Karlsruhe Institute of Technology

# 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, July 11, 2022

**Abstract**

A short summary of what is going on here.

**Deutsche Zusammenfassung**

Kurze Inhaltsangabe auf deutsch.

# Contents

# List of Figures

# List of Tables

# List of Algorithms

# 1. Introduction

For many professional truck drivers, fatigue is a daily companion while en-route. Early surveys [WFFS01, AG03] interviewing drivers in different countries all over the world conclude that fatigue is experienced by drivers daily and within hours of the start of the drive. Supporting drivers in scheduling breaks as required can reduce fatigue. This is especially important since fatigue is a common cause for accidents and near misses. The investigation of [EA05] observed severe truck accidents in Germany for a period of three months. They found that in 16% of the 127 registered accidents that were caused by the truck driver, fatigue was mentioned as the cause of the accident. A study of causes of truck crashes in the US [Fed06] examines 967 crashes during the years 2001 to 2003 and found that 13% of the truck drivers stated that they experienced fatigue during the time of the crash. In [CWNJ01], the authors find a strong relationship between acute fatigue and crash involvement by interviewing crash-involved and non-crash-involved drivers in the same areas and during the same times. In a similar case-control-study, [CKMR01] find a fourteen-fold increased crash risk for drivers who have experienced fatigue to a level where they almost have fallen asleep in the driver's seat. Today, fatigue is still a common cause for general accidents on the road, not only for accidents which involve trucks [Sta21], a finding that is also supported by a large meta-study [MNR19] in which the authors also find that strategies to reduce driver fatigue can effectively reduce the risk of traffic accidents.

Regulators reacted early to reduce the risk of accidents caused by fatigued truck drivers and introduced regulations, often using the term *drivers' working hours* or *hours of service* (HOS) in the United States. In 1985, the Council of the European Communities (since 1993 the Council of the European Union) passed council regulation (EEC) No 3820/85 [Cou85] which, with some exceptions, introduced a mandatory break of 45 min after 4.5 h of driving and a daily rest period of 11 h for professional drivers "involved in the carriage of goods". In addition, drivers must extend the daily rest period to 45 consecutive hours once every week or, if exceptions apply, must compensate for a reduced weekly rest period within three weeks. Directive 2002/15/EC [Eur02] of the European Parliament and of the Council governs working times and introduced a maximum average weekly working time of 48 h over four months and a maximum weekly working time of 60 h. Since driving time is working time, the directive also applies for truck drivers. The regulation that determines the driving time limits, break, and rest times of today was introduced in 2006 as regulation EC 561/2006 of the European Union [Eur06]. It contains well-known principles such as the break after 4.5 h and the daily and weekly rest periods. Since this regulation is of greater importance of this work, we present a brief study of it in Section 2.1.

In the US, the first hours of services regulations for long-haul truck drivers were adopted in the late 1930s - a time in which trucks achieved average speeds on routes of about 40 km/h [Fed00]. The regulation limited working hours to 12 h within a period of 15 h and introduced a weekly maximum of 60 h. The US HOS regulations for truck drivers of today allow a maximum of 8 h of driving until the driver must take a break for at least 30 min and a maximum 11 h of on-duty work until the driver must rest for 10 h off duty [Fed11]. As for the current EU regulation, we will present a more detailed study of the US rules in Section 2.2.

Drivers' working hours regulations increase the road safety and the working conditions of truck drivers. At the same time, they impose a burden on the dispatcher and the truck driver who are responsible for following the regulations and taking the breaks on schedule. Additionally, the regulations impose the challenge for the driver of finding appropriate places for parking along the route which cause as little as possible detours from the original route. To reduce the necessary manual routing effort from the truck driver, the search for parking locations for mandatory breaks ideally is incorporated into the navigation system which is used to navigate towards the destination. This necessitates routing algorithms which are capable of determining the need for breaks on a route and finding parking locations which lead to the smallest possible detours while complying with given drivers' working hours regulations.

Existing work manages to find the routes to a destination which minimize the sum of driving time and break time if the number of breaks on the route is limited to one break. The EU regulations allow routes up to a driving time of 9 h with one break of 45 min at the half-way point. Therefore, limiting the number of possible breaks on a route to one break is suitable for finding shorter routes with a travel time of not longer than a day, whereas long-haul truck drivers who drive multi-day routes are left out with this approach. Additionally, long-haul truck drivers must consider a more complex set of regulations that mandates daily and weekly rest times, many of which are not relevant for planning short routes. We name the problem of finding shortest routes to far-off destinations which require multiple days of driving and extensive rest periods while following a given set of drivers' working hours regulations the Long-Haul Truck Driver Routing Problem (LH-TDRP). An approach which limits the number of breaks on the route leading to a maximum travel time of less than a day does not meet requirements of long-haul truck driver routing. To the best of our knowledge, there exists no approach which solves the LH-TDRP and achieves practicable performance.

In this thesis, we will begin with an overview over related literature on truck driver routing or scheduling in Chapter 2 where we also discuss the drivers' working hours regulations of the EU and the hours of service regulations of the US. Chapter 3 introduces our notations and the foundations on which we base our work. In Chapter 4, we convert the LH-TDRP into a formal definition using abstractions from the EU and US regulations to obtain the Truck Driver Routing Problem (TDRP) and its variants. In Chapter 5, we present a baseline algorithm and use the foundations of Chapter 3 to develop extensions and optimizations that lead to practicable running times. Finally, we conduct a series of experiments regarding the running time of our algorithms on a German and European road network in Chapter 6. Additionally, we investigate how different parameters and road networks influence the performance of our algorithms. We conclude this work in Chapter 7 with a short summary and suggestions on possible future research.

