Efficient Long-Haul Truck Driver Routing

Master Thesis of

Max Oesterle

At the Department of Informatics
Institute of Theoretical Informatics

Reviewers:	Dr. rer. nat. Torsten Ueckerdt
	?
Advisors:	Tim Zeitz
	Alexander Kleff
	Frank Schulz

Time Period: 15th January 2022 – 15th July 2022

Statement of Authorship

Ich versichere wahrheitsgemäß, die Arbeit selbstständig verfasst, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde sowie die Satzung des KIT zur Sicherung guter wissenschaftlicher Praxis in der jeweils gültigen Fassung beachtet zu haben.

Karlsruhe, May 25, 2022

Abstract

A short summary of what is going on here.

Deutsche Zusammenfassung

Kurze Inhaltsangabe auf deutsch.

Contents

1. Introduction

1. introduction routing, spp
2. practical improvements
3. extensions of the spp similar to this thesis and arising problems
4. outline of algorithm and work on which is based
5. outline of thesis

2. Preliminaries

1. introduction, a*, bidir
2. hierarchies in street networks and first algos
3. ch
4. ch potentials
5. what even is a core ch

3. Problem and Definitions

The long-haul truck driver routing problem is an extension to the common shortest path problem (SPP) which is defined as follows. Let $G = (V, E, \omega)$ be a graph where V is the set of nodes, E is the set of edges (u, v) with $u, v \in V$, and ω is the weight function $\omega : E \to \mathbb{R}_{\geq 0}$ which assigns each edge a nonnegative weight or length. Given a start node $s \in V$ and a target node $t \in V$, the SPP searches a shortest path p from s to t, i.e., a path $p = \langle s = v_0, v_1, ..., t = v_k, \rangle$ with $(v_i, v_{i+1}) \in E$ and minimal $len(p) = \sum_{i=0}^{k-1} \omega((v_i, v_{i+1}))$.

We introduce a set $P \subseteq V$ of parking nodes and a set R of driving time constraints r_i. Each driving time constraint is defined by a maximum allowed driving time $r_{i,d}$ and a pause time $r_{i,p}$. Thereby, the driving time constraints define a relation $r_i \leq r_{i+1}$ with $r_i \leq r_j \implies r_{i,d} \leq r_{j,d} \wedge r_{i,p} \leq r_{j,p} \forall i, j$. In words, a greater or equal driving time restriction has a longer or equal driving and pause time and there must be no restriction r_i with a longer driving time limit, but shorter pause time than another restriction r_j. Before exceeding a driving time of $r_{i,d}$, the driver must stop and pause for a time of at least $r_{i,p}$. Afterwards, the driver is allowed to drive for a maximum time of $r_{i,d}$ again without stopping. Stops can only take place at nodes $v \in P$.

A path p from s to t now includes not only the sequence of visited nodes $p = \langle s = v_0, v_1, ..., t = v_k, \rangle$, but also a pause time $\rho : p \to \{0, r_{i,d}\}$ for each node i. It is $\rho(v_i) = 0$ $\forall v_i \notin P$.

Definition 3.1 (Driving Time)**.** *The driving time $len_{dt}(p)$ of a path $p = \langle s = v_0, v_1, ..., t = v_k, \rangle$ is defined analogously to the length of a path in the ordinary SPP. It is $len_{dt}(p) = \sum_{i=0}^{k-1} \omega((v_i, v_{i+1}))$.*

Definition 3.2 (Travel Time)**.** *The travel time $len_{tt}(p)$ of a path $p = \langle s = v_0, v_1, ..., t = v_k, \rangle$ is defined as the sum of the driving time of the path and the total accumulated pause time $len_{tt}(p) = len_{dt}(p) + \sum_{i=0}^{k} \rho(v_i)$.*

Definition 3.3 (Path Compliance)**.** *A valid path $p = \langle s = v_0, v_1, ..., t = v_k, \rangle$ must comply with the driving time restrictions R. The path p complies with R if it complies with all $r_l \in R$.*

Let v_i be the starting node s or any node on the path with $\rho(v_i) \geq r_{l,p}$ and let v_j be the target node t or any node on the path with $\rho(v_j) \geq r_{l,p}$ and $i < j$. Then let q be the subpath of p from v_i to v_j. A path complies with driving time restriction r_l if $len_{dt}(q) < r_{d,l}$ for all possible subpaths q or there is a node v_m on q with $\rho(v_m) \geq r_{l,p}$ and $i < m < j$.

