# Time-Dependent Route Planning for Truck Drivers

Alexander Kleff[1,2(✉)], Christian Bräuer[1,2], Frank Schulz[1], Valentin Buchhold[2], Moritz Baum[2], and Dorothea Wagner[2]

[1] PTV Group, Karlsruhe, Germany
{alexander.kleff,christian.braeuer,frank.schulz}@ptvgroup.com
[2] Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany
{buchhold,moritz.baum,dorothea.wagner}@kit.edu

**Abstract.** We study the problem of computing time-dependent shortest routes for truck drivers. In contrast to conventional route planning, truck drivers have to obey government regulations that impose limits on non-stop driving times. Therefore, route planners must plan *break periods* in advance and select suitable parking lots. To ensure that maximum driving times are not exceeded, predictable congestion due to, e. g., peak hours should also be taken into account. Therefore, we introduce the truck driver routing problem in *time-dependent* road networks. It turns out that the combination of time-dependent driving times with constraints imposed by drivers' working hours requires computation of multiple time-dependent *profiles* for optimal solutions. Although conceptually simple, profile search is expensive. We greatly reduce (empirical) running times by calculating bounds on arrival and departure times during additional search phases to only query partial profiles and only to a fraction of the parking lots. Carefully integrating this approach with a *one-to-many* extension of time-dependent contraction hierarchies makes our approach practical. For even faster queries, we also propose a heuristic variant that works very well in practice. Excellent performance of our algorithms is demonstrated on a recent real-world instance of Germany that is much harder than time-dependent instances considered in previous works.

**Keywords:** Time-dependent shortest paths · Drivers' working hours · Truck driver scheduling · Parking locations

## 1 Introduction

In many countries of the world, truck drivers are legally obligated to take breaks on a regular basis to obviate drivers' fatigue and hence increase road safety. For instance, Regulation (EC) No. 561/2006 of the European Union [15] demands a break of at least 45 minutes after at most 4.5 hours of driving. And according to the hours-of-service regulation in the United States [16], a 30-minutes-break is mandatory after at most eight hours have elapsed. Truck drivers must park their vehicle at a suitable location before taking such a "lunch break". Due to the size of their trucks, the drivers are severely limited compared to car drivers when in search of a parking space. For assistance in finding appropriate and available parking lots, truck drivers use mobile apps like Truck Parking Europe [1] that maintain databases of parking lots and display nearby lots to users. In this work, we investigate the following optimization problem: En route from one customer to another, when and where should the driver take a break (if at all) to conform to the provisions on breaks and arrive at the destination earliest possible?

We only consider one drive from a source to a destination. In general, a truck driver may visit multiple customers per day. In this case, the customers' time windows also have to be regarded. Moreover, if a trip takes more than one day, not only lunch breaks have to be scheduled but also longer rest breaks for the driver to sleep. The problem of scheduling breaks in order to comply with regulations while also taking customer time windows into account is known as the *truck driver scheduling problem* [21]. However, the locations of the parking lots remain disregarded in this setting. In this work, we take a major first step towards combining *time-dependent route planning* and *truck driver scheduling.* We determine not only *when* but also *where* to take a break.

We consider time-dependent driving times to model predictable congestion. In this scenario, it might be beneficial to not depart from source right away, or to prolong a break, or to wait at a parking lot for a time that is too short to count as break. As an example of *short-term waiting*, imagine the following: At the time of arrival at a parking lot, the driving time to the destination would be two minutes longer than the remaining allowed driving time. Luckily, the driver just has to wait ten minutes for the congestion to disperse and for the driving time to drop by these two minutes. In contrast to the European Union, short-term waiting does not pay off in the United States because the lunch break becomes mandatory after eight hours have elapsed, and not after a certain accumulated driving time. In the following, we focus on the EU regulation.

Time-dependency makes the problem particularly challenging, and the question arises whether it can be solved efficiently in practice. We are interested both in optimal and in heuristic approaches. There are a couple of parameters to reduce the run-time, and we seek to shed light on their impact on the solution quality. For our experimental analysis, it is sufficient to assume that the driver stops at only one parking lot (if at all). For a planning horizon of one day, this is no substantial limitation in practice as a daily driving time of 9h (US: 11h) should not be exceeded, even though it may be extended to 10h twice a week. For the sake of completeness, we discuss the implications regarding multiple stops.

*Related Work.* Route planning algorithms have received a large amount of attention in recent years, resulting in a multitude of *speedup techniques* [2]. In the *time-dependent* scenario, *driving time functions* associated with the edges map the time of the day to a driving time [7]. Dijkstra's algorithm [12] can be generalized [14] to answer *earliest arrival* (EA) queries. However, *profile* queries asking for the driving time function between two vertices are not feasible for large road networks [11], as such functions may have superpolynomial complexity [17] and maintaining them for all vertices makes Dijkstra's algorithm impractical.

Several classic speedup techniques have been generalized to the time-dependent scenario [6, 9, 10], typically focusing on fast EA queries. Efficient EA and profile queries at continental scale are provided by TCH [3], a generalization of Contraction Hierarchies (CH) [20]. Batched shorted paths in the time-dependent scenario are studied in [19]. Recently, Strasser [30] introduced a simple heuristic for time-dependent routing that is cheap in time and space, but drops optimality and provides no approximation guarantees.

As far as the truck driver scheduling problem is concerned, the interested reader can find descriptions of optimal algorithms for the EU variant of this problem in [21, 13, 26] and for the US variant in [22, 24, 25]. Of these, [26] and [24] propose a *mixed-integer linear programming* formulation. The former even takes time-dependent driving times into account, the latter is the only one to include real-world data of parking lots (here: interstate rest areas) into their experimental analysis. However, in both cases not only the sequence of customers is fixed but also the path in the road graph. So in the former case the path cannot change over time, and in the latter case truck stops aside the path are disregarded. In [27], time-dependent routes for truck drivers subject to government regulations and time windows are solved heuristically. Finally, other lines of research have considered problems that resemble our setting but are $\mathcal{NP}$-hard, such as crew scheduling [29], routing of electric vehicles [5], or time-dependent pollution-routing [18].

*Contribution and Outline.* We introduce the truck driver routing problem that asks for the fastest route between two customers that complies with legal provisions for truck drivers (Section 2). To the best of our knowledge, we are the first to integrate the choice of routes, breaks and parking lots in one query – unlike previous works that first fix the route and then schedule breaks, possibly missing the optimal solution. Since rush hours severely affect driving times, we consider the time-dependent scenario. We propose a naive approach (Section 3) that would be far too expensive in time and space without at least one of two described acceleration techniques (Sections 3.1 and 3.2): An implementation based on TCHs achieves query times in the order of minutes on the German road network. Sophisticated bounds computations on top of that speed queries up by a factor of 25, yielding running times well below 10 seconds. Finally, a heuristic approach (Section 3.3) enables queries below a second and less. Most of our experiments (Section 4) are performed on a new instance of the German road network, currently used by PTV in production systems. It turns out to be much harder than the ten-year-old instance used in most publications so far. Before we conclude (Section 6), we discuss the implications of allowing multiple stops (Section 5).

