Routing based port assignment and buffering

Dissertation
zur
Erlangung des Doktorgrades (Dr. rer. nat.)
der
Mathematisch-Naturwissenschaftlichen Fakultät
der
Rheinischen Friedrich-Wilhelms-Universität Bonn

vorgelegt von
Fine Foos

aus
Düsseldorf

Bonn, Februar 2025

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

Gutachter/Betreuer: Prof. Dr. Stephan Held
Gutachter: Prof. Dr. Jens Vygen

Tag der Promotion: 14. April 2025
Erscheinungsjahr: 2026

Acknowledgments

During my work on this thesis, I was supported by many people. First of all I want to thank my supervisor Prof. Dr. Stephan Held for the support, contributed ideas and in-depth feedback to the thesis. I enjoyed collaborating in the development of Pangea ReUse. I would also like to thank Prof. Dr. Dr. h.c. Bernhard Korte, Prof. Dr. Jens Vygen and the entire professorial staff of the Institute for Discrete Mathematics for creating a great place to work and think. I will miss the Arithmeum.

My joy in this work came from both the great people and the interesting projects. So I would like to thank all my current and former colleagues for being excellent coworkers. Special thanks go to Dr. Siad Daboul for supervising my work as a student, to Dr. Pietro Saccardi for leaving me the beautiful code of BonnPangea (and chip-scenario), and to Susanne Armbruster for being the best office partner I could have hoped for. Thank you to everyone organizing and participating in our institute's cocktail pong events, especially to those who lost and then brought cake.

Big thanks also go to everyone who helped me proofread preliminary versions of this thesis: Susanne Armbruster, Paula Heinz, Ragni Hinderling, Luise Puhlmann and my mother.

I would also like to thank several IBM employees for the great collaboration regarding BonnPangea and helping me test BonnRouteBuffer: Bill Dougherty, Greg Klos, Genevieve Shafer and Jesse Surprise.

Thank you to my family for supporting me throughout my studies and of course all the years before.

Lastly, I want to thank my partner Svenja Dubendorff. You endured the years of me being away from home (which are finally over!), you support me in everything sensible I do and you gave me the occasional (or less occasional) nudge to actually get up and work. And most importantly: Thank you for bringing a fluffy whirlwind into my life, I did not know what I was missing before.

Contents

1 Introduction **5**

2 Preliminaries **7**
- 2.1 Basic notions of chip design . . . 7
- 2.2 Embedded routing . . . 8
- 2.3 Applications . . . 9
  - 2.3.1 Pangea . . . 9
  - 2.3.2 BonnRouteBuffer . . . 10
- 2.4 Global routing framework . . . 11
  - 2.4.1 Resource sharing algorithm . . . 12
  - 2.4.2 Arrival time customers . . . 13
  - 2.4.3 Incorporating further constraints . . . 13
- 2.5 Timing . . . 14
  - 2.5.1 Elmore delay model . . . 15
  - 2.5.2 Linear delay model . . . 16

3 Algorithms for timing-constrained global routing **17**
- 3.1 Global routing oracle . . . 17
  - 3.1.1 Topology generation . . . 17
  - 3.1.2 Embedding . . . 18
- 3.2 Topology algorithms . . . 19
  - 3.2.1 Reachaware topologies . . . 19
  - 3.2.2 Timing-aware topologies . . . 21
- 3.3 Tighter approximation for the uniform cost-distance Steiner tree problem . . . 23
  - 3.3.1 Main theorems . . . 23
  - 3.3.2 The $(1+\beta)$-approximation algorithm . . . 24
  - 3.3.3 Improving the approximation ratio . . . 25
- 3.4 Cost-distance router . . . 33
  - 3.4.1 Iterative matching algorithm . . . 33
  - 3.4.2 Practical improvements . . . 34

4 Pangea **37**
- 4.1 Preliminaries and problem formulation . . . 37
- 4.2 Related work . . . 41
- 4.3 Pangea topologies in theory . . . 41
- 4.4 Standard pangea flow . . . 45
  - 4.4.1 Continent border blockages . . . 45
  - 4.4.2 Global routing . . . 46
  - 4.4.3 Track assignment . . . 50
  - 4.4.4 Port cutting . . . 51

4.5 Pangea replay . . . 51
4.5.1 Trading off replaying ports with other pangea objectives . . . 53
4.5.2 Replaying ports as resource . . . 54
4.6 Pangea reuse . . . 55
4.6.1 Setting and problem formulation . . . 56
4.6.2 Reuse flow . . . 57
4.6.3 Port intervals . . . 58
4.6.4 Blockages on the continent border . . . 66
4.6.5 First routing pass . . . 66
4.6.6 Applying continent equivalence . . . 67
4.6.7 Second routing pass . . . 70
4.6.8 Difficult and reuse incompatible input . . . 70
4.6.9 Alignment of opposite ports . . . 77
4.6.10 Reusing feed-throughs . . . 86
4.6.11 Pangea reuse in practice . . . 89

5 BonnRouteBuffer **91**
5.1 BonnRouteBuffer in resource sharing . . . 91
5.2 Dynamic program for buffering . . . 92
5.3 Enhancements and fixes . . . 93
5.3.1 Slew computation . . . 93
5.3.2 Speed ups . . . 95
5.3.3 Other improvements . . . 96
5.4 Experimental results . . . 97
5.4.1 Wire synthesis flow . . . 97
5.4.2 Metrics and instances . . . 98
5.4.3 Different slew pessimism values . . . 101
5.4.4 Previous vs. improved BonnRouteBuffer . . . 105
5.4.5 BonnRouteBuffer in the IBM flow . . . 113
5.5 Future work . . . 114
5.5.1 Slew computations . . . 114
5.5.2 Fixing electrical violations in ripup-and-reroute . . . 114
5.5.3 Fast buffering of timing-uncritical nets . . . 115

Summary **117**

Bibliography **119**

Chapter 1

Introduction

As computer hardware becomes more complicated and intricate, so become mathematical problems and algorithms used for their design. Modern computer chips contain billions of transistors and the wire length is measured in kilometers. This just gives a glimpse into the scope of *very large scale integrated (VLSI) chip design*, the field of optimizing processor chips used inside supercomputers.

The tasks of chip design range from designing the logical structure and placing components to routing and timing optimization. However, it is not feasible to merely approach these one after the other. Their objectives are heavily intertwined. During most of the physical design flow, the main objectives are placement density, routing congestion, timing and power consumption.

This thesis contains two main topics, port assignment and global buffering, which we examine from a mathematical and a practical point of view.

Port assignment designs the interface between large chip components. It decides where the routing of one component should connect to the routing of a neighboring one. For a general problem formulation allowing multiply instantiated components, we examine the conditions for feasible solutions to exist and present an algorithm to compute such.

In global buffering, we insert many small components (so-called cells) along the wiring in order to reduce signal delays. Due to the physical properties of wiring and cells, this improves signal delay from roughly quadratic down to almost linear in the wire length [Bar+06]. We need to compute Steiner trees minimizing a combination of net length and path length. For this, we present an algorithm approximating an objective function that occurs when buffering a single net.

We cover the tools BonnPangea and BonnRouteBuffer. These tools compute practical solutions for port assignment and global buffering.

Main contributions

We present an approximation algorithm for the uniform cost-distance Steiner tree problem (Section 3.3, already published in [FHS23]). This problem arises in the Lagrangian relaxation of global buffering. We achieve an approximation factor of <2.05, which improves upon the previously best factor of 2.39 in [KH20]. Given an exact oracle for the Steiner minimum tree problem, our approximation is tight with respect to the lower bound $C_{SMT}(T \cup \{r\}) + D(T, r, w)$. This is because the lower bound can deviate from the optimum value by a factor of $1 + \frac{1}{\sqrt{2}}$, while we can compute a $\left(1 + \frac{1}{\sqrt{2}}\right)$-approximation using the Steiner tree oracle. All previous analyses compare their results against this

lower bound.

We devise the first mathematical problem formulation of the standard pangea problem (Sections 4.1 and 4.3). Here, the goal is to compute a port assignment allowing for short paths, little wiring congestion and short total net length. We also present the first description of BonnPangea, the tool used to solve this problem in practice (Section 4.4). We then extend the problem definition to the case of pangea replay. In pangea replay, a partial port assignment is given in the input. It needs to be extended by the computed result. We describe the adapted algorithm to solve this problem as well (Section 4.5).

In Section 4.6, we present a major new contribution called Pangea ReUse. Here, we compute a port assignment in the presence of multiply instantiated components, an important practical application. The port assignment on equivalent components must coincide. The problem becomes much more difficult compared to traditional port assignment. We formalize the pangea reuse problem and develop two algorithms: A practical algorithm is described in Sections 4.6.2 to 4.6.7. This algorithm was also implemented and integrated into the existing pangea tool as part of this thesis. It works under certain restrictions which are mostly fulfilled in practice. A second algorithm is presented in Section 4.6.9. We can prove that this algorithm solves the pangea reuse problem correctly on instances that are substantially less restricted.

For BonnRouteBuffer, we develop a more accurate propagation of slew limits during the dynamic program and a more accurate propagation of slews through the computed repeater tree. We show that these improvements, together with many minor fixes, lead to more efficient global buffering solutions that also contain significantly fewer electrical violations.

This thesis is organized as follows. Chapter 2 introduces general notions of chip design, focusing on routing and timing optimization. Chapter 3 covers algorithms used for timing-constrained global routing. Chapter 4 gives details on pangea. There we present the three different modes of BonnPangea, including Pangea ReUse. Finally, Chapter 5 addresses BonnRouteBuffer and showcases how the author of this thesis was able to improve on many aspects of it compared to its previous state (see [Rot17], [Dab21] and [Dab+23]).

Chapter 2

Preliminaries

2.1 Basic notions of chip design

This section introduces the most important notions used throughout the entire thesis. A *cell* or *gate* is a small physical component made up of a few connected transistors representing a simple logical function such as *AND*, *NOT* or even the identity function. These functions also create more complex logical functions by composition. Cells represent the building blocks of the chip. To connect wiring to them, cells each have several *pins*. For our purposes, we only ever consider the input pins (e.g. one for a *NOT*-function, two or more for an *AND*-function and so on) and the output pin. In addition to the cells' pins, a chip also has *primary input and output pins*, which are placed on the chip itself. They have a coordinate lying inside the *chip area*, which is a rectangle given in $\mathbb{R}^2$.

To state which pins need to be connected to which, we use the notion of a *net*. A net is a set of pins containing exactly one *source* which is either an output pin of a cell or a primary input. The rest of the net's pins are called *sinks*. They are either input pins of cells or primary outputs. The set of all nets forms the *netlist*.

When routing a net, the task is to compute wiring connecting all of the net's pins. Wiring can be placed on multiple *routing layers*. On each such layer, wire segments may either go only horizontally or only vertically (in the *preferred direction*), and such horizontal and vertical layers alternate. Wires on adjacent layers can be connected by inserting a *via* in between the layers.

In (global) routing, each layer is cut into tiles forming a grid. The tiles are connected according to their layer's preferred direction. Together with edges representing vias, this forms the *global routing graph*, a 3-dimensional grid graph. Pin positions are projected to the tile-centers of the respective surrounding tiles and each net gets connected via a *global route*, a Steiner tree in that grid graph connecting tile-centers to tile-centers. An example global routing graph and a route are depicted in Figure 2.1.

In some applications, *blockages* will play a role. A blockage is a rectangular shape on either the placement layer or a routing layer. No gates may be placed on a blockage on the placement layer, while no routing (including vias) may be placed intersecting a routing blockage. These blockages can be either user-specified or stem from already placed components not allowed to be changed.

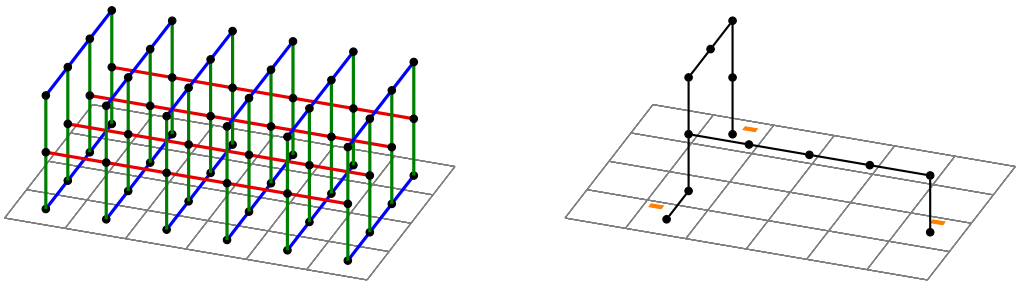


Figure 2.1: A global routing graph with three routing layers (left) and a global route for a three terminal net (right). Global routing tiles are depicted by the gray grid. Horizontal, vertical and via routing edges are red, blue and green, respectively. Pins are orange rectangles.

2.2 Embedded routing

The core of this thesis is to use a very flexible approach to global routing to be able to tune the computed routes towards certain specific goals.

As the routing task itself is so vast, even when only considering feasibility, it usually gets split into global routing and detailed routing. Global routing places wiring on a rough tile grid and only connects tile-centers to tile-centers (see previous section). It balances density constraints with other objectives. This way, not too much wiring is placed in the same area, the total wire length does not get too large, and single connections are not routed with a too large detour. In contrast, detailed routing places the wires on their exact positions and connects directly to the pin shapes. Additionally, it considers further design rules such as minimum distance constraints between unconnected wire segments. To do this in a feasible running time, detailed routing uses the already computed global routes and follows their structure as closely as possible. Because detailed routing considers each net to be routed one after the other, it also relies on the global routing to have made the decisions which nets should be routed through which areas.

The two most natural objectives in global routing are wire length and congestion. Wire length is the sum of the lengths of all wire segments. This is also called net length. Congestion measures how dense the wiring is. For this, we will use the notion of the wACE4 metric [Wei+14], which is a weighted sum of the routing densities in the 5% densest wiring areas. Depending on the problem at hand, global routing must satisfy other constraints as well. This can include timing constraints, i.e. making sure paths to critical sinks are as straight as possible and route on higher layers, which typically induce less wire delay. Or it can include further structural constraints which we will see in Chapter 4.

Combining a high level view to compute the overall structure of a route with a detailed view for the exact tiles that a single route segment should be placed on is a rather hard problem. These two are separated into individual steps: *Topology generation* and *embedding*. First, we compute a 2-dimensional Steiner tree connecting the source to the sinks. Here, we consider length according to the ℓ_1-metric. We can incorporate many further objectives such as regarding blockages or signal delay. Second, we embed this topology into the global routing graph using a variant of Dijkstra's algorithm [Dij59]. Here, we also decide the wire type on each used global routing edge.

The basis of the used routing engine was already developed in [Hel+18] and extended for the use of BonnRouteBuffer by [Dab+23]. Further adaptations, in particular for the use case of pangea, were implemented by the author of this thesis.

2.3 Applications

2.3.1 Pangea

Before the earth's continents emerged the way they look today, there was a single large continent called *pangea* surrounded by the superocean *panthalassa*. During the early Jurassic era, this continent broke apart into the sections we know today[1].

Analogous to this process, the pangea flow is used to break apart one large chip into smaller components called *continents*. The goal of this division is to be able to optimize each continent separately instead of having to deal with one very large instance all at once. This drastically speeds up every computational step and allows to exert much more effort in individual optimizations. To make this possible, the pangea flow makes sure the fully optimized continents properly fit together in the final result. This means that for every net with pins inside multiple continents, there has to be an agreement on where the wiring of that net should cross the continent borders. Having fixed these positions, we know the final continents will be compatible, no matter how their interior components and routing will end up looking like.

The decision which areas should make up continents is left to the user. To achieve the aforementioned benefits, it makes sense to choose continents such that their interdependence is rather low. Further, it is advantageous to choose continents of roughly the same size, which will make the optimization of the individual continents more efficient when done in parallel.

A commitment that a net should cross a continent border at a specified position is called a *port*. In addition to its location, every port comes with a predecessor (the net's source or another port) and a set of successors (net sinks or other ports), determining how the net will split into individual nets inside each continent. The ports of a net together with its pins form an arborescence rooted at the source with the set of leaves equal to the set of net sinks. This arborescence is called the *port graph*.

To compute port graphs for all nets, the pangea flow routes all relevant nets and then creates ports at each intersection of a route with a continent border. Of course, having to route or otherwise optimize the entire chip at once is exactly what pangea aims to avoid. To make this task feasible anyway, a number of simplifications are made. The two most important ones are ignoring all nets contained in single continents and using a very large tile size which leads to a comparably small global routing graph (see Section 2.4 for more details). With these simplifications, it is possible to compute a global routing in acceptable time, because both the number of routing tasks and the scope of each task are reduced.

The computed global routes now go from tile-center to tile-center. Placing ports in this state would make no sense, as many ports would end up lying on top of each other. However, to ensure the computed ports can be routed to, they must not overlap at all. As detailed routing would take much too long and is not necessary here to spread out the wiring, track assignment is used instead. One can think of routing tracks as the routing edges in detailed routing. They represent positions where wire segments may be placed next to each other without violating distance requirements. In track assignment, all wire segments crossing a continent border are assigned to such a routing track, ensuring the resulting ports are overlap-free.

For pangea, the key aspect in this flow is the topology generation, the first step in computing a single route. To minimize unwanted effects of fixing certain points of a route beforehand, the number of ports should be kept low. Furthermore, port positions must not enforce large detours in source-to-sink connections of nets. But they need to simultaneously allow for an overall short net length and low congestion. So topology

[1] https://www.britannica.com/place/Pangea

generation has to make sure that

a) both net length and source-to-sink paths are short and

b) routes do not switch between continents back and forth too often.

This is achieved by using a hierarchical approach: First, compute how the continents should be connected to each other. Then, how the topology within each continent should be constructed.

The new mode PANGEA REUSE, giving rise to a completely revised flow, was developed as part of this thesis together with Stephan Held and Max Mundt (also see [Mun23]). Before, every continent was designed separately using the port positions computed by the pangea flow. However, there are chips on which a certain large component occurs multiple times, e.g. when having many processor cores of the same kind. In these cases, it is preferable to only consider one instance of this continent in the later optimization flow. Otherwise, very similar work has to be carried out multiple times. This would lead to a waste of both human and computational resources. On the other hand, several continents which are exactly identical with regards to placement, nets and port configurations can be optimized as one instance. The finished continent will then simply be instantiated multiple times on the final chip.

The key here is to get the port positions to be identical across all equivalent continents. Simply taking the computed result from one of the individual continents is not feasible because of two reasons: To begin with, the ports of a continent must be compatible with those of adjacent continents. Secondly, blindly using the port positions of one of the equivalent continents for the other ones can introduce large source-to-sink detours as well as degrade congestion and net length. PANGEA REUSE solves these issues by taking each instance of the equivalent continents into account.

A further extension of PANGEA REUSE enables even continents that are not 100% equal to be considered equivalent. Tiny differences arise for example when nets need to route through one continent but not through an equivalent one, or when some pin is connected to something outside the continent, but an equivalent pin is not.

2.3.2 BonnRouteBuffer

Timing optimization is one of the largest tasks in chip design. Without taking into consideration how long a signal takes from one pin to another, chips cannot be optimized well. In fact, optimizing congestion and wire length will in many cases actively worsen signal delays. This means that a trade-off is needed. One way to improve delay characteristics of a route is to insert *repeaters* subdividing the wire segments. Repeaters are gates implementing either the inversion function (inverters) or the identity function (buffers). Whenever inverters are used, an even number of them has to be inserted into each source-to-sink path in order to retain the logical function. The reason that repeaters can decrease signal delay is that a gate shields off the capacitance behind it. Without repeaters, signal delay will depend roughly quadratically on the wire length (see Section 2.5). Thus, breaking up a long wire segment and dividing it into several shorter ones can improve timing. In fact, when done optimally, one can achieve a delay that depends only approximately linearly on the wire length [Bar+06].

BONNROUTEBUFFER (see [Rot17], [Dab21] and [Dab+23]) is a powerful tool for inserting repeaters to speed up the signals. In doing so, many constraints and side-objectives are considered: Repeaters can only be placed in such a way that they do not overlap each other or other gates, and a too high placement density will hinder accessing the pins in routing. Furthermore, there are placement blockages forbidding any repeater from being placed in a given area. Additionally, certain thresholds have to be observed giving bounds on capacitance values and signal slews (see Section 2.5 for more explanation). In addition to timing itself, BONNROUTEBUFFER takes power

consumption into account as a side-objective. Lastly, net length and wire congestion are considered.

BonnRouteBuffer is integrated into the global routing framework described in Section 2.4, meaning that it will route all nets as well as insert repeaters into them. Every time after a net is routed, repeaters are inserted along the route.

When computing routes in the first step, the objectives must already incorporate the fact that repeaters will be inserted as a second step. Therefore, we use a timing model that approximates an optimal repeater placement. This means signal delay is modeled as being linear in the length of a wire. The factors used depend on the exact wire characteristics such as width and layer and are all computed beforehand. However, many nets have more than one sink, hence trees are needed instead of simple paths. A path with many bifurcations will have more signal delay than the very same path without any bifurcations. This is due to the added downstream capacitance given by the paths that are branching off. Hence, it can be beneficial to also keep the number of bifurcations on critical paths in check. Routes going through areas with high placement density or even placement blockages will impede the possibility to insert repeaters. That means such areas have to be considered as well.

Most of these issues need to be addressed by the overall structure of a route, i.e. already in topology generation. When the bifurcation count is considered, it needs to be done in this step as it will remain fixed later on. Whether the route can afford to go over a large placement blockage (where no repeaters can be placed) also needs to be decided in this step. Depending on the main focus of optimization, there are different options for how to compute a good topology. We will discuss these in Section 3.2.

BonnRouteBuffer is integrated into a more extensive timing optimization flow called *wire synthesis*. In addition to inserting repeaters, other delay improving techniques are used there: In the process of *gate sizing*, a specific variant (size) for each gate is chosen. Most logic gates exist in multiple different sizes. Apart from in their footprint, these gates differ in certain delay characteristics as well as in their power consumption. So the task of gate sizing is to improve timing while also keeping the power consumption as low as possible. See for example [Dab+18a] and [Dab+18b]. There are also more local improvement strategies that focus on improving the worst path. The *refine placer* moves gates on the worst path in such a way that the wire length and thus also the signal delay on that path are reduced ([Boc+15], [Lüd23]).

Prior to the work on this thesis, BonnRouteBuffer was in the state of a prototype. It contained inexact computations and heuristics and was not fine-tuned well. Among other improvements, the slew propagation in both the timing analysis inside BonnRouteBuffer as well as the dynamic program computing the actual buffering (the "heart" of BonnRouteBuffer) were rewritten to be more exact and thereby allow for a less pessimistic view of slew limits. Additionally, several bugs were fixed in the process of improving its results.

2.4 Global routing framework

Global routing is the task of computing the rough position of wire segments connecting given positions on a chip. Instead of considering exact wire locations, each layer of the chip area is cut into rectangles called *tiles*. Connecting adjacent tiles leads to a 3-dimensional grid graph. Since layers only allow for either horizontal or vertical routing in an alternating manner, edges in opposing directions are removed. The resulting graph is called the *global routing graph*. An example global routing graph with few tiles and three layers was already seen in Figure 2.1.

A *pin* is an input or output of a cell. Since cells are already placed inside the chip area when routing is conducted, pins are associated with a position on the chip. While the

actual pin shapes are collections of axis-parallel rectangles, for our purposes it suffices to consider a pin position to be a single point on a given layer. The information how to connect these pins is given by the *netlist*. The netlist consists of many *nets*, which in turn consist of one pin acting as the source and one or more pins representing the sinks of the net. Sources are outputs of cells and sinks are inputs.

Connecting a net with wire makes sure a signal can travel from a cell's output to the corresponding next cells' inputs. We *connect* a net by computing a *global route*. A global route is a Steiner tree inside the global routing graph, where the terminals are the tiles corresponding to the net's pins. Further, we associate a *wire type* with each edge of the route giving details on the wire width, resistance and capacitance. These properties will be needed for the timing analysis, see Section 2.5.

A global route only states through which tiles the final wiring should go, but does not specify the exact routing tracks. This simplification is necessary in order to be able to handle all nets simultaneously. To make sure that it will be possible to compute exact and overlap-free wire positions later on, certain constraints have to be considered. In particular, every global routing edge has a *capacity*, stating how many wires may pass through it.

The goal in global routing is, first and foremost, to find routes for all nets such that the capacity constraints are not violated. Under this constraint, the objective in the simple global routing problem is to minimize net length, i.e. the total summed length of all computed routes. A formal definition of this problem is as follows.

THE SIMPLE GLOBAL ROUTING PROBLEM

Instance: A global routing graph $G = (V, E)$ with edge capacities $c : E \to \mathbb{N}$ and a netlist $\mathcal{N} = (N_i \subset V)_i$.

Task: For every net N_i, compute a Steiner tree R_i in G, such that for each global routing edge $e \in E$, we have

$$\sum_i \mathbb{1}_{E(R_i)}(e) \leq c(e).$$

Under all such solutions, minimize

$$\text{net length} := \sum_i \sum_{e \in E(R_i)} \text{length}(e).$$

2.4.1 Resource sharing algorithm

As already computing a single minimum-length Steiner tree is NP-hard [Kar72], there is no hope of solving the simple global routing problem exactly in polynomial time. Instead, we will consider a framework that allows us to approximate an optimum solution as well as add further constraints to the problem. For this, we will have a look at the resource sharing algorithm by [MRV11]. The idea is that we have resources $\mathcal{R}$ which we want to distribute between customers $\mathcal{C}$. Each customer has some freedom in regards to its needed resources. Feasible resource allocations for the individual customers are given by closed convex sets $B_C \subset \mathbb{R}^{\mathcal{R}}$ for each $C \in \mathcal{C}$. A feasible resource allocation for all customers $b : \mathcal{C} \to \mathbb{R}^{\mathcal{R}}$ assigns to each customer C a resource allocation $b(C) \in B_C$ such that $\sum_{C \in \mathcal{C}} b(C)_r \leq 1$ for each $r \in \mathcal{R}$, i.e. such that the total resource usage does not exceed 1 for any resource.

In our case, the global routing edges make up the resources. The customers are given by the nets. Further, for a customer/net C, B_C is the convex hull of incidence vectors of all Steiner trees for C, where each entry is divided by the capacity of the corresponding edge:

$$B_C \coloneqq \text{convex-hull}\left(\left\{\left(\frac{\left(\chi^F\right)_r}{c(r)}\right)_{r\in\mathcal{R}} \;:\; F \text{ Steiner tree for } C\right\}\right)$$

When defined like this, a total usage of 1 is equivalent to an edge being used to its full capacity, but not beyond. Hence, there is a one-to-one correspondence between feasible resource allocations for all customers and convex combinations of Steiner trees for each net, such that the edge capacities are observed.

To minimize net length, an additional resource representing the net length budget can be added. A good capacity for this resource can be chosen using binary search.

The resource sharing algorithm by [MRV11] maintains prices for the resources and repeatedly calls an oracle (see Section 3.1) to compute minimum-cost resource allocations for the individual customers with regard to these prices. Resource prices are updated multiplicatively in a way such that much demanded resources become more expensive over time. This is done over many phases and in the end, a convex combination containing all computed solutions is returned for every customer. Given that the oracle computes a σ-approximate minimum-cost resource allocation for individual customers, this yields an approximation factor arbitrarily close to σ.

After using the resource sharing algorithm, we end up with a fractional solution. To compute an integral solution, the first step is randomized rounding. After outliers are removed, for each net one of the previously computed Steiner trees is chosen at random. The probabilities are proportional to the factors appearing in the convex combination. This is done separately for each net. Using a result from [RT87], this results in a solution that does not violate many resource capacities.

The remaining violations are fixed by a ripup-and-reroute procedure recomputing solutions for individual nets.

2.4.2 Arrival time customers

The model so far can compute routes for all nets while minimizing net length and keeping the wiring congestion feasible. However, there is no mechanism controlling the time a signal needs from its start to its endpoint. But this is a crucial characteristic of a route and especially needed in BonnRouteBuffer.

The approach used throughout this thesis was introduced by [Hel+18]. In addition to the routing resources for the global routing edges and the net length budget, there are *timing resources*: We add one timing resource for every sink in each net. This resource represents the delay budget that a signal has to traverse the net to the given sink. With only these resources, the timing of different nets is still separate. To overcome this problem, *arrival time customers* are used as additional customers. One arrival time customer is added for each gate. The set of feasible solutions for an arrival time customer is a time interval inside which the signal can arrive at the associated gate. The earlier the chosen solution lies inside this interval, the more usage it takes from the timing resources going into the gate. Vice versa, a late solution takes more usage from the timing resources leaving the gate. As shown in [Hel+18], the existence of feasible arrival times at the gates ensures that overall timing constraints are met.

2.4.3 Incorporating further constraints

In BonnRouteBuffer, placement and power constraints are considered next to routing congestion and timing. Both can be incorporated into the global routing framework

natively.

For placement, it is important that the footprints of all gates fit into the area where they were placed. To model area locally as opposed to only making sure all gates can be placed *somewhere* on the chip, the chip area is cut into placement bins. Often times, these are the same as the global routing tiles. Each placement bin is associated with a separate placement resource. Its capacity depends on the size of the bin and overlapping placement blockages. Whenever a repeater is placed during BonnRouteBuffer, the solution has to pay a cost according to the size of the repeater relative to the surrounding placement bin.

Incorporating the consideration of power requires some assumptions to be made. When a power budget is given, it is possible to add a single resource representing power with that capacity. Every route then consumes of this resource exactly the route's power consumption divided by the power budget. In the end, feasibility for the power resource translates directly to the power budget being observed. However, no set budget for power consumption is given. Rather, the solution should have a reasonable power consumption while satisfying the other constraints. In practice, this is achieved by estimating a power budget and adjusting it in every phase of the resource sharing algorithm.

2.5 Timing

This section will introduce notions regarding signals and their delay properties. For this, we will assume routes to be directed away from the source.

A *signal* is a voltage change from 0 to V_{dd} (the chip's operating voltage) or vice versa. A voltage change from 0 to V_{dd} is called a *rise signal*, a change from V_{dd} to 0 is called a *fall signal*. When a signal arrives at the source of a net, it will be transmitted to the sink of said net after a short delay. Similarly, a signal can go from an input to an output of a gate. The *arrival time* of a signal at some position is the point in time when the voltage crosses the 50% V_{dd} threshold at that position. The arrival time at a pin p is denoted as $at(p)$. The *delay* from pin p_1 to pin p_2 is the difference in arrival times at these points: $delay(p_1, p_2) = at(p_2) - at(p1)$. We speak of *wire delay* if the pins are on the same wire path, and of *gate delay* for the time a signal takes to go from a gate's input to the output.

For primary input pins, the arrival times are given in the input. Given delay functions (e.g. by a timing model on fixed routes), we can then compute the arrival times of all pins.

Required arrival times state the latest point in time at which a signal should arrive. The required arrival time of a pin p is denoted by $rat(p)$. Required arrival times for primary outputs are given in the input. Required arrival times on the other pins can then be computed given the delay functions of a timing model.

For any given pin p, we define the *slack* to measure how much later the signal is allowed to arrive at p compared to the current arrival time: $slack(p) := rat(p) - at(p)$. If the slack is non-negative at all pins, every signal arrives in time.

A further important property of a signal is its *slew*, which is the time the voltage is inside the [10% V_{dd}, 90% V_{dd}] interval. Both arrival time and slew are marked on an example signal slope in Figure 2.2. In general, a steeper, i.e. smaller, slew leads to faster signals. Once the slew of a signal crosses a given threshold, the *slew limit*, there is a *slew violation*. The value of the slew violation is given by $slew - slew\_limit$.

Each wire segment further has *capacitance* and *resistance* values. Gate inputs and primary outputs also have a *capacitance* value. The *downstream capacitance* of any point in a route is the sum over all the capacitances of wire segments and pins that can be reached from the point through wiring (see Section 2.5.1). As for slew values, the

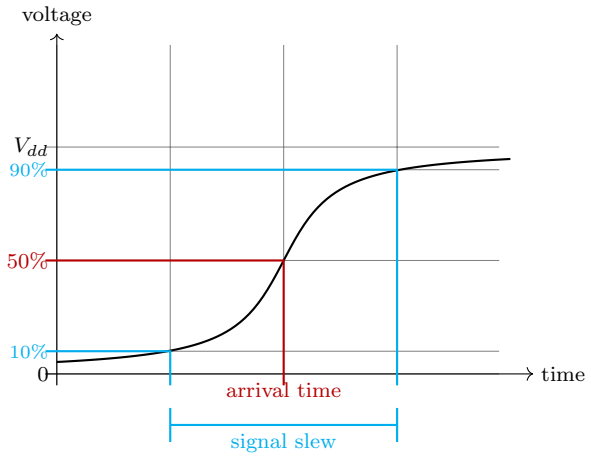


Figure 2.2: An example slope of a rise signal. The arrival time is marked at the 50% V_{dd} threshold. The signal slew is marked as the interval between the 10% and 90% voltage thresholds.

downstream capacitance also must not exceed a certain threshold called the *load limit*. Otherwise, a *load violation* occurs, where again the value of the violation is given by $\mathit{downstream\_capacitance} - \mathit{load\_limit}$.

2.5.1 Elmore delay model

A simple yet quite precise model to describe signal delays is the *Elmore delay model* introduced in [Elm48]. This delay model, also called the *RC-delay model*, models a wiring tree with gates by resistors and capacitors.

We will use the following notation to formally define the Elmore delay. Fix a wiring tree Y rooted at its source r. For a vertex $v \in V(Y)$, let Y_v denote the sub-tree of Y rooted at v. Further, let $\mathrm{downcap}(v)$ be the downstream capacitance of v, which we define recursively by

$$\mathrm{downcap}(v) = \begin{cases} \mathrm{cap}(v) & \text{if } v \text{ is a sink,} \\ \sum_{w \in \Gamma^+(v)} \mathrm{cap}(v,w) + \mathrm{downcap}(w) & \text{else,} \end{cases}$$

where $\mathrm{cap}(v,w)$ denotes the wire capacitance of the (v,w)-segment.

For an edge (v,w) in Y, we can now define the RC-value of the edge by

$$\mathrm{RC}_Y(v,w) \coloneqq \mathrm{res}(v,w) \cdot \left(\frac{\mathrm{cap}(v,w)}{2} + \mathrm{downcap}(w) \right).$$

We naturally extend this by

$$\mathrm{RC}_Y(v,w) \coloneqq \sum_{(p,q) \in E(P_{[v,w]})} \mathrm{RC}_Y(p,q),$$

where $P_{[v,w]}$ denotes the unique path in Y from a node v to some w in Y_v. The wire delay between nodes v and w according to the Elmore model becomes

$$\text{wire-delay}_Y(v,w) \coloneqq \ln 2 \cdot \mathrm{RC}_Y(v,w).$$

For the slew degradation, we will use Bakoglu's metric [Bak90]

$$\text{slew-deg}_Y(v,w) \coloneqq \ln 9 \cdot \mathrm{RC}_Y(v,w),$$

and define the out-slew at w, given an in-slew at v, as in [Kas+04]:

$$\text{out-slew}_Y(v, w, \textit{in-slew}) := \sqrt{\textit{in-slew}^2 + (\text{slew-deg}_Y(v, w))^2}.$$

For a gate g, we will assume a resistance $res(g)$ to be given. Then the delay through g is defined by

$$\text{gate-delay}(g) := res(g) \cdot \text{downcap}(g).$$

2.5.2 Linear delay model

An even simpler delay model is the linear delay model. Here, we assume the signal delay to increase linearly in the wire length, while gate delays are being neglected. The signal speed can vary depending on layer and wire type.

Using a linear delay model is justified for estimating delay in unbuffered scenarios, as proper buffering will lead to near-linear delays. As done in [Bar+06], factors describing the delay per length can be estimated by buffering very long wire segments and calculating the average delay per length.

For the purpose of this thesis, we will say that *linear timing data* is a function

$$\textit{lin-delay} : \text{Layers} \times \text{WireTypes} \to \mathbb{R}^+$$

mapping a layer and wire type to $\mathbb{R}^+$, which we will interpret as the *delay per length* of the wire type on the given layer.

Chapter 3

Algorithms for timing-constrained global routing

3.1 Global routing oracle

As outlined in Section 2.4.1, the resource sharing algorithm makes use of an oracle for computing minimum-cost routes for the considered nets. Put formally, when considering routing and timing resources, the oracle needs to solve the following problem.

Routing Problem inside Resource Sharing

Instance: The global routing graph $G = (V, E)$, a net N with specified source r and sinks T, congestion prices $c : E \to \mathbb{R}^+$ and delay weights $w : T \to \mathbb{R}^+$.

Task: Compute a Steiner tree Y for the net N minimizing

$$\sum_{e \in E(Y)} c(e) + \sum_{t \in T} w(t) \cdot \text{delay}_Y(r, t),$$

where $\text{delay}_Y(r, t)$ denotes the delay through Y from the net source r to the sink t.

The exact definition of $\text{delay}_Y(r, -)$ depends on the timing model used. The simplest version to think of would be to set $\text{delay}_Y(r, -) \equiv \text{dist}_Y(r, -)$. Details on timing models are given in Section 2.5.

As already seen in Section 2.2, the oracle considered in this thesis to compute an approximately minimum-cost route for each net consists of two main steps. First, a Steiner tree for the given net is computed in $(\mathbb{R}^2, \ell_1)$. This Steiner tree is called the *topology*. Afterwards, the tree is embedded into the global routing graph by successive Dijkstra searches. This is based on [Hel+18].

3.1.1 Topology generation

During topology generation, a Steiner tree is computed in $(\mathbb{R}^2, \ell_1)$ for the given net. Different possibilities for both optimization objectives and respective algorithms how to compute this Steiner tree are presented in Section 3.2.

Algorithm 1: Topology embedding algorithm [Hel+18]

Input: Global routing graph $G = (V, E)$, source $r \in V$, sinks $T \subset V$, binary arborescence Y with a mapping $p : V(Y) \to V$ such that $p(\text{root}(Y)) = r$ and the leaves of Y are bijectively mapped to T.

Output: An arborescence Y' in G with root r and leaves T.

1 $labels(v) \coloneqq \{(p(v), 0)\}$ if v is a leaf and $labels(v) \coloneqq \emptyset$ else for all $v \in V(Y)$
2 **foreach** $v \in V(Y)$ *in reverse topological order* **do**
3 $targets \coloneqq$ the set of all vertices with the same x and y coordinates as $p(v)$
4 $heap \coloneqq$ empty-heap
5 **foreach** $c \in \Gamma^+(v)$ **do**
6 insert all elements of $labels(c)$ into the $heap$
7 **while** *not all targets are permanently labeled* **do**
8 propagate the minimum-cost label in $heap$
9 $labels(v) \coloneqq \{\ (p(x), \text{cost}(l_1) + \text{cost}(l_2))\ \mid l_1, l_2$ permanent labels at $x \in V$ from paths starting at both children of $v\ \}$
10 back-track the minimum-cost label in $labels(r)$ to create Y'
11 **return** Y'

For the subsequent embedding, it is necessary that the topology fulfills these structural constraints:

- Every vertex has out-degree at most 2 and
- the leaves are exactly the sinks.

Any tree can be transformed into one fulfilling these constraints in linear time without changing the total net length. This is done by adding vertices on top of existing vertices and edges of length 0.

3.1.2 Embedding

After the computation of a 2-dimensional Steiner tree, each edge needs to be embedded into the global routing graph. The embedding needs to decide the exact global routing edges to use and which wire type to use on which edge. Additionally, the final Steiner point locations are decided during the embedding.

The rough outline of the algorithm can be seen in Algorithm 1. In Line 1, we initialize labels of cost 0 for all sinks. Then, we traverse the nodes of Y in reverse topological order. For each $v \in V(Y)$, we do the following. We initialize the targets to be all global routing nodes at the same x and y coordinates as $p(v)$ (Line 3). In particular, the targets can lie on different layers. We also initialize an empty heap and insert the elements of $labels(c)$ for each child c of v into the heap (Lines 4-6). We then propagate the minimum-cost label in the heap along all adjacent edges as long as not all targets are permanently labeled (Lines 7 and 8). Finally, we merge permanent labels coming from different children of v to create $labels(v)$ by setting

$$labels(v) \coloneqq \{\ (p(x), \text{cost}(l_1) + \text{cost}(l_2))\ \mid l_1, l_2 \text{ permanent labels at } x \in V \text{ from paths starting at both children of } v\ \}.$$

When we have reached the source, we retrieve the final result via back-tracking from the minimum-cost label in $labels(r)$ (Lines 10 and 11).

Figure 3.1: Example net with a blockage-unaware topology (left) and a blockage-aware topology (right). The gray shape represents a blockage. When embedding the left tree, the edge going to s_2 will have to be routed around that blockage. The blockage-aware topology prevents this issue.

The crucial parts are traversing the topology in reverse topological order and starting from all found permanent labels in the next Dijkstra searches to not fix final Steiner point positions too early. During labeling, the global routing graph is considered to have parallel edges between adjacent vertices wherever a choice between multiple wire types arises. The cost function used during the Dijkstra searches depends on the current use case. Resource prices for net length and routing congestion are always considered. In BonnRouteBuffer, additionally the prices for timing, power and placement are taken into account.

3.2 Topology algorithms

Many different options arise for computing a "good" topology in global routing. In this section, we will first consider Steiner tree algorithms dealing with several kinds of blockages. After that, we will present Steiner tree algorithms that can consider additional input from arrival time customers or delay prices.

3.2.1 Reachaware topologies

When only routing congestion and net length play a role, the routing problem itself reduces to a Steiner-minimum-tree (SMT) problem. Hence also the topology can be computed using an SMT approximation algorithm.

However, there are further circumstances that are reasonable to consider: When there is a blockage covering all routing layers in a given area, it is beneficial for the topology generation to already take this into account. Otherwise, the embedding might have to route large detours that could have been easily avoided by changing the structure of the Steiner tree. An example for such a situation is shown in Figure 3.1.

Moreover, there are situations in which not all layers, but all layers of a given direction are blocked for routing. This translates to orientation-restricted blockages in topology generation, i.e. blockages that forbid either horizontal or vertical segments through them.

Lastly, there can be placement blockages. Then, routing is allowed, but no repeaters may be inserted there. Connected components of wiring on a placement blockage need to be driven by a single repeater. This leads to an upper bound of capacitance of connected wiring inside the blockage. The upper bound is given by the capacitance limits of the strongest repeater. As capacitance is linear in the wire length, this is modeled by restricting the length of a connected component inside such a blockage.

These considerations give rise to the Reachaware Steiner Tree Problem:

REACHAWARE STEINER TREE PROBLEM

Instance: Sets $\mathcal{R}_l, \mathcal{R}_h, \mathcal{R}_v \subset \mathbb{R}^2$ that are the union of finitely many axis-parallel rectangles, a finite set of terminals $T \subset \mathbb{R}^2$ and $L \in \mathbb{R}^+$.

Task: Compute a shortest *reachaware* Steiner tree for T, i.e. a rectilinear tree Y satisfying that

1. every edge in $E(Y) \cap \mathcal{R}_h^o$ is horizontal,
2. every edge in $E(Y) \cap \mathcal{R}_v^o$ is vertical and
3. every connected component of $E(Y) \cap (\mathcal{R}_l \cup \mathcal{R}_h \cup \mathcal{R}_v)^o$ has total length $\leq L$.

[Bih15], extending upon [HS14], gives a polynomial-time 2-approximation algorithm for the case where no terminals lie inside $(\mathcal{R}_l \cup \mathcal{R}_h \cup \mathcal{R}_v)^o$. This algorithm works by constructing a *visibility graph* and computing a terminal spanning tree in there. In addition to the terminals, the visibility graph contains certain vertices on the border of the given restriction sets.

Under co-supervision of the author of this thesis, an adaptation of this algorithm was developed and implemented computing reachaware shortest-path trees in [Rei23]. In this version of the problem, one of the terminals T is the designated root r and the resulting tree must contain (reachaware) shortest paths from r to all other terminals. This is important for practical application as short paths are necessary for fast signals.

Similarly to [Rao+92], *sink pairs* are chosen iteratively and a common *parent* is inserted. A sink pair is a pair of vertices in the current branching that do not have an incoming edge. A parent of a sink pair is a node lying on shortest paths from the root to both sinks, and among all these, one that is furthest from the root. The sink pair of each iteration is chosen such that the common parent vertex has maximum distance to the root.

This algorithm yields a 2-approximation on instances where the following holds:

a) $T \subset \mathbb{R}^2 \setminus (\mathcal{R}_l \cup \mathcal{R}_h \cup \mathcal{R}_v)^o$,

b) all terminals and blockages lie in the first quadrant,

c) blockages are disjoint rectangles,

d) $\mathcal{R}_l = \emptyset$ and $\mathcal{R}_h = \mathcal{R}_v$.

