Efficient Route Planning with Temporary Driving Bans, Road Closures, and Rated Parking Areas

Alexander Kleff
PTV Group, Karlsruhe, Germany
alexander.kleff@ptvgroup.com

Frank Schulz
PTV Group, Karlsruhe, Germany
frank.schulz@ptvgroup.com

Jakob Wagenblatt
Karlsruhe Institute of Technology, Germany
jakob.wagenblatt@student.kit.edu

Tim Zeitz
Karlsruhe Institute of Technology, Germany
tim.zeitz@kit.edu

Abstract

We study the problem of planning routes in road networks when certain streets or areas are closed at certain times. For heavy vehicles, such areas may be very large since many European countries impose temporary driving bans during the night or on weekends. In this setting, feasible routes may require waiting at parking areas, and several feasible routes with different trade-offs between waiting and driving detours around closed areas may exist. We propose a novel model in which driving and waiting are assigned abstract costs, and waiting costs are location-dependent to reflect the different quality of the parking areas. Our goal is to find Pareto-optimal routes with regards to arrival time at the destination and total cost. We investigate the complexity of the model and determine a necessary constraint on the cost parameters such that the problem is solvable in polynomial time. We present a thoroughly engineered implementation and perform experiments on a production-grade real world data set. The experiments show that our implementation can answer realistic queries in around a second or less which makes it feasible for practical application.

2012 ACM Subject Classification Theory of computation → Shortest paths; Mathematics of computing → Graph algorithms; Applied computing → Transportation

Keywords and phrases driving bans, realistic road networks, route planning, shortest paths

Digital Object Identifier 10.4230/LIPIcs.SEA.2020.17

Related Version A full version of the paper is available at `https://arxiv.org/abs/2004.09163`.

1 Introduction

Many European countries impose temporary driving bans for heavy vehicles. Driving may be restricted during the night, on weekends, and on public holidays. Such bans may apply to the whole road network of a country or parts of it. When routing a heavy vehicle from a source to a destination, it is crucial to take these temporary driving bans into account. But it is not only about heavy vehicles. Temporary closures of bridges, tunnels, border crossings, mountain pass roads, or certain inner-city areas as well as closures due to roadworks may affect all road users alike. In case of road space rationing in cities, the driving restriction may depend on the license plate number. To sum up, temporary driving restrictions exist in different forms, and the closing and re-opening times of a road segment must be considered in the route planning.

18th International Symposium on Experimental Algorithms (SEA 2020).
Editors: Simone Faro and Domenico Cantone; Article No. 17; pp. 17:1–17:13

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

As a consequence of temporary driving restrictions, waiting times may be inevitable and even last for hours. During such waiting hours, the vehicle must be parked properly, and thus a suitable parking area has to be found. The driving time of the detour from and to such a parking area should also be incorporated in the route planning. Unfortunately, the underlying shortest (here: quickest) path problem becomes *NP*-hard if waiting is only allowed at dedicated locations [14]. This is because in this case, the so-called *FIFO* (first in, first out) property is not satisfied, that is, the property that a driver cannot arrive earlier by departing later. Thus, our first research question is how we can consider dedicated waiting locations without making the underlying problem *NP*-hard. It is our aim to obtain a feasible running time even for long-distance routes.

In practice, we often find that small parking areas without any facilities like public toilets or restaurants cause the least detour. So an algorithm that looks for the shortest route, that is, a route with the shortest driving time, would select small parking areas in these cases, provided that waiting is necessary. But the longer the waiting time is, the more vital a secure and pleasant place for waiting becomes. So it may be important for the driver that nearby facilities of the parking area and their quality are somehow taken into account as well. How to do this is our second research question.

In our setting, a single-criterion objective is not practical. A driver may not always be in favor of the shortest route if that means to spend a very long time waiting and to arrive at the destination considerably later than on the quickest route, that is, a route with the earliest arrival at the destination. Conversely, a driver may not always be interested in a quickest route if that route means to take an unjustified long detour around temporarily closed road segments that could be avoided by waiting in a comfortable place. In other words, an early arrival at the destination (and thus low opportunity costs), little driving time (and thus low fuel costs), and pleasant waiting conditions (and thus high driver satisfaction) are competing criteria. Solutions can differ significantly with regards to these criteria. How to deal with this and find reasonable routes is the third research question.

In this paper, we answer these questions as follows:

1. We present a model in which waiting is allowed at any vertex and any edge at any time in the road graph but waiting on edges and waiting on those vertices that do not correspond to parking areas is penalized. This is done by assigning a cost to time spent waiting there. Since driving comes at a price, too, we also assign a cost per time unit spent driving. As we will show, we can find a route with least costs in polynomial time if both cost parameters are set to the same value.
2. We assume that the nearby facilities of a parking area and their quality can be expressed by some single rating number. To take account of this, we assign a waiting cost to every corresponding vertex as well. This cost is lower than the cost of waiting anywhere else in the road graph, and it is even lower the higher the rating of the parking area is.
3. We return routes that are Pareto-optimal with regards to arrival time at the destination on the one hand and total costs on the other. Despite the potentially larger output, our algorithm still runs in polynomial time under the same condition as before.

As our experiments reveal, many queries within Europe are answered within milliseconds. Except some pathological cases, even more complex queries with four or more Pareto-optimal solutions are solved in less than a second.

