Track Assignment based Detailed Routing

Lukas Kühne

Born 12th January 1992 in Deggendorf, Germany

22nd July 2018

Master's Thesis Mathematics

Advisor: Prof. Dr. Jens Vygen

Second Advisor: Prof. Dr. Stephan Held

Research Institute for Discrete Mathematics

Mathematisch-Naturwissenschaftliche Fakultät der

Rheinischen Friedrich-Wilhelms-Universität Bonn

Contents

1 Introduction

This thesis is concerned with the design of very large-scale integrated (*VLSI*) circuits, or *chips*. These chips can be found in virtually all modern electrical products. Moreover, many problems arising in this area can be successfully solved by tools of combinatorial optimization.

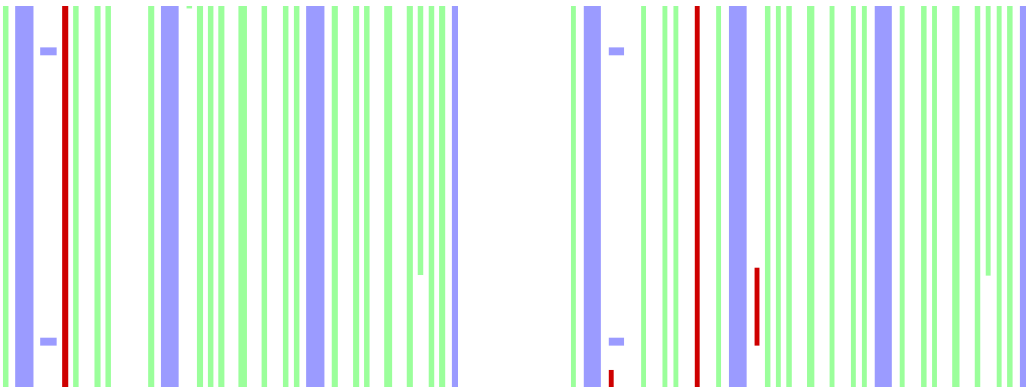
The focus of the thesis lies on the *routing* of a chip which is the task to compute its internal wiring. We are given a set of *pins* which are the access points of the previously placed circuits, together with the information which subsets of pins need to be connected by metal. Each such subset of pins is called a *net*. The length of the metal connections of a net is called the *netlength*. The process of routing tries to minimize the summed netlength while still achieving good timing and electrical properties of the chip. Additionally, many *design rules* have to be fulfilled in order to ensure that a chip can be manufactured reliably.

The task of routing is very complex due to the size of a modern chip which can consist of millions of nets. Therefore, routing is usually divided into the two stages *global routing* and *detailed routing*. In global routing the rough layout of the wiring of a net is computed. In this step, the chip is partitioned into regions, called *tiles*, to compute the connections of nets between different tiles. The goal in global routing is not only to minimize the netlength but also the congestion of wires across the entire chip. Subsequently, the exact wiring of a net is computed during the detailed routing process. Here, the wiring of the nets is required to follow the tiles based on the result of the global routing. Detailed routing is described in more detail in Section 2.2.

To further improve the detailed routing, an intermediate step, called *track assignment*, was proposed, e. g. by Batterywala et al. in [BSNZ02]. The basic idea of track assignment is to distribute the wires almost legally on routing tracks within their global routing tile. This way, one tries to find a better packing of the wires taking the global layout of the wires in a region into account. This could enhance the final wiring of the chip, since contrary to this approach the detailed router usually places the wires of the nets one by one without any global optimization. Refined versions of the track assignment that also try to minimize noise and crosstalk are discussed in [CCHL05] and [Hu12].

This thesis discusses how the track assignment solution can be used in detailed routing. Our approach has two components. On the one hand side, we treat the wires that are legally assigned to a track as reservations belonging to one specific

net. Subsequently, we block the area of a reservation during all path searches of nets different from the one the reservation belongs to. On the other hand, during the routing of each net, we discount the cost of the edges on the reservations of this net. After general preliminaries on the computation of shortest paths in Section 2.1, we give a formal description of our framework of reservations and discounted edge cost in Section 2.3. Figure 1 compares the output of the track assignment with our track assignment based detailed routing in one part of a chip.



(a) Track assignment. (b) Detailed routing.

Figure 1: Output of the track assignment and the track assignment based detailed routing in the same part on one chip. Blue rectangles are blockages. The green and the red lines are the placed wires on the chip. The green ones agree between the track assignment and the track assignment based detailed routing, whereas the red ones differ in the two methods.

An important speed-up technique for the computation of shortest paths are future cost functions. In Section 3, we develop a future cost function for our track assignment based detailed routing path searches. To enhance the introduced future cost function we propose a multilabel framework in Section 4 which tries to utilize the fact that a shortest path can not re-enter a reservation which it has left previously. Based on the multilabel framework, in Section 5 we give theoretical upper bounds on the number of labeled vertices during the path search algorithm in terms of the detour the found path takes compared with an optimal path without any blockages. We have implemented our algorithms within the existing BonnTools software (see [KRV07]) and present experimental results on real world instances in Section 6. Finally, we conclude by giving a brief summary of our results and pointing out directions for further research in Section 7.

Acknowledgments

I would like to thank everyone who supported me during the work on this thesis. I am particularly grateful to my adviser Prof. Dr. Jens Vygen for his excellent supervision. My special thanks go to Markus Ahrens and Niko Klewinghaus of the detailed routing team for numerous discussions which were always very helpful. I would also like to thank Markus Ahrens and Fabain Lenzen for proof-reading this thesis. Lastly, I am grateful to my family and friends for continuously supporting me throughout my studies.

2 Preliminaries

2.1 Shortest Paths and Future Cost Functions

It is a well-known combinatorial optimization problem to find a shortest path between two vertices of a graph. In this section, we give a formal definition of this problem and discuss algorithms to solve it efficiently. The notations concerning concepts from graph theory follow [KV12].

SHORTEST-PATH-PROBLEM

Instance: A graph G (directed or undirected), a cost function $c : E(G) \to \mathbb{R}$ and two vertices $s, t \in V(G)$.

Task: Find a shortest s-t-path S, i. e. one with minimal cost $c(E(S))$, or decide that t is not reachable from s.

If the graph does not contain a negative cycle, a shortest path can be efficiently found with the Moore-Bellman-Ford algorithm as described in [KV12]. We restrict ourselves to the case of a non-negative cost function which simplifies the problem significantly. In this case, one can use *Dijkstra's algorithm* [Dij59] to solve the SHORTEST-PATH-PROBLEM. Its run time is $\mathcal{O}(m + n \log n)$ for a graph G with $n = |V(G)|$ and $m = |E(G)|$ when implemented with Fibonacci heaps as shown by Fredman and Tarjan in [FT87].

There are many tools to speed-up Dijkstra's algorithm in different contexts of applications. Wagner and Willhalm give an overview in [WW07]. For our purposes the most useful approach is the so-called estimated *future cost*. This technique is related to the A^* *search* or *goal oriented path search* and was first mentioned by Hart, Nielsson and Raphael in [HNR68]. It was first applied in the context of VLSI-design by Rubin in [Rub74]. A more modern account of its application in BONNROUT is given by Peyer, Rautenbauch and Vygen in [PRV09] and by Gester et al. in [GMN+13]. Other applications include large road networks as described by Goldberg and Harrelson in [GH05]. Our work is based on the future cost computation for large grid graphs developed in [Hen16].

The main idea of future costs is that one includes the knowledge of a lower bound on the distance to the target in Dijkstra's algorithm. In this way, one tries to guide the path search towards the target rather than scanning the graph uniformly from the source. To formally describe Dijkstra's algorithm with future costs, we start by introducing the notion of a future cost function.

Definition 2.1. Let (G, c) be a directed graph with a cost function $c : E(G) \to \mathbb{R}_{\geq 0}$ and a target $t \in V(G)$. A map $l : V(G) \to \mathbb{R}_{\geq 0}$ is called *future cost function*, if it satisfies

(i) $c'((v,w)) := c((v,w)) + l(w) - l(v) \geq 0$ for all $(v,w) \in E(G)$ and

(ii) $l(t) = 0$.

In this situation, $c' : E(G) \to \mathbb{R}_{\geq 0}$ defines the *reduced costs* with respect to l.

Then, for a future cost function l with reduced costs c', it is easy to see that

$$c'(E(P)) = c(E(P)) + l(w) - l(v) \tag{1}$$

holds for any v-w-path P in the graph G. In the following, we describe Dijkstra's algorithm with a future cost function and prove its correctness. The exposition follows the one given in [GH05] and [Vyg15].

Algorithm 1: Dijkstra's algorithm with future costs

Input:

- A directed graph G,
- a cost function $c : E(G) \to \mathbb{R}_{\geq 0}$,
- a future cost function $l : V(G) \to \mathbb{R}_{\geq 0}$ and
- two vertices $s, t \in V(G)$.

Output: The length of a shortest s-t-path in G with respect to c or ∞ if no such path exists.

```
1  Set d(s) := 0. Set d(v) := ∞ for all v ∈ V(G) \ {s}. Set R := ∅.
2  while R ≠ V(G) do
3      Find a vertex v ∈ V(G) \ R with d(v) + l(v) = min_{w∈V(G)\R}(d(w) + l(w)).
4      if v = t then
5          return d(t).
6      Set R := R ∪ {v}.
7      for all w ∈ V(G)\R with (v, w) ∈ E(G) do
8          if d(w) > d(v) + c((v, w)) then
9              set d(w) := d(v) + c((v, w)) and p(w) := v.
10 return ∞.
```

Proposition 2.2. *Algorithm 1 correctly computes the length of a shortest s-t-path in G with respect to c.*

Proof. In Line 3 of Algorithm 1 we choose a vertex to add to the set R which minimizes the following expression:

$$\begin{aligned} d(w) + l(w) &= \mathrm{dist}_{(G,c)}(s, w) + l(w) \\ &\overset{(1)}{=} \mathrm{dist}_{(G,c')}(s, w) - l(w) + l(s) + l(w) \\ &= \mathrm{dist}_{(G,c')}(s, w) + l(s). \end{aligned}$$

Hence, running Algorithm 1 is equivalent to running Dijkstra's algorithm on the instance (G', c') with source s' and target t where G' is the graph defined by

$$\begin{aligned} V(G') &:= V(G) \cup \{s'\}, \\ E(G') &:= E(G) \cup \{(s', s)\}, \end{aligned}$$

and c' are the reduced cost on $E(G)$ and defined to be $l(s)$ on the additional edge (s', s). Dijkstra's algorithm works correctly on the instance (G', c') since the reduced cost c' are non-negative on G' by the definition of future cost functions. Thus, the algorithm returns the length of a shortest s'-t-path with respect to c'. Due to equation (1) and the requirement $l(t) = 0$ of future cost functions, this equals the length of a shortest s-t-path with respect to c. □

Definition 2.3. A vertex $v \in V(G)$ is called *permanently labeled* if at some point $v \in R$ holds.

Remark 2.4. One can also call Dijkstra's algorithm without an explicit target and skipping the condition of termination in Line 4 of Algorithm 1. Then as the result, one returns the map $d : V(G) \to \mathbb{R}_{\geq 0}$ giving the length of shortest s-v-path with respect to c for any vertex $v \in V(G)$.

Remark 2.5. In practice, one often encounters path search problems with multiple source and target vertices. This means that one tries to find a shortest path from one of the source to one of the target vertices. Algorithm 1 also covers these cases with the following minor adjustments. In the case of multiple source vertices $S \subset V(G)$, one sets $d(s) = 0$ in Line 1 for every $s \in S$. The case of multiple target vertices $T \subset V(G)$ can be handled by replacing the condition of termination in Line 4 by $v \in T$.

One possibility to obtain future cost functions is described in the following lemma.

Lemma 2.6. *Let G be a directed graph with cost $c : E(G) \to \mathbb{R}_{\geq 0}$ and a target $t \in V(G)$. Define a map $l : V(G) \to \mathbb{R}_{\geq 0}$ by setting for $v \in V(G)$*

$$l(v) := \mathrm{dist}_{(G,c)}(v, t).$$

Then, the map l is a future cost function.

Proof. We trivially have $l(t) = \mathrm{dist}_{(G,c)}(t, t) = 0$ since the cost function c is non-negative. Further, let $e = (v, w) \in E(G)$ be any edge in G. Then,

$$\mathrm{dist}_{(G,c)}(v, t) \leq c(e) + \mathrm{dist}_{(G,c)}(w, t),$$

holds since the length of a v-t path starting with the edge e is at least as long as the length of any shortest v-t path with respect to c. Therefore, the map l is a future cost function. □

In general, it is difficult to compute $\mathrm{dist}_{(G,c)}(v, t)$ efficiently for all vertices $v \in V(G)$. In our application however, we will encounter two graphs H, G with H being a subgraph of G. We will execute Algorithm 1 on the graph H and can compute $\mathrm{dist}_{(G,c)}(v, t)$ for all vertices $v \in V(H) \subset V(G)$ efficiently. In this situation, restricting the future cost function obtained from Lemma 2.6 for the graph G to the subgraph H will also be a future cost function for the graph H as shown in the next lemma.

Lemma 2.7. *Let G be a directed graph with cost function $c : E(G) \to \mathbb{R}_{\geq 0}$, a target $t \in V(G)$ and a future cost function $l : V(G) \to \mathbb{R}_{\geq 0}$. Furthermore, let H be a subgraph of G with $t \in V(H)$, c_H the restriction of c on $E(H)$ and l_H the restriction of l on $V(H)$. Then, l_H is a future cost function for the graph H.*

Proof. Due to the fact $t \in V(H)$, the condition $l_H(t) = 0$ trivially holds. Now, let $e = (v, w) \in E(H)$ be an edge in H, which is then by assumption also in the graph G. Hence, due to the definition of the future cost function l on G, we have

$$0 \leq c((v, w)) + l(w) - l(v) = c_H((v, w)) + l_H(w) - l_H(v).$$

Therefore, l_H is a future cost function for the graph H. □

2.2 Routing Graph and Path Search Problem in VLSI–Design

In the following, we give the basic definitions used to formalize detailed routing in the path search context. We start by defining a grid graph and its cost function.

Definition 2.8. For fixed $x^G_{\min}, x^G_{\max}, y^G_{\min}, y^G_{\max}, z^G_{\min}, z^G_{\max} \in \mathbb{Z}$ we define the three-dimensional (directed) *grid graph* G by setting

$$V(G) := \{(x, y, z) \in \mathbb{Z}^3 : x^G_{\min} \leq x \leq x^G_{\max}, y^G_{\min} \leq y \leq y^G_{\max}, z^G_{\min} \leq z \leq z^G_{\max}\},$$
$$E(G) := \{(v, w) : v, w \in V(G), \|v - w\|_1 = 1\}.$$

We define a *cost function* $c : E(G) \to \mathbb{R}_{\geq 0}$ in a grid graph G by fixing the costs c^l_x, c^l_y and c^l_z for one unit in x-, y- and z-direction respectively on each layer $z_{\min} \leq l < z_{\max}$. This means:

$$c(((x_1, y_1, z_1), (x_2, y_2, z_2))) = \begin{cases} c^{z_1}_z & \text{if } x_1 = x_2, y_1 = y_2 \text{ and } z_1 < z_2, \\ c^{z_1}_y & \text{if } x_1 = x_2 \text{ and } z_1 = z_2, \\ c^{z_1}_x & \text{if } y_1 = y_2 \text{ and } z_1 = z_2. \end{cases}$$

We assume that $1 \leq c^l_x, c^l_y, c^l_z$ holds for any layer $z^G_{\min} \leq l \leq z^G_{\max}$. For any vertex $p \in V(G)$, we denote by p_x, p_y and p_z its x-, y- and z-coordinate respectively.

Henke showed in [Hen16] that one can compute $\text{dist}_{(G,c)}(u, v)$ for $u, v \in V(G)$ in $\mathcal{O}\left(z^G_{\max} - z^G_{\min}\right)$-time. In our application, $z^G_{\max} - z^G_{\min}$ is a small constant < 20 and therefore we may assume that $\text{dist}_{(G,c)}(u, v)$ can be computed in constant time.

Due to blockages and previously routed nets, one usually can not use the entire grid graph to route a net. Additionally, one restricts the grid graph to a certain number of tracks on each layer on which wires can be placed. Therefore, we will consider in the following a subgraph G_T of G which we will call the *routing graph*. The graph G_T is sometimes also called *track graph*. We also consider the same cost function c on G_T by restricting $c : E(G) \to \mathbb{R}_{\geq 0}$ to $E(G_T)$. Hence, one of the main problems of routing in VLSI-design can be stated as follows:

PATH-SEARCH-PROBLEM

Instance: A routing graph G_T with the cost function $c : E(G_T) \to \mathbb{R}$ as defined above and two vertices $s, t \in V(G_T)$.

Task: Find a shortest s-t-path S in G_T with respect to c, or decide that t is not reachable from s in G_T.

2.3 Reservations and Discounted Costs

As outlined in the introduction, the aim of this thesis is to improve the detailed routing by using the output of the track assignment. We describe an approach where we reserve the assigned tracks in the routing graph, so that their space is guaranteed to be available when the respective net is routed.

Definition 2.9. For a grid graph G, a *reservation* R is a rectangle defined by

$$R := [x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}] \times \{z\},$$

where $x_{\min}$, $x_{\max}$, $y_{\min}$, $y_{\max}$, $z \in \mathbb{Z}$ and at least one of the defining intervals is in fact a point, i. e. R is a 1-dimensional rectangle. Further, we assume $x^G_{\min} \leq x_{\min} \leq x_{\max} \leq x^G_{\max}$, $y^G_{\min} \leq y_{\min} \leq y_{\max} \leq y^G_{\max}$ and $z^G_{\min} \leq z \leq z^G_{\max}$, i. e. R lies within the grid graph G. We denote by $\mathcal{R} := \{R_1, \ldots, R_k\}$ a *set of reservations* where each R_i for $i = 1, \ldots, k$ is a reservation. The set of reservations does not need to be disjoint.

We call the rectangle R reservation because we reserve the vertices lying in this rectangle for the path search of the specific net a reservation belongs to. Formally, we assume that for any reservation R and for any edge $e = (v, w) \in E(G)$ with $v, w \in R$ we also have $e \in G_T$.

The purpose of introducing a set of reservations is to give the edges on these rectangles a preference in the path search algorithm. To this end, we define the notion of discounted costs in the following definition.

Definition 2.10. For a set of reservations $\mathcal{R}$ we define the *discounted edge costs* $c^{\mathcal{R}} : E(G) \to \mathbb{R}_{\geq 0}$ by setting for an edge $(u, v) \in E(G)$

$$c^{\mathcal{R}}((u,v)) = \begin{cases} q \cdot c((u,v)) & \text{if } u, v \in R_i \text{ for some } R_i \in \mathcal{R}, \\ c((u,v)) & \text{otherwise,} \end{cases}$$

where $0 < q < 1$ is a constant *reduction factor*. Again, we also define the restriction of $c^{\mathcal{R}}$ to $E(G_T)$ to be the discounted cost function $c^{\mathcal{R}}$ on G_T.

For the rest of this thesis we make the following assumption:

$$qc_x^{z_1} \leq c_x^{z_2} \text{ and } qc_y^{z_1} \leq c_y^{z_2}, \tag{2}$$

for any planes $z^G_{\min} \leq z_1, z_2 \leq z^G_{\max}$. This means that the discounted costs $c^{\mathcal{R}}$ on a reservation are less or equal than the usual costs c in that direction on any plane. The assumption is satisfied on all our instances in practice.

The following describes a trivial way to define a future cost function for the routing graph G_T with cost $c^{\mathcal{R}}$.

Proposition 2.11. *Let $l : V(G_T) \to \mathbb{R}_{\geq 0}$ be any future cost function in the routing graph G_T with respect to the cost c. Then, the map $l_t : V(G_T) \to \mathbb{R}_{\geq 0}$ defined by*

$$l_t(v) := ql(v),$$

for any $v \in V(G)$ defines a future cost function in the routing graph G_T with respect to the discounted cost $c^{\mathcal{R}}$.