Here, the last constraint makes sure that area is either not blocked at all or completely blocked.

The algorithm by [Rei23] can be improved further in the following way. In practice, the topology is not required to contain strictly shortest paths. Instead, a small detour (relative or absolute) is often allowed. This is sensible in our use-case, as the subsequent embedding will not adhere exactly to the Steiner point positions in any case. Simply changing the definition of a parent to also allow small detours will not lead to good solutions, though. Instead, the choice of the sink pair and the parent location have to be separated. As previously, the sink pair for which a parent is added is chosen such that the parent location (with the previous definition) is furthest from the root. But instead of placing the parent node at that location, it can be moved to minimize net length instead of maximizing its distance to the root, under the restriction that the detour does not become too large. To be exact, the parent location p is chosen so that it minimizes

$$d(r, p) + d(p, s_1) + d(p, s_2)$$

under all feasible locations for a fixed sink pair s_1, s_2.

Note that this choice coincides with maximizing the distance between parent and root if no detour is allowed:

Proposition 3.1. *Given a graph $G = (V, E)$ with edge lengths $d : E \to \mathbb{R}^+$, a root $r \in V$ and two sinks $s_1, s_2 \in V$, define $P \subseteq V$ to be all nodes lying on a shortest r-s_1-path and a shortest r-s_2-path. Then*

$$\operatorname*{argmin}_{p \in P} \left(d(r,p) + d(p,s_1) + d(p,s_2)\right) = \operatorname*{argmax}_{p \in P} d(r,p).$$

Proof. By the definition of P, we know $d(p, s_i) = d(r, s_i) - d(r, p)$ for $i = 1, 2$ and $p \in P$. Hence

$$\begin{aligned}
& \operatorname*{argmin}_{p \in P} \left(d(r,p) + d(p,s_1) + d(p,s_2)\right) \\
= \; & \operatorname*{argmin}_{p \in P} \left(d(r,p) + (d(r,s_1) - d(r,p)) + (d(r,s_2) - d(r,p))\right) \\
= \; & \operatorname*{argmin}_{p \in P} \left(d(r,s_1) + d(r,s_2) - d(r,p)\right) \\
= \; & \operatorname*{argmax}_{p \in P} d(r,p)
\end{aligned}$$

as claimed. □

Further, the running time of the implementation by [Rei23] can be improved significantly. Instead of computing the parent location of all sink pairs in advance (and maintaining this set when parent nodes are added as terminals), it suffices to compute which parent to consider next. This can be done efficiently by traversing the nodes in descending order of distance to the root, so that we can stop as soon as the considered nodes are closer to the root than the currently best found parent.

3.2.2 Timing-aware topologies

When timing is considered explicitly, two main options are interesting in the context of this thesis. One possibility is to use a bi-criteria algorithm such as described in [KRY95] or [HR13]. Such an algorithm approximately minimizes total tree length under the restriction that source-to-sink distances are bounded.

Instead of considering bounds, delay weights can be given on the sinks. Then, it is the objective to minimize a weighted sum of net length and source-to-sink distances. The resulting problem is called the cost-distance Steiner tree problem, which was first introduced in [MMP08]. This version more closely resembles the task given in the global routing oracle, as delay weights are given by the timing prices.

Other important algorithms for industrial application can be found in [Bar+06], [CTY17], [Alp+95] and [Alp+18].

Bi-criteria

A simple version of the topology problem with delay bounds can be defined as follows.

SHALLOW-LIGHT STEINER TREE PROBLEM

Instance: A metric space (M, d) (usually $(\mathbb{R}^2, \ell_1)$), a root $r \in M$ and finitely many terminals $T \subset M$ together with delay bounds $rat : T \to \mathbb{R}^+$.

Task: Compute a Steiner tree Y rooted at r with leaves T satisfying

$$\sum_{e \in P_{[r,t]}} d(e) \leq rat(t) \quad \forall t \in T,$$

where $P_{[r,t]}$ is the (unique) r-t-path in Y, such that Y minimizes $d(Y)$.

Here, we implicitly assume an embedding $p : V(Y) \to M$ that maps r and the terminals to themselves. We write $d(e) = d(p(v), p(w))$ for an edge $e = \{v, w\}$.

[KRY95] consider the case that $M = \{r\} \cup T$. For a given $\varepsilon > 0$, they compute a solution tree of length at most $(1 + \frac{2}{\varepsilon}) \cdot MST$, such that every r-t-path has length at most $(1+\varepsilon) \cdot \text{dist}_{(M,d)}(r, t)$. Here, MST denotes the length of a minimum spanning tree.

An extension of this problem is considered in [HR13], where additionally to the root-to-terminal distance, bifurcation delays are taken into account. When requiring that the tree is binary and assuming a bifurcation delay of $b > 0$, the feasibility constraint becomes

$$delay_Y(t) \coloneqq \sum_{e \in P_{[r,t]}} d(e) + b \cdot \text{bif}_Y(t) \leq rat(t) \quad \forall t \in T,$$

where $\text{bif}_Y(t)$ denotes the number of bifurcations on the r-t-path in Y. [HR13] show how to compute a Steiner tree Y of length at most

$$\left(1 + \frac{2}{\varepsilon}\right) \text{length}(Y_0) + \frac{4b \cdot |T|}{\varepsilon},$$

where Y_0 is an initial (short) Steiner tree, such that

$$\min_{t \in T} rat(t) - delay_Y(t) \leq -2b - \varepsilon \cdot \max_{t \in T} rat(t).$$

For well chosen distance/delay bounds, very good topologies can be found in practice with this approach. A feasible choice for the distance bounds is to use the already computed interval bounds of the arrival time customers at the net's sinks. These interval bounds can be multiplied by a factor depending on the timing criticality of the net. The bifurcation delay can be used to simulate the fact that side-branches add to the downstream capacitance and hence increase the actual delay after buffering.

While this approach does perform well in practice, it cannot distinguish between the timing criticalities of individual sinks of the same net due to the choice of delay bounds. Either the entire net is considered critical and all paths are kept short, or all source-to-sink paths are allowed to contain more detour. Additionally, the resource prices for sink delay are not considered at all in this formulation. One way to include information about individual timing criticalities would be to replace the interval bounds of the arrival time customers by the actual chosen arrival times, i.e. set $rat(t) = at(t) - at(r)$, where $at(t)$ and $at(r)$ are the current (fractional) solutions of the respective arrival time customers. However, this would be highly heuristic and might lead to negative delay budgets. To eliminate these problems, the delay prices given by the resource sharing algorithm have to be employed directly.

Cost-distance

When considering delay weights and equating delay and distance, the arising problem is exactly the uniform cost-distance Steiner tree problem (see [MMP08] and [KH20]):

UNIFORM COST-DISTANCE STEINER TREE PROBLEM

Instance: A graph $G = (V, E)$, a source $r \in V$ and sinks $T \subseteq V$, edge costs $c : E \to \mathbb{R}^+$ and delay weights $w : T \to \mathbb{R}^+$.

Task: Compute a Steiner tree Y for $\{r\} \cup T$ minimizing

$$\sum_{e \in E(Y)} c(e) + \sum_{t \in T} w(t) \cdot \mathrm{dist}_{(Y,c)}(r, t),$$

where $\mathrm{dist}_{(Y,c)}(r, t)$ denotes the length of the unique r-t-path in Y with respect to c.

This problem can be solved up to a factor of 2.05 within $\mathcal{O}(\Lambda + |T|)$ time, where Λ denotes the time for computing an approximate shortest Steiner tree, see Section 3.3.

In practice, edge *cost* as in the first summand and edge *delay* as in the second summand of the objective function need not coincide. An alternative router solving the non-uniform cost-distance Steiner tree problem, where these two can be unrelated, is presented in Section 3.4. When they only differ by a constant factor, the ratio can be incorporated into the delay weights to remain with the original uniform problem.

Balancing edge costs and delays this way is difficult to do well in practice. When topologies are computed using the uniform cost-distance algorithm from Section 3.3 in BONNROUTEBUFFER, the edge costs are taken to be the average edge cost of a previously embedded solution, while edge delays are given by the linear timing data.

3.3 Tighter approximation for the uniform cost-distance Steiner tree problem

This section is joint work with Stephan Held and Yannik Spitzley. The results are already published in [FHS23]. They improve upon preliminary results in [HS22]. We jointly developed the cutting criterion (Algorithm 3). The author of this thesis contributed in simplifying the analysis compared to [HS22] and the tightness proof. The latter was not yet a part of [HS22].

3.3.1 Main theorems

Theorem 3.2. *The* UNIFORM COST-DISTANCE STEINER TREE PROBLEM *can be approximated in polynomial time with an approximation factor of*

$$\beta + \frac{\beta}{\sqrt{\beta^2 + 1} + \beta - 1},$$

where $\beta \geq 1$ is the approximation guarantee for the minimum-length Steiner tree problem.

With the best known approximation factor for the minimum Steiner tree problem $\beta = \ln(4) + \epsilon$ [Byr+13; TZ22], this results in an approximation factor <2.05 and for $\beta = 1$ this gives the factor $1 + \frac{1}{\sqrt{2}} < 1.71$, clearly improving upon the previously best

factors 2.39 and 2.0 in [KH20]. The polynomial-time approximation scheme by [Aro98] allows choosing β arbitrarily close to one in the Euclidean and the Manhattan planes. However, general metric spaces do not allow $\beta \leq \frac{96}{95}$ unless P = NP [CC08].

Assuming an ideal Steiner tree approximation factor of $\beta = 1$, our new approximation factor is tight with respect to the lower bound $C_{SMT}(T \cup \{r\}) + D(T, r, w)$, where $C_{SMT}(T \cup \{r\})$ is the connection cost of a minimum-length Steiner tree for $T \cup \{r\}$, i.e. a Steiner tree Y for $T \cup \{r\}$ minimizing $\sum_{e \in E(Y)} c(e)$, and $D(T, r, w) \coloneqq \sum_{t \in T} w(t)c(r, t)$ is the sum of weighted root-sink distances.

Theorem 3.3.

$$\sup_{T,r,w} \frac{\mathrm{OPT}(T, r, w)}{C_{SMT}(T \cup \{r\}) + D(T, r, w)} = 1 + \frac{1}{\sqrt{2}},$$

where $\mathrm{OPT}(T, r, w)$ denotes the optimum solution value for an instance (T, r, w) of the Uniform cost-distance Steiner tree problem. Theorem 3.3 is proven in [FHS23]. The part of proof that does not need Theorem 3.2 was already contained in [Foo20] and is therefore not a part of this thesis.

3.3.2 The $(1 + \beta)$-approximation algorithm

For shorter formulas, we will use the following notation in the remainder of Section 3.3. Let A be an arborescence. By A_v we denote the sub-arborescence rooted at v. Furthermore, $T_A \coloneqq V(A) \cap T$ is the set of terminals in A, $W_A \coloneqq w(T_A)$ is the sum of delay weights in A, $C_A \coloneqq c(E(A))$ is the *connection cost* of A and $D_A \coloneqq D_{T_A} \coloneqq \sum_{t \in T_A} w(t)c(r, t)$ the *minimum possible delay cost* for connecting the sinks in T_A (independent of the structure of A).

Recall that $\beta \geq 1$ is the approximation guarantee for the minimum-length Steiner tree problem. The algorithm in [KH20] is described in Algorithm 2. After orienting its edges, we can consider any solution A as an r-arborescence. We use arborescences instead of trees to simplify the algorithmic notation.

Essential steps for a $1 + \beta$ approximation

We quickly recap the essential steps in the analysis of [KH20], which we will use in our analysis. The cost to connect an arborescence $A' \in \mathcal{A}$ to the root r can be estimated as follows:

Lemma 3.4 ([KH20], Lemma 1)**.** *Let $A' \in \mathcal{A}$ with corresponding terminal set T'. By the choice of the port $t \in T'$, the r-arborescence $(A' + \{r, t\})$ has a total cost at most*

$$C_{A'} + \sum_{e=(x,y) \in E(A')} \frac{2W_{A'_y}(W_{A'} - W_{A'_y})}{W_{A'}} c(e) + \left(1 + \frac{1}{W_{A'}}\right) D_{T'} \tag{3.1}$$

$$\leq \left(1 + \frac{W_{A'}}{2}\right) C_{A'} + \left(1 + \frac{1}{W_{A'}}\right) D_{T'} \tag{3.2}$$

$$\leq (1 + \mu)\, C_{A'} + \left(1 + \frac{1}{\mu}\right) D_{T'}. \tag{3.3}$$

For a proof sketch, see the appendix in [FHS23]. We use the bounds (3.1) and (3.2) that were not stated explicitly in [KH20], Lemma 1. While the bounds (3.1) and (3.2) hold for any (sub-)arborescence A', (3.3) depends on the specific way how $A' \in \mathcal{A}$ was cut off during Step 2 of Algorithm 2.

A similar cost bound can be shown easily for the arborescence A_r containing the root r after Step 2. Summing up the resulting cost bounds and choosing $\mu = \frac{1}{\beta}$, [KH20] obtain the approximation factor $(1 + \beta)$.

Algorithm 2: $(1+\beta)$-approximation algorithm by KH20 using a parameter $\mu > 0$.

Step 1 (initial arborescence):
First, compute a β-approximate minimum cost Steiner r-arborescence A_0 for $T \cup \{r\}$ with outdegree 0 at all sinks in T and outdegree 2 at all Steiner vertices in $V(A_0) \setminus (T \cup \{r\})$.

Step 2 (split into branching):
Traverse A_0 bottom-up. For each traversed edge $(x, y) \in E(A_0)$, if $W_{(A_0)_y} > \mu$, remove the edge (x, y) creating a new arborescence $(A_0)_y$ in the branching.

Let $\mathcal{A}$ denote the set of all arborescences that were cut off from A_0 this way.

Step 3 (reconnect arborescences):
Reconnect each sub-arborescence A' that was cut off in Step 2 as follows: Select a vertex $t \in T' := T_{A'}$ that minimizes the cost for serving the sinks in T' through the r-arborescence $A' + (r, t)$, i.e. select a vertex $t \in T'$ as a *port* for T' that minimizes

$$c(r,t) + C_{A'} + \sum_{t' \in T'} w(t') \cdot (c(r,t) + c(E(A'_{[t,t']}))).$$

Let $t_1, \ldots, t_{|\mathcal{A}|} \in T$ be the set of selected port vertices. Return the union of the final branching and the port connections $A_0 + \{(r, t_i) : i \in \{1, \ldots, |\mathcal{A}|\}\}$.

3.3.3 Improving the approximation ratio

Algorithm 2 suffers from the following weakness indicated in Figure 3.2. Assume that after splitting we are given a sub-arborescence $A' \in \mathcal{A}$ with a high delay weight $W_{A'}$, a high connection cost $C_{A'}$, but a low minimum possible delay cost $D_{A'}$, e.g. as shown in Figure 3.2b. Then Algorithm 2 would retain the high delay cost. Instead, it would be better to split the arborescence further to achieve a lower delay cost as in Figure 3.2c.

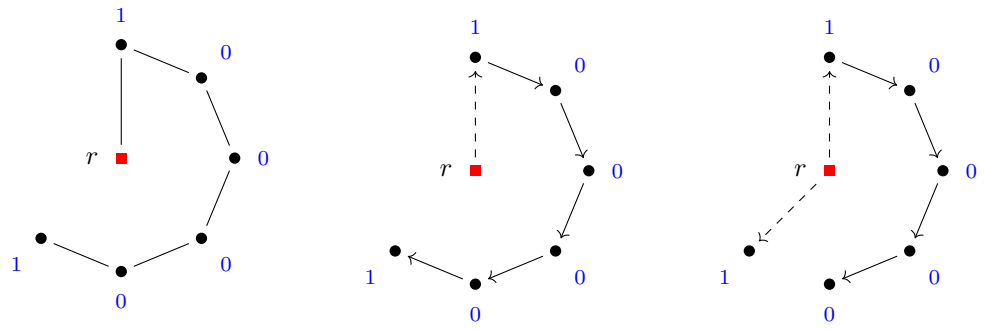
In this section, we propose a refined splitting criterion that provides a better approximation ratio. Instead of using a fixed threshold μ, we allow to split off sub-arborescences earlier if their expected reconnection cost (3.1) is sufficiently cheap. The precise criterion is specified in (3.4) (inside Algorithm 3). Observe that (3.4) provides cheaper solutions than (3.3), as one occurrence of μ is replaced by $\frac{\mu}{2}$.

Then we show in Lemma 3.8 that every sub-arborescence of the remaining root component has delay weight at most μ. This allows us to prove a similar improved cost bound for the root component in Lemma 3.9.

Finally, we simply combine all sub-arborescences and choose μ to prove Theorem 3.2.

Improving the splitting routine

Algorithm 3 shows our improved splitting step, which cuts off a sub-arborescence if we can reconnect it cheaply, i.e. if (3.4) holds. With Lemma 3.4 we immediately get the following result for the cut-off sub-arborescences:



(a) Minimum-length Steiner tree

(b) Cost: $6 + (1 + 6) = 13$

(c) Cost: $6 + (1 + 1) = 8$

Figure 3.2: Weakness of Algorithm 2: (M, c) is induced by a complete graph with seven vertices and unit weights. Delay weights are indicated by the blue node labels and $\mu = 1$. Algorithm 2 might start with the minimum-length Steiner tree on the left. Then the algorithm will cut the edge incident to r and reconnect the sub-arborescence resulting possibly in the solution in the middle. On the right the result from our improved algorithm is shown.

Algorithm 3: Modifying Step 2 of Algorithm 2

Step 2 (split into branching):

Traverse A_0 bottom-up. For each traversed edge $(v, z) \in E(A_0)$ consider $A_z \coloneqq (A_0)_z$: If $W_{A_z} > 0$ and

$$\sum_{e=(p,q)\in E(A_z)} \frac{2W_{(A_z)_q}(W_{A_z} - W_{(A_z)_q})}{W_{A_z}} c(e) + \frac{D_{A_z}}{W_{A_z}} \leq \frac{\mu}{2}\left(C_{A_z} + c(v, z)\right) + \frac{D_{A_z}}{\mu}, \tag{3.4}$$

remove (v, z) creating a new arborescence A_z.

Lemma 3.5. *Let $A' \in \mathcal{A}$ be an arborescence that was cut off in Algorithm 3 and let $e_{A'}$ be the incoming edge in the root of the arborescence A' which was deleted during this step. Then the corresponding terminals in $T_{A'}$ can be connected to the root r with total cost at most*

$$\left(1 + \frac{\mu}{2}\right)\left(C_{A'} + c(e_{A'})\right) + \left(1 + \frac{1}{\mu}\right) D_{A'}.$$

□

After the original Step 2 of Algorithm 2, it is clear that for all edges $(r, x) \in \delta^+_{A_0}(r)$ of the root component the total delay weight $W_{(A_0)_x}$ is at most μ. We show that this also holds after the modified Step 2 in Algorithm 3. However, the analysis is more complicated and uses the following two functions.

Definition 3.6. Let $\mu > 0$ and $X^\mu \coloneqq \{(a, b, c) \in (\mu, 2\mu) \times (0, \mu)^2 : c \leq a - b < \mu\}$. We define the functions $f, g \colon X^\mu \to \mathbb{R}$ as

$$f(a, b, c) \coloneqq \frac{2(a - c)c}{a} - \frac{\mu}{2} + \left(\frac{1}{a} - \frac{1}{\mu}\right) \cdot \frac{1}{\frac{1}{a-b} - \frac{1}{\mu}} \cdot \left(\frac{\mu}{2} - \frac{2((a - b) - c)c}{a - b}\right)$$

$$g(a, b, c) \coloneqq \frac{2(a - c)c}{a} - \frac{\mu}{2} + \left(\frac{1}{a} - \frac{1}{\mu}\right) \cdot \frac{1}{\frac{1}{a-b} - \frac{1}{\mu}} \cdot \frac{\mu}{2}.$$

Lemma 3.7. *For all $(a, b, c) \in X^\mu$, $f(a, b, c) \leq 0$ and $g(a, b, c) \leq 0$.*

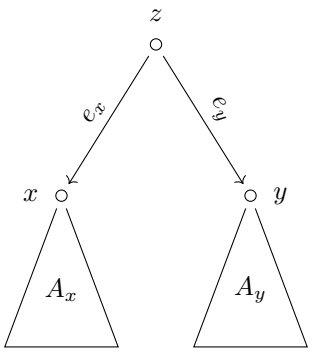


Figure 3.3: Setting in the proof of Lemma 3.8 if z is a Steiner vertex (Case 2).

A proof of Lemma 3.7 based on algebraic transformations can be found in the appendix of [FHS23].

Lemma 3.8. *After cutting off sub-arborescences with Algorithm 3, every child $x \in \Gamma^+_{A_r}(r)$ of r in the remaining root component $A_r := (A_0)_r$ satisfies $W_{(A_r)_x} \leq \mu$.*

Proof. Assume the opposite would be true. Let z be a vertex in $A_r - r$ such that the weight of the sub-arborescence $A_z := (A_r)_z$ exceeds μ and the weight of every child arborescence $(A_z)_x$ is at most μ for all edges $(z, x) \in \delta^+_{A_z}(z)$. We distinguish two cases:

Case 1: z is a terminal: Then z is also a leaf and the left-hand side of (3.4) simplifies to

$$\frac{1}{W_{A_z}} D_{A_z} \leq \frac{1}{\mu} D_{A_z}$$

since A_z does not contain any edges. But then A_z would have been cut-off in Step 2, a contradiction.

Case 2: z is a Steiner vertex: Then z has two outgoing edges $e_x := (z, x), e_y := (z, y) \in \delta^+_{A_z}(z)$ as shown in Figure 3.3. A single outgoing edge would contradict the choice of z. With $A_x := (A_z)_x$ or $A_y := (A_z)_y$ this implies $0 < W_{A_x}, W_{A_y} \leq \mu$. If $W_{A_x} = \mu$, Lemma 3.4, (3.2) shows that A_x satisfied the bound (3.4) when it was considered in Step 2 and would have been cut off. Analogously, $W_{A_y} \neq \mu$. Thus, $W_{A_x}, W_{A_y} < \mu$. Since (3.4) does not hold for A_x, we get (by transforming its negation)

$$\underbrace{\left(\frac{1}{W_{A_x}} - \frac{1}{\mu}\right)}_{>0} D_{A_x} > \sum_{e=(u,v)\in E(A_x)} \left(\frac{\mu}{2} - \frac{2(W_{A_x} - W_{(A_x)_v})W_{(A_x)_v}}{W_{A_x}}\right) c(e) + \frac{\mu}{2} c(e_x).$$

Combining this with the analogue inequality for A_y and using $D_{A_z} = D_{A_x} + D_{A_y}$, we

get

$$\begin{aligned}
&\underbrace{\left(\frac{1}{W_{A_z}}-\frac{1}{\mu}\right)}_{<0} D_{A_z} \\
&< \left(\frac{1}{W_{A_z}}-\frac{1}{\mu}\right)\Bigg(\sum_{e=(u,v)\in E(A_x)} \frac{1}{\frac{1}{W_{A_x}}-\frac{1}{\mu}}\left(\frac{\mu}{2}-\frac{2(W_{A_x}-W_{(A_x)_v})W_{(A_x)_v}}{W_{A_x}}\right)c(e) \\
&\qquad + \frac{\frac{\mu}{2}}{\frac{1}{W_{A_x}}-\frac{1}{\mu}}c(e_x) \\
&\qquad + \sum_{e=(u,v)\in E(A_y)} \frac{1}{\frac{1}{W_{A_y}}-\frac{1}{\mu}}\left(\frac{\mu}{2}-\frac{2(W_{A_y}-W_{(A_y)_v})W_{(A_y)_v}}{W_{A_y}}\right)c(e) \\
&\qquad + \frac{\frac{\mu}{2}}{\frac{1}{W_{A_y}}-\frac{1}{\mu}}c(e_y)\Bigg).
\end{aligned}$$

This inequality together with

$$\begin{aligned}
&\sum_{e=(u,v)\in E(A_z)} \left(\frac{2(W_{A_z}-W_{(A_z)_v})W_{(A_z)_v}}{W_{A_z}}-\frac{\mu}{2}\right)c(e) \\
&= \sum_{e=(u,v)\in E(A_x)} \left(\frac{2(W_{A_z}-W_{(A_z)_v})W_{(A_z)_v}}{W_{A_z}}-\frac{\mu}{2}\right)c(e) \\
&\quad + \left(\frac{2(W_{A_z}-W_{(A_z)_x})W_{(A_z)_x}}{W_{A_z}}-\frac{\mu}{2}\right)c(e_x) \\
&\quad + \sum_{e=(u,v)\in E(A_y)} \left(\frac{2(W_{A_z}-W_{(A_z)_v})W_{(A_z)_v}}{W_{A_z}}-\frac{\mu}{2}\right)c(e) \\
&\quad + \left(\frac{2(W_{A_z}-W_{(A_z)_y})W_{(A_z)_y}}{W_{A_z}}-\frac{\mu}{2}\right)c(e_y)
\end{aligned}$$

yields

$$\begin{aligned}
&\sum_{e=(u,v)\in E(A_z)} \left(\frac{2(W_{A_z}-W_{(A_z)_v})W_{(A_z)_v}}{W_{A_z}}-\frac{\mu}{2}\right)c(e) + \left(\frac{1}{W_{A_z}}-\frac{1}{\mu}\right)D_{A_z} \\
&< \sum_{e=(u,v)\in E(A_x)} f(W_{A_z},W_{A_y},W_{(A_z)_v})c(e) + \sum_{e=(u,v)\in E(A_y)} f(W_{A_z},W_{A_x},W_{(A_z)_v})c(e) \\
&\quad + g(W_{A_z},W_{A_y},W_{(A_z)_v})c(e_x) + g(W_{A_z},W_{A_x},W_{(A_z)_v})c(e_y).
\end{aligned}$$

By Lemma 3.7 and $0 < W_{A_x}, W_{A_y} < \mu$, the last term is non-positive. Therefore A_z satisfied the bound (3.4) when it was considered in Step 2 and would have been cut off, a contradiction. □

In [KH20] the final root arborescence A_r, which was not cut off in Step 2 of Algorithm 2, was kept unaltered. Using Lemma 3.8, we show how to connect it in a better way.

Lemma 3.9. *Let A_r be the sub-arborescence of A_0 rooted at r after the modified Step 2 of Algorithm 2. The terminal set T_{A_r} can be connected to the root r with total cost at most*

$$\left(1+\frac{\mu}{2}\right)C_{A_r} + \left(1+\frac{1}{\mu}\right)D_{A_r}.$$

Proof. Let $(r,x) \in \delta^+_{A_r}(r)$ be arbitrary and A_x the arborescence of $A_r - r$ rooted at x. We show that the terminal set T_{A_x} can be connected to the root r with total cost at most

$$\left(1+\frac{\mu}{2}\right)(C_{A_x}+c(r,x))+\left(1+\frac{1}{\mu}\right)D_{A_x}.$$

Adding this cost for all edges in $\delta^+_{A_r}(r)$, we obtain the claim. We distinguish between two cases:

Case 1:

$$W_{A_x}(C_{A_x}+c(r,x)) \leq \frac{\mu}{2}(C_{A_x}+c(r,x))+\frac{1}{\mu}D_{A_x}.$$

By keeping the arborescence A_x connected through (r,x), the connection cost is $C_{A_x}+c(r,x)$. In particular, for each terminal $t \in T_{A_x}$, the r-t-path in $A_x+(r,x)$ has a length of at most $C_{A_x}+c(r,x)$. We therefore obtain a total cost of at most

$$(1+W_{A_x})(C_{A_x}+c(r,x)) \leq \left(1+\frac{\mu}{2}\right)(C_{A_x}+c(r,x))+\frac{1}{\mu}D_{A_x}.$$

Case 2:

$$W_{A_x}(C_{A_x}+c(r,x)) > \frac{\mu}{2}(C_{A_x}+c(r,x))+\frac{1}{\mu}D_{A_x}. \tag{3.5}$$

Therefore we have $W_{A_x} > 0$ and obtain from (3.5) an upper bound on the minimum possible delay cost of A_x

$$D_{A_x} < \left(W_{A_x}-\frac{\mu}{2}\right)\mu(C_{A_x}+c(r,x)). \tag{3.6}$$

We remove the edge (r,x) and connect the arborescence A_x to the root r. By Lemma 3.8, $W_{A_x} \leq \mu$. As in Lemma 3.4 we obtain total cost of at most

$$\begin{aligned}
&\left(1+\frac{W_{A_x}}{2}\right)C_{A_x}+\left(1+\frac{1}{W_{A_x}}\right)D_{A_x}\\
&\quad=\left(1+\frac{W_{A_x}}{2}\right)C_{A_x}+\left(1+\frac{1}{\mu}\right)D_{A_x}+\underbrace{\frac{\mu-W_{A_x}}{\mu W_{A_x}}}_{\geq 0}D_{A_x}\\
&\quad\overset{(3.6)}{\leq}\left(1+\frac{W_{A_x}}{2}\right)C_{A_x}+\left(1+\frac{1}{\mu}\right)D_{A_x}+\frac{\mu-W_{A_x}}{\mu W_{A_x}}\left(W_{A_x}-\frac{\mu}{2}\right)\mu(C_{A_x}+c(r,x))\\
&\quad\leq\left(1-\frac{W_{A_x}}{2}+\frac{3}{2}\mu-\frac{\mu^2}{2W_{A_x}}\right)(C_{A_x}+c(r,x))+\left(1+\frac{1}{\mu}\right)D_{A_x}.
\end{aligned}$$

With the following estimation we obtain the claimed bound

$$-\frac{W_{A_x}}{2}+\frac{3}{2}\mu-\frac{\mu^2}{2W_{A_x}}=\frac{\mu}{2}-\frac{1}{2}\left(\sqrt{W_{A_x}}-\frac{\mu}{\sqrt{W_{A_x}}}\right)^2\leq\frac{\mu}{2}.$$

□

Theorem 3.10. *Algorithm 3 and the reconnect in Step 3 as well as of the root component can be implemented to run in time $\mathcal{O}(|T|)$.*

Proof. A naive implementation would immediately result in a quadratic running time. We can achieve a linear running time by computing all relevant information incrementally in constant time per node during the bottom-up traversal. Details can be found in [FHS23]. □

Proving Theorem 3.2

We start by analyzing the combination of all sub-arborescences.

Theorem 3.11. *Given an instance (T, r, w) of the* Uniform cost-distance Steiner tree problem*, we can compute in $\mathcal{O}(\Lambda + |T|)$ time a Steiner tree with objective value at most*

$$\left(1 + \frac{\mu}{2}\right) C + \left(1 + \frac{1}{\mu}\right) D, \tag{3.7}$$

where C is the cost of a β-approximate minimum-length Steiner tree and $D := D(T, r, w)$. Here, Λ is the running time for computing a β-approximate minimum Steiner tree for $T \cup \{r\}$.

Proof. We run Algorithm 2 with two modifications:

1. The cut-off routine (Step 2) is modified according to Algorithm 3.
2. The arborescence A_r containing the root r after Step 2 is reconnected to the root r according to Lemma 3.9.

The total cost of the computed solution is upper bounded by the sum of the cost bounds for these r-arborescences, which is (3.7). For the running time analysis, we consider the individual steps of the algorithm:

In Step 1, a β-approximate minimum Steiner tree for $T \cup \{r\}$ is computed in time $\mathcal{O}(\Lambda)$ and transformed into the arborescence A_0 obeying the degree constraints in linear time as in [KH20]. The linear running time of Step 2 and Step 3 follows from Theorem 3.10. □

Finally, we choose the threshold μ based on the quantities C and D to prove Theorem 3.2:

Proof. (of Theorem 3.2) We make the following modification of the algorithm in Theorem 3.11:

If $C = c(E(A_0)) = 0$, each r-t-path, $t \in T$, has length 0 in A_0. So this is already an optimal solution and we just return A_0.

Otherwise, set $\mu := \sqrt{\frac{2D}{C}}$ and the algorithm from Theorem 3.11 provides us with a solution with total cost at most

$$C + D + \sqrt{2}\sqrt{CD} \leq \beta C_{SMT}(T \cup \{r\}) + D + \sqrt{2}\sqrt{\beta C_{SMT}(T \cup \{r\}) \cdot D}.$$

We divide this by the lower bound $C_{SMT}(T \cup \{r\}) + D$. Now, the approximation factor is at most the maximum of the function $h \colon \mathbb{R}_{>0} \times \mathbb{R}_{\geq 0} \to \mathbb{R}$ given by

$$h(x, y) := \frac{\beta x + y + \sqrt{2}\sqrt{\beta x y}}{x + y}.$$

By our assertion, $C_{SMT}(T \cup \{r\}) \geq \frac{C}{\beta} > 0$. In the appendix of [FHS23] we prove for $x + y > 0$ using algebraic reformulations

$$h(x, y) \leq \beta + \frac{\beta}{\sqrt{\beta^2 + 1} + \beta - 1},$$

proving the claimed approximation ratio. □

Using Theorem 3.2 we obtain the approximation factors shown in Table 3.1 (rounded to five decimal digits) for some interesting values of β.

Parameter β	1	$\ln(4)+\epsilon$	$\frac{3}{2}$	2
Algorithm 2, [KH20]	2.00000	2.38630	2.50000	3.00000
Theorem 3.2	1.70711	2.04782	2.15139	2.61804

Table 3.1: Comparison of approximation factors for the Uniform cost-distance Steiner tree problem with different approximation factors β for the minimum-length Steiner tree problem.

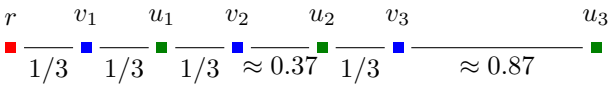


Figure 3.4: Example of the instance demonstrating the tightness of the analysis for $k = 3$.

Tightness of the analysis

We present a family of instances where our algorithm returns solutions that are asymptotically a factor $1+\frac{1}{\sqrt{2}}$ above the optimum, even when starting with a minimum-length Steiner tree. For $k \in \mathbb{N}$, we are given a root r and $2k$ terminals $T = \{u_i, v_i \mid 1 \le i \le k\}$ that are placed on a single line in the order $r < v_1 < u_1 < \cdots < v_k < u_k$ as shown in Figure 3.4 for $k = 3$.

We specify the distances between adjacent terminals. Let $u_0 := r$. The distances are $c(u_{i-1}, v_i) := \frac{1}{k}$ for $1 \le i \le k$, $c(v_1, u_1) = \frac{1}{k}$, and

$$c(v_i, u_i) := \frac{i-\sqrt{2}}{\sqrt{2}k} + \frac{1}{\sqrt{2}} \sum_{j=1}^{i-1} c(v_j, u_j) \quad \text{for } 2 \le i \le k. \tag{3.8}$$

Vertex weights are $w(v_1) = 2$, $w(v_i) = \frac{1}{\sqrt{2}}$ for $2 \le i \le k$, and $w(u_i) = 0$ for $1 \le i \le k$.

Observe that the length of a minimum Steiner tree is

$$C_{SMT} = \sum_{i=1}^{k} (c(u_{i-1}, v_i) + c(v_i, u_i)) = c(u_0, v_1) + c(v_1, u_1) + \sum_{i=2}^{k} (c(u_{i-1}, v_i) + c(v_i, u_i)) \tag{3.9}$$

$$= \frac{2}{k} + \sum_{i=2}^{k} \left(\frac{1}{k} + \frac{i-\sqrt{2}}{\sqrt{2}k} + \frac{1}{\sqrt{2}} \sum_{j=1}^{i-1} c(v_j, u_j) \right) \tag{3.10}$$

$$= \frac{2}{k} + \sum_{i=2}^{k} \frac{1}{\sqrt{2}} \cdot \left(\frac{i}{k} + \sum_{j=1}^{i-1} c(v_j, u_j) \right) \tag{3.11}$$

$$= w(v_1) \cdot c(u_0, v_1) + \sum_{i=2}^{k} w(v_i) \cdot \left(c(u_0, v_1) + \sum_{j=1}^{i-1} (c(u_j, v_{j+1}) + c(v_j, u_j)) \right) \tag{3.12}$$

$$= D(T, r, w). \tag{3.13}$$

In this tree, which is actually a path, every terminal has the minimum possible distance from r. Thus, it is an optimum solution of the uniform cost-distance Steiner tree problem with value $2 \cdot C_{SMT}$.

According to the proof of Theorem 3.2, the algorithm chooses $\mu = \sqrt{\frac{2D(T,r,w)}{C_{SMT}}} = \sqrt{2}$. Thus, edges entering some u_i ($i \in [k]$) will never be deleted, as $w(u_i) = 0$. Now inductively, for each edge entering a vertex v_i ($i = 2, \ldots, k$) in bottom-up order, the left and right side of the deletion criterion (3.4) are both identical to $\frac{i}{k} + \sum_{j=1}^{i-1} c(v_j, u_j)$. Thus the edge (u_{i-1}, v_i) will be deleted. To see this, observe that the first summand of the left side is zero as $w(u_i) = 0$, and its second summand reduces to the length of the r-v_i path. The right side is

$$\begin{aligned}
&\frac{\mu}{2}\left(C_{A_{v_i}} + c(u_{i-1}, v_i)\right) + \frac{D_{A_{v_i}}}{\mu} \\
=&\frac{1}{\sqrt{2}}\left(c(v_i, u_i) + c(u_{i-1}, v_i)\right) + \frac{1}{\sqrt{2}} \cdot \frac{1}{\sqrt{2}} \left(c(u_{i-1}, v_i) + \sum_{j=1}^{i-1} \left(c(u_{j-1}, v_j) + c(v_j, u_j)\right)\right) \\
=&\frac{1}{\sqrt{2}} \left(\frac{i - \sqrt{2}}{\sqrt{2}k} + \frac{1}{\sqrt{2}} \sum_{j=1}^{i-1} c(v_j, u_j) + \frac{1}{k} + \frac{i}{k\sqrt{2}} + \frac{1}{\sqrt{2}} \sum_{j=1}^{i-1} c(v_j, u_j) \right) \\
=&\frac{i}{k} + \sum_{j=1}^{i-1} c(v_j, u_j).
\end{aligned}$$

In a similar computation we see that the deletion criterion also holds for the case $i = 1$, which can be omitted as the component is reconnected with the deleted edge $\{u_0, v_1\}$ and therefore not changing the result. Thus, the algorithm will remove all edges $\{u_{i-1}, v_i\}$ ($i \in \{1, \ldots, k\}$). The cost of the resulting solution is the sum of C_{SMT}, $D(T, r, w)$ ($= C_{SMT}$) and the additional connection cost for replacing the edges (u_{i-1}, v_i) by r-v_i-paths:

$$2 \cdot C_{SMT} + \sum_{i=2}^{k} \sum_{j=1}^{i-1} (c(u_{j-1}, v_j) + c(v_j, u_j)).$$

The deviation factor from the optimum solution is

$$\begin{aligned}
&\frac{2 \cdot C_{SMT} + \sum_{i=2}^{k} \sum_{j=1}^{i-1} (c(u_{j-1}, v_j) + c(v_j, u_j))}{2 \cdot C_{SMT}} \\
=\ & 1 + \frac{\sum_{i=2}^{k} \sum_{j=1}^{i-1} (c(u_{j-1}, v_j) + c(v_j, u_j))}{2\left(\frac{2}{k} + \sum_{i=2}^{k} \frac{1}{\sqrt{2}} \cdot \left(\frac{i}{k} + \sum_{j=1}^{i-1} c(v_j, u_j)\right)\right)} \\
=\ & 1 + \frac{\sum_{i=2}^{k} \sum_{j=1}^{i-1} (c(u_{j-1}, v_j) + c(v_j, u_j))}{2\left(\frac{2}{k} + \frac{1}{\sqrt{2}} + \frac{1}{\sqrt{2}} \sum_{i=2}^{k} \sum_{j=1}^{i-1} (c(u_{j-1}, v_j) + c(v_j, u_j))\right)} \xrightarrow{k \to \infty} 1 + \frac{1}{\sqrt{2}},
\end{aligned}$$

where we substituted C_{SMT} by (3.11) in the first equation and used

$$\sum_{i=2}^{k} \sum_{j=1}^{i-1} (c(u_{j-1}, v_j) + c(v_j, u_j)) \geq \sum_{i=2}^{k} \frac{i-1}{k} = \frac{1}{k} \sum_{i=1}^{k-1} i = \frac{k-1}{2} \xrightarrow{k \to \infty} \infty.$$

3.4 Cost-distance router

The more general version of the cost-distance problem discussed in the previous section can be defined as follows, as first seen in [MMP08].

General cost-distance problem

Instance: A graph $G = (V, E)$, a source $r \in V$ and sinks $T \subseteq V$, edge costs $c : E \to \mathbb{R}^+$, edge lengths $d : E \to \mathbb{R}^+$ and delay weights $w : T \to \mathbb{R}^+$.

Task: Compute a Steiner tree Y for $\{r\} \cup T$ minimizing

$$\sum_{e \in E(Y)} c(e) + \sum_{t \in T} w(t) \cdot \mathrm{dist}_{(Y,d)}(r, t),$$

where $\mathrm{dist}_{(Y,d)}(r, t)$ denotes the length of the unique r-t-path in Y with respect to d.

[MMP08] give an efficient $\mathcal{O}(\log |T|)$-algorithm for this problem. Note that there is no approximation better than $\Omega(\log \log |T|)$ (unless NP $\subset$ DTIME($|T|^{\mathcal{O} \log \log \log |T|}$)), as shown by [Chu+08].

This problem can be used to model the entire routing oracle problem instead of just the topology computation. For that, define the edge costs to contain costs for net length, routing congestion and potentially estimated power and placement costs. The edge lengths are again given by the delay according to the linear timing data.

Under the author's co-supervision, a version of the approximation algorithm by [MMP08], together with many practical improvements, was implemented by the authors of [Hei21] and [Per23]. In the remainder of this section, we sketch out the original approximation algorithm as well as present the implemented improvements to make it fast and good enough to be used in practice.

3.4.1 Iterative matching algorithm

The algorithm in [MMP08] iteratively computes a cheap matching on the (remaining) sinks and connects matched vertices. After a logarithmic number of iterations, all sinks are connected.

The following distance functions depending on delay weights $w' : T \to \mathbb{R}^+$ are used: For a terminal $u \in T$, define

$$M_{ru}^{(w')}(e) \coloneqq M_{ur}^{(w')}(e) \coloneqq c(e) + w'(u)d(e) \quad \forall e \in E,$$

and for two terminals $u, v \in T$, define

$$M_{uv}^{(w')}(e) \coloneqq c(e) + \frac{2w'(u)w'(v)}{w'(u) + w'(v)} d(e) \quad \forall e \in E.$$

The structure of the algorithm is described in Algorithm 4. We start by setting the set of active terminals to $S \coloneqq \{r\} \cup T$ and initialize $w' \coloneqq w$ and $E' \coloneqq \emptyset$ (Line 1). We then iterate until only one active terminal remains (which will be r). In every iteration, we compute a a cheap matching on S (Line 3). The cost of an edge $\{u, v\}$ between terminals is given by their distance according to $M_{uv}^{(w')}$. For every matching edge $\{u, v\}$, we add the edges of a $M_{uv}^{(w')}$-cheapest path to E' (Line 5). For an edge matching r to some terminal t, we remove t from S. For edges matching two sinks u, v, we choose one

Algorithm 4: Iterative matching algorithm [MMP08]

Input: $(G = (V, E), r, T, c, d, w)$ as in the general cost-distance problem.
Output: A Steiner tree Y for $\{r\} \cup T$.

```
1  S := {r} ∪ T, w' := w, E' := ∅
2  while |S| > 1 do
3      Compute a cheap matching M on S with respect to costs dist_{(G,M_uv^(w'))}(u, v)
         for a matching edge {u, v}
4      foreach {u, v} ∈ M do
5          Add edges of a shortest u-v-path according to the costs to E'
6          if r ∈ {u, v} then
7              Remove the sink matched to r from S
8          else
9              Choose u as center with probability w'(u)/(w'(u)+w'(v)), otherwise choose v
                 as center
10             w'(center) := w'(u) + w'(v)
11             S := S \ {non-center}
12 return a d-shortest-path tree from r to T contained in E'
```

of them at random to be the *center* (Line 9). The probability is proportional to $w'(u)$ respective $w'(v)$. We shift the weight according to w' onto the center and remove the other sink from S (Lines 10 and 11).

To achieve the approximation guarantee of $\mathcal{O}(\log |T|)$ in the claimed run-time, a constant fraction of S has to be matched in Line 3 with cost at most a constant factor higher than a min-cost perfect matching. Here, we will assume a greedy matching of half of the nodes. This then has cost at most half the cost of a min-cost perfect matching.

