IND5	33	71	1341	1,03	1364	2	1364	0	1374	0	1360	0,01	inf.	0,00	inf.	0,00
RC01	10	10	25980	0,16	26740	1	25980	0	26040	0	25980	0,04	27360	0,00	26660	0,00
RC02	30	10	41350	0,52	42070	1	42110	0	41570	0	42010	0,09	42830	0,00	42270	0,00
RC03	50	10	54160	0,68	54550	1	56030	0	54620	0	54390	0,08	55160	0,01	54470	0,00
RC04	70	10	59070	0,95	59390	1	59720	0	59860	0	59740	0,09	60010	0,00	60100	0,00
RC05	100	10	74070	1,31	75430	1	75000	0	74770	0	74650	0,07	74930	0,01	75300	0,01
RC06	100	500	79714	335	81903	17	81229	0,03	81854	0,02	81607	0,38	85077	0,03	80719	0,05
RC07	200	500	108740	541	111752	26	110764	0,03	111211	0,03	111542	0,75	114211	0,03	111419	0,05
RC08	200	800	112564	24170	118349	43	116047	0,05	116132	0,04	115931	1,35	120554	0,06	115358	0,08
RC09	200	1000	111005	14174	114928	49	115593	0,06	113559	0,05	113460	1,75	118041	0,06	113118	0,11
RC10	500	100	164150	176	167540	16	168280	0,02	167460	0,01	167620	0,3	168720	0,03	167840	0,03
RC11	1000	100	230837	706	234097	21	234416	0,03	236018	0,02	235283	0,94	237088	0,05	234542	0,07
RC12	1000	10000	n.a.	n.a.	780528	681	756998	1,19	762435	1,2	761606	147,14	795269	0,9	765277	1,85
RL01	5000	5000	n.a.	n.a.	n.a.	n.a.	483027	1,15	n.a.	n.a.	481813	27,39	494238	0,73	482599	1,13
RL02	10000	500	n.a.	n.a.	n.a.	n.a.	637753	1,18	n.a.	n.a.	638439	23,3	641150	0,6	636527	0,89
RL03	10000	100	n.a.	n.a.	n.a.	n.a.	640902	1,13	n.a.	n.a.	642380	21,47	643309	0,59	638864	0,94
RL04	10000	10	n.a.	n.a.	n.a.	n.a.	697125	1,57	n.a.	n.a.	699502	27,72	697993	0,58	694676	1,02
RL05	10000	0	n.a.	n.a.	n.a.	n.a.	728438	0,12	n.a.	n.a.	730857	31,08	729978	0,07	723191	0,12
RT01	10	500	2146	25	2259	17	2191	0	2193	0,01	2231	0,19	2283	0,02	2211	0,04
RT02	50	500	45852	31	48684	18	48156	0,02	47488	0,02	47297	0,64	49821	0,03	47142	0,05
RT03	100	500	7964	840	8347	20	8282	0,03	8231	0,02	8187	0,23	8387	0,03	8190	0,05
RT04	100	1000	9693	34521	10221	40	10330	0,06	9893	0,04	9914	0,43	10609	0,06	9992	0,09
RT05	200	2000	51313	276621	53745	78	54598	0,15	52509	0,12	52473	3,38	55760	0,13	52957	0,25

Table 14: Comparison of wire lengths and running times on standard benchmark instances with results from [29], [39], [3], [41], [14] and [25]. We use the same more restrictive interpretation of obstacles as [25] which is why [25] and our algorithm considers IND5 infeasible.

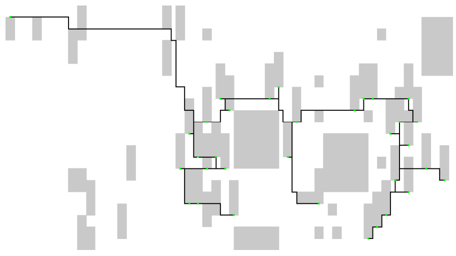


Figure 51: Our solution for IND5 with L = 5% instance width. In the upper right corner there is a vertex surrounded by a ring of obstacles making this instance infeasible for small L.

Table 14 compares our lengths and running times with other publications for the case $L = 0$. At the cost of slightly higher running time, we improve significantly on many instances compared to [25], on which this work is based on. In comparison with [3] and [14], which are the only other publications that also solve the RL instances, we solve 14 respectively 11 instances better. Our running times are similar to those of [3] and much faster than [14]. Optimum lengths are reported from [29]. They have a slightly different definition of obstacle-avoiding. Edges between two rectangles that share a boundary may be used, whereas with our definition they would pass through the interior of the blocked area and thus not be obstacle-avoiding. This is why, for any $L < \infty$, their Steiner trees can be strictly shorter than the optimum according to our definition. Furthermore, our algorithm solves a more general problem.

There are some larger instances for which the optimum solution is not known. Our algorithm is able to improve on the best-known solution from literature for $L = 0$, even with our stricter definition of being obstacle-aware. Our definition renders IND5 infeasible for small L ($\leqslant 1\%$ of the bounding box), as can be seen in Figure 51 where there is a vertex surrounded by obstacles.

Table 15 shows the lengths of RDRSTs found by our algorithm for different values of L, relative to the length of the longer side of the bounding box of the instance including obstacle corners. One can see that, usually, the tree length gradually decreases with increasing value of L. As expected, if all obstacles are turned horizontally- or vertically-restricted, the lengths become generally longer due to the more restricted instances.

The reported running times are generally fast and typically increase with growing L because more and more edges reaching over obstacles are added. For $L = \infty$ and without direction restrictions on the obstacles the running times are fastest, since in that case the obstacles can be completely ignored.

