Route Planning with Breaks and Truck Driving Bans Using Time-Dependent Contraction Hierarchies

van der Tuin, Marieke; de Weerdt, Mathijs; Batz, G. Veit

Publication date
2018

Document Version
Final published version

Published in
Proceedings of The Twenty-Eighth International Conference on Automated Planning and Scheduling

Citation (APA)
van der Tuin, M., de Weerdt, M., & Batz, G. V. (2018). Route Planning with Breaks and Truck Driving Bans Using Time-Dependent Contraction Hierarchies. In M. de Weerdt, S. Koenig, G. Röger, & M. Spaan (Eds.), *Proceedings of The Twenty-Eighth International Conference on Automated Planning and Scheduling* (pp. 356-364). American Association for Artificial Intelligence (AAAI).
https://www.aaai.org/ocs/index.php/ICAPS/ICAPS18/paper/view/17745

Route Planning with Breaks and Truck Driving Bans Using Time-Dependent Contraction Hierarchies

Marieke van der Tuin
Delft University of Technology
Transport and Planning department
Stevinweg 1, Delft
The Netherlands

Mathijs de Weerdt
Delft University of Technology
Algorithmics group
Van Mourik Broekmanweg 6, Delft
The Netherlands

G. Veit Batz
ORTEC B.V.
ORTEC Optimization Technology
Houtsingel 5, Zoetermeer
The Netherlands

Abstract

Mandatory breaks for truck drivers are nowadays scheduled after the route has been decided. However, in some cases it is beneficial to plan these breaks during waiting time caused by truck driving bans. Optimally planning a single break considering driving bans can be done using Dijkstra's algorithm with multiple labels. This has large effects on predicted travel times: 17% of the analysed routes having a night rest obtain an earlier arrival time by 5 hours on average. However, the computation times of this algorithm are long. A novel heuristic version of time-dependent contraction hierarchies leads to significant reductions in computation times from several seconds to several milliseconds per route. Experiments show that the solutions are still optimal for a representative test set consisting of 10,000 route queries.

1 Introduction

Truck drivers are required to take a break after several hours of driving. Currently, sophisticated route planners used by freight transporters do not consider the optimal planning of these breaks. Instead, breaks are inserted after the route has been planned, usually as late as possible. However, this does not always lead to the earliest arrival in case of temporary road closures for trucks, which occur in some European countries. Such driving bans prohibit driving during the night or in the weekend to reduce pollution or noise. In the context of such driving bans, the insertion of a break as late as possible might cause a truck to be too late to pass through the region before the ban starts, resulting in a long wait, while a different (longer) route through other countries may exist that results in an earlier arrival time. Furthermore, it is also possible that a truck has to wait due to a driving ban while soon afterwards a break needs to be taken. By scheduling the break earlier, (partly) during the driving ban, it is possible to efficiently use this waiting time to arrive earlier at the destination. The influence of driving bans on total travel time is substantial, due to the long closures lasting 7 till up to 38 hours. However, as we show later in this paper, optimally planning breaks and the route itself on a time-dependent network can be seen as an optimization problem with two criteria, which is an NP-hard problem (Hansen 1980). This paper describes how to plan routes with a single break adhering to the European drivers' legislation (2006), while taking driving bans into account. A single break on a route is the simplest case possible and provides a good insight into the effects of optimal break planning for this proof-of-concept stage. Still this obtains a large coverage, since 80% of routes driven by trucks in Europe includes only one single night rest. The first contribution of this paper is a quantitative analysis of the effect of optimal break planning on the travel times. This is done by presenting a modification of a multi-label Dijkstra search, and evaluating it experimentally on a detailed network of Europe having 7 million nodes. The test set consists of 10,000 routes based on actual freight flows. These experiments show that the arrival times decrease notably compared to the industry standard.

However, in this detailed network, the algorithm has quite a long runtime. This is a problem if it is used for solving a Vehicle Routing Problem (VRP), determining the best delivery sequences for a fleet of trucks. Therefore, the second contribution of our paper is showing that these routes can be computed fast but accurate using a heuristic version of time-dependent contraction hierarchies. The obtained speed-up is about 200 times, while the solution quality remains good. An even higher speed-up is reached by introducing a single-label heuristic algorithm: it computes routes about 2000 times as fast as the Dijkstra multi-label algorithm. All contributions are based on the first author's thesis (2017).

The structure of this paper is as follows. Section 2 discusses related work in the area of shortest path computation for road networks, including contraction hierarchies. Then the problem of optimal break planning with driving bans is defined formally. Section 4 introduces the optimal algorithm, whereas Section 5 provides the speed-up using contraction hierarchies. The algorithms are experimentally tested and compared to a reference algorithm in Section 6.

2 Related Work

Optimal break planning is a variation of the shortest path problem. This section gives an overview of this well-researched domain based on a review by Sommer (2014).

2.1 Shortest path algorithms

Dijkstra (1959) describes one of the earliest algorithms solving the shortest path problem, and no faster algorithm for

general graphs with constant travel times is developed since. It is still widely used in various applications.

However, a road network is not a general graph but has special properties that are exploited by speed-up techniques: it is *sparse* (a low number of edges per node), it is almost *planar* (relatively few bridges and tunnels) and a particular *layout* is given (the geographic coordinates). Algorithms using such properties are bi-directional search (Dantzig 1962), A^* (Hart, Nilsson, and Raphael 1968), ALT (Goldberg and Harrelson 2005), and SHARC (Bauer and Delling 2009).