3.4.2 Practical improvements

A naive implementation of the pseudo-code presented in Algorithm 4 would have significant drawbacks. In particular, computing the terms $M_{uv}^{(w')}(e)$ for all remaining $u, v \in S$ and all $e \in E$ in every iteration would waste much running time. The implementation in the BonnTools as described in [Per23] uses a modified definition of $M_{uv}^{(w')}$ (we will present the modification with γ set to 1 for simplicity):

$$M_{uv}'^{(w')}(e) := \begin{cases} c(e) + w'(x)d(e) & \text{if } \{u, v\} = \{r, x\}, \\ c(e) + \min\{w'(u), w'(v)\}d(e) & \text{else.} \end{cases}$$

With respect to this cost function, shortest paths between sinks in S only depend on the smaller delay weight instead of on both weights. This has the advantage of reducing the number of necessary Dijkstra searches. In the original algorithm, $\mathcal{O}(|S|^2)$ Dijkstra calls are performed in each iteration. Using this cost function, computing $\mathcal{O}(|S|)$ shortest path trees per iteration suffices: For each $u \in S$, compute a shortest path tree using the cost function $M_u'(e) := c(e) + w'(u)d(e)$. Then, for each pair $u, v \in S$, the shortest path found starting at the vertex with lower delay weight between u and v is a shortest path with respect to the modified costs $M_{uv}'^{(w')}$. [Per23] proved that the modified cost function still leads to an $\mathcal{O}(\log |T|)$-approximation.

As we compute a greedy matching of half of the nodes in S, we make use of a further speed-up: Observe that it suffices to know the cheap matching edges that will eventually

be used, instead of computing the cost for all $\mathcal{O}(|S|^2)$ pairs of nodes in S. To exploit this, the Dijkstra searches from the nodes in S are started simultaneously. In every step, the Dijkstra search with the cheapest label is continued. As soon as one path between not yet matched nodes is found, the corresponding Dijkstra searches can be stopped. More details on this algorithm and further practical improvements can be found in [Per23].

Chapter 4

Pangea

This chapter covers BonnPangea, a tool to compute a port assignment. A port assignment is an interface between separate units on a chip. Given a fixed port assignment, placement and routing inside the units are independent of each other. Splitting a chip this way has the advantage of having significantly smaller units in the design flow, leading to faster optimization cycles. During the computation of a port assignment, BonnPangea can already take objectives such as congestion and timing of the separated units into account. It has been used in the design of 2 generations of IBM microprocessors.

In the following sections, we define the setting and problem formulation of pangea and cover previous approaches how to compute port assignments (Sections 4.1 and 4.2). Then we have a closer look into the mathematical side of an occurring sub-problem, namely pangea topology generation (Section 4.3). Afterwards, we cover the standard pangea flow consisting of global routing, track assignment and port cutting (Section 4.4). Then we come to the replay scenario, which is a mode where some or almost all of the interface is already given and needs to be completed (Section 4.5). This need arises for instance when the design is changed slightly, but most of the previously computed interface should be kept the same so as not to introduce instability in the flow. Finally, the biggest part of this chapter covers in detail a completely new mode in BonnPangea called Pangea ReUse, providing the feature of using specified units several times on the chip. To make this possible, it has to be ensured that the interfaces of equivalent units are the exact same. This new mode requires many adaptations to the flow as well as new algorithms, both of which are covered in Section 4.6.

4.1 Preliminaries and problem formulation

We use the following fundamental definitions. A visualization of these can be found in Figure 4.1.

Definition 4.1. A *continent* is a connected union of finitely many axis-parallel closed rectangles inside the chip area. A finite set of disjoint continents is called a *world*. For a given world $\mathcal{C}$, *panthalassa* refers to the parts of the chip area not belonging to any continent:

$$\text{panthalassa} := \text{chip-area} \setminus \bigcup_{C \in \mathcal{C}} C$$

We sometimes also call panthalassa a continent, even though it does not fit the definition. For this chapter, we fix a given world $\mathcal{C}$, and assume the global routing grid

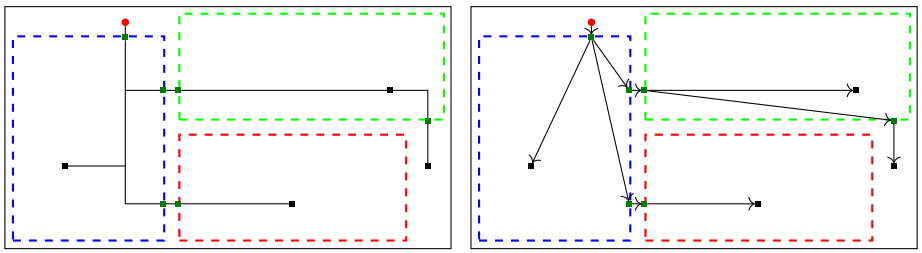


Figure 4.1: Pangea world containing three continents (dashed rectangles) with an example route (solid lines, left) and its corresponding port graph (arrows, right). The red circle represents the net source, black squares are sinks, green squares are ports.

is aligned with the continent borders, i.e. that every global routing tile is contained completely inside a continent.

To specify continent interfaces properly, we need the following notions.

Definition 4.2. A *port* p consists of

- a location $loc(p) \in \bigcup_{C \in \mathcal{C}} \partial C$ and a routing layer $layer(p)$,
- a wire type $wt(p)$, and
- a signal direction $signal\text{-}dir(p) \in \{\text{north}, \text{south}, \text{west}, \text{east}\}$.

We further require that $loc(p)$ is not on a corner point of a continent and that

$$signal\text{-}dir(p) \in \{\text{north}, \text{south}\}$$
$$\iff$$
$$loc(p) \text{ lies on a horizontal segment of } \partial continent(p),$$

where $continent(p)$ refers to the (unique) $C \in \mathcal{C}$ with $loc(p) \in \partial C$.

A port p is called a *subway source* if the signal direction enters the corresponding continent, i.e. if one of the following holds

- $signal\text{-}dir(p) = \text{north}$ and $loc(p)$ is on a southern segment of $\partial continent(p)$.
- $signal\text{-}dir(p) = \text{south}$ and $loc(p)$ is on a northern segment of $\partial continent(p)$.
- $signal\text{-}dir(p) = \text{west}$ and $loc(p)$ is on an eastern segment of $\partial continent(p)$.
- $signal\text{-}dir(p) = \text{east}$ and $loc(p)$ is on a western segment of $\partial continent(p)$.

Otherwise, p is a *subway sink*.

Remark. We defined continents as 2-dimensional objects. In practice, they are allowed to have a ceiling that is lower than the chip's ceiling. In this case, ports can be placed on a continent ceiling and have signal direction *up* or *down*. However, this type of port does not play a role in this thesis and hence was neglected in the definition.

Definition 4.3. A *port graph* A for a given net N with root r and sinks T is an arborescence rooted at r with leaves T such that the following holds:

- All inner vertices of A are ports. These are denoted by $ports(A)$.
- If r lies inside a continent C, all out-going edges of r go to sinks that are also contained in C or to subway sinks of C.

- If r lies inside panthalassa, all out-going edges of r go to sinks that are also contained in panthalassa or to subway sources.
- Edges starting at a subway source of a continent C go to sinks inside C or subway sinks of C.
- Edges starting at a subway sink of a continent C go to subway sources of continents distinct from C or to sinks inside panthalassa.

Using these definitions, we can formulate the Pangea port assignment problem:

Pangea port assignment problem

Instance: A world $\mathcal{C}$ and a netlist $\mathcal{N}$.

Task: For every net $N \in \mathcal{N}$, compute a port graph pg_N that contains at most one subway source within each continent $C \in \mathcal{C}$, such that for any two ports $p \in V(pg_N), p' \in V(pg_{N'})$ of nets $N, N' \in \mathcal{N}$, their locations fulfill the spacing requirements of their respective wire types.

While this problem definition captures what it means for a set of port graphs to be feasible, it does not contain any optimization objectives. To measure properties of the port assignment we need to consider how it restricts the routing space.

Definition 4.4. Let Y be a global route (see Section 2.4) for a net N and let A be a port graph for N. We write $E_{cross}(Y) \subseteq E(Y)$ for the set of edges of Y that cross a continent border. We say that Y *implements* A if there is a bijection $b : ports(A) \to E_{cross}(Y)$ such that

- every port p lies on the border between the global routing tiles connected by the edge $b(p)$,
- for every edge (u, v) in A, the $b(u)$-$b(v)$-path in Y contains no edges crossing a continent border (except potentially $b(u)$ and $b(v)$), where we use the notation $b(t) = t$ for terminals $t \in N$,
- the signal direction of every port p represents the arc direction of the associated $b(p)$ and
- we have $wt(p) = wire\text{-}type(b(p))$ for every port $p \in ports(A)$.

A closer depiction of how a port graph and implementing route may look like is given in Figure 4.2.

As with our definition, routes are always tile-center to tile-center, they do not exactly follow port positions. To capture the exact continent crossing positions, we need additional information.

Definition 4.5. A *track-assigned route* is a global route Y together with a function $t : E_{cross}(Y) \to \mathbb{R}$.

For a track-assigned route Y and port graph A for the same net, we say that Y *implements* A if Y implements A as a global route and

- for every port $p \in ports(A)$ with $signal\text{-}dir(p) \in \{\text{north}, \text{south}\}$, we have $loc(p)_x = center_x(b(p)) + t(b(p))$, and

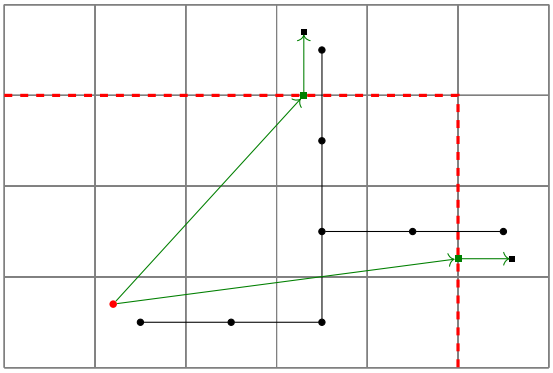


Figure 4.2: A port graph and an implementing route. The gray grid denotes the global routing tiles, the dashed red line represents a continent border. The route graph is shown in black, ports and port graph edges are shown in green.

- for every port $p \in \mathit{ports}(A)$ with $\mathit{signal\text{-}dir}(p) \in \{\text{west}, \text{east}\}$, we have $\mathit{loc}(p)_y = \mathit{center}_y(b(p)) + t(b(p))$.

Here, we used $\mathit{center}_x(e)$ to denote the horizontal center point of the tile column of a vertical edge e. Analogously, $\mathit{center}_y(e)$ to denote the vertical center point of the tile row of a horizontal edge e.

Note that while a global route might implement several port graphs differing slightly in port positions, a track-assigned route can only implement a unique port graph. With these definitions, we can consider the congestion and timing properties of a given port assignment.

PANGEA ROUTING PROBLEM

Instance: A global routing graph $G = (V, E)$ with edge capacities $c : E \to \mathbb{N}$ and a netlist $\mathcal{N} = (N_i \subset V)_i$ as in the simple global routing problem, linear timing data for all layers and wire types, as well as a world $\mathcal{C}$.

Task: Compute a solution $(pg_N)_{N \in \mathcal{N}}$ for the PANGEA PORT ASSIGNMENT PROBLEM given by $\mathcal{C}$ and $\mathcal{N}$ as well as a global route R_N for every net $N \in \mathcal{N}$ implementing pg_N such that the linear-delay timing requirements are met. Further, for each global routing edge $e \in E$, we require

$$\sum_{N \in \mathcal{N}} \mathbb{1}_{E(R_N)}(e) \leq c(e).$$

Under these constraints, the solution should minimize

$$\text{net length} := \sum_{N \in \mathcal{N}} \sum_{e \in E(R_N)} \text{length}(e).$$

In this definition we assumed full timing data to be given. However, due to the fact that the Pangea flow is used on very early stages of the design, timing properties of macros are likely to change significantly later on. Therefore, timing properties known at this stage are not reliable.

This is why, at this stage, it is often not worthwhile to spend the extra amount of running time needed to take timing into account explicitly. Instead, we require the

routes to contain shortest paths up to a factor of $1 + \varepsilon$ from the source to each sink for a fixed $\varepsilon > 0$.

In practice, each net additionally comes with a layer assignment, specifying the lowest layer that ports may be placed on. As higher layers allow for faster signal transmission, this also, implicitly, incorporates information about timing criticality.

4.2 Related work

Early publications consider the problem of pin assignment, where given pins on the border of a unit need to be assigned to connection points outside the unit.

A simple version of this is considered in [Kor72]. The author proposes a purely geometrical algorithm based on mapping points on two concentric circles. Units are considered one by one and routing is not considered explicitly.

[Con91] later combined the pure pin assignment problem with global routing. They first compute a coarse global routing and then obtain the final assignment via a linear time legalization. The author argues that the quality of a pin assignment heavily depends on the used global routing algorithm later on, and that it is difficult to even assess the quality without routing. [Con91] also considers all units simultaneously, eliminating the dependence on a given ordering of the units. However, routing is only allowed in the channels between units, rather than through the units themselves. In particular, all pins need to be specified in advance, in contrast to being determined on the fly as needed. The algorithm was extended in [KWY96].

Another interesting work is [XTW01]. Here, pin assignment and routing are done simultaneously. Given that there are only two-terminal nets from one source block to the other blocks, the authors can compute a solution minimizing net length and via count in polynomial time.

Further works considering pin assignment with routing include [Hon+92], [Che+99] and [PK09]. Also see [Sch09] for an overview over pin assignment in the context of floorplanning.

What all these publications have in common is that the units are positioned with routing space in between. Additionally, only routing between units is considered, while internal routing is neglected. But the connections of internal components to their assigned pins also depend on the computed pin assignment.

[PK09] does push some routing into the units themselves to improve congestion, but it still reserves routing space (the top channel) for inter-unit connections only.

In contrast, BonnPangea gives the entire routing space to the units, which then go up to the chip ceiling. They are also close enough together that no significant routing is done in between. This provides a lot of flexibility and reduces area consumption.

No publications on optimizing a port assignment with a similar problem formulation are known to the author.

4.3 Pangea topologies in theory

In this section we explore the main difference of the pangea router to traditional global routing: pangea topology generation. We formulate the strict pangea topology problem and investigate the number of created ports and the cost in wire length of allowing only one entering port per continent. In practice, the problem corresponds to the first step of the global routing oracle, see Section 3.1.1.

We fix some notation for this section. As we are not in the context of global routing but only consider trees in $(\mathbb{R}^2, \ell_1)$, we use slightly modified definitions: Fix a root $r \in \mathbb{R}^2$ and a terminal set $T \subseteq \mathbb{R}^2$. Let $\mathcal{C}$ be a set of axis-parallel closed rectangles that are

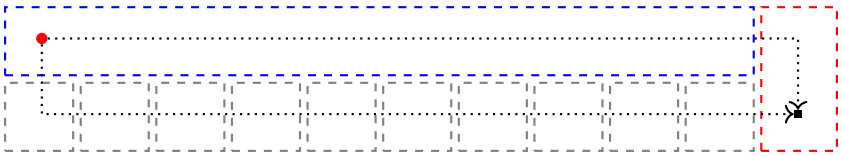


Figure 4.3: An example instance of the Strict Pangea Topology Problem with two optimal solutions (dotted paths). The upper path only induces two continent crossings, while the lower path can induce arbitrarily many depending on the number of gray continents.

disjoint. We call $\mathcal{C}$ the set of *continents*. We call a Steiner tree A on $\{r\} \cup T$ together with an embedding $p : V(A) \setminus (\{r\} \cup T) \to \mathbb{R}^2$ such that all edges are axis-parallel a *topology*, and implicitly assume A to be an r-arborescence. Further, we identify a Steiner node v with its embedding $p(v)$.

Now let us define the notion of a continent crossing in a topology (A, p): We identify an edge $(v, w) \in E(A)$ with the straight segment in $\mathbb{R}^2$ between $p(v)$ and $p(w)$, and say that a *(continent) border crossing* is a point $(x, y) \in \mathbb{R}^2$ together with an edge and a continent C such that (x, y) is contained in the intersection of the edge and ∂C. Moreover, we say that an edge (v, w) *enters* $C \in \mathcal{C}$ if it contains a segment $[a, b]$ such that one of the following holds:

a) $a \notin C$ and $b \in C$ in case $\ell_1(a, v) \leq \ell_1(a, w)$, or

b) $a \in C$ and $b \notin C$ in case $\ell_1(a, v) > \ell_1(a, w)$.

An edge (v, w) *leaves* C if (w, v) enters C. Note that edges may be involved in more than one border crossing. In particular, a single edge can enter and leave the same continent (at different points).

The Strict Pangea Topology Problem is the task of computing a shortest topology under all shortest-path topologies that enter each continent at most once. This formulation most closely follows the original definition of the Pangea routing problem when not allowing detours. While it is very natural to allow only one entry per continent when trying to keep the number of continent crossings low, it does not guarantee a minimum port count. In fact, the number of ports can be arbitrarily high compared to the minimum, even for two-terminal nets. An example for this can be seen in Figure 4.3.

While this constraint does not guarantee minimum port count, it can still increase the length of an optimum solution drastically. The maximum length increase depends on the number of continents. Let $\mathcal{I}_k$ denote the set of all instances of the Strict Pangea Topology Problem with exactly k continents. Then we can say the following about the length of optimum solutions with vs. without the constraint of entering each continent at most once:

Theorem 4.6. *For an instance I of the* Strict Pangea Topology Problem*, let $OPT_1(I)$ denote the length of an optimum solution. Let $OPT_\infty(I)$ be the length of a shortest shortest-path topology (that is allowed to enter each continent many times). Then for $k \in \mathbb{N}$,*

$$\sup_{I \in \mathcal{I}_k} \frac{OPT_1(I)}{OPT_\infty(I)} = k + 1.$$

Proof. We first show that there is a sequence of instances $(I_i)_{i \in \mathbb{N}} \subseteq \mathcal{I}_k$ such that $\sup_{i \in \mathbb{N}} \frac{\mathrm{OPT}_1(I_i)}{\mathrm{OPT}_\infty(I_i)} = k + 1$ for any k, and then prove that the ratio cannot get larger.

So fix $k \in \mathbb{N}$ and define $I_i \in \mathcal{I}_k$ for $i \in \mathbb{N}$ as follows (an instance for $k = 3$ is depicted in Figure 4.4a):

The terminal set consists of

$$\begin{aligned} & T_{\text{panthalassa}} \coloneqq \{(-i,0),(i,0)\} \\ \text{and} \quad & T_{\text{continents}} \coloneqq \{(-i,j),(i,j) \mid j=1,\dots,k\}. \end{aligned}$$

The root r lies at $(0,0)$. Finally $\mathcal{C}$ contains the bounding boxes of $(-i-\frac{1}{3}, j-\frac{1}{3})$ and $(i+\frac{1}{3}, j+\frac{1}{3})$ for each $j=1,\dots,k$.

The unique optimum solution to the Strict Pangea Topology Problem on this instance (as shown in Figure 4.4b) has length $\mathrm{OPT}_1(I_i) = 2i(k+1)+k$, while a shortest shortest-path tree (as in Figure 4.4c) has length $\mathrm{OPT}_\infty(I_i) = 2i+2k$. But

$$\lim_{i\to\infty} \frac{\mathrm{OPT}_1(I_i)}{\mathrm{OPT}_\infty(I_i)} = \lim_{i\to\infty} \frac{2i(k+1)+k}{2i+2k} = k+1$$

as claimed.

Now consider any instance $I \in \mathcal{I}_k$ with a shortest-path tree A_0 of length $\mathrm{OPT}_\infty(I)$. Let $\mathcal{C}' \subseteq \mathcal{C}$ denote all continents that A_0 enters. We construct a solution to the Strict Pangea Topology Problem of length at most $(k+1)\mathrm{OPT}_\infty(I)$. An example instance together with a shortest-path tree A_0 is shown in Figure 4.5a. The edge sets defined below for this instance are indicated in Figure 4.5b.

Let B_0 be the branching arising from A_0 by deleting everything inside each continent, i.e. subdivide edges at border crossings and then delete vertices in the interior of continents and their adjacent edges.

For a continent $C \in \mathcal{C}'$, let A_C be the graph arising from taking A_0, projecting all vertices into C and preserving the edges.

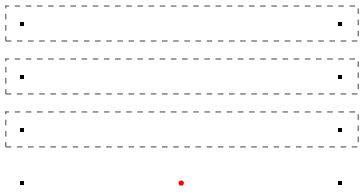
Finally, construct a shortest-path tree D from the root to all points $(C_r)_{C\in\mathcal{C}'}$, where C_r arises from projecting the root into a continent C. We make sure that D enters every continent at most once with the following construction: As long as a segment of D enters a continent C at some position $p \neq C_r$, we remove the incoming segment of p and add a shortest path from C_r to p. This preserves the shortest-path property, because every point in C can be reached by a shortest path from r by going through C_r.

Now take any shortest-path tree A' inside $B_0 \cup \bigcup_{C\in\mathcal{C}'} A_C \cup D$ connecting the root to all sinks. Such a tree is depicted in Figure 4.5c. First note that there exist shortest-paths inside A' to all sinks: For sinks inside a continent $C \in \mathcal{C}'$, D makes sure the continent can be reached via a shortest path, and A_C connects to the sink itself. For a sink in panthalassa, it either gets connected to the root directly by edges of B_0. This is a shortest path as A_0 contains only shortest paths. Or it gets connected by a path ending with leaving a continent followed by some edges of B_0. Both the path up to and through the last continent and the path in B_0 are shortest paths.

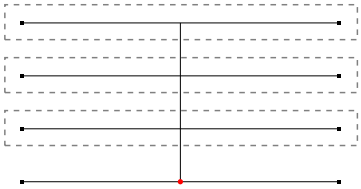
It is clear that A' enters every continent at most once, as B_0 and $\bigcup_{C\in\mathcal{C}'} A_C$ contain no border crossings at all. Write $|E|$ for the total length of an arborescence or edge set E. What is left to show is that $|A'| \leq (k+1)|A_0|$. For that, observe that $|B_0| \leq |A_0|$. Now consider a continent $C \in \mathcal{C}'$. Surely, $|A_C| \leq |A_0|$. But because A_0 contains a path from the root into C which gets removed by the projection, we know $|A_C| + \mathrm{dist}(r,C) \leq |A_0|$. Now using that $|D| \leq \sum_{C\in\mathcal{C}'} \mathrm{dist}(r,C)$, we get

$$\begin{aligned} |A'| &\leq |B_0| + \sum_{C\in\mathcal{C}'} |A_C| + |D| \\ &\leq |A_0| + \sum_{C\in\mathcal{C}'} |A_C| + \sum_{C\in\mathcal{C}'} \mathrm{dist}(r,C) \\ &\leq (|\mathcal{C}'|+1)|A_0| \leq (k+1)|A_0| = (k+1)\mathrm{OPT}_\infty(I). \end{aligned}$$

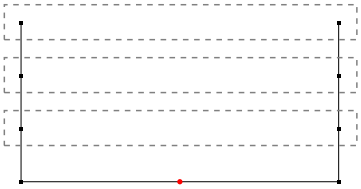
□



(a) An instance as defined in the proof of Theorem 4.6 for the Strict Pangea Topology Problem.

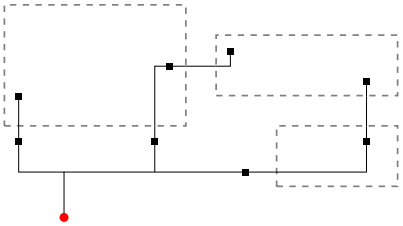


(b) The optimum solution to the instance in Figure 4.4a.

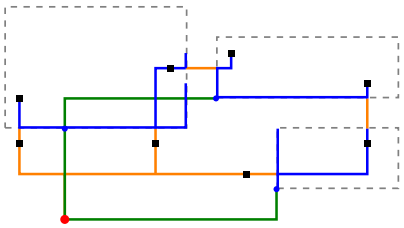


(c) A shortest shortest-path tree for the instance in Figure 4.4a.

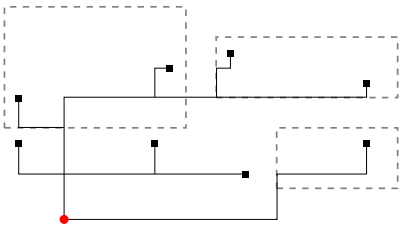
Figure 4.4: The instances used in the proof of Theorem 4.6 together with two Steiner trees. The red point represents the source, the black squares are sinks. The dashed rectangles denote continents.



(a) The instance together with A_0.



(b) Edge sets defined in the proof of Theorem 4.6. B_0 edges are shown in orange. Blue edges belong to A_C for a continent $C \in \mathcal{C}$. Blue circles indicate the projection of the root into every continent. Edges in D are depicted in green.



(c) The resulting tree from the edge sets shown in Figure 4.5b.

Figure 4.5: An instance of the Strict Pangea Topology Problem with a shortest-path tree and the resulting construction as defined in the proof of Theorem 4.6.

Figure 4.6: The instance used to prove a version of Theorem 4.6 where topologies are allowed to enter each continent up to E times, shown for $E = 3$ and $k = 3$.

So in the worst case, restricting the topology to enter every continent at most once can increase the net length by a factor of $|\mathcal{C}| + 1$. Seeing that the shortest shortest-path tree seen in Figure 4.4c enters every continent twice, is seems natural to ask what happens if we bound the number of continent entries by a constant $E > 1$. However, for any fixed $E > 1$, Theorem 4.6 holds analogously when only restricting topologies to enter every continent at most E times. An instance for the example of $E = 3$ is depicted in Figure 4.6.

4.4 Standard pangea flow

The basic approach used in BonnPangea is rather straight-forward. Ports define the positions where future wiring must cross the continent borders. So we estimate (using global routing) where the wiring will be and then place ports on the intersections of the estimated wire with continent borders.

Because global routing creates overlapping wires, a partial track assignment is performed afterwards. The track assignment is restricted to regions around continent borders. It takes global routes going from tile-center to tile-center (and hence overlap) and assigns them to individual routing tracks. The tracks obey all spacing constraints.

After track assignment, all routes are traversed in a depth-first-search. We store information on where the routes cross continent borders and how these points are interconnected. This process is called port cutting.

The pangea flow can either be done in an explicitly timing-aware mode or in a shortest-path mode. In timing-aware mode, the routing uses a linear delay model to make sure that ports allow for fast enough signals. Otherwise, the routing uses algorithms computing almost-shortest source-to-sink paths.

In a more elaborate flow of alternatingly performing port assignment and continent optimization, the timing-aware BonnPangea mode might outperform the almost-shortest-path version. When using pangea to design ports in very early design stages, the shortest-path mode is more efficient.

4.4.1 Continent border blockages

For port cutting, the wire segments crossing continent borders need to be straight: No vias (or jogs) are allowed near continent borders. This is achieved by creating border blockages: Along each horizontal border segment, blockages of fixed height are created on all layers with horizontal routing direction. Analogously, blockages of fixed width are created along vertical border segments on vertical routing layers. An example for this is shown in Figure 4.7. Both the global routing and track assignment respect these blockages. This creates straight wire segments.

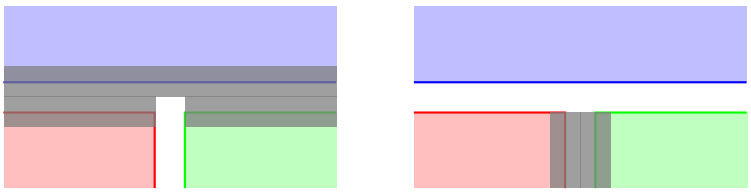


Figure 4.7: Partial world with three continents and border blockages shown as gray overlay on a horizontal routing layer (left) and a vertical routing layer (right).

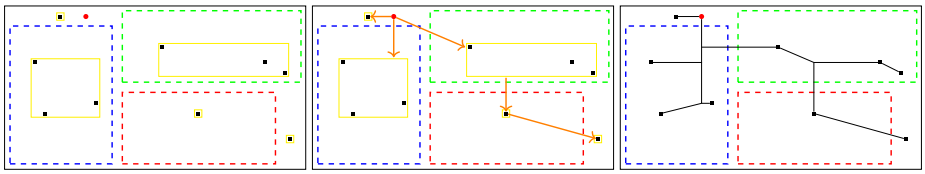


Figure 4.8: The three steps of computing a topology in pangea: clustering, hierarchy computation and tree computation. Yellow rectangles show the sink clustering, orange arrows indicate the cluster hierarchy. The last image shows the final topology.

In practice, neighboring continents are usually close enough together such that these border blockages overlap. This has the effect that no wiring along such continent gaps is allowed, and that connected subway ports across a gap have to be aligned perfectly.

4.4.2 Global routing

Global routing is the basis and largest part of BonnPangea. The global routing framework described in Section 2.4 is used, with small adjustments to the framework itself and the embedding as well as a specialized topology generation algorithm in the routing oracle.

We must align the global routing graph to the continent borders in the sense that all tiles must either lie completely inside or completely outside each continent. Otherwise, there are edges of the global routing graph corresponding to a set of tracks partially inside and partially outside a continent. This would make the subsequent track assignment much more difficult. When all global routing tiles either lie completely inside or outside a continent (or clearly define a continent border crossing), most of the work is already done after global routing. Then, track assignment is merely needed to spread the wires apart.

The following speed-ups make it feasible to compute global routings on large pangea instances: Firstly, all nets completely contained inside single continents are ignored. This does impact the congestion estimates as a significant amount of wiring cannot be seen. But it reduces the scope of the routing task greatly. Secondly, due to the large chip area, global routing is done with a very coarse tile size. This keeps the size of the global routing graph manageable. The coarse tile size does not impact solution quality too much, because the detailed wire positions are computed in track assignment later on. However, a too large tile size increases the running time of track assignment. In practice, we use tile sizes of 400 to 1000 tracks.

Topology generation

To achieve the requirements from Section 4.1 (short paths, entering every continent at most once, short total length), BonnPangea uses its own topology generation algorithm. Topology generation is done using a hierarchical shortest-path tree. The

Algorithm 5: Pangea topology generation

Input: A net $N = \{r\} \cup T \subset \mathbb{R}^2$ and a world $\mathcal{C}$.
Output: A Steiner tree Y rooted at r with leaves T whose Steiner points are positions in $\mathbb{R}^2$.

1 $W :=$ **compute-continent-clusters**$(T, \mathcal{C})$
2 $c : W \to \mathbb{R}^2 :=$ **compute-cluster-roots**(W, r)
3 $Y^{top} :=$ **compute-shortest-path-tree**$(r, c(W))$
4 $w^{top} := \emptyset$
5 **foreach** *cluster* $w \in W$ *in decreasing order of* $\ell_1(r, c(w))$ **do**
6 add $c(w)$ to **find-parent-cluster**$(w, W, c, Y^{top}, \mathcal{C})$ or to w^{top}
7 **foreach** *cluster* $w \in W$ *in reverse topological order* **do**
8 $Y_w :=$ **compute-shortest-path-tree**$(c(w), w)$
9 $Y_{w^{top}} :=$ **compute-shortest-path-tree**(r, w^{top})
10 **return** the union of the trees $(Y_w)_{w \in W}$ and $Y_{w^{top}}$

algorithm is presented in Algorithm 5 and the main steps are illustrated in Figure 4.8. Here, we restrict ourselves to the timing-unaware mode.

Consider Algorithm 5. First, the sinks of a net are clustered according to the containing continents (Line 1). Special care is taken for the sinks in panthalassa due to its highly non-convex shape (see below). Note that the other continents may in principle also be highly non-convex as they are only required to consist of connected rectangles. But in practice, their shapes are at least very close to a single rectangle. After clustering, we estimate virtual roots for all clusters and compute a top-level tree on the net source and the virtual roots as sinks (Lines 2 and 3). We use the top-level tree to compute parent clusters for each cluster (Lines 5 and 6, also see Algorithm 6). These parent-relations make up the cluster hierarchy. Finally, the topology is computed with approximate shortest-path trees according to this hierarchy (Lines 7-10).

Within the `compute-continent-clusters` procedure, sinks inside normal continents are clustered together. This makes sense when the continent shapes are convex or nearly convex. In this case, the Steiner tree for the resulting cluster is (mostly) contained in the continent as well. However, panthalassa is far from being convex because it contains the gaps between the other continents. Thus, putting all panthalassa sinks into a single cluster could lead to unnecessarily long nets and more border crossings than needed:

When sinks are surrounding a continent, a Steiner tree connecting them has most of its length inside the continent. Whenever that continent has its own sinks and with that its own Steiner tree, the net length inside the continent could roughly double. An indicator for this to happen is whenever the bounding boxes of clusters overlap with each other or with other continents.

Therefore, it is better to split the panthalassa sinks into multiple clusters, so that each cluster gets its own region and hence the bounding boxes are disjoint. To achieve this, we use a simple greedy clustering algorithm: The sinks in panthalassa are considered one-by-one and put into the first already existing panthalassa cluster whose bounding box does not intersect a continent when extended by the new sink. If no such cluster exists, a new one is created.

`compute-cluster-roots` computes $c : W \to \mathbb{R}^2$ by projecting the source into the cluster bounding boxes. I.e. it sets

$$c(w) := \operatorname*{argmin}_{c \in bb(w)} \operatorname{dist}_{\ell_1}(r, c),$$

where $bb(w)$ denotes the axis-parallel bounding box of the sinks contained in w. This

Algorithm 6: find-parent-cluster

Input: Finite clusters $W \subset 2^{\mathbb{R}^2}$ and $w \in W$, cluster roots $c : W \to \mathbb{R}^2$, an arborescence Y^{top} in $\mathbb{R}^2$ with root r and leaves $c(W)$ and a world $\mathcal{C}$.
Output: Either some $w' \in W \setminus \{w\}$ or NONE.

1 $v \coloneqq w$
2 **while** $\delta^-_{Y^{top}}(v) \neq \emptyset$ **do**
3 $v \coloneqq$ predecessor of v in Y^{top}
4 **if** $\exists w' \in W$ *in same continent as* v **and** w' is a feasible parent cluster for w **then**
5 **return** w'
6 **if** $\exists w' \in W$ *such that* $v \in boundingbox(w')$ **and** w' is a feasible parent cluster for w **then**
7 **return** w'
8 **return** NONE

way, the cluster root is furthest away from r while still allowing for shortest paths from r to each sink in w.

After the clustering, a shortest-path tree from r to the cluster roots $c(W)$ is computed in `compute-shortest-path-tree`. This is done by either the bi-criteria algorithm (see Section 3.2.2) or the reachaware shortest-path algorithm (see Section 3.2.1). In both cases, relative detour can be restricted or completely forbidden.

The tree Y^{top} is used to compute a hierarchy on the clusters by iteratively finding a parent cluster. We consider the clusters in an order of descending distance to the source, measured from the estimated cluster root locations. `find-parent-cluster` tries to find a parent instance by traversing Y^{top} from the cluster to the root. The algorithm is described in Algorithm 6. We always consider a candidate $v \in V(Y^{top})$. v is initialized to be w and we go up the tree by updating v to its predecessor in every iteration (Lines 1-3). For every candidate v, we check whether there is a cluster in the same continent as v or one whose bounding box contains v. If the cluster also is a feasible parent, we return it (Lines 4-7). Here, a cluster w' being a *feasible parent* for w denotes that the cluster hierarchy stays acyclic when introducing this relation and that a shortest r-$c(w')$-$c(w)$ path is not much longer than a direct r-$c(w)$ connection. When we reach the root of Y^{top} without finding a parent, the algorithm returns NONE (Line 8).

Whenever `find-parent-cluster` returns NONE, the point $c(w)$ is added to w^{top} in Algorithm 5.

In Lines 7 and 8 of Algorithm 5, the cluster hierarchy is then traversed bottom-up, computing a shortest-path tree for each considered cluster. Finally, a tree is computed connecting the roots of all clusters on the top most hierarchy level to the net source (Line 9 of Algorithm 5).

In this algorithm, the cluster roots are placed into the same continent as the sinks of a cluster (assuming the continents are convex). Therefore, we satisfy the requirement that all sinks inside a continent must be connected to the same subway source. This subway source results from the border crossing of the topology edge going into the cluster root. Additionally, each cluster tree and the top-level tree are approximate shortest-path trees and the cluster hierarchy does not deviate from the top-level tree in a way that creates further detours. Therefore, the resulting tree also is an approximate shortest-path tree.

When looking closely at the final topology in Figure 4.8, we can see that a shorter Steiner tree still obeying shortest paths could be found by changing the cluster hierarchy. The bottom right sink in panthalassa is closer to the estimated cluster root in the red

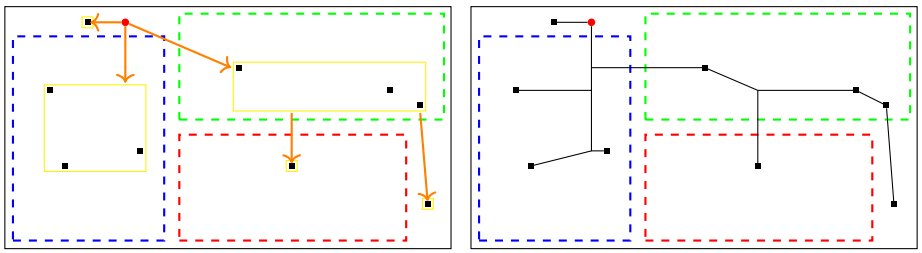


Figure 4.9: Cluster hierarchy (left) and resulting topology (right) after post-optimization.

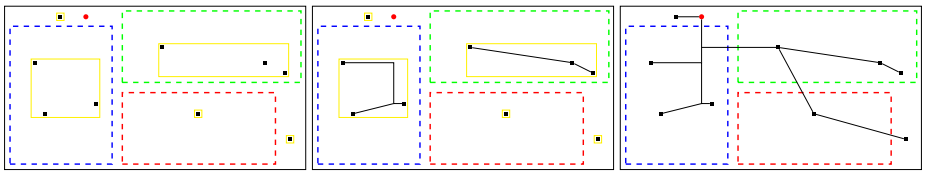


Figure 4.10: Computing a topology using the simpler two-level approach: Compute a tree for each cluster separately, then connect the cluster roots via a single top-level tree.

continent than the one in the green continent. Therefore the sink got connected to the cluster in the red continent. However, it is cheaper to connect the panthalassa sink to the cluster in the green continent: The green continent has a sink which also lies on a shortest path and is closer to the panthalassa sink. We post-optimize the cluster hierarchy to also consider sink positions. In the example, this leads to a new hierarchy and shorter resulting topology as depicted in Figure 4.9. In practice, such a post-optimization is used in between Lines 6 and 7 of Algorithm 5.

Alternative ways to compute a hierarchical topology

Instead of computing the cluster hierarchy, a simpler two-level approach could be used to compute the topology: Start by computing shortest-path trees for each cluster separately. Then connect them directly via a top-level tree from r to the cluster roots.

The results of this approach for the same instance as in Figure 4.8 are depicted in Figure 4.10. However, using the cluster hierarchy leads to shorter trees for some nets because redundant edges can be avoided. This is because in the two-level approach, the top-level tree ignores the previously computed trees. With shorter length, our approach also leads to fewer ports.

A better option than the current implementation would be to compute the cluster hierarchy based on the cluster bounding boxes instead of based on the cluster roots. This approach can already incorporate considerations as described for the post-optimization into the hierarchy computation. For the instance considered in Figure 4.8, the resulting hierarchy would already look like the one depicted in Figure 4.9. However, with this approach we would leave the realm of simple ℓ_1-metrics and no algorithm using this is implemented so far.

Embedding

Embedding the topology is done as described in Section 3.1.2. We have to make sure that paths neither introduce additional detours nor cross too many continent borders. We achieve this by adjusting the cost function in path search. The embedding algorithm itself remains the same.

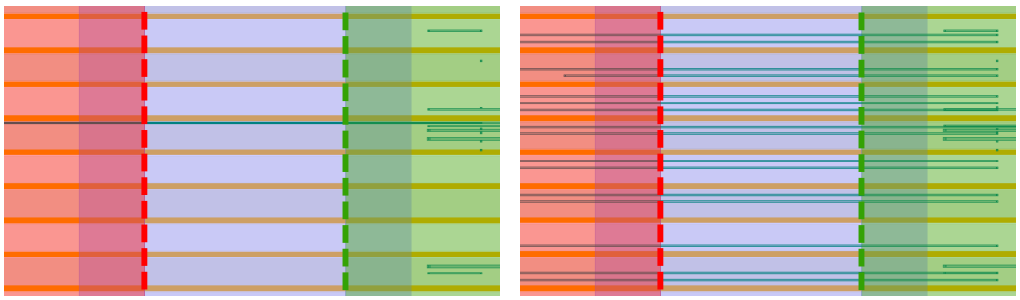


Figure 4.11: A small chip region showing the gap between two continents before (left) and after (right) track assignment. The red and green areas depict the continents. The blue area represents the border region in which the wires are assigned. Orange lines depict power rails. Turquoise lines are wires. In the left picture, all wires of the current layer lie in the tile-center. The wire segments near the right border are on higher layers and were already assigned to tracks. After track assignment, all wire segments are spread.

Undesired edges get an additional high penalty cost to discourage Dijkstra's algorithm from using them. Firstly, it is forbidden to cross a continent border against target direction. This means edges entering or leaving a continent in a direction opposite to the target of the current Dijkstra search get penalized. Secondly, whenever both simultaneously embedded sibling paths lie completely inside a continent, all edges leaving that continent are forbidden. These penalty costs ensure that detours crossing continent borders are prevented, while still enabling the embedding to route through the necessary continents in a cost effective way (instead of a hard restriction to use the minimum number of ports possible).

Further options include adding a small penalty cost to all edges crossing a continent border. This encourages the router to choose paths with fewer continent crossings in total. But it also increases wire congestion significantly, because congestion hot-spots cannot be circumvented as flexibly. To explore this option further, more fine-tuning and tests would be necessary.

4.4.3 Track assignment

Global routing places all wires such that they go from tile-center to tile-center. If we created ports according to the wire positions in this state, all ports inside one tile would be in the same position instead of being spread along the border. To make the global wires overlap-free, they are assigned to routing tracks.

This is the process of *track assignment* as proposed in [Bat+02]. The implementation of track assignment in the BonnTools is described in [Dur24]. It is a dynamic program and was originally implemented by Prof. Dr. Vera Traub.

As only wire segments crossing a continent border are of interest for port positions, wiring outside of an area around the continent borders is left unchanged.

The assignment itself works as follows. The routing layers are traversed top to bottom. On each layer, all wire segments intersecting at least one relevant area are collected. These segments are then assigned to tracks corridor by corridor via a dynamic program. Inside of a routing corridor, the dynamic program moves a sweepline along the wires. It stops at all positions where a wire segment starts or ends and propagates partial solutions. The best solution found at the end is implemented.

A visualization of wire segments before and after track assignment can be found in Figure 4.11.

4.4.4 Port cutting

The final step in BonnPangea is to extract the port locations from the global wires. After track assignment, wire segments crossing continent borders are overlap-free and satisfy spacing constraints. So the port positions can be computed by taking the center of a wire segment on a continent border. To make sure the correct connection data is preserved, routes are traversed by a depth-first-search. By keeping track of the last seen port up-stream, a port graph can be computed in linear time in the size of the route graph.

4.5 Pangea replay

During the design process, port assignment is called repeatedly on slightly changing versions of a design. For stability reasons, it is desirable that the resulting port locations are as similar as possible between such runs. To achieve results as close to the original version as possible, BonnPangea offers the replay mode.

As additional input, port graphs for a subset of nets are given. The objective is to solve the Pangea routing problem such that the input port graphs coincide with the computed port graphs of the respective nets. It is also allowed to have *partial port graphs* in the input. A partial port graph is a sub-graph of a port graph. In this case, the computed port graphs must contain the given partial port graphs for the respective nets. A formal definition of the problem is as follows.

Pangea replay problem

Instance: A global routing graph $G = (V, E)$ with edge capacities $c : E \to \mathbb{N}$, a netlist $\mathcal{N} = (N_i \subset V)_i$ and a world $\mathcal{C}$ as in the Pangea routing problem, as well as (partial) port graphs $(P_N)_{N \in \overline{\mathcal{N}}}$ for a subset of nets $\overline{\mathcal{N}} \subseteq \mathcal{N}$.