	Reach length					Running time in seconds				
Instance	$L = 0$	1%	5%	10%	∞	$L = 0$	1%	5%	10%	∞
IND1	632	632	609	609	609	0.00194	0.00178	0.00091	0.00102	0.00009
IND1 (hor)	614	614	609	609	609	0.00248	0.00229	0.00153	0.00181	0.00231
IND1 (ver)	614	614	618	618	623	0.00226	0.00228	0.00308	0.00328	0.00363
IND2	10500	10500	9100	9100	9100	0.00215	0.00218	0.00149	0.00199	0.00013
IND2 (hor)	10600	10600	9600	9700	9700	0.00339	0.00322	0.00414	0.00546	0.00621
IND2 (ver)	10600	10600	9600	9600	9600	0.00325	0.00324	0.00450	0.00612	0.00700
IND3	670	670	600	587	587	0.00215	0.00217	0.00230	0.00160	0.00010
IND3 (hor)	670	670	600	600	600	0.00345	0.00335	0.00388	0.00463	0.00642
IND3 (ver)	670	670	626	621	625	0.00330	0.00333	0.00398	0.00456	0.00726
IND4	1145	1145	1090	1090	1092	0.00347	0.00328	0.00468	0.00586	0.00023
IND4 (hor)	1145	1145	1101	1095	1092	0.00517	0.00505	0.00785	0.00966	0.01353
IND4 (ver)	1145	1145	1105	1110	1110	0.00505	0.00527	0.00731	0.01013	0.01376
IND5	inf.	inf.	1331	1323	1312	0.00387	0.00375	0.00461	0.00542	0.00029
IND5 (hor)	inf.	inf.	1335	1330	1330	0.00573	0.00583	0.00703	0.00916	0.01039
IND5 (ver)	inf.	inf.	1368	1365	1365	0.00544	0.00549	0.00676	0.00900	0.01048
RC01	26660	26660	25290	25290	25290	0.00133	0.00118	0.00047	0.00055	0.00011
RC01 (hor)	26040	26040	25550	25550	25550	0.00145	0.00136	0.00141	0.00153	0.00174
RC01 (ver)	26040	26040	25780	25780	25780	0.00138	0.00136	0.00167	0.00154	0.00190
RC02	42270	42270	41620	40430	41330	0.00202	0.00210	0.00211	0.00232	0.00032
RC02 (hor)	42280	42280	42020	40750	40720	0.00228	0.00231	0.00229	0.00236	0.00247
RC02 (ver)	42280	42280	42280	42100	42100	0.00225	0.00215	0.00233	0.00232	0.00262
RC03	54470	54470	53050	53360	52470	0.00298	0.00300	0.00299	0.00309	0.00065
RC03 (hor)	54360	54360	54360	54270	54250	0.00342	0.00330	0.00346	0.00346	0.00349
RC03 (ver)	54360	54360	52740	52740	53060	0.00342	0.00338	0.00351	0.00363	0.00372
RC04	60100	60100	56960	55830	55330	0.00373	0.00366	0.00364	0.00370	0.00091
RC04 (hor)	60150	60150	59660	58950	58950	0.00440	0.00422	0.00425	0.00428	0.00428
RC04 (ver)	60150	60150	57590	57590	57590	0.00425	0.00434	0.00429	0.00448	0.00486
RC05	75300	75300	73050	72860	71610	0.00511	0.00528	0.00565	0.00556	0.00125
RC05 (hor)	75070	75070	73600	73350	73350	0.00579	0.00560	0.00558	0.00580	0.00600
RC05 (ver)	75070	75070	74800	74800	74300	0.00549	0.00540	0.00553	0.00546	0.00600
RC06	80719	80920	79118	78720	77472	0.04733	0.04580	0.06157	0.06452	0.00089
RC06 (hor)	80726	80588	79379	79379	79379	0.06279	0.06435	0.07302	0.07238	0.06919
RC06 (ver)	80726	80220	79735	79815	79815	0.05828	0.06328	0.07650	0.07911	0.07810
RC07	111419	109151	107746	107675	107190	0.05022	0.05346	0.07137	0.09688	0.00212
RC07 (hor)	112072	109428	108374	108264	108264	0.11754	0.11226	0.08756	0.12194	0.12852
RC07 (ver)	112072	110880	109583	109583	109583	0.11049	0.11572	0.14500	0.14498	0.14180
RC08	115358	112271	110212	110038	109589	0.10493	0.13243	0.17966	0.18242	0.00221
RC08 (hor)	115577	114528	113292	113560	113560	0.16977	0.17116	0.19385	0.18155	0.17006
RC08 (ver)	115577	113361	112281	112266	112266	0.15269	0.18101	0.20803	0.18239	0.14994
RC09	113118	111616	108688	108663	107561	0.10055	0.13664	0.20710	0.17206	0.00193
RC09 (hor)	114126	113146	110638	111217	111217	0.16382	0.20378	0.18215	0.17594	0.16056
RC09 (ver)	114126	112813	111260	111559	111559	0.13636	0.18721	0.23878	0.23708	0.23803
RC10	167840	167840	164430	165260	164570	0.03167	0.03289	0.03520	0.03977	0.00548
RC10 (hor)	167990	167990	166020	165990	165990	0.03753	0.03714	0.03735	0.03936	0.03997
RC10 (ver)	167990	167990	166820	166670	166670	0.03721	0.03670	0.03980	0.04205	0.04259
RC11	234542	234267	234144	234144	230651	0.06828	0.06996	0.07590	0.07333	0.01203
RC11 (hor)	234716	235062	234523	234523	234523	0.07343	0.07656	0.07824	0.08579	0.08407
RC11 (ver)	234716	234733	234498	234498	234498	0.07477	0.07821	0.07860	0.09495	0.07716
RC12	765277	762584	762584	762584	754414	2.14789	2.90277	2.75320	2.81839	0.01107
RC12 (hor)	768418	762736	762736	762736	762736	3.36088	3.92927	3.61235	3.63293	3.46926
RC12 (ver)	768418	763800	763800	763800	763800	2.55562	2.98395	3.12483	2.97985	2.98284
RL01	482599	476846	476719	476795	472780	1.49863	2.12899	2.29964	2.51280	0.06394
RL01 (hor)	482970	479907	479471	479832	479675	1.92268	1.90503	2.10141	2.11231	2.08375
RL01 (ver)	482970	479484	479719	480134	480004	1.95358	2.37386	2.58948	2.86925	2.81634
RL02	636527	636506	636728	636689	634123	1.27647	1.39860	1.52954	1.62878	0.19333
RL02 (hor)	636606	636685	636656	636638	636714	1.39796	1.42677	1.49955	1.41780	1.49021
RL02 (ver)	636606	636439	636305	636434	636509	1.37749	1.48312	1.69129	1.70332	1.74625
RL03	638864	638466	638264	638420	636541	1.34251	1.27189	1.36364	1.32884	0.17625
RL03 (hor)	638966	638739	638668	638850	638926	1.36539	1.36683	1.12059	1.27097	1.38710
RL03 (ver)	638966	638625	638644	638476	638659	1.39409	1.36182	1.37702	1.14170	1.19863
RL04	694676	694676	691661	691661	691670	1.32500	1.24035	0.16418	0.16173	0.14405
RL04 (hor)	694620	694620	691661	691661	691661	1.29650	1.39276	0.22096	0.19360	0.22246
RL04 (ver)	694620	694620	694501	694506	694403	1.20327	1.23502	1.29069	1.20884	1.43502
RL05	723191	723191	723191	723191	723191	0.16037	0.13078	0.14535	0.23667	0.13193
RL05 (hor)	723191	723191	723191	723191	723191	0.15051	0.17756	0.15985	0.18324	0.15058
RL05 (ver)	723191	723191	723191	723191	723191	0.14171	0.13356	0.13538	0.14091	0.13790
RT01	2211	1857	1817	1817	1817	0.04765	0.04223	0.01703	0.01739	0.00012
RT01 (hor)	2214	1996	1962	1938	1938	0.05312	0.05774	0.06975	0.06098	0.06787
RT01 (ver)	2214	2097	2095	2094	2095	0.05088	0.05720	0.06808	0.06719	0.08034
RT02	47142	45154	45747	45747	45747	0.04996	0.05458	0.01561	0.01676	0.00051
RT02 (hor)	47252	45992	46148	46460	46460	0.06127	0.07020	0.06893	0.07824	0.06907
RT02 (ver)	47252	46510	46256	46063	46063	0.05971	0.07014	0.07112	0.08584	0.10514
RT03	8190	7749	7690	7728	7697	0.04849	0.06041	0.07281	0.07194	0.00110
RT03 (hor)	8162	8005	8002	7992	7992	0.08168	0.09123	0.08539	0.09323	0.08020
RT03 (ver)	8162	7957	7961	7993	7993	0.06712	0.07297	0.08278	0.08440	0.10331
RT04	9992	7880	7788	7788	7788	0.09371	0.13650	0.04900	0.05242	0.00115
RT04 (hor)	10090	8218	8207	8223	8271	0.18067	0.15777	0.16638	0.17339	0.17642
RT04 (ver)	10090	9410	9363	9296	9430	0.12968	0.13517	0.18324	0.18863	0.20825
RT05	52957	44252	44191	43747	43152	0.26029	0.31407	0.36095	0.38126	0.00241
RT05 (hor)	52885	48583	47986	48224	48051	0.32767	0.39366	0.46930	0.49662	0.50661
RT05 (ver)	52885	47250	47111	47074	47259	0.37094	0.38373	0.45282	0.45101	0.52196