Other speed-up techniques use the hierarchical nature of the road network: a typical route starts at a local road, travels via a B-road to the highway and then gets back to a local road. Contraction Hierarchies (CH) is an example of such an algorithm (Geisberger et al. 2008). It consists of two phases: *preprocessing* and *querying*. During preprocessing a hierarchy is constructed where every level contains one node less, which is *contracted* (that is, removed from the network). The minimum travel cost between the remaining nodes in the hierarchy level is preserved by adding an artificial shortcut edge if needed. The resulting hierarchy is then used for fast querying. The querying algorithm uses a bidirectional Dijkstra search, where the forward search only relaxes upward edges (i.e., toward nodes having a higher node ordering), whereas the backward search only relaxes downward edges. This creates so called *up-down-paths*. If the minimum value of both forward and backward priority queues exceeds the travel time of the shortest up-down-path so far, the shortest path has been found, as proven by Geisberger (2008).

The duration of the preprocessing step of an 18 million nodes network of Western Europe is 25 minutes (using an AMD Opteron 2.6 GhZ processor with 16GB of RAM) (Batz and Sanders 2012). Although this is a substantial computation time, this enables fast querying afterwards (on average 0.2 ms).

2.2 Shortest path algorithms with driving bans

The generalized form of shortest paths with driving bans is the NP-hard time-dependent shortest path problem. A condition that makes the problem polynomially solvable is the *FIFO (First In, First Out)* property (Orda and Rom 1990), stating that departing later never results in an earlier arrival time. This property is assumed to be valid in road networks.

Given the FIFO property, the Dijkstra (1959) algorithm is easily modified to in a time-dependent variant. This is not the case for the various speed-up techniques: for a backward search the arrival time at the destination needs to be known, but this is only known after the complete path is computed. However, this can be overcome by first performing a *backward interval search* to determine all possible paths independent of the arrival time (Nannicini et al. 2008; Batz et al. 2013). Then, the *forward search* is continued as a *downward search* to determine the earliest arrival path.

The contraction hierarchies algorithm for time-dependent networks is presented by Batz et al. (2013). Its preprocessing phase consists of subsequently contracting nodes from the network, just as its non time-dependent variant. Shortcut edges represent the fastest route for each possible departure time, and thus represent possibly multiple routes. The resulting hierarchy is used by the querying algorithm, which uses a time-dependent bidirectional search. The time-dependent contraction hierarchies algorithm (TCH) is proven to produce optimal results, and provides the fastest querying times currently known for time-dependent road networks.

2.3 Shortest path algorithms with breaks

As shown in this paper, the problem of scheduling breaks can be seen as a multi-criteria shortest path problem, with criteria of travel time and time since last break. The optimal multi-criteria path is in general not prefix optimal, meaning that it is not necessarily the case that every subpath starting at the start node is optimal as well. This problem is proven to be NP-hard (Hansen 1980) and cannot be solved using the Dijkstra shortest-path algorithm which only maintains prefix optimal paths. Instead, a *Pareto* or *multi-label search* algorithm is applied (Hansen 1980). This algorithm, based on Dijkstra's algorithm, keeps track of all possible *non-dominating* paths at each node. In Section 4, we show how to do this for the optimal break-planning problem.

According to the taxonomy of Lahyani (2015), optimal break planning in time-dependent networks had not yet been studied before their survey. In all discussed literature, breaks are inserted when needed, without considering optimality. For example, Kok et al. (2010) schedules breaks sub-optimally by planning these at customers only (within the VRP context). If the travel time between two customers is too long, artificial customers are inserted beforehand where a break can be scheduled. However, after the survey by Lahyani, the bachelor work of Bräuer (2016) uses contraction hierarchies for optimal break planning in a time-dependent network. He places parking lots on the highest level of the preprocessed hierarchy. Every up-down path then contains a parking lot, where a break is scheduled. Time-dependency is included by daily congestion, driving bans are not taken into account. An optimal solution is computed by considering all paths from start to destination via each of these parking lots. This leads to a runtime of 16 seconds per query for a graph of Germany having 7 million nodes – worse than one would expect by running the Pareto algorithm described earlier. Restricting this approach to only a portion of parking lots, leads to runtimes of 0.2 seconds with an approximation factor of 5%.

Although Bräuers computation times are not fast, previously it was shown that time-dependent contraction hierarchies can be used for solving the general multiple objective shortest path problem in fast computation time (Batz and Sanders 2012). It is even shown that it easily outperforms a multi-label A^* algorithm.

Therefore, it is assumed that it is possible to use TCH to obtain a fast algorithm capable of planning breaks – albeit possibly suboptimally.

3 Problem Definition

In this work we study the problem of optimally planning a single break in a network with truck driving bans. Given is a directed graph $G = (V, E)$ consisting of vertices (nodes) and edges (links). Each edge $e \in E$ is described as a pair of vertices (u, v).

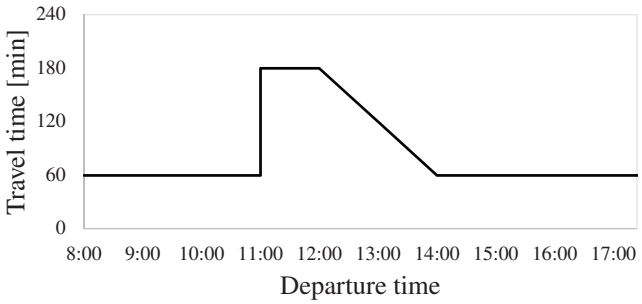

Figure 1: TTF with $\Delta_{\text{FF}} = 60$ and $RB = [12:00, 14:00]$

An optimal break planning path is then described as a path from start node $s \in V$ toward destination node $t \in V$, for a given departure time $\tau_0 \in \mathbb{R}$, and an already driven consecutive driving time of Δ_{d_0}, where the arrival time $\tau_t \in \mathbb{R}$ is minimal and at most one break is taken along the path. The path is represented as a list of nodes $P = \langle s, \ldots, t \rangle$.