## 2 Problem Statement and Preliminaries

The basic input for every variant of the *truck driver routing problem* is the following: Let a road network be given, modeled as a directed *graph* $G = (V, E)$ with $n = |V|$ *vertices* and $m = |E|$ *edges*, where vertices $v \in V$ typically correspond to intersections and edges $(u, v) \in E$ to road segments. The subset

$P \subset V$ of the vertices contains exactly the *parking locations* that represent the parking lots (or even parking spaces) where the driver may take a break. The *minimum break period* and the *maximum driving time* until such a break is mandatory are denoted by *break* and *limit* respectively. These two parameters are sufficient to handle the Regulation (EC) No. 561/2006 of the European Union [15] for a planning horizon of one day.

We are also given a sequence of exactly two customers to be visited, *source* $s \in V \setminus P$ and *destination* $d \in V \setminus P$. An *s–d-path* $Path_{s,d}$ (in $G$) is a sequence $[v_1 = s, v_2, \ldots, v_k = d]$ of vertices such that $(v_i, v_{i+1}) \in E$ and $v_i \neq v_j$ for all $1 \leq i < j \leq k$. A *(truck driver) route* $Route_{s,d}$ from $s$ to $d$ in turn is a sequence $[Path_{u_i,v_i}]_{1 \leq i \leq k}$ of paths such that $u_1 = s$ and $u_i \in P$ for $1 < i \leq k$, $v_k = d$ and $v_i \in P$ for $1 \leq i < k$, and $v_{i-1} = u_i$ for $1 < i \leq k$. In this paper, we will only deal with routes with a sequence length $|Route_{s,d}| := k$ of at most two.

In the time-independent case, the weights on the edges are constants and indicate the driving time along the edge. A path is feasible iff the accumulated driving time along the path is no longer than *limit*, and a route is feasible iff all its paths are. The duration of a truck driver route $Route_{s,d}$ is the sum of the accumulated driving times of its paths plus $(|Route_{s,d}| - 1) \cdot break$. In time-independent truck driver routing, we are interested in a shortest feasible route from $s$ to $d$ if such a feasible route exists.

In time-dependent truck driver routing, we are given *time-dependent driving time functions* for every edge instead of constant driving times. That is, for every edge $(u, v)$ there is a function $\Psi_{u,v} \colon \mathbb{R} \to \mathbb{R}^+$ that maps the time of departure from $u$ to the driving time to $v$. In this work, all functions are supposed to be piecewise-linear. The driver is not allowed to wait at any vertex other than the parking locations or $s$. In this scenario, it is common to demand that the functions fulfill the *FIFO property* because the shortest-path problem would become $\mathcal{NP}$-hard if it was not satisfied for all edges [28, 8]. We even presume that functions are continuous and fulfill the *strict FIFO property*, i. e., for arbitrary $t < t' \in \mathbb{R}$, the condition $t + \Psi(t) < t' + \Psi(t')$ holds for every edge (later departure leads to later arrival). This way, the *arrival time function* $\mathrm{id} + \Psi$ is bijective (id being the identity function) and we can build the inverse $(\mathrm{id} + \Psi)^{-1}$ that maps an arrival time to the appropriate departure time.

To check feasibility of a route $Route_{s,d} = [Path_{u_i,v_i}]_{1 \leq i \leq k}$ , we also ask for *departure and arrival times* $dep(u_i)$ and $arr(v_i)$ for all $i$. This way, the duration of a path $Path_{u,v}$ can easily be computed by $arr(v) - dep(u)$ (must be positive) and the waiting time at a parking location by $dep(u_i) - arr(v_{i-1})$ (must be non-negative). To be feasible, no single path is allowed to be longer than *limit*. In addition, a route $[Path_{s,p}, Path_{p,d}]$ is feasible only if either the sum of the paths' durations does not exceed *limit* or there is a waiting time that counts as break at the parking location $p$ in between. Among all feasible truck driver routes we look for one with the earliest arrival at $d$. To this end, we are also given a *lower bound* on the *earliest departure* $lbED(s)$ from $s$, i.e., we demand $dep(s) \geq lbED(s)$. It is only a lower bound because a feasible route with $dep(s) = lbED(s)$ may not exist. In this paper, we call a vertex $v$ *reachable* from $u$ at time $t$ if there is a feasible route $Route_{u,v}$ with $dep(u) = t$.

A *(driving time) profile* between $u$ and $v$ is a time-dependent function $\psi_{u,v} \colon \mathbb{R} \to \mathbb{R}^+$ that maps every departure time at $u$ to the *shortest* driving time to $v$. If $(u, v) \in E$, the profile is identical to the given driving time function $\Psi_{u,v}$. If not, we can compute the profile $\psi_{u,v}$ recursively either forward or backward using the *link operation* $\odot$ and the *merge operation* $\oplus$:

$$\psi_{u,v} := \bigoplus_{w\colon (w,v)\in E} \psi_{u,w} \odot \Psi_{w,v} \qquad \text{or} \qquad \psi_{u,v} := \bigoplus_{w\colon (u,w)\in E} \Psi_{u,w} \odot \psi_{w,v} \tag{1}$$

where $\psi \odot \varphi$ is defined to be $\psi + \varphi \circ (\mathrm{id} + \psi)$ and $\psi \oplus \varphi$ defined to be $\min(\psi, \varphi)$. A profile search can be implemented as described in [11].

## 3 Solution Approach

We first describe a basic and rather naive approach to compute the earliest arrival at destination $d$. There are three ways in which $d$ may be reachable from $s$: Either without passing a parking location at all, or by taking a break at a parking location, or by short-term waiting at a parking location. Accordingly, we will now compute three values $opt_{none}$, $opt_{break}$, and $opt_{short}$. The minimum of these is then the overall optimal solution.

At first, we investigate whether $d$ can be reached from $s$ without passing a parking location. To do this, we compute the driving time profile $\psi_{s,d}$ from $s$ to $d$ and then look up the earliest feasible departure time $dep_{s,d}$ from $s$ in this respect: $dep_{s,d} := \min\{t : \psi_{s,d}(t) \leq limit \wedge t \geq lbED(s)\}$. With this, we can conclude: $opt_{none} := dep_{s,d} + \psi_{s,d}(dep_{s,d})$.

