Karlsruhe Institute of Technology

Route Planning with Temporary Road Closures

Master Thesis of

Christian Bräuer

At the Department of Informatics
Institute of Theoretical Computer Science

Reviewers:	Prof. Dr. Dorothea Wagner
	Prof. Dr. Peter Sanders
Advisors:	Valentin Buchhold
	Alexander Kleff
	Dr. Frank Schulz
	Tim Zeitz

Time Period: 1st June 2018 – 30th November 2018

Statement of Authorship

I hereby declare that this document has been composed by myself and describes my own work, unless otherwise acknowledged in the text.

Karlsruhe, November 30, 2018

Abstract

We study the problem of route planning with temporary road-closures. In practice, road segments, e.g. in residential or downtown areas, are closed at certain times. In addition, in many countries general weekend and night driving bans apply. Mainly trucks are affected by these driving bans. The road-closures may inflict waiting time along a route for which parking lots are needed. Routes with mathematical earliest arrival at the destination may contain loops and other detours. In this thesis we define a model which yields reasonable routes in practice, where the road-closures are taken into account. We distinguish between the travel time and the driving time of a route. The travel time is the time span between the start and the end of the route. The driving time is the duration in which the driver moves his vehicle. The distinction between travel time and driving time may yield multiple routes per route planning query. Thus, we use the concept of Pareto-optimality. We present an algorithm which calculates Pareto-optimal routes according to the model and adapt known speed up techniques to the algorithm. Using heuristic algorithms, we achieve a running time per query of a few seconds.

Deutsche Zusammenfassung

Wir untersuchen das Problem der Routenplanung mit temporären Straßensperrungen. In der Praxis sind Straßensegmente, zum Beispiel in Wohn- oder Innenstadtgebieten, zu gewissen Uhrzeiten gesperrt. Zusätzlich gelten in vielen Ländern generelle Wochenend- oder Nachtfahrverbote, die vor allem Lkw betreffen. Diese Sperrungen führen dazu, dass auf manchen Routen eine Wartezeit eingelegt werden muss, für die allerdings Parkplätze benötigt werden. Routen mit mathematisch frühester Ankunftszeit am Ziel führen unter Umständen zu Routen, die Kreise und andere Umwege enthalten. In dieser Arbeit definieren wir ein Modell, das in der Praxis sinnvolle Routen zulässt, die diese Einschränkungen berücksichtigen. Wir unterscheiden zwischen der Reisezeit und der Fahrzeit einer Route. Die Reisezeit beschreibt die Zeit zwischen dem Beginn und dem Ende der Route. Die Fahrzeit beschreibt die Dauer, zu der sich der Fahrer in seinem Fahrzeug fortbewegt. Aufgrund des Unterschiedes zwischen Reisezeit und Fahrzeit kommen mehrere Routen als Antwort auf eine Routenplanungsanfrage in Betracht. Dazu nutzen wir das Konzept der Pareto-Optimalität. Wir präsentieren einen Algorithmus, der die Pareto-optimalen Routen passend zum Modell berechnet und passen bekannte Beschleunigungstechniken an den Algorithmus an. Zusammen mit heuristischen Algorithmen erreichen wir so eine Laufzeit pro Routenplanungsanfrage von wenigen Sekunden.

Contents

1. Introduction

A standard route planning algorithm uses road graphs with edges, which are always traversable. In practice, it is preferable to take additional information into account. This thesis focuses on route planning with temporary road-closures. In road networks, certain roads are closed regularly. Mostly trucks are affected by these closures, but some closures also apply to cars. Temporary road-closures can affect areas of different size: In some cities, the downtown areas are closed for trucks at night. Sometimes only certain streets or street segments are restricted. In Hamburg, Germany, there is another special example: Driving into the city with any kind of vehicle on the Sierichstraße is only allowed from 04.00 until 12.00 every day, and leaving the city in the other direction is only allowed from 12.00 to 04.00.

In many European countries, there are driving bans for trucks on weekends, e.g. in Germany from 00.00 until 22.00 on Sundays. In Switzerland and Austria, driving a truck is even forbidden from 22.00 until 05.00 every day. For an overview of the driving bans of some European countries, see Chapter 5.1.

A temporary closed road may inflict waiting time on a route or may delay the departure at the start location of the route. In general, it is not possible for a driver to wait anywhere on the road, but suitable locations are needed. Especially for trucks, large parking lots are required. Waiting times on a route delay the arrival at the destination and a detour around a restricted road segment or even around a whole country where a driving ban applies might be of advantage. However, avoiding a restricted segment or a country increases the driving time along the route. Also, if there is no parking lot nearby where the driver could wait, driving in a cycle until a closed road on the route opens can avoid another greater detour. While this may enable an earlier arrival time at the destination, a driver might not expect this result.

This thesis deals with route planning with temporary road-closures in practice. A driver may only wait at parking lots or at the start location of the route. We define properties of routes, which avoid the unexpected results from above. A route planning query may yield multiple routes, where the arrival times and driving times of each two routes differ.

1.1 Related Work

The standard routing problem is a well-researched topic. After Dijkstra's algorithm [Dij59] was published in 1959, many speed up techniques were developed. Many of these are in

turn based on Dijkstra's algorithm. In this section we focus on speed up techniques relevant for this work. For a broad overview of other speed up techniques, we refer to [BDG+15a].

An important approach for speeding up route planning queries are Contraction Hierarchies [GSSD08]. In the following, we provide a brief overview: For a given road-network, multiple queries are issued. For a Contraction Hierarchy, we invest time in a preprocessing step, to speed up the queries. The preprocessing step and the querying is based on the main idea that a road network is hierarchically ordered. A highway is much more important for long-distance-queries than city streets. Therefore, in the preprocessing step, each vertex is mapped to an "importance"-value and the vertices are removed in increasing order of their importance. To ensure correct results of a route planning query, additional shortcut edges are inserted which preserve the minimum distances between all remaining vertices. After the preprocessing finished, a query is answered by searching from the start vertex and the destination vertex simultaneously to vertices of higher importance.

For routing in graphs with time-dependent edge weights, an extension for Dijkstra's algorithm was presented by Dreyfus et al. [Dre69]. Again, speed up techniques were developed, such as time-dependent Contraction Hierarchies [BGSV13] which use a sophisticated preprocessing and querying phase. For many route planning algorithms, it is assumed that entering an edge later never yields an earlier arrival at the other vertex of the edge. This is called first-in-first-out (fifo)-property. In our case, a detour must be taken, if a driver arrives at a road-closure which is closed and opens up later. Therefore, and as the driver is not allowed to wait at every vertex, the edges of our graphs do not fulfill the fifo-property. There was very little research done for route planning on non-fifo networks, but for other networks, e.g. computer networks, this problem is more common. In general, finding the shortest path in non-fifo networks is known to be $\mathcal{NP}$-hard [OR89]. Orda et al. [OR91] presented an algorithm, which finds shortest paths in non-fifo networks.

A different routing problem with special requirements is electric vehicle routing with charging stations. This problem is partly related to our problem. It depends on the state of charge of the electric vehicle battery whether an edge is traversable. By spending time at a charging station, a previously non-traversable edge may become traversable. Baum et al. [BDG+15b] presented an algorithm which solves this problem. During the work for the thesis, some of their ideas were adapted for our algorithms.

Another related problem is minimal on-road time route scheduling on time-dependent graphs [LHDZ17]. The authors study the problem of route scheduling, where the driving time is minimized on a network with time-dependent edge weights. Taking waiting time is only allowed at parking lots. Their algorithms do not require edge weights fulfilling the fifo-property. They also claim that their algorithms have polynomial running time. However, the authors only evaluated their algorithms on small graphs with random edge weights and random parking lot locations.

1.2 Contribution

In this thesis we address the problem of road networks with constant edge-weights, time-dependent road-closures and parking lots as sole locations for waiting times apart from the start location. The temporary road-closures with a restricted waiting policy, where waiting is only allowed at parking lots, yield the non-fifo property in the edges of the road-graph. We show that finding a valid solution, which takes the road-closures into account, is $\mathcal{NP}$-hard. Even if we find a solution with minimum travel time, a user may not be satisfied with it, as e.g. loops may occur. Therefore, we define a model and a corresponding algorithm, which yields reasonable routes in practice. We adapt speed up techniques to our algorithm and additionally, we propose heuristic approaches. The heuristic approaches enable solving the

problem on large road networks. Finally, we evaluate the performance of our algorithms and compare the quality of the results of the heuristics to the optimal results.

1.3 Outline

Chapter 2 establishes notation used in the thesis and recapitulates Pareto-optimality. We introduce Dijkstra's algorithm and a multicriteria variant, which is used as basis for our algorithms. In Chapter 3, we define three models, which state slightly different problems. For each model we examine important properties and present algorithms, which can be used to obtain the results of a route planning query. In Chapter 4 we show how to use Contraction Hierarchies for our algorithms and present additional performance optimizations. The previously presented algorithms are evaluated in Chapter 5. We describe the test setup and the road data, and analyze the running time of the algorithms and heuristics. The quality of the results are compared to each other. In Chapter 6 we summarize our results and provide an outlook for future work on the topic of non-fifo route planning in practice.

2. Preliminaries

In this chapter we introduce our notation and basic definitions for the thesis. In Section 2.1 we define graphs with edge-guards and routes in graphs with edge-guards. In Section 2.2 we recapitulate basic information about Pareto-sets and Pareto-optimality and in Section 2.3 we show Dijkstra's algorithm and a multicriteria variant on which our algorithms build upon.

2.1 Notation and basic definitions

In this thesis, we call $G = (V, E, c, I, P)$ a graph with edge-guards, where V are the graph's vertices and $E \subseteq V \times V$ are the directed edges of the graph G. The function $c\colon E \to \mathbb{N}_+$ is the cost function. For example, in the context of road networks it yields the driving time. The function I is the traversal function. It maps each edge in E either to the set containing all non-negative numbers $[0, \infty)$ or to a finite disjoint union of half-open intervals $[\alpha_1, \omega_1)\dot\cup \ldots \dot\cup [\alpha_k, \omega_k)$ with $\alpha_i < \omega_i < \alpha_{i+1} < \omega_{i+1}$, $\alpha_i, \omega_i, \alpha_{i+1}, \omega_{i+1} \in \mathbb{N}_0$. Each edge (u, v) has a guard. These guards let drivers enter an edge (u, v) only at integer points in time, which are element of $I(u, v)$. A driver, which is traversing an edge may always continue driving, even if the guard closes the edge. For all points in time which are contained in $I(u, v)$, an edge e is called *traversable*. An edge e which is always traversable, i.e. $I(u, v) = [0, \infty)$, is called *unguarded*. All other edges are called *guarded*. The set $P \subseteq V$ is a set of parking vertices. In the road network, the parking vertices represent parking lots where the driver can wait. We also assume, that the driver can also wait at the start vertex of the route.

Figure 2.1 depicts a graph with edge-guards. For example, edge (v, w) is only traversable at points in time in $[10, 15)$ or in $[20, 25)$. At all other points in time, it is not possible to traverse edge (v, w). Note that no parking lots are displayed in the figure.

Given two finite disjoint unions of half-open intervals $I_1 = [\alpha_1^1, \omega_1^1)\dot\cup \ldots \dot\cup [\alpha_k^1, \omega_k^1)$ and $I_2 = [\alpha_1^2, \omega_1^2)\dot\cup \ldots \dot\cup [\alpha_l^2, \omega_l^2)$, we define:

- the shifting of I_1 by $c \in \mathbb{N}_+$: $I_1 - c \coloneqq [\alpha_1^1 - c, \omega_1^1 - c)\dot\cup \ldots \dot\cup [\alpha_n^1 - c, \omega_n^1 - c)$,
- $I_1^* \coloneqq \{[\alpha_1^1, \omega_1^1), \ldots, [\alpha_k^1, \omega_k^1)\}$ the finite set of half-open intervals of I.
- $I_3 = I_1 \cup I_2$ such that I_3 is a finite disjoint union of half-open intervals $I_3 = [\alpha_1^3, \omega_1^3)\dot\cup \ldots \dot\cup [\alpha_k^3, \omega_k^3)$, $\alpha_i < \omega_i < \alpha_{i+1} < \omega_{i+1}$ and $\alpha_i, \omega_i, \alpha_{i+1}, \omega_{i+1} \in \mathbb{N}_0$.
- $I_3 = I_1 \cap I_2$ accordingly.

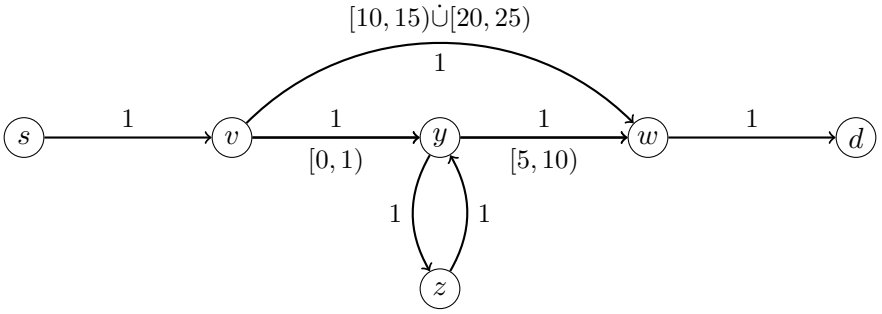


Figure 2.1: Example graph G with edge-guards. The edges weights are the numbers next to the edges. Guarded edges are traversable at points in time which are contained in the intervals in the sets near the edges. Note that no parking lots are displayed in the figure.

In this thesis, the algorithms calculate *routes* in graphs G with edge-guards. A route r is a pair (t_{start}, S) of the start time t_{start} at the first vertex and a sequence S. The sequence $S = [(u_1, t^1_{waiting}), \ldots, (u_n, t^n_{waiting})]$ contains vertex/waiting time-pairs $(u, t_{waiting})$. For two consecutive pairs in S, the vertices in the pairs must be adjacent in G. The driver departs at vertex s at time $t_{start} + t^1_{waiting}$. A non-zero waiting time $t^n_{waiting}$ at the last vertex u_n is never useful. To avoid special case treatment, we assume that the waiting time $t^n_{waiting}$ at the last vertex u_n is always zero.

As an example consider $r_1 = (5, [(s, 0), (v, 5), (w, 0)])$ in the graph with edge-guards in Figure 2.1, where a driver starts at time 5 at vertex u and waits there for 0 time units. Then he moves to v, waits there for 5 time units and then moves to vertex w, where he waits for 0 time units. To faciliate readability, we sometimes drop the waiting time in a sequence if the waiting time is zero. Hence, $r_1 = (0, [s, (v, 5), w])$.

Given a route r, we use $E(r)$ for the multiset of edges of r, that is $E(r) \coloneqq \{(u_i, u_{i+1}) \mid i \in \{1, \ldots, n-1\})\}$. We also define

- $t_{start}(r) \coloneqq t_{start}$ the start time of the route r,
- $t_{waiting}(r) \coloneqq \sum_{i=1}^{n} t^i_{waiting}$ for the accumulated waiting time of the route,
- $t_{driving}(r) \coloneqq \sum_{e \in E(r)} c(e)$ for the accumulated driving time of the route,
- $t_{travel}(r) \coloneqq t_{driving}(r) + t_{waiting}(r)$ for the travel time of the route and
- $t_{arrival}(r) \coloneqq t_{start}(r) + t_{travel}(r)$ for the arrival time at the end of the route at vertex d.

If each edge (u_i, u_{i+1}) is traversable when vertex u_i is departed, then the route is *valid*. In other words, for all $(u_i, u_{i+1}), i \in \{1, \ldots, n-1\}$ we require

$$t_{start}(r) + \sum_{j=1}^{i-1} (t^j_{waiting} + c(u_j, u_{j+1})) + t^i_{waiting} \in I(u_i, u_{i+1}).$$

A route which is not valid, is *invalid*. For a graph $G = (V, E, c, I, P)$ with edge-guards, we define the graph $G' = (V, E, c, I', P)$ with $I'(e) = [0, \infty)$ for each $e \in E$. In G' each edge is unguarded. If r is a shortest s-d-route in G', then r is a shortest *guard-independent* s-d-route in G. A guard-independent route may be invalid.

Again consider route r_1. Route r_1 is valid, as the only guarded edge along r_1 is (v, w) and we traverse (v, w) at time 11, which is when (v, w) is traversable as $11 \in [10, 15)\dot{\cup}[20, 25) = I(v, w)$. Also, r_1 is a shortest guard-independent route. However, route $r_2 = (9, [s, (v, 5), w])$ which starts at time 9 is not valid as edge (v, w) is traversed at time 15 and $15 \notin I(v, w)$. Although r_2 is invalid, it is a shortest guard-independent route.

In the thesis, we use the terms *travel time* and *driving time*. The driving time is used only for the accumulated on-road time of the route, i.e. the time the driver needs to steer his vehicle. The travel time is the time from the start until the end of the route, i.e. the waiting time in addition to the driving time.

During the thesis, we define models, where only valid routes with certain additional properties are allowed. A route with these properties is called *feasible*. As mentioned, the feasibility-definition is model-dependent and changes during the thesis.

In addition, to avoid special case treatment, we define $\min \emptyset = \infty$.

2.2 Pareto-optimality and Pareto-sets

In this section, we recapitulate Pareto-optimality and Pareto-sets. Given two k-tuples $x = (x_1, \ldots, x_k) \in \mathbb{N}_0^k$ and $x' = (x'_1, \ldots, x'_k) \in \mathbb{N}_0^k$, then x *dominates* x' if

1. $\forall i \in \{1, \ldots, k\}\colon x_i \leq x'_i$ and
2. $\exists i \in \{1, \ldots, k\}\colon x_i < x'_i$.

Given a set of k-tuples X, a tuple $x \in X$ is *Pareto-optimal*, if there is no other tuple $x' \in X$ which dominates x. A subset $X' \subseteq X$ is a *Pareto-set* if all Pareto-optimal tuples of X are contained in X'.