# 2. Related Work

The planning of breaks on a route in a routing or scheduling context has been studied beforehand. The introduction of regulation EC 561/2006 of the European Union [Eur06] which regulates driving and rest times has led to the work of [Goe09] who introduce a model for the regulation in the context of the Truck Driver Scheduling Problem (TDSP) which they solve using a label propagation algorithm. In the TDSP, a schedule must be found to visit multiple customers with given distances between them. Each customer must be visited in a given time window. The authors revisit the TDSP in later publications, addressing regulations of different countries, such as the US [GK12]. The authors of [SSVB21] revisit the TDSP and name the respective variants EU-TDSP for the EU and US-TDSP for the US. All variants have in common that breaks must be taken on a route in regular intervals. In [Goe12], the authors introduce the Minimum Truck Driver Scheduling Problem (MD-TDSP) which aims to find a valid truck driver schedule with a minimal total duration while breaks can only be taken at customers and specified rest areas. The problem is solved using mixed integer linear programming. The Truck Driver Scheduling and Routing Problem (TDSRP) presents an extension of the TDSP and has been studied with slightly varying definitions. The work of [Sha08] solves a time-dependent version of the TDSRP heuristically and allows breaks on every node, there are no dedicated parking nodes. The Bachelor's thesis of [Brä16] aims to solve the problem with an exact algorithm but fails to present an algorithm with practicable running time. A heuristic version of the approach achieves running times under a minute. The dissertation of [Kle19], which also addresses the TDSP, extends this work and presents an exact algorithm and a heuristic for the TDSRP. They allow parking only at dedicated nodes in the network for which they introduce the term *no-break-en-route policy*. The number of breaks between customers is limited to one break. The master's thesis of [Bom20] targets the TDSRP with no-break-en-route-policy in its non-time-dependent version. They restrict the number of types of driving time constraints to two (TDSRP-2B) and provide an exact solution using mixed integer linear programming and, additionally, a heuristic approach. The exact solution fails to provide practical running times and the heuristic approach still suffers from long-running times in realistic scenarios. They also briefly present an extension to solve a time-dependent variant of the problem (TD-TDSRP-2B). The TDSRP-2B in contrast to the TDSP is already related to the long-haul truck driver routing problem (LH-TDRP) of our work. The LH-TDRP closely resembles the problem of routing between customers in the TDSRP-2B with regard to parking areas (no-breaks-en-route-policy) and driving time constraints. The work of [MDGCNR20] addresses a variant of the truck driver scheduling problem with intermediate stops (VRTDSPIS). The VRTDSPIS resembles a combined

routing and scheduling problem similar to the TDSRP, the intermediate stops in this work consist of taking a break due to regulatory constraints or refueling. The sequence of customers is not predetermined, only start and destination are given. The authors differentiate between the four types of breaks rest, meal, overnight and weekly break and a break of a certain type must fit the infrastructure of the stop location. The problem is formulated as a mixed integer linear program and using a state-space graph model and solved using a Dijkstra-based algorithm. The authors' goal is to use the approach to enable analysis of the effects of changes in HOS regulations for different stakeholders, e.g., the analysis of the economic impact for policy-makers. They conduct a brief case study in which they use their approach to analyze significant regulatory changes in Brazil.

The LH-TDRP itself or variations of it were also addressed in previous work, although using different terminology. The work of [KBS+17] restricts the LH-TDRP by only allowing one break on a route, but includes traffic predictions using time-dependent road networks. It uses a version of core contraction hierarchies where all the parking nodes are part of the core, which we will revisit later in this work. The authors of [vdTdWB18] also allow only one break on a route and combine their approach with temporary driving bans and road closures. First, they introduce a multi-label version of Dijkstra's algorithm. Second, they present a heuristic version using time-dependent CH with an improved runtime in comparison.

Other publications address different complex real-world scenarios which present an extension to the shortest path problem (SPP). As in this work, most of them strive to integrate the additional requirements into contraction hierarchies (CH) or other speed-up techniques which exploit the hierarchy of road networks [BDG+15]. In [KSWZ20], the authors consider a different scenario in which parking on a route becomes important. The work also addresses temporary driving bans and road closures. In this scenario, it may occur that a driver has to wait for a driving ban or a road closure to be lifted. This waiting time shall be spent in a designated parking area and not en-route. The necessary number of breaks on a route arises from the road network and query, therefore the authors do not restrict the number of possible breaks on a route as done in other publications. The work also includes a rating of parking areas. It therefore searches for pareto-optimal solutions with regard to travel time and ratings of the used parking areas. The approach is too slow to be used in practice, but it uses a combination of the A* algorithm and CH-Potentials which we also will revisit later. Integrating traffic predictions, respectively time-dependent road networks with CH is addressed by [BDSV09, BGSV13]. Turn costs are addressed by [GV11]. Mitigating the complex task of integrating additional information into a CH is addressed by [SZ21] by building a tight potential based on CH which then is used in combination with the A* algorithm. They use the same principle to address live traffic and extend the CH variant to a customizable contraction hierarchy [DSW16] to address temporary driving bans.

## 2.1. Drivers' Working Hours Regulations of the European Union

The following rules originate from the aforementioned regulation EC 561/2006 of the European Union [Eur06] and govern drivers of HGV and passenger vehicles with more than nine seats. According to the regulation, a driver can spend a time period with a break, rest, driving or "other work". A break is a period of time in which the driver is not driving. A rest is a period of time in which the driver is free to spend his time as he wishes. A driver spends time with other work if he is not driving but conducting other work-related activities.

A driver must take a break of at least 45 min after 4.5 h of driving. The break must not necessarily be taken en-bloc, it can be split into a first break of at least 15 min and a second

break of at least 30 h. The second break must take place before the driver exceeds the 4.5 h of driving time.

