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 16, 2022

Abstract

A short summary of what is going on here.

Deutsche Zusammenfassung

Kurze Inhaltsangabe auf deutsch.

Contents

1. Introduction

This chapter should contain

1. A short description of the thesis topic and its background.
2. An overview of related work in this field.
3. Contributions of the thesis.
4. Outline of the thesis.

2. Preliminaries

This chapter should provide the foundations of the thesis.

3. Algorithm

The content chapters of your thesis should of course be renamed. How many chapters you need to write depends on your thesis and cannot be said in general.

3.1. Problem Description

The shortest path problem with driving time constraints consists of a Graph $G = (V, E)$, a set $P \subseteq V$ of parking nodes, a set R of driving time constraints r_i and start and target nodes s and t. 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 $r_i \leq r_j \implies r_{i,d} \leq r_{j,d} \wedge r_{i,p} \leq r_{j,p} \forall i, j$.

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 $v \in P$.

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.

3.1.1. General Approach

We introduce a labeling algorithm which solves the shortest path problem with driving time constraints. todo

Complexity

d

3.2. A* Variant

3.2.1. Dijkstra's Algorithm with One Driving Time Constraint

A driving time constraint is a rule which defines a maximum allowed non-stop driving time t_d and a pause time t_p. Before the driving time limit t_d is exceeded, the driver must park at designated parking for a minimum time period of t_p before continuing.

The base algorithm with one driving time constraint extends a Dijkstra search with pruning rules to comply with the constraint. It uses distance labels which it propagates between

Algorithm 3.1: CSP

```
Input: Graph $G = (V, E, \omega)$, parking nodes $P \subseteq V$, driving time restriction $r$,
       source node $s \in V$
Data: Priority queue Q, per node priority queue $\mathsf{L}(v)$ of labels for all $v \in V$
Output: Distances $\mathsf{d}(v)$ for all $v \in V$, shortest-path tree of $s$ given by $\mathsf{pred}(\cdot)$

// Initialization
Q.insert($s, (0, 0)$)
L($s$).insert($(\perp, \perp), (0, 0)$)
// Main loop
while Q is not empty do
    $u \leftarrow$ Q.deleteMin()
    $(d_0, d_1) \leftarrow$ L($u$).minKey()
    $l \leftarrow$ L($u$).deleteMin()
    if L($u$) is not empty then
        $k_{dist} \leftarrow$ L($u$).minKey()
        Q.insert($u$, $k_{dist}$)
    forall $(u, v) \in E$ do
        if $d_0 + \omega(u, v) < r_d$ then
            $D \leftarrow \{(d_0 + \omega(u, v), d_1 + \omega(u, v))\}$
            if $v \in P$ then
                $D$.insert($(d_0 + \omega(u, v) + r_p, 0)$)
            forall $x \in D$ do
                if $x$ is not dominated by any label in L($v$) then
                    L($v$).removeDominated($x$)
                    L($v$).insert($(l, (u, v)), x$)
                    if Q.contains($v$) then
                        Q.decreaseKey($v, x$)
                    else
                        Q.insert($v, x$)
```

the nodes. The search operates on a graph $G = (V, E, \omega)$ with the available parking nodes defined as a subset $P \subseteq V$. The search can decide to *park* at a node v if $v \in P$.

Each node v holds a set $L(v)$ of *labels*. Each label at a node v holds two distances d_0 and d_1 and a link to the previous label. The chain of linked labels represents a unique *path* from s to v. A path is characterized by the sequence of visited nodes v_i and a subset of all $v_i \in P$ to describe the parking nodes on the path which were used for parking. The distance d_0 describes the distance on the path from the start node s and d_1 since the last pause, i.e., the distance from the last node $v_i \in P$ which was used for parking.

3.2.2. Potential for Driving Time Constraints

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:

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

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

$$len(u,v) - pot(u) + pot(v) \geq 0 \quad \forall (u,v) \in E \tag{3.1}$$

Following example of a query using the graph in Fig. 3.1 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 $len(s,v) = 2$, $len(s,v) - \pi_t(s) + \pi_t(v) = -1$.

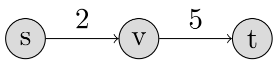

Figure 3.1.: A graph with the potential to break the potential.

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-tpath*. 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.

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

Definition 3.1. *A label potential $pot(l,v)$ with $l \in L(v)$ is feasible if for all $(u,v) \in E$, $l \in L(u)$, and $m \in L(v)$ where l is a direct predecessor of m:*

$$len(u,v) - \pi_t(l,u) + \pi_t(m,v) \geq 0 \tag{3.2}$$

Theorem 3.2. *The potential π_t is a feasible label potential.*

Proof. Given a Graph $G = (V,E)$ with a set of parking nodes $P \subseteq V$, any edge $(u,v) \in E$, label $l \in L(u)$, $m \in L(v)$ with l is a direct predecessor of m. Let $s,t \in V$ be starting node and target node of a query in G and $p,q \in P \cup \{s\}$.