Related Work. Many route planning problems are modeled as shortest path problems. To this day, the theoretically fastest known algorithm to find shortest paths on graphs with static non-negative edge weights is the algorithm of Dijkstra [9]. However, for many practical

applications, it is not fast enough. One approach to speed up the computation is to reduce the search space of Dijkstra's algorithm by guiding the search towards the destination by means of estimates of the remaining distance to the destination. It is known as the A* algorithm [12]. Since the advent of routing services, a lot of research has been done on efficient algorithms for routing in road networks. Routing services have to answer many queries on the same network. This can be used to speed up shortest path queries through precomputed auxiliary data. Many approaches exploit certain characteristics of road networks, for example the hierarchical structure (freeways are more important than rural roads). For an extensive overview, we refer to [1]. One particularly popular speed-up technique are Contraction Hierarchies [11]. During preprocessing, additional shortcut edges are inserted into the graph, which skip over unimportant vertices. This preprocessing typically takes a few minutes. Then, shortest path queries can be answered in less than a millisecond.

A natural approach to handle driving restrictions is to model them as time-dependent travel times [10]. For the blocked time, the travel time of the edge can be set to infinity. Time-dependent route planning has also received some attention and effective speed-up techniques are known [2, 3, 6, 5, 13].

Variants of our problem have been studied in the literature. In [7] a related problem is discussed where nodes (not edges) have time windows and waiting is associated with a cost. In [15] an overview is given over different exact approaches to solving shortest path problems with resource constraints. Time windows on nodes are a specific kind of constraint in this framework. More specialized models for routing applications have been proposed. The authors of [17] study the problem of planning a single break, considering driving restrictions and provisions on driver breaks. They aim to find only the route with the earliest arrival.

Contribution. We present a novel model that helps answer our three research questions in the context of temporary driving restrictions and dedicated waiting locations. To the best of our knowledge, this is the first unifying approach that gives answers to all three research questions. Our theoretical analysis reveals that our model can be solved to optimality in polynomial time, given certain restrictions on the parameterization. The experimental evaluation of our implementation demonstrates a practical running time.

Outline. In Section 2, we give a formal definition of the routing problem at hand. In Section 3, we present an exact algorithm for this problem. In Section 4, we analyze the complexity of the problem and show that our algorithm runs in polynomial time if the costs for driving are the same as for waiting anywhere else than at a dedicated waiting location. In Section 5, we describe techniques to speed-up the computation. In Section 6, we present the main results of our experiments. Finally, we conclude in Section 7.

2 Problem

A problem instance comprises a *road graph with ban intervals on edges, driving costs and location-dependent waiting costs* (or *road graph with ban intervals and costs* for short) as well as a set of *queries*. The road graph is characterized by the following attributes:

- A set V of n vertices and a set E of m directed edges.
- A mapping Φ that maps each edge $e \in E$ to a sequence of disjoint time intervals, where the edge is considered to be *closed* during each interval. Precisely, for any *ban interval* $[t^{closed}, t^{open}) \in \Phi(e)$ of an edge e, t^{open} denotes the first point in time after t^{closed} where the edge is open again. Here and in the following, all points in time are integers and the

length of an interval is denoted by $|[t^{closed}, t^{open})|$ and equals $t^{open} - t^{closed} > 0$. During such a time span, a vehicle on the corresponding road segment must not move. We denote the total number of ban intervals as b.

- A mapping $\delta : E \to \mathbb{N}$ that maps each edge $e := (u, v) \in E$ to the time $\delta(e)$ that it takes to drive from u to v, provided the edge is open.
- A mapping ρ that maps each vertex to a rating in $\{0, 1, \ldots, r\}$ with $r \leq n$. Rating 0 means *unrated*, that is, it is assumed that it is highly difficult, dangerous, and not allowed to park the vehicle there. In contrast to an unrated location, we call a vertex v with $\rho(v) > 0$ a *parking location*.
- A parameter set of abstract costs, consisting of $d \in \mathbb{Q}_{\geq 0}$, the cost per unit of driving time, and $w_i \in \mathbb{Q}_{\geq 0}$ for all i from 0 to r, the cost per time unit of waiting on a vertex with rating i. Edges are always unrated so waiting there costs w_0 per time unit. W.l.o.g. $w_i < w_{i-1}$ holds for all i between 1 and r, that is, we assume that waiting on vertices with a higher rating costs less than waiting on those with a lower rating.

A *u-v-route* is a triple (R, A, D) of three sequences of the same length $\ell := |R| = |A| = |D|$. Here, R is the sequence of vertices along the route. It describes a (not necessarily simple) *path* in the graph that starts at u and ends in v, that is, $e_i := (R[i], R[i+1]) \in E$ for all $1 \leq i < \ell$ and $R[1] = u$ and $R[\ell] = v$. The other two sequences A and D denote the *arrival times* and the *departure times* from the respective vertices, where $A[i] \leq D[i]$ for all $1 \leq i \leq \ell$ and $A[i+1] - D[i] \geq \delta(e_i)$ for all $1 \leq i < \ell$ holds.

A query comprises a *source* $s \in V$ and a *destination* $z \in V$ as well as a *planning horizon* H. The latter is defined as the time interval between an *earliest departure time* t^{min} from s and a *latest arrival time* t^{max} at z. Waiting costs arise as soon as the planning horizon opens. For a given query, we look for *feasible* s-z-routes. A route is feasible with respect to the planning horizon if $A[1] = t^{min}$ and $D[\ell] \leq t^{max}$. In addition, ban intervals must be taken account of. Let $T_i := [D[i], A[i+1])$ be the time interval in which the edge $e_i := (R[i], R[i+1])$ of the route's path is traversed. A route is feasible with respect to the ban intervals if $\sum_{I \in \Phi(e_i)} |T_i \cap I| \leq |T_i| - \delta(e_i)$ for all $1 \leq i < \ell$. Here, $\sum_{I \in \Phi(e_i)} |T_i \cap I|$ is the time during which the edge between $R[i]$ and $R[i+1]$ is closed while the edge is being traversed.