Proof. We trivially have $l_t(t) = ql(t) = 0$ for the target vertex $t \in V(G)$ by definition of the future cost function l. Let $e = (v, w) \in E(G_T)$ be an arbitrary edge in G_T. Then due to the definition of the future cost function l, we have

$$0 \leq c((v, w)) + l(w) - l(v).$$

Multiplying the inequality by q yields

$$0 \leq qc((v, w)) + ql(w) - ql(v) \leq c^{\mathcal{R}}((v, w)) + l_t(w) - l_t(v),$$

since $0 < q < 1$ holds by definition of $c^{\mathcal{R}}$. Therefore, l_t is a future cost function with respect to the cost $c^{\mathcal{R}}$. □

By Lemmata 2.6 and 2.7 the map $l : V(G_T) \to \mathbb{R}\geq 0$ defined by $l(v) = \operatorname{dist}_{(G,c)}(v, t)$ for any $v \in V(G_T)$ defines a future cost function in G_T with respect to the cost c. Then by Proposition 2.11, we can define the corresponding *trivial future cost function with reservations* $l_t : V(G_T) \to \mathbb{R}_{\geq 0}$ by setting for any $v \in V(G_T)$

$$l_t(v) := q \operatorname{dist}_{(G,c)}(v, t).$$

3 Future Cost Function with Reservations

A main task of this thesis is to determine a future cost function in the routing graph G_T with respect to the discounted cost function $c^{\mathcal{R}}$ for a given set of reservations $\mathcal{R}$. By the Lemmata 2.6 and 2.7, the map $l : V(G_T) \to \mathbb{R}_{\geq 0}$ with $l(v) = \text{dist}_{(G,c^{\mathcal{R}})}(v,t)$ for any $v \in V(G)$ and a fixed target t is a future cost function in the routing graph G_T. This motivates the following problem, which we will address in this section.

Shortest-Path-to-Target-With-Reservations (SPTTWR)	
Instance:	A grid graph G, a set of reservations $\mathcal{R}$ and a target $t \in V(G)$.
Task:	Compute $\text{dist}_{(G,c^{\mathcal{R}})}(v,t)$ for any vertex $v \in V(G)$ efficiently.

3.1 Meta Dijkstra Preprocessing

To fulfill this task, we propose a preprocessing method which precomputes $\text{dist}_{(G,c^{\mathcal{R}})}(p,t)$ for $p \in P$ where $P \subset V(G)$ is a certain set of distinguished vertices on the reservations $\mathcal{R}$.

Before stating the algorithm, we introduce the following two auxiliary algorithms. Both take a point $v \in \mathbb{R}^3$ and a reservation R as their input.

Line-Projection(v,R): The algorithm projects the point v onto the line spanned by the given reservation R which is in fact a line segment.

Closest-Point(v,R): The algorithm returns the closest point on the reservation R to a given vertex v with respect to the L_1 distance.

Let $\mathcal{R} = \{R_1, \ldots, R_k\}$ with $R_i := [x^i_{\min}, x^i_{\max}] \times [y^i_{\min}, y^i_{\max}] \times [z^i]$ be a set of reservations in the grid graph G.

Definition 3.1. (i) For a given instance of a grid graph G, reservations $\mathcal{R}$ and a target $t \in V(G)$, we call the vertices $p \in P$ *gate vertices* where P is the set returned by Algorithm 2.

(ii) We call the gate vertices that are endpoints of all the reservations they lie on *outer gate vertices* (OGV) and the other ones *inner gate vertices* (IGV). Note that each inner gate vertex v is induced by one or more outer gate vertices or the target vertex t, which we call the creators of v.

(iii) For an s-t-path S, an edge $e = (u,v) \in E(S)$ is called *gate-violating* if for some reservation $R \in \mathcal{R}$ we either have

Algorithm 2: SPTTWR-Preprocessing: Gate Vertices

Input:

- A grid graph G,
- a set of reservations $\mathcal{R} = \{R_1, \dots, R_k\}$ and
- a target $t \in V(G)$.

Output:

- A set of distinguished vertices on the reservations $P \subset V(G) \cap \bigcup_{i=1}^{k} R_i$ and
- a map $r : P \to \mathcal{P}(\mathcal{R})$ assigning each vertex to the reservations it lies on.

Initialize $P := \emptyset$.
for $i = 1, \dots, k$ **do**
  $p :=$ Line-Projection(t, R_i).
  if $p \in R_i$ **then**
    if $p \notin P$ **then**
      Set $P := P \cup \{p\}$ and $r(p) := \{R_i\}$.
    else
      Set $r(p) := r(p) \cup \{R_i\}$.
  for $j = 1, \dots, k$ **do**
    Set $p_1 :=$ Line-Projection $\Big((x^j_{\min}, y^j_{\min}, z^j), R_i\Big)$.
    if $p_1 \in R_i$ **then**
      if $p_1 \notin P$ **then**
        Set $P := P \cup \{p_1\}$ and $r(p_1) := \{R_i\}$.
      else
        Set $r(p_1) := r(p_1) \cup \{R_i\}$.
    Set $p_2 :=$ Line-Projection $\Big((x^j_{\max}, y^j_{\max}, z^j), R_i\Big)$.
    if $p_2 \in R_i$ **then**
      if $p_1 \notin P$ **then**
        Set $P := P \cup \{p_2\}$ and $r(p_2) := \{R_i\}$.
      else
        Set $r(p_2) := r(p_2) \cup \{R_i\}$.

– $u \in R$, $v \notin R$ but $u \notin P$ and $(w, u) \in E(S)$ for some vertex $w \in R$ or

– $u \notin R$ and $v \in R$ but $v \notin P$ and $(v, w) \in E(S)$ for some vertex $w \in R$.

This means that the edge e uses the reservation R and enters or leaves it through a non-gate vertex.

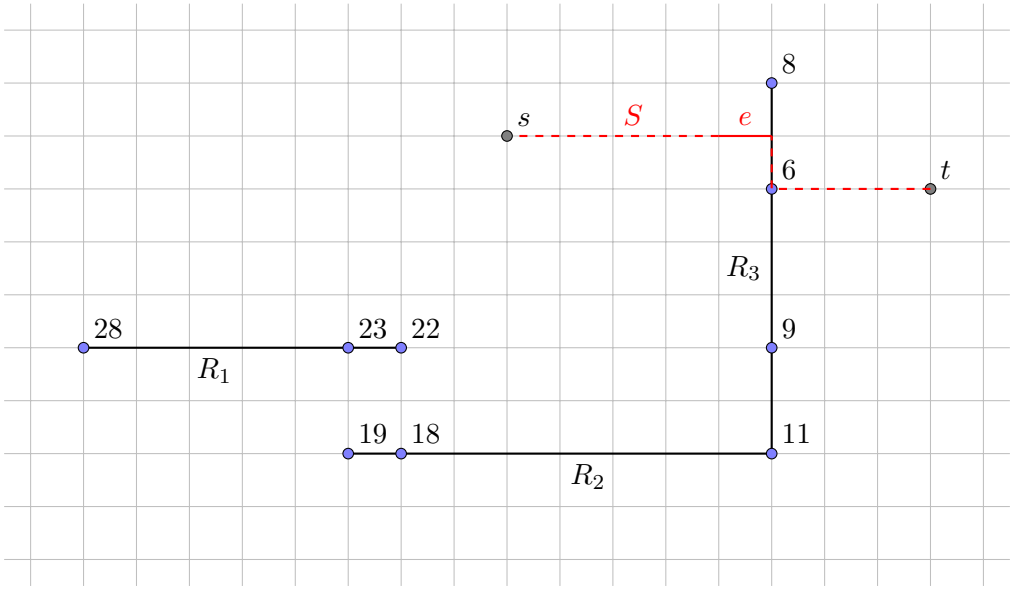
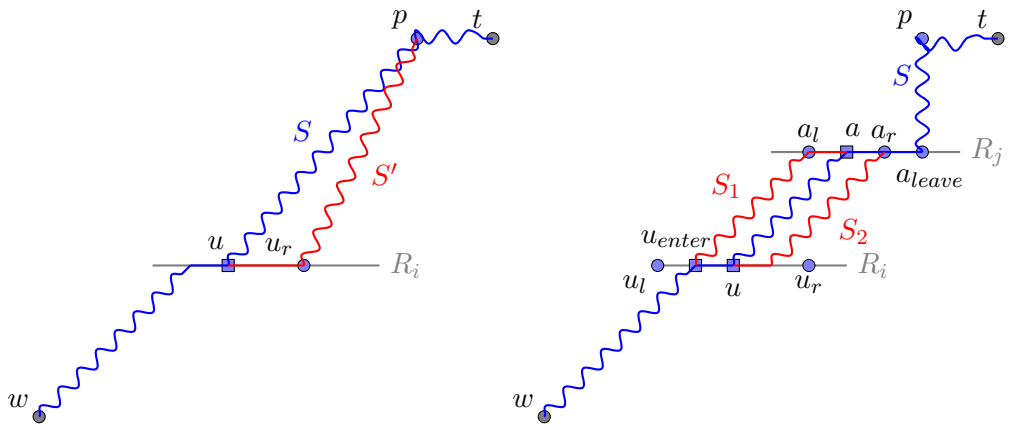


Figure 2: A grid graph with reservations $\mathcal{R} = \{R_1, R_2, R_3\}$ and a target vertex t. Blue vertices denote the gate vertices as computed by Algorithm 2. Assume that each edge on a reservations costs 1 unit and off the reservations 2 units. The numbers on the gate vertices give their respective shortest distance to t as computed by Algorithm 3. The s-t-path S contains the gate-violating edge e.

Clearly, Algorithm 2 has a run time of $\mathcal{O}(|\mathcal{R}|^2)$ and the set of returned gate vertices P is also of size at most $\mathcal{O}(|\mathcal{R}|^2)$. Figure 2 depicts the output of Algorithm 2 for one set of reservations. One advantage of introducing gate vertices lies in the following lemma.

Lemma 3.2. *Let $w \in V(G)$ be any vertex with $w \in R_l$ for some reservation $R_l \in \mathcal{R}$. Then, there exists a shortest w-t-path S in G with respect to $c^{\mathcal{R}}$ not containing any gate-violating edge.*

Proof. On any w-t-path S, we call a gate-violating edge $e = (u, v) \in E(G)$ with $u \in R$ for some $R \in \mathcal{R}$ *leaving*. Analogously, e is called *entering*, if $v \in R$ for some $R \in \mathcal{R}$. Note that an edge e can be leaving and entering simultaneously, if both its endpoints lie on two different reservations.



(a) Case 1.1 (b) Case 1.2

Figure 3: Situation as in Case 1 of the proof of Lemma 3.2. Blue circles denote gate vertices and blue squares non-gate vertices respectively.

Let S be a shortest w-t-path with respect to $c^{\mathcal{R}}$ containing a minimal number of gate-violating edges. Assume this number to be non-zero, otherwise there is nothing to prove. Then, we distinguish the following cases:

Case 1: S contains a leaving gate-violating edge. This situation is depicted in Figure 3. Let $e = (u, v) \in E(G)$ be the last leaving gate-violating edge on S with $u \in R_i$ for some $R_i \in \mathcal{R}$. W. l. o. g. assume that R_i is a reservation in x-direction. Let $p \in V(G)$ be the first gate vertex on the path segment $S_{[u,t]}$, or the target vertex if $S_{[u,t]}$ does not contain any gate vertex. We have to again distinguish the following two cases:

Case 1.1: $S_{[u,p]}$ does not contain any edge on a reservation. Since e is gate-violating, u is not a gate vertex. On the other hand, p is a gate vertex or the target which implies $u_x \neq p_x$ by construction of the gate vertices in Algorithm 2. W. l. o. g. we may assume $u_x < p_x$. Let u_r be the first gate vertex on R_i right of u. By construction of the gate vertices, we then again have $u_{r_x} \leq p_x$. By removing all edges $(f, g) \in E(S)$ of the path segment $S_{[u,p]}$ in x-direction with $u_x \leq f_x \leq u_{r_x}$ and $u_x \leq g_x \leq u_{r_x}$ and concatenating all remaining segments, we obtain a u_r-p-path S'', since we have $u_{r_x} \leq p_x$. Let S' be the w-t-path S where we replace the path segment $S_{[u,p]}$ with the concatenation of the u-u_r-path on the

reservation R_i and S''. Furthermore

$$c^{\mathcal{R}}(E(S')) \leq c^{\mathcal{R}}(E(S))$$

holds, since the costs on non-reserved edges are constant under translation in x-direction (or can at most decrease, if the new path contains reserved edges), the path segment $S_{[u,p]}$ contains no reserved edges and the cost of the u-u_r-path on R_i is smaller or equal to the costs of the deleted edges by assumption (2) in Section 2.3.

Case 1.2: $S_{[u,p]}$ contains an edge on a reservation. Let $a \in R_j$ for some $R_j \in \mathcal{R}$ be the first vertex on the reservation R_j that the path segment $S_{[u,p]}$ uses. We have $a \neq p$, since $S_{[u,p]}$ contains an edge on a reservation. Let $a_l, a_r \in R_j$ be the first gate vertex left of a on R_j and the first gate vertex right of a respectively and analogously define $u_l, u_r \in R_i$ to be the gate vertices left and right of u on R_i.

Since e is a gate-violating edge, S uses the reservation R_i. Let $u_{enter} \in R_i$ be the first vertex on R_i the path segment $S_{[w,u]}$ passes. Similarly, we know by the choice of Case 1.2 that S uses the reservation R_j. So let $a_{leave} \in R_j$ be the last vertex on R_j the path S uses after entering through a, this could also be the target t, if S does not leave R_j. If $u_{enter_x} < u_x$ and $a_{leave_x} < a_x$, we could shorten S by shifting the path segment $S_{[u,a]}$ to the left and deleting the appropriate edges on the reservations R_i and R_j. Analogously, the case $u_{enter_x} > u_x$ and $a_{leave_x} > a_x$ leads to a contradiction. So we can w. l. o. g. assume $u_{enter_x} < u_x$ and $a_{leave_x} > a_x$. Since the path segment $S_{[a,p]}$ can not contain any gate-violating edge by the assumption of Case 1, we know that a_{leave} must be a gate vertex or the target. The vertex a_r is the first gate vertex on R_j right of a. Thus, we must have $a_{r_x} \leq a_{leave_x}$.

Hence, we can shift the path segment $S_{[u,a]}$ by $\min(a_{r_x} - a_x, u_{r_x} - u_x) > 0$ units to the right or by $\min(a_x - a_{l_x}, u_x - u_{l_x}, u_x - u_{enter_x})$ units to the left in x-direction and obtain two new w-t-paths S_1 and S_2 after adding and removing the appropriate edges on the reservations R_i and R_j respectively. We define the cheaper one of these two new paths (with respect to $c^{\mathcal{R}}$) to be the new w-t-path S'. Since the number of added edges on the reservations equals the removed ones on the other reservation and the cost of the translated segment $S_{[u,a]}$ can at most

decrease as in Case 1.1, we again obtain by the choice of S'

$$c^{\mathcal{R}}(E(S')) \leq c^{\mathcal{R}}(E(S)).$$

In fact, we must then have $c^{\mathcal{R}}(E(S')) = c^{\mathcal{R}}(E(S))$, because S was chosen to be a shortest w-t-path with respect to $c^{\mathcal{R}}$. By the choice of the degree of the left or right shift, the new path S' contains at least one gate-violating edge less.

Case 2: S contains no leaving gate-violating edge. Let $e = (u, v) \in E(G)$ be the first entering gate-violating edge on S with $v \in R_i$ for some $R_i \in \mathcal{R}$. By assumption, $w \in R_l$ holds. Since e is an entering gate-violating edge, the path S must leave the reservation R_l before entering R_i. Hence, $R_l \neq R_i$. The path segment $S_{[w,v]}$ does not contain any leaving gate-violating edge by the choice of Case 2. Thus, S leaves R_l via a gate vertex, which implies that the path segment $S_{[w,u]}$ contains a gate vertex. Let $p \in V(G)$ be the last gate vertex on the path segment $S_{[w,v]}$ before v. Thus, $S_{[p,v]}$ contains no edge on a reservation by the choice of p. Since u is not a gate vertex, $u_x \neq p_x$ holds. Hence, w. l. o. g. we may again assume $u_x < p_x$. In conclusion analogously as in Case 1.1, we can again shift parts of the path segment $S_{[p,v]}$ to the right, to obtain a new shortest w-t-path S' containing at least one gate-violating edge less.

In all cases we could reduce the number of gate-violating edges on the new shortest w-t-path S', which contradicts our initial assumption. Hence, the minimal number of gate-violating edges on a shortest w-t-path with respect to $c^{\mathcal{R}}$ is in fact zero, which proves our claim. □

Remark 3.3. Lemma 3.2 does not imply that one can remove all gate-violating edges from the routing graph G_T during the path search algorithm, since its statement is only valid in the grid graph G. In G_T some gate vertices could not be accessible due to blockages or previously routed nets which means that a shortest path might have to use also gate-violating edges.

In the following we will describe the second part of our preprocessing algorithm. We call this component a meta Dijkstra algorithm since we will execute Dijkstra's algorithm on a subgraph $\tilde{G}$ of the complete graph on the gate vertices and the target. Figure 4 depicts an example of the graph $\tilde{G}$.

Algorithm 3: SPTTWR-Preprocessing: Meta Dijkstra's algorithm

Input:

- A grid graph G,
- a set of reservations $\mathcal{R} = \{R_1, \ldots, R_k\}$,
- a target $t \in V(G)$,
- a set of gate vertices P with a map $r : P \to \mathcal{P}(\mathcal{R})$.

Output: $d_P : P \to \mathbb{R}_{\geq 0}$, where $d_p(p) = \text{dist}_{(G,c^{\mathcal{R}})}(p, t)$ for each $p \in P$.

1 Define the graph $\tilde{G}$ by setting $V(\tilde{G}) := P \cup \{t\}$ and $E(\tilde{G}) := E_1 \cup \ldots \cup E_5$ where

$$E_1 := \{\{t, v\} \mid v \in P\},$$
$$E_2 := \{\{u, v\} \mid u, v \text{ gate vertices on the same reservation such that no other gate vertex lies between } u \text{ and } v\},$$
$$E_3 := \{\{u, v\} \mid u, v \text{ OGVs of different reservations}\},$$
$$E_4 := \{\{u, v\} \mid u \text{ IGV, } v \text{ OGV and } v \text{ is a creator of } u\},$$
$$E_5 := \{\{u, v\} \mid u, v \text{ IGVs of different reservations having the same set of creators,}$$

2 i. e. u, v share an x- or y-coordinate$\}$.

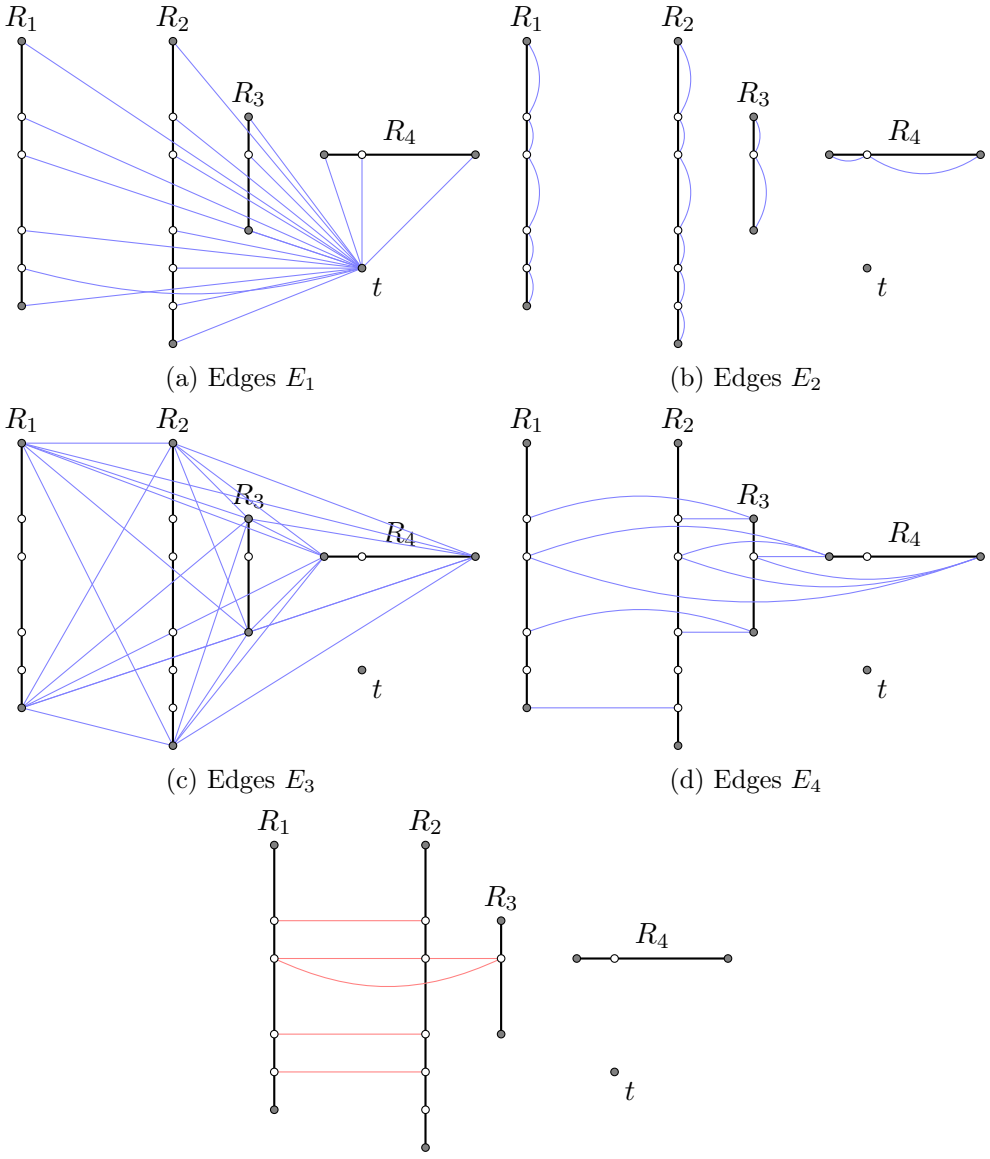
3 Further define the cost function $\tilde{c} : E(\tilde{G}) \to \mathbb{R}_{\geq 0}$ by setting

$$\tilde{c}(\{u, v\}) := \begin{cases} c^{\mathcal{R}}\left(E(P_{uv})\right) & \text{if } u, v \in P \text{ and } r(u) \cap r(v) \neq \emptyset, \\ \text{dist}_{(G,c)}(u, v) & \text{otherwise,} \end{cases}$$

for $\{u, v\} \in E(\tilde{G})$ where P_{uv} is the direct u-v-path in G on the reservation u and v both lie on.