As already mentioned, we define feasibility of routes for each model we discuss. Given a s-d-query, there may be multiple valid and feasible s-d-routes in a graph and we minimize the two criteria travel time t_{travel} and driving time $t_{driving}$. Our algorithms calculate all Pareto-optimal pairs of all $(t_{travel}, t_{driving})$-pairs which have a corresponding valid and feasible route.

2.3 Shortest path algorithms

In this section, we show Dijkstra's algorithm [Dij59] and a multi-criteria version of Dijkstra's algorithm. In Algorithm 2.1, Dijkstra's algorithm is depicted. Given a start vertex s the algorithm calculates the guard-independent driving time $t^u_{driving}$ from s to each vertex $u \in V$. The algorithm uses a priority queue which saves all discovered vertices in order of their tentative driving time. The next minimum vertex u is removed from the queue and all outgoing edges (u, v) are relaxed, i.e. the tentative distance is reduced if the driving time of $t^u_{travel} + c(u, v)$ is less than the tentative distance. If the vertex is not already an element of the queue, then it is inserted. The algorithm stops if the queue is empty.

If each edge of the graph not only has scalar cost like the driving time, but a vectorial cost function $c\colon E \to \mathbb{N}_+^k$, then we need a multicriteria version of Dijkstra's algorithm (MCD) so that we can calculate the Pareto-set L_v containing all Pareto-optimal labels $(x^v_1, \ldots, x^v_k)$ for each vertex $v \in V$. The pseudocode of MCD is depicted in Algorithm 2.2. For each vertex v, the MCD saves a set of labels L_v instead of a tentative distance. The RELAX()-function then creates a set of new labels for each adjacent vertex.

For example, consider $k = 2$ and x_1 is the travel time and x_2 is the distance. The priority queue Q sorts a label (x_1, x_2) by lexicographically by travel time first and distance second. The RELAX()-function always returns exactly one new label, which is the component-wise sum of the old labels travel time and distance and the cost of the edge (u, v). Then MCD calculates all Pareto-optimal travel time/distance-pairs for each vertex.

In this example, the algorithm has the label-setting property: If a label l was taken from the queue, other labels at the same vertex can never dominate l. The running time of this algorithm is dependent on the running time of the RELAX()-function and the maximum

Algorithm 2.1: DIJKSTRA'S ALGORITHM

Input: Graph $G = (V, E, c)$, start vertex s
Data: Priority queue Q
Output: Guard-independent travel time t^v_{travel} for all $v \in V$

```
1  foreach v ∈ V do
2      t^v_driving ← ∞
3  Q.INSERT(s, 0)
4  t^s_driving ← 0
5  while Q is not empty do
6      u ← Q.DELETEMIN()
7      foreach (u, v) ∈ E do
8          if t^u_driving + c(u, v) < t^v_driving then
9              t^v_driving ← t^u_driving + c(u, v)
10             if Q.CONTAINS(v) then
11                 Q.DECREASEKEY(v, t^v_driving)
12             else
13                 Q.INSERT(v, t^v_driving)
```

number of labels. In our example, the RELAX()-function has polynomial running time, but the number of labels at a vertex can grow exponentially. Hence, the algorithm has an exponential running time [Han80].

In our use-case, the MCD uses a scalar cost function, but in addition, it uses the traversal function I. We specify

- the components of the labels $(x_1, \ldots, x_k)$
- the priority of the labels in the queue
- the RELAX()-functions
- a custom domination function used in lines 10 and 12

for each model we present. Then the MCD calculates all Pareto-optimal $(t_{travel}, t_{driving})$-labels which have a corresponding valid and feasible route in the model we presented. As a label may contain more information than travel time and driving time, we drop the additional information and remove all non-Pareto-optimal labels. The resulting labels at vertex d are the return value of a s-d-query. If the running time of the RELAX()-function and the domination function is polynomial in terms of the input size and if the maximum number of labels is polynomial, then the algorithm has a polynomial running time.

Algorithm 2.2: Multi-Criteria Dijkstra (MCD)

Input: Graph $G = (V, E, c, I, P)$, start vertex s, initial label $(x_1^s, \ldots, x_k^s)$
Data: Priority queue Q
Output: Labels $(x_1, \ldots, x_k)$ for all $v \in V$

```
1  foreach v ∈ V do
2      L_v ← ∅
3  Q.INSERT(s, (x_1^s, ..., x_k^s))
4  L_s ← (x_1^s, ..., x_k^s)
5  while Q is not empty do
6      (u, (x_1, ..., x_k)) ← Q.DELETEMIN()
7      foreach (u, v) ∈ E do
8          L ← RELAX((x_1, ..., x_k), (u, v), G)
9          foreach l ∈ L do
10             if no l' ∈ L_v dominates l then
11                 Insert l into L_v
12                 Remove labels from L_v which are dominated by l'
13                 Q.INSERT(v, l)
```

3. Models and Algorithms

In this chapter we present models and corresponding algorithms. For each model, we present a definition of feasible routes. The corresponding problem is to find all Pareto-optimal labels of travel time and driving time. In Section 3.1 we begin with the exact model, where valid routes are feasible if the waiting time is only taken at parking lots or at the start vertex. We show that the corresponding problem is $\mathcal{NP}$-hard and we give an algorithm which solves it. We also show that the solutions may not satisfy the expectation of a user. In Section 3.2, we present a model where a feasible solution may contain waiting time at every vertex of the route and give the corresponding algorithm. In this model, feasible routes also may not satisfy the expectation of a user, as the waiting time cannot be taken anywhere on the road. Compared to the exact model, we present an algorithm with polynomial runtime. Also, feasible routes in this model give lower bounds for the solutions in the exact model. As our main result, we present the model of detour-free routes in Section 3.3, where we give a formal definition of routes that are feasible in practice by waiting only at parking lots and at the start vertex. We give a corresponding algorithm which has polynomial runtime.

3.1 Exact model

In this section, we define feasible routes for the exact model, where the routes are the mathematically shortest ones and where waiting is only allowed at the parking lots or at the start vertex. We give examples and prove an upper bound for the cardinality of a resulting Pareto-set size and prove the $\mathcal{NP}$-hardness of the problem. Also, we give an algorithm that calculates the results according to the exact model. Due to the running time and the memory requirements of the algorithm, we did not evaluate this algorithm in the experiment Chapter 5.

Definition 3.1 (Exact feasibility)**.** *Let G be a graph with edge-guards, s the start vertex, d the destination vertex and t the start time. A route is feasible, if it is valid and for all vertices u with a waiting time greater zero, u is a parking lot or the start vertex.*

Consider the example graph in Figure 3.1, the start vertex s and the destination vertex d. Assume that there is no parking lot (and hence waiting is only allowed at vertex s). The driver's start time is at time 0. There are two Pareto-optimal labels:

1. Label $(t_{travel} = 12, t_{driving} = 3)$ with a corresponding route $(0, [(s, 9), v, w, d])$: The driver waits at s for 9. Then he drives to v and arrives there at 10, hence the next

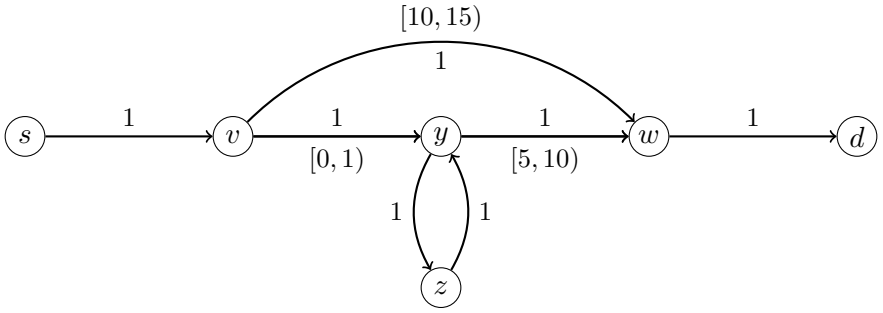


Figure 3.1: Example graph. Each edge has weight 1. Intervals at the edges denote the open intervals.

edge (v, w) is traversable. The driver can continue his route to w and then to d. He arrives at d at time 12, and he has a driving time of 3.

2. Label $(t_{travel} = 8, t_{driving} = 8)$ with a corresponding route $(0, (s, v, y, z, y, z, y, w, d))$. Note that we have omitted the waiting time for better readability, as it is always zero at each vertex. The driver immediately departs at s. Then he drives to v at time 1, hence the next edge (v, y) is traversable. The driver can continue his route to y and arrives there at 2, but the edge (y, w) is not traversable at time 2. The driver now has to drive the cycle y-z-y two times. After the second cycle he arrives at y at time 6, so edge (y, w) is traversable now. The driver now can continue his route to w and then to d. He arrives at d at time 8, which is earlier than when using the route given above. But he needs a driving time of 8 instead of 3.

The second result is not what a user might expect. As a user, we would not want to be routed on a loop. In practice this could for example be a roundabout, where the navigation system suggests to drive around multiple times. Therefore we propose another problem definition in Section 3.3.

Now we assume y is a parking lot. This does not change the first solution, but the second solution can be improved to $(t_{travel} = 7, t_{driving} = 4)$: In the corresponding route, instead of driving the y-z-y-cycle two times, the driver waits for 3 time units after arriving at y at time 2. Then the driver can continue his route and finally arrives at d at time 7 with a driving time of 4.

3.1.1 Cardinality of the Pareto-set

A query yields all Pareto-optimal $(t_{travel}, t_{driving})$-labels. In general a routing query on a graph, where each edge has two unrelated weights like driving time and distance, may yield $\mathcal{O}(2^{|V|})$ Pareto-optimal labels [Han80]. However, in this case both criteria are related, as traversing an edge always increases the travel time just like it increases the driving time. Only when waiting at a vertex, driving time and travel time of a route can deviate. In this case the travel time is increased, but the driving time is not. Therefore, we ask for the cardinality of the resulting Pareto-set. In the remainder of this section, we assume that R is a Pareto-set of labels of a s-d-query with a start time at t_{start} in a graph $G = (V, E, c, I, P)$ with edge-guards. The following theorem gives an upper bound for the cardinality of R:

Theorem 3.2 (Pareto-set cardinality)**.** *Let E_r be the set of guarded edges of E and $W = \sum_{e \in E_r} |I^*(e)|$ the total number of intervals of guarded edges. Then $|R| \leq W + 1$.*

Before we prove this theorem, we prove the following statements.

Lemma 3.3. *There is at most one label $(t_{travel}, t_{driving}) \in R$ with $t_{travel} = t_{driving}$ and the corresponding route of $(t_{travel}, t_{driving})$ has no waiting time.*

Proof (by contradiction). Assume there are two labels $(t^1_{travel}, t^1_{driving}), (t^2_{travel}, t^2_{driving}) \in R$ such that their travel time and driving time is equal. Hence we have $t^1_{travel} = t^1_{driving}$ and $t^2_{travel} = t^2_{driving}$. Without loss of generality let $t^1_{travel} \leq t^2_{travel}$. Then either $t^1_{travel} = t^2_{travel}$ and $t^1_{driving} = t^2_{driving}$, but then both labels are equal which is a contradiction. Or $t^1_{travel} < t^2_{travel}$ and $t^1_{driving} < t^2_{driving}$, but then $(t^1_{travel}, t^1_{driving})$ dominates $(t^2_{travel}, t^2_{driving})$ and hence $(t^2_{travel}, t^2_{driving}) \notin R$, which is a contradiction too. In both cases the waiting time of a route corresponding to $(t^1_{travel}, t^1_{driving})$ is zero as $t^1_{waiting} = t^1_{travel} - t^1_{driving} = 0$. □

Definition 3.4 (Closure-Δ and restrictive edges)**.** *Given a route r. For a* guarded *edge $e = (u, v) \in E(r)$ consider the departure time $t^e_{departure} \in I(e)$ at u. Consider the corresponding interval $[\alpha, \omega) \in I^*(e)$, such that $t^e_{departure} \in [\alpha, \omega)$. Let $\Delta^e = t^e_{departure} - \alpha$ be the difference of the actual traversal of edge e and the opening of edge e. If $\Delta^e = 0$ edge e is called* restrictive edge*. For an* unguarded *edge $e \in E(r)$, let $\Delta^e = \infty$. Let $\Delta \coloneqq \min_{e \in E(r)} \Delta^e$ be the* closure-Δ *of r.*

The closure-Δ of a route is the shortest duration between opening of a guarded edge and its traversal. A restrictive edge is an edge, that is traversed just at the time when it opens up and hence we cannot traverse the edge a time unit earlier. Note that if a route has a restrictive edge, then the closure-Δ is equal to zero. In the following we show that given a route with non-zero waiting time and a closure-$\Delta > 0$, there is always another valid route with reduced waiting time.

Lemma 3.5. *Let r be s-d-route with start time t_{start} in G with a vertex u with waiting time $t^u_{waiting} > 0$ and closure-$\Delta > 0$. Then there exists a valid and feasible s-d-route r' with start time t_{start} in G with the driving time of r and a travel time which is smaller than the travel time of r.*

Proof. Let $r = (t_{start}, [(s, t^s_{waiting}), \ldots, (u, t^u_{waiting}), \ldots, (d, 0)])$ and $\delta = \min\{\Delta, t^u_{waiting}\}$. Consider route $r' = (t_{start}, [(s, t^s_{waiting}), \ldots, (u, t^u_{waiting} - \delta), \ldots, (d, 0)])$ where the waiting time at u was reduced by δ.

We show that r' is valid and feasible in the exact model: Now consider the edges $E(r')$. We show that they are traversable at the time they are traversed in r'. We only need to check the guarded edges $e \in E(r')$, after u was departed as the departure time of the vertices before u were unchanged. Let the departure time at v in r be $t^v_{departure}$, which is in a time-window $[\alpha, \omega) \in I^*(e)$. In r' we depart from v at time $t^v_{departure} - \delta$, and we know that $t^v_{departure} - \delta < t^e_{departure} < \omega$, hence in r' we traverse e before the time-window $[\alpha, \omega]$ closes. Since $\delta \leq \Delta = \min_{f \in E(r)} \Delta^f \leq \Delta^e = t^v_{departure} - \alpha$, we know that the duration by which we reduced the waiting time at u is less than the amount of time between the opening of e at α and the traversal of e in r. Therefore, $t^v_{departure} - \delta \geq t^v_{departure} - t^v_{departure} + \alpha = \alpha$ and in r' we traverse e after the time-window $[\alpha, \omega]$ opens and hence, r' is valid. Since we did not add waiting time in r', every vertex with a non-zero waiting time is a parking lot, otherwise r was not feasible. Hence, r' is feasible.

The driving time of r' is equal to the driving time of r as no edges where changed. Since we reduced the waiting time in r' by δ compared to the waiting time of r, the travel time of r' is $t_{travel}(r') = t_{travel}(r) - \delta$. Hence, r' has a travel time less than the travel time of r. □

With this lemma, we can show that a Pareto-optimal route always has zero waiting time or a restrictive edge (Corollary 3.6) and if there is a restrictive edge in a Pareto-optimal route, then after the last restrictive edge in the route, there is no further waiting time (Corollary 3.7).

Corollary 3.6. *For all labels L in R: If a route r corresponding to the label L has zero waiting time then r has a restrictive edge, i.e. there is an edge that prevents r from having a reduced waiting time.*

Proof (by contradiction). Assume there is a label $(t_{travel}, t_{driving}) \in R$ and a corresponding route r such that $t_{waiting}(r) > 0$ and there is no restrictive edge. Hence, r has a non-zero closure-Δ. By Lemma 3.5, there exists a s-d-route r' which has equal travel time and reduced driving time. Therefore, there exists a label L' which dominates L, hence $(t_{travel}, t_{driving}) \notin R$, which contradicts our assumption. □

Corollary 3.7. *For each label in R: If a route r corresponding to the label has a restrictive edge, then after the last restrictive edge of r there is no vertex u with non-zero waiting time.*

Proof (by contradiction). Assume there is a label in R with a corresponding route $r = (t_{start}, [(s, t^s_{waiting}), (u, t^u_{waiting}), \ldots, (d, 0)]) \in R$ such that for each restrictive edge e, there is a vertex u after e in r with waiting time $t^u_{waiting} > 0$. Consider the last vertex u with non-zero waiting time and let the arrival time at u be t'_{start}. Then consider subroute r^* of r starting at u and ending at d: $r^* = (t'_{start}, [(u, t^u_{waiting}), \ldots, (d, 0)])$ and its closure-Δ which is non-zero since there is no restrictive edge in r^*. By Lemma 3.5, we know that there exists a u-d-route $r' = (t'_{start}, [(u, t'^u_{waiting}), \ldots, (d, 0)])$, with less travel time than r^* and a driving time equal to the driving time of r^*.

Now consider route $r'' = (t_{start}, [(s, t^s_{waiting}), \ldots, (u, t'^u_{waiting}), \ldots, (d, 0)])$, where we replaced the u-d-subroute in r by r'. The route r'' is valid and feasible as r' is valid and feasible and the arrival at u is as in r and r'. Since the driving time of r'' is equal to the driving time of r and the travel time of r'' is less than the travel time of r, $(t_{travel}(r''), t_{driving}(r''))$ dominates $(t_{travel}, t_{driving})$ and hence $(t_{travel}, t_{driving}) \notin R$, which contradicts our assumption. □

Now we prove Theorem 3.2:

Proof of Theorem 3.2 (by contradiction). Assume $|R| > W + 1$. By Lemma 3.3 there is at most one tuple in R such that its corresponding routes have no waiting time. Hence, by Corollary 3.6 there are at least $W + 1$ tuples in R such that their corresponding routes have a non-zero waiting time and a restrictive edge. For each of these routes, consider the last restrictive edge. By pigeonhole principle, there are at least two tuples $(t^1_{travel}, t^1_{driving}) \neq (t^2_{travel}, t^2_{driving})$ with their corresponding routes r_1 and r_2 in R, such and r_1 and r_2 share the same interval $[\alpha, \omega]$ of the same last restrictive edge $e = (u, v)$. Since e is a restrictive edge, in both routes we depart from u at α and arrive at v at $\alpha + c(e) =: t'_{start}$. Consider the v-d-subroutes $r'_1 = (t'_{start}, [(v, 0), \ldots, (d, 0)])$ and $r'_2 = (t'_{start}, [(v, 0), \ldots, (d, 0)])$ of r. By Corollary 3.7 we know that the waiting time at each vertex of each route is zero. Without loss of generality, let r'_1 have a shorter driving time than r'_2.