Table 15: Results for the standard benchmark instances with different reach lengths. (hor) denotes runs in which all obstacles have been turned horizontally-restricted, for (ver) all obstacles have been turned vertically-restricted.

				Reach length			Running time in seconds		
Instance	$\|S\|$	$\|O\|$	$L*$	$L = 0$	$L = L*$	∞	$L = 0$	$L = L*$	∞
BIG1	109	101	90	31326	31101	28497	0.00828	0.00789	0.00056
BIG1 (hor)				31317	31218	29004	0.01987	0.01637	0.01529
BIG1 (ver)				31317	31152	31134	0.01282	0.01349	0.01447
BIG2	23292	54	2400000	363287646	362221731	361716147	1.13470	1.17664	0.18032
BIG2 (hor)				363216228	363215808	363066843	1.23131	1.24668	1.25541
BIG2 (ver)				363216228	362166693	362254641	1.29920	1.28907	1.35333
BIG3	35574	158	1500000	743030175	743103261	735048651	1.69111	1.75364	0.23441
BIG3 (hor)				742991343	743028063	736149153	1.77393	1.73988	1.94106
BIG3 (ver)				742991343	743106183	743038551	1.76380	1.76184	2.10307
BIG4	46269	127	1500000	1069891260	1069129080	1068441780	2.95649	3.06826	0.42039
BIG4 (hor)				1070206260	1070189820	1070804460	3.13122	3.11375	3.15266
BIG4 (ver)				1070206260	1069568280	1071304440	3.10641	3.02021	3.27745
BIG5	108500	141	4200000	1972388850	1962285570	1957072785	7.58347	8.43931	1.06194
BIG5 (hor)				1971575745	1971720045	1971823395	8.13504	8.30368	8.03928
BIG5 (ver)				1971575745	1962581505	1960243890	8.21008	8.77973	8.51052
BIG6	129399	210	1500000	inf.	2604131070	2603190780	8.35788	10.01026	1.52484
BIG6 (hor)				inf.	inf.	2649117810	9.70341	9.64304	10.23198
BIG6 (ver)				inf.	2605043970	2603567850	9.56935	9.07445	10.53251
BIG7	639639	382	4200000	1.1645e+10	1.1612e+10	1.1603e+10	49.63901	55.10863	7.30162
BIG7 (hor)				1.1643e+10	1.1642e+10	1.164e+10	51.76650	61.53187	56.50063
BIG7 (ver)				1.1643e+10	1.1614e+10	1.1614e+10	59.36982	56.91737	55.32617
BIG8	783352	175	1200000	1.4827e+10	1.4824e+10	1.4816e+10	68.58012	74.95037	9.72992
BIG8 (hor)				1.4827e+10	1.4827e+10	1.4827e+10	75.03546	74.51239	72.38267
BIG8 (ver)				1.4827e+10	1.4824e+10	1.4824e+10	74.42497	78.52251	76.44396

Table 16: Results for the BIG instances arising from chip instances provided by IBM.

5.4.2 *Chip Instances*

We created eight instances from chips provided by IBM. They have between 109 and 783352 terminals. The bigger ones represent reset trees with low performance requirements, where short length is a major focus. These instances are published as the *BONN* instances as part of the 11th DIMACS benchmark suite on Steiner trees:

http://dimacs11.zib.de/downloads.html

Table 16 shows the resulting lengths for these instances of an obstacle-avoiding tree ($L = 0$), for the given reach length $L*$ that depends on the technology and the metal stack of the underlying chip, and for ignoring all obstacles ($L = \infty$). Moreover there are runs with all obstacles being treated as horizontally-restricted or vertically-restricted. As for the standard benchmark instances, one can observe that the lengths decrease with growing L, even though major parts of the length are incurred by unblocked clusters of terminals. As expected, the additional direction-restrictions make the trees longer. The running times demonstrate the ability of our algorithm to handle even the largest instances efficiently.

Figure 52 shows plots of BIG6 for $L = L*$. The reach length is to small to allow the tree passing over the biggest obstacles, but some wires cross smaller obstacles in the center of the chip. If all obstacles are horizontally-restricted, BIG6 becomes infeasible.