4 Apply Dijkstra's algorithm to the graph $(\tilde{G}, \tilde{c})$ with source t and without target to compute a complete shortest-path-tree with source t.

5 For any $p \in P$ set $d_P(p) := \text{dist}_{(\tilde{G},\tilde{c})}(p, t)$.



(a) Edges E_1

(b) Edges E_2

(c) Edges E_3

(d) Edges E_4

(e) Edges E_5

Figure 4: The graph $\tilde{G}$ for the reservations $\mathcal{R} = \{R_1, R_2, R_3, R_4\}$ and the target t. OGVs are denoted by filled circles. IGVs are drawn as empty circles.

Theorem 3.4. *Algorithm 3 computes* $\mathrm{dist}_{(G,c^{\mathcal{R}})}(p,t)$ *for all* $p \in P$ *correctly. If we assume that* $\mathrm{dist}_{(G,c)}(u,v)$ *can be computed in constant time for any* $u, v \in V(G)$*, it runs in* $\mathcal{O}(|\mathcal{R}|^3)$*-time.*

Proof. Let K be the undirected complete graph on the vertices $P \cup \{t\}$. Hence, $\tilde{G}$ is a subgraph of K having the same vertices. We extend $\tilde{c}$ to K by setting analogously

$$\tilde{c}(\{u,v\}) := \begin{cases} c^{\mathcal{R}}\left(E(P_{uv})\right) & \text{if } u,v \in P \text{ and } r(u) \cap r(v) \neq \emptyset, \\ \mathrm{dist}_{(G,c)}(u,v) & \text{otherwise,} \end{cases}$$

for $\{u,v\} \in E(K)$ where P_{uv} is the direct u-v-path in G on the reservation u and v both lie on. We will firstly show that we can compute $\mathrm{dist}_{(G,c^{\mathcal{R}})}(p,t)$ by performing Dijkstra's algorithm on the graph K with costs $\tilde{c}$ and target t. Secondly, we will prove that in fact it suffices to invoke Dijkstra's algorithm on the subgraph $\tilde{G}$ to compute $\mathrm{dist}_{(G,c^{\mathcal{R}})}(p,t)$ correctly.

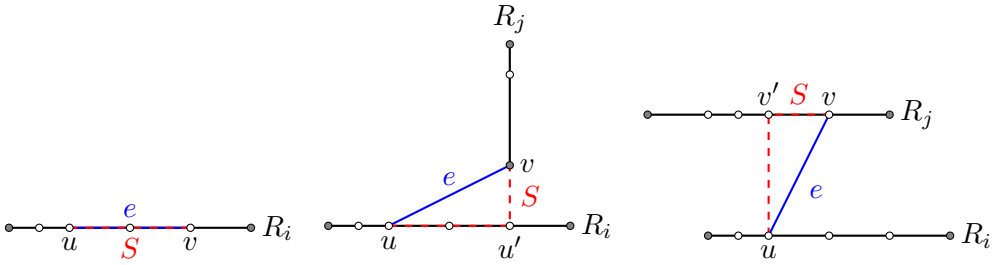
When computing $\mathrm{dist}_{(G,c^{\mathcal{R}})}(p,t)$ for some $p \in P$ we can delete all gate-violating edges of G, since we can still find a shortest p-t-path with respect to $c^{\mathcal{R}}$ among the remaining edges by Lemma 3.2. Any such shortest p-t-path S with respect to $c^{\mathcal{R}}$ and without gate-violating edges gives rise to a p-t-path $\tilde{S}$ in K by replacing each path segment between gate vertices with the corresponding undirected edge in K. By definition of $\tilde{c}$, we then have

$$\tilde{c}(E(\tilde{S})) = c^{\mathcal{R}}(E(S)).$$

Conversely, a p-t-path $\tilde{S}$ in K can be unpacked to a p-t-path S in G without gate-violating edges by replacing each edge in $\tilde{S}$ with a shortest directed path segment in G with respect to $c^{\mathcal{R}}$. Also in this case, $\tilde{c}(E(\tilde{S})) = c^{\mathcal{R}}(E(S))$ holds by definition of $\tilde{c}$. Hence, the length of a shortest p-t-path in G with respect to $c^{\mathcal{R}}$ equals the length of a shortest p-t-path in K with respect to $\tilde{c}$.

By definition of $E(\tilde{G})$, we have $E(K) = E(\tilde{G}) \cup (D_1 \cup D_2 \cup D_3)$ where

$$D_1 := \{\{u,v\} \mid u,v \text{ non-neigboring gate verices on the same reservation}\},$$
$$D_2 := \{\{u,v\} \mid u \text{ an IGV and } v \text{ an OGV such that } v \text{ is not a creator of } u\},$$
$$D_3 := \{\{u,v\} \mid u,v \text{ IGVs of different reservations having different creators}\}.$$



(a) Case 1 (b) Case 2 (c) Case 3.2

Figure 5: Situation as in the proof of Theorem 3.4. Black circles denote gate vertices (the filled ones are the OGVs and the empty ones the IGVs).

Now, we show that any edge $e = \{u, v\} \in E(K) \setminus E(\tilde{G})$ is dominated by a u-v-path S in $\tilde{G}$, i. e. $\tilde{c}(E(S)) \leq \tilde{c}(e)$. Let $e = \{u, v\} \in E(K) \setminus E(\tilde{G})$. In any case, u is a gate vertex, say on reservation R_i. We distinguish the following three cases. Figure 5 depicts the situation of these cases.

Case 1: $e \in D_1$**.** The edge e is dominated by the path of consecutive gate vertices from u to v on the same reservation. By definition of $\tilde{c}$, this path has the same cost as the edge e.

Case 2: $e \in D_2$**.** We have $u \in R_i$ and let u' be the closest point on R_i to v with respect to the L_1 distance. Since v is not a creator of u and u is an IGV, we must have $u \neq u'$. By definition, u' is also a gate vertex. Either u' is an OGV or an IGV created by v. In both cases, we have $\{u', v\} \in E(\tilde{G})$. Let S be the u-v-path in $\tilde{G}$ consisting of the consecutive edges from u to u' on R_i and the edge $\{u', v\}$. By assumption (2) in Section 2.3, we then have $\tilde{c}(E(S)) \leq \tilde{c}(e)$.

Case 3: $e \in D_3$**.** Let v be on the reservation R_j and define v' to be the closest vertex on R_j to u with respect to the L_1 distance. Since u and v have different creators we have $v \neq v'$ and v' is a gate vertex.

Case 3.1: v' **is an OGV.** As in Case 2, we can find a u-v'-path in $E(\tilde{G})$ dominating the edge $\{u, v'\}$ in K. Extending it with the consecutive edges of gate vertices on R_j from v' to v yields a u-v-path in $\tilde{G}$ dominating e.

Case 3.2: v' **is an IGV.** In this case, the reservations R_i and R_j must have the same direction and v' has the same creators as u. Hence, $\{u, v'\} \in E_5 \subset E(\tilde{G})$ holds. Concatenating the edge $\{u, v'\}$ with the

consecutive edges from v' to v on R_j yields a u-v-path in $\tilde{G}$ dominating e again by assumption (2) in Section 2.3.

Thus, Dijkstra's algorithm on the graph $\tilde{G}$ in Line 4 of Algorithm 3 computes $\text{dist}_{(G,c^{\mathcal{R}})}(p,t)$ for any $p \in P$ correctly.

The run time is dominated by the execution of Dijkstra's algorithm in Line 4. Using Fibonacci heaps, Fredman and Tarjan showed in [FT87] that Dijkstra's algorithm can be implemented to run in $\mathcal{O}(m + n \log n)$-time where $m = |E(G)|$ and $n = |V(G)|$ for a graph G. In total, there are at most $\mathcal{O}(|\mathcal{R}|^2)$ gate vertices and $\mathcal{O}(|\mathcal{R}|)$ OGVs. Therefore, the sets $E_1, \ldots, E_4$ are of size at most $\mathcal{O}(|\mathcal{R}|^2)$ and E_5 is of size at most $\mathcal{O}(|\mathcal{R}|^3)$. Hence, we perform Dijkstra's algorithm on a graph with $|P| + 1$ vertices and $\mathcal{O}(|\mathcal{R}|^3)$ edges, which yields the run time $\mathcal{O}(|\mathcal{R}|^3)$. □

Algorithm 3 has run time $\mathcal{O}(|\mathcal{R}|^3)$ because the set of edges E_5 is of size $\mathcal{O}(|\mathcal{R}|^3)$. Now, we describe an approach where we replace $\tilde{G}$ in Line 3 by a graph $\tilde{H}$ whose sets of vertices and edges are both of size $\mathcal{O}(|\mathcal{R}|^2)$ respectively. This yields the improved run time $\mathcal{O}\left(|\mathcal{R}|^2 \log(|\mathcal{R}|)\right)$ but keeps the correctness of the algorithm. To this end, we define $P = I \cup O$ where I and O are the set of inner and outer gate vertices respectively. We define T to be the set of IGVs having the target t as one of their creators. The remaining IGVs in $I \setminus T$ can be partitioned by sets I_i for $i = 1, \ldots, n$ for some $n \in \mathbb{N}$ where the IGVs in such a set I_i have exactly the same OGVs (and not the target t) as creators. This defines a partition of I by

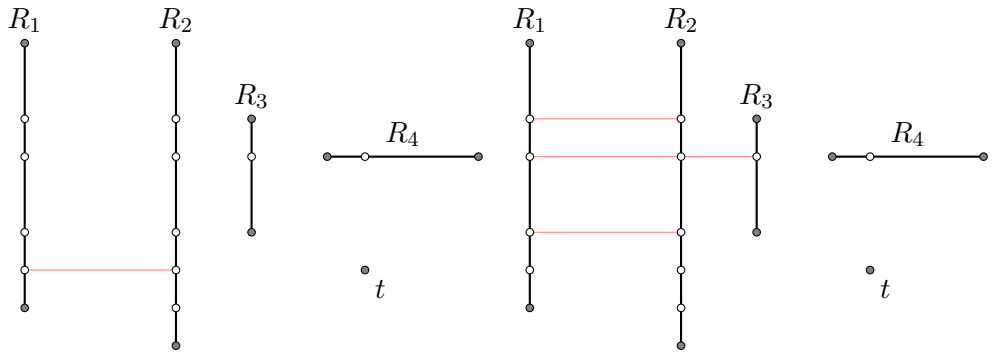
$$I = T \cup \bigcup_{i=1}^{n} I_i.$$

Hence, for $i = 1, \ldots, n$ we must have $u_x = v_x$ for all $u, v \in I_i$ or $u_y = v_y$ for all $u, v \in I_i$. In the first case, we call I_i an x-set of IGVs and an y-set of IGVs otherwise. Thus, we can define a partial order $\preceq$ on each set I_i for $i = 1, \ldots, n$ by defining for $u, v \in I_i$

$$u \prec v :\Leftrightarrow \begin{cases} u_x < v_x, \text{ if } I_i \text{ is a } y\text{-set of IGVs,} \\ u_y < v_y, \text{ if } I_i \text{ is a } x\text{-set of IGVs.} \end{cases}$$

Now we can define the graph $\tilde{H}$ by setting

$$V(\tilde{H}) := \{(u_x, u_y, z) \mid u \in P \cup \{t\}, z^G_{\min} \leq z \leq z^G_{\max}\},$$
$$E(\tilde{H}) := \tilde{E}_1 \cup \ldots \cup \tilde{E}_7,$$



(a) Edges $\tilde{E}_5$ (b) Edges $\tilde{E}_7$

Figure 6: The graph $\tilde{H}$ for the reservations $\mathcal{R} = \{R_1, R_2, R_3, R_4\}$ and the target t. The circles on the reservations are the gate vertices (the filled ones are the OGVs and the empty ones the IGVs). Only the edges in $\tilde{E}_5$ and $\tilde{E}_7$ are drawn. The edges in $\tilde{E}_1, \dots, \tilde{E}_4$ are the corresponding ones drawn in Figure 4. Additionally, all vertices and the edges in $\tilde{E}_7$ are copied for each layer in G. The edges $\tilde{E}_6$ connect any two vertices lying directly above each other after copying the vertices.

where we define

$$\begin{aligned}
\tilde{E}_i &:= \{\{u, v\} && \mid \{u, v\} \in E_i\} \text{ for } i = 1, \dots, 4,\\
\tilde{E}_5 &:= \{\{u, v\} && \mid \{u, v\} \in E_5 \text{ and } u, v \text{ are both created by } t\},\\
\tilde{E}_6 &:= \{\{(u_x, u_y, z), (u_x, u_y, z+1)\} && \mid u \in I_i \text{ for some } 1 \le i \le n \text{ and } z^G_{\min} \le z \le z^G_{\max} - 1\},\\
\tilde{E}_7 &:= \{\{(u_x, u_y, z), (v_x, v_y, z)\} && \mid u, v \in I_i \text{ for some } 1 \le i \le n \text{ with } u \prec v,\\
&&& \quad \text{there exists no } w \in I_i \text{ with } u \prec w \prec v \text{ and}\\
&&& \quad z^G_{\min} \le z \le z^G_{\max}\}.
\end{aligned}$$

We also extend the cost function $\tilde{c}$ to $E(\tilde{H})$ by setting

$$\tilde{c}(\{(u_x, u_y, z_1), (v_x, v_y, z_2)\}) := \begin{cases} c^{\mathcal{R}}\left(E(P_{uv})\right) & \text{if } u_z = z_1, v_z = z_2 \text{ and}\\ & r(u) \cap r(v) \neq \emptyset,\\ \operatorname{dist}_{(G,c)}((u_x, u_y, z_1), (v_x, v_y, z_2)) & \text{otherwise.} \end{cases}$$

for $u, v \in P \cup \{t\}$ with $\{(u_x, u_y, z_1), (v_x, v_y, z_2)\} \in E(\tilde{H})$ for some $z^G_{\min} \le z_1, z_2 \le z^G_{\max}$ where P_{uv} is the direct u-v-path in G on the reservation u and v both lie on.

Figure 6 depicts the graph $\tilde{H}$ for the example graph $\tilde{G}$ of Figure 4.

Theorem 3.5. *Replacing the graph $\tilde{G}$ by $\tilde{H}$ in Line 3 of Algorithm 3 we can keep its correctness and obtain the improved run time $\mathcal{O}(|\mathcal{R}|^2 \log |\mathcal{R}|)$ under the assumption that $z^G_{max} - z^G_{min}$ is a constant.*

Proof. The edges of $\tilde{G}$ and $\tilde{H}$ and their cost with respect to $\tilde{c}$ agree except for E_5 and $\tilde{E}_5, \tilde{E}_6, \tilde{E}_7$. The edges in E_5 between IGVs created by the target t are precisely the ones appearing in $\tilde{E}_5$. So consider $e = \{u, v\} \in E_5$ an edge between IGVs created by the same set of OGVs. By definition of the partition of I, we then have $u, v \in I_i$ for some $1 \leq i \leq n$ and we can w. l. o. g. assume that $u_x = v_x$ holds, i. e. I_i is an x-set of IGVs. The IGVs u and v lie on different reservations because they have the same set of creators. Hence, the cost $\tilde{c}(\{u, v\})$ is the length of a shortest u-v-path in G with respect to c. Henke showed in [Hen16, Lemma 2.4] that there exists a shortest u-v-path consisting of the following three path segments for some $z^G_{\min} \leq z_0 \leq z^G_{\max}$:

1. the straight path in z-direction from u to (u_x, u_y, z_0),
2. the straight path in y-direction from (u_x, u_y, z_0) to (v_x, v_y, z_0) (we have $u_x = v_x$),
3. the straight path in z-direction from (v_x, v_y, z_0) to v.

Thus, the edge e corresponds to a path in $\tilde{H}$ where the first and the third segment consist of edges in $\tilde{E}_6$ and the second of an ordered chain of edges in $\tilde{E}_7$. The cost of this path and the edge e agree by definition of $\tilde{c}$. Conversely, any shortest path between such IGVs of the same OGVs as creators corresponds to a direct edge in $\tilde{G}$ of the same cost. So in total, Dijkstra's algorithm on the graph $\tilde{H}$ also computes $\text{dist}_{(G,c^{\mathcal{R}})}(u, t)$ for $u \in P$ correctly.

For the run time note that by our assumption of $z^G_{\max} - z^G_{\min}$ being a constant, $V(\tilde{H})$ is still of size $\mathcal{O}(|\mathcal{R}|^2)$. The sets $\tilde{E}_1, \ldots, \tilde{E}_5$ are of size $\mathcal{O}(|\mathcal{R}|^2)$ because $E_1, \ldots, E_4$ are of the same size and there are at most $\mathcal{O}(|\mathcal{R}|)$ IGVs created by the target. Due to the assumption of $z^G_{\max} - z^G_{\min}$ being a constant, also $\tilde{E}_6$ is of size $\mathcal{O}(|\mathcal{R}|^2)$.

Let $u \in I_i$ be an IGV created by OGVs only for some $1 \leq i \leq n$ and let $z^G_{\min} \leq z \leq z^G_{\max}$ be any plane. Assume $\{(u_x, u_y, z), (v_x, v_y, z)\} \in \tilde{E}_7$ for some IGV v. Then by definition of $\tilde{E}_7$, v must be the greatest predecessor or the smallest successor of u in I_i with respect to the partial order $\prec$. Due to the fact that there are only $\mathcal{O}(|\mathcal{R}|^2)$ many IGVs, the sets I_i being pairwise disjoint and the assumption of $z^G_{\max} - z^G_{\min}$ being a constant, this also shows that $\tilde{E}_7$ is of size $\mathcal{O}(|\mathcal{R}|^2)$.

Finally, both the set of edges and the set of vertices of the graph $\tilde{H}$ are of size $\mathcal{O}(|\mathcal{R}|^2)$. As cited above, Dijkstra's algorithm can be implemented to run in $\mathcal{O}(m + n \log n)$-time where $m = |E(G)|$ and $n = |V(G)|$ for a graph G. Hence, the improved run time of Algorithm 3 is $\mathcal{O}(|\mathcal{R}|^2 \log |\mathcal{R}|)$. □

3.2 Future Cost Algorithm

Now we give an algorithm to compute $\text{dist}_{(G,c^{\mathcal{R}})}(s,t)$ for any vertex $s \in V(G)$ and the target t which uses the precomputed distances $\text{dist}_{(G,c^{\mathcal{R}})}(p,t)$ for $p \in P$ a gate vertex. This algorithm is iteratively called for computing the future cost function in Algorithm 1. The preprocessing Algorithms 2 and 3 need to be called only once before each path search.

Algorithm 4: Future cost function with reservations.

Input:

- A grid graph G,
- a target $t \in V(G)$ and a vertex $s \in V(G)$,
- a set of reservations $\mathcal{R} = \{R_1, \ldots, R_k\}$ with gate vertices P and distances $d_P : P \to R^+$ as computed in Algorithm 3.

Output: The length of a shortest s-t-path in G with respect to $c^{\mathcal{R}}$.