Task: Compute a solution $(R_i, A_i)_i$ to the Pangea routing problem (see Section 4.1) also satisfying that for all $N_i \in \overline{\mathcal{N}}$, we have $P_{N_i} \subseteq A_i$ and minimizing

$$\text{net length} := \sum_i \sum_{e \in E(R_i)} \text{length}(e).$$

As the (partial) port graphs might stem from an earlier version of a design, they might be associated with slightly modified nets. Whenever a net changed from the previous run producing the port graph to the current run, its port graph must be aligned to the new version of the net before starting the replay flow. For terminals that are only shifted by a small distance, this often requires no changes to the structure of the port graph. But care has to be taken for sinks that were moved to another continent or were newly added to the net.

Roughly, this adaptation consists of three steps. Edges that have become illegal are removed from the port graph, new edges are added to connect new sinks, and ports that have become obsolete by these changes are removed. Note that these adaptations might convert a previously complete port graph into a partial one, as no new ports may be created.

We see another important application of the replay mode, in particular with partial port graphs, in Section 4.6.

Algorithm 7: Pangea topology generation with replay

Input: A net $N = \{r\} \cup T \subset \mathbb{R}^2$, a world $\mathcal{C}$ and a (partial) port graph P for the net N.

Output: A Steiner tree Y rooted at r with leaves T whose Steiner points are positioned in $\mathbb{R}^2$ implementing P.

1 $W \coloneqq$ **compute-clusters-replay**$(T, \mathcal{C}, P)$
2 $c : W \to \mathbb{R}^2 \coloneqq$ **compute-cluster-roots-replay**(W, r, P)
3 $Y^{top} \coloneqq$ **compute-shortest-path-tree**$(r, c(W))$
4 $w^{top} \coloneqq \emptyset$
5 **foreach** *cluster* $w \in W$ *in decreasing order of* $\ell_1(r, c(w))$ **do**
6 add $c(w)$ to **find-parent-cluster-replay**$(w, W, c, Y^{top}, \mathcal{C}, P)$ or to w^{top}
7 **foreach** *cluster* $w \in W$ *in reverse topological order* **do**
8 $Y_w \coloneqq$ **compute-shortest-path-tree**$(c(w), w)$
9 $Y_{w^{top}} \coloneqq$ **compute-shortest-path-tree**(r, w^{top})
10 **return** the union of the trees $(Y_w)_{w \in W}$ and $Y_{w^{top}}$

Topology generation

We define the notion of a topology Y of a net N *implementing* a (partial) port graph P for N as follows: For every port in P, there is a Steiner point at the same position in Y (hence we can write $V(P) \subseteq V(Y)$). For every arc $(a, b) \in E(P)$, there is an a-b-path in Y that does not cross any continent border.

To compute a topology implementing a given (partial) port graph, Algorithm 5 is modified slightly: Consider Algorithm 7. In Line 1, `compute-clusters-replay` creates one cluster for each port in P containing the sinks connected directly to that port. Unconnected sinks inside some continent are added to a cluster in that continent if one exists, otherwise they form a new cluster. Unconnected sinks inside panthalassa are clustered using the same greedy approach as in the standard flow.

In Line 2, `compute-cluster-roots-replay` sets the root of clusters induced by a port to the port position and sets

$$c(w) \coloneqq \operatorname*{argmin}_{c \in bb(w)} \operatorname{dist}_{\ell_1}(r, c)$$

as before for newly created clusters w.

The rest of the algorithm is the same as in Algorithm 5, except for the sub-routine to find a parent cluster. `find-parent-cluster-replay` (in Line 6) checks whether a cluster w was induced by a port p. In this case, if p has a parent port in P, the cluster induced by that parent port is returned. If p is connected to r in P, NONE is returned to indicate that $c(w)$ should be added to w^{top}. Otherwise, the `find-parent-cluster` routine of Algorithm 6 is used.

Assuming that the port graph contains only one subway source into each continent, this procedure makes sure the final topology obeys the fact that all sinks inside a continent are connected to the same subway source. However, for continents with multiple subway sources in the input port graph, this is not guaranteed. In general, matching the input port graph takes precedence over other constraints such as few ports or short paths.

Embedding

When embedding the topology, computed paths must cross continent borders at the correct locations. In order for track assignment to be able to assign a wire to the exact

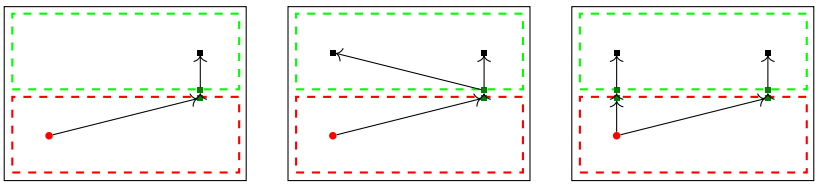


Figure 4.12: Example replay scenario in which the output cannot use only one entering port into the green continent and allow for shortest paths at the same time when replaying the given port. The input port graph is depicted on the left. It is a shortest path and has the minimum number of ports. Solutions following this input data (i.e. keeping existing ports at their location) must either introduce a large detour to the new sink (middle) or add a second port into the green continent (right).

location of the port in the input data, the route must go through the global routing edge corresponding to the predefined port.

We ensure this with an adjustment to the cost function: Whenever a path has an enforced port that is a subway source into a continent, all global routing edges going into that continent except for the one corresponding to the port get a high penalty cost. Analogously, for paths with a subway sink leaving a continent, all other edges leaving the continent are penalized. For high enough penalty prices, this makes the embedding go through the correct edge if at all possible. The penalty prices used here are the same as for completely blocked edges, i.e. edges with capacity zero.

Track assignment

Before the normal track assignment algorithm, all nets are checked for predefined ports. Whenever a wire segment traverses a global routing edge containing a port for the given net, its position is fixed to the port's position in advance. All remaining wires are assigned afterwards, where the pre-assigned wires constitute blockages. This way, all wire segments that were correctly routed in the embedding step are guaranteed to be assigned to the track corresponding to the port location.

4.5.1 Trading off replaying ports with other pangea objectives

In general, no restrictions regarding short net length, short paths or number of ports are imposed on the replay port graphs taken as input. This means that all these criteria might be violated in solutions following the given port graphs. Since the replay mode is used to re-create solutions previously computed by the standard pangea mode, this does not pose problems for nets that do not change in between runs. However, even a port graph that was originally computed fulfilling all criteria might become a hindrance as soon as a single new sink is added to its net.

Consider the simple example in Figure 4.12. First, the depicted net only has one sink in the green continent. In the second pass, a new sink is added also in the green continent further to the left. The original net has only a single sink and the ports are placed in a way that allows for a shortest path and thus also a short net. However, the resulting port graph prevents solutions for the altered net that have only one entering port and short paths to both sinks. When retaining the given port, the new route has to connect the added sink to it as well, producing a significant detour, or add a second port entering the green continent.

Since it is not possible to place ports in a way that all potential future replay computations lead to good solutions, a mechanism to decide which replay port positions should be discarded could greatly improve upon the existing replay mode. Here, discarding a

port means removing it and its adjacent arcs from the port graph before computing the topology. For accurate decisions, it has to be rated how important the port count, short paths and replaying the old ports are relative to each other in a given situation. This in itself is already a difficult problem. The importance of individual short paths to certain sinks can be estimated using the timing criticalities of the sinks. But the number of (entering) ports as well as the value of replaying ports cannot be quantified easily.

When keeping the requirement of only allowing one entering port per continent (as in the standard pangea mode), port count does not need to be taken into consideration. In the example of Figure 4.12, we would not be allowed to use the solution on the right. So this case reduces to the question of re-using the port position and accepting the detour (center solution), or discarding it and creating a port further left to achieve shortest paths. One possibility to approach this is to define a fixed cost for discarding/moving a predefined port and delegate the decision to the resource sharing oracle. Two versions of this proposal are sketched in the following section.

4.5.2 Replaying ports as resource

When replaying ports should be strongly encouraged, but not mandatory in all situations, it seems natural to introduce a penalty cost for discarding a port position. The route computation is conducted in the oracle of the resource sharing framework. So the best way to introduce an additional cost is to model replaying ports as an additional resource. A simple way to do this is as follows.

Denote by P the total number of specified ports in the set of port graphs given as input. Let $a \in (0, 1]$ be a parameter given by the input. It shall denote what portion of the predefined ports may be discarded/moved. In the resource sharing framework, we add an additional resource *replay*. The usage of that resource by a route Y is

$$\mathrm{usg}_{\mathrm{replay}}(Y) \coloneqq \frac{n}{a \cdot P},$$

where n counts the number of predefined ports the route does not follow. The penalty cost of discarding a port now naturally arises as resource cost of the replay resource. The parameter a controls how high that penalty is while still allowing for exponential growth similar to the other resource sharing costs.

This way, the number of not-replayed ports can be kept in check. However, there is no distinction between just moving a port to a neighboring global routing tile and placing it at an opposite end of the continent. To accommodate the difference, usage of the replay resource can be defined proportional to the total (quadratic) movement of the predefined ports. When interpreting the replay resource as total (quadratic) movement, it is harder to define a sensible capacity value. One solution would be to estimate a reasonable amount of port movement for the first resource sharing phase and then adjust the capacity according to a fixed usage rate given as parameter. For usage parameter a, previous relative total resource usage $\mathrm{usg}_{\mathrm{prev}}$ of the replay resource and total (quadratic) movement m of a route Y, usage in the next phase would be defined by

$$\mathrm{usg}_{\mathrm{replay}}(Y) \coloneqq \frac{m \cdot a}{\mathrm{usg}_{\mathrm{prev}}}.$$

To properly estimate port movement, we can compute a matching between the predefined ports and those used in the computed route. We define the bipartite graph $G = (A \cup B, E)$ as follows. A is the set of all predefined ports, B is the set of all border crossings of the route Y plus dummy nodes b_a for all $a \in A$. E contains edges between all pairs $a \in A$, $b \in B$ that represent border crossings of the same continent and the same type (entering a continent/subway source or leaving a continent/subway sink). For each

$a \in A$, E also contains the edge $\{a, b_a\}$. Further we define edge weights $w : E \to \mathbb{R}^+$ to capture the distance (measured on the continent boundary) between the predefined port position and the final border crossing. Edges of type $\{a, b_a\}$ are assigned a weight of half the perimeter of the continent of a, i.e. the maximum distance possible on the continent boundary.

Now we compute a minimum-weight matching M in G that covers A. Such a matching always exists by using the edges $\{a, b_a\}$. An edge $\{a, b_a\} \in M$ represents the fact that the port a was completely discarded. Edges $\{a, b\} \in M$ where b is a border crossing of Y denote that the port a was moved to the position of b. Now the total (quadratic) movement can be computed by iterating over all edges in M.

A further aspect to take into account in addition to port movement is the enforced detour on the source-to-sink paths. One possibility is to forbid moving ports enforcing a too large detour completely. Otherwise, costs of such ports can be increased by incorporating weights into the considered (quadratic) movement values.

Note that in this description, we neglected the fact that the computed routes are tile-center to tile-center, while the predefined ports are given as exact positions. After computing the route, it is still open where inside its tile the final port will lie. To include this, the distances used for the edge weights are the distances between the given port and the tile containing the route's border crossing, instead of the tile's center. This way, using the correct tile does not induce positive edge weight no matter where the predefined port is placed within the tile. Moving a port to a neighboring tile is cheaper on the side where the predefined port's track lies.

In order to use the presented approaches to achieve an optimal trade-off between replaying many ports and preventing large detours on critical paths, it is necessary to consider costs for path lengths in the route computation instead of using a hard restriction on the detour as is mostly done in practice. This can be done by considering timing and using the arrival time customers as presented in Section 2.4.2.

4.6 Pangea reuse

In the setting described in Sections 4.4 and 4.5, it was always assumed that each unit (continent) will be designed individually in later optimization steps. However, there are designs with several instantiations of the same unit, for example when there are multiple equivalent cores on a processor chip. When using the standard pangea flow, each unit would be assigned potentially different port positions. So essentially equivalent continents would still need to be designed individually later on.

However, if we can make sure such equivalent continents are always assigned equivalent port positions, i.e. if the computed interfaces are the same, then the unit needs to be designed only once. This saves both computational and human resources.

Since the best positions for ports depend not only on the interior of a continent, but also on the relative positions of terminals outside, reducing the solution space this way comes with some drawbacks. Overall quality regarding short paths, net length and routing congestion can decrease. Additionally, it can become necessary to put extra wiring into a unit that is only needed for some of its instantiations but not all. Then these wire segments are dead, unconnected wire in the other instantiations.

The problem arising from this setting is tackled by the Pangea ReUse mode that was developed together with Max Mundt [Mun23] who wrote his master's thesis on this topic under the author's co-supervision during the work on this thesis.

The core of this problem is very similar to the one from [LCT22]. However, Pangea ReUse contains many more features needed for practical application.

4.6.1 Setting and problem formulation

We need to introduce the following notions in order to define the PANGEA REUSE problem properly.

Definition 4.7. An *equivalence function pair* is a pair (f_C, f_G) of functions

$$f_C : \mathcal{C} \to \mathcal{C}$$
$$\text{and } f_G : \bigcup_{C \in \mathcal{C}} gates(C) \to \bigcup_{C \in \mathcal{C}} gates(C),$$

such that $f_C \circ f_C = f_C$ and $f_G \circ f_G = f_G$. We further require for a gate $g \in gates(C)$ for some $C \in \mathcal{C}$ that we have $f_G(g) \in gates(f_C(C))$, and that $f_G^{-1}(g)$ is either empty or contains exactly one gate out of each $gates(C')$ for all $C' \in \mathcal{C}$ with $f_C(C') = f_C(C)$ (then $f_G^{-1}(g)$ is a non-trivial gate equivalence class).

Given an equivalence function pair (f_C, f_G), the function f_C maps every continent to the representative of its equivalence class. f_G maps every gate to the equivalent gate in that representative equivalent continent.

The *continent equivalence classes* are now given by the non-empty sets of the form $f_C^{-1}(C)$ for $C \in \mathcal{C}$, and the *gate equivalence classes* by the non-empty sets of the form $f_G^{-1}(g)$ for $g \in \bigcup_{C \in \mathcal{C}} gates(C)$. We call a continent C a *reuse continent* if its equivalence class has cardinality greater 1.

We denote the continent equivalence classes by $\mathcal{C}_1, \ldots, \mathcal{C}_k$. Then $\mathcal{C}$ consists of non-reuse continents together with the union of all equivalence classes $\mathcal{C}_i$:

$$\mathcal{C} = \{C \in \mathcal{C} \mid \nexists\, C' \in \mathcal{C} \setminus \{C\} : f_C(C') = f_C(C)\} \cup \bigcup_{i=1}^{k} \mathcal{C}_i$$

To capture the geometrical aspect of equivalence, we need the following definitions.

Definition 4.8. Let $(x_0, y_0) \in \mathbb{R}$.

A *mirror function* at (x_0, y_0) is a function $m_x : \mathbb{R}^2 \to \mathbb{R}^2$ that is either the identity on $\mathbb{R}^2$ or maps every point to the point mirrored along the x-axis through (x_0, y_0).

A *rotating function* at (x_0, y_0) is a function $rot : \mathbb{R}^2 \to \mathbb{R}^2$ that is either the identity on $\mathbb{R}^2$ or maps every point to the point arising from rotating around the origin (x_0, y_0) by 180 degrees.

A *shifting function* is a function $s : \mathbb{R}^2 \to \mathbb{R}^2$ that can be written as $s(x, y) = (x + \delta_x, y + \delta_y)$ for some $(\delta_x, \delta_y) \in \mathbb{R}^2$.

A *translating function* at (x_0, y_0) is a function $t : \mathbb{R}^2 \to \mathbb{R}^2$ that can be written as $t = s \circ rot \circ m_x$, such that s is a shifting function, rot is a rotating function at (x_0, y_0) and m_x is a mirror function at (x_0, y_0).

Observe that mirroring along the y-axis can be achieved by concatenating rotation with mirroring along the x-axis.

Definition 4.9. Given an equivalence function pair (f_C, f_G), *feasible translation data* is a set $\mathcal{T} = (t_C)_{C \in \mathcal{C}}$ such that the following is satisfied.

For every $C \in \mathcal{C}$, t_C is a translating function at a corner point of C, say o_C, and we have $t_C(C) = f_C(C)$ and $t_C(o_C) = o_{f_C(C)}$. Moreover, for every $C \in \mathcal{C}$ and every $g \in gates(C)$, we have $t_C(g) = f_G(g)$, where we interpret g to denote the shape of the gate.

Problem definition

The interfaces of equivalent continents should be equivalent themselves. This concept is captured by the following notion. We use the notation $\Gamma_G^+(v)$ and $\Gamma_G^-(v)$ to denote the children, respective predecessor sets of a node v in a graph G.

Definition 4.10. Fix an equivalence function pair (f_C, f_G) and feasible translation data $\mathcal{T} = (t_C)_{C \in \mathcal{C}}$. Let $(pg_N)_{N \in \mathcal{N}}$ be a solution to the PANGEA PORT ASSIGNMENT PROBLEM. We call this port assignment *reuse-aware* if the following holds for every net N (where we identify nodes in a port graph with their position):

a) For every subway source $p \in V(pg_N)$ in a reuse continent C, $t_C(p)$ is a subway source p' in some port graph $pg_{N'}$ and

$$\Gamma^+_{pg_{N'}}(p') = \{t_C(c) \mid c \in \Gamma^+_{pg_N}(p)\}.$$

b) For every subway sink $p \in V(pg_N)$ in a reuse continent C, $t_C(p)$ is a subway sink p' in some port graph $pg_{N'}$ and

$$\Gamma^-_{pg_{N'}}(p') = \{t_C(c) \mid c \in \Gamma^-_{pg_N}(p)\}.$$

We similarly call a routing *reuse-aware* if it implements reuse-aware port graphs.

We can now define the pangea reuse problem.

PANGEA REUSE ROUTING PROBLEM

Instance: A global routing graph $G = (V, E)$ with edge capacities $c : E \to \mathbb{N}$, a netlist $\mathcal{N} = (N_i \subset V)_i$, a world $\mathcal{C}$ and a constant $\varepsilon > 0$ as in the PANGEA ROUTING PROBLEM (without timing data), as well as an equivalence function pair (f_C, f_G) and feasible translation data $\mathcal{T}$.

Task: Compute a reuse-aware solution $(pg_N)_{N \in \mathcal{N}}$ with respect to (f_C, f_G) and $\mathcal{T}$ for the PANGEA PORT ASSIGNMENT PROBLEM given by $\mathcal{C}$ and $\mathcal{N}$. Also compute a global route R_N implementing pg_N for every net $N \in \mathcal{N}$ such that the source-to-sink paths in every pg_N are ℓ_1-shortest up to a factor of $1 + \varepsilon$. Further, for each global routing edge $e \in E$, we require

$$\sum_{N \in \mathcal{N}} \mathbb{1}_{E(R_N)}(e) \leq c(e).$$

Under these constraints, the solution should minimize

$$\text{net length} := \sum_{N \in \mathcal{N}} \sum_{e \in E(R_N)} \text{length}(e).$$

4.6.2 Reuse flow

PANGEA REUSE starts with running a first pangea pass modified in such a way that equivalent continents already receive similar port positions. This is achieved by

a) computing intervals for equivalent ports beforehand and

b) copying blockages so that they are equivalent along equivalent borders.

To make sure the interfaces are exactly same, PANGEA REUSE uses a simple leader-follower construct. For every continent equivalence class, one leader continent is chosen

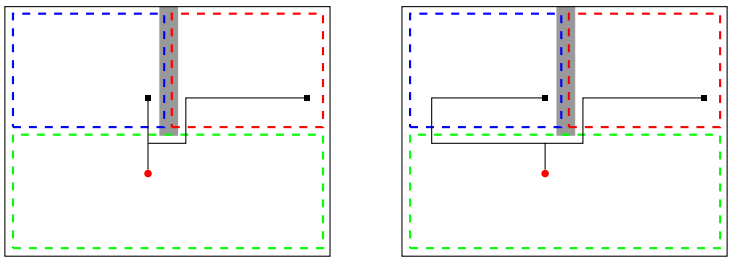


Figure 4.13: Example net that would lead to a long detour when blindly using the leader's port position for the follower continent. The blue and red continents are equivalent. The red continent is the leader. The net source is red, the sinks are black. The gray shape denotes a blockage. The left picture shows how a first routing might look like. The right route is the outcome of forcing the leader port position from the first route to be used in the follower, as well.

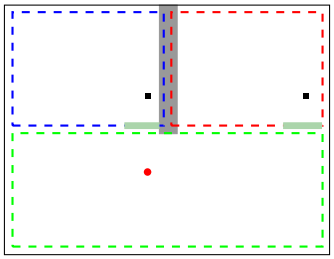


Figure 4.14: Port intervals (green shapes) preventing detours induced by the leader-follower approach for the example of Figure 4.13. As long as the ports are placed within the depicted intervals in the first pangea pass, copying the leader port position over to the follower does not enforce a large detour from the source to the sink.

by the user. After the first pass, the port configurations of the leader continents are copied onto their respective followers. Here, we keep the relative position of each port within the continent. To make sure the changed ports on the follower continents fit together with ports on the other continents, the entire pangea flow is run again. In this second pass the equivalent port positions of the reuse continents are given as replay input (see Section 4.5).

We now cover every step in more detail (Sections 4.6.3–4.6.7) and then proceed to discuss the limitations of this approach (Section 4.6.8) as well as extensions to deal with difficult continent layouts (Sections 4.6.9 and 4.6.10).

4.6.3 Port intervals

When a port configuration computed solely for the leader is copied exactly as is to the follower continents, this can produce large detours. See Figure 4.13 for an example. Note that this very example can be used to prove that (absolute) detours can in fact become arbitrarily long. To see this, simply scale the entire instance in x-direction.

To combat this problem, intervals for port positions are computed as a first step. The port intervals consider all equivalent continents instead of just the leader. For the example of Figure 4.13, good intervals are depicted in Figure 4.14.

In this section, we assume that the equivalent continents themselves are only shifted, but not rotated or mirrored. We do this so that the essential computations become clearer. The assumption can easily be dropped with some additional geometrical considerations.

We now define the Simple port interval problem. We call an axis parallel rectangle in $\mathbb{R}^2$ a *continent*. For a continent C of width W and height H and a point $p \in \partial([0, W] \times [0, H])$, we write $C + p \in \partial C$ for the component-wise sum of the lower left corner of C and p. Further, r_C denotes the projection of a point r into C. For two points $a, b \in \partial C$, we write $\text{dist}_{\partial C}(a, b)$ for the ℓ_1-length of a shortest path from a to b that does not go through the interior of C. Using this, we can define the Simple port interval problem.

Simple port interval problem

Instance: Disjoint continents $\mathcal{C}$ of the same height H and width W, a root $r \in \mathbb{R}^2 \setminus \bigcup_{C \in \mathcal{C}} C$ and sinks $T = (t_C)_{C \in \mathcal{C}} \subset \mathbb{R}^2$ so that $t_C \in C$ for all C and each sink lies inside the interior of its continent in the same relative position.

Task: Compute all points $p \in \partial([0, W] \times [0, H])$ minimizing

$$\max_{C \in \mathcal{C}} \text{detour}(r, p, t_C, C),$$

where

$$\text{detour}(r, p, t_C, C) \coloneqq \ell_1(r, r_C) + \text{dist}_{\partial C}(r_C, C + p) + \ell_1(C + p, t_C) - \ell_1(r, t_C).$$

Observe that $\text{detour}(r, p, t_C, C)$ represents the absolute detour of a path from r to t_C when entering C at $C + p$. When the instance is clear from the context, we simply write $\text{detour}(p)$.

To solve the Simple port interval problem, we consider piecewise linear functions $d_C : [0, 2W + 2H) \to \mathbb{R}^+$ representing the detour introduced by a point on the boundary of one particular continent. It is sufficient to take the pointwise maximum over $|\mathcal{C}|$ such functions and compute the minimum and its preimage of the resulting function. To simplify the notation, we define $p : [0, 2W + 2H) \to \partial([0, W] \times [0, H])$ as follows, visualized in Figure 4.15.

$$p(a) \coloneqq \begin{cases} (a, 0) & \text{if } a < W \\ (W, a - W) & \text{if } a \in [W, W + H) \\ (2W + H - a, H) & \text{if } a \in [W + H, 2W + H) \\ (0, 2W + 2H - a) & \text{if } a \in [2W + H, 2W + 2H). \end{cases}$$

Observe that p is well-defined and bijective. Further, for $a, b \in \partial([0, W] \times [0, H])$ it satisfies

$$\text{dist}_{\partial C}(a, b) = \min\{|p^{-1}(a) - p^{-1}(b)|, 2W + 2H - |p^{-1}(a) - p^{-1}(b)|\}.$$

For $C \in \mathcal{C}$, we can now define $d_C : [0, 2W + 2H) \to \mathbb{R}^+$ by

$$d_C(a) \coloneqq \text{detour}(r, p(a), t_C, C).$$

We know that d_C is piecewise linear, because all summands in the definition of $\text{detour}(-)$ are. Furthermore, the number of breakpoints is bounded from above by the number of Hanan grid points lying on ∂C, where the Hanan grid is induced by the corners of C as well as r and t_C. This is at most 8.

Using the above definitions, we can formulate an algorithm for the Simple port interval problem. Consider Algorithm 8. We construct the functions $(d_C)_{C \in \mathcal{C}}$ (Line 1). We then compute the point-wise maximum of these and call the result $d_{\max}$ (Line 2). Finally, we compute all points minimizing $d_{\max}$ and return their image under p (Line 3).

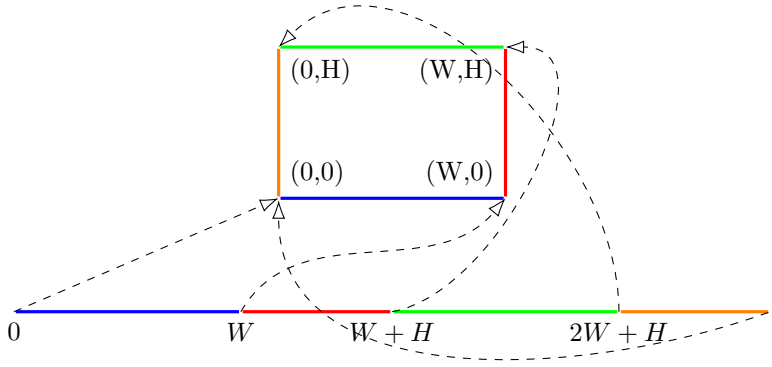


Figure 4.15: Visualization of the function $p : [0, 2W + 2H) \to \partial([0, W] \times [0, H])$.

Algorithm 8: Port interval computation

Input: $\mathcal{C}, H, W, r$ and $(t_C)_{C \in \mathcal{C}}$ as in the Simple port interval problem.
Output: A subset $I \subseteq [0, W] \times [0, H]$.
1 Construct the functions $d_C : [0, 2W + 2H) \to \mathbb{R}^+$ for all $C \in \mathcal{C}$
2 $d_{\max} : [0, 2W + 2H) \to \mathbb{R}^+ \leftarrow$ point-wise maximum of the d_C
3 **return** $p(\operatorname{argmin} d_{\max})$

Theorem 4.11. *Algorithm 8 solves the* Simple port interval problem *in* $\mathcal{O}(k \log k)$ *time, where* k *is the number of continents.*

Proof. We first show that the algorithm works correctly. For $C \in \mathcal{C}$, we have $d_C = \operatorname{detour}(r, -, t_C, C) \circ p$. Therefore

$$\begin{aligned} p\left(\operatorname{argmin}\, d_{\max}\right) &= p\left(\operatorname{argmin}\, \max_{C \in \mathcal{C}} d_C\right) \\ &= p\left(\operatorname{argmin}\, \max_{C \in \mathcal{C}} \left(\operatorname{detour}(r, -, t_C, C) \circ p\right)\right) \\ &= \operatorname{argmin}\, \max_{C \in \mathcal{C}} \operatorname{detour}(r, -, t_C, C) \end{aligned}$$

as required by the bijectivity of p.

Note that Line 2 of Algorithm 8 dominates the running time. We use the algorithm from [Bla+24] to compute the point-wise maximum of the d_C (Theorem 7 in [Bla+24]). This takes $\mathcal{O}(k \log k)$ time, because there are k functions, and each has at most a constant number of breakpoints. □

In this version of the problem, we assumed that the net has exactly one sink inside each equivalent continent and that the root lies outside. This can also be extended to deal with multiple sinks inside each continent.

For equivalent sources inside reused continents, there is the possibility of using multiple port intervals instead of only one. This can reduce the maximum detour significantly. The rest of this section explores one version of this problem.

We use the notion of a *border interval*, which is a set $I \subset \partial([0, W] \times [0, H])$ of the form $p([a, b])$ or $p([a, 2W + 2H)) \cup p([0, b])$. For a border interval $I = p([a, b])$, we

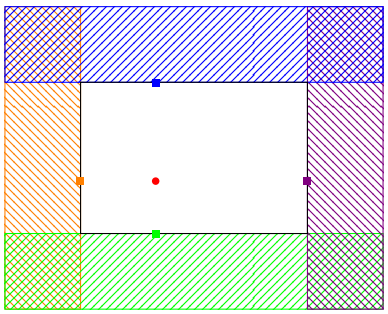


Figure 4.16: A solution of cardinality 4 to the MULTIPLE PORTS INTERVAL PROBLEM. The red circle represents the position of the root inside each continent (black rectangle). The colored squares represent the points of the four border intervals. Striped areas indicate which sinks can be routed to from the source with a shortest path when going through the border interval of the same color.

write $size(I) = b - a$, and for a border interval $I = p([a, 2H)) \cup p([0, b])$, we write $size(I) = 2H - a + b$.

MULTIPLE PORT INTERVALS PROBLEM

Instance: Disjoint continents $\mathcal{C}$ of the same height H and width W, roots $(r_C)_{C\in\mathcal{C}} \subset \mathbb{R}^2$ so that $r_C \in C$ for all C and each root lies inside the interior of its continent in the same relative position, sinks $(t_C)_{C\in\mathcal{C}} \subset \mathbb{R}^2 \setminus \bigcup_{C\in\mathcal{C}} C$, a maximum detour value $D \in \mathbb{R}^+$ and a maximum interval size $L \in \mathbb{R}^+$.

Task: Compute a set $\mathcal{I}$ of border intervals with size at most L of minimum cardinality, such that for all $C \in \mathcal{C}$

$$\text{detour}(r_C, \mathcal{I}, t_C, C) \leq D,$$

where, for x_C being the projection of t_C onto C,

$$\text{detour}(r_C, \mathcal{I}, t_C, C) := \min_{I\in\mathcal{I}} \min_{p\in I} \ell_1(r_C, C + p) + \text{dist}_{\partial C}(C + p, x_C) + \ell_1(x_C, t_C) - \ell_1(r_C, t_C).$$

We first observe that

$$\mathcal{I} = \{\{((x_C)_x - C_x, (x_C)_y - C_y)\} \mid C \in \mathcal{C}\}$$

is a solution of cardinality at most $|\mathcal{C}|$, where C_x and C_y denote the x- and y-coordinates of the lower left corner of C. This is because $C + p = x_C$ for $p = ((x_C)_x - C_x, (x_C)_y - C_y)$ and singleton border intervals have size 0. Further, for $C \in \mathcal{C}$, we have

$$\text{detour}(r_C, \mathcal{I}, t_C, C) \leq \ell_1(r_C, x_C) + \text{dist}_{\partial C}(x_C, x_C) + \ell_1(x_C, t_C) - \ell_1(r_C, t_C) = 0.$$

So it suffices to compute, for given $k \in \mathbb{N}$, a feasible set of k border intervals, if it exists.

Further, there always exists a solution $\mathcal{I}$ with $|\mathcal{I}| \leq 4$: As border intervals, we use the projections of the root position onto $\partial([0, W] \times [0, H])$ in all four axis-parallel directions. Remember that the roots lie in the same relative position inside each continent. Figure 4.16 depicts this solution as well as the areas served by each border interval.

Using these observations, we can now solve the MULTIPLE PORT INTERVALS PROBLEM for the special case $L = 0$. To simplify notation, we shift each continent and the

Algorithm 9: Computing multiple port intervals

Input: W, H, r, T and D as in the SIMPLIFIED MULTIPLE PORT INTERVALS PROBLEM.

1 Let P be the solution of the corresponding SIMPLE PORT INTERVAL PROBLEM
2 **if** *detour of the points in* P *at most* D **then**
3 **return** $\{p\}$ for any $p \in P$
4 $p_1, p_2, d :=$ **find-best-pair**(W, H, r, T) (Algorithm 10)
5 **if** $d \leq D$ **then**
6 **return** $\{p_1, p_2\}$
7 $p_1, p_2, p_3, d :=$ **find-best-triple**(W, H, r, T) (Algorithm 11)
8 **if** $d \leq D$ **then**
9 **return** $\{p_1, p_2, p_3\}$
10 **return** $\{\mathbf{n}, \mathbf{e}, \mathbf{s}, \mathbf{w}\}$

corresponding terminals such that the lower left corner of the continent becomes $(0, 0)$. Further, we directly consider the projections of sinks onto the continent border instead of the sinks themselves, because the detour does not depend on the exact sink position. We end up with the following problem.

SIMPLIFIED MULTIPLE PORT INTERVALS PROBLEM

Instance: Continent size $W > 0, H > 0$, a root position $r \in (0, W) \times (0, H)$, a finite set of sinks $T \subset \partial([0, W] \times [0, H])$ and a maximum detour value $D \in \mathbb{R}^+$.

Task: Compute a minimum cardinality set $I \subset \partial([0, W] \times [0, H])$ such that, for all $t \in T$,

$$\min_{p \in I} \ell_1(r, p) + \text{dist}_{\partial C}(p, t) - \ell_1(r, t) \leq D.$$

In the formulation of the following algorithms, we use the points $\mathbf{n}$, $\mathbf{e}$, $\mathbf{s}$ and $\mathbf{w}$ to denote the north, east, south and west projection points of r onto $\partial([0, W] \times [0, H])$, respectively.

The structure of the algorithm is presented in Algorithm 9. We try to find sets with an increasing number of points, such that the detour is at most D for all sinks: For $k \in \{1, 2, 3\}$, the algorithm checks if a solution of that size achieves a detour of at most D (Lines 1 and 2, Lines 4 and 5, and Lines 7 and 8). If so, the solution is returned (Lines 3, 6 and 9). Otherwise, we know that $\{\mathbf{n}, \mathbf{e}, \mathbf{s}, \mathbf{w}\}$ is an optimum solution, because it always achieves a detour of 0. So we return $\{\mathbf{n}, \mathbf{e}, \mathbf{s}, \mathbf{w}\}$ (Line 10).

Algorithms 10 and 11 compute the best solution using two and three port interval points, respectively. Here, *best* means lowest maximum detour.

In Algorithm 10, we order the sinks along the continent border. Let $q = |T|$. We then try every start point $i = 1, \ldots, q$ and end point $j = i, \ldots, q$ and define $T_1 := \{t_i, \ldots, t_j\}$ and $T_2 := T \setminus T_1$ (Lines 2-4). We now solve the SIMPLE PORT INTERVAL PROBLEM independently for T_1 and T_2 and store the result together with the corresponding maximum detour value (Lines 5 and 6). Finally, we return the best found solution (Line 7).

In Algorithm 11, we do the analogous but with indices $i = 1, \ldots, q-1$, $j = i, \ldots, q-1$ and $k = j, \ldots, q$ and the sink partition $T_1 := \{t_i, \ldots, t_j\}$, $T_2 := \{t_{j+1}, \ldots, t_k\}$ and $T_3 := T \setminus (T_1 \cup T_2)$.

To see that these algorithms are correct, we use the following observation.

Algorithm 10: find-best-pair

Input: $W > 0, H > 0, r \in (0, W) \times (0, H)$, and a finite set $T \subset \partial([0, W] \times [0, H])$.
Output: a pair of points in $\mathbb{R}^2$ and a detour value $d \in \mathbb{R}^+$.

```
Order T = {t_1, ..., t_q} as they appear going around the continent border
foreach i = 1, ..., q do
    foreach j = i, ..., q do
        Let T_1 := {t_i, ..., t_j}, T_2 := T \ T_1
        Solve the Simple Port Interval Problem for the sink sets T_1 and T_2 separately
        Store solution pair (p_1, p_2) with maximum detour value d
return stored triple (p_1, p_2, d) with minimum d
```

Algorithm 11: find-best-triple

Input: $W > 0, H > 0, r \in (0, W) \times (0, H)$, and a finite set $T \subset \partial([0, W] \times [0, H])$.
Output: three points in $\mathbb{R}^2$ and a detour value $d \in \mathbb{R}^+$.

```
Order T = {t_1, ..., t_q} as they appear going around the continent border
foreach i = 1, ..., q - 1 do
    foreach j = i, ..., q - 1 do
        foreach k = j, ..., q do
            Let T_1 := {t_i, ..., t_j}, T_2 := {t_{j+1}, ..., t_k}, T_3 := T \ (T_1 ∪ T_2)
            Solve the Simple Port Interval Problem for the sink sets T_1, T_2 and T_3 separately
            Store solution triple (p_1, p_2, p_3) together with maximum detour value d
return stored tuple (p_1, p_2, p_3, d) with minimum d
```

Proposition 4.12. *Let W, H, r, T and D be given as in the* Simplified multiple port intervals problem. *Let $I = \{p_1, p_2\} \subset \partial([0, W] \times [0, H])$ be a feasible solution, i.e. with detour value at most D.*

Consider an ordering of sinks $T = \{t_1, \dots, t_q\}$ going around the continent border. Then there are indices $1 \le i \le j \le q$ such that, for the sets $T_1 := \{t_i, \dots, t_j\}$ and $T_2 := T \setminus T_1$, we have

$$\forall t \in T_1 : \quad \ell_1(r, p_1) + dist_{\partial C}(p_1, t) - \ell_1(r, t) \le D, \tag{4.1}$$

$$\forall t \in T_2 : \quad \ell_1(r, p_2) + dist_{\partial C}(p_2, t) - \ell_1(r, t) \le D. \tag{4.2}$$

The analogous statement holds for three instead of two points.

Proof. Consider the following function representing the detour when going through the fixed point p_1.

$$f : \partial([0, W] \times [0, H]) \to \mathbb{R},$$
$$f(t) = \ell_1(r, p_1) + \mathrm{dist}_{\partial C}(p_1, t) - \ell_1(r, t).$$

Then the set $\{t \in \partial([0, W] \times [0, H]) \mid f(t) \le D\}$ is connected.

So we can define $T_1 := \{t \in T \mid f_1(t) \le D\}$ and choose i and j accordingly. Since the solution $\{p_1, p_2\}$ is feasible, we know that (4.2) holds for all $t \in T \setminus T_1$. But $T \setminus T_1 = T_2$. □

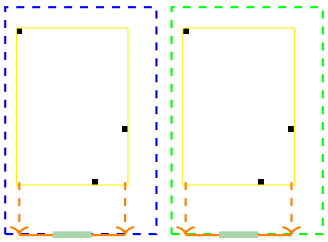


Figure 4.17: The steps to compute port intervals in practice. The sinks are projected into one direction (south in this example). The resulting interval is depicted in orange. Afterwards, the port intervals (green shapes) are obtained by building an interval around the center.

Algorithm 12: Port interval computation in practice

Input: A continent C, sinks $T \subset C$, a direction $D \in \{\texttt{north}, \texttt{south}, \texttt{west}, \texttt{east}\}$ and a constant $W > 0$.
Output: An interval $I \subset \partial C$.

1 $B \leftarrow$ bounding-box of T
2 $I' \leftarrow$ projection of B onto the D side of ∂C
3 $c \leftarrow$ center of I'
4 **return** $I \leftarrow \frac{W}{2}$-ball around c inside I'

Computing port intervals in practice

While it is not possible to guarantee short source-to-sink paths with only one ingoing port in the reuse scenario, the instances occurring in practice allow for simple, straight forward solutions. It is often the case that a fixed continent border edge contains the best port interval. This happens for example when several CPU cores are arranged side by side next to a backbone.

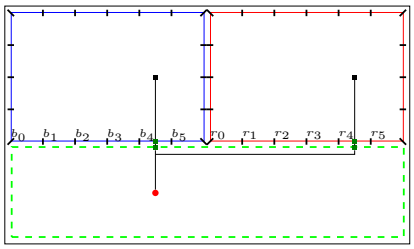
In this case, one direction out of `north`, `south`, `west` and `east` can be chosen per (non-trivial) continent equivalence class by the user, representing the relevant border edge. Now, for a set of sinks T inside one continent, their bounding box is computed and projected onto the continent boundary in that direction. The center of the resulting interval then is the point minimizing absolute detour (across all root placements).

To give global routing more flexibility in avoiding congestion, the computed point is extended to an interval of fixed width along the continent border. This setting is depicted in Figure 4.17 and the computation can be seen in Algorithm 12.

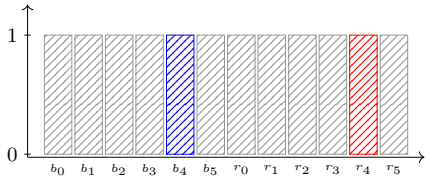
Port customers

Another approach to computing port intervals is to incorporate the issue into the resource sharing framework (see Section 2.4.1). We now briefly sketch an idea how that might be done which was developed together with Max Mundt. It is presented in [Mun23] in more detail. In this section, we consider the case of a net with its source outside of the reuse continents and a single sink inside each reuse continent of an equivalence class. For sets of sinks or equivalent sources inside the reuse continents, the approach can be applied analogously.

For a set of equivalent sinks $(t_C)_{C \in \mathcal{C}}$, we do the following: We add a customer p, called the *port customer*. The solution of the port customer determines where the associated port interval lies. As possible choices for the port intervals, we divide the border of each reuse continent (equivalently) into segments of fixed size. Call the set of all border segments (over the equivalent continents) $\mathcal{S}$. Then the allowed solutions



(a) An example route of a net with two equivalent sinks. The segments are marked on the two equivalent continents (blue and red) and the southern segments are labeled.



(b) Resource usage associated with the solution on the left. The port customer blocks all segments but b_4 and r_4 (gray blocks) and the net uses exactly these two resources by crossing the continent borders in the associated segments (blue block for the blue crossing, red block for the red crossing).

Figure 4.18: Packing in the presence of port customers.

for p are the equivalence classes of segments in $\mathcal{S}$. Further, we introduce one *segment resource* r_S per segment $S \in \mathcal{S}$. A solution of the port customer uses up all of these resources except for the equivalent segments in the solution, i.e. for an equivalence class $\bar{S}$ of segments, we have

$$usg_{r_S}(\bar{S}) = \begin{cases} 0 \text{ if } S \in \bar{S} \\ 1 \text{ else.} \end{cases}$$

A route Y for a net in question uses up the segment resource associated with the position where the route crosses the reuse continent borders:

$$usg_{r_S}(Y) = \begin{cases} 1 \text{ if } Y \text{ goes through } S \\ 0 \text{ else.} \end{cases}$$

Note that the route is allowed to enter each reuse continent at most once. This way, an integral packing observing resource capacities has all equivalent continent crossings of this net within the same segment as portrayed in Figure 4.18.

A slightly different way to define segment resources and their usages is as follows. Instead of introducing a resource for each segment for each copy of the continent, we only add one resource $r_{\bar{S}}$ for each segment equivalence class $\bar{S}$. The port customer uses up all segment resources except for one:

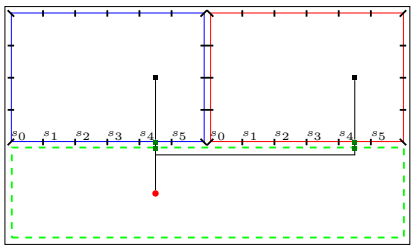
$$usg_{r_{\bar{S}}}(\bar{S}') = \begin{cases} 0 \text{ if } \bar{S}' = \bar{S} \\ 1 \text{ else.} \end{cases}$$

Assuming there are k equivalent continents, a route Y uses $\frac{1}{k}$ of the capacity of the resource belonging to the segment where the border is crossed:

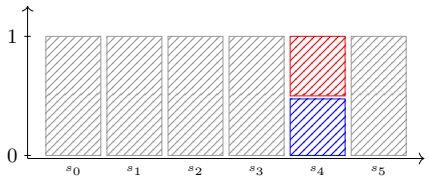
$$usg_{r_{\bar{S}}}(Y) = \begin{cases} \frac{1}{k} \text{ if } Y \text{ goes through some } S \in \bar{S} \\ 0 \text{ else.} \end{cases}$$

For the same scenario as in Figure 4.18a, the associated segments and packing with regard to this definition are depicted in Figure 4.19.

The second approach has the advantage of adding significantly fewer resources. In terms of the integral packing, both methods are identical: The only feasible integral solution is for all paths to cross equivalent segments, namely the segments not blocked by the port customer. However, the second approach has a drawback that comes to light when considering what happens during the resource sharing algorithm discussed in Section 2.4.1:



(a) An example route of a net with two equivalent sinks. The segments are marked on the two equivalent continents (blue and red) and the southern segments are labeled.