The long-haul truck driver routing problem now can be defined as follows.

Long-Haul Truck Driver Routing

Input:	A graph $G = (V, E, \omega)$, a set of parking nodes $P \subseteq V$, a set of driving time constraints R, and start and target nodes $s, t \in V$
Problem:	Find the path p from s to t in G which minimizes travel time $len_{tt}(p)$ and complies with the driving time restrictions R.

In many practical applications, the number of different driving time constraints is limited to only one or two constraints, i.e., $|R| = 1$ or $|R| = 2$. Therefore, we will often only consider one of these special cases.

4. Algorithm

In this chapter, we introduce a labeling algorithm which solves the long-haul truck driver routing problem. We then describe extensions of the base algorithm to achieve better runtimes on road networks in practice. At first, we will restrict the problem to one driving time constraint for simplicity and drop that constraint later.

4.1. Dijkstra's Algorithm with One Driving Time Constraint

Dijkstra's algorithm solves the shortest path problem by maintaining a queue Q of nodes with ascending tentative distance from the starting node s, and iteratively settling the node with the smallest distance. When Dijkstra settles a node u, it tests if the distance to all neighbor nodes v with $(u, v) \in E$ can be improved by choosing the current node as a predecessor. We say it *relaxes* all edges $(u, v) \in E$. The search can be stopped if the target node t was removed from the queue. TODO REFERENCE PROOF

We will adapt Dijkstra's algorithm for solving the long-haul truck driver routing problem with one driving time constraint r and abbreviate this restriction of the problem *1-DTC*. While Dijkstra's algorithm manages a queue of nodes and assigns each node one tentative distance, our algorithm manages a queue Q of labels and a set $L(v)$ of labels for each node $v \in V$.

Labels represent a possible path, respectively a possible solution for a query from s to the node they belong to. A label l contains

- $d_0(l)$, the total travel time from the starting node s
- $d_1(l)$, the driving time since the last pause
- $pred(l)$, its preceding label

4.1.1. Settling a Label

In contrast to Dijkstra, the search *settles* a label $l \in L(u)$ in each iteration instead of a node u. When settling a label, the search first removes l from the queue. Similar to Dijkstra, it then relaxes all edges $(u, v) \in E$ with $l \in L(u)$ as shown in fig. 4.1.

Relaxing an edge consists of the three steps label *propagation*, *pruning* and *dominance* checks.

```
1 Procedure settleNextLabel():
2     $l \leftarrow$ Q.deleteMin()
3     forall $(u, v) \in E$ do
4         relaxEdge($(u, v), l$)
```

Figure 4.1.: Settling a label $l \in L(u)$ removes the label from the queue and relaxes all the outgoing edges of u.

Label Propagation.

Labels can be propagated along edges. Let $l \in L(u)$ be a label at u and $(u, v) = e \in E$, then l can be propagated to v resulting in a label l' with $d_0(l') = d_0(l) + \omega(e)$, $d_1(l') = d_1(l) + \omega(e)$, and $pred(l') = l$.

Label Pruning.

After propagating a label, we discard the label if it violates the driving time constraint r. That is, if $d_1(l) \geq r_d$.

Label Dominance

In general, it is no longer clear when a label presents a better solution than another label since it now contains two distance values. A label l at a node v might represent a path with a shorter travel time from s to v than another label m, but a shorter remaining driving time budget $r_d - d_1(l)$. The label l yields a better solution for a query s-v, but this does not imply that it is part of a better solution for a query from s to t. It might not even yield a path to t at all while m reaches the target due to the greater remaining driving time budget. In one case we can proof that a label $l \in L(v)$ cannot yield a better solution than a label $m \in L(v)$. We say m *dominates* l.