The regulation mandates daily and weekly rest periods. A daily rest period lasts at least 11 h. Similar to a break, the daily rest period can be split into a first period of 3 h and a second period of 9 hour. It is allowed to reduce this daily rest period to 9 h, but this must occur at most three times per week. A weekly rest period must consist of at least 45 h of rest. It can be reduced to a time of at least 24 hours if there is never more than one reduced weekly rest period in any two consecutive weeks. After having completed a weekly rest period, the next weekly rest period must begin within six cycles consisting of driving and a daily rest period.

Additionally, driving is restricted to a maximum of 9 h per day. This can be extended to 10 h twice a week. The maximum allowed driving time within a week is 56 h and the driving time in two consecutive weeks must not exceed 90 h. The regulation regarding drivers's working hours does not apply in some exceptions, e.g., if the vehicle's maximum authorized speed is at most 40 km/h or if the truck carries equipment and machinery, does not leave a 100 km radius from the base, and driving is not the main activity of the person who is driving.

Truck drivers are also affected by the EU's working time regulations of directive 2002/15/EC [Eur02]. Working time includes driving and other work. It must not exceed an average of 48 h per week which is calculated using a rolling window which spans 17 weeks. The maximum allowed working time in a single week is 60 h. Breaks must be taken at least every 6 h for a time period of 15 min. If the total working time exceeds 6 h, this time period increases to 30 min and if it exceeds 9 h it increases to 45 min. In any case, the drivers' working hours rules of EC 561/2006 take precedence over the working time rules if the truck driver is carrying out driving and the working rules would allow a longer time until the next break. The regulations regarding daily and weekly rest periods are the same for drivers' working hours and working time.

## 2.2. Hours of Service Regulations of the United States

The US HOS [Fed11] govern both "property-carrying drivers" (truck drivers) and "passenger-carrying drivers" for whom the mandatory break times, off-duty times, and maximum allowed on-duty times differ. In the following, we only describe the HOS regulations for "property-carrying drivers". The US HOS use different terminology than the EU regulations regarding the classification of time periods. Time periods are classified as driving, "on-duty not driving", "off-duty", or sleeper berth. In comparison to the EU regulations, driving has the same meaning, "on-duty not driving" corresponds to "other work", and "off-duty" means resting. The sleeper berth is only addressed by the US HOS.

A driver must take a break of at least 30 min after having accumulated 8 h of driving time without a break in between. The conditions for a break are satisfied if the driver carries out any action except driving for at least 30 min without interruption. Further, the US HOS differentiate a 11 h *driving* limit and a 14 h limit. The former limits the allowed driving time to at most 11 h if the driver was off-duty for at least 10 h without interruption. The latter mandates that at most 14 h may pass after the driver has come on-duty, until the driver must spend an uninterrupted time of at least 10 h off-duty. Essentially, a driver's work day can last at most 14 h of which at least 3 h must be spent with other work than driving and the work day must be followed by 10 h of rest. If a driver encounters adverse driving conditions, both limits are extended by at most 2 h. Adverse driving conditions include bad weather such as fog or snow or unusual traffic. The exception does not apply if the conditions where known at the beginning of the time on-duty after the driver has spent a period of time off-duty which qualifies for on of the limits above.

The US also limit weekly on-duty hours. When driving on every day of the week, the total time period spent on-duty is limited to 70 h over 8 days. Otherwise, it is limited to 60 h over 7 days. In both cases, the on-duty must be followed by at least 34 h off-duty.

A specialty of the US HOS in comparison to the EU regulations is the sleeper berth provision. The driver is allowed to split the 10 h off-duty time which is mandated by the 11 h and 14 h limits into two parts. One part must be at least 2 h long, the other part must last for at least 7 h and must be spent in the truck's sleeper berth. Additionally, the length of both parts must add up to at least 10 h.

# 3. Preliminaries

In this chapter, we introduce our basic notations and discuss important algorithmic concepts on which the work of this thesis is based.

We define a weighted, directed graph $G$ as a tuple $G = (V, E, \text{len})$. $V$ is the set of nodes and $E$ the set of edges $(u, v) \subseteq V \times V$ between those nodes. The function len is the weight function $\text{len}\colon E \to \mathbb{R}_{\geq 0}$ which assigns each edge a non-negative weight which we often also call length of an edge. A path $p$ in $G$ is defined as a sequence of nodes $p = \langle v_0, v_1, ..., v_k \rangle$ with $(v_i, v_{i+1}) \in E$. For simplicity, we will reuse the same function len which we use to denote the length of an edge, to denote the length of a path $p$ in $G$. The length of a path $\text{len}(p)$ is defined by the sum of the weights of the edges on the path $\text{len}(p) = \sum_{i=0}^{k-1} \text{len}((v_i, v_{i+1}))$. A path must not necessarily be simple, i.e. nodes can appear multiple times in the same path.

Given two nodes $s$ and $t$ in a graph, we denote the shortest distance between them as $\mu(s, t)$. The shortest distance between two nodes is the minimum length of a path between them. The problem of finding the shortest distance and an associated path between two nodes in a graph is called the shortest path problem which we often abbreviate as SPP. The problem of finding the shortest path between nodes in a road network can be formalized as solving the SPP on a weighted, directed graph. Each edge of the graph represents a road and each node represents an intersection. Unless stated otherwise, the length len of an edge $(u, v)$ will always correspond to the time it takes to travel from $u$ to $v$ on the road which the edge represents. A solution of the SPP then yields the shortest time between two intersections in the road network and the associated path between them.