(b) Resource usage associated with the solution on the left. The port customer blocks all segments but s_4 (gray blocks) and the net uses exactly this resource by crossing through the associated segment in both reuse continents (red and blue blocks).

Figure 4.19: Packing in the presence of port customers. The segment resources exist only once as opposed to once per copy of the reused continent.

Assume there is a separate net for each reuse continent. The nets have their sources outside of the reuse continents and one sink inside one of the reuse continents each. Assume the net customers are solved first and the port customers afterwards. Since resource prices rise whenever a resource is being used, one net being routed through a segment s would lead to the price of s increasing. Thus, the subsequent routes would avoid routing through the same segment. This effect leads to a slow convergence in the resource sharing algorithm.

4.6.4 Blockages on the continent border

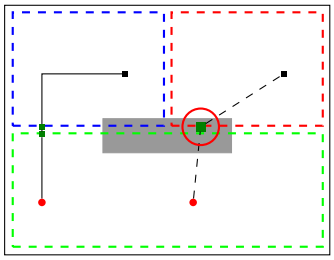
The essence of the reuse flow in BonnPangea is the leader-follower approach. Port positions of one of the reuse continents (the leader) are copied over to the others (the followers). For the port positions of the leader continent to be feasible when copied over to the followers, they must not be placed on a routing blockage. A route might, legally, go from a source outside to a sink inside the leader continent. The location where that route crosses the border of the leader continent determines the port position not only for the leader, but also the follower continents. Hence it has to obey blockages not only inside or close to the leader, but also those placed near the followers.

An example scenario is depicted in Figure 4.20. The routing of the left net shown in Figure 4.20a would lead to the marked port placement for the right net. This is infeasible as there is a blockage surrounding the port position. Instead, as indicated in Figure 4.20b, the part of the blockage that is inside or close to the follower continent must also be taken into account inside the leader. Then, a route can be found using the non-blocked areas according to both equivalent continents. This can be done using simple geometric operations.

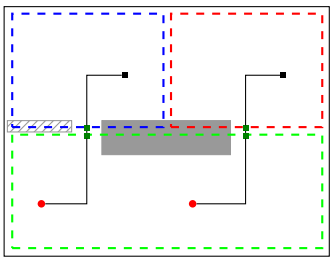
In practice, all blockages touching a continent inside a non-trivial equivalence class are copied into all equivalent continents, instead of just copying from the followers to the leader. This has the additional benefit of aligning the two routing passes as much as possible.

4.6.5 First routing pass

After computing port intervals and dealing with blockages on continent borders, a first routing pass is performed. The port intervals and additional blockages from Sections 4.6.3 and 4.6.4 are observed. Regarding everything else, the first routing pass is conducted as described in Section 4.4.



(a) The route of the left net in itself is legal. But copying the port position over into the red continent leads to a violation as can be seen by the marked port inside the blockage.



(b) The part of the blockage inside and close to the red continent was transferred into the blue continent, the striped area. This way, the route of the left net is still legal when copied over to the right net.

Figure 4.20: A reuse scenario in which it is necessary to consider not only the blockages directly near the continent where the route is placed, but also the blockages near equivalent continents. The blue and red continents are equivalent, the blue continent is the leader.

Algorithm 13: Applying continent equivalence

1 Delete port graph parts outside of reuse continents
2 Copy feed-throughs into equivalent continents
3 Align subway sinks driven by equivalent net sources
4 Align subway sources driving equivalent net sinks
5 Align subway sinks driven by equivalent subway sources

Due to the leader-follower approach, only port locations on the leader continent of each non-trivial equivalence class will remain in the end. Hence, it is an option to only consider nets crossing the boundary of such a leader continent in the first routing pass. While this would speed up the flow (even drastically depending on the distribution of nets), it could also have a negative impact on routing congestion.

Fixing all leader port locations in this first routing pass can lead to violations later on. Such scenarios and how to avoid them are discussed in Section 4.6.8.

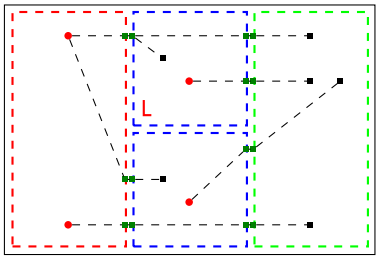
4.6.6 Applying continent equivalence

After the first routing pass, the leader port locations need to be translated and copied over to the follower continents. This is done according to the continents' translating functions (see Definitions 4.8 and 4.9). All the computation is done on the port graphs produced from the first routing pass.

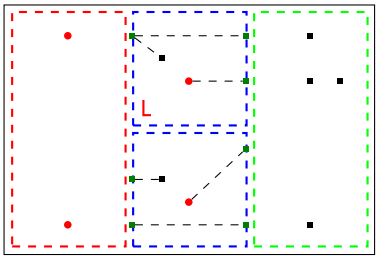
For most nets, this is rather straight forward. However, care has to be taken in situations where ports of a follower continent need to be combined with those of the leader, instead of simply replaced. The general algorithm is described in Algorithm 13. See Figure 4.21 for a visualization.

First, all ports and adjacent edges in the port graphs that are outside of a reuse continent are deleted. These port positions are computed again from scratch in the second routing pass.

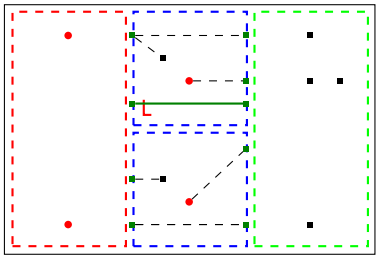
Then, feed-throughs through reuse continents are collected and copied into the equivalent continents. Here, a feed-through is a connected component (after Step 1) of a port graph that does not contain a net source or net sinks. To copy such a component into an equivalent continent, the continent transformations are applied to all relevant port positions. As the super-set of all feed-throughs in an equivalence class might not be overlap-free in general, it needs to be legalized. This can be done using track as-



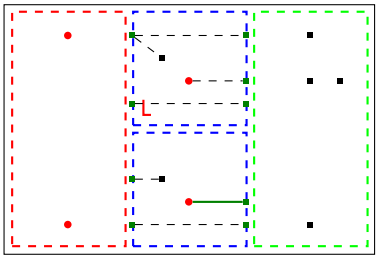
(a) Initial port graphs from first routing pass.



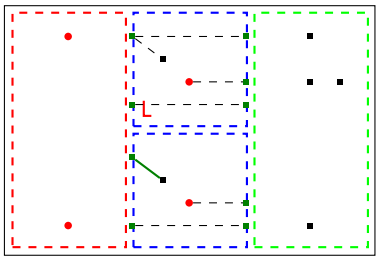
(b) Everything outside reuse continents is deleted.



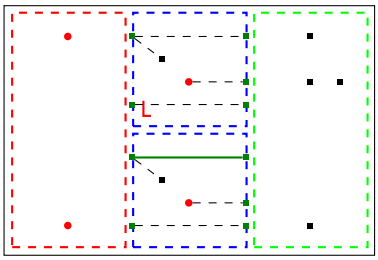
(c) Feed-throughs are copied into equivalent reuse continents.



(d) Subway sinks connected to equivalent net sources are aligned.



(e) Subway sources driving equivalent net sinks are aligned.



(f) Subway sinks connected to equivalent subway sources are aligned.

Figure 4.21: Rough steps of Algorithm 13. The blue continents are equivalent, the top one being the leader. New/changed edges compared to the previous step are shown in green.

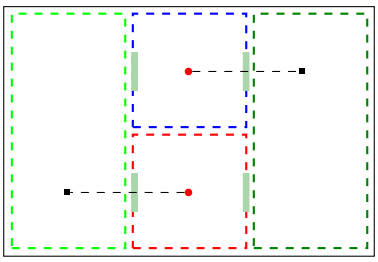


Figure 4.22: Reuse situation in which adopting the leader's port for the follower continent would lead to a large detour. The blue and red continents and the shown sources are equivalent, the blue continent is the leader. In cases where leader and follower ports are on different sides of the continent, it might be better to introduce additional ports.

signment, but was not implemented so far as it was not needed in practice. More on optimizing feed-throughs is discussed in Section 4.6.10.

Aligning subway sinks driven by equivalent net sources

Now, the connected components containing a net source are considered. Assume we have an equivalence class of net sources $\{r_0, \ldots, r_k\}$, where r_0 lies inside the leader continent. The connected components of the port graphs containing these sources are star-shaped with the net sources as center and out-going edges to net sinks inside and subway sinks leaving the respective reuse continent. In this scenario, the subway sinks need to be equated.

In most cases, it is a good solution to simply delete the subway sinks of the follower continents and adopt the subway sinks of the leader continent. However, this could force large detours on paths starting at a follower's source. In particular, this happens if the leader continent has no port inside one of the precomputed port intervals. An example for this is depicted in Figure 4.22. Here, two port intervals were computed due to the fact that the reuse continents can be routed to from the west or east side. However, the leader's net only needs the east port interval as it has no sink on the west side. To avoid these kinds of situations, additionally one follower port is used on each side on which the leader does not have a port (if such a follower port exists). To minimize net length, this port is chosen so that it minimizes its distance to the source.

As a variant, we can also enforce that each out-going port interval is routed to by every continent. This is done by adding the port intervals as dummy sinks to the associated nets. We need this later in Section 4.6.9, Algorithm 14.

Aligning subway sources driving equivalent net sinks

For equating connected components that contain net sinks but no net source, two steps are necessary: The subway sources need to be aligned. This induces an equivalence relation on them, enabling us to also align the subsequent subway sinks.

Note that a connected component of a port graph without a net source is also star-shaped, with a subway source as its center and out-going edges to net sinks and subway sinks. Here, we assume that the netlist is *equivalence compliant*, i.e. that two sinks inside a reuse continent are in the same net if and only if the equivalent sinks inside equivalent continents are in the same net. We revisit this definition in Section 4.6.8. Further, we assume that the first routing pass fulfills the requirement that there is only one port entering each continent that is connected to net sinks inside that continent. This makes aligning the port graphs much simpler.

For a deeper consideration on possible issues if this is not the case, have a look into Section 4.6.8, specifically the examples in Figure 4.23 and Figure 4.24. So assuming the connected component at hand is nice in the above sense, there is a connected port graph component with an equivalent set of sinks for every equivalent continent. We mark the subway sources of these connected components as equivalent and align the follower's subway sources with the leader. This can lead to large detours similarly to aligning subway sinks. To see this, consider the example of Figure 4.22, but switch the roles of source and sink. However, it is not feasible to introduce additional subway sources, as this would create a multi-source net requiring significant changes to the logic.

Aligning subway sinks driven by equivalent subway sources

The last step is to align subway sinks whose predecessors are equivalent. This is done the same way as aligning subway sinks driven by equivalent net sources. Potentially, new subway sinks can be added to the leader in this case, as well.

4.6.7 Second routing pass

After applying the continent equivalence to the connected components of the port graphs, a second routing pass is done. This routing pass gets the partial port graphs adjusted as in Section 4.6.6 as replay input.

For simple instances, this is all that needs to be done and the result of the second routing pass is legal and reuse-aware. However, many scenarios are possible in which this routing has no feasible solution obeying the replay input. These are explored in the following section.

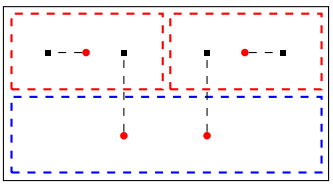
4.6.8 Difficult and reuse incompatible input

In Pangea ReUse, there are some implicit requirements to the input. When the continent layout or the netlist are not *nice* in some sense, it is not possible to fulfill all Pangea ReUse constraints, even when neglecting wiring congestion. However, it is not trivial to formulate necessary and sufficient conditions for an instance to be feasible.

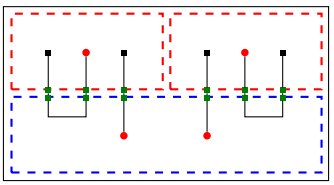
This section compiles different examples of possible input to Pangea ReUse that lead to problems. In doing so, we develop criteria for the netlist and continent placements indicating the input is or is not reuse compatible. Such criteria can especially be useful as a first check when trying to discern whether a specific design is eligible for using the reuse mode of BonnPangea, or which changes need to be done to the netlist or continent layout to make it eligible.

In this section, all figures abide by the following conventions.

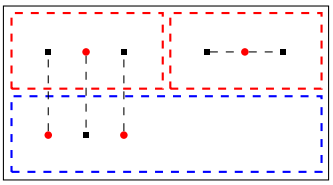
- Continents are equivalent if and only if they have the same color.
- Equivalent continents are not rotated/mirrored unless indicated by the orientation of a gray F-shape.
- Sources are depicted by red circles.
- Sinks are depicted by black squares.
- Ports are depicted by darkgreen squares.
- A net is depicted by dashed black lines from the source to each sink.
- Routes are represented by solid black lines.



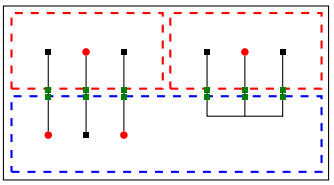
(a) Example for equivalent sources connected to non-equivalent sinks.



(b) Solution for the netlist in Figure 4.23a. Two nets leave a continent and then route back into it.



(c) Example netlist forcing reuse-aware routes to enter a continent at least twice.



(d) Solution for the netlist shown in Figure 4.23c. The right route enters the right red continent twice.

Figure 4.23: Examples of non-equivalent net connectivities and reuse-aware routings.

Bad continent border alignment

So that global routing and track assignment can work together cleanly with regards to equivalent port positions, it is necessary that the tiling of the global routing graph is identical within equivalent continents. This means that translating the area of a tile inside a continent using the translation data must exactly give the area of another tile inside the equivalent continent.

For a general continent placement, this is not easy to achieve, as proven in [Mun23]. However, practical instances mostly have equivalent continents aligned such that at least one coordinate of a reuse continent is shared with an equivalent continent.

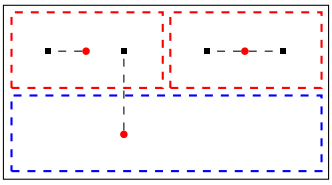
Different connectivities inside equivalent continents

So far, we assumed that the netlist induces equivalent constraints inside equivalent continents. In practice, this condition might be partially violated. We now study netlists which are problematic in this regard.

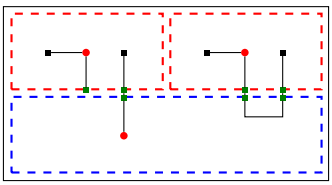
Figure 4.23a depicts an example in which equivalent sources must be connected to non-equivalent sinks inside their own continent. At first glance, this already looks infeasible. But there is a feasible solution (i.e. a reuse-aware routing), as shown in Figure 4.23b. However, this solution (and every other) has large detours. In fact, it is not possible to route the internal nets of the red continents within their respective continent and remain reuse-aware.

As long as there is enough space outside of the reuse continents, every netlist can be routed in a reuse-aware way by connecting the sources directly to subway sinks and the sinks directly to subway sources. However, in addition to producing large detours, this can violate the constraint of entering every continent only once. The instance shown in Figure 4.23c is an example for this: While it can be routed, as depicted in Figure 4.23d, there is no solution with only one ingoing port into the right-hand red continent, even though the continent contains only one net.

Remark. Usually, the pangea flow ignores nets completely contained inside individual continents. For simplicity, this detail is neglected in the examples of this section. To make sure all nets cross a continent boundary, we could add an additional sink inside the blue continent to every net in the examples of Figure 4.23.



(a) Example netlist that is not equivalence compliant, but only weakly equivalence compliant.



(b) Reuse-aware routing for the netlist shown in Figure 4.24a. All nets enter every continent at most once.

Figure 4.24: Example of a netlist that is not equivalence compliant yet still allows for a reuse-aware solution entering every continent at most once.

One possibility to avoid such input is to require the netlist to be identical within equivalent continents.

Definition 4.13. Consider a netlist $\mathcal{N}$ and an equivalence relation on its pins $\sim_p$ induced by the continent equivalence. Within each continent C, define an equivalence relation $\sim_C$ on the sinks inside C by

$$s_1 \sim_C s_2 \iff \exists N \in \mathcal{N}\ s_1, s_2 \in N.$$

We say that the netlist $\mathcal{N}$ is *equivalence compliant* if whenever we have equivalent continents C and C' and sinks $s_1, s_2 \in C$, $s_1', s_2' \in C'$ with $s_1 \sim_p s_1'$, $s_2 \sim_p s_2'$, we have

$$s_1 \sim_C s_2 \iff s_1' \sim_{C'} s_2'.$$

For equivalence compliant netlists, it can never happen that a solution is forced to enter a continent twice due to the reuse constraints (again assuming there is enough routing space outside of the reuse continents). However, being equivalence non-compliant does not imply that all solutions enter a continent twice. An example for this can be found in Figure 4.24.

Note that the reason there is a feasible solution for that instance is that the left sink within each reuse continent is connected directly to the root within the same continent. To formalize this, we need the following definition.

Definition 4.14. Call a sink s inside of a reuse continent an *inner* sink, if the root r of s is inside the same continent as s, and for each equivalent sink s' of s, the root of s' is the pin equivalent to r inside the continent of s'.

Observe that whenever s is an inner sink, all sinks equivalent to s are also inner sinks. For reuse-aware routings entering each continent at most once to exist, it is sufficient to require equivalence compliance on sinks that are not inner.

Definition 4.15. Take $\mathcal{N}, \sim_p$ and $\sim_C$ for each reuse continent C as in the definition of equivalence compliance.

We say that the netlist $\mathcal{N}$ is *weakly equivalence compliant* if whenever we have equivalent continents C and C' and sinks $s_1, s_2 \in C$, $s_1', s_2' \in C'$ with $s_1 \sim_p s_1'$, $s_2 \sim_p s_2'$ and none of these sinks are inner sinks, we have

$$s_1 \sim_C s_2 \iff s_1' \sim_{C'} s_2'.$$

Being weakly equivalence compliant now exactly characterizes a netlist allowing for a routing that enters every continent at most once:

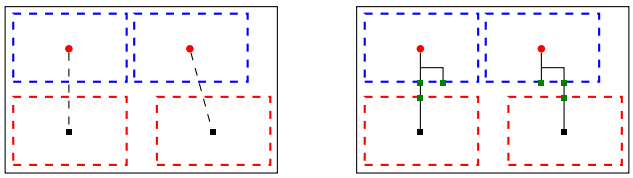


Figure 4.25: Example world and netlist in which opposing ports of equivalent ports are not equivalent. This instance has a feasible solution, but only by using two distinct subway sinks inside each blue continent.

Theorem 4.16. *Consider a netlist $\mathcal{N}$. Assume the reuse continents are spread far enough apart to allow for arbitrary routing connections in between. Then there exists a reuse-aware routing entering every continent at most once if and only if $\mathcal{N}$ is weakly equivalence compliant.*

Proof. First assume there is such a routing. Take equivalent continents C and C' and non-inner sinks $s_1, s_2 \in C$, $s_1', s_2' \in C'$ with $s_1 \sim_p s_1'$, $s_2 \sim_p s_2'$. Observe that since the sinks are not inner sinks, none of them are directly connected to a root within their continent: Otherwise this would have to be true for all equivalent sinks, contradicting the fact that they are not inner sinks by the reuse-awareness of the routing. Now if s_1 and s_2 are inside the same net N, there is a single subway source connected to s_1 and s_2, whose equivalent subway source in C' witnesses $s_1' \sim_{C'} s_2'$. This means that $s_1 \sim_C s_2 \Rightarrow s_1' \sim_{C'} s_2'$. The analogous works for the other direction.

Now assume $\mathcal{N}$ is weakly equivalence compliant. Connect all inner sinks to the respective roots within their continents. By the definition of inner sinks, this produces a reuse-aware routing. For the rest, construct a single subway source $p_{N,C}$ for each net N and each continent C in which N has sinks. Connect $p_{N,C}$ to all sinks of N inside C. By weak equivalence compliance of the netlist, this construction is the same within every equivalent continent. So the routing remains reuse-aware. This routing can be extended without adding additional subway sources. Hence in the end, there is at most one subway source entering each continent per net. □

Shifted equivalent continents

In practice, neighboring continents are usually close enough together that no routing is allowed along the gap in-between, see Section 4.4.1. As a result, ports connected across this gap have to align perfectly. When there are multiple non-trivial continent equivalence classes, this is cause for more difficulty.

It can happen that multiple subway sinks are needed even for two-terminal nets, i.e. for nets with only a single sink. A simple example showcasing this can be found in Figure 4.25. The upper continents are close enough to the lower continents so that no horizontal routing is allowed in between. However, the right red continent is shifted relative to the blue one, compared to the red and blue continents on the left. In the routing shown on the right picture, two equivalent subway sources into the red continents are depicted. They each align to a blue subway sink, but not to equivalent ones.

This is why, while such instances can have feasible solutions, it is much harder to find them algorithmically. Further, the need for two subway sinks per blue continent can lead to a significant increase in congestion and thus might lead to non-routability.

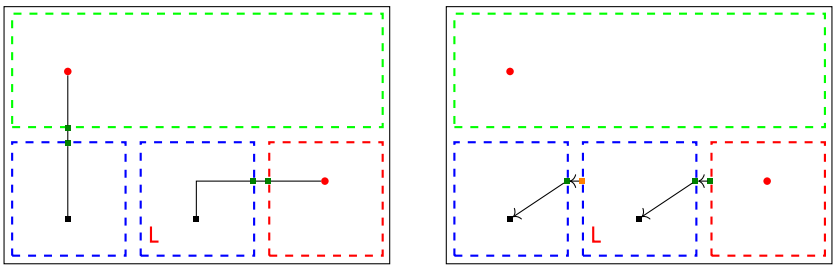


Figure 4.26: Example where neighboring continents are equivalent. The left picture shows the result of the first routing. The right picture shows the result of copying the leader port to the follower. It also shows the necessary subway sinks opposite of the subway sources. One of these subway sources (highlighted in orange) needs to be created inside the leader continent (right blue).

Alignment function

To formalize the additional constraints posed by neighboring continents, we consider an *alignment function* as additional input:

$$f_{opp} : \bigcup_{C \in \mathcal{C}} \partial C \to \bigcup_{C \in \mathcal{C}} \partial C \cup \{\square\}$$

f_{opp} must satisfy $f_{opp}(f_{opp}(a)) = a$ for all $a \in \bigcup_{C \in \mathcal{C}} \partial C$ with $f_{opp}(a) \neq \square$. We interpret $f_{opp}(a) = \square$ as meaning that there is no continent border opposite of the point a. $f_{opp}(a) = b \in \bigcup_{C \in \mathcal{C}} \partial C$ denotes that the point a lies opposite of the point b. This implies the following.

a) A subway sink at position a has at most one child in a port graph. If the child is a subway source, it must lie at position b.

b) If a subway source at position a has a predecessor subway sink, this sink must lie at position b.

We call a pair of subway sink and subway source at positions a, b *opposite ports* if $f_{opp}(a) = b$ and $f_{opp}(b) = a$.

Equivalent neighboring continents

When equivalent continents are neighboring each other, copying port positions from the leader to a follower can necessitate new ports in the leader continent.

An example is shown in Figure 4.26. When the leader's subway source is copied to the follower continent, we need to create the opposing port marked in orange. Otherwise, the shown sink in the follower continent cannot be routed to at all. But this position might already be occupied by another port.

This problem can be avoided by placing full blockages (on all layers) in between equivalent continents. See Figure 4.27 for an example. But note that this can, by the copying of blockages on equivalent continent borders (see Section 4.6.4), lead to drastic restrictions of the routing space.

Equivalence compliance with respect to opposite ports

In the presence of constraints given by an alignment function, we need to extend the notion of equivalence compliance of the netlist. Otherwise, there are instances without feasible solution.

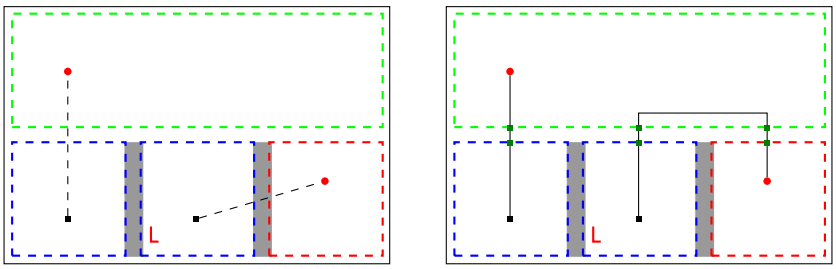


Figure 4.27: Same instance as in Figure 4.26, but with additional blockages to prevent ports between equivalent continents. The blockage between the right blue and the red continent arises from the blockage copying covered in Section 4.6.4. The left picture shows the netlist, the right picture shows a feasible solution.

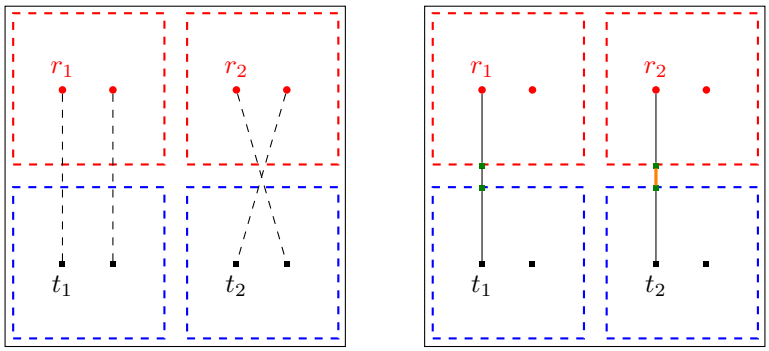


Figure 4.28: Example in which the alignment constraints of opposing ports make the instance infeasible. The example instance is shown on the left. No horizontal routing is allowed in between the red and blue continents due to the border blockages. A partial solution is indicated on the right: Two connected opposing subway ports are needed to route the net from r_1 to t_1. Consider the equivalent subway ports on the right. If they are also connected (indicated in orange), r_2 is connected to t_2. Otherwise, nothing can route to t_2 at all. Both cases are not feasible.

One example for this can be seen in Figure 4.28. Consider the leftmost net going from r_1 to t_1. Although r_1 and t_1 are not in the same continent, every reuse-aware routing connecting them also connects the equivalent pins r_2 and t_2 in the right red and blue continents, or leaves t_2 unconnected.

Nice instances

To summarize the conditions discussed above, we define when an instance is *nice*. An instance being nice implies that

a) equivalent continents are not shifted relative to neighboring equivalent continents (as in Figure 4.25),

b) equivalent continents at not neighboring according to f_{opp}, and

c) scenarios as in Figure 4.28 are forbidden.

Definition 4.17. An instance of the PANGEA REUSE ROUTING PROBLEM (Section 4.6.1) is a *nice instance* if the following holds.

a) Let a, b be geometrically equivalent border positions. Assume $f_{opp}(a) = a' \neq \square$ and $f_{opp}(b) = b' \neq \square$, and a', b' are inside the same non-trivial continent equivalent

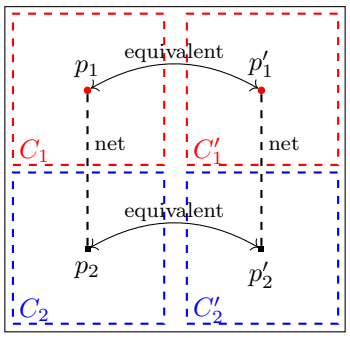


Figure 4.29: Visualization of the third constraint in Definition 4.17. C_1 has equivalent border positions to C_2 and C'_1 has equivalent border positions to C'_2. Therefore, the net from p_1 to p_2 implies the existence of a net from p'_1 to p'_2.

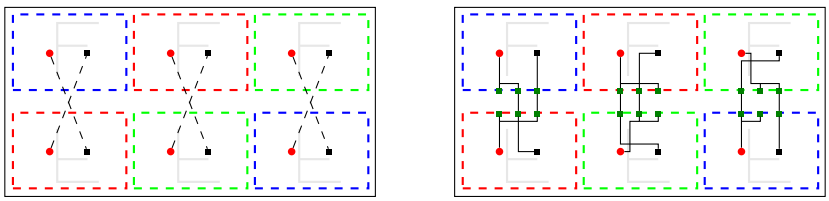


Figure 4.30: Example continent placement and netlist (left). A feasible routing is shown on the right. The continent rows are close enough together that no horizontal routing is allowed in between. Observe that there is no solution with only one subway sink per continent, even though this is a nice instance.

class. Then the positions a', b' are in distinct continents and also geometrically equivalent.

b) For equivalent continents C_1 and C_2, there are no border positions $a \in \partial C_1$ and $b \in \partial C_2$ with $f_{opp}(a) = b$.

c) For pairs of equivalent continents C_1, C'_1 and C_2, C'_2 such that C_1 has opposite border positions to C_2 and C'_1 has opposite border positions to C'_2, we have the following (see also Figure 4.29). A net with a pin p_1 in C_1 and another pin p_2 in C_2 implies the existence of a net containing the equivalent pin to p_1 in C'_1 and the equivalent pin to p_2 in C'_2.

Consider the world and netlist shown on the left in Figure 4.30. This is a nice instance as defined above. A feasible reuse routing is depicted on the right in Figure 4.30. But this solution cannot be found by the algorithms considered so far. A visualization of what goes wrong with the simple leader-follower approach can be found in Figure 4.31: In the first routing pass, connected ports are opposing correctly. But after copying port positions from leader to follower, the resulting ports do not align. So no feasible replay routing exists.

Observe that no choice of leader continents leads to such a routing being computed in the 2-pass routing flow: By symmetry, we can assume without loss of generality that the left blue continent is the blue leader. If the left red continent is the red leader, the port positions between the blue and red continents of the left-most two nets are copied over to the middle and right nets. However, the port positions of the green continents depend on those computed for the green leader and do not necessarily match up with the blue and red ones. On the other hand, when the right red continent is chosen as

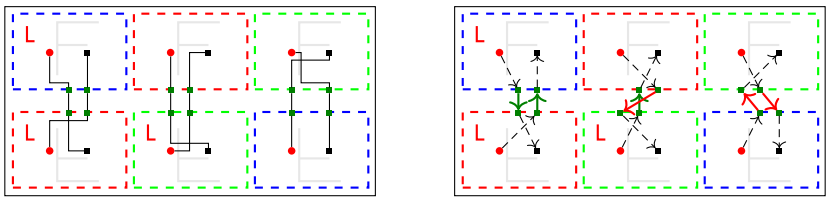


Figure 4.31: An example where the pure leader-follower approach with two routing passes does not suffice when considering alignment of opposite ports. The left picture shows the first routing. The right picture shows the result of copying leader ports and their connections over to the respective follower continents. Non-opposite ports needing to be connected are marked with a red arrow. Correctly opposing ports are connected with a green arrow.

the red leader, the port positions of the red leader are copied onto the red follower on the left. They, in turn, do not necessarily match up with the port positions in the blue leader on the left. This poses a problem even in the scenario where multiple outgoing ports are allowed on the same side of a continent.

The following section presents an improvement upon the 2-pass routing. The resulting algorithm can solve such instances taking the alignment of opposing ports into account.

Remark. Consider the case where the top and bottom continent rows are touching in vertical direction in the instance of Figure 4.30. Then ports of the same x-coordinate on the abutted continent borders are connected automatically. In this case, the depicted netlist does not allow for any reuse routing at all:

The sinks in the green continents must be connected via equivalent subway sources. The position of this subway source forces the subway sinks of the corresponding nets in the right red and blue continents to be at the very same location. Now consider the equivalent subway sinks in the equivalent left red and blue continents. They have the same x-coordinate, hence they, and their routing, connect the two left-most net sources. This is not allowed. Examples like this are one reason why non-abutting continents might be advantageous in practice.

4.6.9 Alignment of opposite ports

As discussed in the previous section, opposite ports of neighboring continents have to align perfectly. In the standard pangea mode, the routing makes sure of this automatically. However, in the reuse flow, port positions are not only given directly by routing segments, but also by port positions of leader continents. Hence we need to consider port alignment explicitly. A visualization of what can happen otherwise was already shown in Figure 4.31. This section presents a multi-pass algorithm computing a reuse-aware routing that obeys given alignment constraints.

To find such a routing solution, we need to handle subway sources and subway sinks differently. This is because we might need to create multiple subway sinks (see Figure 4.30), while having multiple subway sources is infeasible.

We use the concept of a *port reservation*, which contains the same information as a subway source and is associated with a net. We say that a routing *respects* a given port reservation p associated with a net N if no wire belonging to a different net than N entering the continent of p is placed overlapping p (respecting spacing as if p were a subway source). This implies that no subway source of a different net created from the routing will be in conflict with the subway source induced by the port reservation. Wiring of the associated net is allowed to overlap, as well as all wiring leaving the continent.

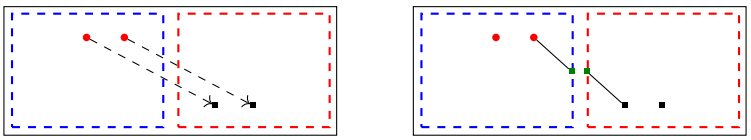


Figure 4.32: Example scenario of a blocked subway source. The left picture shows the netlist. The right picture shows possible port positions creating a conflict. The subway sink in the blue continent prevents all routing to the subway source in the red continent.

Algorithm 14: Refined reuse flow

Input: An instance of the PANGEA REUSE ROUTING PROBLEM and an alignment function f_{opp}.

1 run a routing pass, store resulting port graphs in R
2 $R_{reuse} \leftarrow$ empty set of port graphs
3 $Q \leftarrow$ empty set of port reservations
4 **while** R *not reuse-aware* **do**
5 pick continent equivalence class $\overline{\mathcal{C}}$ for which R is not reuse-aware and that was not chosen before
6 align ports in $\overline{\mathcal{C}}$ according to leader (Algorithm 13) and add result to R_{reuse}
7 **foreach** *added subway source p in Line 6 with* $f_{opp}(\pi(p)) \neq \square$ **do**
8 add subway sink p' at $f_{opp}(\pi(p))$ and edge (p', p) to R_{reuse}
9 **if** *port graph has subway/net source s in continent of p'* **then**
10 add edge (s, p') to R_{reuse}
11 copy new subway sink p' to equivalent continents (Algorithm 15)
12 **foreach** *added subway sink p in Line 6, 8 or 11 with* $f_{opp}(\pi(p)) \neq \square$ **do**
13 **if** *there is no subway port in R_{reuse} at $f_{opp}(\pi(p))$* **then**
14 add port reservation p' at $f_{opp}(\pi(p))$ to Q
15 copy port reservation p' to equivalent continents (Algorithm 16)
16 run a replay routing pass w.r.t. R_{reuse} respecting Q, store resulting port graphs in R

We say that a routing respects a set of port reservations if it respects all port reservations in that set. Similarly, we say that a set of ports respects a set of port reservations if there is no subway source of a different net at the same position as a port reservation (or illegal due to spacing constraints).

These port reservations make sure that the access to a subway source is not blocked by a subway sink of a different net. This would lead to problems as it would prevent any routing going to the subway source. And since sinks can only be connected to one subway source, a configuration like this (also shown in Figure 4.32) would directly lead to a conflict.

Note that the replay mode of Section 4.5 can easily be adapted to compute routings respecting a given set of port reservations by modifying the track assignment: When adding a high penalty cost to forbidden tracks, wires are assigned to a different track.

Additionally, it is beneficial to count port reservations as routing usage during resource sharing. Otherwise, the congestion estimates are too optimistic and track assignment might not find a feasible solution.

Algorithm 15: Copying subway sinks to equivalent continents

Input: A subway sink p in continent C with associated net N and R_{reuse}.

```
1 t ← a terminal of N in C
2 foreach C' ≠ C equivalent to C do
3     π' ← translation of π(p) from C to C'
4     t' ← equivalent terminal to t in C'
5     add a subway sink p' at π' to port graph G of the net of t' in R_reuse
6     if G has subway/net source s in C' then
7         add edge (s, p') to G in R_reuse
```

Algorithm 16: Copying port reservations to equivalent continents

Input: A port reservation p in continent C with associated net N and Q.

```
1 t ← a terminal of N in C
2 foreach C' ≠ C equivalent to C do
3     π' ← translation of π(p) from C to C'
4     t' ← equivalent terminal to t in C'
5     add port reservation at π' associated with the net of t' to Q
```

A refined reuse flow

The algorithm considered in this section aims to solve instances with complex inter-continent dependencies given by continent equivalence and an alignment function f_{opp}. In the investigation of this algorithm, we assume for simplicity that no feed-throughs (see also Section 4.6.10) occur, i.e. that routes only enter continents in which they connect to terminals. This can be achieved by inserting buffers sub-dividing nets with feed-throughs in an initial (non-reuse) routing pass. Further, we assume that there are no terminals in between continents whose ports are opposite of each other according to f_{opp}. Such pins can be temporarily moved into one of the adjacent continents. Also, we use the notation $\pi(p)$ to denote the position of a subway port or port reservation p.

Consider Algorithm 14. We start with a first routing pass (Line 1), including the computation of port intervals and equivalent continent border blockages (Sections 4.6.3 and 4.6.4). Here, we make sure that all out-going port intervals are used, i.e. a subway sink is created inside every port interval during port cutting. This means that routes may contain antennas ending in a subway sink. The resulting port graphs of this routing pass are stored in the set R. We also initialize empty sets R_{reuse} and Q (Lines 2 and 3). These later store port graphs and port reservations.

Starting from Line 4, we iterate the following until the port graphs in R are reuse-aware. A continent equivalence class $\overline{C}$ for which R is not yet reuse-aware is chosen that was not considered before (Line 5). The ports in $\overline{C}$ are aligned according to the leader continent and the result is added to R_{reuse} (Line 6). This is done as described in Algorithm 13.

Afterwards, the following alterations are done to the port graphs in R_{reuse}: In Lines 7-11, we consider each subway source p with $f_{opp}(\pi(p)) \neq \square$ added in the previous execution of Line 6. The induced opposing subway sink p' is added at $f_{opp}(\pi(p))$ and connected via the edge (p', p). If the port graph of p contains a source (net source or subway source) s in the same continent as p', also the edge (s, p') is added. The newly added subway sinks and edges are translated to the equivalent continents and also added to R_{reuse}, see Algorithm 15 (next sub-section).

In Lines 12-15, we consider each subway sink p with $f_{opp}(\pi(p)) \neq \square$ that we added

in one of the previous steps of this iteration. If there is no subway port already at $f_{opp}(\pi(p))$, we add a port reservation p' at $f_{opp}(\pi(p))$ to Q and translate it to the equivalent continents (Algorithm 16, next sub-section). The translated port reservations are also added to Q. Note that we may add overlapping port reservations here.

Finally, a replay routing with respect to R_{reuse} and respecting Q is computed and the resulting port graphs are again stored in R (Line 16). If R is not reuse-aware, we repeat the process starting again from Line 5. The final routing pass, leading to reuse-aware port graphs, yields the desired result.

The difference in dealing with subway sinks and subway sources comes from the fact that actual induced ports are only added in "reverse" direction, i.e. only induced subway sinks are added to existing subway sources. Induced subway sources of existing subway sinks are added only as port reservations.

Copying ports and sinks to equivalent continents

In Algorithm 15, a given subway sink p in a continent C and belonging to net N is copied to equivalent continents. In Line 1, we fix an arbitrary terminal t of N inside C. Here, we use the assumption that no feed-throughs occur, i.e. that the net N does indeed have a terminal inside C. Then, we consider each equivalent continent C' to C. We translate $\pi(p)$ into the equivalent continents of C and call this π' (Line 3). We take the terminal equivalent to t in C' and call this t' (Line 4). Now we add a subway sink at π' to the port graph in R_{reuse} belonging to the net of t' (Line 5). Finally, we potentially add an incoming edge to the new subway sink (Lines 6 and 7).

Algorithm 16 does the same thing in copying port reservations to equivalent continents. Only here, no edge can be added.

The algorithm on an example instance

It is worthwhile to explore this algorithm on a more complex example instance. Have a look at Figure 4.33.

a) The given world contains three pairs of equivalent continents, arranged in three columns as depicted. The bottom row is mirrored on the x-axis compared to the top row. Each continent contains two sources and two sinks, each contained in a net together with a sink/source of the other continent in the same column. The nets are indicated by the dashed lines. Ports between the upper and lower continent of a column must align.

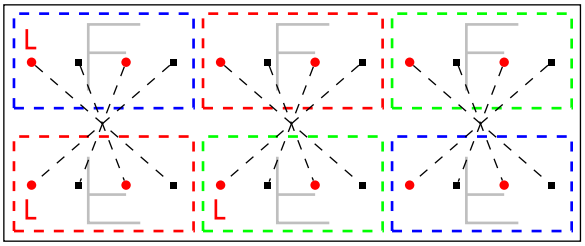
b) After the first routing pass, the routing is not reuse-aware. Every net contains exactly one subway sink and one subway source.

c) The blue equivalence class is chosen in the first iteration. The top left blue continent is the blue leader. The port graph parts of the blue leader are copied into the blue follower.

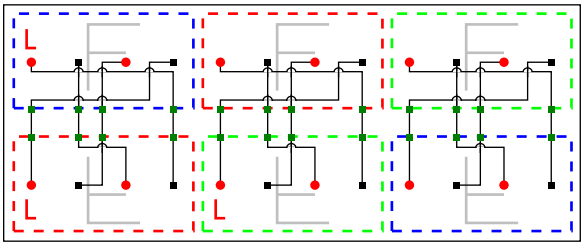
d) Subway sinks opposite of the blue subway sources are created in the red and green continents. They are connected to their respective subway sinks and net sources. The subway sinks and their adjacent edges in the red and green continents are then copied into the equivalent red and green continent.

e) Port reservations (orange) in the red and green continents are added to Q for every subway sink in a blue continent. This is done equivalently on both red and green continents. Dashed orange lines connect port reservations to the ports/terminals they belong to. The new red and green subway sinks in the center column do not lead to new port reservations, because there already are ports opposite of them.

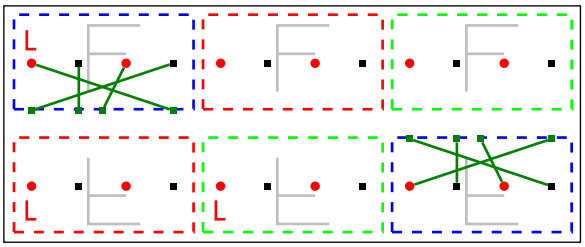
a) Given world and netlist:



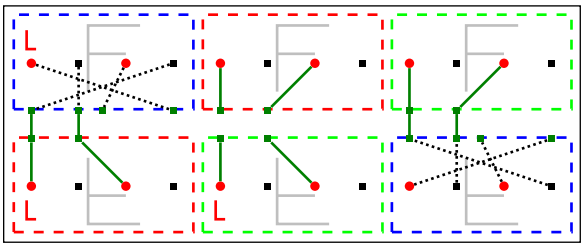
b) First routing pass:



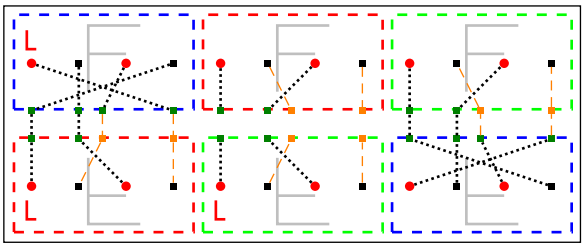
c) Equate blue by leader:



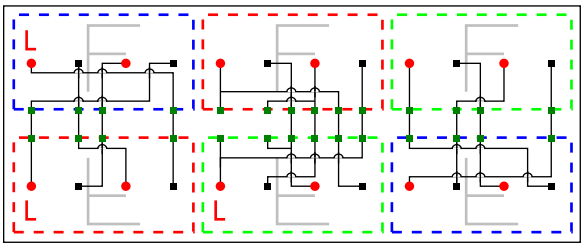
d) Add induced red&green subway sinks and edges:



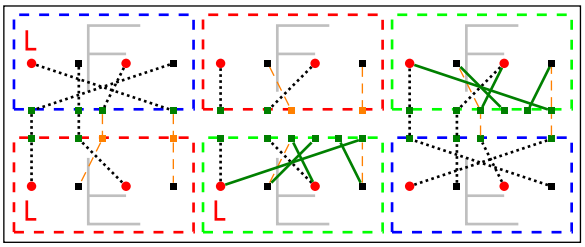
e) Add red&green port reservations:



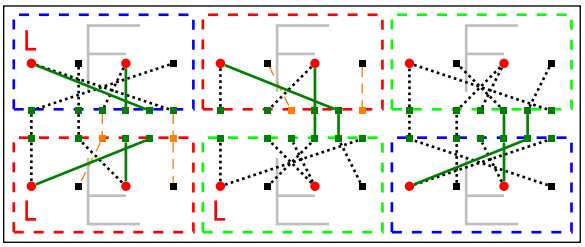
f) Second routing pass:



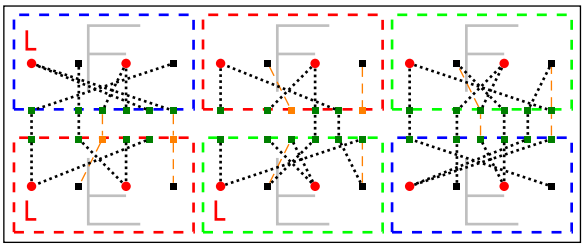
g) Equate green by leader:



h) Add induced blue&red subway sinks and edges:



i) No additional port reservations needed:



j) Third routing pass:

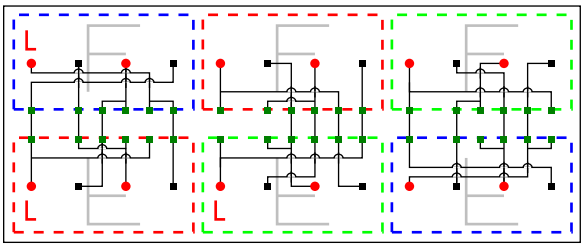


Figure 4.33: Algorithm 14 conducted on an example instance. Most pictures show the port graphs stored in R_{reuse} (dotted lines for existing edges, green lines for newly added edges) and the port reservations in Q (orange, dashed orange lines indicate the associated net). Picture a) shows the netlist, dashed lines represent nets. Pictures b), f) and j) show the results of the routing passes (solid lines).

f) The second routing is done replaying the port graphs and respecting the port reservations depicted in e). The nets in the center column now each contain two subway sinks.

g) In the second iteration, the green continents are chosen. Ports are aligned according to the green leader (lower left green continent).

h) Subway sinks opposite of green subway sources are created in the red and blue continents. They are connected to their respective net sources and green subway sources. The new subway sinks and edges are copied to the equivalent red and blue continents.

i) No port reservations are added, because all newly created subway sinks are already opposite of a subway port or port reservation.

j) After the third routing pass, the port graphs are reuse-aware. The red continent does not need to be aligned explicitly in this case.