To consider the parking locations, we have to search forward and backward in order to compute the driving time profiles $\psi_{s,p}$ and $\psi_{p,d}$ for all $p \in P$. In the case with a break at a parking location, the next step is, similarly as before and for every parking location $p \in P$, to determine the earliest feasible departure time $dep_{s,p}$ from $s$ when going to $p$ as $dep_{s,p} := \min\{t : \psi_{s,p}(t) \leq limit \wedge t \geq lbED(s)\}$, and then to look up the earliest feasible departure time $dep_{p,d}$ from $p$ when going to $d$ after a break as $dep_{p,d} := \min\{t : \psi_{p,d}(t) \leq limit \wedge t \geq dep_{s,p} + \psi_{s,p}(dep_{s,p}) + break\}$. In turn, we can conclude: $opt_{break} := \min_{p\in P}\{dep_{p,d} + \psi_{p,d}(dep_{p,d})\}$.

But maybe the optimal solution consists in just waiting at a parking location for a short time that does not necessarily count as break. To take this case into account, we determine the earliest feasible departure time $dep_{s,p,d}$ from $p$ when going from $s$ to $d$ for every parking location $p \in P$ as follows: $dep_{s,p,d} := \min\{t : \exists t' : \psi_{s,p}(t') + \psi_{p,d}(t) \leq limit \wedge lbED(s) \leq t' \leq t - \psi_{s,p}(t')\}$. Again, we conclude: $opt_{short} := \min_{p\in P}\{dep_{s,p,d} + \psi_{p,d}(dep_{s,p,d})\}$.

This description is only a sketch. It is meant to give an overview. A naive implementation would certainly be far too slow for any practical use. This motivates the following three acceleration approaches: by narrowing down profile searches, by time-dependent contraction hierarchies, and heuristically.

### 3.1 Acceleration by Narrowing Down Searches

Some computations can be performed faster than others. The idea is to spend little extra time on quick computations in order to gain bounds that help us speed up the expensive calculations such as the profile search.

We define $\mathit{ubMax}(\psi)$ as an upper bound on the maximum value of the profile $\psi$, i.e., $\mathit{ubMax}(\psi) \geq \max_{t\in\mathbb{R}} \psi(t)$. Analogously, $\mathit{lbMin}(\psi)$ is a lower bound on the minimum value of $\psi$. A query for these bounds, called a *profile bounds query* here, can be answered by applying Dijkstra's algorithm [14] on a graph where the constant edge weights are the minimum (maximum) values of the respective driving time functions. Given a departure time $t$ in addition, an earliest arrival (EA) query asks for the earliest arrival at $d$ when departing at time $t$. Both queries can be processed rapidly and are described in greater detail in [3]. In our context, a usual EA query only gives a lower bound $\mathit{lbEA}(d)$ on the earliest arrival if $\mathit{lbEA}(d) > t + \mathit{limit}$. To highlight this, we call it an *lbEA query.*

*Computing Partial Profiles.* One of the key acceleration techniques in this paper is to only compute a *partial profile.* A partial profile maps a departure time $t \in \mathbb{R}$ to a driving time in $\mathbb{R}^+ \cup \{\perp\}$, where $\perp$ can be read as *undefined.* We have to distinguish a *partial forward profile* from a *partial backward profile.* More precisely, the following holds for a partial forward profile $\psi^f$ given a *departure time range* $[t^{begin}, t^{end}] \subset \mathbb{R}$: $\psi^f(t) \in \mathbb{R}^+$ for $t^{begin} \leq t \leq t^{end}$ and $\psi^f(t) = \perp$ otherwise. An analog statement holds for a partial backward profile $\psi^b$ given an *arrival time range* $[t^{begin}, t^{end}] \subset \mathbb{R}$: $\psi^b(t) \in \mathbb{R}^+$ for $(\mathrm{id} + \psi^b)^{-1}(t^{begin}) \leq t \leq (\mathrm{id} + \psi^b)^{-1}(t^{end})$ and $\psi^b(t) = \perp$ otherwise.

A partial (forward or backward) profile for a given (departure or arrival time) range can be computed similar to before. If $(u, v) \in E$, we set

$$\psi^f_{u,v}(t) := \begin{cases} \Psi_{u,v}(t), & \text{if } t \text{ in range} \\ \perp, & \text{otherwise} \end{cases} \qquad \psi^b_{u,v}(t) := \begin{cases} \Psi_{u,v}(t), & \text{if } t + \Psi_{u,v}(t) \text{ in range} \\ \perp, & \text{otherwise} \end{cases}$$

If not, we use the same (forward or backward) recursion formula as before in (1). But we have to adjust the definitions of the link and merge operations and distinguish the forward from the backward case. The forward and backward link operations for a partial profile and a driving-time function of some edge are now defined as follows:

$$(\psi^f_{u,v} \odot^f \Psi_{v,w})(t) := \begin{cases} \psi^f_{u,v}(t) + \Psi_{v,w}(t + \psi^f_{u,v}(t)), & \text{if } \psi^f_{u,v}(t) \neq \perp \\ \perp, & \text{otherwise} \end{cases}$$

$$(\Psi_{u,v} \odot^b \psi^b_{v,w})(t) := \begin{cases} \Psi_{u,v}(t) + \psi^b_{v,w}(t + \Psi_{u,v}(t)), & \text{if } \psi^b_{v,w}(t + \Psi_{u,v}(t)) \neq \perp \\ \perp, & \text{otherwise} \end{cases}$$

The forward and backward merge operations for two partial profiles for the same vertex pair and range are now defined as follows:

$$(\psi^f \oplus^f \varphi^f)(t) := \begin{cases} \min\{\psi^f(t), \varphi^f(t)\}, & \text{if } \psi^f(t) \neq \perp \wedge \varphi^f(t) \neq \perp \\ \perp, & \text{otherwise} \end{cases}$$

$$(\psi^b \oplus^b \varphi^b)(t) := \begin{cases} \psi^b(t), & \text{if } \psi^b(t) \neq \perp \wedge (\mathrm{id} + \varphi^b)^{-1}(t + \psi^b(t)) \leq t \\ \varphi^b(t), & \text{if } \varphi^b(t) \neq \perp \wedge (\mathrm{id} + \psi^b)^{-1}(t + \varphi^b(t)) < t \\ \perp, & \text{otherwise} \end{cases}$$

Given a source, a destination, and a range, we call a query for a partial profile a *profile range query.*

*One-to-one queries.* At first, we perform a *one-to-one* lbEA query for $s$ and $d$ and departure time $lbED(s)$, that is, we compute the earliest arrival at $d$ as if there was no break to take when leaving $s$ earliest possible. If this lower bound $lbEA(d)$ on the earliest arrival is no later than $lbED(s) + limit$, it is tight, and we have found the requested earliest arrival at $d$.

The second step is to compute a lower bound $lbMin(\psi_{s,d})$ on the driving time from $s$ to $d$. If this bound is already greater than $2 \cdot limit$, we stop here because $d$ is considered to be not reachable from $s$ as we only take one break into account.