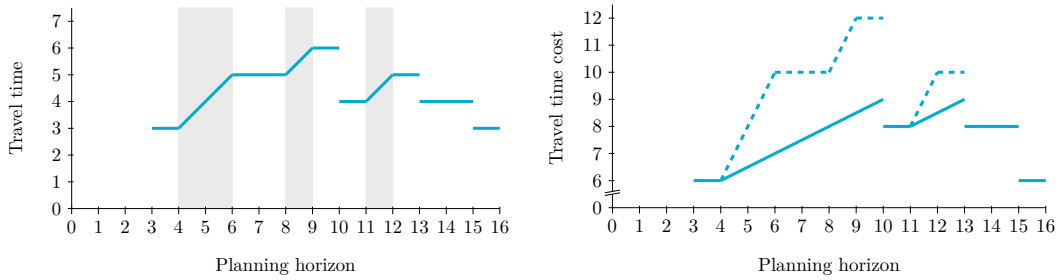
Let *travel time* include driving time and waiting time. The *travel time costs* of a route are the sum of the waiting time costs and the driving time costs. So given a route of length ℓ, the travel time costs are

$$\sum_{i=1}^{\ell} w_{\rho(R[i])} \cdot (D[i] - A[i]) + \sum_{i=1}^{\ell-1} w_0 \cdot (A[i+1] - D[i] - \delta(e_i)) + d \cdot \delta(e_i),$$

where we use $e_i := (R[i], R[i+1])$. We say an s-z-route is *Pareto-optimal* (or simply *optimal*) if it is feasible and if its travel time costs are less or its arrival time at z is earlier or equality holds in both cases compared to any other feasible s-z-route. For a query, the objective is to find a maximal set of (Pareto-)optimal s-z-routes such that no two routes in the set have both the same arrival time at z and the same travel time costs.

3 Algorithm

The algorithm maintains a priority queue. Each entry of the queue consists of a vertex and a point in time within the planning horizon as key. We say a vertex is *visited* at a certain point in time whenever we remove the top entry from the queue, that is, an entry with the earliest time among the entries in the queue. At every vertex $v \in V$, we store a time-dependent



(a) Travel time function $\mathcal{T}_e$ of an edge e with ban intervals (grey) and a driving time $\delta(e)$ of 3. The latest departure to be at v at time t is $t - \mathcal{T}_e(t)$.

(b) Cost profile of vertex v after linking, that is, after considering travel time (dashed) and waiting time at v (solid).

Figure 1 Computing the cost profile of a vertex v. Let v be adjacent to the source s via an edge $e := (s, v)$ with three ban intervals and a driving time $\delta(e)$ of 3. The corresponding travel time function is given in Figure 1a. It is infinite between $0 = t^{min}$ and $3 = \delta(e)$. In Figure 1b, we see the cost profile $\mathcal{C}_v$ after considering the travel time along the edge (dashed) and after considering waiting at v (solid). Here, the assumed cost parameters are $w_{\rho(s)} = 0$, $w_{\rho(v)} = 0.5$, and $d = w_0 = 2$, where $w_{\rho(s)} = 0$ implies that the cost profile $\mathcal{C}_s$ at the source is 0 over the whole planning horizon.

function $\mathcal{C}_v : H \to \mathbb{Q}_{\geq 0} \cup \{\infty\}$. It maps a point in time t within the planning horizon H to an upper bound on the minimum travel time cost over all s-v-routes that end in v at time t. We call this function *cost profile* of v or, more general, *label* of v. The algorithm works in a *label correcting* manner in the sense that a vertex may be visited multiple times, albeit at different times within the planning horizon.

Before we describe the phases of the algorithm in greater detail, we introduce an auxiliary time-dependent function $\mathcal{T}_e$ for every edge $e \in E$. It maps a time t at the head v of an edge $e := (u, v)$ to the *shortest travel time* that it takes to traverse the edge from u to v completely and be at v at time t, possibly including waiting time. That is, for a time t at v, $\mathcal{T}_e(t)$ is the minimum period p such that $p - \sum_{I \in \Phi(e)} |[t-p, t) \cap I| \geq \delta(e)$ holds if such a p exists, and ∞ otherwise. In other words, $t - \mathcal{T}_e(t)$ is the latest departure time from u in order not to arrive at v later than at time t. An example is given in Figure 1a.

In the initialization phase of the algorithm, we set $\mathcal{C}_s(t) := w_{\rho(s)} \cdot (t - t^{min})$ for all $t \in H$. For every other $v \in V \setminus \{s\}$, we set $\mathcal{C}_v(t) := \infty$ for all $t \in H$. Furthermore, we insert the source s with key t^{min} into the priority queue.

As long as the queue is not empty, we are in the main loop of the algorithm. In every iteration of the main loop, we remove the top entry from the queue. Let us suppose we visit a vertex u at time $t^{visit} \geq t^{min}$. Then, we check for every edge $e := (u, v)$ going out of u whether we can improve the cost profile $\mathcal{C}_v$ of v. We do so in three steps. In the first step, we consider the travel time along the edge and set

$$\mathcal{C}'_v(t) := \mathcal{C}_u(t - \mathcal{T}_e(t)) + d \cdot \delta(e) + w_0 \cdot (\mathcal{T}_e(t) - \delta(e)) \tag{1}$$

for all t with $t^{visit} + \mathcal{T}_e(t) \leq t \leq t^{max}$. For all other $t \in H$ we set $\mathcal{C}'_v(t) := \infty$. In the second step, we consider waiting at v at cost $w_{\rho(v)}$ per time unit and set

$$\mathcal{C}'_v(t) := \min\{\mathcal{C}'_v(t') + w_{\rho(v)} \cdot (t - t') \mid t^{min} \leq t' \leq t\} \tag{2}$$

for all $t \in H$. An example of the first two steps is illustrated in Figure 1b. Finally, in the third step, we compare $\mathcal{C}'_v$ and $\mathcal{C}_v$. Let t^* be the earliest point in time such that $\mathcal{C}'_v(t^*)$ is less than $\mathcal{C}_v(t^*)$ if such a time t^* exists. Only if it exists, we set $\mathcal{C}_v(t)$ to the minimum of $\mathcal{C}_v(t)$ and $\mathcal{C}'_v(t)$ for all $t^* \leq t \leq t^{max}$. Furthermore, we insert vertex v with key t^* into the priority queue or decrease the key if v is already contained.