Now consider s-v-routes r''_1 and r''_2 where r'_1 and r'_2 were removed from r_1 and r_2 respectively. Both routes arrive at v at the same time t'_{start}. Now select r''_1 or r''_2 whichever has a lower driving time and call it r. Consider route r^*, where we took route r and added r'_1. This route is valid and feasible since r and r'_1 are valid and feasible. We have constructed route r^* such that it is a valid s-d-route via u. For the driving time of r^* we have $t_{driving}(r^*) = t_{driving}(r) + t_{driving}(r'_1) \leq t_{driving}(r''_i) + t_{driving}(r'_i) = t_{driving}(r_i)$ for $i = 1, 2$. For the travel time we have $t_{travel}(r*) = t'_{start} + t_{travel}(r'_1) = t'_{start} + t_{driving}(r'_1) \leq t'_{start} + t_{driving}(r'_i) = t_{travel}(r_i)$ for $i = 1, 2$. Hence either

1. $(t_{travel}(r^*), t_{driving}(r^*))$ dominates $(t^1_{travel}, t^1_{driving})$ and hence $(t^1_{travel}, t^1_{driving}) \notin R$ which contradicts our assumption $(t^1_{travel}, t^1_{driving}) \in R$ or

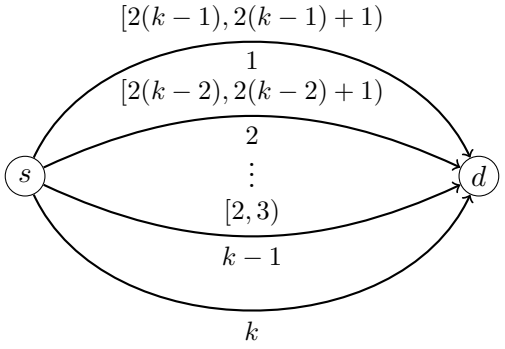


Figure 3.2: Example graph with edge-guards (with parallel edges). There are $k-1$ intervals and k Pareto-optimal routes of a s-d-query with a departure at time 0 (each edge is part of exactly one Pareto-optimal route).

2. $(t_{travel}(r^*), t_{driving}(r^*))$ dominates $(t^2_{travel}, t^2_{driving})$ and hence $(t^2_{travel}, t^2_{driving}) \notin R$ which contradicts our assumption $(t^2_{travel}, t^2_{driving}) \in R$ or
3. $(t^1_{travel}, t^1_{driving}) = (t_{travel}(r^*), t_{driving}(r^*)) = (t^2_{travel}, t^2_{driving})$ which contradicts our assumption $(t^1_{travel}, t^1_{driving}) \neq (t^2_{travel}, t^2_{driving})$.

□

Given e is the number of guarded edges, there might exist a better bound $|R| \leq e + 1$. In Figure 3.2, the graph has $W = k - 1$ intervals and $k - 1 = e$ guarded edges. A s-d-query with a departure at time 0 yields the Pareto-set $R = \{(k, k), (k + 1, k - 1), \ldots, (2k, 1)\}$ with $|R| = W + 1$.

3.1.2 Complexity

For the general time-dependent shortest-path problem with non-fifo weights, Orda et al. have proven, that it is $\mathcal{NP}$-hard to decide whether or not there is a s-d-path shorter than a given value [OR89]. We now show, that our problem is also $\mathcal{NP}$-hard. For a proof, we first define the corresponding decision problem:

Definition 3.8 (Time-dependent road-closure routing decision problem (TDRCRDP))**.** *Given a graph G with edge-guards, vertices s and d and a start time t_{start}. Decide whether or not there exists a valid and feasible s-d-route with start time t_{start}.*

Note that we do not ask for a route shorter than a given value but only for a valid route nor do we ask about all Pareto-optimal routes in terms of travel time and driving time. Instead of using open-intervals I, we could also think of time-dependent non-fifo edge weights

$$c(e, t) \coloneqq \begin{cases} c(e), & t \in I(e) \\ \infty, & \text{otherwise.} \end{cases}$$

Then a route is only valid if it has finite weight. We could also define the TDRCRDP using a parameter k for the maximum allowed route length, but as we will see in the following, we do not need that. For our proof, we use the TDRCRDP as in Definition 3.8 by reducing an instance of TDRCRDP to an instance of Partition which is weak $\mathcal{NP}$-hard [Kar72].

Definition 3.9 (Partition)**.** *Given a set of natural numbers $P = \{p_1, \ldots, p_n\}$. Decide whether or not there is a partition (P_1, P_2) of P such that $P_1 \dot\cup P_2 = P$, and $\sum_{p \in P_1} p = \sum_{p \in P_2} p$.*

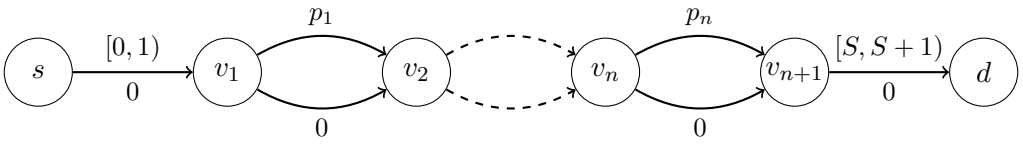


Figure 3.3: Graph G corresponding to an instance of PARTITION as described in the proof.

Theorem 3.10 (Complexity of the TDRCRDP with exact feasibility)**.** *Deciding the time-dependent road-closure route problem with exact feasibility is $\mathcal{NP}$-hard.*

Proof. We reduce an instance of PARTITION to an instance of the TDRCRDP, such that the instance of TDRCRDP is a yes-instance if and only if the instance of PARTITION is a yes-instance: Given an instance $\mathcal{I}$ of PARTITION, i.e. numbers $P = \{p_1, \dots, p_n\}$ and $\sum_{p\in P} p = 2S$. We construct an instance $\mathcal{J}$ of the TDRCRDP. Let $G = (V, E)$ be a graph on $n + 3$ vertices where none of which is a parking lot. Without loss of generality graph G may contain parallel edges. The vertex set V contains vertices $v_1, \dots, v_{n+1}$ and vertices s and d. We have edges $e_s = (s, v_1)$ and $e_t = (v_{n+1}, t)$ with $c(e_s) = c(e_t) = 0$, $I(e_s) = [0, 1)$ and $I(e_t) = [S, S + 1)$. Per (v_i, v_{i+1})-pair ($i \in \{1, \dots, n\}$), we have two unguarded parallel (v_i, v_{i+1})-edges e_i^0 and e_i^p. We set $c(e_i^0) = 0$ and $c(e_i^p) = p_i$. See Figure 3.3 for a visualization of G. Let the start vertex of $\mathcal{J}$ be s and the destination vertex be d. Let the departure time be 0.

We now show that if $\mathcal{I}$ is a yes-instance of PARTITION, then $\mathcal{J}$ is a yes-instance of TDRCRDP: Assume $\mathcal{I}$ is a yes-partition, hence there is a partition (P_1, P_2) of P such that $\sum_{p\in P_1} = S$. A valid s-d-route Q consists of vertices $(s, v_1, \dots, v_{n+1}, d)$. For each (v_i, v_{i+1}) we have to choose an edge (either e_i^0 or e_i^p). For each p_i we choose edge e_i^0 if $p_i \notin P_1$ and edge e_i^p if $p_i \in P_1$. This yields a valid route when waiting nowhere: All edges except e_s and e_d are unguarded. Edge e_s is traversable since we start at s at 0 and $0 \in [0, 1) = I(e_s)$. We arrive at v_{n+1} at

$$\sum_{e\in E(Q)\setminus\{e_d\}} c(e) = c(e_s) + \sum_{i\le n} \begin{cases} p_i, \text{if } e_i^p \in E(Q) \\ 0, \text{otherwise} \end{cases} = \sum_{p\in P} \begin{cases} p, \text{if } p \in P_1 \\ 0, \text{otherwise} \end{cases} = \sum_{p\in P_1} p = S$$

and since e_t is traversable at S as $I(e_t) = [S, S + 1)$, Q is valid. Hence instance $\mathcal{J}$ is a yes-instance of the TDRCRDP.

We now show that if $\mathcal{J}$ is a yes-instance of TDRCRDP, then $\mathcal{I}$ is a yes-instance of PARTITION: There is a valid s-d-route Q starting at 0 in G. Set $P_1 = \{c(e) \mid e \in E(Q) \setminus \{e_s, e_d, e_1^0, \dots, e_n^0\}\}$. $P_1 \subseteq P$ since $E(Q) \setminus \{e_s, e_d, e_1^0, \dots, e_n^0\} \subseteq \{e_1^p, \dots, e_n^p\}$ and $c(e_i^p) \in P$. Also, we have $S = \sum_{e\in E(Q)} c(e)$, otherwise e_d was not traversable. Hence we have

$$\sum_{p\in P_1} p = \sum_{e\in E(Q)\setminus\{e_s,e_d,e_1^0,\dots,e_n^0\}} c(e) = \sum_{e\in E(Q)} c(e) = S.$$

Then for $P_2 = P \setminus P_1$ we have

$$\sum_{p\in P_2} p = \sum_{p\in P} p - \sum_{p\in P_1} p = S = \sum_{p\in P_1} p,$$

hence (P_1, P_2) is a partition of P and hence $\mathcal{I}$ is a yes-instance of PARTITION. □

3.1.3 Algorithm description

In this section, we describe an algorithm, which finds all valid and feasible Pareto-optimal solutions in terms of travel time and driving time. This algorithm is used as basis for the algorithms in the next sections.

We use MCD from Algorithm 2.2, where each label is a tuple consisting of

1. $t_{arrival}$: earliest arrival at the vertex
2. t_{slack}: the *slack*, which is the period of time that we can wait at a previous vertex without invalidating the route
3. $t_{driving}$: driving time from s to d

Given a label $(t_{arrival}, t_{slack}, t_{driving})$ at vertex v, the driver can depart from v at time $t_{arrival}$ with a driving time of $t_{driving}$. Due to the slack, the driver also can depart from v for any amount of time in the interval $[0, t_{slack})$ later. Hence, the driver can depart from v at any time in $[t_{arrival}, t_{arrival} + t_{slack})$.

When starting a query at vertex s with a start time t_{start}, the tuple is initialized as follows:

$$(t_{start}, \infty, 0)$$

The priority of the labels is lexicographically ordered by arrival time $t_{arrival}$ first and driving time $t_{driving}$ second. When calling the RELAX()-function with an unguarded edge $e = (u, v)$ at vertex u with label $(t_{arrival}, t_{slack}, t_{driving})$ to vertex v, then we calculate a new label $(t'_{arrival}, t'_{slack}, t'_{driving})$ at vertex v as follows:

$$(t'_{arrival}, t'_{slack}, t'_{driving}) = (t_{arrival} + c(e), t_{slack}, t_{driving} + c(e))$$

If edge e is guarded, we create at most $|I^*(e)|$ new labels at vertex v. For each interval $[\alpha, \omega] \in I(e)$ we try to create a new label at vertex v. If traversing the edge is allowed at $t_{arrival} \in [\alpha, \omega]$ then we calculate a new label at vertex v as follows:

$$(t'_{arrival}, t'_{slack}, t'_{driving}) = (t_{arrival} + c(e), \min\{t_{slack}, \omega - t_{arrival}\}, t_{driving} + c(e))$$

If $t_{arrival} < \alpha$, i. e. when arriving at vertex u at time $t_{arrival}$, the driver has to wait until the road opens for this interval, then we have a waiting time of $b = \alpha - t_{arrival}$. If $b \leq t_{slack}$, waiting for an additional duration of b at u does not invalidate the current label's route, and we calculate a new label at vertex v as follows:

$$(t'_{arrival}, t'_{slack}, t'_{driving}) = (\alpha + c(e), \min\{t_{slack} - b, \omega - \alpha\}, t_{driving} + c(e))$$

Otherwise we have $b > t_{slack}$, but then we cannot wait for a duration of b without invalidating the current label's route. Or $t_{arrival} > \omega$ and we are too late to traverse this edge. In both cases traversing the edge with the current label and interval is not possible and we do not calculate a new label.

For each label we created, we check whether vertex u represents a parking lot. Then we set the slack to ∞ as we can wait at u as long as we want to:

$$(t'_{arrival}, t'_{slack}, t'_{driving}) = (t_{arrival}, \infty, t_{driving})$$

When creating a label, we check whether or not it dominates another label at the same vertex. If it does, we can delete the dominated label. A label $(t_{arrival}, t_{slack}, t_{driving})$ at vertex u dominates another label $(t'_{arrival}, t'_{slack}, t'_{driving})$ at vertex u, if

1. $[t'_{arrival}, t'_{arrival} + t'_{slack}) \subseteq [t_{arrival}, t_{arrival} + t_{slack})$ and
2. $t'_{driving} \geq t_{driving}$

After the multicriteria dijkstra has finished, there is a number of labels at the destination vertex d. The algorithm then needs to filter all non-Pareto-optimal labels. This is necessary as the slack is part of the domination criteria of dijkstra, but the slack is not part of the solution labels.

Each solution label is correct, as the corresponding routes only traverse edges when they are traversable. Each waiting time needed can be taken on a parking lot along the route or at the start vertex s. Hence for each solution label, there exists a feasible route in the exact model.

3.2 Allowing waiting at every vertex

Instead of waiting only at parking lots and the start vertex, we could allow waiting at every vertex. This may yield routes which are not feasible in practice, especially for drivers of large trucks.

Definition 3.11 (Waiting allowed everywhere-feasibility)**.** *Let G be a graph with edge-guards, s the start vertex, d the destination vertex and t the start time. A s-d-route with the start time t is feasible if it is valid.*

Note that this definition is equivalent to the definition of the exact feasibility when assuming $P = V$. However, we present this algorithm as the complexity of this problem is in $\mathcal{P}$, as we will see later in this section. Also, the algorithm gives lower bounds on the travel time and driving time for the exact algorithm in the section above. We use this bound for quality comparison in the experiment Chapter 5.

For example routes see Figure 3.1 again. As before, we get two solutions for a s-d-query with departure time 0:

1. The route $(0, [s, (v, 9), w, d])$: The driver waits at v for 9 (or alternatively at s). Then he can proceed to the destination vertex d. He arrives at d at time 12, and has a driving time of 3.
2. The route $(0, [s, v, (y, 3), w, d])$: Compared to the exact model where y is not a parking lot, the driver can wait at y and does not need to drive the y-z-y-cycle multiple times, but instead he can wait at y. This gives an arrival time of 7 and a driving time of 4, which is better than the solution in the exact model. However, it may not be possible to wait at y in practice.

As this algorithm is only a special case of the exact algorithm where we assume that every vertex is a parking lot, we can use the exact algorithm presented above. Hence, the cardinality of the resulting Pareto-set of a query is also bounded by $W + 1$ and the example in Figure 3.2 again yields $W + 1$ Pareto-optimal labels. Before we prove this, we present a simplification of the algorithm where we inherently assume that $P = V$.

3.2.1 Algorithm description

Although we can use the exact algorithm presented in the section above, we can simplify the algorithm slightly: We do not need to save the slack information, as every vertex is a parking lot and hence the slack would be set to infinity at every vertex. Therefore, the labels contain the following information:

- $t_{arrival}$: earliest arrival at the current vertex
- $t_{driving}$: driving time to the current vertex

The algorithm settles the labels in lexicographical order of arrival time $t_{arrival}$ first and driving time $t_{driving}$ second. When relaxing an unguarded edge $e = (u, v)$ or a guarded edge which is traversable at $t_{arrival}$ (i.e. $t_{arrival} \in I(e)$) at vertex u with label $(t_{arrival}, t_{driving})$, then we calculate a new label $(t'_{arrival}, t'_{driving})$ at vertex v as follows:

$$(t'_{arrival}, t'_{driving}) = (t_{arrival} + c(e), t_{driving} + c(e))$$

If the edge e is guarded and e is not traversable at $t_{arrival}$ (i.e. $t_{arrival} \notin I(e)$), then let $[\alpha, \omega) \in I^*(e)$ be the earliest interval after $t_{arrival}$. We have to wait until e opens at α:

$$(t'_{arrival}, t'_{driving}) = (\alpha, t_{driving} + c(e))$$

Note that for each guarded edge, we only create at most one label. We do not need to create labels for multiple intervals. By taking the earliest possible interval, we arrive at the next vertex v as early as possible, and we can still wait anywhere later if needed.

As mentioned before, this is just a simplified version of the exact algorithm in the previous section, where we assume a slack of ∞ and where every vertex is a parking lot. Therefore, a label $(t_{arrival}, t_{driving})$ at vertex u dominates another label $(t'_{arrival}, t'_{driving})$ at u, if

1. $t'_{arrival} \geq t_{arrival}$ and
2. $t'_{driving} \geq t_{driving}$.

In this case, the filtering step is not required, as for each vertex there are only the Pareto-optimal labels saved. The correctness of the algorithm follows from the correctness of the algorithm for the exact model.

3.2.2 Implementation details

Instead of one queue that contains all unsettled labels, each vertex has a queue containing all the labels associated with this vertex. A main queue contains each vertex' highest priority label.

In contrast to the results of the exact algorithm, loops and u-turns are never part of a Pareto-optimal solution, because we could wait at each vertex instead. Hence, we do not need to relax edge (v, u) (if it exists) if the current label was created using edge (u, v) as this would only yield a u-turn.