If $lbMin(\psi_{s,d}) \leq limit$, then an optimal solution may incorporate short-term waiting at a parking location. We store this information by setting $lbWaiting := 0$. Otherwise we set $lbWaiting := break$ because it is certain that the driver will have to take a break at one of the parking locations.

*One-to-many-to-one queries.* We perform both an lbEA search and a profile bounds search from $s$ to all potentially reachable parking locations, that is, we compute $lbMin(\psi_{s,p})$, $ubMax(\psi_{s,p})$ and $lbEA(p)$ for all $p \in P$ with $lbMin(\psi_{s,p}) \leq limit$. We insert all those parking locations $p$ with $lbEA(p) \leq lbED(s) + limit$ into a set $Blue_1$. So for all $p \in Blue_1$, the lower bound $lbEA(p)$ is tight and equals the earliest arrival $EA(p)$ at $p$. We add the other potentially reachable parking locations $p$, i.e. with $lbEA(p) > lbED(s) + limit$ (and also $lbMin(\psi_{s,p}) \leq limit$ by construction), to a set $Red_1$. These are the ones for which the lower bound is known to be not tight. The set $Red_1$ may remain empty, especially if waiting at $s$ was not allowed. An empty set $Red_1$ helps to speed up computation as we can omit the forward profile range query later. If both sets are empty, there is no feasible solution. $Blue_1$ and $Red_1$ are each the first element of a sequence of subsets of $P$ that we will construct in the following. An illustration is shown in figure 1.

The next step is to conduct a profile bounds search from $d$ backwards to all potentially reachable parking locations in $Blue_1 \cup Red_1$, i.e., we compute $lbMin(\psi_{p,d})$ and $ubMax(\psi_{p,d})$ for all $p \in Blue_1 \cup Red_1$ with $lbMin(\psi_{p,d}) \leq limit$. Let $Blue_2$ (resp. $Red_2$) be the subset of parking locations in $Blue_1$ (resp. $Red_1$) that are also potentially reachable backwards. Again, if $Blue_2 \cup Red_2$ is empty, there is no feasible solution. With the bounds on the driving time we get (better) bounds on the earliest arrival at $d$. We can set the upper bound $ubEA(d)$ to $\min\{EA(p) + break + ubMax(\psi_{p,d}) : p \in Blue_2 \wedge ubMax(\psi_{p,d}) \leq limit\}$, where the minimum over the empty set is considered to be infinite. If $lbWaiting = break$ and improving, we can update the lower bound $lbEA(d)$ to $\min\{lbEA(p) + break + lbMin(\psi_{p,d}) : p \in Blue_2 \cup Red_2\}$.

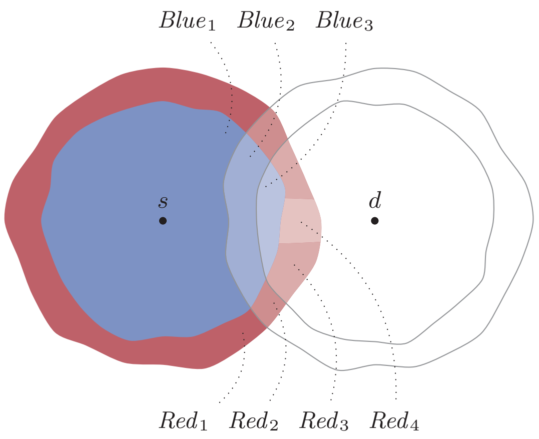

**Fig. 1.** The set sequences $Blue_1 \supset Blue_2 \supset Blue_3$ and $Red_1 \supset Red_2 \supset Red_3 \supset Red_4$. The two sets $Blue_1$ and $Red_1$ are disjoint.

A profile range search backwards from $d$ in the range $[lbEA(d), ubEA(d)]$ to all $p \in Blue_2 \cup Red_2$ yields a partial profile $\psi_{p,d}$ for all these $p$. It is defined for exactly those departure times $t$ from $p$ for which $t + \psi_{p,d}(t) \in [lbEA(d), ubEA(d)]$ holds. For all $p \in Blue_2$ we can now determine an upper bound $ubED(p)$ on the earliest departure from $p$ as the earliest point in time $t$ such that $t \geq EA(p) + break$ and $\psi_{p,d}(t) \leq limit$. In case $lbWaiting = break$, this bound is tight. In the other case, we may be able to improve it by the earliest point in time $t$ for which $t \geq EA(p)$ and $\psi_{p,d}(t) \leq limit - (EA(p) - lbED(s))$ holds. However, we might not be able to find such an upper bound because neither of the conditions are met. So let $Blue_3 \subset Blue_2$ be the set of parking locations for which $ubED(p)$ can be determined. Then, we may improve the upper bound $ubEA(d)$ on the earliest arrival at $d$ by $\min\{ubED(p) + \psi_{p,d}(ubED(p)) : p \in Blue_3\}$.

On the other hand, we calculate a lower bound $lbED(p)$ on the earliest departure from $p$ for all $p \in Red_2$ as the earliest point in time $t$ with $t \geq lbEA(p) + break$ and $\psi_{p,d}(t) \leq limit$. If $lbWaiting = 0$, we may have to lower this bound to the earliest point in time $t$ with $t \geq lbEA(p)$ and $\psi_{p,d}(t) \leq limit - lbMin(\psi_{s,p})$. And, again, let $Red_3 \subset Red_2$ be the set of parking locations for which $lbED(p)$ can be determined.

Let $Red_4 \subset Red_3$ be the set of parking locations $p$ for which $lbED(p) + \psi_{p,d}(lbED(p)) < ubEA(d)$ holds. So $Red_4$ contains those parking locations for which a forward profile range search is inevitable. If this set is empty and $lbWaiting = break$, then $ubEA(d)$ is tight, so we are done. If not, we need to compute an upper bound $ubED(p)$ on the departure time from $p$ for all $p \in Red_4$ (and $p \in Blue_2$ if $lbWaiting = 0$): It is the point in time $t$ with $t + \psi_{p,d}(t) = ubEA(d)$. With the upper bound for all $p$, we can obtain an upper bound $ubED(s)$ on the departure from $s$: It is $\max\{ubED(p) - lbWaiting - lbMin(\psi_{s,p})\}$ over all $p \in Red_4$ (and $p \in Blue_2$ if $lbWaiting = 0$).