1 Set $d := \text{dist}_{(G,c)}(s,t)$.
2 **for** $j = 1, \ldots, k$ **do**
3 Set $w :=$ Closest-Point(R_j, s).
4 Let w_l, w_r be the first gate vertices left and right of w on R_j respectively.
5 Set $d_{current} := \text{dist}_{(G,c)}(s,w) +$
  $\min\Big(d_P(w_l) + \text{dist}_{(G,c^{\mathcal{R}})}(w_l, w), d_P(w_r) + \text{dist}_{(G,c^{\mathcal{R}})}(w, w_r)\Big)$.
6 **if** $d_{current} < d$ **then**
7 Set $d := d_{current}$.
8 **return** d.

Theorem 3.6. *Algorithm 4 computes* $\text{dist}_{(G,c^{\mathcal{R}})}(s,t)$ *correctly. If we again assume that* $\text{dist}_{(G,c)}(u,v)$ *can be computed in constant time for any* $u, v \in V(G)$*, it runs in* $\mathcal{O}(|\mathcal{R}| \log |\mathcal{R}|)$*-time.*

Proof. If there exists a shortest s-t-path with respect to $c^{\mathcal{R}}$ which does not use any reservation, we have $\text{dist}_{(G,c)}(s,t) = \text{dist}_{(G,c^{\mathcal{R}})}(s,t)$. Since we also minimize over $\text{dist}_{(G,c)}(s,t)$ in Line 1, the result of Algorithm 4 is correct in this case.

So assume that every shortest s-t-path with respect to $c^{\mathcal{R}}$ uses at least one reservation. Let S be such a shortest s-t-path with respect to $c^{\mathcal{R}}$ and let R_i be the first reservation it uses after s. By assumption (2) the costs on the reservation R_i are less or equal to the regular costs off the reservations in that direction. Hence, if we set $w :=$ Closest-Point(R_i, s) as in Line 3, we have

$$\text{dist}_{(G,c^{\mathcal{R}})}(s,t) = \text{dist}_{(G,c)}(s,w) + \text{dist}_{(G,c^{\mathcal{R}})}(w,t)$$

since $\text{dist}_{(G,c)}(s,w) = \text{dist}_{(G,c^{\mathcal{R}})}(s,w)$ holds by the assumption that S is a shortest path using the reservation R_i first and $w \in R_i$. By Lemma 3.2 there exists a a shortest w-t-path with respect to $c^{\mathcal{R}}$ that does not contain any gate-violating edge. Thus, we can compute $\text{dist}_{(G,c^{\mathcal{R}})}(w,t)$ by computing the length of a shortest w-t-path via the first gate vertex left or right of w on R_i. In total, this shows that $d_{current}$ in Line 5 for $j = i$ computes the length of S with respect to $c^{\mathcal{R}}$ correctly.

For the run time, note that any reservation contains at most $\mathcal{O}(|\mathcal{R}|)$ gate vertices. Thus, Line 4 can be implemented in $\mathcal{O}(\log(|\mathcal{R}|))$-time using a binary search. Since the for loop in Line 2 iterates over all reservations, the total run time clearly amounts to $\mathcal{O}(|\mathcal{R}| \log |\mathcal{R}|)$. □

Definition 3.7. For the path search in the routing graph G_T with target $t \in V(G_T)$ and cost $c^{\mathcal{R}}$ we define the future cost function $l_R : V(G_T) \to \mathbb{R}_{\geq 0}$ by setting for a vertex $v \in V(G_T)$

$$l_R(v) := \text{dist}_{(G,c^{\mathcal{R}})}(v,t).$$

This defines a future cost function by the Lemmata 2.6 and 2.7. It can efficiently be computed using Algorithm 4.

4 Multilabel Future Cost Function with Reservations

4.1 Construction of Multilabel Framework

Firstly, we give another definition to simplify our notation in the following.

Definition 4.1. An s-t-path S in G or G_T is called $\mathcal{R}$-*reducible*, if it enters a reservation it has previously left without using another reservation in between. This means for some $R \in \mathcal{R}$ there exists two edges $e = (u, v) \in E(S)$ and $e' = (u', v') \in E(S)$ with $u \neq v'$ such that $u, v' \in R$ and $v, u' \notin R$ and the path segment $S_{[v,u']}$ does not use any reservation in $\mathcal{R}$. Otherwise, S is called $\mathcal{R}$-*irreducible*.

This definition is motivated by the following proposition, stating that shortest paths are $\mathcal{R}$-irreducible.

Proposition 4.2. *Let S be a shortest s-t-path with respect to $c^{\mathcal{R}}$ in G or G_T for $s, t \in V(G_T)$. Then, S is $\mathcal{R}$-irreducible.*

Proof. Assume for a contradiction that S is $\mathcal{R}$-reducible, i. e. for some $R \in \mathcal{R}$ there exists two edges $e = (u, v), e' = (u', v') \in E(S)$ in this order with $u \neq v'$ such that $u, v' \in R$ and $v, u' \notin R$. Further, we may assume by the definition of $\mathcal{R}$-irreducible that the path segment $S_{[v,u']}$ does not use any reservation in $\mathcal{R}$. Let S' be the direct u-v'-path on R. The path S' also lies in G_T due to the assumption $R \subset V(G_T)$ made in Section 2.3. The segment S' costs strictly less than the path segment $S_{[u,v']}$ due to the facts that

- $S_{[u,v']}$ contains at least one edge, e. g. the edge e, in a direction orthogonal to R, whereas S' contains no such edge, and
- $S_{[v,u']}$ does not use any reservation, which means that by assumption (2) in Section 2.3 the edges on this path segment parallel to R costs at least as much as the edges on R used by S'.

Hence, replacing the path segment $S_{[u,u']}$ in S by S' yields a shorter s-t-path with respect to $c^{\mathcal{R}}$ which contradicts the assumption on S being a shortest s-t-path. □

We will make use of this fact to speed up the path search algorithm. To this end, assume we are computing a shortest s-t-path S with respect to $c^{\mathcal{R}}$ and inspecting a vertex $v \in V(G_T)$ in Line 3 of Algorithm 1 to decide whether v is permanently labeled next. Assume that the s-v-path considered in Dijkstra's algorithm has most

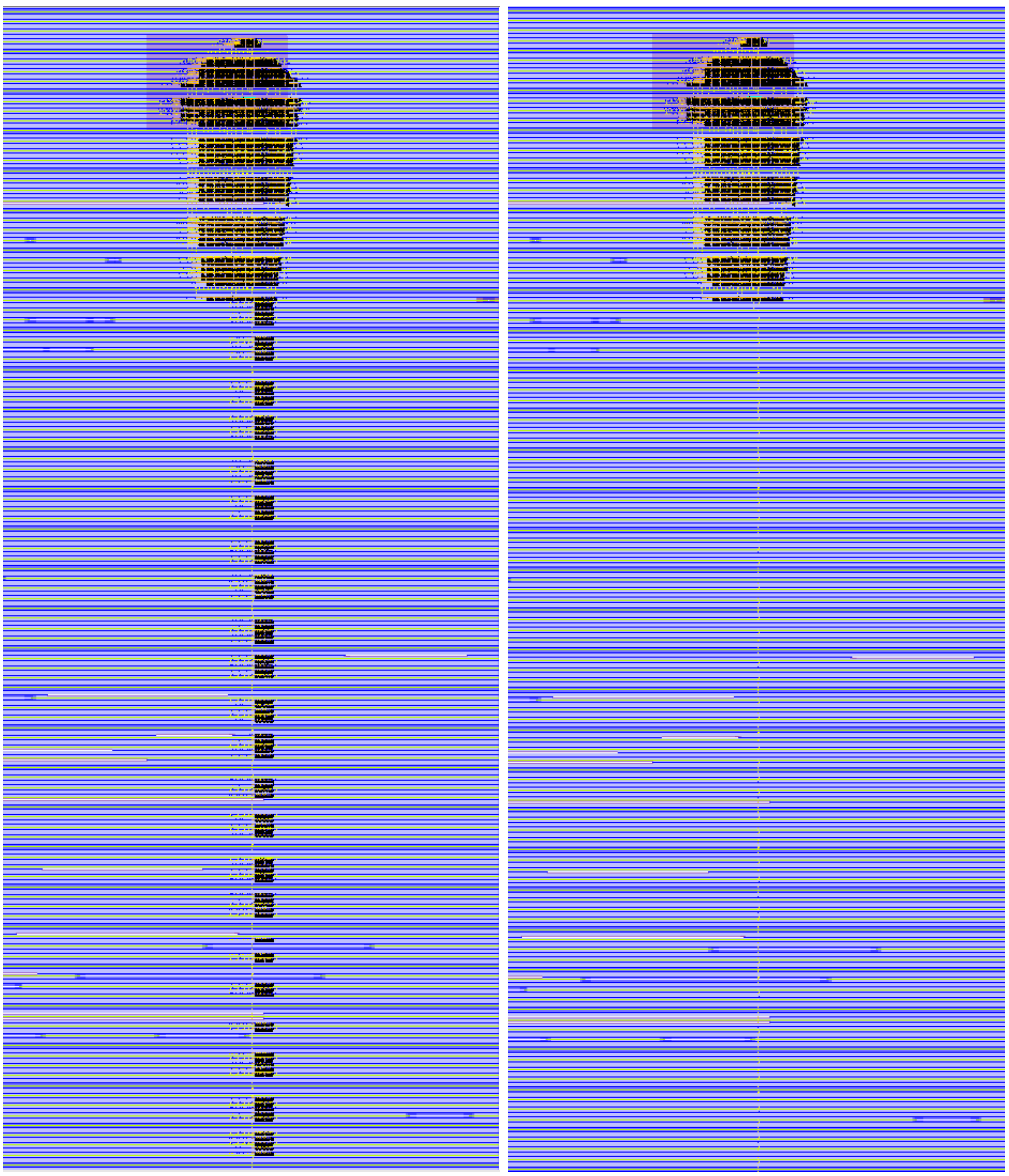
recently left the reservation $R \in \mathcal{R}$. By Proposition 4.2 the remaining path segment $S_{[v,t]}$ must not use the reservation R as next reservation again. Hence, we could compute the future cost function of $l(v)$ as the length of a shortest v-t-path in G with respect to $c^{\mathcal{R}\setminus\{R\}}$, i. e. not discounting the cost of edges on the reservation R.

This approach will likely speed up the path search because previously many vertices next to reservations were labeled. For such a vertex v its future cost function $l(v)$ is computed using the reservation next to v even though the reservation must not be used next for a shortest path if it has been left most recently. For longer reservations, computing $l(v)$ to be the length of a shortest v-t-path with respect to $c^{\mathcal{R}}$ where we do not discount the reservation the path has left previously will significantly increase its future cost function. Hence, this new approach will ensure that many vertices next to reservations will not be permanently labeled anymore. An example of this phenomena can be found in Figure 7.

As explained in Section 2.2, the future cost function used during the path search algorithm is a map $l : V(G_T) \to \mathbb{R}_{\geq 0}$. Hence, it can not depend on the path up to a vertex v. On the other hand, this information is necessary to avoid discounting the cost of the reservation the path has left most recently in the computation of the future cost.

To overcome this limitation, we will use a multilabel framework as introduced by Ahrens et al. in [AGK+15]. The key idea of the multilabel framework is to introduce multiple label types for the vertices of a graph G, e. g. $L = \{1, \ldots, k\}$ for some $k \in \mathbb{N}$, and then construct a new graph G_M having $V(G) \times L$ as its set of vertices. The edges of G_M are constructed according to a specific transition function t. Then the path search is performed in the graph G_M and the result can be related to a path in the original graph G under certain conditions. Using the different label types, one can store information about the history of the path up to a given vertex. Originally, the multilabel framework was introduced to reduce the number of design rule violations. In our context, we will describe a variant of it to use different future cost functions depending on the already used reservations.

We store the information of the reservation which has been left last in the different label types of the multilabel framework. This gives rise to $|\mathcal{R}| + 1$ many different label types; one label type for each reservation and one distinguished label type for the case the path has not yet left any reservation. Furthermore, one needs to compute only two different values of future costs per vertex for all label types as will be shown in Section 4.2.



(a) without multilabel framework (b) with multilabel framework

Figure 7: Part of one layer at the end of the path search using the future cost function with reservations. Permanently labeled vertices are drawn as small black squares. In the layer above the depicted one lies one reservation centrally across the chip from the bottom to the top. The target of the path search lies in a lower layer centrally at the top of the figure.

Now, we will describe the construction of the multilabel framework for our purposes in more detail. Let G_T be the routing graph with reservations $\mathcal{R} = \{R_1, \ldots, R_k\}$. From now on, we assume that the reservations are pairwise disjoint. This assumption is satisfied in our application where the reservations are the assigned wires of the track assignment because we restrict ourselves to only reserving wires in the preferred routing direction on each layer and intersecting reservations of the same direction can be merged to one larger reservation. Now, we define the set of *labels* to be

$$L_M := \{0, \ldots, k\},$$

where the number corresponds to the last left reservation or 0 if none has been left yet. On the reservations themselves, we only use label type 0. To simplify our notation, we introduce a function

$$r : V(G) \to L_M,$$
$$r(v) = \begin{cases} i & \text{if } v \in R_i, \\ 0 & \text{if } v \text{ does not lie on any reservation.} \end{cases}$$

Next, we define a transition function.

Definition 4.3. The *transition function* is a function

$$tr : L_M^3 \to L_M,$$

where $tr(l_v, r(v), r(w))$ gives the next label type for a vertex w if the previous vertex v has label type l_v. For two vertices $v, w \in V(G)$ with $v \neq w$ the values of tr are listed in Table 1.

label type l_v of v	$r(v)$	$r(w)$	$tr(l_v, r(v), r(w))$
0	0	0	0
i	0	0	i
j	j'	i	0
0	i	0	i

Table 1: Definition of the transition function tr where $1 \leq i \leq k$ is an arbitrary reservation index and $j, j' \in L_M$ are arbitrary label types.

By this definition, the following properties hold:

(i) The first two lines of Table 1 ensure that moving between two vertices which are not on reservations keeps the label type constant.

(ii) The third line establishes that a transition to a vertex w lying on a reservation is only possible to label type 0.

(iii) Due to the last line of Table 1, a transition which leaves the reservation R_i to a vertex not lying on any reservation saves this information for the following path search by switching to the label type i.

Now, we can define two new multilabel graphs.

Definition 4.4. The *multilabel graph* G_M is the directed graph defined by

$$V(G_M) := V(G) \times L_M \cup \{(t, \star)\},$$

where $t \in V(G)$ is the designated target vertex and

$$\begin{aligned} E(G_M) := \{((v, l_v), (w, l_w)) \in V(G_M)^2 \mid (v, w) \in E(G),\ & tr(l_v, r(v), r(w)) = l_w \text{ and} \\ & l_v \neq r(w), \text{ if } l_v > 0\} \cup \\ & \{((t, l), (t, \star)) \mid l \in L_M\}. \end{aligned}$$

We can define the same cost function $c^{\mathcal{R}} : E(G_M) \to \mathbb{R}_{\geq 0}$ on G_M by setting

$$c^{\mathcal{R}}((v, l_v), (w, l_w)) := c^{\mathcal{R}}((v, w))$$

for an edge $((v, l_v), (w, l_w)) \in E(G_M)$ not to the designated target $(t, \star)$ and

$$c^{\mathcal{R}}((t, l), (t, \star)) := 0$$

for all remaining edges in $E(G_M)$.

Analogously, we define the *multilabel routing graph* G_{MT} to be the subgraph of G_M consisting of the vertices and edges corresponding to vertices and edges in the routing graph G_T defined in Section 2.2 together with the additional target vertex $((t, \star))$ and the edges of $E(G_M)$ to $((t, \star))$. Lastly, we define the cost function $c^{\mathcal{R}} : E(G_{MT}) \to \mathbb{R}_{\geq 0}$ as the restriction of $c^{\mathcal{R}}$ to $E(G_{GM})$.

The condition $l_v \neq r(w)$ if $l_v > 0$ in the definition of $E(G_M)$ exactly excludes the edges from $E(G_M)$ which would yield $\mathcal{R}$-reducible paths since the label type l_v

indicates that a path has previously used reservation R_{l_v} which would be re-entered if $r(w) = l_v$ would hold.

Proposition 4.5. *Let s be any vertex in G and $l \in L_M$ a fixed label type. Then, there exists a bijection between paths*

$$\{\mathcal{R}\text{-irreducible } s\text{-}t\text{-paths in } G\} \overset{1:1}{\longleftrightarrow} \{(s,l)\text{-}(t,\star)\text{-paths in } G_M\}.$$

This bijection preserves the costs of a path with respect to $c^{\mathcal{R}}$. Furthermore, if we restrict the map to paths in G_T and G_{MT} respectively, it is still a bijection.

Proof. Consider $(e_1, \dots, e_n)$ a sequence of edges of an $\mathcal{R}$-irreducible s-t-path S in G. Let $(v_1, \dots, v_{n+1})$ be the corresponding sequences of vertices ($e_i = (v_i, v_{i+1})$ for each $i = 1, \dots, n$). We recursively define a sequence of label types by setting $l_1 := l$ and $l_{i+1} := tr(l_i, r(v_i), r(v_{i+1}))$ for $i = 1, \dots, n$. Furthermore, for $i = 1, \dots, n$ we set

$$e_{i_M} := ((v_i, l_i), (v_{i+1}, l_{i+1})),$$

and we have $e_{i_M} \in E(G_M)$ since the path S is $\mathcal{R}$-irreducible. Hence,

$$(e_{1_M}, \dots, e_{n_M}, ((v_{n+1}, l_{n+1}), (t, \star)))$$

is a (s,l)-$(t,\star)$-path S_M in G_M with $c^{\mathcal{R}}(E(S)) = c^{\mathcal{R}}(E(S_M))$ by definition of $c^{\mathcal{R}}$.

Conversely, if we have an (s,l)-$(t,\star)$-path S_M in G_M we can remove its last edge to $(t,\star)$ and drop all label types of the vertices of S_M to obtain a s-t-path in G_T. This path is also $\mathcal{R}$-irreducible, since we have removed all edges of G_M which could lead to $\mathcal{R}$-reducible paths in G. Due to the definition of $c^{\mathcal{R}}$ on G_M we then again have

$$c^{\mathcal{R}}(E(S_M)) = c^{\mathcal{R}}(E(S)).$$

These constructions are inverse to each other and hence define a bijection on the set of paths as claimed. The restriction of this map to paths in G_T is also a bijection by definition of the multilabel routing graph G_{MT}. □

This proposition has the following immediate corollary.

Corollary 4.6. *Let $s, t \in V(G_T)$ be source and target of the path search problem. Then, a shortest path with respect to $c^{\mathcal{R}}$ in G_{MT} from $(s,0)$ to $(t,\star)$ yields a shortest s-t-path in G_T under the bijection of Proposition 4.5.*

Therefore, in the following we can apply the path search algorithm on the graph G_{MT} to obtain a shortest path in our original routing graph G_T.

4.2 Multilabel Future Cost Function

In this section, we define a future cost function in the multilabel graph G_M which does not discount the cost of edges on the last used reservation. To simplify our notation, we first make the following definition.

Definition 4.7. We say that a s-t-path S is *R_i-starting* for some $R_i \in \mathcal{R}$, if R_i is the first reservation it uses.

An R_i-starting path may take other non-reserved edges before entering R_i and may use other reservations than R_i afterwards as well. We propose a new future costs algorithm which is based on Algorithm 4 but precomputes for each vertex $v \in G_T$ two possible values of future costs. In the multilabel routing graph G_{MT} the future costs will be chosen depending on the label type l of a vertex (v, l) as shown below.

Proposition 4.8. *Algorithm 5 computes* $\mathrm{dist}_{(G,c^{\mathcal{R}})}(s,t)$ *correctly by computing a shortest s-t-path S with respect to $c^{\mathcal{R}}$. Secondly, S is R_i-starting, where i is the index returned by the algorithm. Lastly, d^+ is the length of a shortest non R_i-starting s-t-path with respect to $c^{\mathcal{R}}$ in G, if $i > 0$. If we again assume that* $\mathrm{dist}_{(G,c)}(u,v)$ *can be computed in constant time for any $u, v \in V(G)$, Algorithm 5 runs in $\mathcal{O}(|\mathcal{R}| \log |\mathcal{R}|)$-time.*

Proof. The correctness of $\mathrm{dist}_{(G,c^{\mathcal{R}})}(s,t)$ and the claim about the run time can be proved analogously as in the proof of Theorem 3.6.