3.2.3 Complexity

As we already mentioned above, the problem is in $\mathcal{P}$. We prove this in the following theorem:

Theorem 3.12 (Complexity of the TDRCRDP with waiting-allowed-everywhere feasibility)**.** *Consider the TDRCRDP with waiting-allowed-everywhere feasibility. Then TDRCRDP* $\in \mathcal{P}$.

Proof. For all guarded edges e, let $W = \sum_e |I(e)|$. By the statement in Chapter 2, the running time of MCD for a s-d-query with a departure at time t has polynomial running time, if

- the RELAX()-function has polynomial running time: The function has a running time of $\mathcal{O}(W)$.
- the maximum number of labels during the execution of the algorithm is bounded by a polynomial in the input size: The cardinality of the label set L_v is bounded by $W + 1$. The maximum number of labels is then bounded by $|V| \cdot (W + 1)$, which is a polynomial in the input size.

Hence, we see that TDRCRDP $\in \mathcal{P}$, if waiting is allowed everywhere. □

3.3 Detour-free route model

Calculating a solution in the exact model is $\mathcal{NP}$-hard and can yield solution we do not want in practice (e.g. driving in a roundabout multiple times to arrive later at a guarded edge). On the other hand, calculating a solution in the model, where waiting is allowed everywhere, is too simplified. Therefore, we try to find another formal description of the problem, which is solvable in practice and yields reasonable results in practice.

We model this, so that waiting time is never taken by driving a detour, but only on parking lots. We first define detour-free t-routes, which do not have waiting time and then based on this definition, we define detour-free feasibility.

Definition 3.13 (detour-free t-route)**.** *Let G be a graph with edge-guards, s the start vertex, d the destination vertex and t the start time. For each vertex v in G, we define the detour-free earliest arrival $dfEA(v)$. For the start vertex s, we set $dfEA(s) = t$. For all other vertices $v \in V \setminus \{s\}$, let $dfEA(v) = \min\{dfEA(u) + c(u,v) \mid (u,v) \in E \wedge dfEA(u) \in I(u,v)\}$.*

The route r is a detour-free t-route if the arrival time at each vertex v in r is equal to $dfEA(v)$.

For example detour-free t routes, consider Figure 3.4.

- For $t \in [4, 9)$, the route $(t, [u, v, w])$ is valid and a detour-free t-route.
- For $t \in [0, \infty)$, the route $(t, [u, v, y, w])$ is valid and for $t \in [0, 4) \cup [9, \infty)$, it is a detour-free t-route.
- For $t \in [3, 8)$, the route $(t, [u, x, v, w])$ is valid, but it is *never* a detour-free t-route.

Now we extend the definition of detour-free t-routes, which may have non-zero waiting time.

Definition 3.14 (Detour-free feasibility)**.** *Let G be a graph with edge-guards, s the start vertex, d the destination vertex and t the start time. Let $p_1, \ldots, p_k \subseteq P$ the parking lot vertices of r in order of their occurence in r. The number of parking lot vertices may be 0. Let $p_0 := s$ and $p_{k+1} := d$.*

The route r is a detour-free route, if

1. *the waiting time is only taken at vertex s or $p_i, i = 1, \ldots, k$, and*
2. *all p_i-p_{i+1}-subroutes of r with a departure from p_i at time t are detour-free t-routes.*

By the definition of detour-free t-routes, a detour-free route between two consecutive parking lots does never contain a loop. Also, a Pareto-optimal detour-free r route never leads to the same parking lot twice. If the route r led to the same parking lot p twice, then there was a detour-free route r', which dominates r: The route r' is equal to the route r, except that the subroute in r between the first departure at the parking lot p and the second arrival at p is cut out and replaced by a longer waiting time at p. However, two distinct subroutes between two consecutive parking lots may contain the same vertex twice.

For examples of Pareto-optimal detour-free routes, consider Figure 3.4, which does not contain parking lots.

- The route $(t, [u, v, w])$ is a Pareto-optimal detour-free feasible route for $t \in [4, 9)$. The route $(t, [(u, 4 - t), v, w])$ is a Pareto-optimal detour-free feasible route for $t \in [0, 4)$
- The route $(t, [u, v, y, w])$ is a detour-free feasible route for $t \in [0, \infty)$. However, it is only Pareto-optimal for $t \in [0, 3) \cup [9, \infty)$, as for $t \in [3, 9)$ the route above has an lower travel time and a lower driving time.

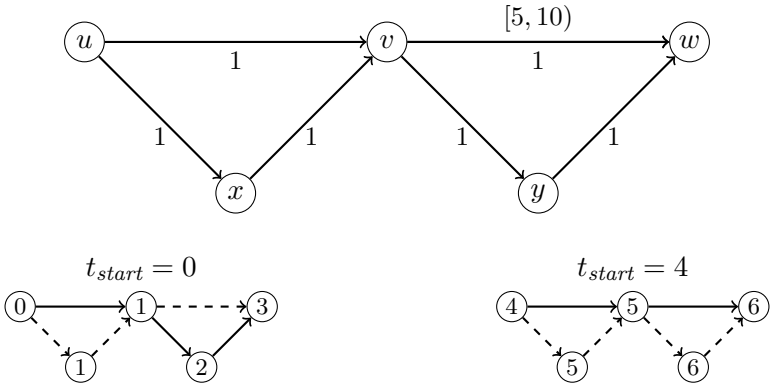


Figure 3.4: Example graph G with edge-guards. The graphs at the bottom have the detour-free earliest arrival for a u-w-query as vertex labels. The graph in the bottom left-hand corner is for a start time of 0, the graph in the bottom right-hand corner is for a start time of 4. The solid lines depict the edges of the corresponding detour-free t-route.

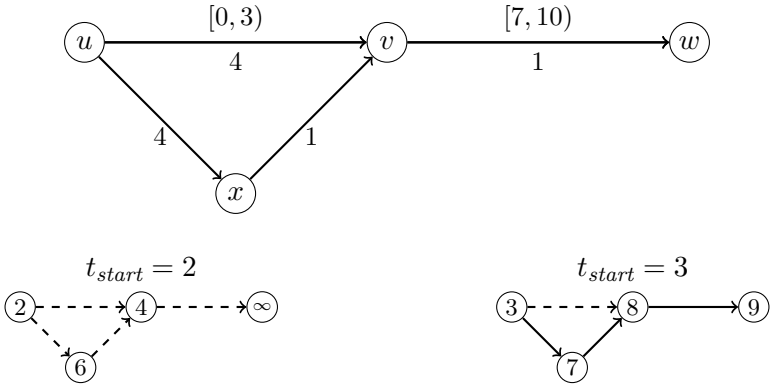


Figure 3.5: Example graph G with edge-guards for comparison of the detour-free routes and the routes in the exact-model. The graphs at the bottom have the detour-free earliest arrival for a u-w-query as vertex labels. The graph in the bottom left-hand corner is for a start time of 2, where there is no detour-free 2-route. The graph in the bottom right-hand corner is for a start time of 3. The solid lines depict the edges of the corresponding detour-free t-route.

For detour-free routes, more restrictions apply than for the routes in the exact model. In Figure 3.5, there is a graph with edge-guards. We note that the edge (u, v) is never part of a valid u-w-route regardless of the underlying model, as the arrival at v using edge (u, v) is always before time 7. In the exact model, route $(0, [(u, 2), x, v, w])$ is a valid and feasible u-w-route with the start time 0. However, this route is not a detour-free route, as the detour-free earliest arrival at v for a departure from u at $t \in [0, 3)$ is always determined by edge (u, v). Then edge (v, w) is not traversable. For $t = 3$, the detour-free earliest arrival at v is determined by edge (x, v), which yields an arrival at u at time 8. Hence, route $(0, [(u, 3), x, v, w])$ is a detour-free route.

3.3.1 Algorithm description

Again, we use the algorithm of Section 3.1 and adapt it so that it meets our requirements. In addition to the arrival time, the slack and the driving time, we also save the following information at each label:

- last parking lot seen p or start vertex s and

- driving time t_{dp} since departing from the last parking lot seen p.

The last parking lot seen gets initialized with vertex s. The driving time since departing from the last parking lot (i.e. the start vertex) is initialized to zero.

When relaxing edge $e = (u, v)$, we calculate the new values p' and t'_{dp} as follows:

$$(p', t'_{dp}) = \begin{cases} (v, 0), \text{if } v \in P \\ (p, t_{dp} + c(e)), \text{otherwise} \end{cases}$$

If v is a parking lot, we set v to the new parking lot and reset the driving time since departure from p to zero.

Let $L = (t_{arrival}, t_{slack}, t_{driving}, p, t_{dp})$ be a label calculated by the detour-free route algorithm at vertex u. When the driver departs at $t \in [t_{arrival} - t_{dp}, t_{arrival} + t_{slack} - t_{dp})$ from the parking lot p and drives to u along the route corresponding to L, then he arrives at vertex u at time $t + t_{dp} \in [t_{arrival}, t_{arrival} + t_{slack})$. From parking lot p to vertex u he drove for a duration of t_{dp}. Hence, the detour-free earliest arrival for a departure from p at time t is less than or equal to t_{dp}.

When inserting the label L into the queue of the multicriteria dijkstra, the labels in the queue at the same vertex u and with the same parking lot p have to be modified, such that the detour-free earliest arrivals at u are correctly represented by the intervals $[t_{arrival} - t_{dp}, t_{arrival} + t_{slack} - t_{dp})$ and the driving times t_{dp}. Therefore, the labels in the queue at vertex u which have the same parking lot p are processed in increasing order of their driving time t_{dp}: The arrival time $t_{arrival}$ and the slack t_{slack} is altered, such that the new interval $[t_{arrival} - t_{dp}, t_{arrival} + t_{slack} - t_{dp})$ is a subset of the old one and does not overlap with the corresponding intervals of the labels that were processed before. If needed, a label has to be splitted into several new labels.

Instead of the domination criteria from Section 3.1, we use the following criteria. A label $L = (t_{arrival}, t_{slack}, t_{driving}, p, t_{dp})$ at vertex u dominates a label $(t'_{arrival}, t'_{slack}, t'_{driving}, p', t'_{dp})$ at the same vertex if

1. $p' = p$ and
2. $[t'_{arrival} - t'_{dp}, t'_{arrival} + t'_{slack} - t'_{dp}) \subseteq [t_{arrival} - t_{dp}, t_{arrival} + t_{slack} - t_{dp})$ and
3. $t'_{dp} \geq t_{dp}$.

After the multicriteria dijkstra has finished, there is a number of labels at the destination vertex d. As in the algorithms presented above, the algorithm needs to filter all non-Pareto-optimal labels. Each solution label which was not filtered corresponds to a detour-free route.

3.3.2 Implementation details

Instead of one queue that contains all unsettled labels, each vertex has a queue containing all the labels associated with this vertex. A main queue contains each vertex' highest priority label.

As in the section before, there are no loops an u-turns in a route corresponding to a solution label, except at parking lots where u-turns may occur. Hence, if the current vertex v is not a parking lot, we do not need to relax edge (v, u) (if it exists) if the current label was created using edge (u, v) as this would yield a u-turn at v.

For more advanced speedup techniques, see Chapter 4.

4. Speeding up shortest detour-free route calculation

In this chapter, we present speedup techniques for the detour-free model. We begin with a Contraction Hierarchy in Section 4.1. Then we show how to efficiently calculate upper bounds for the travel time and for the driving time of the resulting labels. We show how these bounds can be used for pruning labels. In addition, we present two heuristic algorithms. These calculate detour-free labels, but Pareto-optimal results may be missed.

4.1 Contraction Hierarchy

As described in Section 1.1, a Contraction Hierarchy (CH) uses a hierarchical vertex ordering. Vertices are contracted in ascending order of their importance and, if necessary, shortcuts are inserted to preserve shortest distances among the remaining vertex. A s-d-query is answered via a forward-search from start vertex s and a backward-search from destination vertex d. The forward-search and the backward-search only relax edges which lead to vertices of higher importance. See [GSSD08] for a thorough description of Contraction Hierarchies. We adapt this approach for our needs, so that we are able to efficiently calculate the results of a query in the detour-free model.

Each suitable parking lot has to be considered for waiting time, when searching upwards in the hierarchy. Therefore, we enforce that no parking lot is contracted by assigning each parking lot vertex to a high importance and by running the contraction process until only a certain amount of vertices determined by the core size parameter remains. The *core* is the set of remaining uncontracted vertices. Each parking lot is then part of the core. We say that an edge is a core edge, if both of its endpoint vertices are elements of the core. See Section 4.1.1 for a more detailed description of the contraction process.

A Contraction Hierarchy query by default uses a forward search and a backward search which run in an interleaved manner. Our algorithm from Section 3.3.1 uses multiple label components and hence, a backward search beginning at the destination vertex is rather complex. Therefore, we modify the querying phase: The idea is to use a forward search in the search space of the CH for a s-d-query consisting of the forward search space, the backward search space, and the core. We first mark the backward search space, and then, we start the forward search, which searches in the forward search space, the core and the previously marked backward search space. The backward search space marking is based on a breadth-first search beginning at the destination vertex d. When processing vertex u, a

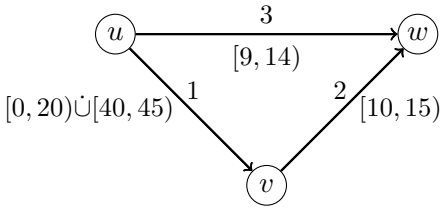


Figure 4.1: Shortcut from u to w which is created when contracting vertex v.

vertex v gets inserted into the queue of the breadth-first search if v has a higher importance than u and if there is a non-core edge (v, u). Edge (v, u) is then marked. By construction of the backward search, only non-core edges get marked.

After the backward search has finished, we start the forward search. When processing a label, we decide which edge we have to relax: We relax the edge (u, v) if the current label at a vertex u was created via

- an upward non-core edge, and edge (u, v) is an upward edge, a core edge or a downward edge that was marked during the backward search.
- a core edge, and edge (u, v) is a core edge or is a downward edge that was marked during the backward search.
- a downward edge that was marked in the backward search, and edge (u, v) is a downward edge that was marked during the backward search.

Each edge is relaxed as we described in Section 3.3.1.

4.1.1 CH-Preprocessing for shortest detour-free routes

When contracting a vertex v, we calculate a shortcut from all adjacent vertices to each other. No contracted vertex is a parking lot, as the parking lots are element of the core and no vertex in the core is contracted. Given two edges (u, v) and (v, w) we calculate the function value of the cost function $c(u, w)$ and the traversal function $I(u, w)$ of a shortcut edge (u, w) as follows:

$$c(u, w) = c(u, v) + c(v, w)$$
$$I(u, w) = I(u, v) \cap (I(v, w) - c(u, v))$$

See Figure 4.1 for an example. In the following, we prove, that the number of intervals in $I^*(u, w)$ is bounded by $|I^*(u, v)| + |I^*(v, w)| - 1$.

Theorem 4.1 (Size of shortcut-interval sets)**.** *Given two edges (u, v) and (v, w), then for the corresponding shortcut (u, w) after contracting v we have: $|I^*(u, w)| \leq |I^*(u, v)| + |I^*(v, w)| - 1$.*

Proof (by induction). Let $A \coloneqq I(u, v), n = |A^*|$, $B \coloneqq I(v, w) - c(u, v), k = |B^*|$ and $C \coloneqq I(u, w)$. We show that $|C^*| \leq n + k - 1$ by induction over n:

Induction base: $n = 1$. There is exactly one interval in $A^* = \{[\alpha_a, \omega_a)\}$ and there are k intervals in B^*. We show, that $|C^*| \leq n + k - 1 = k$.

If there were more than k intervals in C^*, then an interval of $B*$ was split, i.e. there are elements $t_1 < t_2 < t_3 \in B$ such that $t_1, t_3 \in A$ and $t_2 \notin A$. Then we had $\alpha_a \leq t_1 < t_2 < t_3 < \omega_a$, and hence, $t_2 \in A$ which is a contradiction.

Induction step: $n \to n+1$. Assume that $|C^*| \leq n + k - 1$ holds for some value of n (induction hypothesis). Let $|A^*| = n + 1$. We show that $|C^*| \leq (n+1) + k - 1$. We select the earliest interval $I \in A^*$ and split A into $A_1 = I$ and $A_2 = A \setminus I$. Then we split B into $B_1 = B \cap (-\infty, \sup I)$ and $B_2 = B \cap [\sup I, \infty)$. At most one interval in B^* is split into one interval in B_1^* and into another interval in B_2^*, hence we have $|B_1^*| + |B_2^*| - 1 \leq |B^*|$. Let $(A_1 \cap B_1) = C_1$ and $(A_2 \cap B_2) = C_2$.

We now show that $|C^*| \leq |C_1^*| + |C_2^*|$: If there is an interval $J \in C$, then either $J \in C_1^*$ or $J \in C_2^*$. Assume there is an interval $J = [\alpha, \omega) \in C$. Then there exists a corresponding interval $J_a = [\alpha_a, \omega_a)$ in A and a corresponding interval $J_b = [\alpha_b, \omega_b)$ in B, such that $J_a \cap J_b = J$. We either have $\omega \leq \sup I$ or $\alpha > \sup I$. Otherwise we had $\omega > \sup I$ and $\alpha \leq \sup I$, but then $\sup I \in A$, which is a contradiction.

If $\omega \leq \sup I$, then $J_a = I \in A_1$ and $J_b \in B_1$. Then $J \in C_1$. If $\alpha > \sup I$, then $J_a \in A_2$ and $J_b \in B_2$. Hence J either exists in C_1^* or in C_2^*.

We now have:

1. $|C_1^*| \leq |A_1^*| + |B_1^*| - 1 = |B_1^*|$ (by the argument in induction base as $|A_1 * | = 1$) and
2. $|C_2^*| \leq |A_2^*| + |B_2^*| - 1$ (by the induction hypothesis) and
3. $|B_2^*| \leq |B^*| - |B_1^*| + 1$.