Definition 4.1 (Label Dominancef for 1-DTC)**.** *A label $l \in L(v)$ dominates another label $m \in L(v)$ if $d_0(m) \geq d_0(l)$ and $d_1(m) \geq d_1(l)$.*

If a label $l \in L(v)$ is dominated by another label $m \in L(v)$, then m represents a path from s to t with a shorter or equal total travel time and longer remaining driving time budget until the next pause. Therefore, in each solution which uses the label l, l can trivially be replaced by the label m. The solution will still comply with the driving time constraint r and yield a shorter or equal total travel time, so we are allowed to simply discard dominated labels in our search.

Definition 4.2 (Pareto-Optimal Label)**.** *A label $l \in L(v)$ is pareto-optimal if it is not dominated by any other label $m \in L(v)$.*

We will only add a label l to a label set $L(v)$ if it is pareto-optimal. Labels $m \in L(v)$ are then removed from $L(v)$ if l dominates them. $L(v)$ therefore is the set of pareto-optimal solutions at v. In fig. 4.2 we define the procedure removeDominated(l) as a label set operation.

We now use removeDominated to define the procedure relaxEdge′ as described in fig. 4.3 which propagates a label along an edge and updates the neighbor node's label set if necessary.

```
Procedure removeDominated($l$):
    forall $m \in L$ do
        if $l$ dominates $m$ then
            L.remove ($m$);
```

Figure 4.2.: The operation which is defined on a label set L, removes all labels from the set which are dominated by the label l.

```
Procedure relaxEdge′((u,v), $l$):
    if $d_1(l) + \omega(u,v) < r_d$ then
        $l' \leftarrow \{(d_0(l) + \omega(u,v), d_1(l) + \omega(u,v)), l\}$
        if $l'$ is not dominated by any label in $\mathsf{L}(v)$ then
            $\mathsf{L}(v)$.removeDominated($l'$)
            $\mathsf{L}(v)$.insert($l'$)
            Q.insert($l'$)
```

Figure 4.3.: Relaxing an edge $(u, v) \in E$ when settling a label $l \in L(u)$.

4.1.2. Parking at a Node

The procedure relaxEdge′ does not account for parking nodes. When propagating a label $l \in L(u)$ along an edge $(u, v) \in E$ and $v \in P$ then we have to consider pausing at v. Since we do not know if pausing at v or continuing without a pause is the better solution, we generate both labels and them to label set $L(v)$ and the queue Q as defined in fig. 4.4.

```
Procedure relaxEdge((u,v), $l$):
    if $d_1(l) + \omega(u,v) < r_d$ then
        D.insert($(d_0(l) + \omega(u,v), d_1(l) + \omega(u,v), l)$)
        if $v \in P$ then
            D.insert($(d_0(l) + \omega(u,v) + d_p, 0, l)$)
        forall $l' \in D$ do
            if $l'$ is not dominated by any label in $\mathsf{L}(v)$ then
                $\mathsf{L}(v)$.removeDominated($l'$)
                $\mathsf{L}(v)$.insert($l'$)
                Q.insert($l'$)
```

Figure 4.4.: Relaxing an edge $(u, v) \in E$ when settling a label $l \in L(u)$ with regard to parking nodes.

4.1.3. Initialization and Stopping Criterion

We initialize the label set $L(s)$ of s and the queue Q with a label which only contains distances of zero and a dummy element as a predecessor. We stop the search when t was removed from Q. The definition of the final algorithm 4.1 Dijkstra+1-DTC is now trivial.

Algorithm 4.1: Dijkstra+1-DTC

```
1 Returnreturn
  Input: Graph $G = (V, E, \omega)$, set of parking nodes $P \subseteq V$, set of driving time
         constraints $R$, start and target nodes $s, t \in V$
  Data: Priority queue Q, per node set $\mathsf{L}(v)$ of labels for all $v \in V$
  Output: Shortest travel time from $s$ to $t$ and corresponding $s$-$t$-path given by the
          predecessors of the label $l_t \in L(t)$
  // Initialization
2 Q.insert($(0, 0, \perp)$)
3 L($s$).insert($(0, 0, \perp)$)
  // Main loop
4 while Q is not empty do
5     settleNextNode()
6     if minimum of Q is label at t then
7         return
```