Now, we show the claim on d^+. As shown in the proof of Theorem 3.6, in each iteration of the for loop in Line 2 of Algorithm 5 $d_{current}$ is set in Line 5 to the length of a shortest R_j-starting s-t-path in G with respect to $c^{\mathcal{R}}$. Hence after the iteration $j = k$ of the for loop, d is set to the length of a shortest s-t-path in G with respect to $c^{\mathcal{R}}$ which is R_l-starting for $1 \leq l \leq k$ or uses no reservation at all. The index i is set to the index of the reservation this shortest path uses first or 0, if it does not use any reservation. Furthermore after the same iteration $j = k$ of the for loop, d^+ is set to the second lowest value of $d_{current}$ which has occurred up to that iteration or $d_{(G,c)}(s,t)$ if this value is smaller. Therefore in the end, d^+ is the length of a shortest non R_i-starting s-t-path in G with respect to $c^{\mathcal{R}}$. □

Algorithm 5: Multilabel-Future-Costs.

Input:

- A grid graph G,
- a target $t \in V(G)$ and a source $s \in V(G)$,
- a set of reservations $\mathcal{R} = \{R_1, \ldots, R_k\}$ with gate vertices P and distances $d_P : P \to \mathbb{R}_{\geq 0}$ as computed in Algorithm 3.

Output:

- The length d of a shortest s-t-path S in G with respect to $c^{\mathcal{R}}$,
- the index i if S is R_i-starting and 0 otherwise,
- the length d^+ of a shortest non R_i-starting s-t-path S^+ in G with respect to $c^{\mathcal{R}}$

  if $i > 0$ and d otherwise.

Set $d := \text{dist}_{(G,c)}(s,t)$, $i := 0$ and $d^+ := d$.
for $j = 1, \ldots, k$ **do**
    Set $w :=$ Closest-Point(R_j, s).
    Let w_l, w_r be the first gate vertices left and right of w on R_j respectively or both equal to w, if w itself is a gate vertex.
    Set $d_{current} :=$ $\text{dist}_{(G,c)}(s,w) + \min(d_P(w_l) + \text{dist}_{(G,c^{\mathcal{R}})}(w_l, w), d_P(w_r) + \text{dist}_{(G,c^{\mathcal{R}})}(w, w_r))$.
    if $d_{current} < d$ **then**
        Set $d^+ := d$.
        Set $d := d_{current}$ and $i := j$.
    else if $d_{current} < d^+$ **then**
        Set $d^+ := d_{current}$
return (d, i, d^+).

Definition 4.9. For the multilabel routing graph G_{MT} we define the *multilabel future cost function* l_M as

$$l_M : V(G_{MT}) \to \mathbb{R}_{\geq 0},$$

where for an arbitrary vertex $(v, l) \in V(G_{MT})$ with

$$(d, i, d^+) := \text{Multilabel-Future-Costs}(G, t, v, \mathcal{R}),$$

as computed by Algorithm 5 we set

$$l_M(v, l) := \begin{cases} d^+ & \text{if } i = l, \\ d & \text{otherwise, and} \end{cases}$$
$$l_M(t, \star) := 0.$$

A practical advantage of this definition is that Algorithm 5 needs to be called only once for each vertex $v \in G_T$. If one saves the values (d, i, d^+) one can easily compute the values $l_M(v, l)$ for all $l \in L$ as needed during the execution of the path search algorithm.

Proposition 4.10. *Let $(v, l) \in V(G_{MT})$ be a fixed vertex in the multilabel routing graph. Then the length of a shortest (v, l)-$(t, \star)$-path with respect to $c^{\mathcal{R}}$ in the multilabel graph G_M equals $l_M((v, l))$.*

Proof. By Proposition 4.5, we can compute the length of the shortest (v, l)-$(t, \star)$-path in G_M by computing the length of a shortest v-t-path in G that additionally fulfills the condition which reservation must not be used first. Define

$$(d, i, d^+) := \text{Multilabel-Future-Costs}(G, t, v, \mathcal{R}).$$

We consider the following three cases.

Case 1: $l = 0$. By Proposition 4.8, we have

$$l_M((v, l)) = d = \text{dist}_{(G, c^{\mathcal{R}})}(v, t),$$

In this case, the claim holds trivially.

Case 2: $l > 0$ and $l \neq i$. By definition, we then have $l_M((v, l)) = d = \text{dist}_{(G_T, c^{\mathcal{R}})}(v, t)$ and there exists a shortest s-t-path which is R_i-starting by construction of

Algorithm 5. Hence, this path is not R_l-starting, as $l \neq i$ holds. It is also a shortest non R_l-starting path, since it is a general shortest s-t-path with respect to $c^{\mathcal{R}}$.

Case 3: $l > 0$ and $l = i$. In this case, $l_M((v,l)) = d^+$ holds by definition of l_M, which is exactly the length of a shortest non R_l-starting s-t-path with respect to $c^{\mathcal{R}}$ by Proposition 4.8. □

This proposition yields the corollary that l_M is indeed a future cost function.

Corollary 4.11. *The map l_M is a future cost function as defined in Section 2.1 in the multilabel routing graph G_{MT} for the target $(t, \star)$ with respect to $c^{\mathcal{R}}$.*

Proof. As explained in Section 2.1 the proof can be deduced immediately from the Lemmata 2.6 and 2.7. □

4.3 Grouped Multilabel Future Cost Function

The approach described in the last two sections bears the following two problems:

1. We have introduced $|\mathcal{R}| + 1$-many label types and therefore, the multilabel routing graph is $|\mathcal{R}| + 1$ times larger than the original routing graph G_T. This can easily lead to a higher number of labeled vertices during the path search algorithm.

2. We only avoid discounting the last left reservation in the computation of the future costs. This may be inaccurate at vertices close to two different reservations as shown in Example 4.12 below.

Example 4.12. For the grid graph and reservations given in Figure 8, assume that each edge on a reservation costs 1 unit and off the reservations 2 units with respect to $c^{\mathcal{R}}$. We examine the situation during the path search algorithm where the vertex v is permanently labeled but w is not. Thus, we consider the vertex w as a neighbor of v as a candidate to be permanently labeled next and compute $l(w)$ to this end. For the future costs $l_R(w)$ defined as the length of a shortest w-t-path in G with respect to $c^{\mathcal{R}}$ we have

$$l_R(w) = \mathrm{dist}_{(G,c^{\mathcal{R}})}(w,t) = c^{\mathcal{R}}(E(S_1)) = 13.$$

Due to the fact that S_1 is the unique shortest w-t-path with respect to $c^{\mathcal{R}}$ and R_2-starting, the index returned by Algorithm 5 is 2. Hence by definition of the

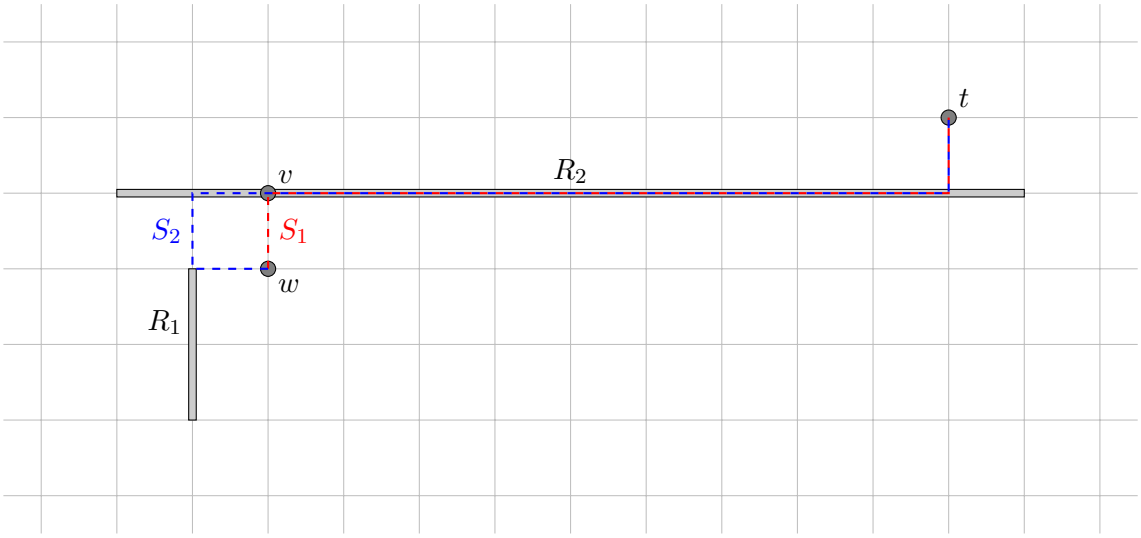


Figure 8: A grid graph with reservations $\mathcal{R} = \{R_1, R_2\}$, vertices v, w, t and two w-t-paths S_1, S_2.

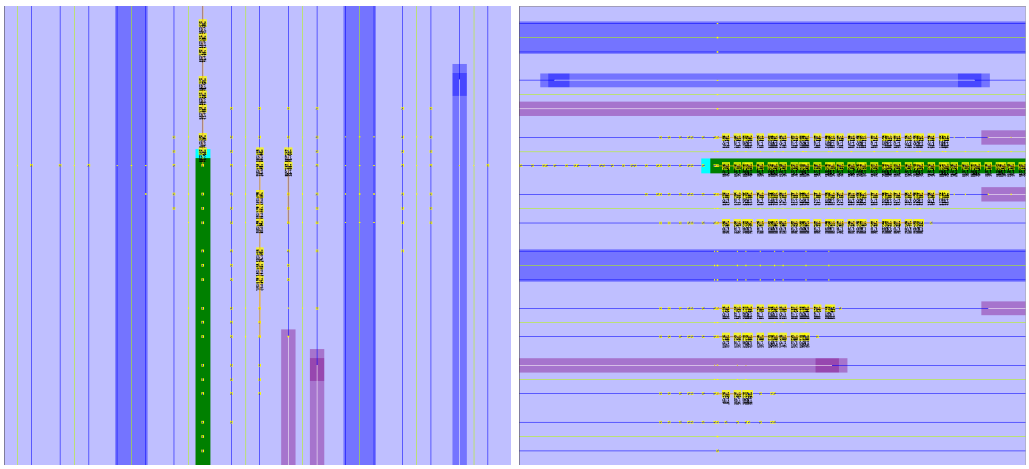
multilabel future cost function, $l_M(w)$ equals the length of a shortest w-t-path with respect to $c^{\mathcal{R}}$ which is not R_2-starting. The path S_2 is such a shortest path and we thus have

$$l_M(w) = c^{\mathcal{R}}(E(S_2)) = 16.$$

Hence, the multilabel future costs $l_M(w)$ of the vertex w are not significantly increased by the multilabel approach in this situation. On the other hand, one would hope to obtain future costs close to $\text{dist}_{(G,c^{\mathcal{R}\setminus\{R_2\}})}(w,t) = 22$ since we know that a shortest path can not take the reservation R_2 again after leaving it through the vertex v. This leads to many superfluous permanent labels. An example of a routing of a real chip with unnecessary permanent labels in an area close to two reservations is given in Figure 9.

To address the above problems we propose a method which firstly groups reservations and then introduces a label type for each group of reservations. We call a subset $K \subseteq \mathcal{R}$ a *group of reservations* and a partition $\mathcal{K} = \{K_1, \ldots, K_n\}$ of $\mathcal{R}$ a *grouping of* $\mathcal{R}$.

The key fact which allowed us to define the multilabel routing graph G_{MT} and valid future costs l_M in the previous sections was shown in Proposition 4.2, namely that shortest paths are $\mathcal{R}$-irreducible. Firstly, we generalize this notation to a grouping of $\mathcal{R}$.



(a) Layer 3 (b) Layer 4

Figure 9: Part of two consecutive layers at the end of the path search using the future cost multilabel framework. The target is located on a lower layer towards the right border of the figure. Each layer contains one reservation drawn in green. Blue and purple rectangles are previously routed nets and blockages. Permanently labeled vertices are drawn as small yellow and black squares. Only the labeled vertices with the label type which indicates that the path has left the reservation on Layer 3 are shown.

Definition 4.13. For a grouping $\mathcal{K}$, we define a s-t-path S to be $\mathcal{K}$-*reducible* if it re-enters the group it has left most recently. This means, for a fixed group $K \in \mathcal{K}$ there exists two edges $(u, v), (u', v') \in E(S)$ such that

- $u \in R$ and $v' \in R'$ for some potentially identical reservations $R, R' \in K$,
- $v, u' \notin R''$ for any $R'' \in K$,
- $(u', v') \in E(S_{[u,t]})$ and
- for all edges $(u'', v'') \in E(S_{[v,u']})$ it holds that $u'' \notin R''$ and $v'' \notin R''$ for any $R'' \in K$.

Otherwise, the path S is called $\mathcal{K}$-*irreducible*.

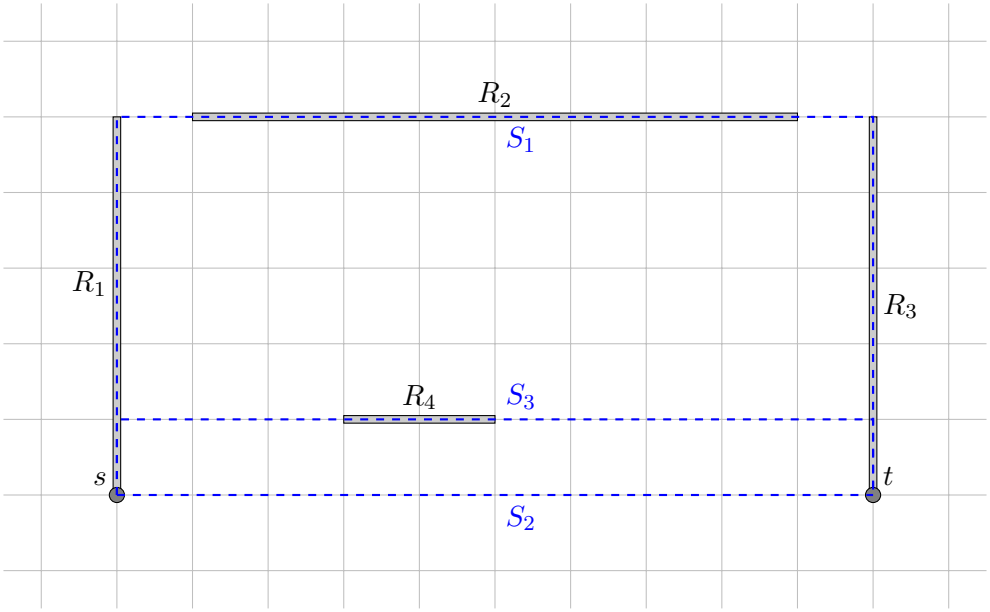


Figure 10: A grid graph with reservations $\mathcal{R} = \{R_1, R_2, R_3, R_4\}$, vertices s, t and three s-t-paths S_1, S_2, S_3.

Example 4.14. For the reservations and paths shown in Figure 10 consider the grouping

$$\mathcal{K} := \{\{R_1, R_2, R_3\}, \{R_4\}\}.$$

The path S_1 is $\mathcal{K}$-irreducible since it does not leave the group $\{R_1, R_2, R_3\}$ at all. Furthermore, the path S_2 is $\mathcal{K}$-reducible because it leaves the group $\{R_1, R_2, R_3\}$ and then re-enters it without using another group in between. On the other hand,

the path S_3 enters the group $\{R_4\}$ between leaving and re-entering the first group and is therefore $\mathcal{K}$-irreducible.

Remark 4.15. Due to our assumption that the reservations are pairwise disjoint, we have only reservations in preferred routing direction on every layer in practice. We can then group two reservations, if they are adjacent over one edge in z-dimension, called a *via*, or in direction orthogonal to the preferred direction on the layer, called a *jog*. However, we would also like to group reservations that are adjacent over a stacked via (vias which are longer than one edge) or a longer jog. To this end, we introduce point shaped reservations for all vertices on stacked vias or long jogs strictly in between its endpoints stemming from the track assignment. Hence, a path traversing these points between two reservations in preferred direction does not leave the group of reservations consisting of these point reservations and the two proper reservations.

For an arbitrary grouping of reservations a shortest s-t-path may be $\mathcal{K}$-reducible. In fact, even all shortest s-t-paths could be $\mathcal{K}$-reducible as shown in the following example.

Example 4.16. We consider again Figure 10 and now assume that each edge on a reservation costs 1 unit and off the reservations 2 units with respect to $c^{\mathcal{R}}$. Hence,

$$c^{\mathcal{R}}(E(S_1)) = 22 > 20 = c^{\mathcal{R}}(E(S_2)) = c^{\mathcal{R}}(E(S_3)),$$

and obviously S_2 and S_3 are shortest s-t-paths with respect to $c^{\mathcal{R}}$. Thus continuing Example 4.14, there exists both a $\mathcal{K}$-reducible and a $\mathcal{K}$-irreducible shortest s-t-path for the grouping $\mathcal{K}$. Furthermore, it is easy to see that for the grouping $\mathcal{K}' := \{\{R_1, R_2, R_3, R_4\}\}$ any shortest s-t-path is $\mathcal{K}'$-reducible.

Thus, we firstly define a property of a grouping $\mathcal{K}$ which will ensure that there exists at least one shortest path between any pair of vertices in G_T which is $\mathcal{K}$-irreducible.

Definition 4.17. Let $\mathcal{R}$ be a set of reservations and $\mathcal{K}$ be a grouping of $\mathcal{R}$. We call $\mathcal{K}$ *shortcut-free* if for any $K \in \mathcal{K}$, $R, R' \in K$ and any two vertices $v \in R$ and $w \in R'$ there exists a $\mathcal{K}$-irreducible shortest v-w-path with respect to $c^{\mathcal{R}}$ in the routing graph G_T.

For example in Figure 10, the grouping $\mathcal{K} = \{\{R_1, R_2, R_3\}, \{R_4\}\}$ is shortcut-free whereas the grouping $\mathcal{K}' = \{\{R_1, R_2, R_3, R_4\}\}$ is not shortcut-free as shown in

Example 4.16. The following proposition shows that for shortcut-free groupings one can always find $\mathcal{K}$-irreducible shortest paths.

Proposition 4.18. *Let $\mathcal{K}$ be a shortcut-free grouping and $s, t \in V(G_T)$ such that there exists an s-t-path in the routing graph G_T. Then there exists a shortest s-t-path with respect to $c^{\mathcal{R}}$ in G_T which is $\mathcal{K}$-irreducible.*

Proof. Let S be a shortest s-t-path with respect to $c^{\mathcal{R}}$ in the routing graph G_T and assume S is $\mathcal{K}$-reducible. Then due to the definition of shortcut-freeness, we can replace any $\mathcal{K}$-reducible path segment $S_{[u,v]}$ for a pair $u, v \in V(G_T)$ where both u, v lie on reservations of the same group by a $\mathcal{K}$-irreducible shortest u-v-path. Therefore, if we iteratively replace all such $\mathcal{K}$-reducible path segments by $\mathcal{K}$-irreducible ones of the same length (both segments are shortest between their endpoints), we obtain a shortest s-t-path which is $\mathcal{K}$-irreducible. □

This motivates the following problem:

SHORTCUT-FREE-GROUPING

Instance: A routing graph G_T and a set of reservations $\mathcal{R}$.

Task: Find a shortcut-free grouping $\mathcal{K}$ of $\mathcal{R}$.

A trivial solution would be to group $\mathcal{R}$ by groups consisting of one reservation each. However, this would not solve the problems discussed above. In the following we describe a more effective approach. The basic idea is to use the path search tree computed in the Preprocessing–Algorithm 3 and group the reservations by the connected components of this tree.

In Line 4 of Algorithm 3 we executed Dijkstra's algorithm on a graph having $P \cup \{t\}$ as its set of vertices where P are the gate vertices on the reservations $\mathcal{R}$. Now we assume that during the computation we also save a map $pre : P \to P \cup \{t\}$ giving the predecessor of each vertex in the path search tree.

Definition 4.19. For a group of reservations K we define the group graph T_K by setting:

$$\begin{aligned} V(T_K) :=& \{\text{endpoints of reservations in } K\} \cup \\ & \{v, w \mid v \in R, w \in R' \text{ for } R, R' \in K \text{ with } R \neq R' \text{ and } (v, w), (w, v) \in E(G_T)\}, \\ E(T_K) :=& \{\{v, w\} \mid v, w \in V(T_K) \text{ with } (v, w), (w, v) \in E(G_T) \text{ or } v, w \in R \\ & \text{for } R \in K \text{ such that } V(T_K) \text{ does not contain any vertex between } v, w \text{ on } R\}. \end{aligned}$$

Further we define a cost function $c^K : E(T_K) \to \mathbb{R}_{\geq 0}$ by setting for $\{v, w\} \in E(T_K)$

$$c^K(\{v,w\}) := \begin{cases} c^{\mathcal{R}}(E(P_{vw})) \text{ if } v, w \in R \text{ for some reservation } R \in K, \\ \text{dist}_{(G,c)}(v,w) \text{ otherwise,} \end{cases}$$

where P_{vw} is the v-w-path on the reservation R in G_T.