Dijkstra's algorithm, published in 1959, solves the SPP [Dij59]. It operates on the graph $G$ without any additionally information or precomputed data structures. It maintains a queue $Q$ of nodes with ascending tentative distance from the starting node $s$ and two arrays, a distance value $d[v]$ and a predecessor node $pred[v]$ for each node. At the beginning, $Q$ only contains the start node $s$ with the distance zero. The two arrays are initialized with $d[v] = \infty$ and $pred[v] = \perp$ except for $d[s] = 0$ and $pred[s] = s$. Iteratively, the node $u$ with the minimum distance is removed from $Q$ and each outgoing edge $(u, v) \in E$ of $u$ is *relaxed*. We call this process *settling* a node $u$. Relaxing an edge $(u, v)$ consists of three steps: First, the sum $d[u] + \text{len}((u, v))$ is calculated. Second, it is tested if the distance $d[v]$ can be improved by choosing $u$ as a predecessor. Finally, if that is the case, the queue key of the node $v$ is decreased. If $v$ is not contained in $Q$ yet, the node is inserted into $Q$. The search can be stopped if the target node $t$ was removed from the queue [Dij59].

For many practical applications and for large graphs, Dijkstra's algorithm is too slow. A common extension is the A* algorithm [HNR68]. A* uses a *heuristic* which yields a lower bound for the distance from each node to the target node to direct the search towards the goal. With a tight heuristic, A* can significantly reduce the search space, i.e., the amount of nodes it touches during the search in comparison to Dijkstra's algorithm. In the route planning context, the term *potential* is often used for the heuristic. We will denote the potential of a node $v$ to a node $t$ with $\pi_t(v)$.

To further reduce the search space, it is possible to run a *bidirectional* search. A bidirectional search to solve the SPP from $s$ to $t$ on a graph $G$ consists of a forward search and a backward search. The forward search operates on $G$ with start node $s$ and target node $t$ and maintains a forward queue $\overrightarrow{Q}$, a forward distance array $\overrightarrow{d}[v]$, and a forward predecessor array $\overrightarrow{pred}[v]$. The backward search operates on a backward graph $\overleftarrow{G}$ with start node $t$ and target node $s$ and maintains a backward queue $\overleftarrow{Q}$, a backward distance array $\overleftarrow{d}[v]$, and a backward predecessor array $\overleftarrow{pred}[v]$. The backward graph is defined as the graph $G$ with inverted edges, i.e, $\overleftarrow{G} = (V, \overleftarrow{E}, \overleftarrow{\text{len}})$ with $\overleftarrow{E} = \{(v,u) \in V \times V \mid (u,v) \in E\}$ and $\overleftarrow{\text{len}}(u,v) = \text{len}(v,u)$. Additionally, an array $d[v]$ is maintained for combined tentative distances of forward and backward search and is initialized with $\infty$ for all nodes. A value $d[v]$ constitutes the shortest known distance for an $s$-$t$ path using $v$.

Forward and backward search now alternately settle a node $v$. If $v$ was settled by both searches, forward and backward search met at $v$. The value $d[v]$ is updated to the sum of $\overrightarrow{d}[v] + \overleftarrow{d}[v]$ if it is an improvement over the old value, i.e., it yields a shorter distance for an $s$-$t$ path via $v$.

The bidirectional search only yields an advantage over a unidirectional search if the two searches are stopped earlier than in a unidirectional search. If not, the bidirectional search would simply execute the work of an $s$-$t$ search twice. Therefore, stronger stopping criteria are introduced. When introducing a strong stopping criterion for a bidirectional A* search, the stopping criterion also depends on the potential being used. The work of [GH05] introduces a potential and stopping criterion which leads to an improvement over a unidirectional A* search. If a forward potential $\overrightarrow{\pi}_t$ and a backward potential $\overleftarrow{\pi}_s$ is used for the forward and the backward search, the search can be stopped when the minimum key of the forward queue $\text{minKey}(\overrightarrow{Q})$ or the minimum key of the backward queue $\text{minKey}(\overrightarrow{Q})$ is greater than the currently known shortest distance between $s$ and $t$. If $\overrightarrow{\pi}_t + \overleftarrow{\pi}_s \equiv const.$ holds for the potentials, then the search can be stopped when $\text{minKey}(\overrightarrow{Q}) + \text{minKey}(\overleftarrow{Q})$ exceeds the shortest know distance between $s$ and $t$.

## 3.1. Contraction Hierarchies

Road networks have strong hierarchies since some roads are more important than others. For example, a highway allows high average speeds and therefore is a preferred connection between points in a road network in comparison to smaller roads. Contraction Hierarchies [GSSV12] are a speed-up technique which exploits these hierarchies.

Contraction Hierarchies (CH) use a two phase approach. In the preprocessing phase, the CH is constructed given a graph $G = (V, E, \text{len})$ and a node order which sorts the nodes by importance. For example, an important node of a road network might be a node in a highway interchange, an unimportant node might be the head of a dead end. We denote the CH as $G^+ = (V^+, E^+, \text{len}^+)$. The node order can be computed by searching for unimportant nodes [GSSV12] or for important nodes first [ADGW12, DGPW14]. The nodes then are sorted into multiple levels of increasing importance. Two nodes of the same level must not have an edge between them. We obtain the CH $G^+$ by iteratively

contracting the least important node according to the node order by adding shortcut edges between its neighbors.

An outgoing edge of a node to another node of a higher level is called an *upward* edge, the opposite is called a *downward* edge. A path which consists of only upward edges is called an *up-path*, a path which consists of only downward edges is called a *down-path*. Finally, a path which consists of an up-path, followed by a down-path, is called an *up-down-path*. The node on an up-down-path with the highest level, i.e. the node which separates up-path and down-path, is called the *middle node* $m$ of the path.