Correctness of Algorithm 14

We now prove that on nice instances (see Definition 4.17), Algorithm 14 computes reuse-aware solutions obeying the restrictions given by an alignment function. We neglect the wire types of individual ports. This is because, by the nature of the algorithm, we can already identify subway sinks or subway sources when they have identical positions (including the layer) and belong to the same net.

We start with some observations.

Proposition 4.18. *R_{reuse} and Q are reuse-aware at the start of Lines 12 and 16 in every iteration.*

Proof. R_{reuse} only contains ports aligned in Line 6, which are reuse-aware, and subway sinks added in the first for-loop, which are made reuse-aware by Line 11. Q only contains port reservations which are made reuse-aware by Line 15. □

Proposition 4.19. *At the end of every while-loop-iteration, the following holds.*

1. *For every subway source p in R_{reuse} with $f_{opp}(\pi(p)) \neq \square$, there is a subway sink p' in R_{reuse} at $f_{opp}(\pi(p))$ and an edge (p', p).*
2. *For every subway sink p in R_{reuse} with $f_{opp}(\pi(p)) \neq \square$, there is either*
   a) *a subway source p' in R_{reuse} at $f_{opp}(\pi(p))$ and an edge (p, p'), or*
   b) *a subway sink in R_{reuse} at $f_{opp}(\pi(p))$, or*
   c) *a port reservation in Q at $f_{opp}(\pi(p))$.*

Proof.
1. Let p be a subway source in R_{reuse} with $f_{opp}(\pi(p)) \neq \square$. p was added to R_{reuse} in Line 6 of some iteration i. In that iteration, p is considered in the first for-loop, and a subway sink p' at $f_{opp}(\pi(p))$ with an edge (p', p) is added to R_{reuse} in Line 8. We never remove ports or edges from R_{reuse}.
2. Let p be a subway sink in R_{reuse} with $f_{opp}(\pi(p)) \neq \square$. p is considered in the second for-loop of the while-loop-iteration in which it was added to R_{reuse}. If there is no subway port or port reservation at $f_{opp}(\pi(p))$ at that point, a port reservation is created and never removed. If there is a subway source at $f_{opp}(\pi(p))$, p was created in Line 8 and is hence connected to the subway source at $f_{opp}(\pi(p))$.

□

We can now show that Line 6 does not create conflicts.

Lemma 4.20. *Consider an iteration of the while-loop. Let p be in the result of aligning ports according to the leader (Line 6). Then p was either already contained in R_{reuse} before Line 6, or there was no port in R_{reuse} at the same position.*

Proof. Let P be the result of aligning ports according to the leader.

A subway port $p \in P$ in the leader continent comes from the previous routing pass. But the previous routing pass replays R_{reuse}, so either there is no subway port at $\pi(p)$ in R_{reuse}, or R_{reuse} already contains p as claimed.

A subway port $p \in P$ in a follower continent comes from an equivalent port p_L in the leader. Assume that there is a subway port p' in R_{reuse} at $\pi(p)$. Using that R_{reuse} is reuse-aware, let p'_L be the equivalent port to p' in the leader. By the above, we know that $p'_L = p_L$. Now by equivalence, we have $p' = p$. □

Lemma 4.21. *Whenever a subway port is added to R_{reuse}, it respects Q.*

Proof. Subway sinks always respect port reservations and subway sources are only added to R_{reuse} in Line 6. So consider a subway source p added to R_{reuse} in Line 6. If p is in a leader continent, it respects Q because the routing respects Q. If p is in a follower continent, it respects Q because Q is equivalent within equivalent continents (Proposition 4.18): If p did not respect a port reservation q, then the equivalent port to p in the leader did not respect the equivalent port to q in the leader. □

Next, we show that Line 8 does not create a conflict.

Lemma 4.22. *Assuming the instance is nice (Definition 4.17), we have the following. Consider an iteration of the first for-loop where we consider a subway source p. Before the execution of Line 8, R_{reuse} either already contains a subway sink at $f_{opp}(\pi(p))$ belonging to the same net as p, or it contains no subway port at $f_{opp}(\pi(p))$.*

Proof. Let p be a subway source as in Line 7. Assume there already is a port p' at $f_{opp}(\pi(p)) \neq \square$ in R_{reuse} before the execution of Line 8. We want to show that p' is a subway sink and that p and p' belong to the same net.

Call the current while-loop iteration i. p' was not created in Line 6 of iteration i, because p' does not belong to a continent in $\overline{C}$ by Definition 4.17 b). So either p' was in R_{reuse} already at the start of i, or it was added to R_{reuse} in a previous iteration of the first for-loop.

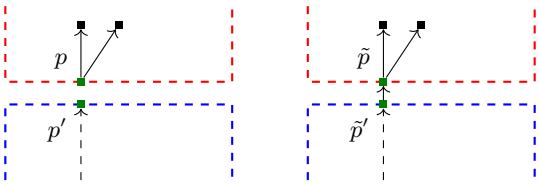
Case 1: p' was in R_{reuse} already at the start of i:
Assume p' is a subway source. Then by Proposition 4.19, there was already a subway sink at $\pi(p)$ at the start of i. Contradiction to Lemma 4.20, because p is a subway source. So p' is a subway sink.

Let j be the while-loop iteration when p' was added to R_{reuse}. In j, a port reservation at $\pi(p) = f_{opp}(\pi(p'))$ associated with the net of p' was added to Q, or there already was a subway port at $\pi(p)$. But there was no subway port at $\pi(p)$, because there was still no subway port at $\pi(p)$ at the beginning of iteration i. This is because p was newly added in iteration i. So in iteration i there is a port reservation q associated with the net of p' at $\pi(p)$.

By Lemma 4.21, p respects q. Because p is a subway source, it must be of the same net as q, and thus as p'.

Case 2: p' was added to R_{reuse} in a previous iteration of the for-loop:
Call the for-loop-iteration in which p' was added j_1. Call the current for-loop-iteration j_2. p' is a subway sink, because no subway sources are added to R_{reuse} in the for-loop. If p' was added in Line 8 of j_1, then j_1 also considered a subway source at the position $f_{opp}(\pi(p')) = \pi(p)$, contradiction. So p' was added in Line 11 of j_1.

Let $\tilde{p}'$ be the subway sink added to R_{reuse} in Line 8 of iteration j_1. Then p' is equivalent to $\tilde{p}'$. Let $\tilde{p}$ be the subway source with $f_{opp}(\pi(\tilde{p})) \neq \square$ considered in j_1. Then $f_{opp}(\pi(\tilde{p})) = \pi(\tilde{p}')$. This configuration is shown here:



As Line 6 only adds ports in one continent equivalence class to R_{reuse}, $\tilde{p}$ and p are ports in the same continent equivalence class. Now we can use the fact that we have a nice instance to see that $\tilde{p}$ and p are equivalent (requirement a) in Definition 4.17).

We can now apply requirement c) in Definition 4.17) to a terminal $\tilde{t}$ connected to $\tilde{p}$ and a terminal $\tilde{t}'$ connected to $\tilde{p}'$. We get that p and p' are in the same net as claimed. Here, we used the fact that the instances we consider do not have feed-throughs, i.e. that terminals exist in every continent relevant for a net. □

Similarly, Line 11 does not create a conflict.

Lemma 4.23. *Consider an iteration of the first for-loop where we add a subway sink p' in Line 8. Assume there already exists a subway port $\tilde{p}'$ with $\pi(\tilde{p}')$ equivalent to $\pi(p')$ before the execution of Line 11. Then $\tilde{p}'$ is a subway sink and belongs to the net that Algorithm 15 creates the subway sink at $\pi(\tilde{p}')$ for.*

Proof. Consider an iteration i of the first for-loop. Let p' be the subway sink from Line 8 of iteration i in some continent C.

Assume p' did not exist before Line 8. Then it did not exist at the end of the last while-loop-iteration (no ports are removed). By Proposition 4.18, there is also no port at an equivalent position to $\pi(p')$ in R_{reuse}. Contradiction to the assumption of the lemma.

So p' already existed before the execution of Line 8. p' was either added to R_{reuse} in a previous while-loop-iteration, or it was added in the same while-loop-iteration but in a previous iteration of the for-loop.

Case 1: p' was added to R_{reuse} in a previous while-loop-iteration:
At the end of that while-loop-iteration, R_{reuse} was reuse-aware by Proposition 4.18. This means that R_{reuse} contains equivalent subway sinks to p'. By their equivalence, they belong to the same nets as considered in Algorithm 15.

Case 2: p' was added to R_{reuse} in a previous iteration of the for-loop:
Call the for-loop-iteration in which p' was added iteration j. As in Case 2 in the proof of Lemma 4.22, p' must be a subway sink and was added in Line 11. Iteration j also added equivalent subway sinks using Algorithm 15. By the definition of Algorithm 15, they respectively belong to the correct nets. □

Finally, Lines 14 and 15 do not create a conflict.

Lemma 4.24. *Whenever a port reservation is added to Q (in Lines 14 and 15), all ports in R_{reuse} at that time already respect it.*

Proof. In Line 14, we only add a port reservation p' at $f_{opp}(\pi(p))$ if there is no subway port or port reservation at that position before. Hence p' is respected automatically.

By Proposition 4.18, there is also no subway port at a position that is equivalent to $f_{opp}(\pi(p))$. So the port reservations added by Line 15 are also respected. □

The previous statements together show that, in each routing pass of Line 16, it is possible to fully replay R_{reuse} and respect the port reservations in Q. We can finally state:

Theorem 4.25. *Given a nice instance (Definition 4.17) and assuming each routing pass finds a solution using at most one port per port interval, Algorithm 14 terminates using at most $k+1$ routing passes. Here, k denotes the number of non-trivial continent equivalence classes. Further, at the end R is reuse-aware for all continent equivalence classes and obeys the restrictions imposed by f_{opp}.*

Proof. Under the above assumptions, we show that Algorithm 14 terminates after at most k iterations of the while-loop. Together with Line 1, this gives at most $k+1$ routing passes.

So let us show that R is reuse-aware after the while-loop was executed for every non-trivial continent equivalence class. Assume that, at the end of the k-th while-loop-iteration, R contains a subway port p in some continent C that does not exist equivalently in an equivalent continent C'.

Because R_{reuse} is reuse-aware (Proposition 4.18), we know that p is not contained in R_{reuse}. By assumption, we know that p is the only subway port inside its port interval. But when Algorithm 13 was executed for the equivalence class of C and C', equivalent subway ports inside this port interval and its equivalent intervals were added to R_{reuse}. Contradiction. □

Considering independent continent equivalent classes simultaneously

Algorithm 14 considers each continent equivalence class separately. This makes the analysis simpler, but takes more iterations than strictly necessary. Continent equivalence classes that are independent enough can be dealt with in the same iteration of the while-loop. So let us investigate what *independent* means here.

For a continent equivalence class $\overline{\mathcal{C}}$, let $D(\overline{\mathcal{C}})$ be a set containing the following continents:

a) continents in $\overline{\mathcal{C}}$,

b) continents neighboring a continent in $\overline{\mathcal{C}}$ and

c) continents equivalent to a continent satisfying b).

In a while-loop iteration in Algorithm 14 where a continent equivalence class $\overline{\mathcal{C}}$ is considered, all ports and port reservations added to R_{reuse} or Q are contained in a continent in $D(\overline{\mathcal{C}})$. No ports or port reservations inside other continents are created. This leads to the following definition.

Definition 4.26. Continent equivalence classes $\overline{\mathcal{C}_1} \neq \overline{\mathcal{C}_2}$ are *independent* if $D(\overline{\mathcal{C}_1})$ and $D(\overline{\mathcal{C}_2})$ are disjoint.

The following observation justifies this definition.

Proposition 4.27. *Consider independent continent equivalence classes $\overline{\mathcal{C}_1} \neq \overline{\mathcal{C}_2}$. Assume that in some while-loop iteration i of Algorithm 14, $\overline{\mathcal{C}_1}$ was picked in Line 5. Let $\mathcal{W}$ denote the routing solution computed directly before that (either in Line 1 or Line 16 of the previous while-loop iteration).*

Then the wire segments of $\mathcal{W}$ that are inside $\overline{\mathcal{C}_2}$ do not conflict with R_{reuse} or Q at the end of iteration i.

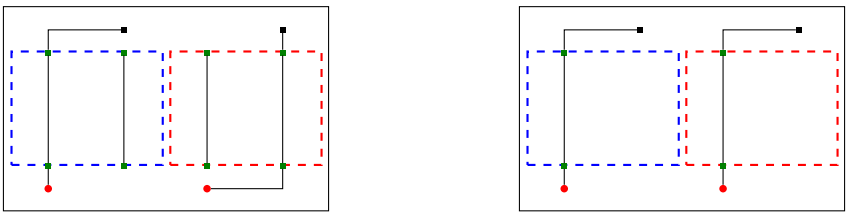


Figure 4.34: Example routing solution with copied feed-throughs (left) versus one reused feed-through (right). The blue and red continents are equivalent.

Proof. As $\overline{C_1}$ and $\overline{C_2}$ are independent and distinct, no ports or port reservations in $\overline{C_2}$ were created in iteration i. All ports or port reservations already present before iteration i are respected by the routing. □

In Algorithm 14, the operations inside one iteration of the while-loop do not depend on anything outside of the considered continent equivalence class. Together with the previous proposition, we can conclude: When considering several pairwise independent continent equivalence classes in each while-loop iteration, Theorem 4.25 still holds.

4.6.10 Reusing feed-throughs

In general, some nets need to be routed through a reuse continent C without connecting to a terminal inside C. In these cases, the connected component of the port graph (or wiring) inside C is called a *feed-through*. In this section, we consider feed-throughs with one sink, i.e. a segment going from a subway source to a single subway sink inside the same continent. However, everything can be extended to deal with multi-fanout feed-throughs as well.

When multiple equivalent continents contain feed-throughs after the first routing, Algorithm 13 copies each of them into each equivalent continent. This results in much dead wiring: Every feed-through is used only in one continent. Its copies are left unconnected.

However, when feed-throughs of equivalent continents are similar, they can be used to implement different connections. A simple example of the result of copying feed-throughs versus reusing them is depicted in Figure 4.34. This section explores how copying feed-throughs can be avoided while not impacting path lengths too much.

This topic is discussed in a more abstract way in [Mun23]. The author shows that several formulations of the problem are NP-hard to solve optimally via a reduction from 3D-matching. Additionally, Mundt gives a polynomial time algorithm for solving a specific simplified version with two equivalent continents as well as an approximation algorithm for more continents using submodular function maximization. In contrast, here we consider how good solutions can be found in practice as part of the existing Pangea ReUse flow.

Two equivalent continents

If there are only two equivalent continents, we can provide a flexible framework that models reusing feed-throughs and allows for optimal solutions in polynomial time. This approach is similar to the algorithm in [Mun23]. Both are based on bipartite matching. However our new algorithm can solve more general instances compared to the one in [Mun23].

We consider the following setting. Of all given continents $\mathcal{C}$, only the continents $A, B \in \mathcal{C}$ are equivalent. Further, we ran a first routing pass such that port graphs are

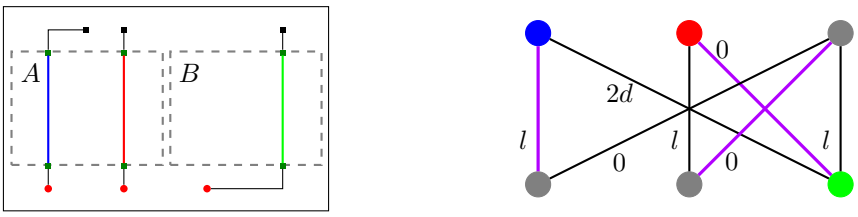


Figure 4.35: Example routing (left) and corresponding bipartite feed-through graph with minimum-cost perfect matching (right). The continents A and B are equivalent. The colored vertices correspond to the feed-throughs of the same color. The gray vertices are dummy vertices. The edge weight is denoted next to the edges. Here we say that the length of the feed-throughs is l. The distance of the blue and red feed-throughs is d. The matching edges are shown in purple.

given for all nets. We assume a full routing to be given as it is not clear a priori which nets need feed-throughs through which continents.

Remark. Multiple pairs of equivalent continents can be considered easily as long as port positions of one pair do not influence the feasibility of port positions on another pair, see Definition 4.26.

We now construct a weighted bipartite graph $(G = A' \cup B' \cup D, E, c)$ as follows. The structure of G for an example setting is shown in Figure 4.35. For every single-sink feed-through in A, there is a vertex in A' and similar for B and B'. Additionally, there are $|A'| + |B'|$ (dummy) vertices in D. These represent copying a feed-through instead of reusing it.

The edge set E is constructed as follows. For any pair of feed-throughs a in A and b in B, add an edge between the corresponding vertices if and only if it is feasible to reuse one of them to replace the other. Say it is cheaper to copy a into B and remove b than the other way around. Then the weight $c(e)$ of this edge is given by

$$c(e) \coloneqq \textit{added-detour-cost} + \textit{routing-cost}(a) - \textit{routing-cost}(b),$$

where *added-detour-cost* represents the cost of the added detour when using the copy of a in B instead of the original segment b for the net through B.

A simple estimation for the detour cost is to take the summed up distance between the start and endpoints of a and b (after transforming them to one continent). This would represent routing from the previous port position to the new one, traverse the continent via the copied feed-through and then route back. The situation described here is depicted in Figure 4.36 with large detour for clarity.

In addition to these edges, connect each vertex in $A' \cup B'$ to a distinct vertex in D. Such an edge e connected to a feed-through f has the cost

$$c(e) \coloneqq \textit{copying-cost}(f).$$

This can simply be the routing cost associated with the feed-through.

Finally, connect every dummy vertex connected to some vertex in A' to all dummy vertices connected to vertices in B' and vice versa. These edges have cost 0.

Note that this graph is in fact bipartite and allows for a perfect matching by using the edges between dummy vertices and vertices in $A' \cup B'$. Furthermore, each perfect matching corresponds to a choice of reusing some feed-throughs and copying the rest, and of the same cost:

An edge between vertices corresponding to feed-throughs a in A and b in B represents using one of these feed-throughs for both connections and removing the other one.

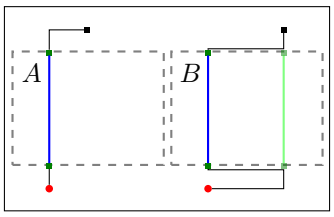


Figure 4.36: Routes representing the upper-bound cost for reusing a feed-through. When using the copied blue feed-through instead of the green one, the additional cost of the summed distances of the start and end points is justified by the possibility to route to and from the copied blue feed-through as shown.

An edge between a feed-through vertex and a dummy vertex represents that the feed-through in question is copied into the equivalent continent. Lastly, the edges between dummy vertices make sure we indeed have a perfect matching.

In the constructed graph, we can compute a minimum-cost perfect matching in polynomial time. The matching can then be translated back into partial port graphs.

Remark. To avoid overlapping ports, modify all ports in the solution to be port intervals instead. If the intervals are large enough, the router can find feasible positions for previously overlapping ports.

More than two equivalent continents

When there are more than two continents in one equivalence class, an analogous construction can be done. However, already for three equivalent continents, the problem to be solved is 3-dimensional matching. In general, for k equivalent continents, we need to compute vertex clusters of size k. Assuming P≠NP, this cannot be solved in polynomial time. Instead, we propose a greedy algorithm described in this section to be used in practice for these cases.

We use the following metrics to measure the quality of a given clustering of feed-throughs. First, define a *cluster* to be a set of feed-throughs in one continent equivalence class that contains at most one feed-through in each continent. Note that it is allowed to have a cluster with fewer feed-throughs then the number of continents in the given equivalence class. For a cluster A, the *cluster lead* denoted by $lead(A)$ is the feed-through that is picked to be used for all connections. This can be chosen to be a feed-through close to the center of all feed-throughs in A, or simply the feed-through through the leader continent. Further, the *cluster value* denoted by $v(A)$ is the saved routing cost when using the copied cluster lead for all connections in the given cluster, compared to copying all feed-throughs. The cluster value is composed of k times the routing cost for every non-lead feed-through in the cluster, where k is the number of equivalent continents, minus the distances of the start and end points of $lead(A)$ to the other feed-throughs:

$$\begin{aligned} v(A) \coloneqq & \sum_{a \in A \setminus \{lead(A)\}} k \cdot cost(a) \\ & - \operatorname{dist}(lead(A)^{(start)}, a^{(start)}) \\ & - \operatorname{dist}(lead(A)^{(end)}, a^{(end)}). \end{aligned}$$

Remark. Whenever we speak of a position of a feed-through (or its start/end point), we mean the relative position within its continent. In particular, the distance between feed-throughs through different equivalent continents is to be taken after applying the continent transformations accordingly.

As ground set, the algorithm uses all single-sink feed-throughs through one of the continents in the equivalence class. Clusters are built up greedily. While there are still un-clustered feed-throughs, pick one with largest cost and call it a. For every equivalent continent, pick a feed-through whose start and end points are closest to a. For a continent for which there are no feed-throughs with close enough start and end points, pick no feed-through. Put the found feed-throughs into a cluster and choose a cluster lead maximizing the cluster value.

In practice, the criterion of close enough start and end points can be extended to having the same wire type. Further post-optimizations can be implemented, such as local optimizations or a chain-opt algorithm.

The final solution is then translated back to partial port graphs. Ports might need to be extended to port intervals to guarantee feasibility as above.

4.6.11 Pangea reuse in practice

The Pangea ReUse flow was first developed specifically to be used for a large design containing one continent four times. By aligning the equivalent continents, the total number could be reduced from 8 down to 5 continents. Compared to manually equating the four equivalent continents, the Pangea ReUse flow reduces the time needed for one design cycle from about 2 weeks down to 3-4 days. This is a huge improvement.

Going forward, the reuse mode enables BonnPangea to be used for more designs by reducing the number of continents.

Chapter 5

BonnRouteBuffer

Alongside the ongoing development of BonnPangea, the author of this thesis has been working on improving the practical application of BonnRouteBuffer. Because long wire segments lead to large signal delays and can cause slew or load violations, it is necessary to break up such wiring into smaller segments. This is done by *buffering*, the process of inserting repeaters (buffers and inverters) on a chip to subdivide nets. As the name suggests, BonnRouteBuffer does not only do that, but also routes the nets. This makes sure good repeater positions can be chosen and allows for a much improved timing analysis compared to placing repeaters only based on pin positions.

In the sections of this chapter, first the workings of the previously existing version of BonnRouteBuffer are outlined. There we see which alterations to the resource sharing framework doing global routing are necessary (Section 5.1) and give a brief overview how the net customer oracle performs the actual buffering (Section 5.2). Afterwards, we highlight some problems in the previous implementation of BonnRouteBuffer and the improvements done during the work on this thesis (Section 5.3). Finally, we experimentally compare the new version of BonnRouteBuffer to the old version as well as to the industrial competition in use by IBM today (Section 5.4).

We close with a discussion of possible future work items proposed by the author (Section 5.5).

5.1 BonnRouteBuffer in resource sharing

BonnRouteBuffer is implemented using the resource sharing framework and can be thought of as an extension to global routing as presented in Section 2.4. Within resource sharing, additional resources are incorporated to account not only for net length and wiring congestion, but also timing, placement density and power consumption. This is also outlined in Section 2.4.

Compared to global routing, the two main differences are collecting the nets and the oracle used for the net customers. In normal global routing, a net starts at the output of some gate and has the successor gates' inputs as sinks. During buffering, all repeaters not fixed in place by the user are skipped in this process, as the buffering inserts new repeaters from scratch. This means that when collecting the instances, repeaters that BonnRouteBuffer is allowed to remove are treated the same as a piece of wire.

During this process, sinks with an odd number of inverters on the path from their source are marked as `invert`. A buffering solution must also have an odd number of inverters on the path from the source to these sinks.

The oracle for net customers is similar to the one presented in Section 3.1. First, a 2-dimensional topology is computed on the source and sink positions. Here, length

and delay are taken into account. The default algorithm used for this is the bi-criteria algorithm (without bifurcation delay) from Section 3.2.2. Afterwards, the topology is embedded into the global routing graph. The embedding takes into account estimations for power and placement costs of the subsequent buffering. It uses a linear timing model.

After the route is computed, repeaters are inserted along the wiring. This is done via a dynamic program going bottom-up from the sinks to the source. This algorithm is presented briefly in the following section.

5.2 Dynamic program for buffering

The core of BonnRouteBuffer is a dynamic program traversing a route in reverse topological order. This section provides enough information to understand the subsequent modifications and improvements to BonnRouteBuffer. More details can be found in [Rot17], [Dab21] and [Dab+23]. In particular, we will not consider the option of changing the topology during buffering. This was first described in [Bar+09] and is also part of the BonnRouteBuffer buffering oracle.

The dynamic program creates labels at each node of the global route. A label represents a buffering of the sub-tree rooted at that label's node. Put formally, a label l consists of the following:

- a node $v(l)$ of the global route,
- a position $pos(l)$ on the incoming edge of $v(l)$,
- a polarity $pol(l) \in \{\texttt{ident}, \texttt{invert}\}$,
- a value $rep(l) \in \textit{repeater-set} \cup \{\square\}$ representing the repeater to be inserted at that position ($\square$ indicating no repeater),
- a capacitance value $cap(l)$,
- a slew limit $\textit{slew-limit}(l)$,
- a predecessor label $pred(l)$ (can be None),
- and costs $cost(l)$

A simplified account of the label propagation is as follows. For each sink t, a label l is created with $v(l)$, $pos(l)$, $pol(l)$ and $\textit{slew-limit}(l)$ being the respective properties of t, and further $rep(l) = \square$, $cap(l) = \textit{input-cap}(t)$, $pred(l) = \text{None}$ and $cost(l) = 0$.

Then, the global route is traversed bottom-up. At each node v, we do the following. If v has two outgoing edges, we first merge labels coming from the two child-trees. For two labels of the same polarity, this can essentially be done by adding the cost and capacitance values and taking the minimum of the slew limits. For labels of different polarities, an additional inverter is inserted into one of the labels beforehand.

Afterwards, or if v has fewer outgoing edges, all (resulting) labels at v are considered. For a label l , we first try not inserting a repeater and create a label representing this at the parent of v. The new label has an increased capacitance value and a decreased slew limit according to the wire properties.

Now for all repeater types, we find the (approximately) earliest position on the edge $(parent(v), v)$ on which the repeater can be placed without introducing violations. For this position, a new label is created with this repeater type. The new label will be associated with $parent(v)$ if this position was violation-free. Otherwise, the new label will still be associated with v and later considered in this step as well.

After arriving at the root, it is allowed to add a last repeater to every label. Finally, the best label is chosen. The result can be retrieved by backtracking.

5.3 Enhancements and fixes

5.3.1 Slew computation

Slew pessimism

The main factor preventing the existing implementation of BonnRouteBuffer from computing faster repeater trees was a significant pessimism regarding slew limits. When creating the initial label for the sink of an instance, the slew limit at that sink is multiplied with a constant less than 1. This way, inaccuracies regarding slew computation will not immediately lead to slew violations.

In practice, a rather extreme value of 0.7 was used. This meant that the dynamic program was forced to add more repeaters than necessary to avoid paying a penalty for slew violations. The additional repeaters led to increased power consumption, higher area usage and worse timing properties.

Running the same old BonnRouteBuffer version but changing this pessimism value to 0.9 already improves upon these metrics drastically, see Section 5.4.3. But naturally, reducing the pessimism leads to more slew violations. A comparison of using these different pessimism values in the old BonnRouteBuffer implementation can be found in Section 5.4.3.

Backwards propagation of slew limits

In the label definition of Section 5.2, every label l is associated with a single slew limit value $\mathit{slew\text{-}limit}(l)$. Whenever this label is propagated through a wiring edge (i.e. without repeater), an inverse slew function is applied to the slew limit. We compute the slew limit of the new label l' by

$$\mathit{slew\text{-}limit}(l') \coloneqq \mathit{slew}^{-1}\left(rc_{wire}, \mathit{slew\text{-}limit}(l)\right),$$

where rc_{wire} denotes the rc-value of the wire segment.

When a label l gets propagated through two consecutive wire segments with rc-values rc_1 and rc_2, the resulting label l' then gets assigned the slew limit value

$$\mathit{slew\text{-}limit}(l') = \mathit{slew}^{-1}\left(rc_2, \mathit{slew}^{-1}\left(rc_1, \mathit{slew\text{-}limit}(l)\right)\right).$$

However, this is not the same as

$$\mathit{slew}^{-1}\left(rc_1 + rc_2, \mathit{slew\text{-}limit}(l)\right),$$

which would be used if l got propagated through both wire segments at once, and which is more exact.

When using the delay model defined in Section 2.5.1 ([Bak90] and [Kas+04]), we have

$$\mathit{slew}^{-1}(rc, \mathit{outslew}) = \sqrt{\mathit{outslew}^2 - \ln^2 9 \cdot rc^2},$$

so the above terms become

$$\mathit{slew}^{-1}\left(rc_2, \mathit{slew}^{-1}\left(rc_1, \mathit{slew\text{-}limit}(l)\right)\right) = \sqrt{\mathit{slew\text{-}limit}(l)^2 - \ln^2 9 \cdot (rc_1^2 + rc_2^2)}$$

and

$$\mathit{slew}^{-1}\left(rc_1 + rc_2, \mathit{slew\text{-}limit}(l)\right) = \sqrt{\mathit{slew\text{-}limit}(l)^2 - \ln^2 9 \cdot (rc_1 + rc_2)^2}.$$

In particular, the total considered slew degradation when propagating in two steps compared to propagating in one step is smaller by

$$\ln 9 \sqrt{(rc_1 + rc_2)^2 - (rc_1^2 + rc_2^2)} = \ln 9 \sqrt{2 \cdot rc_1 \cdot rc_2}.$$

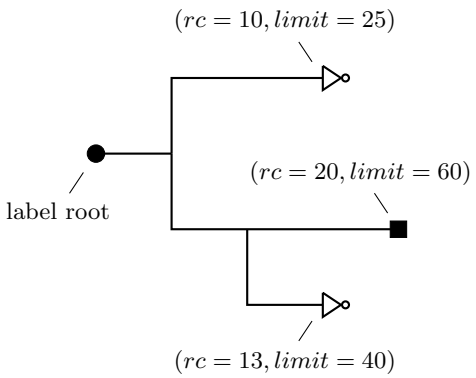


Figure 5.1: An example partial repeater tree associated with a label with $\textit{slew-limit-data} = \{(10, 25), (20, 60), (13, 40)\}$.

This means that propagating small distances at a time is too optimistic regarding slew limit degradation.

We want to keep the flexibility of short propagation distances, but improve upon the accuracy of the propagation. The single slew limit value of a label l is replaced by a set of rc-value, slew limit pairs denoted by $\textit{slew-limit-data}(l)$. This slew limit data can be interpreted as follows. We consider the wiring tree rooted at $v(l)$ with leaves being the repeaters and net sinks reachable from $v(l)$ according to l. An example for such a tree is indicated in Figure 5.1. Now for every leaf in this tree, $\textit{slew-limit-data}(l)$ contains a pair $(rc, \textit{limit})$, where rc is the rc-value of the path from $v(l)$ to the leaf, and $\textit{limit}$ is the slew limit at that leaf.

When merging two labels at a node, the union of the slew limit data of both labels is taken. When a label gets propagated through a wire segment with rc-value rc, the new slew limit data arises by adding rc to the rc-value of every pair. When a repeater is added to a label, the slew limit at its output is computed by taking

$$\min\{\textit{slew}^{-1}(rc, \textit{limit}) \mid (rc, \textit{limit}) \in \textit{slew-limit-data}(l)\}.$$

This value is then propagated through the repeater as usual. This results in a singleton slew limit data of the form $\{(0, \textit{limit})\}$.

A lower rc-value and higher slew limit at a leaf can never lead to a smaller slew limit at the root. We define a dominance relation $>_D$ on these pairs by

$$(rc_1, \textit{limit}_1) >_D (rc_2, \textit{limit}_2) \iff rc_1 \geq rc_2 \text{ and } \textit{limit}_1 \leq \textit{limit}_2.$$

We drop dominated pairs during a merge.

This modification slightly increases the memory consumption as more data is stored for each label. However, it greatly improves the accuracy of the backwards slew propagation. This leads to fewer and less severe slew violations in the result.

Forwards propagation of slews

When timing through the final repeater tree, an analogous improvement is possible. Instead of propagating slew (and delay) values through the wiring segment by segment, entire paths from repeater/root to the next repeater/sink are now considered at once. This is more accurate for the same reasons as above. Depending on the delay model, we may have

$$\textit{slew}(rc_2, \textit{slew}(rc_1, \textit{slew}_0)) \neq \textit{slew}(rc_1 + rc_2, \textit{slew}_0).$$

In particular, for the slew computation defined in Section 2.5.1, we have

$$\begin{aligned} slew(rc_2, slew(rc_1, slew_0)) &= \sqrt{slew_0^2 + \ln^2 9 \cdot (rc_1^2 + rc_2^2)} \\ &\neq \sqrt{slew_0^2 + \ln^2 9 \cdot (rc_1 + rc_2)^2} \\ &= slew(rc_1 + rc_2, slew_0). \end{aligned}$$

Slew violations in the final dynamic program step

In the previous version of BonnRouteBuffer, slew violations occurring directly at the root node were neglected by the dynamic program. This led to further inaccuracies and especially to the selection of sub-optimal labels. When propagating through the root gate, we now consider slew violations in the same way as inside the repeater tree itself.

5.3.2 Speed ups

In addition to improving the quality of results, the new version of BonnRouteBuffer also contains several speed-ups compared to the previous version. While these changes can impact the result a bit, they save enough running time to outweigh the drawbacks.

Avoid neighboring repeaters

In the dynamic program as outlined above, labels do not remember where the last repeater was inserted. For that reason, many labels are created that place repeaters in adjacent global routing tiles. Because this distance is so small, these labels will not contribute to a good solution unless such repeaters are needed to avoid slew or load violations.

Now, when a label l was created by inserting a repeater, we will first only propagate it further through the next wire segment without another repeater. If this results in a label without slew or load violations, then we will not propagate l with repeater at all. Otherwise, we will proceed to insert different repeater kinds as before.

With this, we save the computation time of creating labels with neighboring repeaters when they are not needed.

Limit recomputations on nets with many sinks

The running time of the dynamic program increases drastically on nets with many sinks. There are typically only a few such nets. So they do not affect the overall result quality significantly. Limiting the effort on such nets can have a huge impact on the running time. But it will not worsen total results notably.

The new version of BonnRouteBuffer limits the oracle calls of nets with at least 200 sinks. They will be recomputed at most in every fourth resource sharing phase. In other phases, the previously computed solution is used again.

Fewer resource sharing phases

Before, the resource sharing algorithm in the BonnTools was run for 25 phases. Further testing showed that for BonnRouteBuffer, we can achieve comparable results when running only 15 phases. As the resource sharing algorithm contributes a large amount of the running time, this leads to a significant speed-up. Refer to Section 5.4.4 for further details.

5.3.3 Other improvements

Consider via-stack capacitance when computing repeater distance

Consider the buffering of a single wire segment without bifurcations. When a label is extended by inserting a repeater, the placement is usually chosen as a tile-center of a global routing tile.

However, this can be too coarse with regards to slew or load violations. So when the next location would create an unattainable slew limit or a too high capacitance, we instead compute a closer location where a given repeater can be inserted without violations. This way, an almost continuous choice of position is possible.

In this computation, the capacitance downstream of a new repeater impacts potential slew or load violations. Previously, only the wire capacitance was considered here. However, whenever a repeater is inserted, via-stacks are needed. They connect the routing layer to the input and output pins. The capacitance of the via-stack at the repeater output is now correctly taken into account together with the wire capacitance.

Remove solutions with load violations before randomized rounding

The resource sharing algorithm results in a fractional solution for every net. To get an integral solution, BonnRouteBuffer performs randomized rounding.

Before that, we remove outliers from the set of solutions for every net. To prevent unnecessary load violations, solutions containing such are now also removed before randomized rounding for nets with at least 3 violation-free solutions.

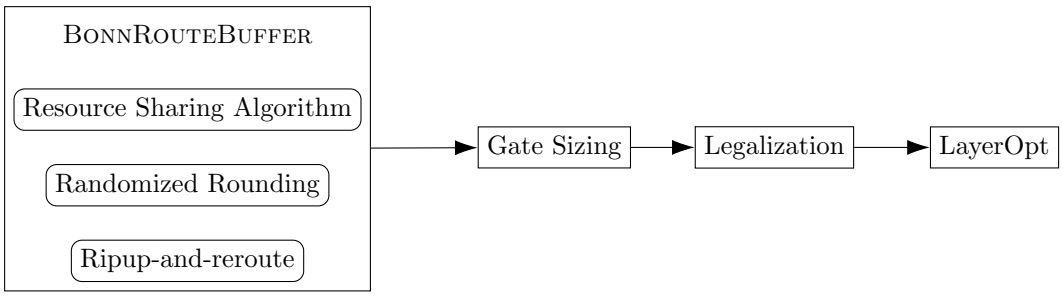


Figure 5.2: Flow surrounding BonnRouteBuffer. These steps were run in the results of Tables 5.2, 5.4, 5.5, 5.6 and 5.7.

5.4 Experimental results

This section presents experimental results of BonnRouteBuffer. We start by introducing the wire synthesis flow in Section 5.4.1 and the considered metrics and test instances in Section 5.4.2. In Section 5.4.3, we showcase the importance of slew computations and slew limits. We see that a seemingly small change regarding slew limits can impact many metrics. This indicates a path of potential improvement and further development. Section 5.4.4 then compares the previous and new BonnRouteBuffer versions. Finally, Section 5.4.5 demonstrates how the new BonnRouteBuffer version compares against the current wire synthesis flow used in industrial practice at IBM.

The tests in Sections 5.4.3 and 5.4.4 were performed on a RedHat Enterprise 8.10 machine with an AMD EPYC 9684X processor. Each run used 32 threads. All results are archived at the Institute for Discrete Mathematics of the University of Bonn.

5.4.1 Wire synthesis flow

BonnRouteBuffer itself consists of the resource sharing algorithm and the subsequent randomized rounding and ripup-and-reroute procedure (see Section 2.4.1). It is embedded into a wire synthesis flow, see Figure 5.2. After the nets are routed and buffered, a global gate sizing is performed [Dab+18a]. This gate sizing can adapt the sizes of repeaters and non-repeater gates alike. Then, the placement is legalized. Recall that BonnRouteBuffer only bounds the local placement density, but does not prevent overlapping repeaters. Finally, the nets are assigned wire types and layers. Optimization routines after this flow are mostly not able to work on global wires, so this step discards the global routing.

The flow indicated in Figure 5.2 was used for the runs shown in Tables 5.2, 5.4, 5.5, 5.6 and 5.7.

5.4.2 Metrics and instances

The tested instances are specified in Table 5.1. We compare different buffering results according to the following metrics. The metrics of wACE4, wire length and number of vias are based on a fractional routing computed afterwards.

Reps : Number of repeaters inserted during buffering.

Wall Time : Total time needed for the entire run. Includes loading the chip and assessing the final result. Format is HH::MM::SS.

BRB Time : Running time of BonnRouteBuffer: sum of resource sharing time and the ripup-and-reroute routine. Format is HH::MM::SS.

WSL : Worst slack. Minimum difference between required arrival time and arrival time on any pin, measured in picoseconds:
$\text{WSL} = \min_{\text{pin } p}\{slack(p)\}$.

SNSL : Sum of negative slacks, measured in picoseconds:
$\text{SNSL} = \sum_{\text{pin } p} \min\{0, slack(p)\}$.

wACE4 : Average wiring congestion of the 5, 2, 1 and 0.5% most congested area [Wei+14].

WL : Total wire length. Considers all routes as going tile-center to tile-center. Measured in meters.

#Vias : Number of vias.

Power : Total power consumption. Includes both dynamic and leakage power. Measured in milliwatt.

$\sum$Load Vios : Total load violations. Sum over all capacitances exceeding the given load limit.

#Load Vios : Number of load violations. Counts the number of times a set load limit is exceeded.

$\sum$Slew Vios : Total slew violations. Sum over all slews exceeding the given slew limit.

#Slew Vios : Number of slew violations. Counts the number of times a set slew limit is exceeded.

Chip	Technology	Type	#Buffering instances	Chip area [mm^2]	#Routing Layers	#Wire types
C_1	5nm	ASIC	223,026	0.487×0.172	12	1
C_2	5nm	Processor	193,719	1.275×0.290	16	2
C_3	5nm	Processor	373,005	0.380×0.627	10	4
C_4	5nm	Processor	454,330	1.306×0.467	16	2
C_5	5nm	Processor	530,913	0.645×0.560	16	2
C_6	5nm	Processor	622,161	0.691×0.736	16	2
C_7	5nm	ASIC	45,384	0.063×0.160	8	1
C_8	5nm	ASIC	48,425	0.104×0.160	9	1
C_9	5nm	Processor	52,809	0.449×0.480	12	5
C_{10}	5nm	ASIC	68,742	0.845×0.264	10	2
C_{11}	5nm	Processor	75,284	4.800×2.105	16	5

Table 5.1: Basic characteristics of the test instances. #Buffering instances shows number of nets to be buffered. #Wire types counts available configurations of wire widths and spacings.