Algorithm 6: Shortcut-free grouping

Input:

- The routing graph G_T,
- a set of reservations $\mathcal{R} = \{R_1, \ldots, R_k\}$,
- a target $t \in V(G)$,
- a set of gate vertices P,
- a reservations map $r : P \to \mathcal{R}$,
- a predecessor map $pre : P \to P \cup \{t\}$.

Output: A shortcut-free grouping $\mathcal{K}$ of $\mathcal{R}$.

1 Define the tree T by setting $V(T) := P \cup \{t\}$ and

$$E(T) := \{\{u,v\} \mid u, v \in V(T) \text{ and } pre(u) = v\}.$$

2 Compute the partition $\{P_1, \ldots, P_n\}$ of P in connected components along the tree T where two vertices $p, q \in P$ are defined to be connected, if $r(p) = r(q)$ or $(p,q), (q,p) \in E(G_T)$ holds.

3 Compute the grouping $\mathcal{K} = \{K_1, \ldots, K_k\}$ to be the partition of $\mathcal{R}$ where two reservations are in the same group if and only if all their gate vertices are in the same part P_i for some $i = 1, \ldots, n$.

4 **for** $i = 1, \ldots, k$ **do**

5 | **while** T_{K_i} *contains a cycle* **do**

6 | | Remove a reservation in K_i which lies on a cycle in T_{K_i} and add it to the partition as a singleton.

7 | For all vertices $v_j, v_k \in V(T_{K_i})$ compute $\text{dist}_{(T_{K_i}, c^{K_i})}(v_j, v_k)$ in T_{K_i}.

8 | **if** *for any vertices* $v_j, v_k \in V(T_{K_i})$ *it holds that* $\text{dist}_{(T_{K_i}, c^{K_i})}(v_j, v_k) > \text{dist}_{(G,c)}(v_j, v_k)$ **then**

9 | | Replace K_i by the trivial partition consisting of single elements only.

Theorem 4.20. *Algorithm 6 correctly computes a shortcut-free grouping of $\mathcal{R}$. If we again assume that $\mathrm{dist}_{(G,c)}(u,v)$ can be computed in constant time for any $u,v \in V(G)$, it can be implemented to run in $\mathcal{O}(|\mathcal{R}|^2)$-time.*

Proof. First, we prove the correctness of the algorithm. So let $K \in \mathcal{K}$ be a group of the grouping $\mathcal{K}$ returned by the algorithm. Assume $|K| > 1$ holds since otherwise any shortest path between vertices in K is trivially $\mathcal{K}$-irreducible. Pick vertices $v, w \in V(G_T)$ with $v \in R$ and $w \in R'$ for some different reservations $R, R' \in K$. We need to show that there exists a shortest v-w-path in G_T which is $\mathcal{K}$-irreducible. Let v_l and v_r be the vertices in T_K that are the left and right neighbor of v respectively on the reservation R. If v itself is a vertex in T_K define v_l and v_r to be equal to v. Analogously, define the vertices w_l and w_r in T_K for w.

Due to the condition in Line 5 we can assume that the graph T_K is cycle-free. It is in fact a tree, since it is connected due to the choice of the partition in Line 2. Hence, we can define S_{ab} to be the unique v_a-v_b-path in T_K for $a, b \in \{l, r\}$. By definition of the tree T_K, these paths naturally give rise to v_a-v_b-paths in G_T of the same length with respect to $c^{\mathcal{R}}$, we define them to be S^R_{ab} for $a, b \in \{l, r\}$. Due to the condition in Line 8, S^R_{ab} is a shortest v_a-v_b-path for $a, b \in \{l, r\}$ in G_T with respect to $c^{\mathcal{R}}$ where we only discount the costs on reservations in K. Due to the fact that T_K is a tree and the choice of v_l, v_r and w_l, w_r respectively, at least one of the paths S^R_{ab} for $a, b \in \{l, r\}$ passes both v and w; call this path S. Therefore, the path segment $S_{[v,w]}$ is trivially also a shortest v-w-path in G_T with respect to $c^{\mathcal{R}}$ where we again only discount the costs on reservations in K. Furthermore, the path segment $S_{[v,w]}$ is $\mathcal{K}$-irreducible, since all its vertices are contained in the group K. Thus, either $S_{[v,w]}$ is a proper shortest v-w-path in G_T with respect to $c^{\mathcal{R}}$ or there exists another shortest v-w-path in G_T with respect to $c^{\mathcal{R}}$ which uses at least one reservation not contained in K which makes this path also $\mathcal{K}$-irreducible. So, in any case there exists a $\mathcal{K}$-irreducible shortest v-w-path in G_T with respect to $c^{\mathcal{R}}$ which shows that the group K is shortcut-free.

Now we show the claim on the run time. One can compute the connected components of an undirected graph in linear time as described for example in Proposition 2.17 in [KV12]. Since the tree T is of size $\mathcal{O}(|\mathcal{R}|^2)$, Line 2 also has a run time of $\mathcal{O}(|\mathcal{R}|^2)$.

On the other hand, Line 3 can be trivially implemented to run in $\mathcal{O}(|\mathcal{R}|^2)$-time in the following two steps. Firstly, one iterates through all gate vertices $p \in P$ with $p \in P_i$ for some $1 \leq i \leq n$ and saves the index i on the reservation $r(p)$. Secondly, one inserts reservation $R \in \mathcal{R}$ in group K_i for some $1 \leq i \leq n$, if all saved indices

of the gate vertices on R equal i or into a singleton new group, if the saved indices on R are ambiguous.

The condition in Line 5 can be checked via a depth first search on the reservations in each group which also yields the reservations which need to removed in order for T_{K_i} to be a tree. Hence, the total run time of Line 5 for all groups is $\mathcal{O}(|\mathcal{R}|^2)$.

Next, we claim that $V(T_{K_i})$ is of size $\mathcal{O}(|K_i|)$ for each $i = 1, \dots, k$ which is considered in the else block from Line 7 on. We show this claim for an arbitrary but fixed index $1 \leq i \leq k$ for which K_i is considered from Line 7 on. Since T_{K_i} is connected there exists an order $\{R_1, \dots, R_\ell\}$ of the reservations in K_i such that R_j is connected[1] to at least one of the reservations $\{R_1, \dots, R_{j-1}\}$ for $j = 2, \dots, \ell$. In fact, R_j must then be connected to exactly one of the reservations $\{R_1, \dots, R_{j-1}\}$, since the reservations $\{R_1, \dots, R_{j-1}\}$ are connected[2] by the choice of the order, and we would otherwise obtain a cycle in the graph T_{K_i}. Thus, every reservation $R_j \in K$ for $j = 1, \dots, \ell$ contributes at most four vertices to T_{K_i}, namely its two endpoints and the two vertices at the connection to one of the reservations $\{R_1, \dots, R_{j-1}\}$. Therefore, $V(T_{K_i})$ is of size $\mathcal{O}(|K_i|)$.

Thus, we can compute for any vertex $v \in V(T_{K_i})$ a shortest-path-tree with source v by simply propagating the edge costs along the tree T_{K_i} in $\mathcal{O}(|K_i|)$-time. Using this fact, both Line 7 and Line 8 can easily be implemented to run in $\mathcal{O}(|K_i|^2)$ for each iteration. So over all iterations, these lines also have a run time of $\mathcal{O}(|\mathcal{R}|^2)$. Therefore, the total run time of the algorithm is $\mathcal{O}(|\mathcal{R}|^2)$. □

Analogously to the previous two sections, we can define for a given shortcut-free grouping $\mathcal{K} = \{K_1 \dots, K_n\}$ the *grouped multilabel graph* $G_\mathcal{K}$ and the *grouped routing multilabel graph* $G_{\mathcal{K}T}$. In this case, we introduce the label types $L_\mathcal{K} := \{0, \dots, n\}$ for each group in $\mathcal{K}$ instead of for each reservation. The graphs $G_\mathcal{K}$ and $G_{\mathcal{K}R}$ are then defined entirely analogously as the multilabel graph G_M and the multilabel routing graph G_{MT}. We again do not include the potential edges $((v, l), (v', l')) \in V(G_\mathcal{K})^2$ where l equals the group index of the reservation v' lies on, because these edges would yield $\mathcal{K}$-reducible paths. We can still find a shortest path for any pair of source and target vertex in $G_\mathcal{K}$ by Proposition 4.18. Finally, one can analogously define a valid future costs function $l_\mathcal{K}$ based on a slightly modified version of Algorithm 5. One just needs to keep track of the index of the group a shortest paths starts with and return this index at the end of the algorithm.

[1] We call two reservations R, R' connected if there exists $v \in R$ and $w \in R'$ such that $(v, w), (w, v) \in E(G_T)$.

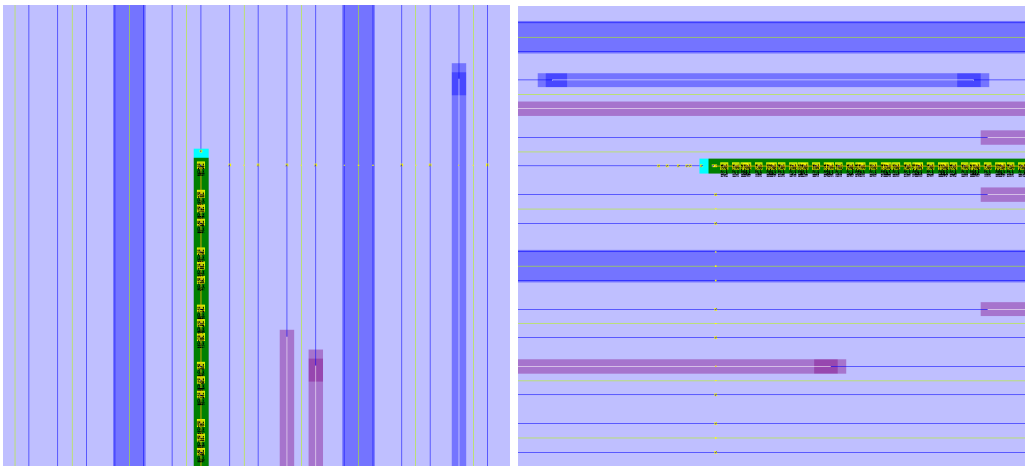
[2] We call a group of reservations K connected, if their graph T_K is connected.

Example 4.21. We again consider Figure 8 as discussed in Example 4.12. The grouping $\mathcal{K} := \{\{R_1, R_2\}\}$ is clearly shortcut-free. Hence, the future cost $l_{\mathcal{K}}(w)$ equals the length of a shortest w-t-path with respect to $c^{\mathcal{R}}$ which is not R_1- and R_2-starting. Therefore, we have

$$l_{\mathcal{K}}(w) = \text{dist}_{(G, c^{\mathcal{R}\setminus\{R_1,R_2\}})} = 22$$

as desired. Figure 11 shows that one can avoid the unnecessary permanent labels occurring in Figure 9 by using the grouped multilabel framework.

.



(a) Layer 3 (b) Layer 4

Figure 11: Same part of chip as in Figure 9 after the path search using the grouped multilabel framework. The two reservations on layer 3 and 4 form one group. The drawn permanently labeled vertices are of label type 0 which indicates that these vertices have not yet left the shown group of reservations. There are no other permanently labeled vertices of other label types in this region of the chip.

5 Theoretical Bounds on the Number of Permanent Labels

Using the future cost function building on the multilabel framework developed in the previous section, we now give upper bounds on the number of vertices permanently labeled during the execution of Dijkstra's algorithm.

5.1 Path Search without Reservations

Firstly, we consider the usual path search without any reservations. Before proving an upper bound in this case, we make the following definitions to simplify our notation in the following arguments. The basic idea is to introduce approximative edge cost which are uniform in x- and y-direction. Subsequently, we obtain extended bounding boxes using this cost function which will contain all permanently labeled vertices.

- We define coarser edge cost $c^q_{approx} : E(G) \to \mathbb{R}_{\geq 0}$ for some $0 < q \leq 1$ by setting for an edge $((x_1, y_1, z_1), (x_2, y_2, z_2)) \in E(G)$

$$c^q_{approx}(((x_1, y_1, z_1), (x_2, y_2, z_2))) = \begin{cases} q & \text{if } z_1 = z_2, \\ qc_z & \text{otherwise,} \end{cases}$$

  where $c_z := \min_{z^G_{\min} \leq z_0 \leq z^G_{\max}} c^{z_0}_z$.

- Further, for s, t vertices in G we define the *bounding box* $BB(s,t)$ by setting

$$BB(s,t) := \\ [\min(s_x, t_x), \max(s_x, t_x)] \times [\min(s_y, t_y), \max(s_y, t_y)] \times [\min(s_z, t_z), \max(s_z, t_z)].$$

- Additionally, for $s, t \in V(G)$, any $0 < q \leq 1$ and a constant C with $\text{dist}_{(G, c^q_{approx})}(s,t) < C$, we define the *extended bounding box* $EBB(s,t,q,C)$ by setting for $\Delta := C - \text{dist}_{(G, c^q_{approx})}(s,t)$

$$\begin{aligned} EBB(s,t,q,C) := &\left[\min(s_x, t_x) - \left\lceil \frac{\Delta}{2q} \right\rceil, \max(s_x, t_x) + \left\lceil \frac{\Delta}{2q} \right\rceil\right] \times \\ &\left[\min(s_y, t_y) - \left\lceil \frac{\Delta}{2q} \right\rceil, \max(s_y, t_y) + \left\lceil \frac{\Delta}{2q} \right\rceil\right] \times \\ &\left[\min(s_z, t_z) - \left\lceil \frac{\Delta}{2qc_z} \right\rceil, \max(s_z, t_z) + \left\lceil \frac{\Delta}{2qc_z} \right\rceil\right]. \end{aligned}$$

- Lastly, for any rectangular box $Q := [a_x, b_x] \times [a_y, b_y] \times [a_z, b_z]$ with $a, b \in V(G)$ we define $NGP(Q)$ to be the number of grid points in Q given by the formula

$$(b_x - a_x + 1)(b_z - a_z + 1)(b_z - a_z + 1).$$

The application of the extended bounding box becomes apparent in the following lemma.

Lemma 5.1. *For given vertices $s, t \in V(G)$, any $0 < q \leq 1$ and a constant C with $\mathrm{dist}_{(G, c^q_{approx})}(s, t) < C$ we consider a vertex $v \in V(G)$ such that $v \notin EBB(s, t, q, C)$. Then, it holds that*

$$\mathrm{dist}_{(G, c^q_{approx})}(s, v) + \mathrm{dist}_{(G, c^q_{approx})}(v, t) > C.$$

Proof. Let $e, b \in V(G)$ be the closest vertex to v with respect to c^q_{approx} in $EBB(s, t, q, C)$ and $BB(s, t)$ respectively. Figure 12 depicts the situation of the proof. Then, we can decompose the distances as follows

$$\begin{aligned}
\mathrm{dist}_{(G, c^q_{approx})}(s, v) &= \mathrm{dist}_{(G, c^q_{approx})}(s, b) + \mathrm{dist}_{(G, c^q_{approx})}(b, e) + \mathrm{dist}_{(G, c^q_{approx})}(e, v), \\
\mathrm{dist}_{(G, c^q_{approx})}(v, t) &= \mathrm{dist}_{(G, c^q_{approx})}(v, e) + \mathrm{dist}_{(G, c^q_{approx})}(e, b) + \mathrm{dist}_{(G, c^q_{approx})}(b, t).
\end{aligned}$$

Summing up these equations yields

$$\begin{aligned}
&\mathrm{dist}_{(G, c^q_{approx})}(s, v) + \mathrm{dist}_{(G, c^q_{approx})}(v, t) \\
=& \mathrm{dist}_{(G, c^q_{approx})}(s, b) + 2\,\mathrm{dist}_{(G, c^q_{approx})}(b, e) + 2\,\mathrm{dist}_{(G, c^q_{approx})}(e, v) + \mathrm{dist}_{(G, c^q_{approx})}(b, t) \\
=& \mathrm{dist}_{(G, c^q_{approx})}(s, t) + 2\,\mathrm{dist}_{(G, c^q_{approx})}(b, e) + 2\,\mathrm{dist}_{(G, c^q_{approx})}(e, v).
\end{aligned}$$

Recall that we defined $\Delta := C - \mathrm{dist}_{(G, c^q_{approx})}(s, t)$. Thus due to the definition of the extended bounding box $EBB(s, t, q, C)$, we can further simplify the last equation to obtain

$$\begin{aligned}
&\mathrm{dist}_{(G, c^q_{approx})}(s, v) + \mathrm{dist}_{(G, c^q_{approx})}(v, t) \\
\geq& \mathrm{dist}_{(G, c^q_{approx})}(s, t) + \Delta + 2\,\mathrm{dist}_{(G, c^q_{approx})}(e, v) \\
=& C + 2\,\mathrm{dist}_{(G, c^q_{approx})}(e, v).
\end{aligned}$$

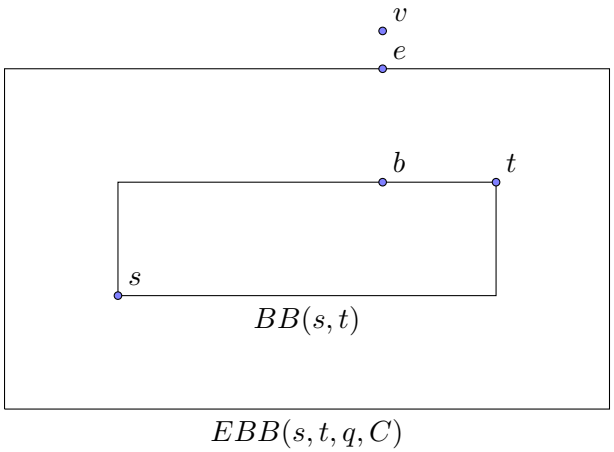


Figure 12: The situation as in the proof of Lemma 5.1.

Due to the fact that v was chosen to lie outside of $EBB(s, t, q, C)$, we have

$$\mathrm{dist}_{(G,c^q_{approx})}(e, v) > 0,$$

and the claim is proven. □

A first corollary of Lemma 5.1 easily yields an upper bound for the number of permanent labels during the path search without reservations.

Corollary 5.2. *Let NL be the number of permanent labels marked by Algorithm 1 in the routing graph G_T with source s, target t, cost c, and future cost function $l : V(G_T) \to \mathbb{R}_{\geq 0}$ defined by $l(v) := \mathrm{dist}_{(G,c)}(v, t)$. Then,*

$$NL \leq NGP(EBB(s, t, 1, \mathrm{dist}_{(G_T,c)}(s, t))).$$

Proof. By construction of Algorithm 1 with future cost function l, a vertex $v \in V(G)$ can only be labeled permanently, if

$$\mathrm{dist}_{(G,c)}(s, v) + \mathrm{dist}_{(G,c)}(v, t) \leq \mathrm{dist}_{(G_T,c)}(s, t)$$

holds, since $\mathrm{dist}_{(G_T,c)}(s, t)$ is the output of Algorithm 1 called on this instance. Hence, applying Lemma 5.1 with $q = 1$ and $C = \mathrm{dist}_{(G_T,c)}(s, t)$ shows that only vertices in $EBB(s, t, 1, \mathrm{dist}_{(G_T,c)}(s, t))$ can be labeled permanently, since c^1_{approx} is a lower bound of c for any edge $e \in E(G)$ by the definitions of c^q_{approx} and c. □

5.2 Path search with Reservations

As a second step, we study the path search with reservations in the multilabel framework introduced in Section 4.1 where we apply the future cost function l_M defined in Section 4.2. We again aim to give an upper bound on the number of permanently labeled vertices in terms of the length of the path found by the path search algorithm in G_{MT} with cost $c^{\mathcal{R}}$.

To this end, let $\mathcal{R}$ be a set of k-many reservations and $(s,0),(t,\star) \in V(G_{MT})$ be source and target of a path search problem in G_{MT}. We generalize the edge cost c^1_{approx} to a cost function $c^{\mathcal{R}}_{approx} : E(G_M) \to \mathbb{R}_{\geq 0}$ by setting for an edge $((v,l)(v',l')) \in E(G_M)$

$$c^{\mathcal{R}}_{approx}(((v,l)(v',l'))) := \begin{cases} c^q_{approx}((v,v')) \text{ if } v,v' \in R \text{ for some } R \in \mathcal{R}, \\ c^1_{approx}((v,v')) \text{ otherwise,} \end{cases}$$

where q is the reduction factor of the cost function $c^{\mathcal{R}}$ and (v,v') is the corresponding edge in G. The edges of the form $((t,l),(t,\star))$ for any label type $l \in L_M$ still have cost 0 with respect to $c^{\mathcal{R}}_{approx}$. Then by assumption on $c : E(G) \to \mathbb{R}_{\geq 0}$ it holds that

$$c^q_{approx}(e) \leq c^{\mathcal{R}}_{approx}(e) \leq c^{\mathcal{R}}(e) \tag{3}$$

for any $e \in E(G_M)$.