Finally, we conduct a forward profile range search from $s$ to all $p \in \mathit{Red}_4$ (and $p \in \mathit{Blue}_2$ if $\mathit{lbWaiting} = 0$) for the departure time range $[\mathit{lbED}(s), \mathit{ubED}(s)]$. Now we have everything we need together: In case $\mathit{lbWaiting} = \mathit{break}$, we compute $\mathit{opt}_{\mathit{break}}$ similar to before, except that the earliest arrival at $d$ via the parking locations in $\mathit{Blue}_3$ is already known and has to be determined only for $\mathit{Red}_4$. In case $\mathit{lbWaiting} = 0$, we have to compute $\mathit{opt}_{\mathit{short}}$ in addition, but only for $\mathit{Blue}_2 \cup \mathit{Red}_4$, and also $\mathit{opt}_{\mathit{none}}$ (provided that waiting at $s$ is allowed). To speed up the computation of $\mathit{opt}_{\mathit{none}}$, we only perform a forward profile range search from $s$ to $d$ for the range $[\mathit{lbED}(s), \mathit{ubEA}(d) - \mathit{lbMin}(\psi_{s,d})]$.

## 3.2 Acceleration by Contraction Hierarchies

In the previous section, we proposed techniques to reduce the number of profile searches and restrict the remaining profile searches to smaller ranges. We accelerate our approach even further by speeding up the profile searches (and EA queries) themselves using *time-dependent contraction hierarchies* [3]. (T)CHs were originally proposed for point-to-point queries, whereas we also need to compute a variant of one-to-many queries (from a source vertex to all parking lots). In this section we recap the (time-dependent) contraction hierarchies algorithm and describe our modifications of it.

A contraction hierarchy (CH) [20] is built by *contracting* the vertices of a graph in increasing order of importance. Intuitively, vertices that lie on many shortest paths (such as vertices on highways) are considered important. To contract a vertex $v$, it is (temporarily) removed from the graph, and *shortcuts* are added between its neighbors in order to preserve distances in the remaining graph. *Witness searches* are performed to determine whether a shortcut is necessary or can be discarded. For each pair of neighbors $u, w$ with $(u, v) \in E, (v, w) \in E$, we run a Dijkstra search from $u$ to $w$. Only if the path via $v$ is the *unique* shortest $u$–$w$-path, we add the shortcut between $u$ and $w$. In the time-dependent case, we need to run a profile search from $u$ to $w$. A shortcut can only be omitted if it is not needed *at any point in time*.

CH queries are a modified variant of bidirectional Dijkstra, where both forward and reverse search relax only upward edges, i. e., edges going from less to more important vertices. In the time-dependent scenario, the reverse search is particularly difficult, because the time of arrival at the target is unknown. In a basic query variant, the reverse search only marks all edges in the reverse search space from $d$, and the forward search is allowed to additionally relax all marked arcs. More sophisticated query variants compute bounds during the reverse search that guide the forward search into the direction of $d$.

The obvious approach to compute EA queries or profiles from a source to all parking lots $P$ runs $|P|$ point-to-point TCH queries. However, we can do better with the following modification. During the contraction process, we *block* all vertices representing parking lots, i. e., we disallow to contract them. After contraction, there remains a *core graph* at the top of the hierarchy, consisting of all parking lots and (shortcut) arcs between them. Queries from a source $s$ to all parking lots now boil down to a forward search from $s$ that relaxes no edges to less important vertices. As long as the query has not yet reached the core, it behaves like a normal forward CH search. On the core graph, it behaves like a standard Dijkstra search. We can accelerate the search using the *stall-on-demand* optimization [20] and stop it as soon as all parking lot vertices are settled, or a certain time limit is reached. Since blocking arbitrary vertices can lead to suboptimal contraction orders, we do not contract all vertices but the parking lots, but rather stop contraction as soon as the remaining graph becomes too dense.

### 3.3 Heuristic Acceleration

In our study, we schedule waiting times on the assumption that the time-dependent driving times are deterministic. This is not the case in real-life. So it is questionable whether a route with, for instance, scheduled short-term waiting would be acceptable in practice. This is the motivation for the *restricted waiting policy* that disallows waiting at $s$, short-term waiting at any parking location, and the prolongation of a break. To conform to this policy, the driver must depart immediately at time $lbED(s)$ and may take a break of exactly 45 minutes if inevitable. In this scenario, it is not necessary to query any profiles, even if $d$ cannot be reached directly without break. Then, the $Red$ sets are ignored, and instead of computing partial profiles backwards from $d$ to $Blue_2$, we conduct multiple $lbEA$ searches forward from the parking locations in $Blue_2$, getting a better and better upper bound on the earliest arrival at $d$.

## 4 Experiments

In this section, we first describe the data and the test setup and then analyze run-time and solution quality of the described approaches. Our experiments are based on two versions of the road network of Germany with time-dependent driving time functions, see Table 1. The older network from the year 2006 has been used by several other studies related to time-dependent routing (see Section 1) and contains car driving times based on a traffic model. The very recent data from 2017 is quite different: The new data is more detailed with respect to time dependency, there are more edges with driving time functions that are not constant, and the total number of breakpoints representing the functions is larger. The driving times are based on historic data provided by TomTom which is post-processed by PTV such that it models truck driving times.

We use the database of PTV Group's Truck Parking Europe app [1]. It contains currently more than 25 000 parking lots all over Europe. Some parking lots cannot be linked to the old road network of 2006. Therefore, the number of parking locations is a bit lower than in the road network of 2017. The database does not only contain rest areas with fuel stations, restrooms, and restaurants but also parking areas without any facilities. It is not clear if or under what circumstances the choice of a parking area without facilities would be acceptable in practice. We will take this into account by also testing our algorithm with a

**Table 1.** Key figures of the input data used for the experiments. TD Edges denotes the relative number of edges with a time-dependent and not constant driving time function.

| Road network | Vertices | Edges | TD Edges | Breakpoints | Parking set | Parking subset |
|---|---|---|---|---|---|---|
| Germany 2017 | 7.2 M | 15.7 M | 28.6 % | 136.9 M | 6 596 | 759 |
| Germany 2006 | 5.1 M | 12.6 M | 3.7 % | 20.9 M | 6 447 | 731 |

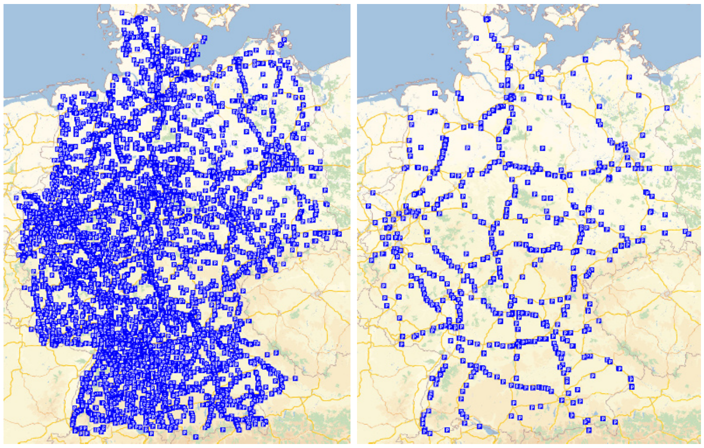

**Fig. 2.** The left image shows all available parking lots in Germany, the right image shows the reduced set with only big parking lots.

smaller subset of parking lots that offer 30 parking bays or more each. Figure 2 shows these two sets of parking lots.