The route should adhere to the drivers legislation. Therefore, the consecutive driving time $\Delta_d \in \mathbb{R}_{\geq 0}$ may not exceed a certain threshold T. A break of duration Δ_b can be taken anywhere on the route to reset Δ_d to 0. Only one break is allowed on each route. If Δ_d exceeds T and a break already has been modelled, the route is considered infeasible.

The travel time of an edge at a certain departure time is influenced by delays caused by driving bans, referred to as a road block. Such a road block is described as follows:

Definition 1 (Road Block)**.** *A Road Block (*RB*) is given by an interval $[\tau_{bs}, \tau_{be}]$, indicating the start and end times. During the interval, it is not allowed to drive the road segment.*

Every edge in the road network can have multiple road blocks. These road blocks may not overlap. The road blocks, together with the free flow travel time Δ_{FF}, form the *travel time function (TTF)* $f_e : \mathbb{R} \to \mathbb{R}_{\geq 0}$. This function represents the travel time for a given start time as a piecewise linear function (Ichoua, Gendreau, and Potvin 2003), while fulfilling the *FIFO (First In, First Out)* property. An example of such a TTF with a road block is shown in Figure 1.

To keep track of the allowed consecutive driving time Δ_d, a shift duration function is used. This function gives the new consecutive driving time after travelling an edge e at time τ and is denoted by $s_e(\Delta_d, \tau)$. Waiting for a road block does not count as driving time (but may be used as break time). As a result, $s_e(\Delta_d, \tau) = \Delta_d + \Delta_{\text{FF}}$, if no break is scheduled on the respective edge. If a break is scheduled on the specific edge, the function returns 0, such that the consecutive driving time Δ_d is reset as well.

Example 1. *A truck driver needs to travel from Venice toward Regensburg, as shown in Figure 2. The maximum consecutive driving time T is 4.5 hours and the break time Δ_b is 45 minutes. The truck driver departs Saturday morning at 9:43. The current industry practice —computing the fastest route and later on adding breaks where needed— results in the dark route on the right. However, the insertion of a break after 4.5 hours causes the truck to end up in the road block of Austria, resulting in an arrival time of 7:55 on Monday! The optimal break planning path is shown on the left, which entails that the required break is scheduled in Germany instead of Austria. This gives an arrival time of 18:01 on Saturday.*

Figure 2: Example route from Italy to Germany demonstrating the consequences of optimal break planning

This example shows a significant improvement in arrival time caused by optimal break planning. In the next sections four algorithms are proposed for computing such paths.

4 Break Planning

In this section we show how to model the network such that an optimal break planning path can be computed using the multi-label (Pareto) version of the Dijkstra algorithm.

4.1 Stacked break graph

The break planning problem is modelled using a modified graph. For each edge (u_1, v_1) in the original graph, an edge (u_1, v_2) is added, where v_2 is a copy of node v_1. The edge (u_1, v_2) represents travelling through the edge and taking a break somewhere on that edge. This graph is called a *Stacked Break Graph (SBG)* and is defined as follows:

Definition 2 (Stacked Break Graph)**.** *Given a graph (V, E), a* Stacked Break Graph (SBG) *is a graph $G' = (V_1 \cup V_2, E_1 \cup E_2 \cup EB)$ where $V_1 = V$ and $E_1 = E$, V_2 and E_2 are exact copies of V and E, and EB represents possible breaks along an edge in E_1, i.e., $EB = \{(u_1, v_2) | (u_1, v_1) \in E_1$ and $v_2 \in V_2$ is a copy of $v_1\}$.*

The *SBG* graph has $2|V_1|$ nodes and $3|E_1|$ edges. A visualisation of a stacked break graph is shown in Figure 3.

For edges e in EB, the TTF $f_e(\tau)$ is redefined, given Δ_{FF}, the break time Δ_b and nearest road block $[\tau_{bs}, \tau_{be}]$:

$$f_e(\tau) = \begin{cases} \Delta_{\text{FF}} + \tau_{be} - \tau_{bs} & \text{if } \tau \geq \tau_{bs} - \Delta_{\text{FF}} \\ & \text{and } \tau < \tau_{be} - \Delta_b \\ \Delta_{\text{FF}} + \Delta_b - (\tau_{be} - \tau) & \text{if } \tau \geq \tau_{be} - \Delta_b \\ & \text{and } \tau < \tau_{be} \\ \Delta_{\text{FF}} + \Delta_b & \text{else} \end{cases}$$

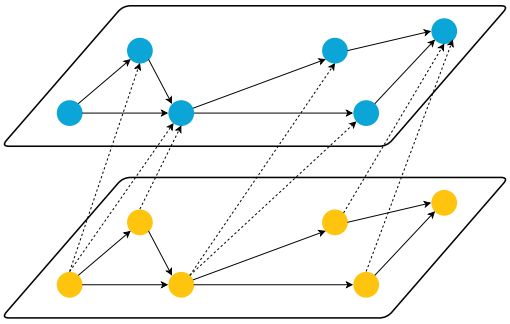

Figure 3: In this stacked break graph the original graph is below, the copy is on top, and break edges are dashed.

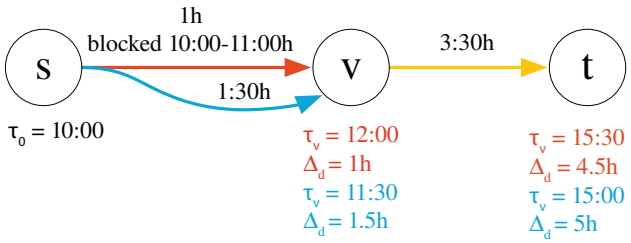

Figure 4: Let s be the start node with departure time 10:00, T is 4.5 hours, and Δ_b is 45 minutes. The arrival time at v is 11:30 via the lower route, and 12:00 via the upper route. Dijkstra only maintains the lower path at v. However, the best path towards destination t is the other one, since Δ_d of the lower path exceeds 4.5 hours and is therefore infeasible.