Now, consider the following two paths:

- S_R is a shortest $(s,0)$-$(t,\star)$-path in the multilabel routing graph G_{MT} with respect to the cost function $c^{\mathcal{R}}$,
- S_G is a shortest $(s,0)$-$(t,\star)$-path in the multilabel graph G_M with respect to the cost function $c^{\mathcal{R}}_{approx}$,

and set

$$\delta := c^{\mathcal{R}}(E(S_R)) - c^{\mathcal{R}}_{approx}(E(S_G)).$$

To simplify our arguments we make the following definition.

Definition 5.3. We call an $(s,0)$-$(t,\star)$-path in G_M of cost at most $c^{\mathcal{R}}(E(S_R))$ with respect to $c^{\mathcal{R}}_{approx}$ a *labeling path.*

Lemma 5.4. *Let (v,l) be a vertex in G_M which was permanently labeled by the path search algorithm applied to the graph G_M with source $(s,0)$, target $(t,\star)$, cost*

$c^{\mathcal{R}}$ *and future cost function* l_M*. Then there exists a labeling path passing the vertex* (v, l)*.*

Proof. The fact that (v, l) is permanently labeled implies by construction of Algorithm 1 that

$$\text{dist}_{(G_{MT}, c^{\mathcal{R}})}((s, 0), (v, l)) + l_M((v, l)) \leq c^{\mathcal{R}}(E(S_R)). \tag{4}$$

As shown in Proposition 4.10, $l_M((v, l))$ equals the length of a shortest (v, l)-$(t, \star)$-path with respect to $c^{\mathcal{R}}$ in G_M. Hence, concatenating a shortest $(s, 0)$-(v, l)-path with a shortest (v, l)-$(t, \star)$-path both in G_M with respect to $c^{\mathcal{R}}$ yields an $(s, 0)$-$(t, \star)$-path in G_M of cost at most $c^{\mathcal{R}}(E(S_R))$ with respect to $c^{\mathcal{R}}$ due to inequality (4). Due to inequality (3), its cost is also at most $c^{\mathcal{R}}(E(S_R))$ with respect to $c^{\mathcal{R}}_{approx}$. Thus, this path is a labeling path which passes (v, l). □

We will use the converse statement of Lemma 5.4 to show that a vertex is not permanently labeled. To prove upper bounds on the number of permanent labels for the path search with reservations, we make the following key assumptions:

(i) Every shortest $(s, 0)$-$(t, \star)$-path in G_M with respect to $c^{\mathcal{R}}_{approx}$ uses each reservation in the fixed order $(R_1, \ldots, R_k)$. Additionally, each such shortest path accesses and exits every reservation $R_i \in \mathcal{R}$ through the same vertices; we call these vertices *access vertex* a_i and *exit vertex* e_i of R_i for $i = 1, \ldots, k$.

(ii) Every labeling path uses each reservation exactly once and in the order $(R_1, \ldots, R_k)$.

Proposition 5.5. *For every reservation* $R_i \in \mathcal{R}$ *it holds that:*

(a) There exists a last access *vertex* la_i *and a* first exit *vertex* fe_i *on* R_i*. This means that every labeling path uses the reservation* R_i *at least between the vertices* la_i *and* fe_i*.*

(b) The vertices la_i *and* fe_i *lie between the vertices* a_i *and* e_i *on* R_i*. Furthermore, the distance between* a_i *and* la_i *and between* fe_i *and* e_i *is* $\left\lfloor \frac{\delta}{1-q} \right\rfloor$ *with respect to the* L_1 *distance.*

Proof. Figure 13 depicts the situation of the proof in one example. Let S be a labeling path. By assumption (ii) S uses every reservation exactly once and in the order $(R_1, \ldots, R_k)$. Hence, we can define a_i^S to be the vertex on which S accesses the reservation R_i and e_i^S the vertex through which S exits R_i for $i = 1, \ldots, k$.[3]

[3] If s lies on R_1, we define a_1^S to be $(s, 0)$. Similarly, if t lies on R_k, we set e_k^S to be $(t, 0)$.

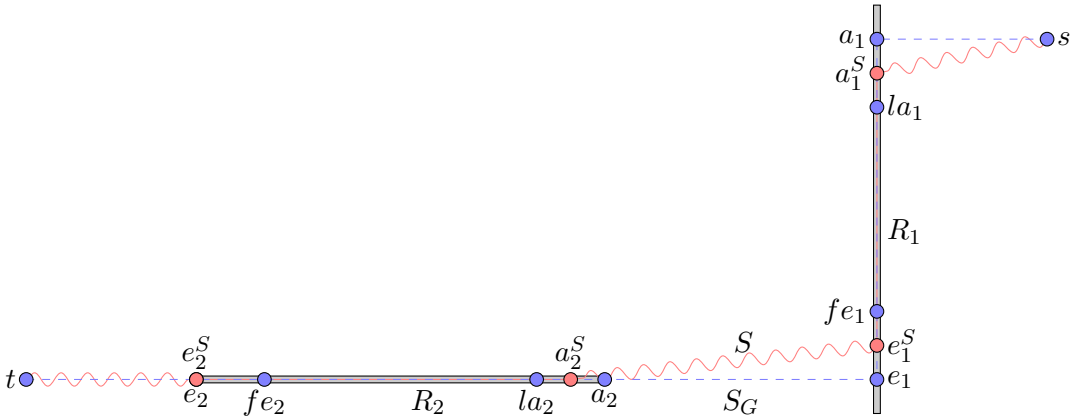


Figure 13: The situation as in the proof of Proposition 5.5.

Additionally, we set

$$\Delta a_i^S := \begin{cases} \|a_i^S - a_i\|_1 & \text{if } a_i^S \text{ lies between } a_i \text{ and } e_i, \\ 0 & \text{otherwise,} \end{cases}$$

$$\Delta e_i^S := \begin{cases} \|e_i^S - e_i\|_1 & \text{if } e_i^S \text{ lies between } a_i \text{ and } e_i, \\ 0 & \text{otherwise.} \end{cases}$$

for $i = 1, \ldots, k$. The numbers Δa_i^S and Δe_i^S count how many edges on R_i the path S does not use between the vertices a_i and e_i.

Now, we consider for a fixed i with $1 \leq i \leq k-1$ the path segment $S_{[e_i^S, a_{i+1}^S]}$. The segment $S_{[e_i^S, a_{i+1}^S]}$ uses at least as many edges as the path segment $S_{G_{[e_i^S, a_{i+1}^S]}}$, since S_G is a shortest $(s,0)$-$(t,\star)$-path with respect to $c^{\mathcal{R}}_{approx}$ which exits R_i through e_i and enters R_{i+1} via a_{i+1} by assumption (i). By definition of $c^{\mathcal{R}}_{approx}$, every edge which does not lie on a reservation costs at least $1-q$ units more than a reserved edge. By assumption (ii) $S_{[e_i^S, a_{i+1}^S]}$ can not use any other reservation than R_i and R_{i+1}. Combining the last three arguments yields

$$(1-q)\left(\Delta e_i^S + \Delta a_{i+1}^S\right) \leq c^{\mathcal{R}}_{approx}\left(E\left(S_{[e_i^S, a_{i+1}^S]}\right)\right) - c^{\mathcal{R}}_{approx}\left(E\left(S_{G_{[e_i^S, a_{i+1}^S]}}\right)\right). \tag{5}$$

Analogously, one can argue about the segments of S before accessing R_1 and after exiting R_k to obtain

$$(1-q)\Delta a_1^S \leq c^{\mathcal{R}}_{approx}\left(E\left(S_{\left[(s,0),a_1^S\right]}\right)\right) - c^{\mathcal{R}}_{approx}\left(E\left(S_{G_{\left[(s,0),a_1^S\right]}}\right)\right), \tag{6}$$

$$(1-q)\Delta e_k^S \leq c^{\mathcal{R}}_{approx}\left(E\left(S_{\left[e_k^S,(t,\star)\right]}\right)\right) - c^{\mathcal{R}}_{approx}\left(E\left(S_{G_{\left[e_k^S,(t,\star)\right]}}\right)\right). \tag{7}$$

Due to the fact that the paths S and S_G agree between a_i^S and e_i^S on the reservation R_i for $i = 1, \ldots, k$, we obtain by summing up inequality (5) for $i = 1, \ldots, k-1$ with the two inequalities (6) and (7)

$$(1-q)\sum_{i=1}^{k}\left(\Delta e_i^S + \Delta a_i^S\right) \leq c^{\mathcal{R}}_{approx}(E(S)) - c^{\mathcal{R}}_{approx}(E(S_G)). \tag{8}$$

By definition of the labeling path S we have

$$c^{\mathcal{R}}_{approx}(E(S)) \leq c^{\mathcal{R}}(E(S_R)) = c^{\mathcal{R}}_{approx}(E(S_G)) + \delta,$$

and hence

$$c^{\mathcal{R}}_{approx}(E(S)) - c^{\mathcal{R}}_{approx}(E(S_G)) \leq \delta.$$

Combining the last inequality with (8) we obtain

$$(1-q)\sum_{i=1}^{k}\left(\Delta e_i^S + \Delta a_i^S\right) \leq \delta,$$

and therefore it holds that

$$\Delta e_i^S \leq \left\lfloor \frac{\delta}{1-q} \right\rfloor,$$
$$\Delta a_i^S \leq \left\lfloor \frac{\delta}{1-q} \right\rfloor,$$

for all $i = 1, \ldots, k$. Due to the fact that S is an arbitrary labeling path, this shows that every reservation has a last access vertex and a first exit vertex. Furthermore, these vertices lie between a_i and e_i and their respective L_1 distance to a_i and e_i is at most $\left\lfloor \frac{\delta}{1-q} \right\rfloor$ as claimed. □

In particular, Proposition 5.5 implies that every reservation $R \in \mathcal{R}$ is at least $\left\lfloor \frac{\delta}{1-q} \right\rfloor$ long under the assumptions (i) and (ii). We will denote by fe_i and la_i both the vertices in G and the vertices $(fe_i, 0)$ and $(la_i, 0)$ to simplify the notation. It will be clear from the context to which vertex we refer at any given argument. For convenience we also define $fe_0 := s$ and $la_{k+1} := t$ and set for all $i = 0, \ldots, k$,

$$C_i := \mathrm{dist}_{(G_M, c^{\mathcal{R}}_{approx})}(fe_i, la_{i+1}) + \delta.$$

Theorem 5.6. *Let NLM be the number of permanent labels marked by Algorithm 1 in the multilabel routing graph G_{MT} with source $(s, 0)$, target $(t, \star)$, cost $c^{\mathcal{R}}$, and future cost function l_M. Then,*

$$NLM \leq \sum_{i=1}^{k} \|e_i - a_i\|_1 + \sum_{i=0}^{k} NGP(EBB(fe_i, la_{i+1}, q, C_i)) + 1. \tag{9}$$

Proof. The first term on the right hand side of (9) equals the number of vertices in G_M on the reservations of label type 0. By definition of the multilabel graph G_M, these are the only possible labeled vertices on the reservations. The term 1 on the right hand side stems from the fact that we need to label the target vertex $(t, \star)$ separately. Hence, from now on we focus on the permanent labels on vertices not lying on reservations and show that their cardinality is bounded by the middle term on the right hand side of inequality (9).

Let $(v, l) \in V(G_M)$ with $l \in L_M$ be a vertex such that v does not lie on any reservation and not in the extended bounding box $EBB(fe_l, la_{l+1}, q, C_l)$. Assume (v, l) is labeled permanently. Then by Lemma 5.4, there exists a labeling path S in G_M passing through (v, l). By Proposition 5.5, S passes all vertices fe_i and la_{i+1} for $i = 0, \ldots, k$ in that order. By definition of the multilabel framework S passes (v, l) between fe_l and la_{l+1} since the label type l indicates that S has left the reservation R_l most recently. The label type l could also be 0 which means that S has not entered any reservation yet. Due to the fact that v does not lie in $EBB(fe_l, la_{l+1}, q, C_j)$, we can deduce from Lemma 5.1,

$$\mathrm{dist}_{(G, c^q_{approx})}(fe_l, v) + \mathrm{dist}_{(G, c^q_{approx})}(v, la_{l+1}) > C_l.$$

The inequality (3) implies

$$\mathrm{dist}_{(G_M, c^{\mathcal{R}}_{approx})}(fe_l, (v, l)) + \mathrm{dist}_{(G_M, c^{\mathcal{R}}_{approx})}((v, l), la_{j+1}) > C_l.$$

Due to the fact that the path segment $S_{[fe_l,la_{l+1}]}$ passes (v,l), we can deduce from this inequality

$$c^{\mathcal{R}}_{approx}(E(S_{[fe_l,la_{l+1}]})) > C_l = \text{dist}_{(G_M,c^{\mathcal{R}}_{approx})}(fe_l, la_{l+1}) + \delta. \tag{10}$$

On the one hand side we have

$$\begin{aligned}
&c^{\mathcal{R}}_{approx}(E(S)) \\
=&c^{\mathcal{R}}_{approx}(E(S_{[(s,0),fe_l]})) + c^{\mathcal{R}}_{approx}(E(S_{[fe_l,la_{l+1}]})) + c^{\mathcal{R}}_{approx}(E(S_{[la_{l+1},(t,\star)]})) \\
\geq &\text{dist}_{(G_M,c^{\mathcal{R}}_{approx})}((s,0), fe_l) + c^{\mathcal{R}}_{approx}(E(S_{[fe_l,la_{l+1}]})) + \text{dist}_{(G_M,c^{\mathcal{R}}_{approx})}(la_{l+1}, (t,\star)).
\end{aligned}$$

On the other hand the fact that S_G is a shortest $(s,0)$-$(t,\star)$-path in G_M with respect to $c^{\mathcal{R}}_{approx}$ which passes both fe_l and la_{l+1} implies

$$\begin{aligned}
c^{\mathcal{R}}_{approx}(E(S_G)) = &\text{dist}_{(G_M,c^{\mathcal{R}}_{approx})}((s,0), fe_l) + \text{dist}_{(G_M,c^{\mathcal{R}}_{approx})}(fe_l, la_{l+1}) + \\
&\text{dist}_{(G_M,c^{\mathcal{R}}_{approx})}(la_{l+1}, (t,\star))
\end{aligned}$$

Therefore, adding $\text{dist}_{(G_M,c^{\mathcal{R}}_{approx})}((s,0), fe_l) + \text{dist}_{(G_M,c^{\mathcal{R}}_{approx})}(la_{l+1}, (t,\star))$ to both sides of inequality (10) and using the last two arguments, we obtain

$$c^{\mathcal{R}}_{approx}(E(S)) > c^{\mathcal{R}}_{approx}(E(S_G)) + \delta = c^{\mathcal{R}}(S_R).$$

This contradicts the fact that S is a labeling path. Hence, (v,l) can not be labeled permanently. This proves that a vertex (v,l) such that v does not lie on any reservation, can only be labeled permanently, if we have

$$v \in EBB(fe_l, la_{l+1}, q, C_l).$$

This immediately implies the theorem. □

Figure 14 shows the labeled area in the example introduced in Figure 13. Also in the real world instance depicted in Figure 7, one can see that using the multilabel framework only a small rectangle between the target and an endpoint of a reservation is labeled in this part of the design.

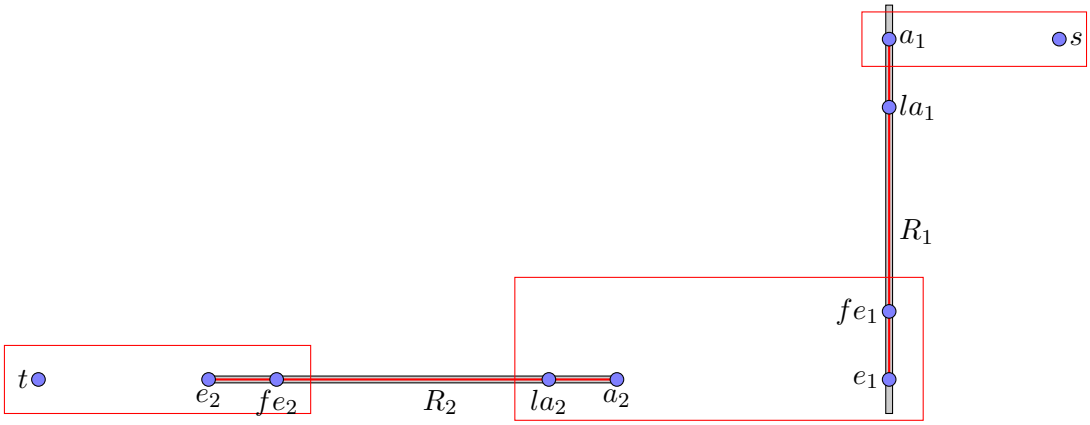


Figure 14: The example as in Figure 13. The red rectangles and lines contain all vertices which can be permanently labeled due to Theorem 5.6.

6 Results

In this section we describe the implementation of our algorithms within BonnRoute. Subsequently, we present experimental results of these algorithms on a testbed of large-scale real world designs provided by our industry partner IBM.

6.1 Implementation in BonnRoute

BonnTools are a software package stemming from a longstanding cooperation between the Research Institute for Discrete Mathematics at the University of Bonn and IBM. The aim of this project is to develop mathematical programs which design complex integrated circuits efficiently. For an overview of BonnTools see [KRV07]. The part of BonnTools we are mostly concerned with is BonnRoute, its routing tool. The algorithms presented in Sections 3 and 4 have been implemented and tested in BonnRoute.

The basis of the path search algorithm applied by BonnRoute is Algorithm 1 with the future cost function described by Henke in [Hen16]. Additional effort is made to handle more general situations such as multiple source and target vertices. A chip consists of a set of *routing layers* each having one *preferred direction*. An edge orthogonal to this preferred direction is called *jog* whereas an edge connecting two neighboring layers is called *via*. The cost of one via edge is usually higher than the one of a jog which is in turn higher than the cost of an edge in preferred direction.

Recently, Neuwohner and Traub implemented a new track assignment algorithm in BonnRoute which is based on a dynamic programming approach. The track assignment tries to compute a preliminary wiring of each net by assigning wires to tracks on each routing layer. We take the wires which have been successfully assigned to tracks by their method as input for our reservations. To increase the efficiency, we filter the assigned tracks by imposing the following conditions:

- We only use the assigned tracks which can legally be placed on the chip with respect to a number of diff-net design rules.
- Furthermore, x- and y-dimensional assigned tracks are only reserved, if their direction agrees with the preferred direction on their layer and their length is longer than ten times the minimal distance between two routing tracks on their routing layer.
- An assigned track in z-dimension, i. e. a sequence of vias, is only reserved, if both its endpoints are part of already saved reservations in x- or y-direction.

As presented in Section 2.3, our technique of basing the detailed routing on the track assignment has two components. On the one hand side, we reserve each reservation which fulfills the above conditions during the path search of all nets different from the net the track belongs to. In BonnRoute there exists a data structure of a grid, which handles queries of the path search algorithm whether a specific vertex is available or not for the routing of one net. Before the routing of any net, we insert each reservation into this grid so that their space is reserved during the path searches of nets a reservation does not belong to.

On the other hand during each path search of a net, we discount the cost of the edges on the reservations belonging to this net by a reduction factor q. In order to compute the cost of an edge efficiently in this case, we initialize a data structure of multiple weighted interval trees which store the information on the location of the current reservations before each path search. Then, we can reduce the cost of an edge in Line 8 of Algorithm 1, if both its endpoints lie on the same reservation and the edge is in the preferred direction of the routing layer.

Furthermore, we call the preprocessing Algorithms 2 and 3 before each path search of a net for which a set of reservation has been saved. These algorithms compute a set of gate vertices and their distances to the target in the grid graph G. This information enables us to compute the values of the future cost function defined in Section 3.2 during the execution of the path search algorithm using Algorithm 4.

Optionally, one can additionally activate the (grouped) multilabel future cost framework described in Section 4. We have implemented both as an extension in the existing multilabel framework in BonnRoute as described in [AGK+15]. Unfortunately, an extensive number of new label types stemming from the multilabel future cost framework easily leads to worse run times. To avoid this effect, we only introduce label types for reservations or groups which are longer than 60 µm. All other reservations are used for discounting the cost of the edges lying on them but do not induce a change of the label type when a path enters or leaves them.