				Lengths			Running times in seconds		
Instance	|S|	|O|	L∗	L = 0	L = L∗	no obst.	L = 0	L = L∗	no obst.
Net 1	27	6809	158571	3570869	3316044	3241525	0.28605	6.94602	0.00041
Net 2	27	6809	158571	3567062	3310007	3148652	0.29015	6.95211	0.00042
Net 3	30	6809	158571	3635507	3507721	3313738	0.29373	6.95189	0.00044
Net 4	179	6809	158571	6188916	5827283	5737373	0.30901	6.98753	0.00158
Net 5	49	6809	158571	2854609	2635307	2582880	0.28108	6.91907	0.00068
Net 6	49	6809	158571	2845119	2581950	2557111	0.30171	6.92185	0.00059
Net 7	27	6809	158571	3570413	3282960	3246599	0.30061	6.97059	0.00038
Net 8	26	6809	158571	1745077	1665501	1615335	0.26909	6.90393	0.00033
Net 9	26	6809	158571	1954896	1880497	1837458	0.27356	6.92398	0.00035
Net 10	49	6809	158571	2454448	2229314	2144957	0.27614	6.90590	0.00057
Net 11	44	6809	158571	2262645	2080426	2050934	0.27789	6.90964	0.00057
Net 12	31	6809	158571	4038285	3736192	3646412	0.28854	6.95349	0.00041

Table 17: Results for BIG9. This instance contains both horizontally- and vertically-restricted obstacles.

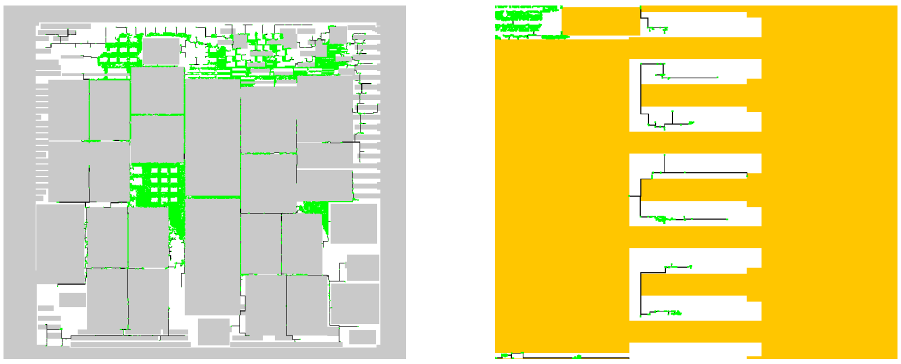


Figure 52: Result for instance BIG6 with L = L∗ (left). Restricting all obstacles horizontally (orange, right) makes BIG6 infeasible for L = L∗ due to the enclosed terminals (figure shows excerpt only).

Table 17 shows results for several nets of chip BIG9. This chip contains obstacles reaching to the very top layers, resulting in fully blocked area. At some places, exactly one layer remains unblocked creating horizontally- or vertically-restricted obstacles. One can see that letting the nets cross obstacles in the allowed direction (L = L∗) reduces the length significantly compared to the obstacle-avoiding case (L = 0). Figure 53 shows such a net that has to take a large detour in the latter case.

5.4.3 *Computing Lower Bounds in BonnRouteGlobal*

This section presents an application of the reach- and direction-restricted Steiner tree problem in global routing. Route costs do not only comprise congestion and timing costs but also so-called objective costs, which are accounted per wire length and per via and de-

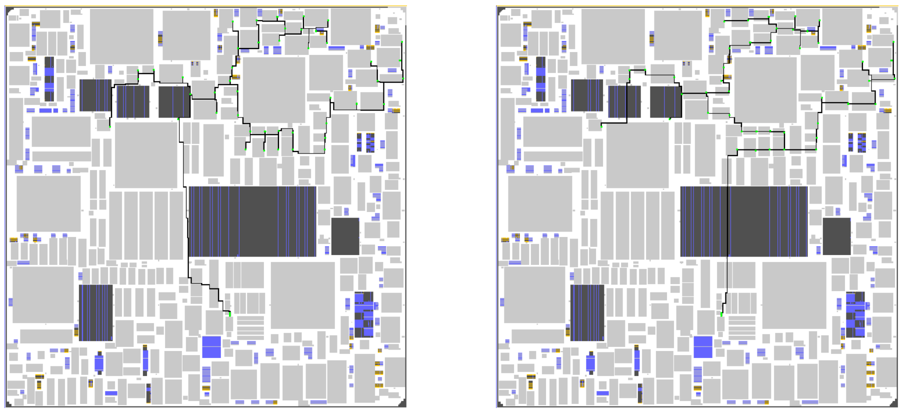


Figure 53: Net 4 of BIG9 with $L = 0$ on the left and $L = 158571$ on the right. There are obstacles that are fully-blocked (dark gray), vertically-restricted (blue), horizontally-restricted (orange), and length-restricted only (light gray). For $L = 158571$ the big obstacle in the center may be crossed vertically, reducing length substantially.

pend on the layer. This is to encourage the creation of short routes using a small number of vias even if congestion and timing resource prices are low. As for congestion, BonnRouteGlobal creates a resource for objective usage [49]. In this way, the overall wire length and via number can be controlled by the capacity of this resource.

To speed up the resource sharing algorithm, not all nets are rerouted in every phase [49]. Instead, if the route that was computed for a net in the previous phase is not much more expensive with regard to the current resource prices compared to the prices of the last phase, the net is not rerouted. To improve the estimation whether a net should be rerouted, BonnRouteGlobal computes a lower bound on the increase of the costs that is unavoidable for any route. Since for any global routing graph edge there is usually always a possible route avoiding that edge, the lower bound computation is restricted to the objective costs.

For a net, let R_{lb} a route with minimum possible objective resource usage. During resource sharing, given the costs c_{old} and the route R_{old} of the previous iteration and the costs c_{new} of the current iteration, a new route for the net is computed only if, for a fixed $\epsilon > 1$,

$$c_{new}(R_{old}) > \epsilon \cdot \left(c_{old}(R_{old}) + \sigma\left(c_{new\,|\,obj}(R_{lb}) - c_{old\,|\,obj}(R_{lb})\right)\right) \quad (14)$$

Here, $c_{old\,|\,obj}$ denotes the cost function c_{old} restricted to the objective resource, that is, all other resources have price zero. σ is the approximation factor of the oracle used in the resource sharing algorithm. If (14) does not hold the old route R_{old} is reused in the current iteration, saving running time. [49] shows that this modified oracle still has an approximation guarantee of $\epsilon\sigma$.