Algorithm 4.2: A*+1-DTC

```
1 Returnreturn
  Input: Graph $G = (V, E, \omega)$, set of parking nodes $P \subseteq V$, set of driving time
         constraints $R$, start and target nodes $s, t \in V$, potential $pot()$
  Data: Priority queue Q, per node set $\mathsf{L}(v)$ of labels for all $v \in V$
  Output: Shortest travel time from $s$ to $t$ and corresponding $s$-$t$-path given by the
          predecessors of the label $l_t \in L(t)$
  // Initialization
2 Q.insert($pot((0, 0, \perp))$)
3 L($s$).insert($(0, 0, \perp)$)
  // Main loop
4 while Q is not empty do
5     settleNextNode()
6     if minimum of Q is label at t then
7         return
```

4.1.4. Correctness

TODO correctness stopping criterion

4.2. A* with One Driving Time Constraint

TODO dijkstra slow blabla therefore with ch potential just add potential everywhere

TODO settlenextlabel is like in fig. 4.1 but relax ege is now as in fig. 4.5

4.2.1. Potential for One Driving Time Constraint

Given a target node t, the CH potential $\pi_{t,ch}$ yields a perfect estimate for the distance $d_{direct}(v, t)$ from v to t without regard for driving time restrictions and pauses. A lower bound for the time $d(v, t)$ from v to t with breaks due to the driving time limit can be calculated by taking the minimum necessary amount of breaks on the shortest path into account:

```
Procedure relaxEdge((u,v), l):
    if $d_1(l) + \omega(u,v) < r_d$ then
        D.insert($(d_0(l) + \omega(u,v), d_1(l) + \omega(u,v), l)$)
        if $v \in P$ then
            D.insert($(d_0(l) + \omega(u,v) + d_p, 0, l)$)
        forall $l' \in D$ do
            if $l'$ is not dominated by any label in L($v$) then
                L($v$).removeDominated($l'$)
                L($v$).insert($l'$)
                Q.insert($pot(l')$)
```

Figure 4.5.: Relaxing an edge with regard to the potential.

$$\pi'_t(v) = \left\lfloor \frac{d_{direct}(v,t)}{t_d} \right\rfloor * t_p + d_{direct}(v,t)$$

A node potential is called *feasible* if it does not overestimate the distance of any edge in the graph, i.e.

$$len(u,v) - pot(u) + pot(v) \geq 0 \quad \forall (u,v) \in E \tag{4.1}$$

Following example of a query using the graph in Fig. 4.6 shows that π'_t is not feasible. With a driving time limit of 6 and a pause time of 1, the potential here will yield a value $\pi_t(s) = 8$ since the potential includes the minimum required pause time for a path from s to t. Consequently, with $\pi_t(v) = 5$ and $len(s,v) = 2$, $len(s,v) - \pi_t(s) + \pi_t(v) = -1$.

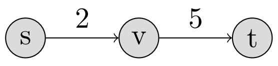

Figure 4.6.: A graph with the potential to break the potential.

A variant of the potential accounts for the distance $d(p,v)$ with p being the last parking node that was used for a pause to calculate the minimum required pause time on the v-t path. Since the potential now uses information from a label l with $l \in L(v)$, it no longer is a node potential but also depends on the chosen label at v.

$$\begin{aligned}\pi_t(l,v) &= \left\lfloor \frac{d_{direct}(p,v) + d_{direct}(v,t)}{t_d} \right\rfloor * t_p + d_{direct}(v,t) \\ &= \left\lfloor \frac{d_1(l) + d_{direct}(v,t)}{t_d} \right\rfloor * t_p + d_{direct}(v,t)\end{aligned}$$

Since the potential π_t now uses label information it no longer is a node potential and the feasibility definition as defined in inequality 4.1 can no longer be applied. We still want to use potential and label information to calculate lower bound estimates for the length of paths.

Lemma 4.3. *Let $p = \langle s = v_0, v_1, ..., t = v_k, \rangle$ be a path with labels l_i at nodes v_i. Then $d_0(l_{i-1}) + \pi_t(l_{i-1}, v_{i-1}) \leq d_0(l_i) + \pi_t(l_i, v_i)$.*

The lower bound estimate for the length of the entire path to which a label belongs can only increase when propagating labels to a next node.