$$
\begin{aligned}
&len(u,v) - \pi_t(l,u) + \pi_t(m,v) \\
&\qquad = len(u,v) - \left(\left\lfloor \frac{d_1(l) + d_{direct}(u,t)}{t_d} \right\rfloor * t_p + d_{direct}(u,t)\right) \\
&\qquad\quad + \left\lfloor \frac{d_1(m) + d_{direct}(v,t)}{t_d} \right\rfloor * t_p + d_{direct}(v,t) \\
&\qquad = \left\lfloor \frac{d_1(m) + d_{direct}(v,t)}{t_d} \right\rfloor * t_p - \left\lfloor \frac{d_1(l) + d_{direct}(u,t)}{t_d} \right\rfloor * t_p \\
&\qquad\quad + len(u,v) + d_{direct}(v,t) - d_{direct}(u,t) \\
&\qquad = \left\lfloor \frac{d_1(m) + d_{direct}(v,t)}{t_d} \right\rfloor * t_p - \left\lfloor \frac{d_1(l) + d_{direct}(u,t)}{t_d} \right\rfloor * t_p \\
&\qquad = \underbrace{\left\lfloor \frac{d_{direct}(q,v) + d_{direct}(v,t)}{t_d} \right\rfloor * t_p}_{\text{min. pause time on q-v-t path}} \\
&\qquad\quad - \underbrace{\left\lfloor \frac{d_{direct}(p,u) + d_{direct}(u,t)}{t_d} \right\rfloor * t_p}_{\text{min. pause time on p-u-t path}}
\end{aligned}
\tag{3.3}
$$

The nodes p, q represent the last used parking node of the labels l, m or the starting node if no parking node was used. The condition for feasibility of a label potential as in 3.1 simplifies to

$$\textit{number of breaks on q-v-t path} - \textit{number of breaks on p-u-t path} \geq 0$$

Case 1: $u \notin P \wedge v \notin P$. Since both u and v are no parking nodes and l is a direct predecessor of m, both label used the same parking node for their last pause or did not pause, therefore $q = p$ and

$$
\begin{aligned}
d_{direct}(p,u) + d_{direct}(u,t) &= d_{direct}(p,u) + len(u,v) + d_{direct}(v,t) \\
&= d_{direct}(q,u) + len(u,v) + d_{direct}(v,t) \\
&= d_{direct}(q,v) + d_{direct}(v,t)
\end{aligned}
\tag{3.4}
$$

With equations 3.3 and 3.4 follows $len(u,v) - \pi_t(l,u) + \pi_t(m,v) = 0$ and π_t is a feasible label potential.

Case 2: $u \in P \wedge v \notin P$. In this case, $u = q = p$ and π_t is a feasible label potential as shown in case 1.

Case 3: $u \notin P \wedge v \in P$. In this case, $q \neq p$ and $d_{direct}(q,v) = 0$.

$$
\begin{aligned}
&\left\lfloor \frac{d_{direct}(q,v) + d_{direct}(v,t)}{t_d} \right\rfloor * t_p - \left\lfloor \frac{d_{direct}(p,u) + d_{direct}(u,t)}{t_d} \right\rfloor * t_p \\
&\qquad = \left\lfloor \frac{d_{direct}(v,t)}{t_d} \right\rfloor * t_p - \left\lfloor \frac{d_{direct}(p,u) + d_{direct}(u,t)}{t_d} \right\rfloor * t_p
\end{aligned}
\tag{3.5}
$$

□

Algorithm 3.2: CSPA*

```
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
Q.INSERT($s$, $(0, 0)$)
L($s$).INSERT($((\perp, \perp), $ pot(*(0,0)*)$)$)

// Main loop
while Q is not empty do
    $u \leftarrow$ Q.DELETEMIN()
    $(d_0, d_1) \leftarrow$ L($u$).MINKEY()
    $l \leftarrow$ L($u$).DELETEMIN()
    if L($u$) is not empty then
        $k_{dist} \leftarrow$ L($u$).MINKEY()
        Q.INSERT($u$, $k_{dist}$)
    forall $(u, v) \in E$ do
        if $d_0 + \omega(u, v) < r_d$ then
            $D \leftarrow \{(d_0 + \omega(u, v), d_1 + \omega(u, v))\}$
            if $v \in P$ then
                $D$.INSERT($(d_0 + \omega(u, v) + r_p, 0)$)
            forall $x \in D$ do
                if $x$ is not dominated by any label in L($v$) then
                    L($v$).REMOVEDOMINATED($x$)
                    L($v$).INSERT($(l, (u, v)), x$)
                    if Q.CONTAINS($v$) then
                        Q.DECREASEKEY($v, x$)
                    else
                        Q.INSERT($v, x$)
```

3.2.3. A* with Driving Time Constraints

3.2.4. Multiple Driving Time Constraints

3.3. Core Contraction Hierarchy Variant

Algorithm 3.3: 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 $\mathsf{L}(v)$ 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($v, x$)
```

3.3.1. Building the Contraction Hierarchy

3.4. Combining A* and Core Contraction Hierarchy

4. Evaluation

heyyya

5. Conclusion

Summary and outlook.

Bibliography

Appendix

A. Appendix Section 1

ein Bild

Figure A.1.: A figure