When the priority queue is empty, we enter the finalization phase of the algorithm. We say a time-cost-pair $(t, \mathcal{C}_z(t))$ with $t \in H$ and $\mathcal{C}_z(t) < \infty$ is Pareto-optimal if there is no time t' with $t^{min} \leq t' < t$ and $\mathcal{C}_z(t') \leq \mathcal{C}_z(t)$. In the finalization phase, we extract an s-z-route for every Pareto-optimal time-cost-pair. So let such a time-cost-pair $(t, \mathcal{C}_z(t))$ be given. In order to find a corresponding route (R, A, D), we initially push z and t and t to the front of the (empty) sequences R and A and D, respectively. The following is done iteratively until we reach the source, that is, $R[1] = s$ holds. First, we look for an incoming edge $e := (u, R[1])$ of $R[1]$ and a departure time t from u with

$$\mathcal{C}_u(t) + d \cdot \delta(e) + w_0 \cdot (\mathcal{T}_e(A[1]) - \delta(e)) = \mathcal{C}_{R[1]}(A[1])$$

which must exist. We push u and t to the front of R and D, respectively. Then, we push the earliest time $t \leq D[1]$ such that

$$\mathcal{C}_{R[1]}(t) + w_{\rho(R[1])} \cdot (D[1] - t) = \mathcal{C}_{R[1]}(D[1])$$

holds to the front of the arrival time sequence A, and continue with the next iteration. This concludes the description of the finalization phase and thus the whole algorithm.

For the correctness of the algorithm it is important that the upper bound $\mathcal{C}_v(t)$ on the minimum travel time cost is tight for all $t \leq t^{visit}$ and all $v \in V$ whenever we visit a vertex at time t^{visit}. After the main loop, it is tight for every $t \in H$ and all $v \in V$, especially for z. This can be proven by induction on the time of visit. The time of visiting a vertex increases monotonically because whenever a vertex is inserted into the queue or its key is decreased, the (new) value of that key can only be later than the current time of visit.

4 Analysis

In this section, we first show the intractability of the general problem. Then, we restrict the problem by requiring the driving cost d to be equal to the unrated waiting cost w_0, and prove that our algorithm solves the restricted problem in polynomial time.

Intractability of the General Problem. The first two theorems show the intractability of the general problem if $d \neq w_0$. Parking locations are not used in the proofs, so already the simplified problem without parking locations is intractable if $d \neq w_0$.

▶ **Theorem 1.** *If $d < w_0$ then it is NP-complete to decide whether there is a feasible route with travel time costs less than or equal to a given threshold k.*

We prove this theorem in the full version of this paper by reduction from PARTITION.

▶ **Theorem 2.** *If $d > w_0$ then the number of Pareto-optimal routes can be exponential in the number of vertices.*

Given a number of vertices a graph with ban intervals can be constructed that has exponentially many routes which are all Pareto-optimal. The construction and the proof that all those routes are Pareto-optimal can be found in the full version of this paper.

Tractable Problem Variant. For the remaining analysis we assume $d = w_0$. In the setting without parking locations, there is only one optimal solution, since the quickest solution has also the least cost. Hence, this setting is a single-criterion shortest path problem with time-dependent edge weights that fulfill the *FIFO* property and can be solved in polynomial

time with a time-dependent variant of Dijkstra's algorithm [10], and also our algorithm reduces to such a time-dependent Dijkstra variant and has polynomial running time. Now we turn to the setting $d = w_0$ with parking locations and show that it is still tractable.

Cost profiles are piecewise linear functions. An important aspect of our polynomial time proof is to count the non-differentiable points of the profiles. The running time of each profile operation of our algorithm is linear in the number of non-differentiable points of the involved profiles. These points are either *convex*, *concave*, or *discontinuous*, meaning an environment around such a point exists in which the profile is convex or concave or discontinuous, respectively. In a discontinuous point, a profile is always jumping down.

The non-differentiable points in the cost profiles are induced by the travel time functions. In our example of Figure 1a, the convex points are $\{4, 8, 11\}$, the concave points are $\{6, 9, 12\}$, and the discontinuous points are $\{10, 13, 15\}$. For a travel time function $\mathcal{T}_e$ of an edge e, we can assign a convex point t to the beginning of a ban interval in t, a concave point t to the end of a ban interval in t, and a discontinuous point t to the end of a ban interval in $t - \mathcal{T}_e(t)$. From this initial assignment, we can derive a ban interval assignment of the convex or discontinuous points of cost profiles. We omit to count the number of concave points of a cost profile because every gradient of a piece must be in $\{w_0, \dots, w_r\}$, so the number of consecutive concave points in a cost profile is limited by r.