Proof. Given a Graph $G = (V, E)$ with a set of parking nodes $P \subseteq V$, let $p = \langle s = v_0, v_1, ..., t = v_k, \rangle$ be a path in G with labels l_i at nodes v_i. Let $p, q \in P \cup \{s\}$ the last parking node which was used by label l_{i-1} and l_i or s, if no parking node was used.

$$\begin{aligned}
d_0(l_{i-1}) + \pi_t(l_{i-1}, v_{i-1}) &= d_0(l_{i-1}) + \left\lfloor \frac{d_1(l_{i-1}) + d_{direct}(v_{i-1}, t)}{t_d} \right\rfloor * t_p + d_{direct}(v_{i-1}, t) \\
&= d_0(l_{i-1}) + \left\lfloor \frac{d_{direct}(p, v_{i-1}) + d_{direct}(v_{i-1}, t)}{t_d} \right\rfloor * t_p + d_{direct}(v_{i-1}, t) \\
&= d(s, p) + d_{direct}(p, v_{i-1}) \\
&\quad + \underbrace{\left\lfloor \frac{d_{direct}(p, v_{i-1}) + d_{direct}(v_{i-1}, t)}{t_d} \right\rfloor * t_p}_{\text{minimum required pause time on p-t subpath}} + d_{direct}(v_{i-1}, t)
\end{aligned} \tag{4.2}$$

Case 1: $p = q$

$$\begin{aligned}
d_{direct}(p, v_{i-1}) + d_{direct}(v_{i-1}, t) &= d_{direct}(p, v_{i-1}) + len(v_{i-1}, v_i) + d_{direct}(v_i, t) \\
&= d_{direct}(q, v_{i-1}) + len(v_{i-1}, v_i) + d_{direct}(v_i, t) \\
&= d_{direct}(q, v_i) + d_{direct}(v_i, t)
\end{aligned} \tag{4.3}$$

With equations 4.2 follows

$$\begin{aligned}
d_0(l_{i-1}) + \pi_t(l_{i-1}, v_{i-1}) &= d(s, p) + d_{direct}(p, v_{i-1}) + d_{direct}(v_{i-1}, t) \\
&\quad + \left\lfloor \frac{d_{direct}(p, v_{i-1}) + d_{direct}(v_{i-1}, t)}{t_d} \right\rfloor * t_p \\
&= d(s, q) + d_{direct}(q, v_i) + d_{direct}(v_i, t) \\
&\quad + \left\lfloor \frac{d_{direct}(q, v_{i1}) + d_{direct}(v_i, t)}{t_d} \right\rfloor * t_p \\
&= d_0(l_i) + \pi_t(l_i, v_i)
\end{aligned} \tag{4.4}$$

Case 2: $p \neq q$. In this case, $q = v_i$ and $d(p, v_i) = d(p, q) = d_{direct}(p, v_i) + t_p =$. With 4.2 follows

$$\begin{aligned}
d_0(l_{i-1}) + \pi_t(l_{i-1}, v_{i-1}) &= d(s,p) + d_{direct}(p, v_{i-1}) + d_{direct}(v_{i-1}, t) \\
&\quad + \left\lfloor \frac{d_{direct}(p, v_{i-1}) + d_{direct}(v_{i-1}, t)}{t_d} \right\rfloor * t_p \\
&= d(s,p) + d_{direct}(p, v_i) + d_{direct}(v_i, t) \\
&\quad + \left\lfloor \frac{d_{direct}(p, v_i) + d_{direct}(v_i, t)}{t_d} \right\rfloor * t_p \\
&\leq d(s,p) + d(p,q) - t_p + d_{direct}(v_i, t) \\
&\quad + \left\lfloor \frac{d_{direct}(v_i, t)}{t_d} \right\rfloor * t_p + t_p \qquad (4.5) \\
&= d(s,q) + 0 + d_{direct}(v_i, t) \\
&\quad + \left\lfloor \frac{0 + d_{direct}(v_i, t)}{t_d} \right\rfloor * t_p \\
&= d(s,q) + d_{direct}(q, v_i) + d_{direct}(v_i, t) \\
&\quad + \left\lfloor \frac{d_{direct}(q, v_i) + d_{direct}(v_i, t)}{t_d} \right\rfloor * t_p \\
&= d_0(l_i) + \pi_t(l_{i1}, v_i)
\end{aligned}$$