BonnRouteGlobal with dynamic local usage computes approximations for the lower bound routes R_{lb} by performing a path search on the global routing graph followed by the post-processing described in Section 3.3. Only objective costs are considered while congestion and timing costs are set to zero. We denote the resulting approximate lower bounds by *graph-based initial routing values*. This approach comes with the disadvantage that the global routing graph can be huge, making the path search slow. If there are only objective costs the granularity of the global routing graph is not needed. However, blockages must still be respected. The problem of finding a shortest coarse route under these circumstances can be formulated as a length- and direction-restricted Steiner tree problem: We choose points in the pin shapes as terminals. Let $L_n \subset \mathcal{L}$ the layers that net n is allowed to use for routing. Let A^l the union of the global routing graph edge areas on layer $l \in L_n$ for which the capacity has been estimated to be zero due to blockages and wires in the input. We then set $A_h := \bigcap_{l \in L_n \cap \mathcal{L}_y} A^l$ and $A_v := \bigcap_{l \in L_n \cap \mathcal{L}_x} A^l$. In this way, A_h denotes the area where there is no y-dimension layer with a positive global routing graph edge capacity, implying that the Steiner tree cannot run in y-dimension in A_h. Analogously, this holds for A_v and x-dimension layers. In the following experiments, we choose $L = \infty$ as the reach length because most buffering has already been performed beforehand.

After solving the reach- and direction-restricted Steiner tree problem, one can obtain a *geometric initial routing value* as follows: We multiply the length of the tree by the minimum objective wire usage over the routing layers L_n and add the costs of the minimum number of vias required to reach the routing layers from the pin shapes. However, in practice, this has been found to be a weak lower bound on the objective usage, leading to more resource sharing reroutes than the graph-based initial routing values obtained by the path search.

To improve the geometric approach, the Steiner tree can be embedded into the layers using the algorithm described in Section 3.3.2. This approach properly considers vias between pin shapes on different layers and at Steiner points of the tree, while also respecting different wire objective costs on the layers. However, this is slightly too pessimistic since the Steiner tree algorithm does not optimize the number of corners (in the embedded representation in $\mathbb{R}^2$), resulting in more vias than are obtained by the path search and the post-processing described in Section 3.3.

To address this issue, we perform the xy-optimization as described in Section 3.3.1 before embedding the Steiner tree into the layers. The xy-optimization attempts to embed adjacent Steiner nodes on the same x- or y-coordinates if doing so incurs the same costs, thus reducing the number of vias required to embed the route into the layers.

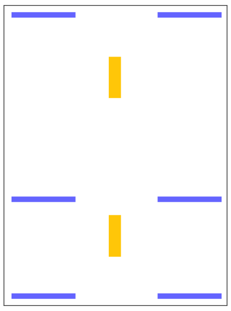


Figure 54: Blockages A_h (orange) and A_v (blue) on chip C_3.

Table 18 presents the results obtained by BonnRouteGlobal with the dynamic local usage and using the geometric initial routing values. The geometric initial routing values are almost identical to those obtained using the graph-based approach. On two instances the initial routing values of the geometric approach are even shorter, which can be well-explained by the superior post-optimization of the length restricted Steiner tree routine. An exception are A_1 and C_3, which have particularly many large blockages (Figure 54). If a pin is located on such a blockage, the current implementation of the Steiner tree algorithm connects it to all corners of the boundary of the blockage before connecting it to the remaining tree. In case of a large blockage, this might result in sub-optimal Steiner trees if nearby pins are located on a large blockage. During the computation of the coarse routes, BonnRouteGlobal adjusts the search-area on the global routing graph according to the computed initial routing values. If those are very large, substantial parts of the chip are labeled. This explains the large running time of the resource sharing algorithm on A_1, where a small number of nets account for large parts of the increase in running time compared to the graph-based initial routing values, despite the fact that the number of reroutes is almost the same as with the graph-based approach.

The geometric initial routing values themselves are computed much faster than the graph-based initial routing values, in particular on the largest instance A_1. The gain in running time compensates for the increased resource sharing running time. Moreover, the geometric approach shows a very small advantage in global routing quality in terms of wACE4, wire length, and via number. This demonstrates the feasibility of the geometric initial routing. In combination with a better handling of terminals on blockages and a slight increase of the value of ϵ in inequality (14), reflecting the better post-optimization, it should be possible to bring the number of resource sharing reroutes and their running time to the same level as in the runs with the graph-based approach. Altogether, this shows the potential of the length- and direction-restricted Steiner tree algorithm to deliver as good or even slightly better initial routing values than the graph-based approach, with a significantly shorter running time.

Chip	Nets	Approach	Initial routing time		Initial routing value		R.S. time		R.S. reroutes		wACE4		GR Wirelength		GR Vias	
A_1	1783142	Graph-based	02:50:27		1180200		16:33:11		4253149		96.85		302.7398		17442352	
		Geometric	00:51:58	-69.50 %	1181900	+0.14 %	18:26:13	+11.38 %	4249160	-0.09 %	96.88	+0.03 %	302.7106	-0.01 %	17444899	+0.01 %
B_1	1732566	Graph-based	00:35:52		133400		03:44:56		6498177		82.41		26.7469		15563943	
		Geometric	00:32:55	-8.22 %	133450	+0.04 %	03:43:58	-0.43 %	6402371	-1.47 %	82.40	-0.01 %	26.7485	+0.01 %	15558503	-0.03 %
B_2	1202357	Graph-based	00:23:25		97910		03:01:35		4472815		83.93		19.6469		11209677	
		Geometric	00:21:07	-9.81 %	97910	+0.00 %	03:06:49	+2.88 %	4512686	+0.89 %	83.78	-0.15 %	19.6440	-0.01 %	11204043	-0.05 %
B_3	372283	Graph-based	00:08:48		30447		00:31:22		882239		81.31		6.0654		3406589	
		Geometric	00:07:56	-9.77 %	30442	-0.02 %	00:34:22	+9.57 %	963801	+9.24 %	81.31	+0.00 %	6.0646	-0.01 %	3403339	-0.10 %
C_1	373556	Graph-based	00:03:56		14144		00:19:55		1216705		81.60		2.1312		3081397	
		Geometric	00:03:43	-5.51 %	14138	-0.04 %	00:20:12	+1.38 %	1223024	+0.52 %	81.61	+0.01 %	2.1311	+0.00 %	3081275	+0.00 %
C_2	245161	Graph-based	00:02:44		14915		00:10:23		478578		79.08		2.8789		2009673	
		Geometric	00:02:28	-10.14 %	14917	+0.01 %	00:10:39	+2.49 %	481811	+0.68 %	79.03	-0.05 %	2.8783	-0.02 %	2009102	-0.03 %
C_3	147877	Graph-based	00:01:44		10201		00:08:50		341595		77.11		2.0337		1205177	
		Geometric	00:01:30	-14.10 %	10236	+0.34 %	00:09:23	+6.25 %	363598	+6.44 %	76.91	-0.20 %	2.0337	+0.00 %	1205172	+0.00 %
Summary		Graph-based	04:06:59		1481217		1d 00:30:16		18143258		83.18		362.2428		53918808	
		Geometric	02:01:40	-50.74 %	1482993	+0.12 %	1d 02:31:39	+8.26 %	18196451	+0.29 %	83.13	-0.05 %	362.2108	-0.01 %	53906333	-0.02 %