Furthermore, the shift duration function $s_e(\Delta_d, \tau)$ is redefined for all edges in EB. For these edges, the shift duration is reset after taking the break (so Δ_d is here irrelevant):

$$s_e(\tau) = \begin{cases} \Delta_{\mathrm{FF}} - \max(\tau_{bs} - \tau, 0) & \text{if } \tau \geq \tau_{bs} - \Delta_{\mathrm{FF}} \\ & \text{and } \tau < \tau_{be} - \Delta_b \\ \Delta_{\mathrm{FF}} & \text{if } \tau \geq \tau_{be} - \Delta_b \\ & \text{and } \tau < \tau_{be} \\ 0 & \text{else} \end{cases}$$

4.2 Single-label search

Running Dijkstra's shortest path algorithm on the presented Stacked Break Graph does not lead to an optimal route, since no notion is made of the consecutive driving time threshold and the optimal break planning path is not necessarily prefix optimal. This is shown by the example in Figure 4.

We therefore propose the following heuristic algorithm called Single-Label Break Planning (SL-BP). First we use Dijkstra's single-label algorithm to settle the lower layer of nodes (i.e., all routes without a break), followed by settling the upper layer. To prevent replacing routes by others that have an earlier arrival time but higher shift duration, the priority queue of the upper layer is ordered on lowest shift duration instead of arrival time. Such (possibly optimal) routes are still not maintained at the lower layer. However, this situation (i.e., Figure 4) is not expected to occur often: we expect that most problems occur with planning of breaks.

4.3 Optimal multi-label search

The single-label heuristic does not necessarily find optimal paths. Therefore, a modified version of the *Pareto* or *multi-label search* algorithm is applied (Hansen 1980). Each label represents a subpath and is defined as follows:

Definition 3 (Label)**.** *A label is a tuple $\langle u, \tau, \Delta_d, p \rangle$, representing a path to node u arriving at time τ with consecutive driving time Δ_d and having predecessor (label) p.*

Labels are included if they represented a path that is not dominated by another path.

Definition 4 (Domination)**.** *A path i is* dominated *by a path j, if $\tau_j \leq \tau_i$ and $\Delta_{d_j} \leq \Delta_{d_i}$, with at least one strict equality, thus $\tau_j < \tau_i$ or $\Delta_{d_j} < \Delta_{d_i}$.*

The list of labels to further explore is maintained in a priority queue PQ. The rest of the algorithm is equal to the original version (Hansen 1980). The full Optimal Break Planning (OBP) algorithm is shown in Algorithm 1.

Algorithm 1: OBP: a Pareto algorithm for the SBG

```
Initialize L[u] = ∅ for every node u ∈ V
l_s ← ⟨s, τ_0, Δ_{d_0}, ⊥⟩
L[s].add(l_s), PQ.add(l_s)
while !PQ.empty() do
    l_u = ⟨u, τ, Δ_d, l_p⟩ ← PQ.min()
    foreach e = (u, v) ∈ E do
        l_v ← ⟨v, τ + f_e(τ), s_e(Δ_d, τ), l_u⟩
        if s_e(Δ_d, τ) ≤ T then
            if l_v is not dominated by l ∈ L[v] then
                foreach l in L[v] do
                    if l is dominated by l_v then
                        L[v].remove(l)
                        PQ.remove(l)
                L[v].add(l_v), PQ.add(l_v)
return l ∈ L[t] having earliest arrival time τ
```

To prove the optimality of the algorithm, it is first shown that there always exists at least one Pareto prefix optimal path. Since Pareto prefix optimal paths are always found by a Pareto search algorithm (Hansen 1980), it is then shown that an optimal break planning path is always a Pareto optimal path. In these proofs, it is assumed that function $f(P_{st})$ represents the travel time of path P_{st} for some departure time τ_0, thus $\tau_t = \tau_0 + f(P_{st})$. Function $s(P_{st})$ represents the shift duration at node t of this path for the same departure time τ_0 and starting consecutive driving time Δ_{d_0}. These proofs are based on work of Batz (2013) and Ehrgott (2006).

Lemma 1. *Let $G = (V, E)$ be a SBG such that the TTFs f and shift duration functions s are non-negative. Consider $s, t \in V$, where t is reachable from s in G, even if considering drivers legislation rules. Then there always exist a Pareto optimal path $P_{st} = \langle s = u_1, u_2, \ldots, u_k = t \rangle$, such that for every $i = 1, \ldots, k$, the prefix path $P_{si} = \langle s = u_1, \ldots, u_i \rangle$ is a Pareto optimal path as well.*

Proof. Assume P_{si} is not a Pareto optimal path. Then there is a path $P'_{si} \neq P_{si}$ that dominates P_{si}, thus $f(P'_{si}) \leq$

$f(P_{si})$ and $s(P'_{si}) \leq s(P_{si})$ with at least one strict equality, thus $f(P'_{si}) < f(P_{si})$ or $s(P'_{si}) < s(P_{si})$. Therefore, a path P'_{st} consisting of P'_{si} followed by P_{it} (the subpath of P from i to t) is an $s-t$ path that contradicts the Pareto optimality of P_{st} due to the following three reasons: first, $f(P'_{st}) \leq f(P_{st})$, since it is not possible to arrive later at t by departing earlier at node i due to FIFO-property of the network. Second, $s(P'_{st}) \leq s(P_{st})$, since $s(P'_{si}) \leq s(P_{si})$ and $s(P_{it})$ is equal to the free flow travel time of this subpath, and thus independent of the departure time at node i. Third, it holds that $f(P'_{st}) < f(P_{st})$ or $s(P'_{st}) < s(P_{st})$, since the subpath toward node i also has at least one strict equality. Thus, path P_{st} is strictly dominated by path P'_{st}, violating the assumption of Pareto optimality of P_{st}, and there always exists a prefix path P_{si} that is Pareto optimal. □

The Pareto search algorithm always finds all prefix optimal paths if they exist (Hansen 1980), only ruling out paths with the same arrival time and shift duration. To complete the proof of optimality of the break planning algorithm, we now only need to prove that the algorithm indeed returns an optimal break planning path, implying that the optimal break planning path is a Pareto optimal path.