□

Lemma 4.4. *The potential $\pi_t(l, v)$ of a label l at a node v is a lower bound for the distance including pauses from v to t.*

Proof. Let $p = \langle s = v_0, v_1, ..., t = v_k, \rangle$ be a path with labels l_i at nodes v_i. With $d_0(l_{i-1}) + \pi_t(l_{i-1}, v_{i-1}) \geq d_0(l_i) + \pi_t(l_i, v_i)$ for all edges on p, the total length $len(p)$ of the path must follow $\pi_t(l_i, v_i) \leq len(p) + \pi_t(l_k, t) \Leftrightarrow l(p) \geq \pi_t(l_i, v_i) - \pi_t(l_k, t)$. Since $\pi_t(l_k, t) = 0$, $l(p) \geq \pi_t(l_i, v_i)$ holds. □

Theorem 4.5. *The search can be stopped when the first label at t is removed from the queue.*

Proof. When a label l at t is removed from the queue during a s-t query, all remaining label m of a node v in the queue fulfill $d_0(t) + \pi_t(l, t) \leq d_0(v) + \pi_t(m, v)$. Assume that $d_0(t)$ is not the shortest distance from s to t. Then, a shorter path $p = \langle s = v_0, v_1, ..., t = v_k, \rangle$ exists which uses at least one unsettled label $m \in L(v_i)$. Since l was already removed from the queue, $d_0(t) = d_0(t) + \pi_t(l, t) \leq d_0(v) + \pi_t(m, v) \leq l(p)$ which contradicts the assumption that p yields a shorter s-t distance than $d_0(t)$. □

4.3. Multiple Driving Time Constraints

Dijkstra's algorithm with one driving time constraint (1-DTC) can easily be adapted to handle multiple driving time constraints r_i. We abbreviate the generalized problem *n-DTC*. For a number of n driving time constraints, a label now contains the total travel time d_0 and n driving time values $d_1, ..., d_n$. Each d_i represents the driving time since the last pause at a node v with pause time $\rho(v) \geq r_{i,p}$. Pausing at a node occurs with one of the available pause times $r_{i,p}$ of a driving time restriction $r_i \in R$. Pausing with an arbitrary pause time is possible but yields longer travel times and no advantage. When a path pauses at v for a time $r_{i,p}$, the corresponding label $l \in L(v)$ has $d_j(l) = 0$ for all $0 < j \leq i$ since the pauses with shorter pause times are *included* in the longer pause.

Label Propagation.

Label propagation simply extends the component-wise addition of the edge weight to all elements of the distance vector. Let $l \in L(u)$ be a label at u and $(u, v) = e \in E$, then l can be propagated to v resulting in a label $l^{'}$ with $d_i(l^{'}) = d_i(l) + \omega(e)\ \forall i \leq |R|$, and $pred(l^{'}) = l$.

Label Pruning.

The pruning rule for driving time constraints generalizes is a similar way. A label is discarded if $d_i(l) \geq r_{d,i}$ for any i with $0 < i \leq |R|$.

Label Dominance

At last, label dominance can be generalized to multiple driving time constraints as follows.

Definition 4.6 (Label Dominance). *A label $l \in L(v)$ dominates another label $m \in L(v)$ if $d_i(m) \geq d_i(l)\ \forall i \leq |R|$.*

4.4. Potential for Multiple Driving Time Constraints

In section 4.2.1 we defined the potential $\pi_t(l, v)$ to extend Dijkstra to an A* search with one driving time constraint. We will now generalize π_t for the use with an arbitrary number n of driving time constraints.