Table 18: Results of BonnRouteGlobal with the dynamic local usage on 7nm instances (see Sections 4.1 and 4.2 for more details on the testbed and the metrics); once with the *graph-based* and once with the *geometric* approach using the length- and direction-restricted Steiner tree routine. In both approaches, this is followed by the xy- and z-optimization. *Lower bounds time* depicts the running time needed to compute initial routing values for all the nets (in *hours* : *minutes* : *seconds*); *Lower bounds value* is the computed value summed over all nets (no unit). All tests were performed without wires in the input, as they are not supported by the current implementation of the reach- and direction-restricted Steiner tree algorithm. To make the number of resource sharing phases more stable and to better compare running times, the initial routing values and the resource sharing algorithm were run single-threaded.

6 SUMMARY

In this thesis we consider the global routing problem which is a central task in chip design. Up to millions of sets of pins on a chip, so called nets, have to be connected through wires without intersecting each other and meeting many design rules. Global routing operates on a coarse grid graph modeling a condensed version of the chip, allowing to optimize global objectives. The result is used as guidance by a detailed routing algorithm to compute the exact wiring.

BonnRouteGlobal, the global router that is developed at the Research Institute for Discrete Mathematics, employs the resource sharing framework from [49] to share available routing space among the nets and to optimize signal delay through the wires. Traditionally, the impact on space of shorter wires that connect to pins is pre-estimated before long connections on the coarse grid graph are computed. In this thesis we establish a new optimization model, called the dynamic local usage, in which all wiring can be optimized simultaneously.

We discuss the impact of the dynamic local usage on BonnRouteGlobal and, based on [52, 53], develop algorithms to approximately optimal embed local wires, that connect to pins, into the chip image and into the layers. The algorithms implemented consider the current prices of the resource sharing algorithm and thus fit into the resource sharing framework. We evaluate the resulting global routing quality in terms of routability during detailed routing and demonstrate the capability of the dynamic local usage to persistently improve results in the routing flow of IBM with better signal delay, shorter wire length, and less design rule violations. At the time of writing, IBM is performing tests with the aim of enabling the dynamic local usage as default.

Another problem that often occurs during routing is the rectilinear Steiner tree problem. We consider a variant in which we are given a set of rectangles: Some may not be crossed at all, some only in horizontal direction, and some only in vertical direction. Moreover the length of any tree component on such a rectangle is bounded. This models the situation during buffering, which is also an important step in chip design. To speed up signal propagation, nets are subdivided by buffers, which must not be placed on top of obstacles, motivating bounding the length of unbuffered wiring on top of them. Special obstacle structures can make it impossible to place horizontal or vertical wires.

We present a fast 2-approximation algorithm for this problem. We provide a more thorough case distinction in the proof of the main theorem than [6, 25] which this work is based on, closing gaps in the previous proofs. Moreover, we slightly improve the extraction of the Steiner tree and extend the post-optimization by an edge substitution method from [20]. As a side result, we improve on the pre-processing time of the algorithm by [31] for the online maximum cost edge on tree path problem.

At the end, we present an application of the Steiner tree algorithm for the computation of lower bounds on wire length and via numbers with the dynamic local usage. The results point out the potential of our algorithm to generate strong lower bounds significantly faster than the currently used approach in BonnRouteGlobal.

BIBLIOGRAPHY

[1] Markus Ahrens, Michael Gester, Niko Klewinghaus, Dirk Müller, Sven Peyer, Christian Schulte, and Gustavo Tellez. "Detailed Routing Algorithms for Advanced Technology Nodes." In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 34.4 (2014), pp. 563–576.

[2] Markus Ahrens, Dorothee Henke, Stefan Rabenstein, and Jens Vygen. "Faster Goal-Oriented Shortest Path Search for Bulk and Incremental Detailed Routing." In: *Integer Programming and Combinatorial Optimization: 23rd International Conference, IPCO 2022, Eindhoven, The Netherlands, June 27–29, 2022, Proceedings*. Springer. 2022, pp. 15–28.

[3] Gaurav Ajwani, Chris Chu, and Wai-Kei Mak. "FOARS: FLUTE based Obstacle-Avoiding Rectilinear Steiner Tree Construction." In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 30.2 (2011), pp. 194–204.

[4] Charles Alpert, Andrew Kahng, CN Sze, and Qinke Wang. "Timing-Driven Steiner Trees are (Practically) Free." In: *Proceedings of the 43rd Annual Design Automation Conference*. ACM. 2006, pp. 389–392.

[5] Christoph Bartoschek, Stephan Held, Jens Maßberg, Dieter Rautenbach, and Jens Vygen. "The Repeater Tree Construction Problem." In: *Information Processing Letters* 110.24 (2010), pp. 1079–1083.

[6] Tilmann Bihler. "Rektilineare Steinerbäume mit längen- und richtungsbeschränkenden Blockaden." Bachelor's thesis. Research Institute for Discrete Mathematics, University of Bonn, 2015.

[7] Tilmann Bihler. "Rounding Fractional Global Routings." Master's thesis. Research Institute for Discrete Mathematics, University of Bonn, 2017.

[8] Daniel Blankenburg. "Resource Sharing Revisited: Local Weak Duality and Optimal Convergence." In: *30th Annual European Symposium on Algorithms (ESA 2022)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik. 2022.

[9] Jannis Blauth. "Steinerbäume mit Blockaden." Bachelor's thesis. Research Institute for Discrete Mathematics, University of Bonn, 2018.