Initially, a profile $\mathcal{C}_v$ of a vertex v has no convex or discontinuous points. Such points may be introduced in the third step of an iteration of the algorithm when the auxiliary profile $\mathcal{C}'_v$ is merged into $\mathcal{C}_v$. In the second step of an iteration, no new convex or discontinuous points can arise in $\mathcal{C}'_v$, so all such points must be created in $\mathcal{C}'_v$ in the first step. Since $d = w_0$, $\mathcal{C}'_v(t)$ is set to $\mathcal{C}_u(t - \mathcal{T}_e(t)) + d \cdot \mathcal{T}_e(t)$ (compare Equation (1)) for some edge $e = (u, v)$ in this step. If t_v is a convex or discontinuous point of $\mathcal{C}'_v$, then $\mathcal{T}_e$ must be convex or discontinuous in the same point in time, or $\mathcal{C}_u$ must be convex or discontinuous in $t_u := t_v - \mathcal{T}_e(t_v)$. In the former case, t_v inherits the assignment of the same point in time in $\mathcal{T}_e$, whereas in the latter case, t_v inherits the assignment of t_u in $\mathcal{C}_u$. Since the cost profiles change during the algorithm, we do not only assign a ban interval to every convex or discontinuous point but also an iteration. Again, in the former case, t_v is assigned the current iteration, whereas in the latter case, t_v inherits the iteration assignment of t_u in $\mathcal{C}_u$.

▶ **Lemma 3.** *If $d = w_0$ then a cost profile after iteration i has at most ib convex and at most ib discontinuous points.*

Proof. In the following, we denote the state of the profile C_v after iteration i by C_v^i. Let t_v be a convex or discontinuous point of $\mathcal{C}_v^i$ that is assigned both to an iteration k and to a ban interval of some edge with head x. We can follow the inheritance relation until we finally reach a convex or discontinuous point t_x in $\mathcal{C}_x^k$. By induction, we have $\mathcal{C}_v^i(t_v) = \mathcal{C}_x^k(t_x) + d \cdot (t_v - t_x)$. Now suppose there are two convex or two discontinuous points $t_v^1 < t_v^2$ in the profile $\mathcal{C}_v^i$ that are assigned to the same ban interval and the same iteration k, so they can be traced back to the same point t_x in $\mathcal{C}_x^k$. Then the previous observation implies that $\mathcal{C}_v^i(t_v^2) - \mathcal{C}_v^i(t_v^1) = d \cdot (t_v^2 - t_v^1)$ holds, that is, the profile $\mathcal{C}_v^i$ must contain a piece with gradient d that contains both t_v^1 and t_v^2. But then t_v^2 can neither be convex nor discontinuous. Hence, two convex or two discontinuous points must differ in their assigned ban interval or their assigned iteration and there can only be ib discontinuous and convex points, respectively. ◀

▶ **Lemma 4.** *If $d = w_0$ then the total number of iterations is at most $2n(b(r + 1) + 1)$.*

As in the proof of Lemma 3 we use the ban interval assignment of convex and discontinuous points. Every visit of a vertex can either be assigned to the start or end of a ban interval, or it can be assigned to a concave point of the final cost profile of the vertex. The detailed proof is omitted due to space limitations and can be found in the full version of this paper.

▶ **Theorem 5.** *If $d = w_0$ then the running time of the algorithm is polynomial.*

Proof. From Lemma 3 with the bound from Lemma 4 it follows that the number of pieces of any profile that is constructed during the algorithm is polynomial.

We now estimate the overall running time of our algorithm: Lemma 4 states that the total number of iterations is polynomial. In every iteration of the algorithm one vertex is considered and for its outgoing edges the profiles are updated with a running time linear in the number of pieces of the profiles. The adjacent vertices are inserted into the priority queue or their keys are decreased. Since the size of the priority queue is at most the total number of vertices also the running time of the priority queue operations is polynomial. ◀

5 Implementation

The past decade has seen a lot of research effort on the engineering of efficient route planning algorithms. This section describes the speed-up techniques we employ in our implementation and some implementation details.

We store cost profiles as a sorted list of pieces. Each piece is represented as a triple: a point in time from which this piece is valid, the costs it takes to reach the vertex at the beginning of the piece and the incline of the piece. For each piece we also store a parent vertex. This allows us to efficiently reconstruct routes by traversing the parent pointers.

We employ A* to guide the search toward the destination vertex. The queue is ordered by the original key plus an estimate of the remaining distance (here: driving time) to the destination. The estimate for vertex u is denoted by $\pi_z(u)$. We use the exact shortest driving time to z without driving restrictions as the potential. This is the best possible potential in our case. We efficiently extract these exact distances from a Contraction Hierarchy [11], as described in [16]. Since our algorithm has to run until the queue is empty, we can not immediately terminate when we reach the destination. However, we get a tentative cost profile at the destination. This allows for effective pruning. Additionally, we do not need to insert a vertex u into the queue when $t^{visit} + \pi_z(u) > t^{max}$ holds, that is, we cannot reach the destination from u within the planning horizon.

We employ pruning to avoid linking and merging when possible using the following rules:

- Consider a vertex u that is visited at t^{visit}. Before relaxing any outgoing edges, we first check if u can actually contribute to any optimal route to z. If $\mathcal{C}_u(t) + \pi_z(u) \cdot d > \mathcal{C}_z(t + \pi_z(u))$ for all t with $t^{visit} \leq t < t^{max}$, u can not contribute to an optimal route to z and can thus be skipped.
- Let $\alpha(u) := \min\{t \mid \mathcal{C}_u(t) < \infty\}$ be the first point in time such that u can be reached with finite costs and ∞ if no such point exists. For each vertex u, we maintain a lower bound $\beta(u) := \min_t\{\mathcal{C}_u(t)\}$ and an upper bound $\gamma(u) := \max_{t>\alpha(u)}\{\mathcal{C}_u(t)\}$ or ∞, if there are no finite costs. They can be updated efficiently during the merge operation. An edge (u, v) only needs to be relaxed if $\beta(u) + \delta(u, v) \cdot d \leq \gamma(v)$ or $\alpha(u) + \delta(u, v) < \alpha(v)$.
- When all of the pieces of the cost profile of a vertex u share the same parent vertex v and $\rho(u) = 0$, the edge (u, v) back to the parent does not need to be relaxed as loops can never be part of an optimal route unless they include waiting at a parking location.