Theorem 1. *The Pareto search algorithm returns an optimal break planning path*

Proof. Let P_{st} be the path returned by the Pareto search algorithm, meaning that this path has the earliest arrival time of the set of paths to destination node t found by the algorithm. Now assume that this path P_{st} is not an optimal break planning path. This implies that there exists some path P'_{st} with $f(P'_{st}) < f(P_{st})$. But this implies that P'_{st} is a Pareto optimal path, since it is not dominated by any other path. However, the Pareto search algorithm finds all Pareto optimal paths (except those having identical arrival times and shift durations). So, it is not possible that such a path P'_{st} exists, and the path returned by the Pareto search algorithm is an optimal break planning path. □

The number of Pareto optimal paths is exponentially large in the worst case even with two criteria (Müller-Hannemann and Weihe 2006). Especially in detailed networks enormous numbers of paths are generated, leading to bad runtimes. Therefore, next we discuss a speed-up technique using contraction hierarchies which reduces the size of graph.

5 Speed-up using Time-Dependent Contraction Hierarchies

When solving VRP for trucking companies, shortest path computations are done extremely often. Therefore, a very fast algorithm is required. The algorithm that currently has the best runtimes for shortest path computation in road networks is contraction hierarchies. After the execution of an extensive preprocessing phase, it is able to compute routes in a few milliseconds. However, with contraction hierarchies two potential problems can be identified: all optimal break planning paths should be maintained during preprocessing, and correct travel times should be computed during querying if a break is taken on a shortcut edge.

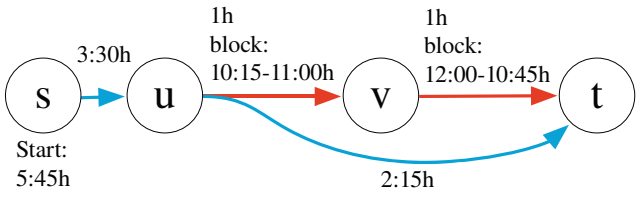

Figure 5: Preprocessing might remove optimal break planning paths: if v is contracted, the red edges are removed.

5.1 Preprocessing and optimal break planning

During preprocessing, only paths are maintained that provide a shortest path at any moment of time. To be able to compute optimal break planning paths during querying, alternatives that are optimal only in a specific time interval (related to a road block) should not be removed during preprocessing, as shown in the following example.

Example 2. *If node v is contracted during preprocessing from the graph shown in Figure 5, the path $\langle u, v, t\rangle$ is removed from the graph since it never provides an earliest arrival path between u and t. The edge between v and t can only be driven between 10:45 and 12:00 every day, causing at least 30 minutes of waiting time for travelling the complete segment. But if a trucks starts driving at 5:45 at node s heading for t, it will arrive at u at 9:15, just before the driving ban starts. Continuing toward v results in an arrival time of 10:15, after which a break of 45 minutes can be taken. The trip can be continued toward t at 11:00 giving an arrival time at t of 12:00. Taking the direct route between u and t results in a later arrival time of 12:15.*

As shown in the example, it might be the case that the optimal break planning path is not present in the preprocessed graph, resulting in non-optimal break planning paths independent of the chosen querying algorithm.

Let us first discuss two ideas for dealing with this problem: using the method by Bräuer, or preprocessing the SBG instead of the regular network.

Bräuer (2016) solves the problem by adapting the node ordering such that all parking lots are on top of the preprocessed tree, as explained in Section 2.3. However, this method results in worse runtimes of 16 seconds per query on average in a graph having 7 million nodes.

Preprocessing the SBG does not necessarily lead to maintaining all optimal break planning paths in the hierarchy. The optimal position of a break in the path depends on the start and destination locations, as well as the consecutive driving time. However, this information is not known during preprocessing.

Both these approaches thus do not lead to our objective of computing optimal break planning routes with very small querying times. However, we do observe that optimal break planning paths sometimes remain after preprocessing.

Theorem 2. *The optimal break planning path remains in the preprocessed graph, if any path can be driven at free flow travel time once, meaning that it is possible to drive any path at some moment without being delayed by a driving ban.*

Proof. Assume that path P'_{st} provides the earliest arrival time for τ_0 without considering break planning. Assume that another path P_{st} is not included in the preprocessed hierarchy, but does provide the optimal break planning path for τ_0. This is only possible if the time required to take a break is less, thus $f(P_{st}) - f_N(P_{st}) < f(P'_{st}) - f_N(P'_{st})$, with f_N indicating the normal travel time without considering breaks. But this implies that path P_{st} encounters a road block without break planning, which can be avoided or used efficiently when planning a break. But then, Δ_{FF} of P_{st} is smaller than Δ_{FF} of P'_{st}. However, if both routes can be driven in free flow travel time at one moment, P_{st} would provide the fastest path at some moment in time, and thus will still be present in the preprocessed hierarchy. □

Referring to Example 2: path $\langle s, u, v, t \rangle$ has an increased travel time of 15 minutes by adding a break, whereas path $\langle s, u, t \rangle$ has an increased travel time of 45 minutes. If $\langle s, u, v, t \rangle$ could be driven in free flow travel time (i.e., in 5:30 hours) at some time, it was not discarded during preprocessing and the optimal break planning path was maintained. Although it might seem obvious that each route can be driven at some time without having to wait for any road block, this is not guaranteed due to the possible creation of very long shortcut edges in the preprocessed graph.