*Test Setup.* We run our experiments on a VMware ESX cluster. Our machine has four cores of a 2.2 GHz Intel Xeon E5-2698 v4, 64 GB main memory, and runs Ubuntu 16.04. Besides the construction of the contraction hierarchies the algorithms use only one core. Our code is written in C++ and compiled with gcc 5.4, optimization level -O3. Our CH implementation is based on the code by Batz [3, 4] and has been extended as described in Section 3. We set the size of the CH cores to 0.2 % of the vertices in case of the whole parking set and 0.02% in case of the subset. This results in a CH search graph size of 38.90 GB in the former and 37.28 GB in the latter case (and 2.03 GB in the case of the 2006 road network).

Since our test data is the road network of Germany, we consider the EU regulation, i.e., *break*=45 min and *limit*=4.5 h. We generate 10 000 truck driver route queries for both versions of the road network. To this end, we randomly select vertices $s$ and $d$ and (a lower bound on) the earliest departure from $s$ between 6 am and 9 am. Since the run-time of these queries can differ a lot, we assign each of them to one of five categories: Category C1 comprises the queries for which the *lbEA* query suffices, i. e., $\mathit{lbEA}(d) \leq \mathit{lbED}(s) + \mathit{limit}$. Category

**Table 2.** Number of truck driver route queries per category.

| | C1 | C2 | C3 | C4 | C5 | Over all |
|---|---|---|---|---|---|---|
| Query set 2017 | 4278 | 210 | 4943 | 165 | 404 | 10000 |
| Query subset 2017 | 877 | 36 | 980 | 31 | 76 | 2000 |
| Query set 2006 | 7109 | 126 | 2754 | 1 | 10 | 10000 |

**Table 3.** Mean run-time per category in seconds for different scenarios.

| Scenario | C1 | C2 | C3 | C4 | C5 | Over all |
|---|---|---|---|---|---|---|
| Default scenario | 0.0038 | 18.1756 | 5.9549 | 121.9516 | 0.0053 | 5.3392 |
| Restricted waiting | 0.0033 | 0.2925 | 0.2187 | 0.1163 | 0.0910 | 0.1212 |
| Parking subset | 0.0041 | 5.8109 | 1.0646 | 7.8424 | 0.0057 | 0.7796 |
| Naive approach | 2.8018 | 287.1991 | 227.5335 | 228.4150 | 195.4254 | 128.8562 |
| Query subset 2017 | 0.0039 | 18.5811 | 5.8160 | 121.5858 | 0.0056 | 5.0708 |
| Germany 2006 | 0.0013 | 0.9829 | 0.3932 | 23.8170 | 0.0021 | 0.1239 |

C2 contains the ones with $lbEA(d) > lbED(s) + limit$ and $lbMin(\psi_{s,d}) \leq limit$, category C3 the ones with $lbMin(\psi_{s,d}) > limit$ and $ubMax(\psi_{s,d}) \leq 2 \cdot limit$, and category C4 the ones with $ubMax(\psi_{s,d}) > 2 \cdot limit$ and $lbMin(\psi_{s,d}) \leq 2 \cdot limit$. Finally, category C5 holds the instances with $lbMin(\psi_{s,d}) > 2 \cdot limit$ that cannot be solved.

Table 2 lists how these queries are distributed among the five categories. In case of the *query set 2006*, there are far more queries in C1 because with car driving times the vehicle's range is larger. Also in this list is the *query subset 2017*. We need this smaller subset of queries to measure the run-time of the long running naive approach.

*Results on Run-Time.* Table 3 shows the mean run-time for different scenarios, broken down into the five categories. The categories themselves are not part of the input of the algorithm. For the *default scenario*, we use the 2017 road graph, all described acceleration techniques, all parking locations, and allow waiting of any duration. The other scenarios deviate from this in one aspect each. In the default scenario, the run-time varies a lot with the category. A query from category C4 takes more than 30 000 times longer than one from C1. Since there are far more queries in C1 than in C4, the mean run-time over all 10 000 queries is still less than 6 seconds. Queries from C4 take so long because in 106 cases no upper bound on the earliest arrival at $d$ can be determined, so a full backward profile search is necessary. Figure 3 illustrates the run-time distribution among the 10 000 queries.

In case of the *restricted waiting* policy, waiting at $s$ is not allowed and waiting at any parking location is only allowed if the waiting time equals exactly the time for a break. This speeds the calculation up by a factor of 40 over all queries. In the *parking subset* scenario, we allow waiting only at larger parking lots. We

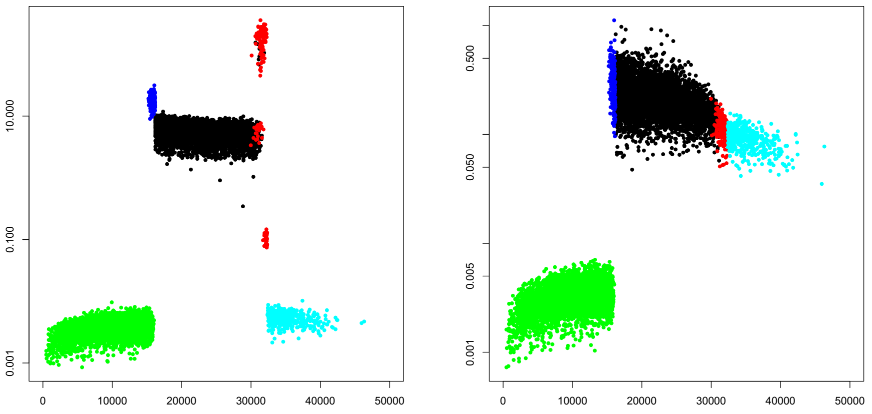

**Fig. 3.** Run-time of each $s$-$d$-query in the default scenario (left) and according to restricted waiting policy (right), $lbMin(\psi_{s,d})$ on abscissa and run-time in seconds on ordinate (on a logarithmic scale). Points are colored by category. Scales differ.

run the algorithm on the Germany 2017 network but the search graphs differ. Compared to the default setting, the smaller size of the core graph leads to faster one-to-many profile range queries (approx. by a factor of 7.5) but slower one-to-one profile range queries (approx. by a factor of 1.2). In both of these scenarios, not all queries can be solved. Solution quality is discussed later.