Table 1 Rating and default waiting cost by capacity of parking locations. The driving cost is the same as the cost for waiting at unrated vertices.

Capacity of parking locations	≥ 80	≥ 40	≥ 15	≥ 5	≥ 1	-
Rating	5	4	3	2	1	0
Default waiting costs	3	4	5	6	7	14
Number of parking locations	448	997	2 664	5 418	5 748	21.9 M

6 Experimental Evaluation

Our algorithm is implemented in C++14 and compiled with Visual C++. For the CH-potentials, we build upon the Contraction Hierarchy implementation of RoutingKit[1] [8]. All experiments were conducted on a Windows 10 Pro machine with an Intel i7-7600 CPU with a base frequency of 3.4 GHz and 32 GB of DDR4 RAM. The implementation is single-threaded.

Our experimental setup is taken from [4]. We perform experiments on a road network used in production by PTV[2]. The network is adapted from data by TomTom[3]. It covers Austria, France, Germany, Italy, Liechtenstein, Luxembourg, and Switzerland. It has 21.9 million vertices and 47.6 million edges. We use travel times, driving bans, and road closures for a truck with a gross combined weight of 40 tons. Driving bans were derived from the current legislation of the respective countries. This includes Sunday driving bans in all countries, a late Saturday driving ban in Austria and night driving bans in Austria, Liechtenstein and Switzerland. Additionally, there is a Saturday driving ban in Italy during the summer holidays. The dataset also includes several local road closures in city centers.

Parking locations were taken from data by Truck Parking Europe[4]. There is a total of 15 317 vertices classified as parking locations in our data set. The dataset also contains the capacity of each parking location. We assign each parking location a rating between 1 and 5 depending on its capacity. Table 1 shows the number of parking locations for each rating and our default waiting costs. We also evaluate different parameterizations. The waiting costs are calculated such that for an hour of waiting a detour of up to four minutes will be taken to get to a parking location rated better by one. For waiting at the source vertex of a query, we assign zero waiting costs regardless of the rating.

We generate two sets of source-destination pairs and combine them with different planning horizons. The first set is used to evaluate the practicality of our model. It is designed to make the algorithm cope with the night driving ban in Austria and Switzerland. We select 100 pairs of vertices. One vertex is randomly selected from the area around southern Germany. The other vertex is selected from the area around northern Italy. See the full version of this paper for coordinates and a visualization. We store each pair in both directions. Hence, we have 200 vertex pairs in this set. The planning horizon starts at Monday 2018/7/2, 18:00 with length one day (query set A1) and two days (A2). Figure 2 depicts an example from A1.

The second set is generated by selecting 100 source vertices uniformly at random. From each source vertex, we run Dijkstra's algorithm without a specific target ignoring any driving restrictions. Dijkstra's algorithm explores the graph by traversing vertices in increasing distance of the source vertex. We use the order in which vertices are settled to select

[1] https://github.com/RoutingKit/RoutingKit
[2] https://ptvgroup.com
[3] https://tomtom.com
[4] https://truckparkingeurope.com

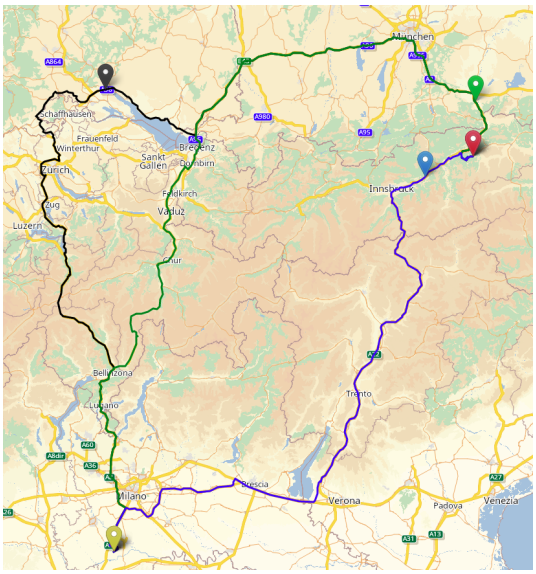


Figure 2 Optimal paths of an example query from northwestern Austria to northern Italy, slightly south of Milano. The source is indicated by a red, the destination by a yellow marker. The other markers indicate the parking locations along the respective routes. The blue route in the east has the shortest driving time, around 10.5 hours, but the latest arrival. It schedules a waiting time of seven hours during the night driving ban at a parking location of rating 4 and afterwards takes the fastest route to the destination. The green route in the middle arrives an hour earlier at the destination but the driving time is over two hours longer. This route includes three hours of waiting at a parking location of rating 5. The black route in the west takes 16 hours to drive, includes only a few minutes of waiting and arrives six minutes before the green one.

destination vertices with different distances from the source. Every 2^ith settled vertex with $i \in [12, 24]$ is stored. We denote i as the *rank* of the query. This results in 1 300 source-destination pairs. We combine these vertex pairs with four planning horizons: starting at Friday 2018/7/6, 06:00 for one day (denoted as query set B1), for two days (B2) and starting later that day at 18:00 for one day (B3) and for two days (B4).