By 1. and 2. we have $|C_2^*| \leq |A_2^*| + |B^*| - |B_1^*|$. Hence, we have $|C^*| \leq |C_1^*| + |C_2^*| \leq |B_1^*| + |A_2^*| + |B^*| - |B_1^*| = |A_2^*| + |B^*| = |A^*| - 1 + |B^*| = (n+1) + k - 1$. □

Inserting a shortcut into the Contraction Hierarchy is not always necessary. For each shortcut (u, w), we search for a witness that proves, that the shortcut (u, w) is superfluous. A shortcut is superfluous, if all shortest detour-free routes with start and destination vertices of higher importance than the contracted vertex v do not contain vertex v. The process of searching for such witnesses is called *witness search*. If we do not find an existing witness and insert a superfluous shortcut, this does not corrupt correctness of the contraction hierarchy. Hence, we abort the witness search after a certain number of labels were settled to speed up the preprocessing phase. Also, we simplify the witness search by ignoring the possibility of breaks at parking lots.

When inserting a shortcut (u, w), parallel (u, w)-edges may emerge. Assume there are two parallel edges e and e'. We can modify $I(e')$, if

- $c(e) < c(e')$: Set $I(e') \leftarrow I(e') \setminus I(e)$. Note that if $I(e') = \emptyset$, the witness search did not find the existing witness e and we can delete edge e'.
- $c(e) = c(e')$: Delete both edges and create a new edge e^* with $c(e^*) = c(e)$ and $I(e^*) = I(e) \cup I(e')$. However, extra care must be taken when constructing the corresponding route, as the original edges for this shortcut change over time.

4.2 Upper bounds

Given a s-d-query with a departure at time t, we are interested in the set R of Pareto-optimal pairs of travel time and driving time. Let $r_{travel} = (t^{max}_{travel}, t^{min}_{driving}) \in R$ be a resulting label such that there is no label $r' \in R$ with longer travel time and less driving time and let $r_{driving} = (t^{min}_{travel}, t^{max}_{driving})$ accordingly. In this section, we present ways to calculate r_{travel} and $r_{driving}$ fast. If both labels are equal, we do not run the base detour-free algorithm, as $r_{travel} = r_{driving}$ is the only resulting label in R. If both labels are not equal, we can speed up the base algorithm, as we will see in the following subsections.

4.2.1 Upper bound on travel time

We calculate the label r_{travel} as follows:

1. Calculate the shortest guard-independent s-d-route (using a standard Contraction Hierarchy).
2. If necessary, add waiting time at the start vertex s until the route gets valid.

The second step may fail, if the shortest guard-independent s-d-route is invalid for all waiting times at the start vertex. If the route is valid, then it is also a feasible route in the detour-free model and the corresponding label is equal to r_{travel}, as no label may have less driving time and hence, there is no Pareto-optimal label of longer travel time. However, this approach may fail. See Chapter 5 about how often this approach fails.

If we found the label $r_{travel} = (t^{max}_{travel}, t^{min}_{driving})$, and if we run the base algorithm, we can prune a label $(t_{arrival}, t_{slack}, t_{driving}, p, t_{dp})$ at vertex v before relaxing v's incident edges, if:

$$t_{arrival} - t_{start} + (t^{min}_{driving} - t_{driving}) > t^{max}_{travel}$$

where $t^{min}_{driving} - t_{driving}$ is the minimum time the driver still has to drive before he can reach the destination vertex d. In addition, we can stop the algorithm, if $t_{travel} > t^{max}_{travel}$, as every label in the queue is pruned by the criteria above.

4.2.2 Upper bound on driving time

We calculate the label $r_{driving}$ using the algorithm where waiting was allowed everywhere from Section 3.2. However, we do not calculate the set of Pareto-optimal labels, but stop the algorithm after the first result was calculated. As the waiting times in the corresponding route in general are not at parking lots, we try to move all waiting times to the start vertex. As for the upper bound on the travel time, this may fail. Again see Chapter 5 about how often this approach fails.

This procedure finds an upper bound on the driving time: Given a detour-free route, there is always a route where waiting is allowed everywhere, which has equal or less travel time and driving time, as the waiting-allowed-everywhere model is the least restrictive one. Hence, if we found the label $r_{driving}$, the travel time of this result is not longer than any of the travel times of all shortest detour-free s-d-routes. Accordingly, the driving time of $r_{driving}$ is the maximum of the driving time of all Pareto-optimal labels. Assume there was another Pareto-optimal label with longer driving time. Then it had a shorter travel time than $r_{driving}$, which is a contradiction.

Note that calculating this bound using a CH for shortest detour-free routes may not give the shortest travel time in the original graph using the waiting-allowed-everywhere model. However, there is no route in the CH with less travel time, and as the CH contains all shortest detour-free routes, there is no shortest detour-free route with less travel time. Hence, again the route's driving time is maximal.

4.3 Heuristics

For further speed up we present two heuristics. Both heuristics are based on the algorithm for shortest detour-free routes, but their domination criteria are modified. The heuristics yield shortest detour-free routes, but not necessarily all Pareto-optimal solutions. The second heuristic is a simplification of the first heuristic and hence, the solution quality of the first heuristic is better than the solution quality of the second heuristic. However, the second heuristic has better performance. For an evaluation of the solution quality and the running time of the heuristics see Chapter 5.

4.3.1 Heuristic 1

In contrast to the shortest detour-free routes algorithm, we added the dominance criteria from the exact algorithm.

The complete domination criteria is shown hereinafter. A label $(t'_{arrival}, t'_{slack}, t'_{driving}, p', t'_{dp})$ is dominated by a label $(t_{arrival}, t_{slack}, t_{driving}, p, t_{dp})$ at the same vertex, if

1. $p' = p$ and
2. $[t'_{arrival} - t'_{dp}, t'_{arrival} + t'_{slack} - t'_{dp}) \subseteq [t_{arrival} - t_{dp}, t_{arrival} + t_{slack} - t_{dp})$ and
3. $t'_{dp} \geq t_{dp}$

or

4. $[t'_{arrival}, t_{arrival} + t'_{slack}) \subseteq [t'_{arrival}, t_{arrival} + t_{slack})$ and
5. $t'_{driving} \geq t_{driving}$.

The domination criteria in 4. and 5. only dominate labels L' which yield non-Pareto-optimal solutions in the exact model. That means, there is a label L, which yields a solution in the exact model which dominates the solution of L'. However, label L might not yield a detour-free route and hence, label L is also dominated later. Thus, label L' was dominated erroneously and the solution yielded by L' is not found by Heuristic 1.

4.3.2 Heuristic 2

This heuristic further simplifies the first heuristic. This heuristic yields results corresponding to shortest detour-free routes. The quality decreases compared to the heuristic above, but the performance increases.

The complete domination criteria is shown hereinafter. A label $(t'_{arrival}, t'_{slack}, t'_{driving}, p', t'_{dp})$ is dominated by a label $(t_{arrival}, t_{slack}, t_{driving}, p, t_{dp})$ at the same vertex, if

1. $p' = p$ and
2. $t'_{arrival} \geq t_{arrival}$ and
3. $t'_{dp} \geq t_{dp}$

or

4. $[t'_{arrival}, t_{arrival} + t'_{slack}) \subseteq [t'_{arrival}, t_{arrival} + t_{slack})$ and
5. $t'_{driving} \geq t_{driving}$.

As we ignore the slack in criterium 2., a label L' might be dominated by another label L at a vertex u, because the arrival time $t_{arrival}$ of L at u is earlier and the driving time t_{dp} is lower. However, assume that the slack of L' is much longer than the slack of L. When arriving at a guarded edge (u, v), the slack of L may not be sufficiently long, and hence, no labels are created at vertex v. Then Heuristic 2 misses a solution which Heuristic 1 and the optimal detour-free algorithm found. We point out that for each resulting label $(t_{travel}, t_{driving})$ of Heuristic 2, there is an equal or even a Pareto-optimal result found by Heuristic 1.

5. Experiments

In this chapter, we evaluate the algorithms described in the chapters above. We first present the road network and the test set we used for our experiments. Then we evaluate the algorithms for calculating the bounds from Section 4.2, the performance and quality of the shortest detour-free algorithm and the heuristics.

The implementation of the algorithms is based on RoutingKit [Str17] by Strasser and was compiled with GCC 5.4.1 (64 bit) with optimization level 3. All performance measurements were executed on a VMware ESXi machine with 4 cores of an Intel Xeon E5-2680 v3, 2.5 GHz and 64 GiB RAM. The operating system was Ubuntu 16.04.01 LTS. The algorithms make no use of parallelism.

5.1 Road network

The road network was provided by PTV AG and contains Austria, France, Germany, Italy, Liechtenstein, Luxembourg and Switzerland. All time specifications in this chapter refer to UTC+1, which is the standard time zone for all the countries mentioned. The travel times and road closures were adapted to a truck with a trailer with a gross combined weight of 40 tons. The map has 21,903,924 vertices and 47,635,384 edges and contains 4429 parking lots, which are mainly near highways or country borders. As mentioned, road closures are dependent on the weekday and holidays. See Table 5.1 for an overview of the weekend and night driving bans of the countries above. We refer to [Arm18] for a thorough overview of night, weekend, holiday and other special driving bans. In addition, the road closures concern certain downtown areas and roads which are not mentioned here.

5.2 Test sets

For performance and quality tests we used four different test sets. For each test set we classify the queries by the distance of the start vertex and the destination vertex. The settling order of the vertices in Dijkstra's algorithm in a guard-independent route query is used to rank the vertices. The rank of a query is the rank of its destination vertex.

The first three test sets only differ in the start time. For each of the test sets $1 - 3$, we use 100 randomly generated start vertices. For each start vertex, 13 destination vertices of rank $2^i, i = 12, \ldots, 24$ were selected. For the start time of the queries in the test sets, see Table 5.2.

Table 5.1: Overview of weekend and night driving bans for 40-ton trucks with a trailer in 2018.

country	saturday driving ban	sunday driving ban	night driving ban
Austria[1]	15.00 – 24.00	00.00 – 22.00	22.00 – 05.00
France[2]	22.00 – 24.00	00.00 – 22.00	–
Germany[3]	–	00.00 – 22.00	–
Italy[4]	–	07.00 – 22.00 (Jun. – Sep.) 09.00 – 22.00 (Oct. – May)	–
Liechtenstein[5]	–	00.00 – 24.00	22.00 – 05.00
Luxembourg[6]	21.30 – 24.00 or[7] 23.30 – 24.00	00.00 – 21.45	–
Switzerland[8]	–	00.00 – 24.00	22.00 – 05.00

[1] Austrian road traffic regulations: StVO §42(1)+(6), 12th July 2018
[2] French government gazette: Journal officiel de la République française: n°0302 du 28 décembre 2017, texte n°120
[3] German road traffic regulations: StVO §30(3), 6th October 2017
[4] Italian Ministry of Infrastructure and Transport: Decreto Ministeriale numero 571 del 19-12-2017, Art. 1, 1a)+b)
[5] Liechtenstein road traffic regulations: VRV Art. 89 1)+2), 1st July 2018
[6] Luxembourg road traffic regulations: Réglement grand-ducal modifié du 19 juillet 1997 relatif aux limitations de la circulation des poids lourds pendant les dimanches et jours fériés Art. 1
[7] 21.30 – 24.00 when heading towards France or 23.30 – 24.00 when heading towards Germany
[8] Swiss road traffic regulations: SVG Art. 2(2), 1st January 2018

Table 5.2: Earliest departure time for each test set.

test set	start time
Test set 1	Mo, 2nd July 2018, 05.00
Test set 2	Fr, 6nd July 2018, 05.00
Test set 3	Fr, 6nd July 2018, 17.00
Test set 4	Mo, 2nd July 2018, 17.00

For test set 4, we chose two areas. Area A is the area southeast of 49° N 4° E and northwest of 47° N 18° E. Area B is the area southeast of 46° N 4° E and northwest of 42° N 18° E. Figure 5.1 depicts both areas.

For the queries of test set 4, we randomly selected 100 pairs of geographic coordinates, one coordinate from area A and one coordinate from area B. For both coordinates, the nearest vertices v and u in the graph were searched. For each pair (u, v), we insert the u-v-query and the v-u-query into the test set. Hence, in total we have 200 queries. From 200 queries, 2 have rank 20, 9 have rank 21, 65 have rank 22, 97 have rank 23 and the 27 remaining queries have a rank of 24. Again see Table 5.2 for the start time of the queries in this test set. The test set is made such that for many queries, the shortest guard-independent route goes through Switzerland or Austria, where the night driving ban applies. Hence, we expect multiple resulting labels per query, where some corresponding routes lead around Switzerland or Austria.

In the following evaluation, we use box plots. The abscissa of the plot indicates the rank. The lower end of each box denotes the lower quartile, the upper end denotes the upper quartile of the measurements of the queries in the current rank. The median is shown as thick line. The whiskers extend to up to 1.5 times the interquartile range from the top/bottom of the box to the furthest datum within the range. Points above or below the

Figure 5.1: Area A and B, where start and destination vertices of each query in test set 4 lie in. The left upper corner of area A is at 49° N 4° E, the left lower corner is at 47° N 18° E. The left upper corner of area B is at 46° N 4° E, the left lower corner is at 42° N 18° E.

Table 5.3: CH preprocessing

horizon begin	horizon duration	preprocessing time
Mo, 2nd July 2018, 05.00	48 hours	1 h 20 min
Mo, 2nd July 2018, 05.00	24 hours	1 h 03 min
Mo, 2nd July 2018, 17.00	48 hours	1 h 24 min
Mo, 2nd July 2018, 17.00	24 hours	1 h 07 min
Fr, 6nd July 2018, 05.00	48 hours	7 h 25 min
Fr, 6nd July 2018, 05.00	24 hours	1 h 03 min
Fr, 6nd July 2018, 17.00	48 hours	7 h 35 min
Fr, 6nd July 2018, 17.00	24 hours	5 h 53 min

whiskers are outliers and are depicted as circles. If there is only one measurement per rank, then only its value is shown with a thick line.

5.3 Contraction Hierarchy Preprocessing

We start with the evaluation of the CH preprocessing for shortest detour-free routes. As the preprocessing depends on the road-closures, we evaluate the preprocessing with varying *(planning) horizons*. The traversal function I maps each edge to a subset of the horizon. A CH is suitable for answering queries, which corresponding routes start and end during the planning horizon. For each graph, we ran the preprocessing until only 0.5 % of the vertices remained uncontracted. These uncontracted vertices are the elements of the core. Table 5.3 contains the times needed for the preprocessing.

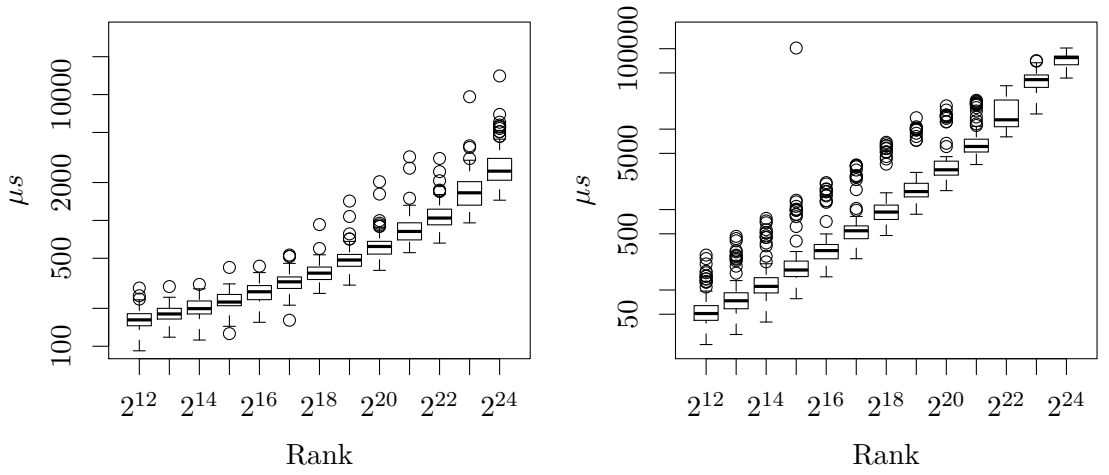
We see that the calculation times are highly dependent on the weekdays in the planning horizon contains. While a computation with a horizon ending at or before Saturday 05.00 only has a preprocessing time of 1 to 1.5 hours, it increases to up to about 7.5 hours if the planning horizon contains Saturday night or Sunday. The reason is that witness search gets much more costly. Before Saturday, there are only smaller areas, where driving is prohibited for trucks and at night, there are only driving bans in Switzerland and Austria. However, in each country a weekend driving ban applies, which slows down the preprocessing drastically.

5.4 Bounds

In this section, we evaluate the running time of the algorithms calculating the upper bounds on travel time ($t^{max}_{travel}, t^{min}_{driving}$) and the upper bound on driving time ($t^{min}_{travel}, t^{max}_{driving}$). For the sake of clarity, we reference t^{max}_{travel} as the maximum travel time and t^{min}_{travel} as the minimum travel time. Correspondingly, we reference $t^{max}_{driving}$ as the maximum driving time and $t^{min}_{driving}$ the minimum driving time. We evaluate how often the minimum and maximum travel times are found, are (un-)equal, and we quantify their deviation. Then we proceed analogously for the minimum and maximum driving times.

5.4.1 Running time

The running time of the bound calculation was evaluated using a Contraction Hierarchy, as described in the previous chapter. The planning horizon begins at the departure of the queries and ends 24 hours later. Figure 5.2 shows the running time for the algorithms calculating the upper bound on the travel time and the upper bound on the driving time for test set 1. As the results of the other test sets are very similar, the running time of the algorithms for the other test sets can be found in Figure A.1 in the Appendix. For



(a) Running time for the upper bound on the travel time.

(b) Running time for the upper bound on the driving time.