For every shortest path between node $s, t \in V$, there exists an up-down $s$-$m$-$t$ path in $G^+$ of the exact same length [GSSV12]. Therefore, we can restrict the search to finding this exact path. We run a bidirectional search from $s$ and $t$. The forward search from $s$ may only use upward edges and finds the subpath $s$-$m$, the backward search from $t$ may only use the inverted downward edges and finds the subpath $m$-$t$. The search is stopped if the minimum queue key of both queues of forward and backward search is greater or equal to the tentative minimum distance $\mu(s, t)$ [GSSV12].

## 3.2. Core Contraction Hierarchies

The core contraction hierarchy presents a compromise between constructing a full CH $G^+$ and running a bidirectional search on $G$. The core CH $G^* = (V^*, E^*, \text{len}^*)$ is obtained by stopping the iterative contraction of nodes of increasing importance during the construction of the CH early. This leads to a set $C \subseteq V^*$ of so-called core nodes which are uncontracted. The set of nodes $V^* = V$ therefore is separated into a set of core nodes $C$ and a set of contracted nodes $V^+ = V^* \setminus C$. The graph $G^+ = (V^+, E^+, \text{len}^*)$ is a valid contraction hierarchy containing only contracted nodes and (shortcut) edges between those nodes. The graph $G_C = (V^* \setminus V^+, E^* \setminus E^+, \text{len}^*)$ is called the core graph.

The core CH query again is a bidirectional search from $s$ and $t$. The query represents a normal CH query while settling nodes $v \in V^+$, i.e., it consists of a forward search from $s$ using only upward edges and a backward search from $t$ using only inverted downward edges. If a search reaches an uncontracted core node, it considers all outgoing edges. Thus, the core CH query can be characterized as a CH query in $G^+$ which transitions into a full bidirectional search if it reaches the core $G_C$. We can use the same stopping criterion as for the CH query since the stopping criterion for a CH is more conservative than the stopping criterion for a pure bidirectional search.

## 3.3. CH-Potentials

CH-Potentials [SZ21] are an extension of CH to efficiently calculate distances $\mu(v, t)$ for many nodes $v \in V$ to a fixed node $t$. CH-Potentials are based on PHAST [DGNW11] which itself is an extension of CH to efficiently run all-to-one queries.

PHAST runs in two steps. The first step is a backward one-to-all search from $t$ using only inverted downward edges. This is the same as running the backward search of a CH query from $t$ without any stopping criterion. We obtain an array $B$ where $B[v]$ is the length of the shortest down path from $v$ to $t$ or $B[v] = \infty$ if there is no such path. We then iteratively calculate the distance $\mu(v, t)$ of nodes of the same level, starting at the highest level. This is possible since nodes of the same level must not have edges between them. To obtain the distances of nodes $u$ of the next lower level, we find the minimum $\min(B[u], \min_v(\text{len}(u, v) + \mu(v, t)))$ for all upward edges $(u, v)$ of the node $v$. The distances $\mu(v, t)$ were already computed in the previous iteration.

We can use PHAST as to compute potentials if we run its two steps beforehand as a preprocessing step and then look up the respective distances. This preprocessing would be slow since it computes the distances to all nodes. CH-Potentials mitigate this problem by computing the results of the second PHAST step lazily while the first step remains the same. We do not calculate the distances of all nodes beforehand. To compute the potential of a node $u$, we recursively compute the potential for all nodes $v$ which are connected to $u$ by upward edges $(u, v) \in E^+$. We then can compute the distance to $t$ as in the PHAST algorithm by finding $\min(B[u], \min_v(\text{len}(u, v) + \mu(v, t)))$. We save all the calculated distances $\mu(v, t)$ in order to not compute any distance value twice.

CH-Potentials can be optimized further as [SZ21] describes in detail.

# 4. Problem Definition

In our introduction in Chapter 1, we introduced the Long-Haul Truck Driver Routing Problem as a problem which arises from the practical challenge of finding optimal multi-day routes without violating regulations regarding truck driver's driving times, working hours, breaks, and rest periods. In this chapter, we provide a formal definition of this problem using abstractions from the regulations which we described in the Sections 2.1 and 2.2.

Both, the drivers' working hours regulations of the EU and the hours of service regulations of the US are characterized by repeating cycles, consisting of a limited time period in which the driver is actively driving, and a period in which the driver must rest or may only conduct other work for a minimum amount of time. We model this characteristic using a set of *driving time constraints*. A driving time constraint $c$ is a pair of two values, a maximum allowed driving time or driving time limit $c^d$ and a minimum break time $c^b$. A driver must take an uninterrupted break of length $c^b$ before exceeding an accumulated driving time since the last break of $c^d$. A set of multiple driving time constraints is denoted as the set $C$ of driving time constraints $c_i \in C$. We assume an order among the constraints $c_i$, a driving time constraint with a given index consists of a driving time limit and a break time which are each greater than or equal to a driving time constraint with a smaller index. We can now model the regulations of the EU and the US using a set $C$ of driving time constraints.

The EU's regulations are designed around the two central concepts of a break of 45 min after a maximum driving time of 4.5 h and a rest time of 11 h after a maximum driving time of 9 h. We therefore model the EU's regulations as $C_{EU} = \{c_1, c_2\}$ with $c_1^d = 4.5\,\text{h}$, $c_1^b = 0.75\,\text{h}$, $c_2^d = 9\,\text{h}$, and $c_2^b = 11\,\text{h}$.

The US regulations are centered around the concept of a break of 30 min after at most 8 h of driving and a mandatory off-duty time of 10 h after 11 h of driving. We ignore the 14 h limit of on-duty time because the driver is not allowed to drive during the additional 3 h, rendering the rule uninteresting for our routing problem. This leads to the set of driving time constraints $C_{US} = \{c_1, c_2\}$ with $c_1^d = 8\,\text{h}$, $c_1^b = 0.5\,\text{h}$, $c_2^d = 11\,\text{h}$, and $c_2^b = 10\,\text{h}$.