6.2 Experimental Results

To assess the performance of the presented algorithms in practice, we apply them to a testbed consisting of 10 designs. These designs are real world 14 nm instances provided by our industry partner IBM. Design E is of particular nature since it is an integration top level instance. Hence, its routable area is very large and most nets are very long.

The track assignment aims at improving the routing on large instances which are hard to route conventionally. Therefore, the size of the instances in our testbed ranges from 105,645 nets on Design A to 456,862 nets on Design J and thus contains only the largest available designs in Bonn. The designs have between 6 and 14 routing layers.

All tests were performed on Intel Xeon 5690 3.47 GHz machines using 12 threads. We compare the performance of the existing BonnRoute (BR) without reservations against the track assignment based detailed routing approach using the future cost function with reservations l_R (FCR) described in Section 3.2. Additionally, we investigate the effect of enabling the multilabel future cost framework (MFC) and the grouped future cost framework (GMFC) on the run time of the path search algorithm.

We evaluate the resulting routings according to the following criteria:

Netlength: Total length of the placed wires during the routing for each net.

Vias: Number of placed vias.

Scenics: Number of nets whose netlength is at least 25 µm and exceeds the length of an approximation of a shortest possible route by at least 25%.

Shorts: Number of times two wires of different nets are electrically connected. This is often caused by the fact that BonnRoute places so-called guide

wires which do not obey the usual distance rules between wires in the case no legal connection between the source and target pin was found by the path search algorithm.

Errors: Number of violations of the design rules the routing needs to obey. There are both rules applying to wires of the same net (same-net-rules) as to wires of different nets (diff-net-rules).

To compare the efficiency of the different methods we list both the time spent during detailed routing and the number of vertices which have been labeled permanently over the course of all path searches.

During detailed routing, one tries to avoid packing the wires to densely on the chip since this can lead to poor results in terms of timing and electrical noise. Unfortunately, the track assignment algorithm is not yet able to spread the wires reliably. In order to obtain comparable results with BonnRoute, we have disabled the spreading of the wires for the layers we are using the output of the track assignment both in BonnRoute as well as in our track assignment based detailed routing approach

Table 2 shows that using FCR leads to a slightly increased netlength (+0.2%), a reduced number of vias (-0.9%), and a slightly increased number of scenics (+2%) in total. The slightly increased netlength is consistent on all tested designs. For the number of vias, the FCR method works particularly well on Designs C, H, and J (-1.7%, -1.2%, and -2.8% respectively) and worse on the top level Design E (+0.8%). The number of shorts decreases (-26.3%) and the number of errors increases (+19.2%). This trend is not consistent across all tested designs, since on the Designs C, D, E, I, and J we observe a drastic worsening in terms of errors whereas on Designs A, F, G the number of errors significantly decreases. For the number of shorts, we obtain significantly worse results on Designs C and D and better results on Designs A, E, F, and G.

Table 3 shows the effect of different choices of reduction factors q and routing layers from which we reserve assigned tracks. We call this layer *starting layer* in the following. Whereas the total netlength only slightly differs across the different FCR routings, the number of vias increases steadily the higher the starting layer is chosen. Furthermore, we note that the number of errors of the routings with reduction factors 0.25, 0.5, or starting layer 3 is increased compared to all other routings with the FCR method. A special feature of layer 3 is the presence of side pins on multiple designs. The access wires of these pins have to satisfy a

number of design rules which the track assignment often violates when accessing these particular pins. This could partially explain the increase of errors of the routing with starting layer 3. Under these considerations, we have decided to use the reduction factor 0.75 and starting layer 4 for all other routings with our FCR method. When the track assignment algorithm can correctly handle the particular pins on layer 3, it could be desirable to use starting layer 3 with reduction factor 0.75 in the future.

Now we focus on the efficiency of the proposed future cost reservations methods. In Table 4 we compare the run time of BonnRoute with the multilabel future cost method which is currently our best version in terms of the run time. Table 4 shows that the FCR method needs less run time (-17.2%) and less permanent labels (-41.7%) than BonnRoute in total. This trend is consistent on all designs (only on the smallest Design A, MFCR is slightly slower (+1.3%)). The accumulated run time for all nets of the preprocessing methods Algorithms 2 and 3 ranges from 8 seconds on Design A to 1.2 minutes on Design J and can therefore be neglected compared to the total detailed routing run time. The run time of the track assignment algorithm is not included in the detailed routing timings. It ranges from 29.2 minutes on Design A to 2.8 hours on Design E. The track assignment algorithm does not run in parallel yet and therefore bears potential for future speed-ups.

Next, we investigate the efficiency of the multilabel future cost framework (MFC) and the grouped multilabel future cost framework (GMFC) in comparison with the future cost with reservations method (FCR) presented in Section 3. Table 5 shows that compared to FCR the methods MFC and GMFC save run time (-2.9% and -2.3% respectively) and save even more permanent labels (-10.6% and -15.0% respectively). The reduced number of permanent labels is consistent across all designs. The multilabel future cost methods work particularly well on the toplevel Design E where MFC and GMFC save 7.6% and 10.4% run time respectively. On the Designs A and D the MFC method is significantly slower than the FCR method (+15.4% and +15.5% respectively). The increased run time could be related to the problems of the MFC method discussed in Section 4.3. In particular, the additional number of label types could slow down the path search even though the total number of permanent labels consistently decreases with the MFC method. A reason for that could be that multilabel path searches are slower than usual path searches in general since one needs to compute new label types for the neighboring vertices at each step. On the Designs A, F, H, I, and J the GMFC method is slower

than the FCR method and the MFC method. This effect can be related to the fact that the grouped multilabel future cost method performs more multilabel path searches on these designs than the MFC method since the groups are more often longer than the cutoff value of 60 µm compared to the single reservations. One could perform further experiments to find an optimal cutoff value such that the GMFC method not only saves permanent labels but also run time. The additional execution of the preprocessing Algorithm 6 to compute short-cut free groupings for the GMFC method has no measurable effect on the run time.

Lastly, Table 6 compares the length and the usage of the reservations with the total netlength. On all designs except for Design E and J the proportion of wires which are reserved but not used is approximately 5% of the total netlength. On the toplevel Design E this difference is only 1.8% whereas on Design J this difference amounts to 9.8% of the total netlength. Additionally, Table 6 compares the total netlength with the netlength from layer 4 on (the layer from which we use the output of the track assignment) and the length of assigned wires which can not be placed legally during detailed routing. On all instances except for Designs C and E the illegally assigned wires make up between 4.8% and 11.4% of the total netlength. The toplevel Design E again behaves differently since here 55.0% of the assigned wires are illegal during detailed routing. The greatest part of these wires can not be placed since they have not been placed on the correct track patterns of the chips. If the track assignment would be able to apply the correct track patterns for all types of wires on all layers of the chip in the future, the performance of all track assignment based detailed routing methods could be improved in terms of routing quality and run time.

Design CA (mm×mm)	Layers	Nets (K)	Version	Netlength (m)	Vias (K)	Scenics	Shorts	Errors
A 0.4×0.2	6	106	BR	1.2743	855	529	247	983
			FCR	1.2777 (+0.3%)	855 (+0.1%)	534 (+0.9%)	186 (-24.7%)	810 (-17.6%)
B 0.4×0.4	7	139	BR	2.3560	1,129	275	23	751
			FCR	2.3628(+0.3%)	1,130 (+0.2%)	294 (+6.9%)	25 (+8.7%)	693 (-7.7%)
C 0.9×0.3	12	142	BR	3.7808	1,445	259	56	2,673
			FCR	3.7909 (+0.3%)	1,421 (-1.7%)	328 (+26.6%)	73 (+30.4%)	4,751 (+77.7%)
D 0.4×0.3	7	186	BR	3.0635	1,518	1,306	8	623
			FCR	3.0695 (+0.2%)	1,506 (-0.8%)	1,253(-4.1%)	79 (+887.5%)	914 (+46.7%)
E 2.0×4.1	14	195	BR	36.5969	2,413	107	959	18,165
			FCR	36.6534 (+0.2%)	2,432 (+0.8%)	141 (+31.8%)	570 (-40.6%)	20,647 (+13.7%)
F 0.6×0.7	6	216	BR	3.7997	1,645	797	160	905
			FCR	3.8091 (+0.2%)	1,648 (+0.2%)	813(+2%)	70 (-56.3%)	748 (-17.3%)
G 0.5×0.7	9	333	BR	6.2316	2,749	438	463	2,050
			FCR	6.2470 (+0.2%)	2,740 (-0.3%)	456 (+4.1%)	160 (-65.4%)	1,833 (-10.6%)
H 0.6×0.7	8	367	BR	9.9307	3,402	1,105	222	1,960
			FCR	9.9462 (+0.2%)	3,360 (-1.2%)	1,034 (-6.4%)	230 (+3.6%)	2,062 (+5.2%)
I 0.7×1.2	12	442	BR	10.4371	3,873	488	563	3,795
			FCR	10.4655(+0.3%)	3,848 (-0.7%)	537 (+10%)	552 (-2 %)	4,238 (+11.7%)
J 0.5×0.8	12	457	BR	9.7017	4,246	1,433	185	4,292
			FCR	9.7158 (+0.1%)	4,127 (-2.8%)	1,485 (+3.6%)	183 (-1.1%)	6,447 (+50.2%)
Sum		2,583	BR	87.1723	23,275	6,737	2,886	36,197
			FCR	87.3379 (+0.2%)	23,067 (-0.9%)	6,875 (+2%)	2,128 (-26.3%)	43,143 (+19.2%)

Table 2: Comparison of the routing quality between BonnRoute (BR) and the future cost with reservations (FCR) method using 0.75 as reduction factor q and reserving the assigned tracks from layer 4. CA denotes the Chip Area of the design. The column layers gives the number of routing layers of the design.

Version	q	Layer	Netlength (m)	Vias (K)	Scenics	Shorts	Errors
BR	–	–	87.1723	23,275	6,737	2,886	36,197
FCR	0.75	3	87.4526 (+0.3%)	22,822 (-1.9%)	7,154 (+6.2%)	2,381 (-17.5%)	44,166 (+22%)
FCR	0.75	4	87.3379 (+0.2%)	23,067 (-0.9%)	6,875 (+2%)	2,128 (-26.3%)	43,143 (+19.2%)
FCR	0.75	5	87.2922 (+0.1%)	23,257 (-0.1%)	6,652 (-1.3%)	2,184 (-24.3%)	43,403 (+19.9%)
FCR	0.75	6	87.3548 (+0.2%)	23,534 (+1.1%)	6,983 (+3.7%)	2,151 (-25.5%)	40,467 (+11.8%)
FCR	0.25	4	87.4429 (+0.3%)	23,158 (-0.5%)	7,287 (+8.2%)	1,910 (-33.8%)	46,185 (+27.6%)
FCR	0.5	4	87.3997 (+0.3%)	23,118 (-0.7%)	7,107 (+5.5%)	1,945 (-32.6%)	44,526 (+23%)

Table 3: Comparison of the routing quality between BonnRoute (BR) and the future cost with reservations (FCR) method depending on the reduction factor q and the layer from which the output of the track assignment is used. Only the summed values across the entire testbed are displayed.

Design	Version	Time (hh:mm:ss)	Change	Labels (M)	Change
A	BR	38:44		924	
	MFCR	39:14	+1.3%	599	-35.1%
B	BR	25:20		2,001	
	MFCR	23:27	-7.4%	1,279	-36.1%
C	BR	1:50:35		5,611	
	MFCR	1:45:38	-4.5%	3,335	-40.6%
D	BR	39:55		3,101	
	MFCR	34:51	-12.7%	1,707	-44.9%
E	BR	11:18:56		59,909	
	MFCR	9:29:16	-16.2%	36,281	-39.4%
F	BR	51:40		4,061	
	MFCR	40:41	-21.3%	2,825	-30.4%
G	BR	1:22:15		5,291	
	MFCR	1:09:55	-15.0%	2,878	-45.6%
H	BR	2:55:24		12,209	
	MFCR	2:17:42	-21.5%	7,054	-42.2%
I	BR	4:18:19		12,158	
	MFCR	3:31:56	-18.0%	6,626	-45.5%
J	BR	5:53:39		12,159	
	MFCR	4:30:14	-23.6%	5,905	-51.4%
Sum	BR	30:14:47		117,424	
	MFCR	25:02:54	-17.2%	68,490	-41.7%

Table 4: Comparison of the run time and permanent labels between BonnRoute (BR) and the multilabel future cost with reservations method MFCR with reduction factor 0.75 and starting layer 4. Only the detailed routing time is displayed.

Design	Version	Time (hh:mm:ss)	Change	Labels (M)	Change
A	FCR	34:00		594	
	MFC	39:14	+15.4%	599	+0.9%
	GMFC	35:47	+5.2%	590	-0.6%
B	FCR	25:39		1,298	
	MFC	23:27	-8.6%	1,279	-1.5%
	GMFC	24:42	-3.7%	1,262	-2.8%
C	FCR	1:38:06		3,436	
	MFC	1:45:38	+7.7%	3,335	-2.9%
	GMFC	1:37:54	-0.2%	3,227	-6.1%
D	FCR	30:10		1,726	
	MFC	34:51	+15.5%	1,707	-1.1%
	GMFC	29:43	-1.5%	1,734	+0.5%
E	FCR	10:16:11		43,503	
	MFC	9:29:16	-7.6%	36,281	-16.6%
	GMFC	9:12:22	-10.4%	33,375	-23.3%
F	FCR	40:58		2,839	
	MFC	40:41	-0.7%	2,825	-0.5%
	GMFC	41:44	+1.9%	2,785	-1.9%
G	FCR	1:09:48		2,993	
	MFC	1:09:55	+0.2%	2,878	-3.8%
	GMFC	1:07:14	-3.7%	2,949	-1.5%
H	FCR	2:20:59		7,100	
	MFC	2:17:42	-2.3%	7,054	-0.7%
	GMFC	2:30:39	+6.9%	6,993	-1.5%
I	FCR	3:37:33		7,008	
	MFC	3:31:56	-2.6%	6,626	-5.4%
	GMFC	3:39:50	+1.0%	6,413	-8.5%
J	FCR	4:34:53		6,135	
	MFC	4:30:14	-1.7%	5,905	-3.8%
	GMFC	4:53:15	+6.7%	5,818	-5.2%
Sum	FCR	25:48:17		76,633	
	MFC	25:02:54	-2.9%	68,490	-10.6%
	GMFC	25:13:10	-2.3%	65,147	-15.0%

Table 5: Comparison of the run time and permanent labels between FCR, the multilabel frameworks MFC and the grouped multilabel framework GMFC. Again only the detailed routing time is displayed.

Design	Netlength	Netlength	Illegal wires	Reservations	
		from Layer 4	from TA	Total Length	Used Length
	(m)	(%)	(%)	(%)	(%)
A	1.2777	37.0	4.8	32.8	27.8
B	2.3628	49.5	6.7	42.6	37.2
C	3.7909	66.7	18.6	48.6	42.3
D	3.0695	49.0	4.9	44.5	39.7
E	36.6534	96.6	55.0	41.5	39.7
F	3.8091	52.6	7.3	45.3	40.8
G	6.2470	53.3	5.4	48.4	43.2
H	9.9462	55.7	4.1	51.4	47.4
I	10.4655	60.3	10.9	49.8	43.4
J	9.7158	58.4	11.4	47.1	37.3
Total	87.3376	73.2	28.1	45.2	40.9

Table 6: Comparison of the total netlength with the netlength from layer 4 on and the length of wires returned by the track assignment which are illegal. Additionally, the last two columns compare the total netlength with the length of the reservations, and the length of the reservations which have been used during detailed routing.

7 Conclusion

7.1 Summary

In this thesis, we have developed a framework of reservations and discounted edge cost to integrate the track assignment into the detailed routing process. This has enhanced the result of detailed routing on our testbed of 10 large designs by reducing the number of vias (-0.9%) and the number of shorts (-26.3%) in total compared to the previous BonnRoute version. On the other hand, the netlength and the number of design rule violations increases in total (+0.2% and +19.2% respectively). We have described a speed-up technique based on a multilabel future cost function with reservations, which reduces the run time (-17.2 %) in total compared to the previous BonnRoute version. The multilabel future cost framework has also enabled us to prove upper bounds on the number of vertices permanently labeled during the path search.

7.2 Future Work

With our track assignment based detailed routing approach any enhanced version of the track assignment directly leads to better results of the detailed routing. On the one hand side, one could incorporate a global optimization of the wires with respect to electrical coupling and noise which was not possible previously with the conventional detailed routing method. On the other hand, the output of the track assignment could be enhanced by respecting the track patterns of the detailed routing and spreading the wires across the chip. In this way, one could potentially increase the number of assigned wires which can be used during detailed routing and subsequently reduce the number of design rule violations in the resulting routing.

Furthermore, the grouped multilabel future cost approach could be refined by also being able to group reservations connected by stacked vias or long jogs. One possible solution is briefly sketched in Remark 4.15.

A different application of our framework of reservations with discounted edge cost are the so-called *engineering change orders (ECOs)*. In this scenario, the design of a chip is slightly changed after a routing of the previous version has been computed. Then, one tries to recompute a routing for the nets affected by the change which does not differ greatly from the previous routing. In this case, one can again use the old routing as reservations with discounted costs during the routing of the new design. Using this method we hope to efficiently recompute a routing similar to the old one.

References

[AGK+15] M. Ahrens, M. Gester, N. Klewinghaus, D. Müller, S. Peyer, C. Schulte, and G. Téllez, *Detailed routing algorithms for advanced technology nodes*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **34** (2015), no. 4, 563–576.

[BSNZ02] S. Batterywala, N. Shenoy, W. Nicholls, and H. Zhou, *Track assignment: A desirable intermediate step between global routing and detailed routing*, Proceedings of the 2002 IEEE/ACM International Conference on Computer-Aided Design (New York, NY, USA), ICCAD '02, ACM, 2002, pp. 59–66.

[CCHL05] Y. Chang, S. Chen, T. Ho, and D. Lee, *Crosstalk- and performance-driven multilevel full-chip routing*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **24** (2005), no. 6, 869–878.

[Dij59] E. W. Dijkstra, *A note on two problems in connexion with graphs*, Numer. Math. **1** (1959), 269–271. MR 0107609

[FT87] M. L. Fredman and R. E. Tarjan, *Fibonacci heaps and their uses in improved network optimization algorithms*, J. Assoc. Comput. Mach. **34** (1987), no. 3, 596–615. MR 904195

[GH05] A. V. Goldberg and C. Harrelson, *Computing the shortest path: A^* search meets graph theory*, Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, 2005, pp. 156–165. MR 2298261

[GMN+13] M. Gester, D. Müller, T. Nieberg, C. Panten, C. Schulte, and J. Vygen, *BonnRoute: Algorithms and data structures for fast and good VLSI routing*, ACM Trans. Des. Autom. Electron. Syst. **18** (2013), no. 2, 32:1–32:24.

[Hen16] D. Henke, *Pfadsuche im Detailed Routing*, Rheinische Friedrich-Wilhelms-Universität Bonn, Bachelorarbeit (2016).

[HNR68] P. E. Hart, N. J. Nilsson, and B. Raphael, *A formal basis for the heuristic determination of minimum cost paths*, IEEE Transactions on Systems Science and Cybernetics **4** (1968), no. 2, 100–107.

[Hu12] J. Hu, *Track assignment considering crosstalk-induced performance degradation*, Proceedings of the 2012 IEEE 30th International Conference on Computer Design (ICCD 2012) (Washington, DC, USA), ICCD '12, IEEE Computer Society, 2012, pp. 506–507.

[KRV07] B. Korte, D. Rautenbach, and J. Vygen, *BonnTools: Mathematical innovation for layout and timing closure of systems on a chip*, Proceedings of the IEEE **95** (2007), no. 3, 555–572.

[KV12] B. Korte and J. Vygen, *Combinatorial optimization*, fifth ed., Algorithms and Combinatorics, vol. 21, Springer, Heidelberg, 2012, Theory and algorithms. MR 2850465

[PRV09] S. Peyer, D. Rautenbach, and J. Vygen, *A generalization of Dijkstra's shortest path algorithm with applications to VLSI routing*, J. Discrete Algorithms **7** (2009), no. 4, 377–390. MR 2558995

[Rub74] F. Rubin, *The Lee path connection algorithm*, IEEE Trans. Comput. **23** (1974), no. 9, 907–914.

[Vyg15] J. Vygen, *Chip design – theory and practice of VLSI layout*, Rheinische Friedrich-Wilhelms-Universität Bonn, Lecture Notes (2015).

[WW07] D. Wagner and T. Willhalm, *Speed-up techniques for shortest-path computations*, pp. 23–36, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.