Figure 5.2: Running time for the upper bounds for test set 1.

the upper bound on travel time, we only need a plain CH without any further adaptions. Hence, the upper bound on the travel time is calculated in at most 10 ms for queries with a rank of 2^{24}. For the upper bound on the driving time, we need the adaptions explained in Section 4.1. These slow down the calculation for higher ranks compared to the plain CH and hence, 100 ms are needed for queries with rank of 2^{24}.

In Figure 5.2(b), there is a rank 2^{15}-query where the algorithm runs for about 100 ms for the upper bound on the driving time. The algorithm could not find a feasible route when waiting was allowed everywhere. Hence, the algorithm searched the whole graph, which results in a long running time, comparable to the running time of a rank 2^{24}-query. For this query, an upper bound on travel time could not be calculated either, and none of the algorithms we evaluate in the following found any solution.

5.4.2 Quality

We now evaluate how often the bounds were found and how often they are equal to each other. We begin with the upper bound on the travel time. Table 5.4 shows the percentage of the queries, where the upper bound on the travel time was found. For all queries of rank 2^{21} and less, independent of the test set, 99 % of the queries yielded an upper bound. For test sets 1 – 3, the percentage of upper bounds found decreases with increasing rank. In test set 1, the horizon contains Monday to Tuesday morning, whereas in test sets 2 and 3 the weekend is included. Hence, driving bans of all countries in the road network apply for test sets 2 and 3 and the bound calculation fails more often.

Test set 4 contains only Monday to Tuesday. Although most routes in test set 4 lead through Switzerland or Austria, where a night driving ban applies, the upper bound on travel time is always calculated successfully.

For the upper bound on driving time, there is exactly one query in test sets 1 – 3, where no upper bound on driving time could be calculated (see the outlier in the section above). In test set 4, the upper bound on driving time was always successfully calculated.

Table 5.5 shows the percentage of the queries, where the upper bound on the travel time is equal to the upper bound on the driving time. In these cases, there is exactly one solution to output. For short queries, this bound-checking is often successful and thus saves a lot of

Table 5.4: Percentage of queries, where the upper bound on travel time was calculated successfully.

Test set	Rank												
	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}
Test set 1	100	100	100	99	100	100	99	99	99	100	98	99	99
Test set 2	100	100	100	99	100	100	99	99	99	100	98	95	87
Test set 3	100	100	100	98	100	100	98	99	98	99	93	78	64
Test set 4									100	100	100	100	100

Table 5.5: Percentage of the queries, where both upper bunds exist and where the upper bound on the travel time is equal to the upper bound on the driving time.

Test set	Rank												
	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}
Test set 1	99	100	100	97	99	99	96	97	96	96	92	85	87
Test set 2	99	100	100	97	98	98	94	93	84	78	76	67	51
Test set 3	98	98	98	96	98	98	92	92	82	70	63	46	32
Test set 4									100	55	52	48	26

calculation time, as we will see in the following sections. However, for queries from test set 3 and 4 with a rank greater than or equal to 2^{21}, in over 30 % the bounds are unequal or could not be calculated. For ranks of 2^{23} and greater, even over 50 % are unequal or could not be calculated. Clearly, the complexity of the queries increases with the rank. Also, the complexity of the queries seems to increase with the test set number.

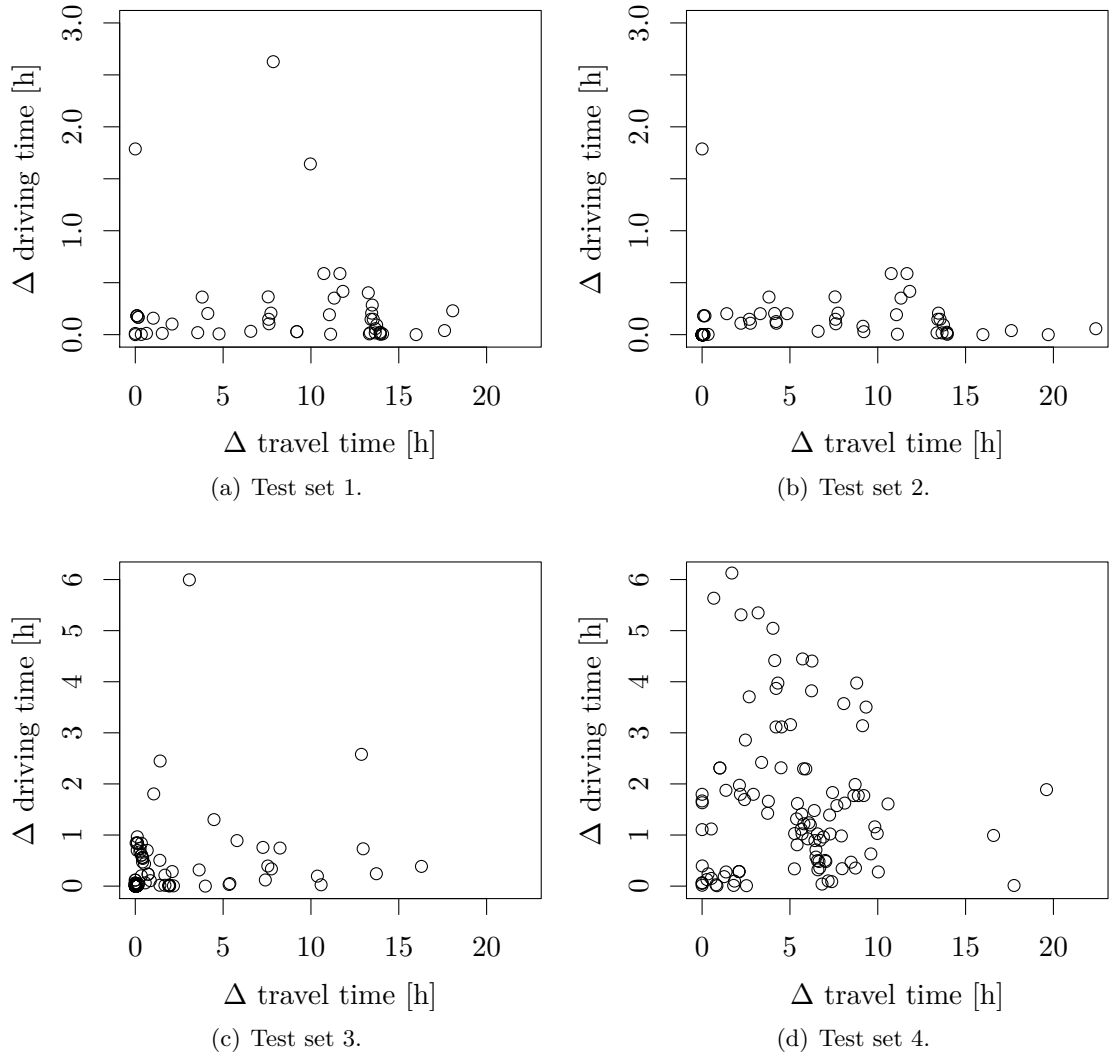
We now evaluate the margin between the upper bound on the travel time $(t^{max}_{travel}, t^{min}_{driving})$ and the upper bound on the driving time $(t^{min}_{travel}, t^{max}_{driving})$. In the following, we only consider queries, where both bounds were found and deviate from each other. In Figures 5.3(a)–5.3(d), for each query and its corresponding upper bounds, a point at $(t^{max}_{travel} - t^{min}_{travel}, t^{max}_{driving} - t^{min}_{driving})$ was inserted into the plot. We call $t^{max}_{driving} - t^{min}_{driving}$ the *driving time difference*.

For test set 1 and 2, the Figures 5.3(a) – 5.3(b) show that in most cases, only a detour of half an hour pays off. When accepting half an hour additional driving time, the travel time can be reduced by up to 20 hours. In these cases, road-closures have a much greater influence on the travel time, than on the driving time. Often, the reason for short detours are only local road-closures, which can be avoided with minor additional driving time, in contrast to country-wide driving bans.

Also for test set 3, a rather small detour yields a shorter travel time. However, the majority of queries in test set 3 have a driving time difference of under one hour. There is also a query, where a detour of over 6 hours reduces the travel time. The query corresponding to this data point is quite similar to the queries in test set 4, where a detour around Switzerland through France reduces the travel time. In test set 4, longer detours of several hours are more common. As many of the guard-independent routes lead through Switzerland or Austria, a detour avoiding both countries can reduce the travel time, although the driving time rises drastically. See Section 5.5.3 for examples queries, where this effect occurs.

5.5 Detour-free route algorithms

In this section, we evaluate the running time of the shortest detour-free route algorithms using the speed up techniques presented in Chapter 4. We also evaluate the cardinality



(a) Test set 1.

(b) Test set 2.

(c) Test set 3.

(d) Test set 4.

Figure 5.3: Each point in the figures correspond to exactly one query. Given upper bounds $(t^{max}_{travel}, t^{min}_{driving})$ and $(t^{min}_{travel}, t^{max}_{driving})$, a point at $(t^{max}_{travel} - t^{min}_{travel}, t^{max}_{driving} - t^{min}_{driving})$ was inserted into the plot.

Table 5.6: Percentage of the queries without successfully calculated upper bounds or unequal bounds, where the calculation finished without a timeout after half an hour calculation time. Empty cells indicate, that there was no query where both upper bounds were not found or unequal.

Test set	Rank												
	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}
Test set 1	100			66	100	100	100	100	100	100	100	69	0
Test set 2	100			66	100	100	100	100	100	100	100	59	0
Test set 3	100	100	100	80	100	100	100	100	100	82	71	46	29
Test set 4										0	16	17	0

and the distribution of the resulting labels per query. Then we present the results and the corresponding routes of two example queries in greater detail.

5.5.1 Running time

In this section, we evaluate the running time of the detour-free route algorithm. For the measurements, we made use of all the speed up techniques presented in Chapter 4. The planning horizon of the CH begins at the departure of the queries and ends 24 hours later. We stopped the calculation after half an hour per query, if it was not already finished. For the running time of the algorithm for shortest detour-free routes, we only evaluate the running time of the queries, where the upper bounds were not successfully calculated or where the upper bounds are unequal. If for a query the upper bounds were successfully calculated and are equal, then the resulting label set contains only one label and the detour-free algorithm is not started.

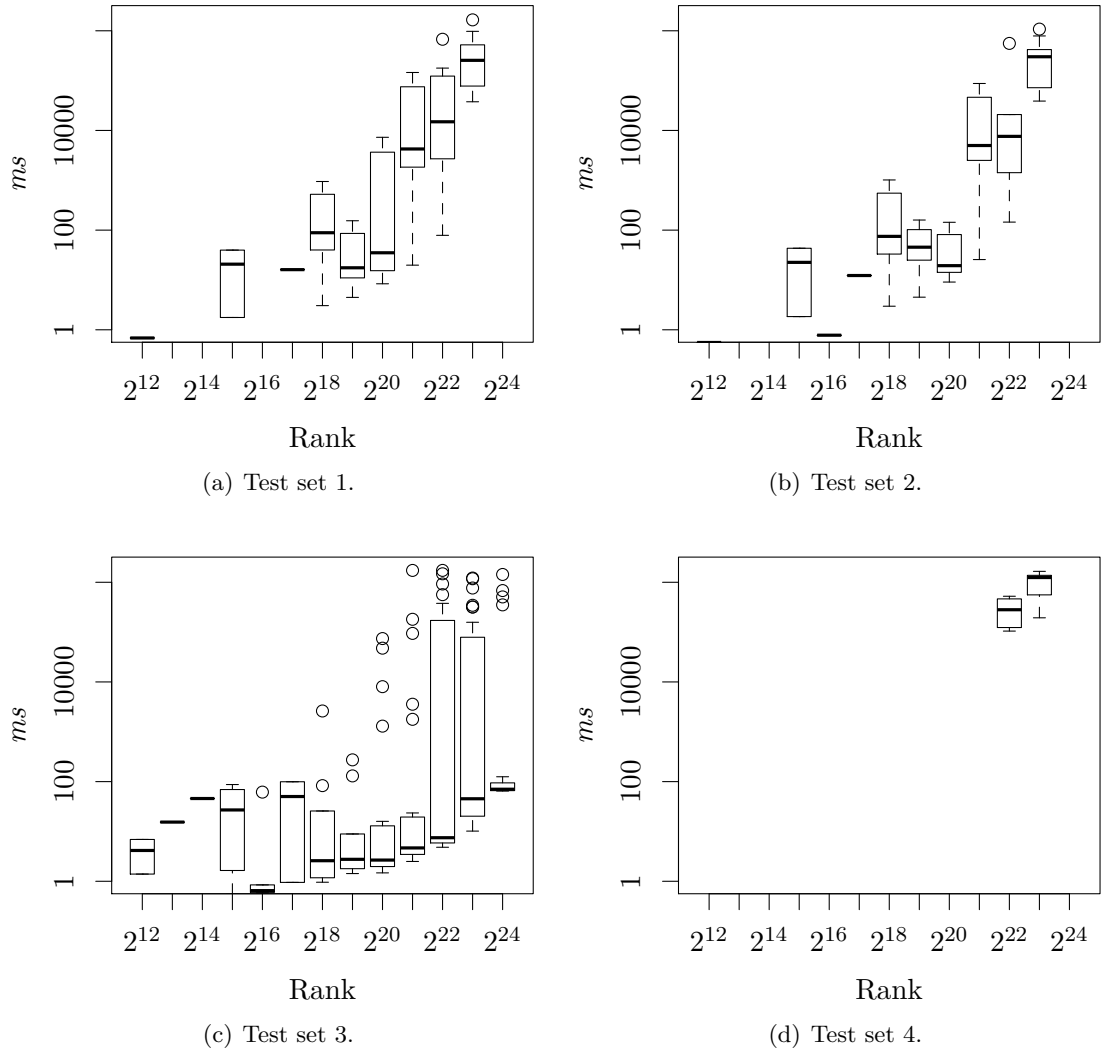
Table 5.6 shows the percentage of the queries without successfully calculated upper bounds or unequal bounds, where the calculation finished without a timeout. The algorithm timed out for all rank 2^{24}-queries, except for queries of test set 3. In test set 4, the algorithm timed out in over 80 % of the queries. Hence, the queries of test set 4 seem to be more difficult than the queries of the same rank from the other test sets.

For the calculation times for queries with equal bounds, see the section above. Figure 5.4 depicts the running time of the detour-free route algorithm for queries in test sets $1 - 4$, where an upper bound was not successfully calculated or where the bounds were unequal. The running time of the queries which timed out are not included. The measurements include the time needed for the calculation of the upper bounds. Apart from the outliers in the test set 3, the computation time is less than one second for queries of rank 2^{20} and less. The computation slows down drastically for greater ranks. In test set 4, the calculation time is slightly higher than for the other test sets.

The heuristic algorithms never timed out for any of the queries. The running time of Heuristic 1 and 2 is shown in Figure 5.5 and in Figure 5.6 respectively. With a maximum running time over all queries from all test sets of 10.4 seconds, Heuristic 1 is clearly faster than the shortest detour-free route algorithm. Heuristic 2 (see Figure 5.6) improves the maximum calculation time to 2.8 seconds. The running time of both heuristics increase drastically for a rank of 2^{21} and higher. For queries of small rank, the algorithms spend little time searching in the core, while for queries of higher rank, this time increases.

5.5.2 Solution quality

The detour-free route algorithm calculates a set of labels for each query. We only evaluate the subset of the queries in the test sets, where more than one label is returned by the



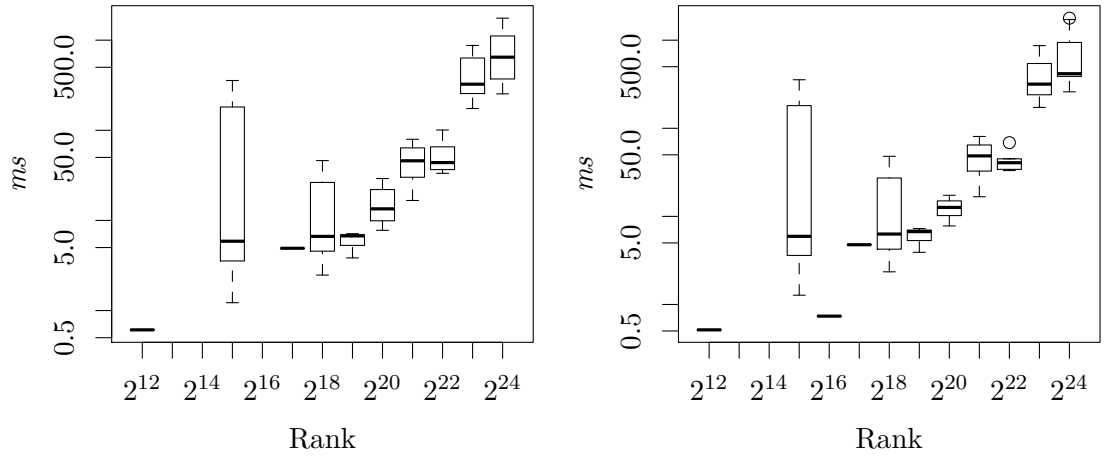
(a) Test set 1.

(b) Test set 2.

(c) Test set 3.

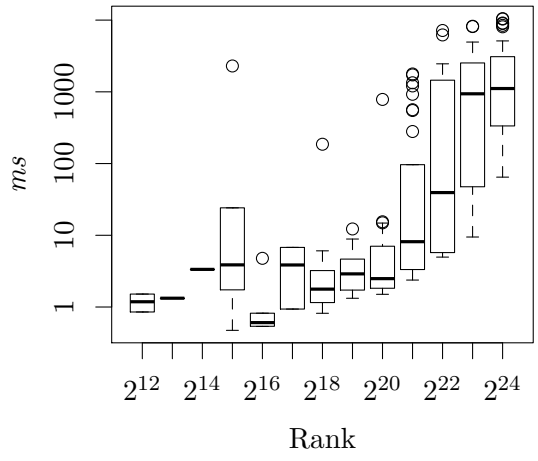
(d) Test set 4.