The *naive approach* does not make use of the acceleration based on partial profiles as described in Section 3.1 but still CH as in Section 3.2. Because of the long run-time of the naive approach, the run-times are based on the reduced query subset 2017 (see Table 2). For better comparability, we also give the run-times of the default scenario for the reduced query subset. An achieved speed-up of 25 over all queries proves the effectiveness of our described acceleration in general. The main aspect of it is the computation of only partial profiles that concerns category C3 primarily. Here, we even achieve a speed-up of almost 40.

In case of the *Germany 2006*, we run the accelerated approach on the 2006 road graph that was used in the original TCH publication [3]. The run-time is smaller by an order of magnitude compared to our recent data.

Some more numbers are of interest. A crucial issue of our bounds-based acceleration is to find a (good) upper bound $ubEA(d)$. In the default scenario, there are 113 cases in which such a bound cannot be determined and so a complete profile needs to be searched for backwards. A complete profile search backwards takes 138.7 s on average. In contrast, a profile range search is performed in 5168 cases and takes 5.8 s on average. The mean length of these ranges, i. e. $ubEA(d) - lbEA(d)$, is 604 s. A second important aspect of the acceleration is to avoid the profile (range) search forward if the set $Red_4$ is empty. This set contains elements only in 50 cases and then only a few, most often just one. Figure 4 shows a sample query with empty set $Red_4$.

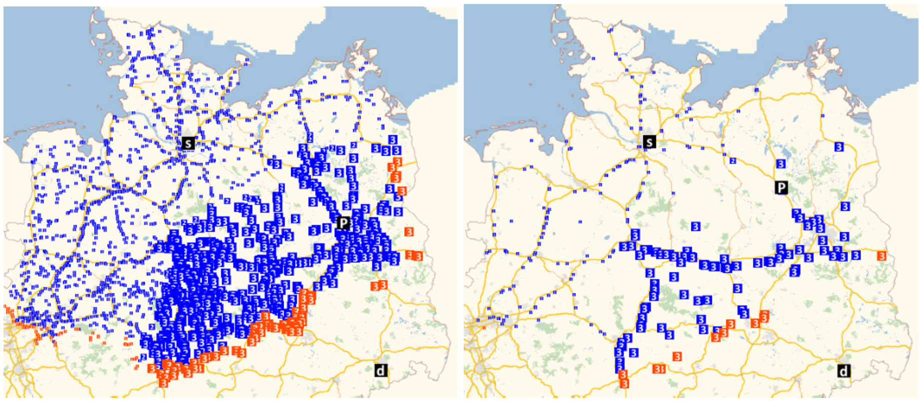

**Fig. 4.** Sample query from Hamburg to Dresden in the default scenario (left) and in the parking subset scenario (right). Different parking lots (P) are selected. The largest squares represent the sets $Blue_3$ and $Red_3$.

*Results on Quality.* Table 4 compares the solution quality of the default scenario to the *restricted waiting* and the *parking subset* scenario. The results of the *naive approach* are identical to the default, and the results of *Germany 2006* are hardly comparable, particularly since the driving times in this setting are based on a car model.

In the default setting, 9558 of 10 000 queries can be solved. We observe that the travel time, i. e., the driving time plus all waiting time (at $s$ and at parking), exceeds 15 hours in some cases, presumably to exploit the short driving times during the night. Such a solution is feasible according to our problem statement but most likely it would neither be acceptable in practice nor legal as truck drivers have to take a sleep rest daily. In the following, we call a solved query legal if the travel time does not exceed 15 hours. In case of the restricted waiting policy, a solved query is always legal.

We also state how many queries are solved (legally and) optimally, i. e., how often is the calculated earliest arrival at $d$ identical to the default scenario. In the parking subset scenario, this happens in 58% of the cases, even though there are less than 12% of the parking lots in the subset. Parking lots with more than 30 parking bays are most often located right next to a freeway (Autobahn in

**Table 4.** Comparison of solution quality for different scenarios. Mean and maximum deviation is in seconds over all queries that are legal but not optimal.

| Scenario | solved | legal | optimal & legal | mean dev | max dev |
|---|---|---|---|---|---|
| Default scenario | 9558 | 9512 | 9512 | 0 | 0 |
| Restricted waiting | 9474 | 9474 | 9453 | 1211 | 2265 |
| Parking subset | 9518 | 9470 | 5518 | 127 | 17559 |

Germany), whereas many of the small parking lots are further away from it. In the restricted waiting scenario, only 21 of the solved queries are not solved optimally. So in the vast majority of the cases, the computational effort spent on taking waiting of any duration into account does not pay off. For instance, short-term waiting is scheduled only 11 times in the default scenario.

## 5 Enhancement to Multiple Stops

Our algorithm is tailored to the one-stop case. What are the implications if we allow more than one stop? For instance, if there were two drivers on board, they could take turns and stop three times for a change before they must take a rest and sleep. From a conceptual perspective, the multi-stop case is not too difficult. Let $P_s$ be the parking locations that are reachable from $s$ at some point in time without taking a break along the path, and let $P_d$ be the parking locations that are potentially reachable backwards from $d$, i. e., $\mathit{lbMin}(\psi_{p_d,d}) \leq \mathit{limit}$ for all $p_d \in P_d$. Moreover, suppose we had precomputed a $|P| \times |P|$ matrix $M$ of travel time profiles such that for two parking locations $p_s$ and $p_d$, $M[p_s, p_d]$ maps the departure time from $p_s$ (where the driver is expected to have taken a break) to the shortest travel time to $p_d$, including as many breaks as needed and also one at $p_d$ (unless $p_s = p_d$). With this, a truck driver route query boils down to three steps: First, we compute the earliest arrival at every $p_s \in P_s$. Then, we determine the earliest departure from every $p_d \in P_d$ with the help of $M$ as follows:

$$\mathit{ED}(p_d) = \min_{p_s \in P_s} \mathit{EA}(p_s) + \mathit{break} + M[p_s, p_d](\mathit{EA}(p_s) + \mathit{break})$$

Having done that, we can finally calculate the earliest arrival at $d$, also checking if $d$ could be reached without any break.

We could easily adapt the restricted waiting policy heuristic to this general case. It is short-term waiting that makes the computation of the earliest arrival at every $p_s \in P_s$ challenging. In order to do so, we could propagate a time-dependent function forward (here: mapping an arrival time to the minimum accumulated driving time). But as we have seen, propagating a time-dependent function is expensive. So from a practical point of view, it would be important to again find ways of narrowing down the search, like finding good bounds and only propagating partial functions as we have demonstrated before. In addition to this challenge, our assumption that we have a matrix $M$ in memory is not realistic. Due to the superpolynomial complexity of the travel time profiles, we would most likely need hundreds of GB of main memory for the parking lots in Germany. So the question is raised what a good trade-off would be between memory consumption and computational effort (and solution quality).