[10] Jannis Blauth, Stephan Held, Dirk Müller, Niklas Schlomberg, Vera Traub, Thorben Tröbst, and Jens Vygen. "Vehicle Routing with Time-Dependent Travel Times: Theory, Practice, and Benchmarks." In: *arXiv preprint arXiv:2205.00889* (2022).

[11] Manuel Blum, Robert Floyd, Vaughan Pratt, Ronald Rivest, Robert Tarjan, et al. "Time Bounds for Selection." In: *Journal of Computer and System Sciences* 7.4 (1973), pp. 448–461.

[12] Jaroslaw Byrka, Fabrizio Grandoni, Thomas Rothvoß, and Laura Sanita. "An Improved LP-Based Approximation for Steiner Tree." In: *Proceedings of the 42nd ACM Symposium on Theory of Computing*. 2010, pp. 583–592.

[13] Po-Yan Chen, Bing-Ting Ke, Tai-Cheng Lee, I-Ching Tsai, Tai-Wei Kung, Li-Yi Lin, En-Cheng Liu, Yun-Chih Chang, Yih-Lang Li, and Mango C-T Chao. "A Reinforcement Learning Agent for Obstacle-Avoiding Rectilinear Steiner Tree Construction." In: *Proceedings of the 2022 International Symposium on Physical Design*. 2022, pp. 107–115.

[14] Wing-Kai Chow, Liang Li, Evangeline FY Young, and Chiu-Wing Sham. "Obstacle-Avoiding Rectilinear Steiner Tree Construction in Sequential and Parallel Approach." In: *Integration, the VLSI Journal* 47.1 (2014), pp. 105–114.

[15] Chris Chu and Yiu-Chung Wong. "FLUTE: Fast Lookup Table based Rectilinear Steiner Minimal Tree Algorithm for VLSI Design." In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 27.1 (2008), pp. 70–83.

[16] Kapoor Clarkson, Sanjiv Kapoor, and P. Vaidya. "Rectilinear Shortest Paths through Polygonal Obstacles in $\mathcal{O}\left(n(\log n)^2\right)$ Time." In: *Proceedings of the 3rd Annual Symposium on Computational Geometry*. ACM. 1987, pp. 251–257.

[17] Wikimedia Commons. *File: Intel 2nd Generation Core microprocessor codenamed Sandy Bridge Wafer.jpg — Wikimedia Commons, the Free Media Repository*. Licensed under CC BY-SA 2.0. 2022. URL: https://commons.wikimedia.org/w/index.php?title=File:Intel_2nd_Generation_Core_microprocessor_codenamed_Sandy_Bridge_Wafer.jpg&oldid=677531895.

[18] William Elmore. "The Transient Response of Damped Linear Networks with Particular Regard to Wideband Amplifiers." In: *Journal of Applied Physics* 19.1 (1948), pp. 55–63.

[19] Zhe Feng, Yu Hu, Tong Jing, Xianlong Hong, Xiaodong Hu, and Guiying Yan. "An $\mathcal{O}(n \log n)$ Algorithm for Obstacle-Avoiding Routing Tree Construction in the λ-Geometry Plane." In: *Proceedings of the 2006 International Symposium on Physical Design*. ACM. 2006, pp. 48–55.

[20] Stefan Fritsch. "Konstruktion von blockadenvermeidenden rektilinearen Steinerbäumen im VLSI-Design." Diploma thesis. Research Institute for Discrete Mathematics, University of Bonn, 2011.

[21] Michael Gester, Dirk Müller, Tim Nieberg, Christian Panten, Christian Schulte, and Jens Vygen. "BonnRoute: Algorithms and Data Structures for Fast and Good VLSI Routing." In: *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 18.2 (2013), pp. 1–24.

[22] Sudipto Guha and Samir Khuller. "Improved Methods for Approximating Node Weighted Steiner Trees and Connected Dominating Sets." In: *Information and Computation* 150.1 (1999), pp. 57–74.

[23] Eran Halperin and Robert Krauthgamer. "Polylogarithmic Inapproximability." In: *Proceedings of the 35th Annual ACM Symposium on Theory of Computing*. 2003, pp. 585–594.

[24] Stephan Held and Daniel Rotter. "Shallow-Light Steiner Arborescences with Vertex Delays." In: *Integer Programming and Combinatorial Optimization: 16th International Conference, IPCO 2013, Valparaíso, Chile, March 18-20, 2013. Proceedings 16*. Springer. 2013, pp. 229–241.

[25] Stephan Held and Sophie Spirkl. "A Fast Algorithm for Rectilinear Steiner Trees with Length Restrictions on Obstacles." In: *Proceedings of the 2014 International Symposium on Physical Design*. ACM. 2014, pp. 37–44.

[26] Stephan Held, Bernhard Korte, Dieter Rautenbach, and Jens Vygen. "Combinatorial Optimization in VLSI Design." In: *Combinatorial Optimization* (2011), pp. 33–96.

[27] Stephan Held, Dirk Müller, Daniel Rotter, Rudolf Scheifele, Vera Traub, and Jens Vygen. "Global Routing with Timing Constraints." In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37.2 (2017), pp. 406–419.

[28] Stefan Hougardy, Jannik Silvanus, and Jens Vygen. "Dijkstra Meets Steiner: A Fast Exact Goal-Oriented Steiner Tree Algorithm." In: *Mathematical Programming Computation* 9 (2017), pp. 135–202.

[29] Tao Huang and Evangeline FY Young. "ObSteiner: An Exact Algorithm for the Construction of Rectilinear Steiner Minimum Trees in the Presence of Complex Rectilinear Obstacles." In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 32.6 (2013), pp. 882–893.

[30] Xing Huang, Wenzhong Guo, Genggeng Liu, and Guolong Chen. "FH-OAOS: A Fast Four-Step Heuristic for Obstacle-Avoiding Octilinear Steiner Tree Construction." In: *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 21.3 (2016), p. 48.

[31] Andrew Kahng, Ion Măndoiu, and Alexander Zelikovsky. "Highly Scalable Algorithms for Rectilinear and Octilinear Steiner Trees." In: *Proceedings of the 2003 Asia and South Pacific Design Automation Conference*. ACM. 2003, pp. 827–833.

[32] Richard Karp. *Reducibility among Combinatorial Problems, Complexity of Computer Computations (RE Miller and JW Thatcher, editors)*. 1972.

[33] Annika Kiefner. "Minimizing Path Lengths in Rectilinear Steiner Minimum Trees with Fixed Topology." In: *Operations Research Letters* 44.6 (2016), pp. 835–838.