Figure 5.4: Running time of the detour-free route algorithm for queries, where an upper bound was not successfully calculated or where the bounds were unequal. Running times of the algorithm for queries which timed out are not included.

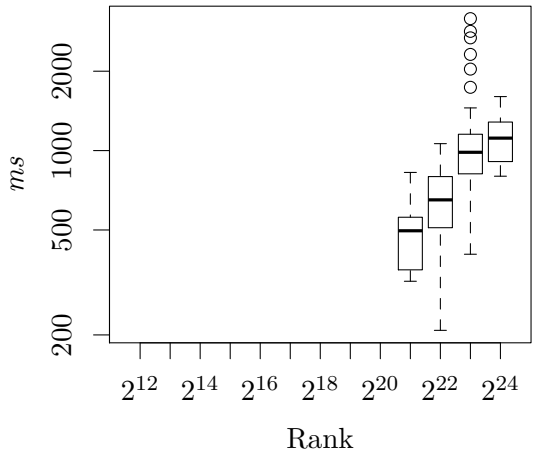


(a) Test set 1.

(b) Test set 2.

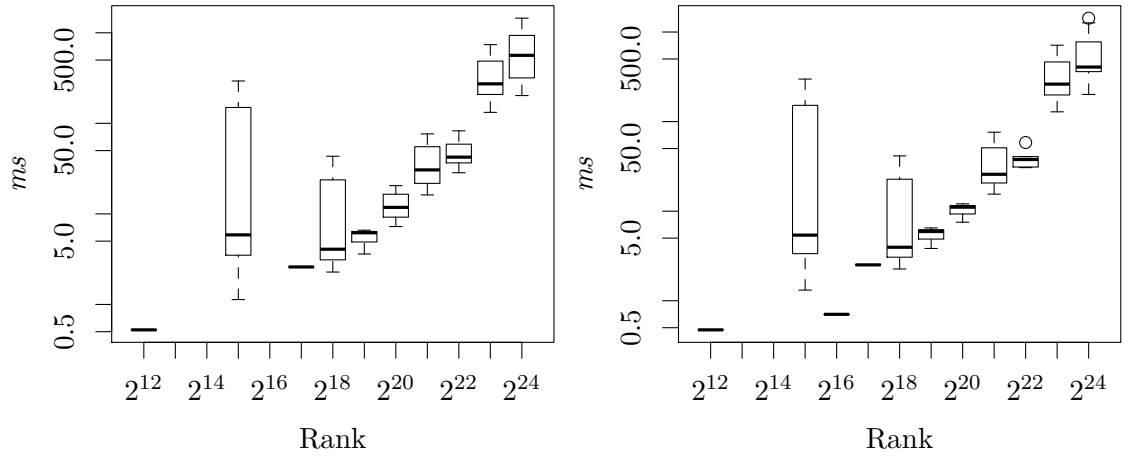


(c) Test set 3.



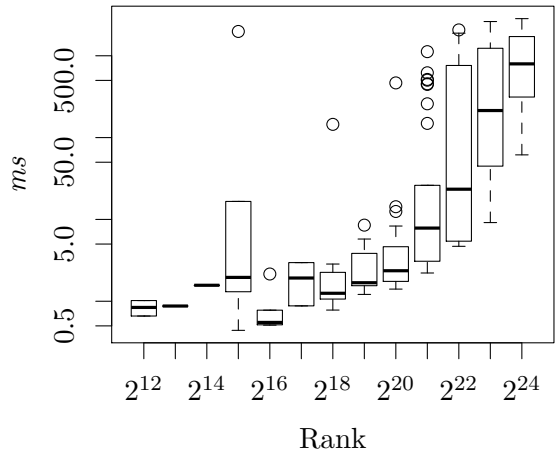
(d) Test set 4.

Figure 5.5: Running time of Heuristic 1 for queries, where an upper bound was not successfully calculated or where the bounds were unequal.

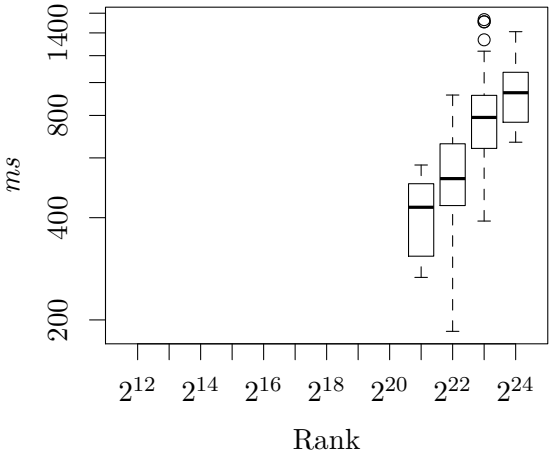


(a) Test set 1.

(b) Test set 2.



(c) Test set 3.



(d) Test set 4.

Figure 5.6: Running time of Heuristic 2 for queries, where an upper bound was not successfully calculated or where the bounds were unequal.

algorithm. The detour-free route algorithm always calculated less than 5 labels for each query. Indeed, most queries only yielded 2 labels. For each test set and rank, there is only at most one query, which yielded more than 2 labels. For a rank of 2^{12}, there was a query in each test set, were 5 labels were returned by the detour-free route algorithm. The destination vertex of the query was behind a downtown area, which is temporarily closed for trucks. Driving around the temporarily closed area yields the earliest arrival with a longer driving time. In addition, there were routes which led only partly around the temporarily closed area and hence saved driving time, but increased travel time compared to the route, which led completely around the temporarily closed area.

Heuristic 1 performed well compared to the exact detour-free routes: There was not a single difference in any of the test cases. However, the detour-free route algorithm timed out often. Hence, the data basis on which we can compare the detour-free route algorithm and Heuristic 1 is rather small.

For a detailed evaluation of the margins between the resulting labels of the queries, we use plots similar to the ones used for the evaluation of the upper bounds. Given a *s-d*-query, the algorithms return a set of labels R. Two labels $(t^1_{travel}, t^1_{driving}), (t^2_{travel}, t^2_{driving}) \in R, t^1_{travel} < t^2_{travel}$ are *consecutive* in R, if there are is no label in between, i.e. there is no label $(t^3_{travel}, t^3_{driving})$ in R such that

1. $t^3_{travel} \in [t^1_{travel}, t^2_{travel}]$ and
2. $t^3_{driving} \in [t^2_{driving}, t^1_{driving}]$.

For each query, Figures 5.7(a) – 5.7(d) show a data point $(t^\Delta_{travel}, t^\Delta_{driving})$ for each two consecutive labels $(t^1_{travel}, t^1_{driving}), (t^2_{travel}, t^2_{driving})$ of a query, where

1. $t^1_{travel} < t^2_{travel}$,
2. $t^\Delta_{travel} \coloneqq t^1_{travel} - t^2_{travel}$ and
3. $t^\Delta_{driving} \coloneqq t^2_{driving} - t^1_{driving}$.

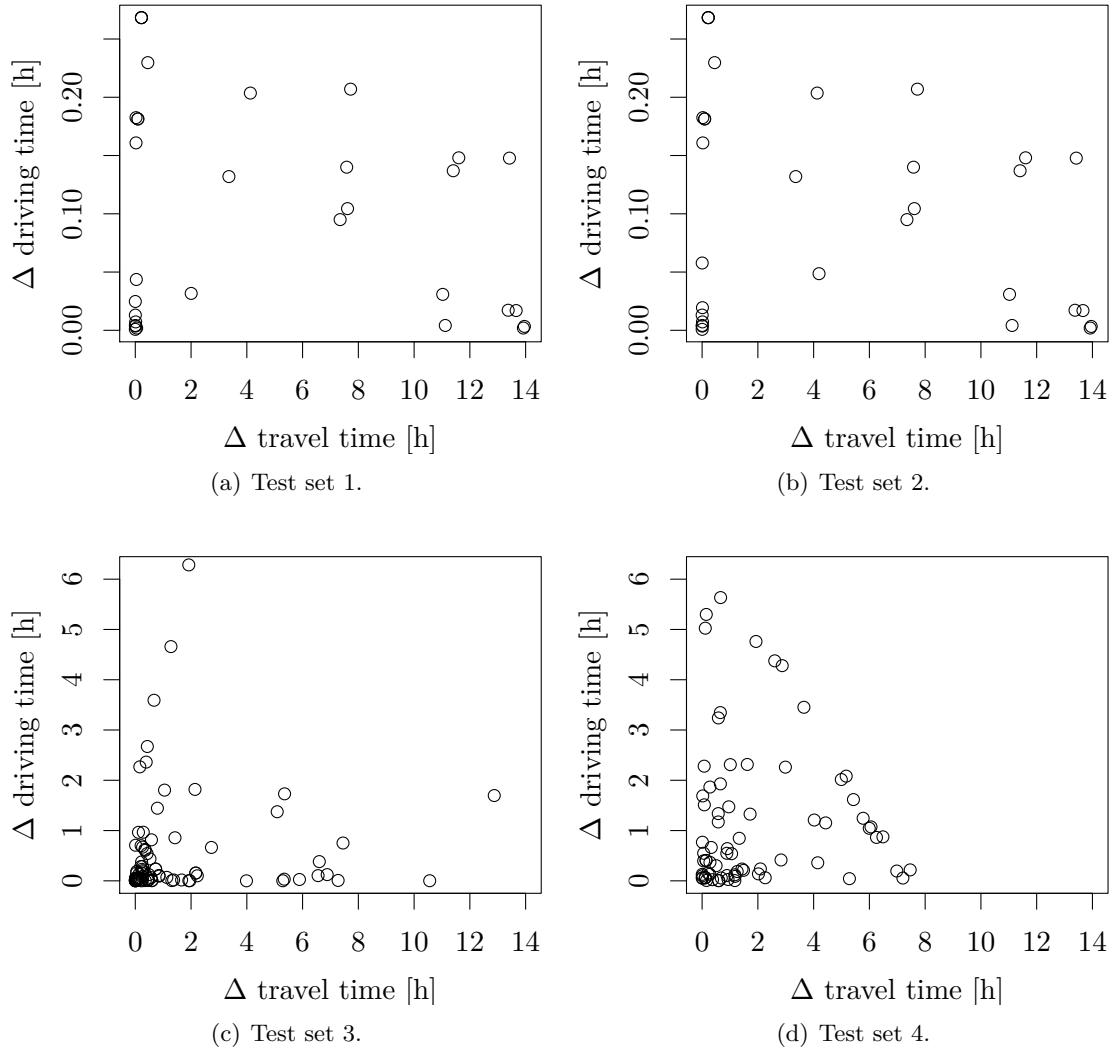
In the following, the values t^Δ_{travel} and $t^\Delta_{driving}$ are called travel time differences and driving time differences.

The plots for test set 1 and 2 contain 32 and 31 data points, respectively. The data points originate from 18 queries which yielded more than only 1 label. In the plot for test set 3, there are 93 points of 49 queries, and in the plot for test set 4, there are 74 points of 44 queries.

For test sets 1 and 2, there was no query, where a detour of over 20 minutes reduced the travel time. The reason, why only a short duration of driving time can reduce the travel time significantly, was already described in the Section 5.4.2.

In contrast, for test set 3, there are queries, where a detour of multiple hours can decrease the arrival time by only minutes. In test set 3, 33 data points in the plot have an abscissa of more than 1 hours or an ordinate of more than 1 hour. Hence, there are 33 pairs of consecutive labels, which have at least a 1-hour difference in driving time and in travel time. The same applies for 48 data points of test set 4.

We now compare the results of Heuristic 1 to the results of Heuristic 2. The results of Heurstic 2 are very similar to the results of Heuristic 1: In test set 1, 2 and 4, Heuristic 2 did always find exactly the results, Heuristic 1 found. In test set 3, there are five queries, where the resulting label sets differ. The queries have a rank of 2^{21} and higher. In two of the five queries, the number of resulting labels was identical. In both cases, a single label calculated by Heuristic 2 deviates by approximately 1 – 2 % in travel time and driving time from its counterpart calculated by Heuristic 1.



(a) Test set 1.

(b) Test set 2.

(c) Test set 3.

(d) Test set 4.

Figure 5.7: Results of Heuristic 1. For two consecutive labels $(t^1_{travel}, t^1_{driving}), (t^2_{travel}, t^2_{driving})$ in the resulting set of a query with $t^1_{travel} < t^2_{travel}$, there is a point $(t^2_{travel} - t^1_{travel}, t^1_{driving} - t^2_{driving})$ in the plot with the corresponding test set.

In the other three queries, Heuristic 2 found exactly one label less than Heuristic 1. All three labels missing are the labels with maximum travel time and minimum driving time. The missing labels all have a travel time which is several hours longer than the travel time of their consecutive labels. The driving time differs by 2 – 4 hours. Figure B.1 shows the travel time and driving time differences for the labels calculated by Heuristic 2. As is it does not contain new information compared to Figure 5.7, it can be found in the Appendix.

5.5.3 Example queries

In the following, we discuss two example queries in detail. We show their resulting label set and present the routes on a map. Using the detour-free route algorithm, the first example query yields two labels, whereas the second example query yields six labels.

5.5.3.1 Example 1

We evaluate a rank 2^{23}-query from Pettenasco, Italy to Erlangen, Germany, with a start time at Friday, 6th July 2018 at 18.00. The detour-free route algorithm returns two labels, whose corresponding routes are shown in Figure 5.8 in red and blue. Parking lots where the driver waits are represented as markers in the route's color.

Using the red route, the driver arrives in Erlangen after 11 hours and 28 minutes of driving and a waiting time of 7 hours and 26 minutes. The driver waits at a parking lot, where he arrives at 21.34 in Switzerland. The waiting time is needed because of the night driving ban in Switzerland starting at 22.00. The driver departs from the parking lot at exactly 05:00 on Saturday, which is when the night driving ban is over again. Then he drives to Erlangen without further waiting time.

Using the blue route, the driver arrives in Erlangen after 14 hours and 9 minutes of driving and a waiting time of 4 hours and 20 minutes. The driver waits at a parking lot, where he arrives at 00.38 on Saturday, 7th July, in Italy at the border to Austria. The waiting time is needed because of the night driving ban in Austria which starts at 22.00. The driver departs from the parking lot such that he arrives at the Austrian border at exactly 05.00, which is when the night driving ban is over again. Then he drives to Erlangen without further waiting time.

The red route has a lower driving time, but a total travel time of 18 hours and 54 minutes compared to 18 hours and 29 minutes, hence there is a 25 minutes difference. Using the blue route, the driver is able to drive in Italy until 00.38. Using the red route, the driver can only drive until 21.34. When the night driving bans in Switzerland and Austria are over at 05.00, the driver on the blue route is already nearer to his destination compared to the driver on the red route and hence, the driver on the blue route arrives earlier in Erlangen, although he has a longer driving time.

5.5.3.2 Example 2

We evaluate a rank 2^{22}-query from Milano, Italy to Freiburg im Breisgau, Germany, with a start time at Monday, 2nd July 2018 at 21.00. The detour-free route algorithm returns six labels as shown in Table 5.7, whose corresponding routes are shown in Figures 5.9(a)–5.9(e).

Consider route number 1 in Figure 5.9(a): The map shows that the driver is routed from Italy via France to Germany to avoid the night driving ban in Switzerland and in Austria. Hence we get the earliest arrival compared to the other routes, but the driving time is the highest of all routes.

As we can see in the other figures, it is possible to drive through Switzerland, but then the driver has to wait near the Swiss border until the night driving ban in Switzerland

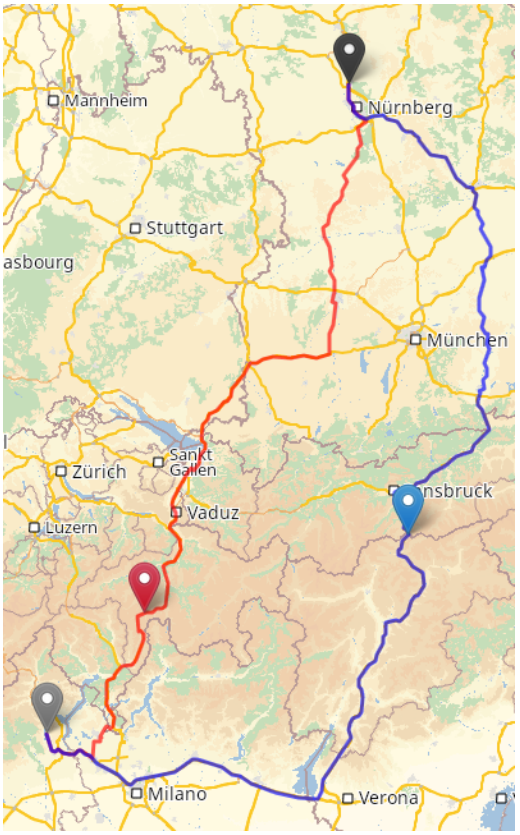


Figure 5.8: The two routes of a rank 2^{23}-query from Pettenasco, Italy to Erlangen, Germany. The start is at the grey marker, the destination is at the black marker. The red and blue markers indicate the location of parking lots, where the driver waits.