To calculate $\pi_t(l, v)$, we used the distance without regard for pausing from v to t and the distance $d_1(l)$ to calculate a lower bound for the amount of necessary pauses until we reach the target node. With one driving time restriction it is

$$minimum\ number\ of\ pauses = \left\lfloor \frac{d_1(l) + d_{direct}(v, t)}{t_{d,i}} \right\rfloor.$$

We now have to calculate the lower bound with respect to the different driving time constraints. How many pauses of which duration do we need at least to comply with all driving time restrictions r_i? We need a smaller or equal amount of pauses for longer driving time restrictions since they have a longer maximum allowed driving time $r_{d,i}$. At the same time, a pause of length $r_{p,i}$ will also include a pause of lengths $r_{p,j}$ with $j < i$. We therefore start with calculating the amount of necessary pauses of the longest restriction r_n. Then, we calculate how many pauses we need at least for the next smaller constraint and subtract the amount of pauses we just calculated for r_n. We repeat this process for each remaining driving time restriction and accumulate the product of number of pauses for r_i and its corresponding pause time $r_{p,i}$ to yield the minimum total pause time on the remaining path to t. Finally, we add the raw driving time $d_{direct}v, t$. We now can define π_t as

$$\pi_t(v, l) = d_{direct}(v, t) + \sum_{i=1}^{n} n_i * t_{p,i}$$

with

$$n_i = \begin{cases} \left\lfloor \dfrac{d_i(l) + d_{direct}(v, t)}{t_{i,d}} \right\rfloor - \left\lfloor \dfrac{d_{i+1}(l) + d_{direct}(v, t)}{t_{d,i+1}} \right\rfloor & 0 < i < n \\ \left\lfloor \dfrac{d_i(l) + d_{direct}(v, t)}{t_d} \right\rfloor & i = n \end{cases}$$

The value of n_i for $0 < i < n$ can also be calculated by reusing already computed values n_j with $j > i$:

$$n_i = \begin{cases} \left\lfloor \dfrac{d_i(l) + d_{direct}(v,t)}{t_{i,d}} \right\rfloor - \sum\limits_{j=i+1}^{n} n_j & 0 < i < n \\ \left\lfloor \dfrac{d_i(l) + d_{direct}(v,t)}{t_d} \right\rfloor & i = n \end{cases}$$

If lemma 4.3 still holds then, lemma 4.4 and theorem 4.5 follow as a consequence.

Lemma 4.7. *Lemma 4.3 still holds for an arbitrary number n of driving time constraints.*

Proof. Given a Graph $G = (V, E)$ with a set of parking nodes $P \subseteq V$, let $p = \langle s = v_0, v_1, ..., t = v_k, \rangle$ be a path in G with labels l_i at nodes v_i. Let $p, q \in P \cup \{s\}$ the last parking node which was used by label l_{i-1} and l_i or s, if no parking node was used.

$$\begin{aligned} d_0(l_{i-1}) + \pi_t(l_{i-1}, v_{i-1}) &= d_0(l_{i-1}) + \left\lfloor \frac{d_1(l_{i-1}) + d_{direct}(v_{i-1}, t)}{t_d} \right\rfloor * t_p + d_{direct}(v_{i-1}, t) \\ &= d_0(l_{i-1}) + \left\lfloor \frac{d_{direct}(p, v_{i-1}) + d_{direct}(v_{i-1}, t)}{t_d} \right\rfloor * t_p + d_{direct}(v_{i-1}, t) \\ &= d(s, p) + d_{direct}(p, v_{i-1}) \\ &\quad + \underbrace{\left\lfloor \frac{d_{direct}(p, v_{i-1}) + d_{direct}(v_{i-1}, t)}{t_d} \right\rfloor * t_p}_{\text{minimum required pause time on p-t subpath}} + d_{direct}(v_{i-1}, t) \end{aligned} \tag{4.6}$$

Case 1: $p = q$

$$\begin{aligned} d_{direct}(p, v_{i-1}) + d_{direct}(v_{i-1}, t) &= d_{direct}(p, v_{i-1}) + len(v_{i-1}, v_i) + d_{direct}(v_i, t) \\ &= d_{direct}(q, v_{i-1}) + len(v_{i-1}, v_i) + d_{direct}(v_i, t) \\ &= d_{direct}(q, v_i) + d_{direct}(v_i, t) \end{aligned} \tag{4.7}$$

With equations 4.6 follows

$$\begin{aligned} d_0(l_{i-1}) + \pi_t(l_{i-1}, v_{i-1}) &= d(s, p) + d_{direct}(p, v_{i-1}) + d_{direct}(v_{i-1}, t) \\ &\quad + \left\lfloor \frac{d_{direct}(p, v_{i-1}) + d_{direct}(v_{i-1}, t)}{t_d} \right\rfloor * t_p \\ &= d(s, q) + d_{direct}(q, v_i) + d_{direct}(v_i, t) \\ &\quad + \left\lfloor \frac{d_{direct}(q, v_{i1}) + d_{direct}(v_i, t)}{t_d} \right\rfloor * t_p \\ &= d_0(l_i) + \pi_t(l_i, v_i) \end{aligned} \tag{4.8}$$