We now formalize our routing problem as an extension of the shortest path problem which accounts for driving time limits and mandatory breaks which we name the Truck Driver Routing Problem (TDRP). Let $G = (V, E, \text{len})$ be a graph and $s$ and $t$ nodes with $s, t \in V$. We extend the graph with a set $P \subseteq V$ of parking nodes and a set $C$ of driving time constraints $c_i$. A route $r$ from $s$ to $t$ includes the path of visited nodes $p = \langle s = v_0, v_1, \ldots, t = v_k \rangle$ and a break time function breakTime: $p \to \{0, c_1^b, \ldots, c_{|C|}^b\}$ at

each node $i$. For non-parking nodes $v_i \notin P$, the break time must be zero since breaks can only be scheduled at nodes $v \in P$. We also define the breakTime of an entire route $r$ on $p$ as $\text{breakTime}(r) = \sum_{i=0}^{k} \text{breakTime}(v_i)$.

**Definition 4.1** (Feasible Route)**.** *A feasible route with path* $p = \langle s = v_0, v_1, ..., t = v_k \rangle$ *must comply with all driving time constraints in* $C$*. A route complies with a specific driving time constraint* $c \in C$ *if there is no subpath* $p'$ *between two nodes* $u, w \in P' = \{s, t\} \cup \{v_i \in p \mid \text{breakTime}(v_i) \geq c^b\}$ *on the path which exceeds the driving time limit* $\text{len}(p') > c^d$ *and has no third node* $v_i \in P'$ *in between* $u$ *and* $w$*.*

We define the travel time of a route and the shortest route between two nodes as follows.

**Definition 4.2** (Travel Time of a Route)**.** *The travel time* $\text{travelTime}(r)$ *of a route* $r$ *is the sum of the length of its path* $\text{len}(p)$ *and the accumulated break time* $\text{breakTime}(p)$*.*

**Definition 4.3** (Shortest Route)**.** *A route between two nodes* $s$ *and* $t$ *is called a shortest route if it is feasible and there exists no other feasible route between* $s$ *and* $t$ *with a smaller travel time.*

The shortest travel time between two nodes $s$ and $t$, i.e., the travel time of the shortest route between them is denoted as $\mu_{tt}(s, t)$. The TDRP can now can be defined as follows.

Truck Driver Routing Problem

**Input:** A graph $G = (V, E, \text{len})$, a set of parking nodes $P \subseteq V$, a set of driving time constraints $C$, and start and target nodes $s, t \in V$

**Problem:** Find a shortest route $r$ from $s$ to $t$ in $G$.

We differentiate the cases in which we allow an arbitrary number of driving time constraints (TDRP-nDTC) or restrict the number of constraints to a certain number. As demonstrated above, we can model the most important characteristic of real-world driving time regulations using two constraints, i.e., $|C| = 2$ (TDRP-2DTC). We do not consider a restriction to $|C| = 1$ (TDRP-1DTC) a Long-Haul Truck Driver Routingsince it does not allow multi-day routes with a realistic parameter setting for driving time limit and break time.

# 5. Algorithm

In this chapter, we introduce a labeling algorithm for the Truck Driver Routing Problem. At first, we will restrict the problem to one driving time constraint for simplicity and drop that constraint later. We then describe extensions of the base algorithm to achieve better running times on realistic problem instances.

## 5.1. Dijkstra's Algorithm with One Driving Time Constraint

We will adapt Dijkstra's algorithm to solve the TDRP with one driving time constraint (TDRP-1DTC). 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 in a label set $L(v)$ represent a possible route, respectively a possible solution for a query from $s$ to $v$. A label $l \in L(v)$ may represent suboptimal routes to $v$, i.e., routes which are not a shortest route between $s$ and $v$. Nevertheless, we will ensure that a label set never contains labels which represent infeasible routes according to $c$. A label $l$ contains

- travelTime($l$), the total travel time from the starting node $s$
- breakDist($l$), the distance since the last break
- pred($l$), its preceding label

As in Dijkstra's algorithm, the queue is a min-Queue with ascending keys. The key of a label $l$ is its accumulated $\mu_{tt}(l)$.

### 5.1.1. Settling a Label

In contrast to Dijkstra's algorithm, 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 Figure 5.1.

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

**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 travelTime($l'$) = travelTime($l$) + len($e$), breakDist($l'$) = breakDist($l$) + len($e$), and pred($l'$) = $l$.

**Algorithm 5.1:** Settling a label $l \in L(u)$ removes the label from the queue and relaxes all the outgoing edges of $u$.

```
1 Procedure SettleNextLabel():
2     l ← Q.deleteMin()
3     forall (u, v) ∈ E do
4         RelaxEdge((u, v), l)
```

**Label Pruning.** After propagating a label, we discard the label if it violates the driving time constraint $c$, that is, if $\text{breakDist}(l) > c^d$.

**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 shorter route from $s$ to $v$ than another label $l'$ but might have shorter remaining driving time budget $c^d - \text{breakDist}(l)$. 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 $s$-$t$. It might not even yield a feasible route to $t$ at all while $l'$ reaches the target due to the greater remaining driving time budget. In one case, we can prove that a label $l \in L(v)$ cannot yield a better solution than a label $l' \in L(v)$. We say $l'$ *dominates* $l$.

**Definition 5.1** (Label Dominance for 1DTC)**.** *A label $l \in L(v)$ dominates another label $l' \in L(v)$ if* $\text{travelTime}(l') > \text{travelTime}(l)$ *and* $\text{breakDist}(l') \geq \text{breakDist}(l)$ *or* $\text{travelTime}(l') \geq \text{travelTime}(l)$ *and* $\text{breakDist}(l') > \text{breakDist}(l)$.