Table 5.7: Travel times and driving times for the resulting labels of the rank 2^{22}-query in Figure 5.9.

route	figure	color	travel time (hh:mm:ss)	driving time (hh:mm:ss)
1	5.9(a)	red	11:26:14	11:26:14
2	5.9(b)	red	12:00:42	10:38:16
3	5.9(b)	blue	13:11:34	10:32:36
4	5.9(c)	red	13:16:00	06:33:53
5	5.9(c)	blue	15:26:09	06:24:38
6	5.9(c)	magenta	15:26:49	06:21:35

is over at 05.00. This increases the travel time, but yields a route with a lower driving time. However, there are multiple routes through Switzerland. With increasing route number as in Table 5.7, the driver gets nearer to his destination before he waits at the Swiss border, but he has to drive a greater detour. This effect can be seen on a large-scale in Figures 5.9(a) – 5.9(c), but also on a much smaller scale:

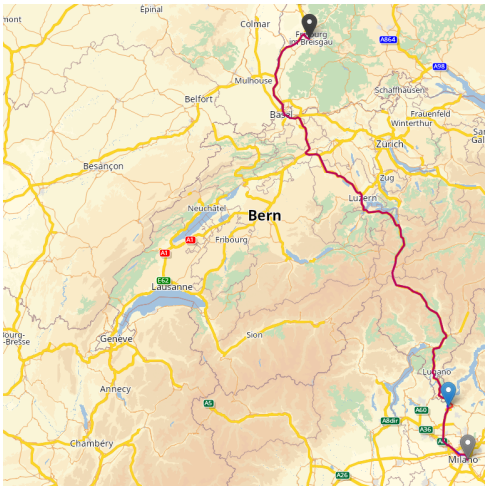
Consider Figure 5.9(d). The driver arrives from the south with either the red, blue or magenta route. The driver can now drive along on the highway where there is a temporary road-closure because of which he had to wait at the start location (magenta route). In contrast, the driver can also leave the highway to avoid the guarded segment completely (red route).

Another possibility is the blue route: The driver leaves the highway just before the first highway segment begins where the road-closure applies. Then he waits at a nearby parking lot (blue marker) until the following highway segment opens at 07.00 and the driver enters the highway again. The driver then has a higher driving time due to the detour to the parking lot, but when the segment opens, he already has skipped a short closed highway segment. In comparison to the magenta route where the driver does not leave the highway, the driver is about one minute nearer to his destination.

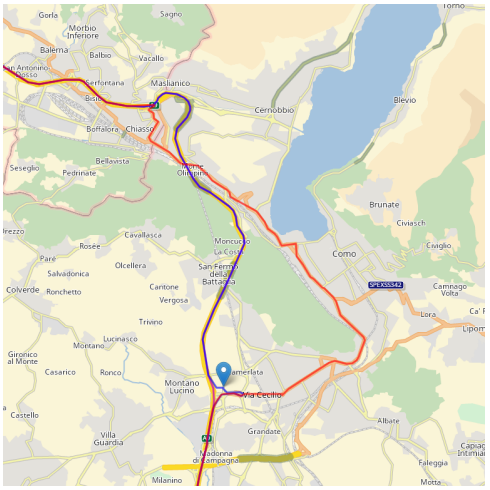
Apart from the deviations described here, the example routes sometimes have other minor deviations, as we can see in Figure 5.9(e). We do not discuss these here as they only have a minor impact on the travel time and driving time compared to the deviations shown above.

(a) Route 1 without waiting times.

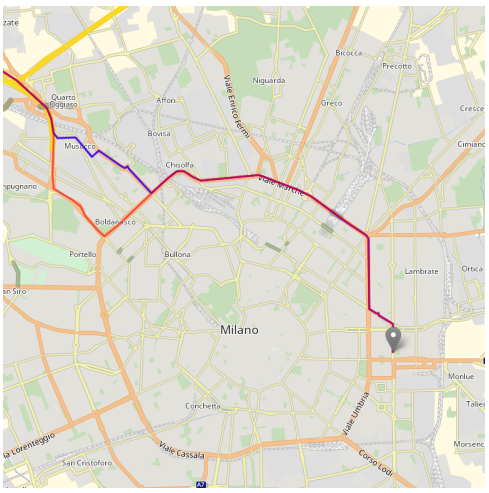
(b) Route 2 and 3.



(c) Route 4, 5 and 6.



(d) Closeup of a difference of route 4, 5 and 6.



(e) Closeup of a difference of route 4, 5 and 6. The magenta route is completely concealed by the blue route.

Figure 5.9: The six routes of a rank 2^{22}-query from Milano, Italy to Freiburg im Breisgau, Germany. The start is at the grey marker, the destination is at the black marker. The colored markers indicate the location of parking lots, where the driver waits.

6. Conclusion

We summarize the results obtained in the thesis and give a recommendation for practical usage. Thereafter, we give an outlook for future work in this topic.

6.1 Summary

In this thesis, we considered route planning with temporary road closures, where a graph has guarded edges and where waiting is only allowed at parking lot vertices or at the start vertex of a query. We showed that calculating mathematically shortest routes is $\mathcal{NP}$-hard and does not satisfy the demand of route planning in practice.

We introduced the detour-free route model, which defines the properties of routes that yield reasonable results. Based on this definition, we developed an algorithm that calculates all Pareto-optimal detour-free routes in terms of travel time and driving time.

Next we focused on speed up techniques for the detour-free route algorithm. We adapted the Contraction Hierarchy preprocessing, and presented a way to calculate upper bounds for the results of a query. These can be used for further speeding up the running time of the algorithm, or even completely avoiding a costly computation. We specified heuristics, which answer queries in several seconds with good quality.

From a practical point of view, it may suffice to calculate the upper bounds on travel time and on driving time and return these as the answer to a query. The bounds-calculation has acceptable performance and in the majority of the queries, the bounds are successfully calculated. Often, there is only one Pareto-optimal result. In the other case of multiple resulting labels, the upper bounds correspond to at least two results: The result with the earliest arrival time, and with the lowest driving time.

6.2 Future work

First of all, improving the running time of the detour-free route algorithm and the heuristics could be possible. In particular, speeding up the search in the core is a topic, where additional time should be invested in. Adapting known speed up techniques might be sufficient for this.

Furthermore, a method could be developed, where periodical road-closures are taken into account, without computing a whole new Contraction Hierarchy for each new planning

horizon. This method can be used for weekend and night driving bans. However, not all driving bans are perfectly regular. There are driving bans on holiday like easter, which follow complex rules.

A possibility is to develop a customizable version of the Contraction Hierarchy presented in this thesis. Then real-time information about road-closures due to accidents could be used. Also the entrances to overcrowded parking lots at the highway could be closed depending on real-time data.

As many road-closures affect only trucks, it also seems to be rational to integrate a truck driver scheduling algorithm. Then the legal driving and rest periods of truck drivers could be taken into account.

Bibliography

[Arm18] Donald Armour, editor. *European Road Freight Guide*. Freight Transport Association Limited, 23rd edition, April 2018.

[BDG+15a] Hannah Bast, Daniel Delling, Andrew Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F. Werneck. Route planning in transportation networks. Technical report, ArXiv e-prints, 2015.

[BDG+15b] Moritz Baum, Julian Dibbelt, Andreas Gemsa, Dorothea Wagner, and Tobias Zündorf. Shortest feasible paths with charging stops for battery electric vehicles. In *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*, SIGSPATIAL '15, pages 44:1–44:10, New York, NY, USA, 2015. ACM.

[BGSV13] Gernot Veit Batz, Robert Geisberger, Peter Sanders, and Christian Vetter. Minimum Time-Dependent Travel Times with Contraction Hierarchies. *ACM Journal of Experimental Algorithmics*, 18(1.4):1–43, April 2013.

[Dij59] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, Dec 1959.

[Dre69] Stuart E. Dreyfus. An Appraisal of Some Shortest-Path Algorithms. *Operations Research*, 17(3):395–412, 1969.

[GSSD08] Robert Geisberger, Peter Sanders, Dominik Schultes, and Daniel Delling. Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks. In *Proceedings of the 7th Workshop on Experimental Algorithms (WEA'08)*, volume 5038 of *Lecture Notes in Computer Science*, pages 319–333. Springer, June 2008.

[Han80] Pierre Hansen. Bicriterion path problems. In Günter Fandel and Tomas Gal, editors, *Multiple Criteria Decision Making Theory and Application*, pages 109–127, Berlin, Heidelberg, 1980. Springer Berlin Heidelberg.

[Kar72] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

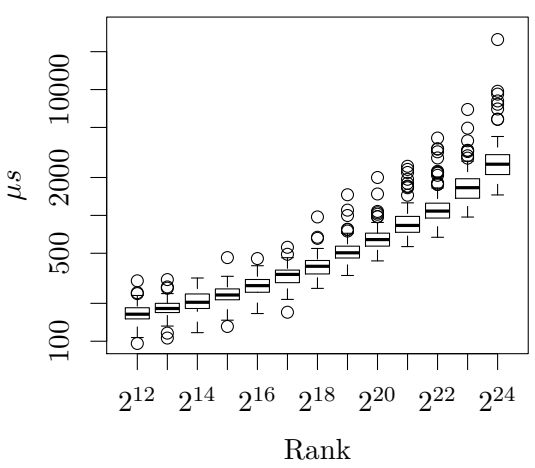
[LHDZ17] Lei Li, Wen Hua, Xingzhong Du, and Xiaofang Zhou. Minimal on-road time route scheduling on time-dependent graphs. *Proc. VLDB Endow.*, 10(11):1274–1285, August 2017.

[OR89] Ariel Orda and Raphael Rom. Traveling without Waiting in Time-Dependent Networks Is NP-hard. March 1989.

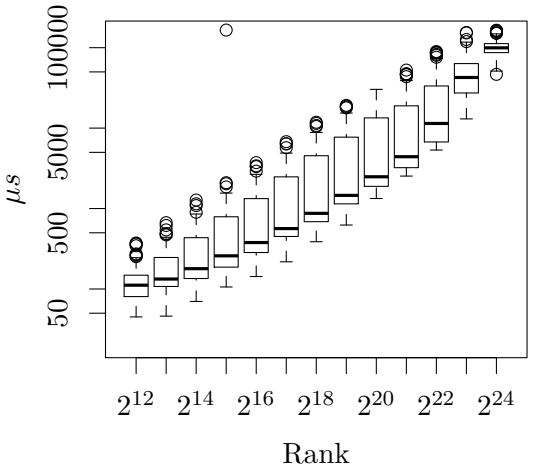
[OR91] Ariel Orda and Raphael Rom. Minimum Weight Paths in Time-Dependent Networks. *Networks*, 21:295–319, 1991.

[Str17] Ben Strasser. RoutingKit. GitHub `https://github.com/RoutingKit/RoutingKit/`, November 2017. Commit 4342bfd.

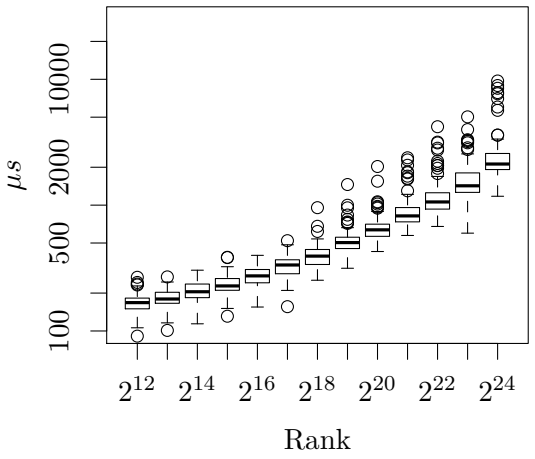
Appendix



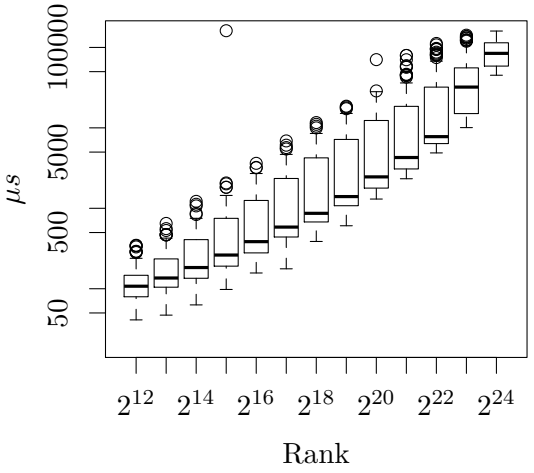
(a) Running time for the upper bound on travel time for test set 2.



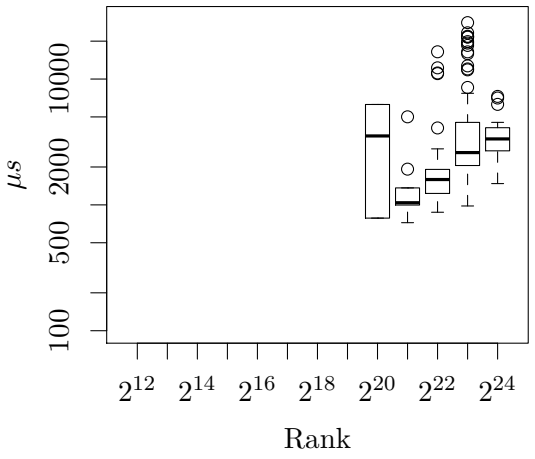
(b) Running time for the upper bound on driving time for test set 2.



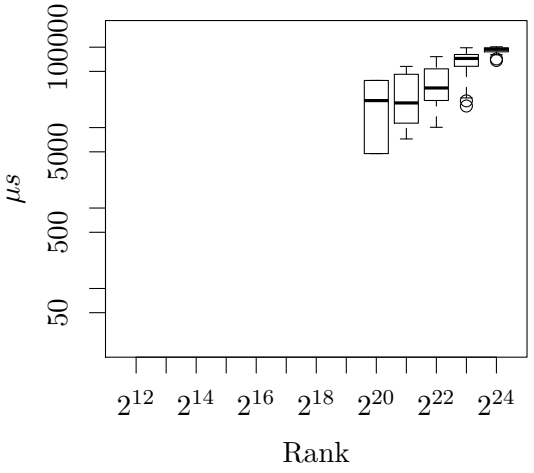
(c) Running time for the upper bound on travel time for test set 3.



(d) Running time for the upper bound on driving time for test set 3.

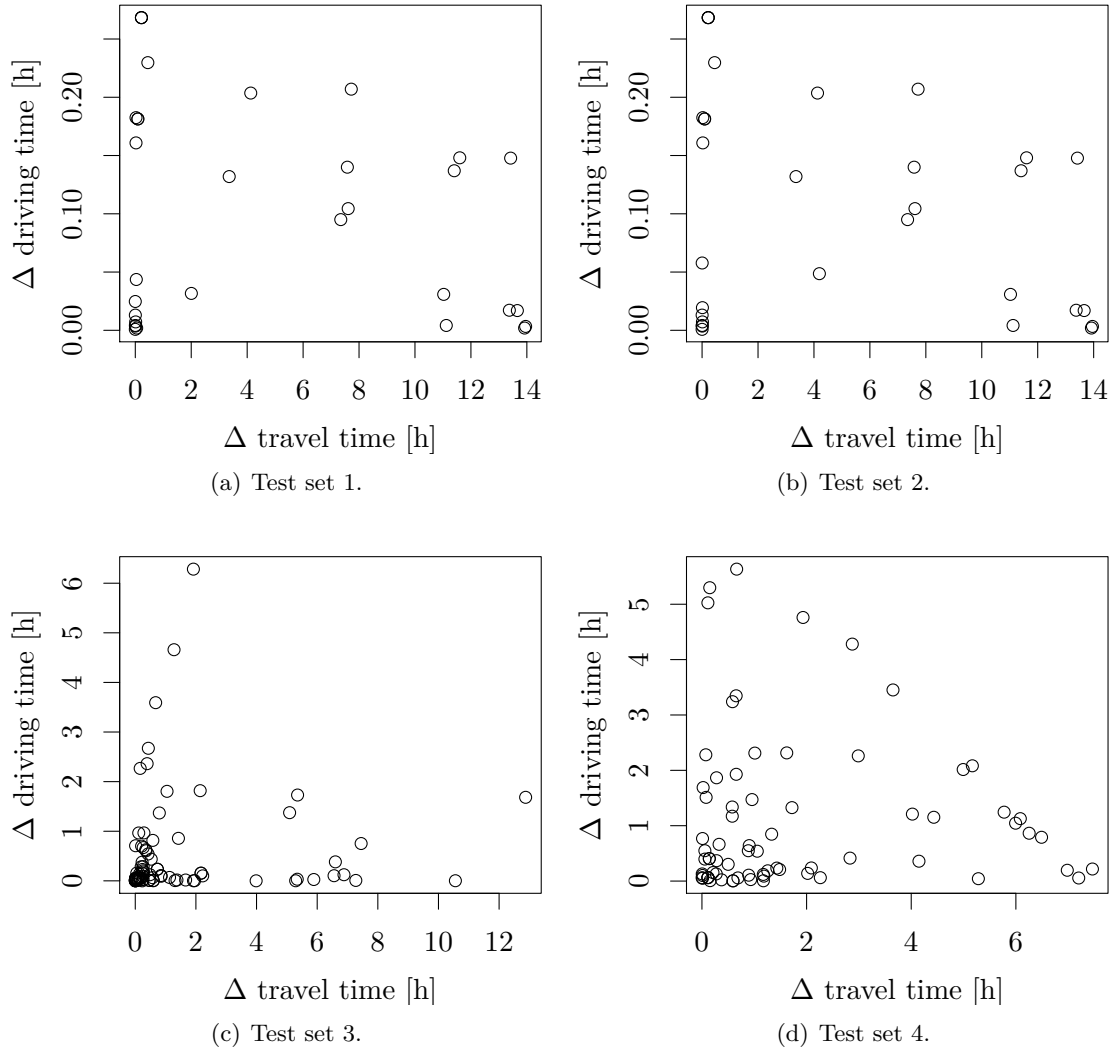


(e) Running time for the upper bound on travel time for test set 4.



(f) Running time for the upper bound on driving time for test set 4.

Figure A.1: Running time for the upper bounds for the test sets 2 – 4. The measurements for test set 1 are shown in Figure 5.2.



(a) Test set 1.

(b) Test set 2.

(c) Test set 3.

(d) Test set 4.

Figure B.1: Results of Heuristic 2. For two consecutive labels $(t^1_{travel}, t^1_{driving}), (t^2_{travel}, t^2_{driving})$ in the resulting set of a query with $t^1_{travel} < t^2_{travel}$, there is a point $(t^2_{travel} - t^1_{travel}, t^1_{driving} - t^2_{driving})$ in the plot with the corresponding test set.