Chip	Run	BRB Time		# Reps		WSL		SNSL		wACE4		GR WL [m]		GR Vias		Power		∑Load Vios		∑Slew Vios	
C_1	old_pess=0.7	00:42:12		34,684		-465.0		-1,868,833		92.21 %		1.73 m		2,176,053		62,084		0		5,003	
	old_pess=0.8	00:47:38	+13 %	34,444	-1 %	-475.4	-10.4	-1,965,800	-96,967	92.36 %	0.15 %	1.73 m	+0.00 %	2,184,846	+0.40 %	61,959	-0.20 %	0	+0	5,459	+9 %
	old_pess=0.9	00:52:36	+25 %	34,026	-2 %	-496.2	-31.2	-1,970,639	-101,806	92.24 %	0.03 %	1.72 m	-0.12 %	2,171,721	-0.20 %	61,951	-0.21 %	0	+0	6,658	+33 %
	old_pess=1.0	00:55:39	+32 %	33,671	-3 %	-472.1	-7.1	-2,053,024	-184,191	92.18 %	-0.03 %	1.72 m	-0.23 %	2,171,949	-0.19 %	61,838	-0.40 %	0	+0	10,232	+105 %
	old_pess=1.1	00:53:29	+27 %	34,190	-1 %	-483.4	-18.4	-1,969,105	-100,272	92.60 %	0.39 %	1.72 m	-0.23 %	2,162,728	-0.61 %	61,838	-0.40 %	0	+0	18,372	+267 %
	old_pess=1.2	00:52:49	+25 %	34,167	-1 %	-490.9	-25.9	-1,970,232	-101,399	92.37 %	0.16 %	1.72 m	-0.29 %	2,151,239	-1.14 %	61,909	-0.28 %	3	+3	29,347	+487 %
C_2	old_pess=0.7	00:54:50		190,361		-146.1		-483,100		93.25 %		6.50 m		1,262,371		552		169		57,388	
	old_pess=0.8	00:51:40	-6 %	142,851	-25 %	-126.1	+20.0	-347,149	+135,951	93.13 %	-0.12 %	6.54 m	+0.61 %	1,242,172	-1.60 %	515	-6.57 %	12	-93 %	78,418	+37 %
	old_pess=0.9	00:49:21	-10 %	122,144	-36 %	-82.6	+63.5	-281,396	+201,704	94.95 %	1.70 %	6.72 m	+3.34 %	1,265,972	+0.29 %	502	-9.03 %	47	-72 %	121,235	+111 %
	old_pess=1.0	00:52:23	-4 %	112,796	-41 %	-106.5	+39.6	-328,182	+154,918	94.85 %	1.60 %	6.61 m	+1.61 %	1,213,381	-3.88 %	498	-9.76 %	25	-85 %	189,631	+230 %
	old_pess=1.1	00:48:04	-12 %	109,997	-42 %	-104.8	+41.4	-298,623	+184,477	94.45 %	1.20 %	6.63 m	+1.92 %	1,218,245	-3.50 %	494	-10.51 %	97	-43 %	239,639	+318 %
	old_pess=1.2	00:50:20	-8 %	107,641	-43 %	-106.3	+39.8	-305,233	+177,867	92.91 %	-0.34 %	6.45 m	-0.85 %	1,238,765	-1.87 %	493	-10.73 %	27	-84 %	303,307	+429 %
C_3	old_pess=0.7	00:17:04		122,383		-425.1		-4,161,703		92.17 %		4.53 m		1,062,385		678		150		11,301	
	old_pess=0.8	00:18:45	+10 %	105,888	-13 %	-426.7	-1.6	-3,927,991	+233,712	92.53 %	0.36 %	4.51 m	-0.53 %	1,024,900	-3.53 %	666	-1.84 %	323	+116 %	19,898	+76 %
	old_pess=0.9	00:20:29	+20 %	97,869	-20 %	-440.8	-15.7	-3,849,251	+312,452	92.01 %	-0.16 %	4.51 m	-0.55 %	1,014,571	-4.50 %	662	-2.31 %	80	-47 %	32,628	+189 %
	old_pess=1.0	00:19:10	+12 %	94,973	-22 %	-430.9	-5.8	-3,877,425	+284,277	91.87 %	-0.30 %	4.49 m	-0.99 %	1,004,960	-5.41 %	660	-2.66 %	96	-36 %	48,719	+331 %
	old_pess=1.1	00:19:29	+14 %	94,149	-23 %	-411.0	+14.1	-3,768,328	+393,375	92.92 %	0.75 %	4.47 m	-1.46 %	1,123,894	+5.79 %	660	-2.62 %	84	-44 %	82,261	+628 %
	old_pess=1.2	00:17:04	-0 %	93,788	-23 %	-421.7	+3.5	-3,834,659	+327,044	91.63 %	-0.54 %	4.47 m	-1.39 %	1,006,186	-5.29 %	660	-2.74 %	85	-43 %	118,091	+945 %
C_4	old_pess=0.7	01:53:07		318,694		-408.6		-1,166,069		96.09 %		13.45 m		2,244,235		610		2,608		394,238	
	old_pess=0.8	01:47:58	-5 %	246,260	-23 %	-407.4	+1.3	-872,897	+293,171	93.67 %	-2.42 %	13.10 m	-2.57 %	2,156,141	-3.93 %	572	-6.19 %	2,439	-6 %	423,783	+7 %
	old_pess=0.9	01:48:28	-4 %	207,301	-35 %	-156.0	+252.6	-715,481	+450,588	94.68 %	-1.41 %	13.11 m	-2.54 %	1,966,815	-12.36 %	556	-8.89 %	2,624	+1 %	501,130	+27 %
	old_pess=1.0	01:52:44	-0 %	188,465	-41 %	-202.9	+205.8	-716,863	+449,206	94.43 %	-1.66 %	13.00 m	-3.30 %	2,011,983	-10.35 %	548	-10.22 %	3,098	+19 %	686,064	+74 %
	old_pess=1.1	01:56:47	+3 %	183,183	-43 %	-195.1	+213.6	-859,045	+307,024	93.80 %	-2.29 %	12.96 m	-3.63 %	1,996,684	-11.03 %	547	-10.23 %	2,063	-21 %	799,604	+103 %
	old_pess=1.2	01:56:53	+3 %	178,364	-44 %	-159.2	+249.4	-661,245	+504,824	97.18 %	1.09 %	13.11 m	-2.51 %	2,024,132	-9.81 %	542	-11.09 %	1,874	-28 %	915,657	+132 %
C_5	old_pess=0.7	02:09:23		404,584		-151.6		-1,444,427		92.99 %		12.45 m		2,629,812		1,161		2,348		94,020	
	old_pess=0.8	02:05:01	-3 %	297,361	-27 %	-134.8	+16.8	-815,200	+629,226	92.97 %	-0.02 %	12.34 m	-0.91 %	2,435,946	-7.37 %	988	-14.84 %	2,875	+22 %	161,527	+72 %
	old_pess=0.9	01:59:45	-7 %	250,727	-38 %	-96.2	+55.5	-683,381	+761,046	97.06 %	4.07 %	12.29 m	-1.32 %	2,332,377	-11.31 %	924	-20.38 %	2,947	+26 %	227,406	+142 %
	old_pess=1.0	02:18:53	+7 %	228,608	-43 %	-100.4	+51.2	-610,531	+833,895	94.82 %	1.83 %	12.38 m	-0.54 %	2,511,459	-4.50 %	900	-22.47 %	2,665	+13 %	324,113	+245 %
	old_pess=1.1	02:02:57	-5 %	221,534	-45 %	-80.3	+71.3	-530,973	+913,454	97.08 %	4.09 %	12.61 m	+1.27 %	2,339,125	-11.05 %	890	-23.37 %	3,340	+42 %	418,052	+345 %
	old_pess=1.2	02:08:47	-0 %	215,801	-47 %	-113.4	+38.2	-678,202	+766,225	93.99 %	1.00 %	12.19 m	-2.05 %	2,252,510	-14.35 %	895	-22.93 %	3,155	+34 %	537,962	+472 %
C_6	old_pess=0.7	03:09:04		392,126		-190.3		-1,746,606		95.92 %		14.44 m		3,211,093		1,214		819		113,884	
	old_pess=0.8	02:55:00	-7 %	300,810	-23 %	-150.6	+39.7	-1,111,794	+634,812	108.57 %	12.65 %	14.48 m	+0.30 %	3,343,639	+4.13 %	1,136	-6.49 %	1,378	+68 %	186,945	+64 %
	old_pess=0.9	02:54:24	-8 %	254,944	-35 %	-136.4	+54.0	-1,092,546	+654,059	116.10 %	20.18 %	14.46 m	+0.16 %	2,937,676	-8.51 %	1,096	-9.76 %	685	-16 %	328,750	+189 %
	old_pess=1.0	02:41:07	-15 %	228,034	-42 %	-113.7	+76.6	-1,096,283	+650,323	121.13 %	25.21 %	14.46 m	+0.19 %	3,136,850	-2.31 %	1,074	-11.58 %	979	+20 %	449,593	+295 %
	old_pess=1.1	03:00:06	-5 %	218,688	-44 %	-126.1	+64.2	-1,153,566	+593,040	123.52 %	27.60 %	14.53 m	+0.64 %	3,108,075	-3.21 %	1,066	-12.18 %	981	+20 %	582,913	+412 %
	old_pess=1.2	03:03:30	-3 %	212,046	-46 %	-154.3	+36.0	-1,365,291	+381,314	108.31 %	12.39 %	14.33 m	-0.76 %	3,041,945	-5.27 %	1,063	-12.44 %	929	+13 %	710,211	+524 %

Table 5.2: Comparison of different slew limit factors used in BonnRouteBuffer on a set of large chips.

Chip	Run	wACE4	
C_5	old_pess=0.7	94.45 %	
	old_pess=0.8	93.88 %	-0.57 %
	old_pess=0.9	94.02 %	-0.43 %
	old_pess=1.0	93.75 %	-0.70 %
	old_pess=1.1	93.44 %	-1.01 %
	old_pess=1.2	93.63 %	-0.82 %
C_6	old_pess=0.7	118.30 %	
	old_pess=0.8	117.20 %	-1.10 %
	old_pess=0.9	116.72 %	-1.58 %
	old_pess=1.0	117.10 %	-1.20 %
	old_pess=1.1	116.57 %	-1.73 %
	old_pess=1.2	116.71 %	-1.59 %

Table 5.3: wACE4 values of C_5 and C_6 directly after BonnRouteBuffer. These are the runs from Table 5.2.

5.4.3 Different slew pessimism values

The main reason preventing the previous version of BonnRouteBuffer from computing better solutions was a high pessimism regarding slew limits. In fact, all slew limits were scaled down by a factor of 0.7 during the entire wire synthesis flow in the BonnTools. While this successfully reduced the number and degree of slew violations in the final result, it came at the cost of huge quality degradations in many other metrics.

In Tables 5.2 and 5.4, 6 runs are listed for each instance. The runs used pessimism factors on the slew limits of 0.7, 0.8, 0.9, 1.0, 1.1 and 1.2. Refer to Section 5.4.2 for an explanation of the shown metrics. Comparisons are always to the base run with a pessimism factor of 0.7, which was the default.

In Table 5.2, we compare the results on a set of large chips. Most prominently, the slew violations increase drastically with larger factors. This phenomenon is expected for the slew limit factor. The results demonstrate why a very pessimistic slew limit factor is necessary to keep slew violations in a reasonable range.

On the other hand, most metrics improve (in part very significantly) with less pessimism on most tested chips. Worst slack, sum of negative slacks, via count and power consumption all improve quite drastically except for C_1 and the worst slack on C_3. The wACE4 values increase or remain similar. Load violations and wire length changes in different directions depending on the instance.

On C_1, the slack stands out from the rest. It worsens, but not significantly. Also, no other metric improves notably with less pessimism. This shows that on this instance, lower slews are needed not only due to the slew limits, but also to achieve timing closure. Note that C_1 is the only instance in which the number of inserted repeaters does not differ notably between runs. This suggests that on C_1, only few repeaters were inserted to fix slew violations. Instead, almost all are needed for faster signals.

In contrast, on the instances C_2 to C_6, increasing the slew limit factor leads to up to 47% fewer inserted repeaters and improves timing values. This shows that many repeaters were needed only to satisfy the artificially low slew limits.

The significantly lower repeater count on runs with less pessimism explains the lower via count and power consumption. However, increasing the slew pessimism above 0.9 does not yield more changes to the number of repeaters, slack values or power consumption.

On C_6, the wACE4 value increases significantly for slew pessimism values other than 0.7. On C_5, it also increases. This only happens after layer assignment. See Table 5.3 for the wACE4 values directly after BonnRouteBuffer. C_6 is completely congested in all runs. Increasing the slew limit factor slightly improves congestion in both designs.

Chip	Run	BRB Time		# Reps		WSL		SNSL		wACE4		GR WL [m]		GR Vias		Power		∑Load Vios		∑Slew Vios	
C_7	old_pess=0.7	00:03:42		6,513		-189.0		-25,399		94.59 %		0.41 m		413,254		880		259		48,046	
	old_pess=0.8	00:03:36	-3 %	5,477	-16 %	-190.0	-1.0	-36,366	-10,966	93.31 %	-1.28 %	0.40 m	-3.40 %	404,014	-2.24 %	867	-1.48 %	139	-46 %	58,607	+22 %
	old_pess=0.9	00:03:37	-2 %	4,997	-23 %	-275.8	-86.8	-45,566	-20,167	94.04 %	-0.55 %	0.40 m	-2.43 %	406,443	-1.65 %	858	-2.53 %	153	-41 %	60,992	+27 %
	old_pess=1.0	00:03:27	-7 %	4,813	-26 %	-240.2	-51.2	-37,811	-12,411	93.66 %	-0.93 %	0.40 m	-4.13 %	402,123	-2.69 %	849	-3.50 %	195	-25 %	68,731	+43 %
	old_pess=1.1	00:03:30	-6 %	4,680	-28 %	-171.3	+17.7	-33,189	-7,789	93.59 %	-1.00 %	0.39 m	-4.61 %	401,304	-2.89 %	851	-3.26 %	212	-18 %	89,093	+85 %
	old_pess=1.2	00:03:39	-2 %	4,667	-28 %	-198.8	-9.8	-39,948	-14,549	93.30 %	-1.29 %	0.39 m	-5.83 %	400,725	-3.03 %	850	-3.38 %	207	-20 %	100,373	+109 %
C_8	old_pess=0.7	00:06:01		9,650		-38.4		-827		102.91 %		0.70 m		682,775		3,623		627		30,232	
	old_pess=0.8	00:05:53	-2 %	7,560	-22 %	-52.1	-13.7	-754	+73	104.02 %	1.11 %	0.70 m	-0.85 %	669,850	-1.89 %	3,496	-3.50 %	979	+56 %	28,785	-5 %
	old_pess=0.9	00:05:56	-1 %	6,670	-31 %	-42.8	-4.4	-611	+216	104.49 %	1.58 %	0.70 m	-0.71 %	668,673	-2.07 %	3,456	-4.60 %	1,272	+103 %	30,887	+2 %
	old_pess=1.0	00:06:09	+2 %	6,358	-34 %	-36.1	+2.4	-736	+91	105.21 %	2.30 %	0.70 m	-0.14 %	664,191	-2.72 %	3,434	-5.20 %	1,493	+138 %	36,089	+19 %
	old_pess=1.1	00:05:49	-3 %	6,237	-35 %	-59.9	-21.4	-396	+431	103.37 %	0.46 %	0.69 m	-2.13 %	659,246	-3.45 %	3,417	-5.69 %	1,721	+175 %	40,494	+34 %
	old_pess=1.2	00:06:04	+1 %	6,108	-37 %	-67.0	-28.5	-1,411	-584	104.11 %	1.20 %	0.69 m	-1.99 %	658,526	-3.55 %	3,423	-5.53 %	1,756	+180 %	54,247	+79 %
C_9	old_pess=0.7	00:11:01		21,501		-244.4		-419,159		91.83 %		2.33 m		275,972		163		206		14,677	
	old_pess=0.8	00:10:13	-7 %	16,767	-22 %	-166.1	+78.2	-277,435	+141,723	92.54 %	0.71 %	2.34 m	+0.26 %	284,316	+3.02 %	160	-2.04 %	206	-0 %	16,863	+15 %
	old_pess=0.9	00:10:48	-2 %	14,943	-31 %	-126.8	+117.6	-180,948	+238,210	92.71 %	0.88 %	2.34 m	+0.21 %	280,527	+1.65 %	159	-2.46 %	210	+2 %	18,836	+28 %
	old_pess=1.0	00:10:07	-8 %	14,109	-34 %	-162.6	+81.7	-298,396	+120,762	93.26 %	1.43 %	2.33 m	+0.00 %	277,940	+0.71 %	159	-2.71 %	209	+2 %	23,488	+60 %
	old_pess=1.1	00:10:49	-2 %	13,729	-36 %	-129.4	+114.9	-199,417	+219,742	92.53 %	0.70 %	2.33 m	-0.34 %	276,726	+0.27 %	157	-3.42 %	209	+1 %	32,201	+119 %
	old_pess=1.2	00:09:54	-10 %	13,618	-37 %	-128.8	+115.6	-204,862	+214,296	92.65 %	0.82 %	2.34 m	+0.09 %	280,957	+1.81 %	159	-2.51 %	209	+1 %	47,585	+224 %
C_{10}	old_pess=0.7	00:06:39		27,672		-98.9		-53,860		87.12 %		1.04 m		629,189		3,600		247		42,079	
	old_pess=0.8	00:06:03	-9 %	19,787	-28 %	-99.4	-0.5	-53,498	+362	87.69 %	0.57 %	1.03 m	-0.67 %	609,739	-3.09 %	3,496	-2.88 %	776	+214 %	42,795	+2 %
	old_pess=0.9	00:06:00	-10 %	15,519	-44 %	-98.9	+0.0	-54,152	-293	88.84 %	1.72 %	1.02 m	-1.83 %	603,756	-4.04 %	3,437	-4.52 %	837	+239 %	53,173	+26 %
	old_pess=1.0	00:06:06	-8 %	12,877	-53 %	-99.0	-0.1	-54,328	-468	89.00 %	1.88 %	1.01 m	-2.31 %	597,493	-5.04 %	3,406	-5.39 %	831	+237 %	63,205	+50 %
	old_pess=1.1	00:05:55	-11 %	12,166	-56 %	-109.4	-10.5	-54,479	-619	88.84 %	1.72 %	1.01 m	-2.60 %	593,936	-5.60 %	3,386	-5.95 %	881	+257 %	89,972	+114 %
	old_pess=1.2	00:06:11	-7 %	11,498	-58 %	-111.3	-12.4	-54,361	-501	88.95 %	1.83 %	1.01 m	-2.79 %	589,196	-6.36 %	3,376	-6.21 %	997	+304 %	120,596	+187 %
C_{11}	old_pess=0.7	02:24:03		105,897		-1051.0		-716,245		95.20 %		29.39 m		774,038		92		285,380		525,218	
	old_pess=0.8	01:39:58	-31 %	87,372	-17 %	-1045.1	+5.9	-743,572	-27,327	94.93 %	-0.27 %	29.34 m	-0.16 %	722,786	-6.62 %	110	+18.92 %	223,735	-22 %	672,441	+28 %
	old_pess=0.9	01:29:55	-38 %	75,524	-29 %	-1052.0	-1.1	-756,043	-39,798	95.91 %	0.71 %	29.22 m	-0.58 %	659,859	-14.75 %	109	+18.78 %	204,960	-28 %	689,339	+31 %
	old_pess=1.0	01:29:22	-38 %	69,265	-35 %	-1410.2	-359.2	-843,744	-127,498	98.28 %	3.08 %	29.30 m	-0.33 %	644,750	-16.70 %	88	-3.92 %	176,200	-38 %	765,606	+46 %
	old_pess=1.1	01:26:10	-40 %	65,892	-38 %	-1042.9	+8.1	-748,826	-32,581	96.18 %	0.98 %	29.12 m	-0.94 %	618,855	-20.05 %	105	+14.25 %	180,533	-37 %	891,085	+70 %
	old_pess=1.2	01:25:49	-40 %	63,607	-40 %	-1322.8	-271.8	-851,479	-135,233	95.90 %	0.70 %	29.07 m	-1.09 %	612,190	-20.91 %	87	-5.78 %	176,772	-38 %	1,259,550	+140 %

Table 5.4: Comparison of different slew limit factors used in BonnRouteBuffer on a set of small chips.

In Table 5.4, we compare the same runs on a set of chips with fewer nets. As BonnRouteBuffer is mainly designed for buffering large instances, these results are less significant. However, they still show the same phenomenon: On all chips, less pessimistic slew limits lead to fewer inserted repeaters in buffering. This reduction reaches 58% for `pess=1.2` and 44% on `pess=0.9` on C_{10}. Fewer repeaters in turn lead to a lower via count and power consumption. However, the timing values get worse in many runs.

On C_{11}, the variation between runs is very large. For `pess=1.0` and `pess=1.2`, the worst slack is significantly worse than in the other runs, while the power consumption is significantly better then for `pess=0.8`, `pess=0.9` and `pess=1.1`. Note also that C_{11} has significantly more load violations than any other chip.

In both sets of chips, changes in load violations vary from instance to instance: Increasing the slew limit factor sometimes reduces the load violations significantly (e.g. C_2, C_3, C_7, C_{11}). Sometimes it increases the load violations by a factor of up to 3 (e.g. C_{10}).

We can draw two main conclusions from these results. Firstly, it is necessary to consider the effect of slew on the signal delay explicitly. Currently, the dynamic program considers only the slew limits in the labels, not the actual slew that attains the used delay.

Secondly, slew and load violations are not modeled accurately through out the flow. In fact, the only place where they are considered is the dynamic program, where violations are assigned a penalty cost. Since violations are not modeled as resources in the resource sharing framework, they also do not influence the ripup-and-reroute step (see Section 2.4.1). Consequently, they are not fixed specifically after randomized rounding.

Chip	Run	BRB Time		# Reps		WSL		SNSL		wACE4		GR WL [m]		GR Vias		Power		∑Load Vios		∑Slew Vios	
C_1	old_pess=0.7	00:42:12		34,684		-465.0		-1,868,833		92.21 %		1.73 m		2,176,053		62,084		0		5,003	
	new_pess=0.7	00:58:18	+38 %	34,603	-0 %	-451.8	+13.2	-1,631,188	+237,645	92.50 %	0.29 %	1.74 m	+0.70 %	2,198,914	+1.05 %	61,939	-0.23 %	0	+0	357	-93 %
	new_pess=0.8	00:59:48	+42 %	33,965	-2 %	-447.2	+17.9	-1,669,791	+199,043	92.45 %	0.24 %	1.73 m	+0.52 %	2,190,989	+0.69 %	61,867	-0.35 %	0	+0	642	-87 %
	new_pess=0.9	01:00:46	+44 %	33,516	-3 %	-450.2	+14.8	-1,725,457	+143,376	92.18 %	-0.03 %	1.73 m	+0.46 %	2,179,181	+0.14 %	61,862	-0.36 %	0	+0	1,500	-70 %
	new_pess=1.0	00:59:54	+42 %	33,602	-3 %	-440.5	+24.5	-1,834,084	+34,749	92.15 %	-0.06 %	1.73 m	+0.52 %	2,180,337	+0.20 %	61,921	-0.26 %	0	+0	4,433	-11 %
	new_pess=1.1	00:59:03	+40 %	33,682	-3 %	-424.1	+41.0	-1,775,073	+93,760	92.26 %	0.05 %	1.73 m	+0.46 %	2,177,900	+0.08 %	61,822	-0.42 %	0	+0	11,926	+138 %
	new_pess=1.2	01:00:20	+43 %	33,933	-2 %	-421.1	+44.0	-1,706,391	+162,442	92.19 %	-0.02 %	1.73 m	+0.46 %	2,156,633	-0.89 %	61,856	-0.37 %	0	+0	23,014	+360 %
C_2	old_pess=0.7	00:54:50		190,361		-146.1		-483,100		93.25 %		6.50 m		1,262,371		552		169		57,388	
	new_pess=0.7	01:15:46	+38 %	172,837	-9 %	-161.6	-15.5	-543,898	-60,797	92.88 %	-0.37 %	6.50 m	-0.09 %	1,279,153	+1.33 %	553	+0.27 %	2	-99 %	230,443	+302 %
	new_pess=0.8	01:12:28	+32 %	142,463	-25 %	-133.1	+13.1	-444,674	+38,426	93.38 %	0.13 %	6.46 m	-0.71 %	1,323,237	+4.82 %	528	-4.25 %	2	-99 %	42,098	-27 %
	new_pess=0.9	01:13:53	+35 %	129,120	-32 %	-129.3	+16.8	-356,152	+126,948	93.11 %	-0.14 %	6.42 m	-1.32 %	1,288,185	+2.04 %	515	-6.69 %	2	-99 %	206,875	+260 %
	new_pess=1.0	01:05:25	+19 %	121,578	-36 %	-124.8	+21.3	-357,669	+125,431	93.91 %	0.66 %	6.49 m	-0.20 %	1,227,820	-2.74 %	507	-8.14 %	2	-99 %	60,388	+5 %
	new_pess=1.1	01:07:11	+23 %	118,812	-38 %	-110.2	+35.9	-331,192	+151,908	92.96 %	-0.29 %	6.42 m	-1.37 %	1,268,038	+0.45 %	502	-9.03 %	2	-99 %	103,026	+80 %
	new_pess=1.2	01:06:01	+20 %	116,388	-39 %	-160.9	-14.8	-404,941	+78,159	92.87 %	-0.38 %	6.37 m	-2.03 %	1,235,772	-2.11 %	503	-8.74 %	2	-99 %	166,972	+191 %
C_3	old_pess=0.7	00:17:04		122,383		-425.1		-4,161,703		92.17 %		4.53 m		1,062,385		678		150		11,301	
	new_pess=0.7	00:30:08	+77 %	126,805	+4 %	-438.1	-13.0	-4,422,258	-260,555	92.85 %	0.68 %	4.55 m	+0.46 %	1,074,845	+1.17 %	684	+0.92 %	101	-32 %	2,009	-82 %
	new_pess=0.8	00:30:05	+76 %	114,129	-7 %	-422.9	+2.2	-4,210,289	-48,586	92.77 %	0.60 %	4.54 m	+0.13 %	1,048,343	-1.32 %	673	-0.76 %	98	-35 %	4,001	-65 %
	new_pess=0.9	00:30:28	+78 %	109,025	-11 %	-445.5	-20.3	-4,117,295	+44,408	92.70 %	0.53 %	4.53 m	+0.04 %	1,042,232	-1.90 %	671	-1.02 %	69	-54 %	2,928	-74 %
	new_pess=1.0	00:27:48	+63 %	106,514	-13 %	-451.4	-26.3	-4,242,271	-80,569	92.81 %	0.64 %	4.52 m	-0.26 %	1,035,135	-2.56 %	671	-1.11 %	69	-54 %	11,897	+5 %
	new_pess=1.1	00:29:14	+71 %	105,574	-14 %	-439.3	-14.2	-4,030,942	+130,761	93.49 %	1.32 %	4.52 m	-0.24 %	1,033,747	-2.70 %	669	-1.36 %	88	-41 %	39,495	+249 %
	new_pess=1.2	00:27:43	+62 %	104,873	-14 %	-429.5	-4.4	-4,016,715	+144,988	93.51 %	1.34 %	4.52 m	-0.24 %	1,035,091	-2.57 %	671	-0.98 %	89	-40 %	77,214	+583 %
C_4	old_pess=0.7	01:53:07		318,694		-408.6		-1,166,069		96.09 %		13.45 m		2,244,235		610		2,608		394,238	
	new_pess=0.7	02:37:24	+39 %	278,424	-13 %	-249.3	+159.3	-1,451,031	-284,962	94.47 %	-1.62 %	13.29 m	-1.21 %	2,338,363	+4.19 %	608	-0.35 %	827	-68 %	359,840	-9 %
	new_pess=0.8	02:38:52	+40 %	231,095	-27 %	-168.0	+240.6	-972,950	+193,119	94.13 %	-1.96 %	13.10 m	-2.57 %	2,223,699	-0.92 %	584	-4.25 %	708	-73 %	263,237	-33 %
	new_pess=0.9	02:31:27	+34 %	208,435	-35 %	-163.9	+244.7	-920,780	+245,289	93.96 %	-2.13 %	13.03 m	-3.12 %	2,164,835	-3.54 %	573	-6.08 %	1,350	-48 %	249,063	-37 %
	new_pess=1.0	02:24:45	+28 %	195,332	-39 %	-142.4	+266.3	-797,365	+368,704	94.41 %	-1.68 %	12.97 m	-3.53 %	2,105,340	-6.19 %	561	-7.99 %	1,249	-52 %	260,770	-34 %
	new_pess=1.1	02:20:12	+24 %	189,418	-41 %	-146.6	+262.0	-769,772	+396,297	94.37 %	-1.72 %	12.95 m	-3.72 %	2,080,448	-7.30 %	558	-8.48 %	1,469	-44 %	367,268	-7 %
	new_pess=1.2	02:18:01	+22 %	185,207	-42 %	-187.8	+220.8	-723,828	+442,240	94.36 %	-1.73 %	12.93 m	-3.83 %	2,073,026	-7.63 %	552	-9.44 %	1,085	-58 %	525,709	+33 %
C_5	old_pess=0.7	02:09:23		404,584		-151.6		-1,444,427		92.99 %		12.45 m		2,629,812		1,161		2,348		94,020	
	new_pess=0.7	03:39:40	+70 %	359,266	-11 %	-205.6	-54.0	-1,976,550	-532,123	92.79 %	-0.20 %	12.37 m	-0.60 %	2,628,140	-0.06 %	1,135	-2.26 %	2,144	-9 %	99,901	+6 %
	new_pess=0.8	03:33:56	+65 %	286,173	-29 %	-210.1	-58.4	-1,427,425	+17,002	91.97 %	-1.02 %	12.18 m	-2.14 %	2,629,144	-0.03 %	1,050	-9.50 %	1,102	-53 %	74,100	-21 %
	new_pess=0.9	03:25:41	+59 %	253,796	-37 %	-178.9	-27.3	-1,136,545	+307,882	92.27 %	-0.72 %	12.08 m	-2.97 %	2,571,400	-2.22 %	1,001	-13.76 %	1,609	-31 %	70,095	-25 %
	new_pess=1.0	03:07:10	+45 %	237,308	-41 %	-147.4	+4.2	-1,036,849	+407,577	92.17 %	-0.82 %	12.01 m	-3.55 %	2,326,982	-11.52 %	973	-16.18 %	1,450	-38 %	86,621	-8 %
	new_pess=1.1	03:09:34	+47 %	232,177	-43 %	-141.9	+9.7	-953,777	+490,650	92.49 %	-0.50 %	12.02 m	-3.47 %	2,497,929	-5.01 %	961	-17.25 %	1,074	-54 %	168,716	+79 %
	new_pess=1.2	02:53:31	+34 %	226,943	-44 %	-165.2	-13.6	-1,204,173	+240,253	91.91 %	-1.08 %	11.92 m	-4.23 %	2,262,545	-13.97 %	974	-16.05 %	1,089	-54 %	292,973	+212 %
C_6	old_pess=0.7	03:09:04		392,126		-190.3		-1,746,606		95.92 %		14.44 m		3,211,093		1,214		819		113,884	
	new_pess=0.7	04:52:03	+54 %	361,112	-8 %	-277.9	-87.6	-2,340,932	-594,326	93.57 %	-2.35 %	14.15 m	-2.01 %	3,472,713	+8.15 %	1,200	-1.17 %	0	-100 %	123,627	+9 %
	new_pess=0.8	04:31:36	+44 %	297,888	-24 %	-166.5	+23.8	-1,394,264	+352,342	94.10 %	-1.82 %	14.08 m	-2.47 %	3,342,161	+4.08 %	1,144	-5.78 %	0	-100 %	85,790	-25 %
	new_pess=0.9	04:23:35	+39 %	263,050	-33 %	-169.9	+20.4	-1,296,890	+449,716	93.34 %	-2.58 %	13.72 m	-4.97 %	2,899,174	-9.71 %	1,113	-8.38 %	0	-100 %	93,682	-18 %
	new_pess=1.0	04:17:32	+36 %	241,687	-38 %	-166.5	+23.8	-1,440,502	+306,103	93.72 %	-2.20 %	13.70 m	-5.08 %	3,094,233	-3.64 %	1,093	-9.95 %	9	-99 %	134,916	+18 %
	new_pess=1.1	04:15:07	+35 %	233,370	-40 %	-157.5	+32.9	-1,373,542	+373,064	93.77 %	-2.15 %	13.61 m	-5.69 %	3,017,080	-6.04 %	1,084	-10.72 %	0	-100 %	232,931	+105 %
	new_pess=1.2	04:10:58	+33 %	227,381	-42 %	-178.9	+11.4	-1,415,013	+331,593	94.60 %	-1.32 %	13.74 m	-4.84 %	3,037,738	-5.40 %	1,081	-11.01 %	40	-95 %	368,127	+223 %

Table 5.5: Comparison of old vs. new BonnRouteBuffer on a set of large chips.

5.4.4 Previous vs. improved BonnRouteBuffer

Tables 5.5 and 5.6 show results for the same instances as Tables 5.2 and 5.4. The base run `old_pess=0.7` is the same. The other 6 runs use the same flow, but with the new and improved BonnRouteBuffer version containing all improvements from Section 5.3. They use slew limit factors of 0.7, 0.8, 0.9, 1.0, 1.1 and 1.2.

In Table 5.5, we see that all metrics could be improved on most of the instances. In particular, the load violations are significantly lower, regardless of slew pessimism. For `pess=0.8` and `pess=0.9`, slew violations also reduce quite significantly (except for `pess=0.9` on C_2).

Power consumption consistently improves by up to 17% (see C_5). The number of vias and the total net length also decrease (except for C_1 and net length on C_2). These improvements are due to the significantly lower number of inserted repeaters we see on all chips but C_1 and for slew limit factors higher than 0.7. The wACE4 values improve on C_4 to C_6 and remain similar on C_1 to C_3 for slew limit factors of up to 0.9. The sum of negative slacks is considerably better in most runs (except for `pess=0.7`). Lastly, the worst slack improves (except for `pess=0.7` and `pess=1.2` on C_2) on the instances C_1, C_2, C_4 and C_6. On C_4, this is a very significant improvement.

The worst slack on C_3 and C_5 could not be improved, although it does improve on C_5 when only changing the slew pessimism (see Table 5.2). On C_5, the sum of negative slacks could be improved for slew pessimism factors of 0.9 and higher. This suggests that the timing issues on this chip only concern individual nets. On C_1, the worst slack and sum of negative slacks could be consistently improved, whereas this did not happen when only increasing the slew limit factor (see Table 5.2).

The running time increases compared to the old version. This is mainly due to the new slew limit propagation.

Chip	Run	BRB Time		# Reps		WSL		SNSL		wACE4		GR WL [m]		GR Vias		Power		∑Load Vios		∑Slew Vios	
C_7	old_pess=0.7	00:03:42		6,513		-189.0		-25,399		94.59 %		0.41 m		413,254		880		259		48,046	
	new_pess=0.7	00:04:24	+19 %	6,158	-5 %	-174.2	+14.8	-27,420	-2,021	93.80 %	-0.79 %	0.40 m	-1.94 %	408,814	-1.07 %	864	-1.78 %	0	-100 %	32,100	-33 %
	new_pess=0.8	00:04:10	+12 %	5,427	-17 %	-140.0	+48.9	-19,576	+5,823	93.31 %	-1.28 %	0.40 m	-4.13 %	404,933	-2.01 %	846	-3.89 %	0	-100 %	33,900	-29 %
	new_pess=0.9	00:04:05	+10 %	5,083	-22 %	-146.2	+42.7	-30,227	-4,828	93.19 %	-1.40 %	0.39 m	-5.58 %	399,659	-3.29 %	843	-4.24 %	0	-100 %	46,179	-4 %
	new_pess=1.0	00:04:07	+11 %	4,788	-26 %	-234.3	-45.4	-54,604	-29,205	93.57 %	-1.02 %	0.39 m	-4.37 %	400,975	-2.97 %	843	-4.17 %	0	-100 %	54,291	+13 %
	new_pess=1.1	00:04:10	+13 %	4,755	-27 %	-134.3	+54.7	-33,544	-8,145	93.39 %	-1.20 %	0.39 m	-4.61 %	401,161	-2.93 %	834	-5.22 %	0	-100 %	61,581	+28 %
	new_pess=1.2	00:04:21	+17 %	4,738	-27 %	-176.1	+12.9	-38,143	-12,744	93.50 %	-1.09 %	0.39 m	-5.58 %	399,202	-3.40 %	838	-4.80 %	0	-100 %	69,668	+45 %
C_8	old_pess=0.7	00:06:01		9,650		-38.4		-827		102.91 %		0.70 m		682,775		3,623		627		30,232	
	new_pess=0.7	00:06:46	+12 %	8,380	-13 %	-24.3	+14.2	-328	+499	100.24 %	-2.67 %	0.69 m	-2.70 %	660,906	-3.20 %	3,439	-5.07 %	83	-87 %	20,899	-31 %
	new_pess=0.8	00:07:00	+16 %	7,107	-26 %	-34.5	+4.0	-113	+714	105.34 %	2.43 %	0.71 m	+1.42 %	674,870	-1.16 %	3,385	-6.55 %	84	-87 %	17,969	-41 %
	new_pess=0.9	00:07:46	+29 %	6,173	-36 %	-36.8	+1.7	-852	-25	103.44 %	0.53 %	0.70 m	-0.71 %	664,642	-2.66 %	3,351	-7.49 %	0	-100 %	27,123	-10 %
	new_pess=1.0	00:08:02	+33 %	5,497	-43 %	-44.1	-5.7	-756	+71	102.56 %	-0.35 %	0.69 m	-2.27 %	654,986	-4.07 %	3,350	-7.53 %	0	-100 %	33,401	+10 %
	new_pess=1.1	00:07:55	+31 %	5,660	-41 %	-51.0	-12.6	-1,550	-723	101.93 %	-0.98 %	0.68 m	-3.40 %	650,009	-4.80 %	3,357	-7.33 %	0	-100 %	48,634	+61 %
	new_pess=1.2	00:07:50	+30 %	5,780	-40 %	-51.9	-13.5	-1,523	-696	101.72 %	-1.19 %	0.68 m	-3.12 %	645,136	-5.51 %	3,353	-7.46 %	0	-100 %	50,641	+68 %
C_9	old_pess=0.7	00:11:01		21,501		-244.4		-419,159		91.83 %		2.33 m		275,972		163		206		14,677	
	new_pess=0.7	00:15:01	+36 %	20,993	-2 %	-161.4	+82.9	-213,697	+205,462	91.82 %	-0.01 %	2.36 m	+1.16 %	297,718	+7.88 %	164	+0.48 %	206	-0 %	13,005	-11 %
	new_pess=0.8	00:14:49	+34 %	16,995	-21 %	-152.1	+92.2	-161,052	+258,107	93.11 %	1.28 %	2.40 m	+2.96 %	301,480	+9.24 %	162	-0.84 %	206	-0 %	12,060	-18 %
	new_pess=0.9	00:14:10	+29 %	15,014	-30 %	-141.2	+103.2	-191,187	+227,971	92.04 %	0.21 %	2.36 m	+0.94 %	281,614	+2.04 %	160	-1.85 %	206	-0 %	12,692	-14 %
	new_pess=1.0	00:14:24	+31 %	14,196	-34 %	-129.0	+115.4	-201,894	+217,264	92.69 %	0.86 %	2.35 m	+0.81 %	277,362	+0.50 %	161	-1.45 %	206	-0 %	14,823	+1 %
	new_pess=1.1	00:12:30	+13 %	13,489	-37 %	-180.3	+64.1	-277,613	+141,546	92.07 %	0.24 %	2.36 m	+0.99 %	277,533	+0.57 %	159	-2.47 %	206	-0 %	27,877	+90 %
	new_pess=1.2	00:12:25	+13 %	12,979	-40 %	-217.0	+27.3	-377,546	+41,613	92.18 %	0.35 %	2.35 m	+0.64 %	274,973	-0.36 %	159	-2.22 %	206	-0 %	51,125	+248 %
C_{10}	old_pess=0.7	00:06:39		27,672		-98.9		-53,860		87.12 %		1.04 m		629,189		3,600		247		42,079	
	new_pess=0.7	00:08:20	+25 %	24,520	-11 %	-102.3	-3.4	-53,790	+70	89.03 %	1.91 %	1.04 m	+0.58 %	650,148	+3.33 %	3,588	-0.32 %	4	-99 %	42,963	+2 %
	new_pess=0.8	00:07:34	+14 %	17,600	-36 %	-99.0	-0.1	-53,624	+236	89.31 %	2.19 %	1.03 m	-0.58 %	628,830	-0.06 %	3,469	-3.63 %	190	-23 %	44,829	+7 %
	new_pess=0.9	00:07:55	+19 %	14,144	-49 %	-105.1	-6.3	-53,719	+141	89.30 %	2.18 %	1.03 m	-0.67 %	617,348	-1.88 %	3,407	-5.37 %	361	+46 %	39,095	-7 %
	new_pess=1.0	00:08:16	+24 %	12,493	-55 %	-111.2	-12.3	-54,470	-610	87.54 %	0.42 %	1.03 m	-0.67 %	615,397	-2.19 %	3,397	-5.65 %	489	+98 %	67,291	+60 %
	new_pess=1.1	00:07:17	+9 %	11,860	-57 %	-111.3	-12.4	-54,695	-835	87.66 %	0.54 %	1.04 m	-0.19 %	614,440	-2.34 %	3,378	-6.16 %	512	+108 %	99,754	+137 %
	new_pess=1.2	00:08:13	+24 %	11,185	-60 %	-109.6	-10.7	-54,626	-766	87.82 %	0.70 %	1.03 m	-0.67 %	614,530	-2.33 %	3,349	-6.98 %	573	+132 %	143,175	+240 %
C_{11}	old_pess=0.7	02:24:03		105,897		-1051.0		-716,245		95.20 %		29.39 m		774,038		92		285,380		525,218	
	new_pess=0.7	04:43:07	+97 %	131,597	+24 %	-1133.6	-82.6	-791,670	-75,425	95.43 %	0.23 %	29.74 m	+1.17 %	883,544	+14.15 %	118	+27.87 %	7,030	-98 %	457,184	-13 %
	new_pess=0.8	04:06:42	+71 %	117,128	+11 %	-1157.7	-106.7	-793,072	-76,827	95.30 %	0.10 %	29.58 m	+0.64 %	827,854	+6.95 %	112	+21.30 %	5,680	-98 %	460,851	-12 %
	new_pess=0.9	03:23:39	+41 %	100,797	-5 %	-1111.5	-60.5	-892,543	-176,298	95.95 %	0.75 %	29.31 m	-0.28 %	756,688	-2.24 %	139	+50.64 %	8,324	-97 %	546,561	+4 %
	new_pess=1.0	03:19:26	+38 %	91,940	-13 %	-1135.8	-84.8	-793,848	-77,603	95.03 %	-0.17 %	29.29 m	-0.35 %	722,705	-6.63 %	101	+9.82 %	6,027	-98 %	517,380	-1 %
	new_pess=1.1	03:15:28	+36 %	87,967	-17 %	-1386.8	-335.8	-930,118	-213,873	95.30 %	0.10 %	29.16 m	-0.81 %	706,666	-8.70 %	98	+6.44 %	8,477	-97 %	710,589	+35 %
	new_pess=1.2	03:05:28	+29 %	83,814	-21 %	-1352.8	-301.8	-885,173	-168,928	94.57 %	-0.63 %	29.06 m	-1.12 %	702,120	-9.29 %	124	+34.52 %	11,563	-96 %	851,958	+62 %

Table 5.6: Comparison of old vs. new BonnRouteBuffer on a set of small chips.

For the other set of instances, the picture is less clear as shown in Table 5.6. For pessimism values of 0.7 to 0.9, we still see a consistent reduction in slew violations across all chips (except for C_{10}), although this is less pronounced than above. The load violations could also be reduced significantly. On all chips except for C_{10}, this even holds regardless of slew pessimism values.

Consider the runs `new_pess=0.8` and `new_pess=0.9`. Here, we could improve both worst slack and sum of negative slacks quite significantly, except on C_{11}. Recall that C_{11} is special in the sense that the old BonnRouteBuffer produces an immense amount of load violations on this instance. Hence all other metrics worsen when fixing these violations.

In total, the slew limit factors 0.8 and 0.9 give the best results.