If a label $l \in L(v)$ is dominated by another label $l' \in L(v)$, then $l'$ represents a route from $s$ to $t$ with a shorter or equal total travel time and longer or equal remaining driving time budget until the next break. Therefore, in each solution which uses the label $l$, $l$ can trivially be replaced by the label $l'$. The solution will still comply with the driving time constraint $c$ and yield a shorter or equal total travel time, so we are allowed to simply discard dominated labels in our search.

**Definition 5.2** (Pareto-Optimal Label)**.** *A label $l \in L(v)$ is pareto-optimal if it is not dominated by any other label $l' \in L(v)$.*

A label $l$ will only be inserted into a label set $L(v)$ if it is pareto-optimal. If a label $l$ is inserted into $L(v)$, labels $l' \in L(v)$ are removed from $L(v)$ if $l$ dominates them. $L(v)$ therefore is the set of known pareto-optimal solutions at $v$. In Figure 5.2 we define the procedure RemoveDominated($l$) as an operation on a label set.

**Algorithm 5.2:** The procedure $L$.RemoveDominated($l$) removes all labels from the label set $L$ which are dominated by the label $l$.

```
1 Procedure RemoveDominated(l):
2     forall l' ∈ L do
3         if l dominates l' then
4             L.remove (l');
```

### 5.1.2. Parking at a Node

When propagating a label $l \in L(u)$ along an edge $(u, v) \in E$ and $v \in P$, we have to consider pausing at $v$. Since we do not know if pausing at $v$ or continuing without a break is the better solution, we generate both labels and add them to the label set $L(v)$ and the queue $Q$. We now can define the procedure RelaxEdge as in Figure 5.3.

**Algorithm 5.3:** Relaxing an edge $(u, v) \in E$ when settling a label $l \in L(u)$ with regard to parking nodes.

```
1  Procedure RelaxEdge((u,v), l):
2      D ← {}
3      if breakDist(l) + len(u, v) ≤ c^d then
4          D.insert((travelTime(l) + len(u, v), breakDist(l) + len(u, v), l))
5          if v ∈ P then
6              D.insert((travelTime(l) + len(u, v) + c^b, 0, l))
7          forall l' ∈ D do
8              if l' is not dominated by any label in L(v) then
9                  L(v).RemoveDominated(l')
10                 L(v).insert(l')
11                 Q.queueInsert(travelTime(l'),l')
```

### 5.1.3. Initialization and Main Loop

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 is given as Algorithm 5.4 Dijkstra+1DTC.

**Algorithm 5.4:** Dijkstra for TDRP-1DTC

**Input:** Graph $G = (V, E, \text{len})$, set of parking nodes $P \subseteq V$, set of driving time constraints $C = \{r\}$, start and target nodes $s, t \in V$
**Data:** Priority queue Q, per node set L($v$) of labels for all $v \in V$
**Output:** Shortest route with travelTime($j$) $= \mu_{tt}(s, t)$

```
// Initialization
1  Q.queueInsert(0,(0, 0, ⊥))
2  L(s).insert((0, 0, ⊥))
// Main loop
3  while Q is not empty do
4      SettleNextLabel()
5      if label at t was settled then
6          return
```

### 5.1.4. Correctness

Given a start node $s$ and a target node $t$, Dijkstra's algorithm returns the shortest distance between $s$ and $t$. The algorithm can be stopped after $t$ was removed from the queue since all the following nodes in the queue have larger distances and the edge lengths are non-negative by definition. Therefore, relaxing an outgoing edge of these nodes cannot lead to an improvement of the distance at $t$.

In our case, the algorithm shall return the shortest travel time $\mu_{tt}(s, t)$ between two nodes $s$ and $t$. We have a queue of labels which is sorted in ascending order by their travel time. When removing a label $l$ from the queue, all the other labels in the queue therefore have a larger travel time than travelTime($l$). Additionally, all the edge lengths and the break times are non-negative. Relaxing an edge thus can only increase the travel time. The same

**Algorithm 5.5:** A* FOR TDRP-1DTC

**Input:** Graph $G = (V, E, \text{len})$, set of parking nodes $P \subseteq V$, a set of driving time constraints $C = \{r\}$, start and target nodes $s, t \in V$, potential $\text{pot}_t()$
**Data:** Priority queue $\mathsf{Q}$, per node set $\mathsf{L}(v)$ of labels for all $v \in V$
**Output:** Shortest route with $\text{travelTime}(j) = \mu_{tt}(s, t)$

```
// Initialization
l_s ← (0, 0, ⊥)
Q.QUEUEINSERT(pot_t((l_s), s), l_s)
L(s).INSERT(l_s)
// Main loop
while Q is not empty do
    SETTLENEXTNODE()
    if minimum of Q is label at t then
        return
```

argument as for the correctness of Dijkstra's algorithm applies: Since relaxing an edge can only increase travel time, labels which are added later to the queue will also have a larger travel time than $\text{travelTime}(l)$. Therefore, when we remove the first label $l_t \in L(t)$ at $t$ from the queue, we know that this time cannot be improved further and it is correct to stop the search.

When relaxing an edge $(u, v)$ and propagating a label $l \in L(u)$, we check if the new label complies with the driving time constraints. We do not insert the new label into $L(v)$ and the queue if it violates a constraint. Therefore, label sets and the queue only contain labels which represent a feasible route and $l_t$ must also represent a feasible route.