We first investigate whether allowing waiting everywhere (albeit penalized) may lead to unwanted results in practice. On the one hand, routes with many stops are impractical. Our experiments indicate that this is not the case: Accross all routes for A1, there is at most one additional stop scheduled (0.2 on average). On the other hand, let us call a route *precarious* if waiting is scheduled at an unrated location (other than the source vertex). For 187 of the 200 queries of A1, there is no precarious route in the Pareto set. For the other 13 queries, the Pareto set always contains more than one route, and it is always only the quickest route in the Pareto set that is precarious. So filtering out such routes in a postprocessing step does not make a query infeasible. On average, the second quickest route in the Pareto set arrives 422 s later than the quickest but precarious route (minimum 38 s, maximum 877 s).

We also evaluate the influence of different waiting cost parameterizations on the performance and the results of our algorithm. Table 2 depicts the results. We observe that the parametrization has only limited influence on the results of the algorithm. The average number of optimal routes and the arrival time deviation change only very little even between the two most extreme configurations. Since waiting at the source vertex costs nothing, the majority of the waiting in all configurations is scheduled there. When waiting at parking

Table 2 Query statistics for different waiting cost parameters for query set A1. The first six columns show the waiting cost parameters. Waiting costs at the source are always set to zero. The waiting time columns depict the share of the time spent waiting at vertices with the respective rating summed up over all routes. The routes column gives the average number of optimal routes per query. The arrival time deviation column contains the average of the difference between earliest and latest arrival time among all optimal routes for all queries. Running times are also averaged.

						Waiting time by rating [%]							Optimal Routes	Arrival time deviation	Running time
w_5	w_4	w_3	w_2	w_1	$w_0 = d$	s	5	4	3	2	1	0	[#]	[h:mm]	[ms]
1	10	50	100	1000	10000	59.4	2.5	5.8	23.3	3.1	2.8	3.1	3.02	2:21	364.1
1	2	4	8	16	128	62.2	3.5	6.5	19.8	2.1	2.8	3.1	3.02	2:20	412.3
1	2	4	8	16	32	70.8	6.0	5.1	12.1	1.0	1.9	3.1	2.96	2:20	435.4
3	4	5	6	7	14	79.3	6.2	3.1	4.6	1.5	2.0	3.3	2.86	2:17	529.4
16	24	28	30	31	32	85.2	4.6	1.1	3.3	1.1	1.1	3.6	2.71	2:14	742.2

locations is much cheaper than driving, less waiting time will be scheduled at the source and more waiting at parking locations. Also, clear differences between the costs lead to a better running time, because cost profiles become less complex.

We next investigate the algorithm's performance for each of the different query sets. We report the same numbers limited to non-trivial queries. A query is denoted as *trivial* if there is exactly one optimal route which is also optimal when ignoring all driving restrictions. Table 3 depicts the results. Clearly, the query set has a strong influence on the running time of the algorithm. Average running times range from ten milliseconds to one second when looking at all queries. However, median query times are significantly smaller. The reason for this is that our algorithm can answer trivial queries in a few milliseconds or less. Due to the perfect potentials, the algorithm only traverses the optimal path. Once the destination is reached, because of the target pruning, all other vertices in the queue are skipped and the algorithm terminates. Excluding trivial queries, we get a clearer picture of the algorithm's performance when solving the harder part of the problem.

For the query sets B1 and B2, only 4% to 5% of the queries have to deal with driving restrictions. This is mostly due to closures for individual roads in certain cities and not country-wide driving bans. When the planning horizon begins later at 18:00 (B3 and B4), we get around twice as many non-trivial queries. These are primarily caused by the night driving bans in Austria and Switzerland. Road closures and country-wide driving bans lead to different optimal routes. When there is a road closure on the shortest path ignoring any driving restrictions, we often have two optimal routes. One which takes a (small) detour around the closure, and one waiting at the source until the closed road opens and then taking that slightly shorter path. Thus, we have two routes with very similar driving times but (often vastly) diverging arrival times. When dealing with night driving bans, we get more optimal results with different trade-offs as in the example of Figure 2.

Increasing the length of the planning horizon to two days leads to more non-trivial queries, more optimal routes per query, and a greater deviation in arrival time. The reason are routes with a travel time longer than 24 hours which were not valid for the shorter planning horizon.

Even when we restrict ourselves to queries with non-trivial results, running times still vary depending on the query set. Average and median deviate not as strong as when considering all queries, but the distribution of running times is still skewed by a few long running queries,

Table 3 Query statistics for all six query sets. First, for all queries. Second, only for non-trivial queries. A query is denoted as trivial if there is exactly one optimal route which is also optimal when ignoring all driving restrictions. All numbers are averages unless reported otherwise. The arrival time deviation column contains the average of the difference between earliest and latest arrival time among all optimal routes for all queries. The routes column contains the number of optimal routes.