In the remainder we therefore use the regular preprocessing method without modifications —with the chance that optimal break planning paths are removed— and we analyse in the experimental section the effect on optimality.

5.2 Maintaining road block information

By using the original preprocessed graph there is a possibility that potential optimal break planning paths are removed during preprocessing. However, even if this path was not removed, there is another problem for finding these paths: the travel times still need to be calculated in the case a break is scheduled on the edge. This is not trivial, as shown next:

Example 3. *Due to the contraction of node x during preprocessing, two shortcut edges are added between nodes u and v and vice versa, as shown in Figure 6. Now assume that a truck departs at 11:00 from u with an already consecutive driving time Δ_d of 4 hours, T of 4.5 hours and a break time Δ_b of 45 minutes. The corresponding TTF value indicates a travel time of 90 minutes at this departing time. However, the driver needs to take the mandatory break after 30 minutes of driving. This implies that the truck accidentally gets stuck in the road block at 13:00, just before reaching v. At the end, this leads to an arrival time of 15:15. On the other hand, if the truck was driving from v to u at the same time (which also implies a TTF value of 90 minutes), one can take the break halfway x and v from 11:30 till 12:15 and then continue to drive to x without facing the road block. This leads to an arrival time of 13:15, break time Δ_b later than expected.*

The example shows that planning a break lead to very different travel times, although the TTF values were equal. A method to compute correct travel times including a break needs to know the exact times of the road blocks on the road.

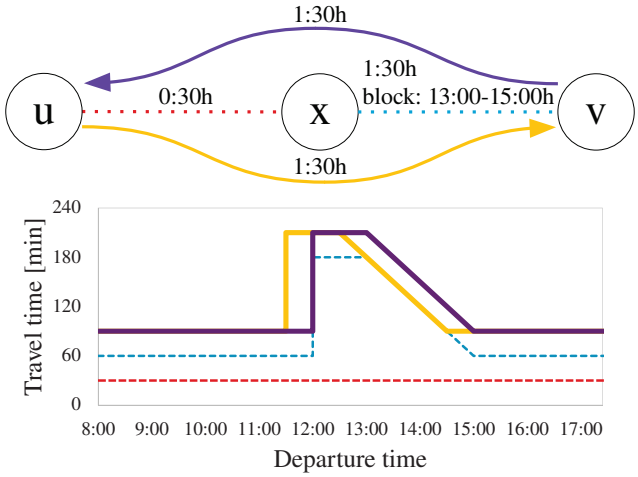

Figure 6: Contracting node X creates two shortcut edges from nodes u to v and v to u, including their respective travel time functions.

It should be noted that these block times cannot be derived from the TTF itself: due to the merging of edges with and without road blocks the typical road block shape as shown in Figure 1 may be shifted. However, the information on how much the block is shifted in the TTF can be used to derive the actual travel times if a break is planned on the route.

5.3 Single-label search

The single-label heuristic as introduced in Section 4.2 does not work correctly using TCH (time-dependent contraction hierarchies): the preprocessing step creates several long shortcut edges, resulting in multiple possibilities of planning a break along this edge, each having a very different travel time. Therefore, the following modification, called SL-TCH-BP, schedules breaks using two obvious strategies: planning a break as late as possible along the route (i.e., at T), or during a road block (which is retrieved using the maintained road block information). The strategy resulting in the earliest arrival time is returned as the best route.

5.4 Multi-label search

The single-label heuristic does not necessarily result in optimal routes, even if they are remained in the preprocessed graph (i.e., Theorem 2 holds). A better method (denoted by TCH-BP) is to adjust the Pareto search algorithm shown in Algorithm 1. The label tuple gets an additional boolean b, indicating whether a break has been taken or not. For a label to be dominated, b needs to have the same value. If a label without a break is relaxed, two labels are created instead of one (line 7): one without, and one with a break. The required travel times including the scheduled breaks are identical to the OBP Pareto search algorithm, with the addition of using the road block information as explained in Section 5.2. The quality and runtime of this heuristic algorithm, as well as the other algorithms introduced previously, are tested experimentally in the following section.

Table 1: Used road blocks in Europe

Country	Night block	Weekend block
Austria	22:00-5:00	Sat 15:00 - Sun 22:00
Czech Republic	-	Sun 13:00-22:00
France, Hungary	-	Sat 22:00 - Sun 22:00
Germany, Slovakia	-	Sun 0:00-22:00
Italy	-	Sun 8:00-22:00
Slovenia	-	Sun 8:00-21:00
Switzerland	22:00-5:00	Sun 0:00-22:00

Table 2: Results of 1D and 2D

	1D	2D
Total number of routes	10,000	10,000
Feasible routes	4042	7894
Routes with breaks	2645	3908
Routes improving	163 (6%)	664 (17%)
Avg. τ_t of improved routes	15:49 h	27:31 h
Avg. absolute improvement	2:45 h	5:14 h
Avg. relative improvement	84.9%	89.9%

6 Experimental Analysis

The two multi-label algorithms (OBP and TCH-BP), and their single-label variants (SL-BP and SL-TCH-BP) are evaluated to show the influence of optimal break planning compared to the industry standard reference algorithm. Additionally, the quality and runtime of the heuristic single-label variants and heuristic TCH-BP are shown.

6.1 Experimental setup

All algorithms are run on a detailed network of Europe, consisting of highways, secondary roads and having 7 million nodes and 13 million edges. Road blocks in Europe that are active throughout the year are considered. This implies blocks on Sundays in roughly half of the countries and night blocks in Austria and Switzerland, as shown in Table 1. In the used network, 41% of the links is affected.

Ideally test instances exactly represent the currently driven start-destination pairs by trucks. However, we could not find such a data set. Therefore, we generated our own test set as realistically as possible using data on freight transport provided by the ETIS-project (2014). This European database gives the amount of transported goods by trucks between each of the countries in Europe. To generate test instances we randomly picked start and end nodes from the given countries for each test instance according to the distribution from this database. In total, 10,000 start-destination pairs are generated.