## 6 Conclusion and Outlook

We have introduced the truck driver routing problem and described an exact algorithm for it. While a naive approach would be far too costly in time and space, it can be made feasible using our two proposed acceleration methods. One is a modification of TCH. Additionally narrowing down TCH searches by several fast bounds computations and queries of only partial profiles results in an extra speed-up of 25 and practical run-times. We have also suggested a heuristic based on the policy of restricted waiting and analyzed its effect. In this setting, truck driver route queries take well below one second without losing too much solution quality. Similarly effective is the restriction of the parking set to the more relevant parking locations.

In this paper, we have left out our experiments with approximated driving time functions. Using the algorithm of Imai and Iri [23] to approximate the functions of both original and shortcut edges further reduces the run-time, especially of profile (range) queries. In doing so, we only sacrifice a precision that is not justified in practice. Future work includes a solution to the combined truck driver routing and scheduling problem for a given sequence of customers by using the results of this paper as a building block. Moreover, it would be interesting to reevaluate the existing work on algorithms for time-dependent route planning on the new benchmark instance. We conjecture that other shortcut-based methods such as TD-CRP [6] also suffer significantly from the new instance. It could be promising to further investigate shortcut-free approaches like the ALT algorithm [10].

Our algorithm will also be evaluated in the EU research projects AEOLIX and Clusters 2.0.

## References

1. Truck Parking Europe, https://truckparkingeurope.com/
2. Bast, H., Delling, D., Goldberg, A.V., Müller–Hannemann, M., Pajor, T., Sanders, P., Wagner, D., Werneck, R.F.: Route Planning in Transportation Networks. LNCS, vol. 9220, pp. 19–80. Springer (2016)
3. Batz, G.V., Geisberger, R., Sanders, P., Vetter, C.: Minimum Time-Dependent Travel Times with Contraction Hierarchies. ACM J. Exp. Algorithmics 18, 1.4:1–1.4:43 (2013)
4. Batz, G.V.: KaTCH, https://github.com/GVeitBatz/KaTCH/
5. Baum, M., Dibbelt, J., Gemsa, A., Wagner, D., Zündorf, T.: Shortest Feasible Paths with Charging Stops for Battery Electric Vehicles. ACM SIGSPATIAL'15, pp. 44:1–44:10. ACM (2015)
6. Baum, M., Dibbelt, J., Pajor, T., Wagner, D.: Dynamic Time-Dependent Route Planning in Road Networks with User Preferences. SEA'16, LNCS, vol. 9685, pp. 33–49. Springer (2016)
7. Cooke, K.L., Halsey, E.: The Shortest Route Through a Network with Time-Dependent Internodal Transit Times. J. Math. Anal. Appl. 14(3), 493–498 (1966)
8. Dean, B.C.: Algorithms for Minimum-Cost Paths in Time-Dependent Networks with Waiting Policies. Networks 44(1), 41–46 (2004)
9. Delling, D.: Time-Dependent SHARC-Routing. Algorithmica 60(1), 60–94 (2011)
10. Delling, D., Nannicini, G.: Core Routing on Dynamic Time-Dependent Road Networks. Informs J. Comput. 24(2), 187–201 (2012)
11. Delling, D., Wagner, D.: Time-Dependent Route Planning, LNCS, vol. 5868, pp. 207–230. Springer (2009)

12. Dijkstra, E.W.: A Note on Two Problems in Connexion with Graphs. Numer. Math. 1(1), 269–271 (1959)
13. Drexl, M., Prescott-Gagnon, E.: Labelling Algorithms for the Elementary Shortest Path Problem with Resource Constraints Considering EU Drivers' Rules. Logistics Research 2(2), 79–96 (2010)
14. Dreyfus, S.E.: An Appraisal of Some Shortest-Path Algorithms. Oper. Res. 17(3), 395–412 (1969)
15. European Parliament, Council of the European Union: Regulation (EC) No. 561/2006 of the European Parliament and of the Council of 15 March 2006 on the harmonisation of certain social legislation relating to road transport and amending Council Regulations (EEC) No. 3821/85 and (EC) No. 2135/98 and repealing Council Regulation (EEC) No. 3820/85. OJ L 102(1), 1–13 (2006)
16. Federal Motor Carrier Safety Administration: Hours of Service of Drivers. Fed. Reg. 76(248), 81133–81188 (2011)
17. Foschini, L., Hershberger, J., Suri, S.: On the Complexity of Time-Dependent Shortest Paths. Algorithmica 68(4), 1075–1097 (2014)
18. Franceschetti, A., Honhon, D., Van Woensel, T., Bektaş, T., Laporte, G.: The Time-Dependent Pollution-Routing Problem. Transportation Res. B - Meth. 56, 265–293 (2013)
19. Geisberger, R., Sanders, P.: Engineering Time-Dependent Many-to-Many Shortest Paths Computation. ATMOS'10, OASIcs, vol. 14, pp. 74–87 (2010)
20. Geisberger, R., Sanders, P., Schultes, D., Vetter, C.: Exact Routing in Large Road Networks Using Contraction Hierarchies. Transport. Sci. 46(3), 388–404 (2012)
21. Goel, A.: Truck Driver Scheduling in the European Union. Transport. Sci. 44(4), 429–441 (2010)
22. Goel, A.: Hours of Service Regulations in the United States and the 2013 Rule Change. Transp. Policy 33, 48–55 (2014)
23. Imai, H., Iri, M.: An Optimal Algorithm for Approximating a Piecewise Linear Function. Journal of Information Processing 9(3), 159–162 (1987)
24. Koç, C., Bektaş, T., Jabali, O., Laporte, G.: A Comparison of Three Idling Options in Long-Haul Truck Scheduling. Transportation Res. B - Meth. 93, Part A, 631 – 647 (2016)
25. Koç, Ç., Jabali, O., Laporte, G.: Long-Haul Vehicle Routing and Scheduling with Idling Options. J. Oper. Res. Soc. (forthcoming)
26. Kok, A., Hans, E., Schutten, J.: Optimizing Departure Times in Vehicle Routes. Eur. J. Oper. Res. 210(3), 579 – 587 (2011)
27. Shah, V.D.: Time Dependent Truck Routing and Driver Scheduling Problem with Hours of Service Regulations. Master's thesis, Northeastern University (2008)
28. Sherali, H.D., Ozbay, K., Subramanian, S.: The Time-Dependent Shortest Pair of Disjoint Paths Problem: Complexity, Models, and Algorithms. Networks 31(4), 259–272 (1998)
29. Smith, O.J., Boland, N., Waterer, H.: Solving Shortest Path Problems with a Weight Constraint and Replenishment Arcs. Comput. Oper. Res. 39(5), 964–984 (2012)
30. Strasser, B.: Intriguingly Simple and Efficient Time-Dependent Routing in Road Networks. CoRR abs/1606.06636 (2016)