	Set	Planning horizon	Query share [%]	Optimal Routes [#]	Arrival time deviation [h:mm]	Running time Avg. [ms]	Running time Median [ms]
	A1	Mon. 18:00, 1 day	100.0	2.86	2:17	529.4	266.3
	A2	Mon. 18:00, 2 days	100.0	3.54	3:19	648.1	405.6
	B1	Fri. 06:00, 1 day	100.0	1.04	0:10	10.0	0.6
	B2	Fri. 06:00, 2 days	100.0	1.08	0:16	79.5	0.7
	B3	Fri. 18:00, 1 day	100.0	1.13	0:08	205.8	0.6
	B4	Fri. 18:00, 2 days	100.0	1.32	0:20	1 028.1	0.7
Only non-trivial	A1	Mon. 18:00, 1 day	67.5	3.82	3:13	764.1	560.6
Only non-trivial	A2	Mon. 18:00, 2 days	72.0	4.53	4:37	899.2	655.0
Only non-trivial	B1	Fri. 06:00, 1 day	4.1	2.19	4:10	42.5	6.6
Only non-trivial	B2	Fri. 06:00, 2 days	4.8	2.76	5:43	1 105.6	35.8
Only non-trivial	B3	Fri. 18:00, 1 day	9.2	2.73	1:25	1 359.0	475.2
Only non-trivial	B4	Fri. 18:00, 2 days	11.6	3.79	2:51	5 819.4	1 947.2

especially on set B4. The reason for this is that the running time heavily depends on the types and lengths of driving restrictions in the search space. The Saturday driving ban in Italy causes heavy outliers in B4 (but also B2 and B3), when the destination lies in an area blocked for most of the planning horizon. This causes the algorithm to explore large parts of the graph, until the driving ban is over. The worst of these queries took 49 seconds to answer. Nevertheless, when looking at query sets A1 and A2, we clearly see that the algorithm can answer queries affected by country-wide night driving bans in less than a second.

7 Conclusion

We have introduced a variant of the shortest path problem where driving on edges may be forbidden at times, both driving and waiting entail costs, and the cost for waiting depends on the rating of the respective location. The objective is to find a Pareto set of both quickest paths and minimum cost paths in a road graph. We have presented an exact algorithm for this problem and shown that it runs in polynomial time if the cost for driving is the same as for waiting in an unrated location. With this algorithm, we can solve routing problems that arise in practice in the context of temporary driving bans for trucks as well as temporary closures of roads or even larger parts of the road network.

Our experiments demonstrate that our implementation can answer queries with realistic driving restrictions in less than a second on average. There are a few slow outlier queries when the destination vertex lies in a blocked area. A promising angle to improve this could be to study bidirectional variants of our algorithm. We exploit Contraction Hierarchies to efficiently obtain good A* potentials. The algorithm can also be used in a dynamic (or live or online) scenario when combined with Customizable Contraction Hierarchies [8]. A natural extension of our problem at hand is to consider time-dependent driving times or rules for truck drivers that enforce a break after a certain accumulated driving time.

References

1 Hannah Bast, Daniel Delling, Andrew V. Goldberg, Matthias Müller–Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F. Werneck. Route Planning in Transportation Networks. In Lasse Kliemann and Peter Sanders, editors, *Algorithm Engineering - Selected Results and Surveys*, volume 9220 of *Lecture Notes in Computer Science*, pages 19–80. Springer, 2016. URL: http://www.springer.com/gp/book/9783319494869.

2 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.

3 Moritz Baum, Julian Dibbelt, Thomas Pajor, and Dorothea Wagner. Dynamic Time-Dependent Route Planning in Road Networks with User Preferences. In *Proceedings of the 15th International Symposium on Experimental Algorithms (SEA'16)*, volume 9685 of *Lecture Notes in Computer Science*, pages 33–49. Springer, 2016. URL: http://link.springer.com/chapter/10.1007/978-3-319-38851-9_3.

4 Christian Bräuer. Route Planning with Temporary Road Closures. Master's thesis, Karlsruhe Institute of Technology, 2018.

5 Daniel Delling. Time-Dependent SHARC-Routing. *Algorithmica*, 60(1):60–94, May 2011. doi:10.1007/s00453-009-9341-0.

6 Daniel Delling and Giacomo Nannicini. Core Routing on Dynamic Time-Dependent Road Networks. *Informs Journal on Computing*, 24(2):187–201, 2012.

7 Guy Desaulniers and Daniel Villeneuve. The shortest path problem with time windows and linear waiting costs. *Transportation Science*, 34(3):312–319, 2000.

8 Julian Dibbelt, Ben Strasser, and Dorothea Wagner. Customizable Contraction Hierarchies. *ACM Journal of Experimental Algorithmics*, 21(1):1.5:1–1.5:49, April 2016. doi:10.1145/2886843.

9 Edsger W. Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

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

11 Robert Geisberger, Peter Sanders, Dominik Schultes, and Christian Vetter. Exact Routing in Large Road Networks Using Contraction Hierarchies. *Transportation Science*, 46(3):388–404, August 2012.

12 Peter E. Hart, Nils Nilsson, and Bertram Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4:100–107, 1968.

13 Giacomo Nannicini, Daniel Delling, Leo Liberti, and Dominik Schultes. Bidirectional A* Search on Time-Dependent Road Networks. *Networks*, 59:240–251, 2012. Best Paper Award.

14 Ariel Orda and Raphael Rom. Traveling without waiting in time-dependent networks is NP-hard. Technical report, Dept. Electrical Engineering, Technion-Israel Institute of Technology, 1989.

15 Luigi Di Puglia Pugliese and Francesca Guerriero. A survey of resource constrained shortest path problems: Exact solution approaches. *Networks*, 62(3):183–200, 2013.

16 Ben Strasser and Tim Zeitz. A* with perfect potentials, 2019. arXiv:1910.12526.

17 Marieke van der Tuin, Mathijs de Weerdt, and Gernot Veit Batz. Route Planning with Breaks and Truck Driving Bans Using Time-Dependent Contraction Hierarchies. In *Proceedings of the Twenty-Eigth International Conference on Automated Planning and Scheduling*. AAAI Press, 2018. URL: https://www.semanticscholar.org/paper/Route-Planning-with-Breaks-and-Truck-Driving-Bans-Tuin-Weerdt/85c067c0a033f11166d114fcfde093d3250bb8fd.