Chip	Run	BRB Time		# Reps		WSL		SNSL		wACE4		GR WL [m]		GR Vias		Power		∑Load Vios		∑Slew Vios	
C_1	old_P=25	00:42:12		34,684		-465.0		-1,868,833		92.21 %		1.73 m		2,176,053		62,084		0		5,003	
	new_P=20	00:59:00	+40 %	33,547	-3 %	-465.9	-0.8	-1,853,580	+15,253	92.43 %	0.22 %	1.73 m	+0.52 %	2,181,064	+0.23 %	61,899	-0.30 %	0	+0	2,170	-57 %
	new_P=15	01:04:20	+52 %	33,609	-3 %	-454.4	+10.7	-1,841,586	+27,248	92.36 %	0.15 %	1.73 m	+0.35 %	2,177,224	+0.05 %	61,939	-0.23 %	0	+0	2,203	-56 %
	new_P=10	00:54:46	+30 %	33,501	-3 %	-484.9	-19.9	-2,329,400	-460,567	92.06 %	-0.15 %	1.73 m	+0.23 %	2,179,044	+0.14 %	61,989	-0.15 %	0	+0	3,211	-36 %
C_2	old_P=25	00:54:50		190,361		-146.1		-483,100		93.25 %		6.50 m		1,262,371		552		169		57,388	
	new_P=20	00:54:58	+0 %	129,391	-32 %	-102.8	+43.3	-298,215	+184,885	93.72 %	0.47 %	6.50 m	-0.05 %	1,250,454	-0.94 %	514	-6.88 %	2	-99 %	44,181	-23 %
	new_P=15	00:45:57	-16 %	130,010	-32 %	-113.6	+32.6	-349,833	+133,267	93.82 %	0.57 %	6.47 m	-0.60 %	1,307,696	+3.59 %	515	-6.65 %	2	-99 %	44,788	-22 %
	new_P=10	00:40:53	-25 %	130,393	-32 %	-155.4	-9.3	-451,479	+31,622	92.74 %	-0.51 %	6.39 m	-1.77 %	1,278,934	+1.31 %	522	-5.36 %	2	-99 %	123,385	+115 %
C_3	old_P=25	00:17:04		122,383		-425.1		-4,161,703		92.17 %		4.53 m		1,062,385		678		150		11,301	
	new_P=20	00:22:17	+31 %	109,300	-11 %	-438.0	-12.9	-4,198,676	-36,973	93.11 %	0.94 %	4.53 m	-0.09 %	1,038,140	-2.28 %	672	-0.88 %	73	-52 %	5,475	-52 %
	new_P=15	00:18:35	+9 %	109,564	-10 %	-443.7	-18.6	-4,288,599	-126,896	92.55 %	0.38 %	4.53 m	+0.04 %	1,044,303	-1.70 %	673	-0.75 %	69	-54 %	6,383	-44 %
	new_P=10	00:13:26	-21 %	110,686	-10 %	-434.9	-9.8	-4,158,811	+2,892	92.39 %	0.22 %	4.52 m	-0.22 %	1,044,568	-1.68 %	671	-0.98 %	90	-40 %	6,110	-46 %
C_4	old_P=25	01:53:07		318,694		-408.6		-1,166,069		96.09 %		13.45 m		2,244,235		610		2,608		394,238	
	new_P=20	02:00:59	+7 %	208,611	-35 %	-146.7	+261.9	-885,963	+280,106	94.13 %	-1.96 %	13.04 m	-3.03 %	2,164,558	-3.55 %	572	-6.17 %	897	-66 %	266,538	-32 %
	new_P=15	01:42:16	-10 %	208,395	-35 %	-233.7	+174.9	-1,108,883	+57,185	93.96 %	-2.13 %	13.01 m	-3.28 %	2,159,356	-3.78 %	576	-5.63 %	1,142	-56 %	251,280	-36 %
	new_P=10	01:27:22	-23 %	208,110	-35 %	-171.2	+237.5	-993,595	+172,474	94.00 %	-2.09 %	13.05 m	-2.95 %	2,185,982	-2.60 %	578	-5.25 %	1,190	-54 %	242,516	-38 %
C_5	old_P=25	02:09:23		404,584		-151.6		-1,444,427		92.99 %		12.45 m		2,629,812		1,161		2,348		94,020	
	new_P=20	02:35:47	+20 %	253,372	-37 %	-236.8	-85.2	-1,307,373	+137,053	92.01 %	-0.98 %	12.06 m	-3.13 %	2,547,073	-3.15 %	1,005	-13.45 %	1,335	-43 %	71,447	-24 %
	new_P=15	02:15:18	+5 %	254,161	-37 %	-195.7	-44.0	-1,384,888	+59,538	92.25 %	-0.74 %	12.08 m	-2.97 %	2,557,632	-2.74 %	1,015	-12.52 %	1,602	-32 %	71,084	-24 %
	new_P=10	01:43:33	-20 %	256,348	-37 %	-236.7	-85.1	-1,275,592	+168,834	92.04 %	-0.95 %	12.07 m	-3.03 %	2,377,716	-9.59 %	1,007	-13.21 %	1,630	-31 %	70,260	-25 %
C_6	old_P=25	03:09:04		392,126		-190.3		-1,746,606		95.92 %		14.44 m		3,211,093		1,214		819		113,884	
	new_P=20	03:26:42	+9 %	263,429	-33 %	-165.8	+24.5	-1,426,033	+320,573	93.25 %	-2.67 %	13.77 m	-4.61 %	3,153,962	-1.78 %	1,114	-8.25 %	22	-97 %	98,175	-14 %
	new_P=15	02:45:47	-12 %	264,581	-33 %	-178.5	+11.9	-1,501,898	+244,707	93.54 %	-2.38 %	13.82 m	-4.26 %	2,925,707	-8.89 %	1,119	-7.83 %	26	-97 %	98,837	-13 %
	new_P=10	02:18:25	-27 %	266,190	-32 %	-207.4	-17.1	-1,512,656	+233,950	93.62 %	-2.30 %	13.83 m	-4.23 %	2,955,066	-7.97 %	1,125	-7.36 %	36	-96 %	98,833	-13 %
C_7	old_P=25	00:03:42		6,513		-189.0		-25,399		94.59 %		0.41 m		413,254		880		259		48,046	
	new_P=20	00:03:46	+2 %	5,061	-22 %	-222.2	-33.2	-43,263	-17,863	93.01 %	-1.58 %	0.39 m	-5.58 %	398,191	-3.64 %	850	-3.44 %	0	-100 %	50,489	+5 %
	new_P=15	00:03:37	-2 %	5,048	-22 %	-177.7	+11.2	-35,769	-10,369	93.32 %	-1.27 %	0.40 m	-3.88 %	404,034	-2.23 %	848	-3.63 %	0	-100 %	57,721	+20 %
	new_P=10	00:03:12	-14 %	4,968	-24 %	-245.2	-56.2	-40,630	-15,231	93.23 %	-1.36 %	0.39 m	-4.37 %	396,913	-3.95 %	864	-1.82 %	7	-97 %	74,699	+55 %
C_8	old_P=25	00:06:01		9,650		-38.4		-827		102.91 %		0.70 m		682,775		3,623		627		30,232	
	new_P=20	00:07:10	+19 %	6,496	-33 %	-43.5	-5.1	-1,064	-237	102.11 %	-0.80 %	0.69 m	-1.42 %	660,148	-3.31 %	3,377	-6.80 %	50	-92 %	24,767	-18 %
	new_P=15	00:06:14	+4 %	6,790	-30 %	-43.6	-5.2	-929	-102	102.06 %	-0.85 %	0.69 m	-1.42 %	659,498	-3.41 %	3,379	-6.73 %	145	-77 %	25,645	-15 %
	new_P=10	00:05:24	-10 %	7,177	-26 %	-65.7	-27.2	-498	+329	99.89 %	-3.02 %	0.68 m	-3.40 %	651,148	-4.63 %	3,380	-6.72 %	178	-72 %	31,652	+5 %
C_9	old_P=25	00:11:01		21,501		-244.4		-419,159		91.83 %		2.33 m		275,972		163		206		14,677	
	new_P=20	00:12:05	+10 %	15,055	-30 %	-165.8	+78.5	-191,980	+227,179	91.79 %	-0.04 %	2.35 m	+0.69 %	278,459	+0.90 %	160	-1.95 %	206	-0 %	12,803	-13 %
	new_P=15	00:10:43	-3 %	15,171	-29 %	-143.0	+101.4	-160,704	+258,454	92.31 %	0.48 %	2.36 m	+0.90 %	295,493	+7.07 %	160	-1.79 %	206	-0 %	12,991	-11 %
	new_P=10	00:09:19	-15 %	15,957	-26 %	-130.2	+114.1	-149,538	+269,621	93.05 %	1.22 %	2.37 m	+1.67 %	293,965	+6.52 %	160	-2.16 %	206	-0 %	14,219	-3 %
C_{10}	old_P=25	00:06:39		27,672		-98.9		-53,860		87.12 %		1.04 m		629,189		3,600		247		42,079	
	new_P=20	00:06:25	-3 %	14,219	-49 %	-105.1	-6.3	-53,794	+65	89.24 %	2.12 %	1.03 m	-0.87 %	620,606	-1.36 %	3,404	-5.44 %	376	+52 %	42,516	+1 %
	new_P=15	00:06:00	-10 %	14,291	-48 %	-105.1	-6.3	-53,711	+149	89.40 %	2.28 %	1.03 m	-0.67 %	619,446	-1.55 %	3,411	-5.26 %	434	+76 %	45,044	+7 %
	new_P=10	00:04:57	-25 %	14,657	-47 %	-111.2	-12.3	-53,867	-8	89.11 %	1.99 %	1.03 m	-0.58 %	619,543	-1.53 %	3,414	-5.18 %	378	+53 %	59,487	+41 %
C_{11}	old_P=25	02:24:03		105,897		-1051.0		-716,245		95.20 %		29.39 m		774,038		92		285,380		525,218	
	new_P=20	02:33:53	+7 %	100,873	-5 %	-1285.9	-234.9	-865,805	-149,560	95.01 %	-0.19 %	29.32 m	-0.26 %	761,693	-1.59 %	139	+51.48 %	9,702	-97 %	568,703	+8 %
	new_P=15	02:17:28	-5 %	100,292	-5 %	-1203.7	-152.8	-861,148	-144,903	96.14 %	0.94 %	29.45 m	+0.20 %	771,570	-0.32 %	107	+16.51 %	8,137	-97 %	576,920	+10 %
	new_P=10	01:55:59	-19 %	100,926	-5 %	-1220.0	-169.0	-906,195	-189,950	95.76 %	0.56 %	29.53 m	+0.48 %	780,783	+0.87 %	109	+18.38 %	9,509	-97 %	557,055	+6 %

Table 5.7: Comparison of old vs. new BonnRouteBuffer with different number of resource sharing phases.

Number of resource sharing phases

As the new BonnRouteBuffer version is slower, it makes sense to explore reducing the number of resource sharing phases. Table 5.7 contains the results for all instances with different number of phases. The base run `old_P=25` is the same as `old_pess=0.7`. It uses a slew pessimism factor of 0.7 and 25 resource sharing phases. The other runs use the new BonnRouteBuffer version, a slew pessimism factor of 0.9 and 20, 15, and 10 resource sharing phases each.

Most metrics do not significantly depend on the number of resource sharing phases. The sum of negative slacks and the sum of slew violations are better when running more phases. Running fewer phases significantly improves the running time. We propose to use 15 phases per run.

Chip	Run	Wall Time		# Reps		WSL		SNSL		wACE4		GR WL [m]		GR Vias		Power		∑Load Vios		∑Slew Vios	
C_1	old_pess=0.7_P=25	01:46:40		34,684		-432.0		-1,421,466		85.20 %		1.77 m		1,931,662		62,083		12		7,350	
	new_pess=0.8_P=25	02:16:25	+28 %	33,965	-2 %	-403.3	+28.7	-1,217,793	+203,673	85.29 %	0.09 %	1.77 m	-0.15 %	1,949,988	+0.95 %	61,865	-0.35 %	21	+73 %	2,192	-70 %
	new_pess=0.9_P=25	02:16:27	+28 %	33,516	-3 %	-424.4	+7.6	-1,332,024	+89,442	85.26 %	0.06 %	1.77 m	-0.05 %	1,941,745	+0.52 %	61,861	-0.36 %	45	+263 %	4,388	-40 %
	new_pess=0.9_P=15	02:20:31	+32 %	33,609	-3 %	-421.9	+10.1	-1,409,016	+12,450	85.26 %	0.06 %	1.77 m	-0.24 %	1,927,548	-0.21 %	61,938	-0.23 %	115	+830 %	4,239	-42 %
C_2	old_pess=0.7_P=25	02:19:57		190,361		-661.5		-1,564,577		85.78 %		6.42 m		2,867,839		548		1,012		539,020	
	new_pess=0.8_P=25	02:44:13	+17 %	142,463	-25 %	-550.1	+111.4	-1,361,300	+203,277	84.28 %	-1.50 %	6.31 m	-1.70 %	2,694,249	-6.05 %	524	-4.27 %	117	-88 %	412,119	-24 %
	new_pess=0.9_P=25	02:45:32	+18 %	129,120	-32 %	-587.4	+74.1	-1,505,625	+58,952	83.87 %	-1.91 %	6.29 m	-2.04 %	2,633,605	-8.17 %	511	-6.63 %	249	-75 %	676,789	+26 %
	new_pess=0.9_P=15	02:15:04	-3 %	130,010	-32 %	-485.8	+175.7	-1,369,848	+194,730	83.89 %	-1.89 %	6.29 m	-2.12 %	2,644,813	-7.78 %	512	-6.61 %	121	-88 %	547,954	+2 %
C_3	old_pess=0.7_P=25	01:54:41		122,383		-592.2		-6,303,326		84.94 %		4.54 m		3,721,313		668		149		141,938	
	new_pess=0.8_P=25	02:39:43	+39 %	114,129	-7 %	-577.2	+15.0	-6,216,134	+87,192	85.03 %	0.09 %	4.53 m	-0.18 %	3,671,906	-1.33 %	664	-0.66 %	106	-29 %	124,854	-12 %
	new_pess=0.9_P=25	02:41:38	+41 %	109,025	-11 %	-608.4	-16.2	-6,413,216	-109,889	84.75 %	-0.19 %	4.53 m	-0.26 %	3,646,133	-2.02 %	662	-0.97 %	69	-54 %	150,150	+6 %
	new_pess=0.9_P=15	02:21:56	+24 %	109,564	-10 %	-609.2	-17.0	-6,362,312	-58,986	84.95 %	0.01 %	4.53 m	-0.18 %	3,659,067	-1.67 %	664	-0.70 %	94	-37 %	150,208	+6 %
C_4	old_pess=0.7_P=25	06:00:42		318,694		-1232.8		-3,835,930		87.82 %		13.29 m		6,246,406		605		3,951		1,582,032	
	new_pess=0.8_P=25	07:42:02	+28 %	231,095	-27 %	-717.7	+515.1	-3,402,229	+433,701	85.15 %	-2.67 %	13.01 m	-2.10 %	5,859,876	-6.19 %	579	-4.18 %	1,984	-50 %	1,522,146	-4 %
	new_pess=0.9_P=25	07:39:40	+27 %	208,435	-35 %	-4757.8	-3,525.0	-3,536,728	+299,202	85.00 %	-2.82 %	12.96 m	-2.53 %	5,759,054	-7.80 %	569	-5.97 %	2,431	-38 %	1,582,021	-0 %
	new_pess=0.9_P=15	06:28:40	+8 %	208,395	-35 %	-1190.9	+42.0	-3,745,672	+90,259	84.97 %	-2.85 %	12.95 m	-2.55 %	5,757,364	-7.83 %	571	-5.57 %	1,913	-52 %	1,656,550	+5 %
C_5	old_pess=0.7_P=25	05:04:09		404,584		-727.9		-3,038,686		86.94 %		12.51 m		7,660,923		1,156		4,435		653,629	
	new_pess=0.8_P=25	06:55:00	+36 %	286,173	-29 %	-706.2	+21.8	-3,402,508	-363,822	85.03 %	-1.91 %	12.22 m	-2.28 %	7,021,130	-8.35 %	1,045	-9.54 %	1,618	-64 %	753,059	+15 %
	new_pess=0.9_P=25	06:58:48	+38 %	253,796	-37 %	-799.0	-71.0	-3,377,676	-338,991	84.41 %	-2.53 %	12.10 m	-3.25 %	6,838,958	-10.73 %	996	-13.81 %	2,421	-45 %	987,756	+51 %
	new_pess=0.9_P=15	05:48:01	+14 %	254,161	-37 %	-686.5	+41.4	-3,466,070	-427,384	84.36 %	-2.58 %	12.11 m	-3.17 %	6,837,977	-10.74 %	1,010	-12.57 %	2,116	-52 %	1,018,254	+56 %
C_6	old_pess=0.7_P=25	06:38:57		392,126		-1025.1		-4,289,955		85.45 %		14.15 m		8,660,785		1,198		2,013		1,299,792	
	new_pess=0.8_P=25	08:47:48	+32 %	297,888	-24 %	-1359.6	-334.6	-4,270,095	+19,860	83.95 %	-1.50 %	13.72 m	-3.00 %	8,124,994	-6.19 %	1,127	-5.93 %	40	-98 %	1,352,349	+4 %
	new_pess=0.9_P=25	08:30:26	+28 %	263,050	-33 %	-687.3	+337.8	-4,096,351	+193,605	84.56 %	-0.89 %	13.60 m	-3.85 %	7,997,044	-7.66 %	1,098	-8.40 %	59	-97 %	1,722,307	+33 %
	new_pess=0.9_P=15	06:51:02	+3 %	264,581	-33 %	-1031.4	-6.4	-4,516,968	-227,013	84.56 %	-0.89 %	13.62 m	-3.76 %	7,992,719	-7.71 %	1,104	-7.87 %	18	-99 %	1,882,451	+45 %
C_7	old_pess=0.7_P=25	00:21:05		6,513		-521.1		-343,175		92.74 %		0.39 m		368,426		880		304		40,917	
	new_pess=0.8_P=25	00:21:15	+1 %	5,427	-17 %	-484.3	+36.7	-413,255	-70,080	92.32 %	-0.42 %	0.38 m	-2.20 %	361,731	-1.82 %	846	-3.89 %	0	-100 %	32,841	-20 %
	new_pess=0.9_P=25	00:21:43	+3 %	5,083	-22 %	-564.6	-43.5	-397,611	-54,436	92.39 %	-0.35 %	0.38 m	-2.71 %	359,438	-2.44 %	843	-4.24 %	0	-100 %	47,301	+16 %
	new_pess=0.9_P=15	00:21:12	+1 %	5,048	-22 %	-657.8	-136.8	-496,795	-153,620	92.44 %	-0.30 %	0.38 m	-1.81 %	359,415	-2.45 %	849	-3.63 %	0	-100 %	63,020	+54 %
C_8	old_pess=0.7_P=25	00:23:16		9,650		-500.0		-121,751		92.46 %		0.62 m		564,458		3,623		667		47,203	
	new_pess=0.8_P=25	00:25:02	+8 %	7,107	-26 %	-644.7	-144.7	-130,848	-9,096	92.31 %	-0.15 %	0.62 m	-0.55 %	546,816	-3.13 %	3,386	-6.55 %	86	-87 %	61,145	+30 %
	new_pess=0.9_P=25	00:26:36	+14 %	6,173	-36 %	-762.8	-262.8	-143,970	-22,218	92.34 %	-0.12 %	0.62 m	-1.01 %	538,974	-4.51 %	3,352	-7.49 %	0	-100 %	96,753	+105 %
	new_pess=0.9_P=15	00:24:19	+5 %	6,790	-30 %	-681.6	-181.6	-114,720	+7,031	92.35 %	-0.11 %	0.62 m	-0.50 %	544,214	-3.59 %	3,379	-6.73 %	153	-77 %	65,466	+39 %
C_9	old_pess=0.7_P=25	00:36:54		21,501		-449.8		-744,872		81.02 %		2.32 m		798,340		161		411		169,919	
	new_pess=0.8_P=25	00:42:03	+14 %	16,995	-21 %	-757.5	-307.7	-509,992	+234,879	81.19 %	0.17 %	2.32 m	+0.13 %	789,427	-1.12 %	160	-0.65 %	206	-50 %	178,120	+5 %
	new_pess=0.9_P=25	00:43:42	+18 %	15,014	-30 %	-408.6	+41.2	-556,801	+188,070	81.02 %	0.00 %	2.32 m	+0.11 %	778,303	-2.51 %	159	-1.74 %	263	-36 %	200,133	+18 %
	new_pess=0.9_P=15	00:39:42	+8 %	15,171	-29 %	-502.0	-52.2	-529,379	+215,493	79.90 %	-1.12 %	2.31 m	-0.40 %	775,931	-2.81 %	158	-1.85 %	206	-50 %	207,525	+22 %
C_{10}	old_pess=0.7_P=25	00:28:00		27,672		-133.6		-58,057		75.62 %		1.04 m		701,497		3,599		245		56,652	
	new_pess=0.8_P=25	00:31:32	+13 %	17,600	-36 %	-122.6	+11.0	-59,471	-1,414	75.36 %	-0.26 %	1.03 m	-0.46 %	649,156	-7.46 %	3,469	-3.63 %	307	+26 %	82,191	+45 %
	new_pess=0.9_P=25	00:31:33	+13 %	14,144	-49 %	-132.1	+1.5	-59,735	-1,678	75.36 %	-0.26 %	1.03 m	-0.65 %	630,056	-10.18 %	3,406	-5.37 %	525	+114 %	90,748	+60 %
	new_pess=0.9_P=15	00:30:01	+7 %	14,291	-48 %	-131.4	+2.2	-59,718	-1,661	75.06 %	-0.56 %	1.03 m	-0.66 %	630,883	-10.07 %	3,410	-5.25 %	633	+159 %	95,679	+69 %
C_{11}	old_pess=0.7_P=25	05:21:52		105,897		-4301.3		-3,467,227		71.14 %		28.38 m		1,743,334		90		443,575		13,144,553	
	new_pess=0.8_P=25	07:00:11	+31 %	117,128	+11 %	-31046.3	-26,745.1	-5,834,243	-2,367,016	71.51 %	0.37 %	28.41 m	+0.11 %	1,859,663	+6.67 %	110	+21.90 %	18,269	-96 %	12,748,473	-3 %
	new_pess=0.9_P=25	06:26:15	+20 %	100,797	-5 %	-4930.6	-629.3	-4,227,912	-760,685	71.88 %	0.74 %	28.34 m	-0.13 %	1,750,252	+0.40 %	138	+53.01 %	25,349	-94 %	14,738,913	+12 %
	new_pess=0.9_P=15	05:18:42	-1 %	100,292	-5 %	-7078.6	-2,777.4	-4,490,963	-1,023,736	71.76 %	0.62 %	28.37 m	-0.03 %	1,736,303	-0.40 %	107	+18.65 %	20,732	-95 %	13,191,595	+0 %

Table 5.8: Comparison of old vs. new BonnRouteBuffer with subsequent RC-aware global routing.

Results after RC-aware global routing

Table 5.8 shows results after RC-aware global routing. The base run is the same as in the other tables. The three runs with the new BonnRouteBuffer version are the same ones using slew limit factors of 0.8 and 0.9 with 25 resource sharing phases and a slew limit factor of 0.9 with 15 resource sharing phases. After the wire synthesis flow (refer to Section 5.4.1), a new global routing was computed. This global routing uses arrival time customers (see Section 2.4.2) and the Elmore delay model (see Section 2.5.1). Note that the global routing oracle does not consider slew or load violations. It always uses the default wire type.

Changes to the worst slack and sum of negative slacks vary from chip to chip. The worst slack values on C_4 in `new_pess=0.9_P=25` and on C_{11} in `new_pess=0.8_P=25` and `new_pess=0.9_P=15` are due to enormous slew values of 6 to 32 nanoseconds. These nets have large fan-outs (e.g. 159 on C_{11}, `new_pess=0.8_P=25`) and would need a different wire type.

The wACE4 values, total net length and via count remain similar or improve on all chips. This also holds for cases in which the metrics were worse before RC-aware routing, e.g. on C_9 in `new_pess=0.8` (see Table 5.6).

Power decreases on all chips but C_{11}. This is the same tendency as measured before RC-aware routing.

Load violations improve on all chips but C_1 and C_{10}. On C_1, there are very few load violations in total (12 in `new_pess=0.9_P=15` and 4 to 5 in the other runs). So single nets have a large impact. On C_{10}, load violations already increased before RC-aware routing for a slew pessimism factor of 0.9.

Slew violations get worse on most chips. This is expected: The new BonnRouteBuffer inserts significantly fewer repeaters. With fewer repeaters, the slew-unaware routing oracle will produce more slew violations. Note that on C_1 and C_{11}, where the number of repeaters stays roughly the same, also the slew violations do not increase.

Chip	Run	#Nets	Wall Time		BRB Time	WSL		SNSL		wACE4		WL		Power		#Load Vios		#Slew Vios	
C_{12}	ibm	6,573	05:55:58			-59		-4,169		89.28 %		865,487		2		0		2	
	bonn	7,005	02:23:31	-60 %	00:05:54	-56	3	-4,745	-576	88.96 %	-0.32 %	884,982	+2.25 %	2	+0.81 %	0	+0 %	1	-50 %
C_{13}	ibm	75,563	14:26:16			-146		-18,168		86.65 %		18,562,298		10		21		92	
	bonn	81,988	1d 00:41:26	+71 %	06:07:48	-143	3	-19,051	-883	88.09 %	1.44 %	18,398,825	-0.88 %	11	+0.69 %	12	-43 %	103	+12 %
C_{14}	ibm	82,884	12:40:16			-20		-3,946		95.06 %		19,590,038		35		12		146	
	bonn	89,885	17:54:21	+41 %	05:29:44	-19	1	-3,302	644	91.10 %	-3.96 %	19,981,661	+2.00 %	35	-0.04 %	34	+183 %	176	+21 %
C_{15}	ibm	135,669	21:43:06			-19		-795		90.49 %		25,552,151		11		0		463	
	bonn	146,922	1d 05:38:58	+37 %	06:11:48	-11	8	-497	298	88.47 %	-2.02 %	25,432,548	-0.47 %	11	+0.98 %	0	+0 %	474	+2 %
C_{16}	ibm	181,097	16:06:43			-0		-4		87.40 %		48,113,703		16		3		188	
	bonn	193,905	1d 18:03:36	+161 %	10:06:55	-2	-1	-6	-2	87.34 %	-0.06 %	47,988,284	-0.26 %	16	-0.11 %	3	+0 %	199	+6 %
C_{17}	ibm	196,059	1d 08:39:35			-13		-400		96.91 %		42,928,285		48		0		160	
	bonn	197,113	2d 20:21:39	+109 %	1d 03:03:22	-10	3	-265	135	95.12 %	-1.79 %	41,594,203	-3.11 %	46	-2.70 %	0	+0 %	146	-9 %
C_{18}	ibm	271,370	1d 23:48:43			-27		-3,519		93.62 %		67,437,153		119		7		235	
	bonn	275,860	2d 02:51:12	+6 %	04:57:07	-25	2	-3,168	351	92.90 %	-0.72 %	66,948,638	-0.72 %	118	-1.24 %	10	+43 %	244	+4 %
C_{19}	ibm	360,557	1d 01:28:09			-1,002		-1,351,775		93.50 %		65,720,681		21		3		69	
	bonn	367,914	1d 13:58:02	+49 %	12:07:05	-1,002	1	-1,352,224	-449	101.17 %	7.67 %	64,833,106	-1.35 %	21	+0.67 %	0	-100 %	62	-10 %

Table 5.9: Comparison of the IBM PDS flow with and without BonnRouteBuffer.

5.4.5 BonnRouteBuffer in the IBM flow

BonnRouteBuffer is intended to run inside the PDS (placement driven synthesis) flow by IBM. The flow indicated in Figure 5.2 is run twice with 5 and 25 resource sharing phases each. In between, gate sizing is performed again and the inserted repeaters are removed. This makes sure the non-repeater gates are sized appropriately when the second (and main) BonnRouteBuffer pass starts.

Table 5.9 compares results after PDS of using the BonnRouteBuffer flow and the industry tool in use today. The runs denoted by `bonn` use the new BonnRouteBuffer version with a slew limit factor of 0.9.

In most metrics, BonnRouteBuffer performs similar or worse. The total running time (`Wall Time`) is much longer on most instances. Note that in this flow, BonnRouteBuffer is run with only 4 threads. Raising the number of threads will significantly improve the running time.

The worst slack is better on all chips but C_{16}. The sum of negative slacks is better on the chips that finish the PDS flow with little negative slack.

The wACE4 value is better on almost all tested chips. On C_2, wACE4 is so low in both runs that the increase does not have significant impact. On C_{19}, the result goes from almost routable (ibm results) to completely over-congested (bonn results).

On more congested chips, i.e. chips with wACE4 values of 90% and above (except on C_{19}), BonnRouteBuffer is able to find slightly better solutions with regard to timing. However, this does not justify the extensive running time.

5.5 Future work

In the testing of the improved BonnRouteBuffer version, more development opportunities came to light. As the focus of this thesis lies more on BonnPangea, they are not yet implemented.

5.5.1 Slew computations

Slew-dependent delay

After a net is routed and buffered by the resource sharing oracle, we propagate delay and slews through the buffered route. Here, we compute the delay depending on the propagated in-slew. However, this is the only place where the actual slew value is used to compute delay.

During the dynamic program of BonnRouteBuffer, delays through repeaters or wire segments are always computed using the slew target instead of an actual slew value. This is because the labels only know a slew limit, but no incoming slew value.

One way to address this is to add a *slew* property to the labels. Then, the dynamic program would start with many different labels at the net sinks, which differ only in their slew values. This increases the number of considered labels. Consequently, more label pruning is required to make this approach fast enough for practical applications.

As a second option, we can adapt parts of the buffered trees after the computation is complete. For this, we check how much the actual delay and the delay seen during the dynamic program differ. In sub-trees where this becomes too large, we restart the dynamic program to compute a better buffering. For this step, we can adapt the slew value used to compute the delay during the dynamic program.

Propagating slew through non-repeater gates

We need to propagate slews not only through a single buffered route, but also through the sink gates to the subsequent net source. This introduces a dependency between solutions for individual net customers. So it is difficult to incorporate this accurately into the resource sharing framework.

Currently, a default slew is assumed at the input of every non-repeater gate in the first resource sharing phase. Whenever the oracle routes and buffers a net N, the input slews at the successor nets are updated. However, this does not account for the fractional character of the solutions in resource sharing. Additionally, the input slew is not adapted after randomized rounding. This potentially leads to significant errors, as almost all nets are assigned solutions from a different phase than the last.

We propose propagating slews through the most recent solutions for all nets every few phases. In particular, we need to propagate slews through the solutions chosen in randomized rounding. Then we use the resulting values as input in the buffering of subsequent nets.

5.5.2 Fixing electrical violations in ripup-and-reroute

Neither slew violations nor load violations are modeled in the resource sharing framework. They are still considered during the oracle calls themselves. But in the ripup-and-reroute step, the nets to re-buffer are chosen solely depending on their resource usage. This means that nets with high violations are not visible here. We propose to additionally re-route and re-buffer nets with high violations in this step.

5.5.3 Fast buffering of timing-uncritical nets

For nets with low timing criticality, we can save running time by using a faster buffering routine. Instead of the cost-based algorithm considering all resources explicitly, it is much faster to only consider slew and load violations during the buffering. For nets where this results in reasonably buffered routes, we then do not need to perform the standard buffering algorithm.

Summary

In this thesis, we presented mathematical and practical progress in BonnPangea for port assignment and in BonnRouteBuffer for global buffering. Both are used in chip design. They are part of the BonnTools, developed by the Institute for Discrete Mathematics at the University of Bonn in collaboration with IBM.

We presented an approximation algorithm for the uniform cost-distance Steiner tree problem. Our approximation factor of <2.05 improves upon the previously best factor of 2.39 in [KH20]. Our algorithm is tight with respect to the lower bound $C_{SMT}(T \cup \{r\}) + D(T, r, w)$.

We considered the pangea routing problem in theory and practice. Here, the task is to compute a global routing on a chip partitioned into components ("continents"), such that the routing interfaces of each component (which are also part of the output) are as simple as possible. The interfaces should not enforce detours on source-to-sink connections and allow for routing with small net length and low congestion. BonnPangea is being used for the design of all IBM microprocessors. We saw that the currently used constraint of routes being allowed to enter every continent at most once has significant downsides regarding total net length.

After providing the first written account of the standard pangea flow as well as pangea replay, we considered the completely new problem of reusing a continent multiple times. This happens for example when there are several cores on a processor chip. We then need to compute equivalent interfaces on all instances of the continent. Pangea ReUse is the first tool in the industry able to deal with this case.

We presented a new collection of algorithms for Pangea ReUse that was developed and implemented at part of this thesis. For the step of computing port intervals, which are intervals on continent boundaries through which the routing has to go, we proposed several problem formulations and showed how to solve some variants optimally in polynomial time.

Pangea ReUse was a huge success in practice. Compared to the previous workflow, it saves one week in going from netlist to port creation. Pangea ReUse also enables the use of BonnPangea on designs that would not have been eligible without it.

Afterwards, we explored necessary conditions for eligible solutions. This led to an advanced algorithm for Pangea ReUse. We could show that this algorithm is able to solve much more general instances correctly. The algorithm especially focuses on aligning the interfaces of neighboring continents. This is necessary to correctly solve instances with several continent equivalence classes.

We then addressed BonnRouteBuffer. This is a tool for simultaneously routing and buffering large designs. Doing so, BonnRouteBuffer can optimize routing congestion, timing properties, power consumption and placement density.

We briefly reviewed the existing algorithms used inside BonnRouteBuffer and uncovered a major inaccuracy in the slew computations. Then we presented the improvements that were devised and implemented as part of this thesis. The computation of both the backwards propagation of slew limits and the forwards propagation of slews

were rewritten. The new slew model allowed us to reduce the pessimism in the slew calculations, leading to improvements in almost all metrics.

Additionally, a number of speed ups were discussed. These concern saving computational resources in situations in which exactness is of less importance. Finally, enhancements concerning via capacitance and load violations were outlined.

From the experimental results, we could draw several conclusions. Firstly, the inaccuracy in the slew calculations was a main reason why very sub-optimal buffering solutions were computed. A refinement of these computations already led to significant improvements in many metrics. Secondly, both slew and load violations play a major role for the buffering quality. This is not reflected well in the model. Finally, further speed-ups of BonnRouteBuffer are necessary to make it suitable to replace industrial buffering flows.

In the last section, we proposed future improvements to BonnRouteBuffer. The most prominent ones are global slew propagation and the explicit consideration of slew and load violations in the resource sharing framework. With respect to running time, we proposed to spend less time dealing with timing-uncritical nets.

Bibliography

[Alp+18] Charles J. Alpert, Wing-Kai Chow, Kwangsoo Han, Andrew B. Kahng, Zhuo Li, Derong Liu, and Sriram Venkatesh. "Prim-Dijkstra Revisited: Achieving Superior Timing-driven Routing Trees". In: *Proceedings of the 2018 International Symposium on Physical Design*. Association for Computing Machinery, 2018, pp. 10–17.

[Alp+95] C.J. Alpert, T.C. Hu, J.H. Huang, A.B. Kahng, and D. Karger. "Prim-Dijkstra tradeoffs for improved performance-driven routing tree design". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 14.7 (1995), pp. 890–896.

[Aro98] Sanjeev Arora. "Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems." In: *Journal of the ACM* 45.5 (1998), pp. 753–782.

[Bak90] H.B. Bakoglu. *Circuits, Interconnects, and Packaging for VLSI*. Addison-Wesley Publishing Company, 1990.

[Bar+06] Christoph Bartoschek, Stephan Held, Dieter Rautenbach, and Jens Vygen. "Efficient Generation of Short and Fast Repeater Tree Topologies". In: *Proceedings of the International Symposium on Physical Design* (2006), pp. 120–127.

[Bar+09] Christoph Bartoschek, Stephan Held, Dieter Rautenbach, and Jens Vygen. "Fast Buffering for Optimizing Worst Slack and Resource Consumption in Repeater Trees". In: *Proceedings of the International Symposium on Physical Design* (2009), pp. 43–50.

[Bat+02] S. Batterywala, N. Shenoy, W. Nicholls, and H. Zhou. "Track assignment: a desirable intermediate step between global routing and detailed routing". In: *IEEE/ACM International Conference on Computer Aided Design*. 2002, pp. 59–66.

[Bih15] Tilmann Bihler. "Rektilineare Steinerbäume mit längen- und richtungsbeschränkenden Blockaden". BA thesis. Universität Bonn, 2015.

[Bla+24] 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: *Discrete Optimization* 53 (2024).

[Boc+15] Adrian Bock, Stephan Held, Nicolas Kämmerling, and Ulrike Schorr. "Local search algorithms for timing-driven placement under arbitrary delay models". In: *Proceedings of the 52nd Annual Design Automation Conference*. DAC '15. Association for Computing Machinery, 2015.

[Byr+13] Jarosław Byrka, Fabrizio Grandoni, Thomas Rothvoss, and Laura Sanità. "Steiner tree approximation via iterative randomized rounding". In: *Journal of the ACM* 60.1 (2013), article 6.

[CC08] Miroslav Chlebík and Janka Chlebíková. "The Steiner tree problem on graphs: Inapproximability results". In: *Theoretical Computer Science* 406.3 (2008), pp. 207–214.

[Che+99] H.-M. Chen, H. Zhou, F. Y. Young, D. F. Wong, H. H. Yang, and N. Sherwani. "Integrated Floorplanning and Interconnect Planning". In: *Proceedings of the 1999 IEEE/ACM International Conference on Computer-aided Design*. ICCAD '99. IEEE Press, 1999, pp. 354–357.

[Chu+08] Julia Chuzhoy, Anupam Gupta, Joseph (Seffi) Naor, and Amitabh Sinha. "On the approximability of some network design problems". In: *ACM Trans. Algorithms* 4.2 (May 2008). ISSN: 1549-6325.

[Con91] Jason Cong. "Pin assignment with global routing for general cell designs". In: *IEEE transactions on computer-aided design of integrated circuits and systems* 10.11 (1991), pp. 1401–1412.

[CTY17] Gengjie Chen, Peishan Tu, and Evangeline F. Y. Young. "SALT: Provably good routing topology by a novel steiner shallow-light tree algorithm". In: *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 2017, pp. 569–576.

[Dab+18a] Siad Daboul, Nicolai Hähnle, Stephan Held, and Ulrike Schorr. "Provably Fast and Near-Optimum Gate Sizing". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2018).

[Dab+18b] Siad Daboul, Stephan Held, Jens Vygen, and Sonja Wittke. "An approximation algorithm for threshold voltage optimization". In: *ACM Transactions on Design Automation of Electronic Systems* 23.6 (2018).

[Dab+23] Siad Daboul, Stephan Held, Bento Natura, and Daniel Rotter. "Global Interconnect Optimization". In: *ACM Transactions on Design Automation of Electronic Systems* 5.72 (28 2023).

[Dab21] Siad Daboul. "Global Timing Optimization in Chip Design". Dissertation. Universität Bonn, 2021.

[Dij59] Edsger W Dijkstra. "A note on two problems in connexion with graphs". In: *Numerische mathematik* 1.1 (1959), pp. 269–271.

[Dur24] Victoria Durán Fernández. "Infeasibility Minimization in Track Assignment". BA thesis. Universität Bonn, 2024.

[Elm48] William C. 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.

[FHS23] Josefine Foos, Stephan Held, and Yannik Kyle Dustin Spitzley. "Tighter Approximation for the Uniform Cost-Distance Steiner Tree Problem". In: *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2023)* 275 (2023), 19:1–19:16.

[Foo20] Josefine Foos. "Algorithms for buffered Global Routing". MA thesis. Universität Bonn, 2020.

[Hei21] Paula Heinz. "Hardness of Cost-Distance Steiner Trees". BA thesis. Universität Bonn, 2021.

[Hel+18] Stephan Held, Dirk Müller, Daniel Rotter, Scheifele Rudolf, Vera Traub, and Jens Vygen. "Global routing with Timing Constraints". In: *IEEE. Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37 (2018), pp. 406–419.

[Hon+92] X.-L. Hong, J. Huang, C.-K. Cheng, and E. S. Kuh. "FARM: An Efficient Feed-through Pin Assignment Algorithm". In: *Proceedings of the 29th ACM/IEEE Design Automation Conference*. DAC '92. IEEE Computer Society Press, 1992, pp. 530–535.

[HR13] Stephan Held and Daniel Rotter. "Shallow-Light Steiner Arborescences with Vertex Delays". In: *Integer Programming and Combinatorial Optimization; Proceedings of the 16th International IPCO Conference* (2013), pp. 229–241.

[HS14] Stephan Held and Sophie Spirkl. "A fast algorithm for the rectilinear Steiner tree problem with length restrictions on obstacles". In: *Proceedings of the 2014 on International Symposium on Physical Design* (2014), pp. 37–44.

[HS22] Stephan Held and Yannik Kyle Dustin Spitzley. *Further Improvements on Approximating the Uniform Cost-Distance Steiner Tree Problem*. 2022. arXiv: 2211.03830 [cs.DS]. URL: https://arxiv.org/abs/2211.03830.

[Kar72] Richard M. Karp. "Reducibility among Combinatorial Problems". In: *Reducibility among Combinatorial Problems*. Ed. by Raymond E. Miller, James W. Thatcher, and Jean D. Bohlinger. Springer US, 1972, pp. 85–103.

[Kas+04] C.V. Kashyap, C.J. Alpert, F. Liu, and A. Devgan. "Closed-form expressions for extending step delay and slew metrics to ramp inputs for RC trees". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 23.4 (2004), pp. 509–516.

[KH20] A. Khazraei and S. Held. "An Improved Approximation Algorithm for the Uniform Cost-Distance Steiner Tree Problem". In: *WAOA* (2020).

[Kor72] Norman L. Koren. "Pin assignment in automated printed circuit board design". In: *Proceedings of the 9th Design Automation Workshop*. ACM. 1972, pp. 72–79.

[KRY95] S. Khuller, B. Raghavachari, and N. Young. "Balancing minimum spanning trees and shortest-path trees". In: *Algorithmica* 14 (1995), pp. 305–321.

[KWY96] T. Koide, S. Wakabayashi, and N. Yoshida. "Pin assignment with global routing for VLSI building block layout". In: *IEEE transactions on computer-aided design of integrated circuits and systems* 15.12 (1996), pp. 1575–1583.

[LCT22] Bohai Liu, Hao Cai, and Ji Tian. *Multiply instantiated block-aware top-level router*. Synopsys. 2022. URL: https://eda.icisc.cn/en/file/cacheFile/ee280e8ccf5647ff987ea0e9fc6d9357.pdf.

[Lüd23] Sebastian Lüderssen. "Local Search in VLSI Placement". MA thesis. Universität Bonn, 2023.

[MMP08] Adam Meyerson, Kamesh Munagala, and Serge Plotkin. "Cost-Distance: Two Metric Network Design". In: *SIAM Journal on Computing* 38.4 (2008), pp. 1648–1659.

[MRV11] Dirk Müller, Klaus Radke, and Jens Vygen. "Faster min-max resource sharing in theory and practice". In: *Mathematical Programming Computation* 3 (2011), pp. 1–35.

[Mun23] Max Mundt. “Port Assignment in presence of reusable chip-units”. MA thesis. Universität Bonn, 2023.

[Per23] Edgar Perner. “The Two-Metric Cost-Distance Problem in Chip Design”. BA thesis. Universität Bonn, 2023.

[PK09] Shashank Prasad and Anuj Kumar. “Simultaneous Routing and Feedthrough Algorithm to Decongest Top Channel”. In: *22nd International Conference on VLSI Design*. IEEE. 2009, pp. 399–403.

[Rao+92] Sailesh K. Rao, P. Sadayappan, Frank K. Hwang, and Peter W. Shor. “The rectilinear steiner arborescence problem”. In: *Algorithmica* 7 (1992), pp. 277–288.

[Rei23] Timo Reichert. “Cost-Distance Steinerbäume mit Blockaden”. BA thesis. Universität Bonn, 2023.

[Rot17] Daniel Rotter. “Timing-Constrained Global Routing with Buffered Steiner Trees”. Dissertation. Universität Bonn, 2017.

[RT87] P. Raghavan and C.D. Thompson. “Randomized rounding: a technique for provably good algorithms and algorithmic proofs”. In: *Combinatorica* 7 (1987), pp. 365–374.

[Sch09] Louis K. Scheffer. “Industrial Floorplanning and Prototyping”. In: *Handbook of Algorithms for Physical Design Automation*. Ed. by Charles J. Alpert, Dinesh P. Mehta, and Sachin S. Sapatnekar. CRC Press, Taylor & Francis Group, 2009. Chap. 13, pp. 257–273.

[TZ22] Vera Traub and Rico Zenklusen. “Local search for weighted tree augmentation and Steiner tree”. In: *Proc. ACM-SIAM Symposium on Discrete Algorithms (SODA ’22)*. 2022, pp. 3253–3272.

[Wei+14] Yaoguang Wei, Cliff Sze, Natarajan Viswanathan, Zhuo Li, Charles J. Alpert, Lakshmi Reddy, Andrew D. Huber, Gustavo E. Tellez, Douglas Keller, and Sachin S. Sapatnekar. “Techniques for scalable and effective routability evaluation”. In: *ACM Trans. Des. Autom. Electron. Syst.* 19.2 (2014).

[XTW01] Hua Xiang, Xiaoping Tang, and D.F. Wong. “An algorithm for simultaneous pin assignment and routing”. In: *IEEE/ACM International Conference on Computer Aided Design. ICCAD 2001. IEEE/ACM Digest of Technical Papers (Cat. No.01CH37281)*. 2001, pp. 232–238.