[34] Bernhard Korte, Dieter Rautenbach, and Jens Vygen. "BonnTools: Mathematical Innovation for Layout and Timing Closure of Systems on a Chip." In: *Proceedings of the IEEE* 95.3 (2007), pp. 555–572.

[35] Bernhard Korte and Jens Vygen. "Combinatorial Optimization: Theory and Algorithms, 6th edition." In: Springer, 2018. ISBN: 978-3-662-56039-6.

[36] Chung-Wei Lin, Shih-Lun Huang, Kai-Chi Hsu, Meng-Xiang Lee, and Yao-Wen Chang. "Multilayer Obstacle-Avoiding Rectilinear Steiner Tree Construction based on Spanning Graphs." In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 27.11 (2008), pp. 2007–2016.

[37] Chung-Wei Lin, Szu-Yu Chen, Chi-Feng Li, Yao-Wen Chang, and Chia-Lin Yang. "Obstacle-Avoiding Rectilinear Steiner Tree Construction based on Spanning Graphs." In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 27.4 (2008), pp. 643–653.

[38] Kuen-Wey Lin, Yeh-Sheng Lin, Yih-Lang Li, and Rung-Bin Lin. "A Maze Routing-Based Methodology with Bounded Exploration and Path-Assessed Retracing for Constrained Multilayer Obstacle-Avoiding Rectilinear Steiner Tree Construction." In: *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 23.4 (2018), pp. 1–26.

[39] Chih-Hung Liu, Shih-Yi Yuan, Sy-Yen Kuo, and Yao-Hsin Chou. "An $\mathcal{O}(n \log n)$ Path-Based Obstacle-Avoiding Algorithm for Rectilinear Steiner Tree Construction." In: *Proceedings of the 46th Annual Design Automation Conference*. ACM. 2009, pp. 314–319.

[40] Chih-Hung Liu, Shih-Yi Yuan, Sy-Yen Kuo, and Szu-Chi Wang. "High-Performance Obstacle-Avoiding Rectilinear Steiner Tree Construction." In: *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 14.3 (2009), p. 45.

[41] Chih-Hung Liu, Shih-Yi Yuan, Sy-Yen Kuo, and Jung-Hung Weng. "Obstacle-Avoiding Rectilinear Steiner Tree Construction based on Steiner Point Selection." In: *Proceedings of the 2009 International Conference on Computer-Aided Design*. ACM. 2009, pp. 26–32.

[42] Chih-Hung Liu, Chun-Xun Lin, I-Che Chen, DT Lee, and Ting-Chi Wang. "Efficient Multilayer Obstacle-Avoiding Rectilinear Steiner Tree Construction based on Geometric Reduction." In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 33.12 (2014), pp. 1928–1941.

[43] Jieyi Long, Hai Zhou, and Seda Ogrenci Memik. "EBOARST: An Efficient Edge-Based Obstacle-Avoiding Rectilinear Steiner Tree Construction Algorithm." In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 27.12 (2008), pp. 2169–2182.

[44] Kurt Mehlhorn. "A Faster Approximation Algorithm for the Steiner Problem in Graphs." In: *Information Processing Letters* 27.3 (1988), pp. 125–128.

[45] Friederike Michaelis. "Capacity Estimation in Global Routing." Master's thesis. Research Institute for Discrete Mathematics, University of Bonn, 2017.

[46] Matthias Müller-Hannemann and Sven Peyer. "Approximation of Rectilinear Steiner Trees with Length Restrictions on Obstacles." In: *Algorithms and Data Structures: 8th International Workshop, WADS 2003, Ottawa, Ontario, Canada, July 30-August 1, 2003. Proceedings 8*. Springer. 2003, pp. 207–218.

[47] Matthias Müller-Hannemann and Siamak Tazari. "A Near Linear Time Approximation Scheme for Steiner Tree among Obstacles in the Plane." In: *Computational Geometry* 43.4 (2010), pp. 395–409.

[48] Dirk Müller. "Fast Resource Sharing in VLSI Design." PhD thesis. Research Institute for Discrete Mathematics, University of Bonn, 2009.

[49] Dirk Müller, Klaus Radke, and Jens Vygen. "Faster Min–Max Resource Sharing in Theory and Practice." In: *Mathematical Programming Computation* 3.1 (2011), pp. 1–35.

[50] Daniel Rotter. "Timing-Constrained Global Routing with Buffered Steiner Trees." PhD thesis. Research Institute for Discrete Mathematics, University of Bonn, 2017.

[51] Pietro Saccardi. "Continuous Routing." PhD thesis. Research Institute for Discrete Mathematics, University of Bonn, 2022.

[52] David Sankoff and Pascale Rousseau. "Locating the Vertices of a Steiner Tree in an Arbitrary Metric Space." In: *Mathematical Programming* 9.1 (1975), pp. 240–246.

[53] Rudolf Scheifele. "Timing-Constrained Global Routing with RC-Aware Steiner Trees and Routing Based Optimization." PhD thesis. Research Institute for Discrete Mathematics, University of Bonn, 2019.

[54] Vera Traub and Rico Zenklusen. "Local Search for Weighted Tree Augmentation and Steiner Tree." In: *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM. 2022, pp. 3253–3272.

[55] David Warme, Pawel Winter, and Martin Zachariasen. "Exact Algorithms for Plane Steiner Tree Problems: A Computational Study." In: *Advances in Steiner Trees*. Springer, 2000, pp. 81–116.

[56] Yaoguang Wei, Cliff Sze, Natarajan Viswanathan, Zhuo Li, Charles Alpert, Lakshmi Reddy, Andrew Huber, Gustavo Tellez, Douglas Keller, and Sachin Sapatnekar. "GLARE: Global and Local Wiring Aware Routability Evaluation." In: *Proceedings of the 49th Annual Design Automation Conference*. ACM. 2012, pp. 768–773.

[57] Hao Zhang, Dong-Yi Ye, and Wen-Zhong Guo. "A Heuristic for Constructing a Rectilinear Steiner Tree by Reusing Routing Resources over Obstacles." In: *Integration, the VLSI Journal* 55 (2016), pp. 162 –175.

[58] Yilin Zhang, Ashutosh Chakraborty, Salim Chowdhury, and David Pan. "Reclaiming Over-the-IP-Block Routing Resources with Buffering-Aware Rectilinear Steiner Minimum Tree Construction." In: *Computer-Aided Design (ICCAD), 2012 IEEE/ACM International Conference on*. IEEE. 2012, pp. 137–143.