The start time τ_0 for each of the test instances is picked randomly from a uniform distribution between Monday 0:00 and Sunday 23:59. It is assumed that all routes start with an already driven consecutive driving time Δ_{d_0} of 0:00.

For each query, two variations are run: simulating *one day* (referred to as *1D*) and *two days* (*2D*) of driving. 1D corresponds to the daily drivers legislation (European Parliament 2006) and has a maximum consecutive driving time T of 4.5 hours and a break time Δ_b of 45 minutes. 2D corresponds to driving two consecutive days, with threshold T 9 hours and night rest Δ_b 11 hours. The required breaks of 45 minutes are neglected in the two day variation.

All tests are run on a laptop, having 16GB memory and an Intel Core i7-3740QM 2.70GHz processor with 4 cores. The used algorithms and test instances are available online (van der Tuin 2018).

The resulting arrival times of the test instances are compared to the industry standard of planning breaks. This implies computing the shortest paths —not taking any break planning into account— and after that inserting breaks at the moment the driving threshold is exceeded. The arrival time is then recalculated by following the path from the break location. Finally, it is checked whether the threshold of consecutive driving time is not exceeded on the remainder of the path. This method is identical to the dark route shown in Example 1, and is referred to as the Industry Standard (IS). It should be noted that bad routes produced by this algorithm are not driven in reality, but changed manually by the logistics planner beforehand, for example by shifting the departure time. These modifications are not taken into account.

6.2 Influences on the arrival time

The 10,000 routes are computed for both IS and OBP, as well as for both 1D and 2D variations. Results are shown in Table 2. For 1D, 4042 routes out of 10,000 are feasible, meaning that these can be driven within the maximum allowed driving time of twice 4.5 hours. Only the routes that need a break (2645) can possibly be improved using the optimal break planning algorithm. Out of these, 163 instances show an improvement in travel time, with an average improvement of 2 hours and 45 minutes. For 2D, 664 routes improved with an average improvement of 5 hours and 14 minutes. On average, the travel time of the optimal break planning routes is 84.9% or 89.9% of the IS travel time for 1D and 2D, respectively.

It can be concluded that only part of the routes improve, but that these improving routes do show large differences in travel time. This is due to the long block durations: facing a block due to the scheduling of a break leads to waiting for at least 7 hours (the shortest block duration). In Europe, most road blocks are active on Sunday, sometimes starting at Saturday afternoon or evening. It is therefore not remarkable that the improving routes mostly start in the afternoon or in the weekend, as shown in Figure 7.

6.3 Quality of the TCH-OPB heuristic

As was shown with a counter example in Figure 5, the TCH-BP algorithm is not necessarily able to provide an optimal solution due to the removal of these optimal routes during preprocessing. However, the experiments show that all results of the TCH-BP algorithm are identical to the results of the optimal OBP algorithm. This means that for this specific road network and test set, there do not exist edges that provide shortest paths for optimal break planning but are removed during preprocessing. The quality of the heuristic is

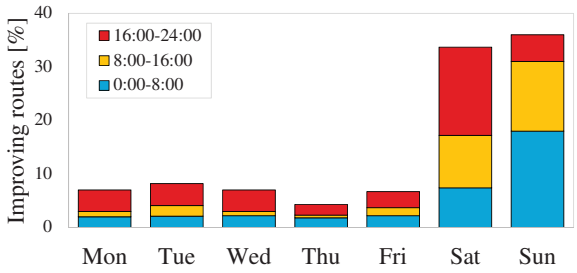

Figure 7: Start time distribution of improving routes of 2D

Table 3: Average and maximum query runtimes in seconds

	1D		2D	
	avg.	max.	avg.	max.
OBP	12.77	49.90	23.91	66.77
TCH-BP	0.064	0.670	0.154	1.014
SL-BP	10.18	31.66	18.25	47.41
SL-TCH-BP	0.0067	0.030	0.012	0.034

therefore considered to be very good.

6.4 Quality of the single-label heuristics

Both single-label algorithms give results in travel time that are identical to the optimal routes for all 10,000 routes. However, there do exist situations where potential good partial solutions are not maintained during preprocessing, that is, a route having a later arrival time but an earlier shift duration. This can be easily detected in the SL-BP algorithm, leading to 816 potential mistakes at the 1D variation and 2537 potential mistakes at the 2D variation. A possible solution would be to rerun these routes using the optimal algorithm.

For SL-TCH-BP it is not possible to record whether the solution is possibly non-optimal, since routes already may be discarded during preprocessing. Furthermore, the limited amount of paths available at a certain departure time in the hierarchy cause that paths found by the Pareto algorithm may not be discovered by its single-label variant.

6.5 Runtime analysis

As expected, the TCH-BP algorithm is able to perform the queries in much faster runtimes: 0.064 (1D) or 0.154 (2D) seconds instead of more than 12 seconds used by the OBP algorithm, as shown in Table 3. This is still much larger than the query times of less than 1.2 ms of the unmodified TCH algorithm. This is mainly explainable by the creation of multiple paths. SL-BP results only in a minor speed-up of the OBP algorithm. Although its quality is good, it resettles many nodes in the upper layer of the SBG due to the ordering of the priority queue. The SL-TCH-BP variant gives a much larger speed-up ($\approx$ 10 times faster than TCH-BP).

7 Conclusions

This paper addresses the problem of planning a single break while also taking driving bans in Europe into account. An optimal Pareto search algorithm and several heuristics to solve this problem are proposed. Optimal break planning routes are compared to the industry standard, which inserts breaks after the route is planned (such as shown in Figure 2). The experiments show that 6% of the one-day routes driven by trucks through Europe improve by optimal break planning. For two-day routes this is about 17 %. The savings of improved routes are significant: 2:45 hours (for a maximum driving time of 9 hours) or 5:14 hours (18 hours limit) compared to the industry standard reference algorithm.