Case 2: $p \neq q$. In this case, $q = v_i$ and $d(p, v_i) = d(p, q) = d_{direct}(p, v_i) + t_p =$. With 4.6 follows

$$
\begin{aligned}
d_0(l_{i-1}) + \pi_t(l_{i-1}, v_{i-1}) &= d(s,p) + d_{direct}(p, v_{i-1}) + d_{direct}(v_{i-1}, t) \\
&\quad + \left\lfloor \frac{d_{direct}(p, v_{i-1}) + d_{direct}(v_{i-1}, t)}{t_d} \right\rfloor * t_p \\
&= d(s,p) + d_{direct}(p, v_i) + d_{direct}(v_i, t) \\
&\quad + \left\lfloor \frac{d_{direct}(p, v_i) + d_{direct}(v_i, t)}{t_d} \right\rfloor * t_p \\
&\leq d(s,p) + d(p,q) - t_p + d_{direct}(v_i, t) \\
&\quad + \left\lfloor \frac{d_{direct}(v_i, t)}{t_d} \right\rfloor * t_p + t_p \\
&= d(s,q) + 0 + d_{direct}(v_i, t) \\
&\quad + \left\lfloor \frac{0 + d_{direct}(v_i, t)}{t_d} \right\rfloor * t_p \\
&= d(s,q) + d_{direct}(q, v_i) + d_{direct}(v_i, t) \\
&\quad + \left\lfloor \frac{d_{direct}(q, v_i) + d_{direct}(v_i, t)}{t_d} \right\rfloor * t_p \\
&= d_0(l_i) + \pi_t(l_{i1}, v_i)
\end{aligned} \tag{4.9}
$$

□

4.5. Core Contraction Hierarchy Variant

4.5.1. Building the Contraction Hierarchy

4.6. Combining A* and Core Contraction Hierarchy

Algorithm 4.3: CORE-CH WITH DRIVING TIME CONSTRAINTS

```
Input: Graph $G = (V, E, \omega)$, parking nodes $P \subseteq V$, driving time restriction $r$,
       potential pot(), source node $s \in V$
Data: Priority queue Q, per node priority queue L($v$) of labels for all $v \in V$
Output: Distances for all $v \in V$, tree of allowed shortest paths according to the
        restriction $r$ from $s$, given by $l_{pred}$
// Initialization
1  Q.INSERT($s, (0, 0)$)
2  L($s$).INSERT($(\perp, \perp)$, pot($(0,0)$))
// Main loop
3  while Q is not empty do
4      $u \leftarrow$ Q.DELETEMIN()
5      $(d_0, d_1) \leftarrow$ L($u$).MINKEY()
6      $l \leftarrow$ L($u$).DELETEMIN()
7      if L($u$) is not empty then
8          $k_{dist} \leftarrow$ L($u$).MINKEY()
9          Q.INSERT($u$, $k_{dist}$)
10     forall $(u, v) \in E$ do
11         if $d_0 + \omega(u, v) < r_d$ then
12             $D \leftarrow \{(d_0 + \omega(u, v), d_1 + \omega(u, v))\}$
13             if $v \in P$ then
14                 $D$.INSERT($(d_0 + \omega(u, v) + r_p, 0)$)
15             forall $x \in D$ do
16                 if $x$ is not dominated by any label in L($v$) then
17                     L($v$).REMOVEDOMINATED($x$)
18                     L($v$).INSERT($(l, (u, v)), x$)
19                     if Q.CONTAINS($v$) then
20                         Q.DECREASEKEY($v, x$)
21                     else
22                         Q.INSERT($v, x$)
```

5. Evaluation

1. theoretical complexity?
2. experiments, setup, methodology, results

6. Conclusion

Summary and outlook.

Bibliography

Appendix

A. Appendix Section 1

ein Bild

Figure A.1.: A figure