To decrease computation times, a heuristic single-label algorithm (SL-BP) is introduced, but this gives only a small gain in runtime (18 instead of 24 seconds per query). Furthermore, in 25% of the 10,000 routes a potentially optimal solution is discarded during the execution of the algorithm. Rerunning only these using the optimal algorithm results in a longer average runtime than running only the optimal algorithm, and is therefore not advisable.

A more effective way of decreasing computation times is using a heuristic Pareto search using time-dependent contraction hierarchies (TCH-BP). This algorithm gives speed-ups of up to 200 times compared to the optimal algorithm, making it suitable to be used within the context of VRP. Additionally, the results are identical to the optimal routes. This suggests that the optimality condition (Theorem 2) is true for the used network. This is probably because all roads can be driven without delay during the week in daylight hours.

The heuristic single-label TCH variant (SL-TCH-BP) also produces identical results, within a further 10 times smaller runtime (0.012 instead of 0.154 seconds per query). Interestingly, these strategies of planning a break as late as possible or as efficiently as possible (i.e., during a road block) result in good routes for the considered test set. It seems that the inclusion of break planning in this basic form already leads to a large improvement compared to adding breaks after the route has been planned (as the industry standard). Since it is not straightforward to determine whether a potentially optimal solution is discarded during preprocessing or querying, an important direction for further research is to identify precisely why and when these strategies (do not) work.

Although planning a single break gives good insights in the need for optimally planning breaks, the presented methods are not useful in an industry application yet, since for longer routes multiple breaks need to be scheduled. However, the concept of the stacked break graph and the multi-label algorithms are easily extendible to a multi-layered graph each representing a different break taken. A disadvantage of this method is the fixed number of breaks during the route – which should be pre-specified as the number of layers. Another way of implementing multiple breaks would be to keep track of the truck drivers' status using several driving time counters, and adjusting the Pareto labels accordingly. The algorithm should then explore all eligible routes. Although enormous amounts of labels are expected to be generated, computation times can be shortened by introducing a heuristic that discards labels that are not likely to form good solutions, for example paths with more breaks and a higher consecutive driving time.

For now we conclude that optimal break planning leads

to a large gain in travel time, and by using the TCH Pareto search algorithm these routes can be computed fast and presumably optimal.

8 Acknowledgements

This work is based on the first author's master's thesis (2017) supported by ORTEC B.V.. We would like to thank the committee members C.G. Chorus, V.L. Knoop and S. van Cranenburgh of Delft University of Technology for their input during the thesis project.

References

Batz, G. V., and Sanders, P. 2012. Time-dependent route planning with generalized objective functions. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 7501 LNCS:169–180.

Batz, G. V.; Geisberger, R.; Sanders, P.; and Vetter, C. 2013. Minimum time-dependent travel times with contraction hierarchies. *Journal of Experimental Algorithmics* 18(1):1.1–1.43.

Bauer, R., and Delling, D. 2009. SHARC: Fast and robust unidirectional routing. *Journal of Experimental Algorithmics (JEA)* 2:13–26.

Bräuer, C., and Baum, M. 2016. Optimale zeitabhängige Pausenplanung für LKW-Fahrer mit integrierter Parkplatzwahl.

Dantzig, G. B. 1962. *Linear programming and extensions*. Princeton university press.

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

Ehrgott, M. 2006. *Multicriteria optimization*. Springer Science & Business Media.

European Parliament. 2006. 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 Co.

European Transport policy Information Systems. 2014. ETIS database. http://viewer.etisplus.eu/.

Geisberger, R.; Sanders, P.; Schultes, D.; and Delling, D. 2008. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 5038 LNCS(July):319–333.

Goldberg, A. V., and Harrelson, C. 2005. Computing the shortest path: A search meets graph theory. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '05, 156–165. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics.

Hansen, P. 1980. Bicriterion path problems. In *Multiple criteria decision making theory and application*. Springer. 109–127.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics* 4(2):100–107.

Ichoua, S.; Gendreau, M.; and Potvin, J. Y. 2003. Vehicle dispatching with time-dependent travel times. *European Journal of Operational Research* 144(2):379–396.

Kok, A.; Meyer, C.; Kopfer, H.; and Schutten, J. 2010. A Dynamic Programming Heuristic for the Vehicle Routing Problem with Time Windows and European Community Social Legislation. *Transportation Science* 44(4):442–454.

Lahyani, R.; Khemakhem, M.; and Semet, F. 2015. Rich vehicle routing problems: From a taxonomy to a definition. *European Journal of Operational Research* 241(1):1–14.

Müller-Hannemann, M., and Weihe, K. 2006. On the cardinality of the Pareto set in bicriteria shortest path problems. *Annals of Operations Research* 147(1):269–286.

Nannicini, G.; Delling, D.; Liberti, L.; and Schultes, D. 2008. Bidirectional A* search for time-dependent fast paths. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 5038 LNCS(2):334–346.

Orda, A., and Rom, R. 1990. Shortest-Path and Minimum-Delay Algorithms in Networks with Time-Dependent Edge-Length. *Journal of the Association for Computing Machinery* 37(3):607–625.

Sommer, C. 2014. Shortest-path queries in static networks. *ACM Computing Surveys* 46(4):1–31.

van der Tuin, M. 2017. Time-dependent earliest arrival routes with drivers legislation and preferences. Master's thesis, Delft University of Technology.

van der Tuin, M. 2018. Break planning with multi-label Dijkstra and time-dependent contraction hierarchies. https://doi.org/10.4121/uuid:508b228d-e542-40c9-95cc-1b985d86e3b2.