If we propagated a label to a parking node, we produce a second label since we have the two choices of parking and not parking at the node. We treat both labels equally and insert both labels into label set and queue if they are not dominated or represent an infeasible route. The label with the smaller travel time will be removed first from the queue, and the second label will be handled at a later stage when its travel time becomes the smallest of all labels in the queue. It is not possible to miss possible routes between $s$ and $t$ because we produce all the labels and only discard dominated labels and labels representing infeasible routes. There therefore cannot be a label with a better travel time from $s$ to $t$ than the label $l_t$ which we removed as the first label at t from the queue. Therefore, its travel time $\text{travelTime}(l_t)$ is equal to the shortest travel time $\mu_{tt}(s, t)$ between $s$ and $t$.

## 5.2. Goal-Directed Search with One Driving Time Constraint

In this section, we extend the base algorithm described in Section 5.1 to a goal-directed search with the A* algorithm. We introduce a new potential $\text{pot}_t$ based on the CH-Potentials in Section 3.3. We then show that we still can stop the search when the first label at $t$ is removed from the queue.

The difference between Dijkstra and A* is the order in which nodes are being removed from the queue. In our case, this corresponds to the order of labels being removed from the queue. Instead of using their travel time $\text{travelTime}(l)$ as a queue key, a label $l \in L(v)$ is added to the queue with the key $\text{travelTime}(l) + \text{pot}_t(l, v)$. Algorithm 5.5 shows the adaption of the coarse algorithm.

The only thing left is the adaption of RELAXEDGE in Figure 5.3 where we change the queue keys to use $\text{travelTime}(l) + \text{pot}(l, v)$ instead. The result is shown in Figure 5.6.

**Algorithm 5.6:** Relaxing an edge with regard to the potential.

```
1  Procedure RelaxEdge((u, v), l):
2      D ← {}
3      if breakDist(l) + len(u, v) ≤ c^d then
4          D.insert((travelTime(l) + len(u, v), breakDist(l) + len(u, v), l))
5          if v ∈ P then
6              D.insert((travelTime(l) + len(u, v) + c^b, 0, l))
7          forall l' ∈ D do
8              if l' is not dominated by any label in L(v) then
9                  L(v).RemoveDominated(l')
10                 L(v).insert(l')
11                 Q.queueInsert(travelTime(l') + pot_t(l', v), l')
```

### 5.2.1. Potential for One Driving Time Constraint

In general, every feasible potential can be used for the goal-directed algorithm. We use CH-Potentials as foundation to build a potential which accounts for necessary break times on the route.

Given a target node $t$, the CH-Potentials yield a perfect estimate for the distance $\mu(v, t)$ from $v$ to $t$ without regard for driving time constraints and breaks. This is a lower bound for the remaining travel time for any label at $v$. A better lower bound for the remaining travel time of a label at $v$ to $t$, including breaks due to the driving time limit, can be calculated by taking the minimum necessary amount of breaks into account. We define minBreaks($d$) as a function of time which calculates the minimum amount of necessary breaks for any arbitrary driving time $d$.

$$\text{minBreaks}(d) = \begin{cases} \left\lceil \frac{d}{c^d} \right\rceil - 1 & d > 0 \\ 0 & else \end{cases} \tag{5.1}$$

Simply using $\left\lfloor \frac{d}{c^d} \right\rfloor$ is not sufficient since we do not need to pause for a driving time of exactly $c^d$. We now can calculate a lower bound for the minimum necessary break time given an arbitrary driving time $d$

$$\text{minBreakTime}(d) = \text{minBreaks}(d) \cdot c^b \tag{5.2}$$

and finally define our node potential as

$$\text{pot}'_t(v) = \text{minBreakTime}(\text{chPot}_t(v)) + \text{chPot}_t(v) \tag{5.3}$$

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

$$len(u, v) - \text{pot}_t(u) + \text{pot}_t(v) \geq 0 \quad \forall (u, v) \in E \tag{5.4}$$

A feasible node potential allows us to stop the A* search when the node $t$, respectively the first label at $t$, was removed from the queue. Following counterexample of a query using the graph in Fig. 5.1 shows that $\text{pot}'_t$ is not feasible. With a driving time limit of 6 and a break time of 1, the potential here will yield a value $\text{pot}_t(s) = 8$ since the potential includes the minimum required break time for a path from s to t. Consequently, with $\text{pot}'_t(v) = 5$ and $len(s, v) = 2$, $len(s, v) - \text{pot}'_t(s) + \text{pot}'_t(v) = -1$.

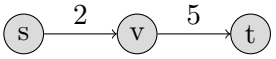



Figure 5.1.: Example graph for which the feasibility condition of 5.4 does not always hold with the potential $\text{pot}'$.

A variant of the potential accounts for the distance since the last break of a label $\text{breakDist}(l)$ to calculate the minimum required break time on the $v$-$t$ path.

$$\text{pot}_t(l, v) = \text{minBreakTime}(\text{breakDist}(l) + \text{chPot}_t(v)) + \text{chPot}_t(v) \tag{5.5}$$

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 $v$. The feasibility definition as defined in 5.4 can no longer be applied. We therefore have to show that queue keys of labels can only increase over time.

**Lemma 5.3.** *Let $l$ be a label in the label set $L(u)$ which the algorithm propagates along the edge $(u, v) \in E$ to create a label $l' \in L(v)$. The sum of travel time and potential of a label can only increase, i.e,* $\text{travelTime}(l) + \text{pot}_t(l, u) \leq \text{travelTime}(l') + \text{pot}_t(l', v)$.

*Proof.* Let $(u, v) \in E$ be an edge. The procedure RelaxEdge in Figure 5.6 can produce two new labels at a node $v$ for each label at $u$, depending on if $v$ is a parking node. We differentiate the two cases not parking at $v$ and parking at $v$. Let $l \in L(u)$ and $l' \in L(v)$.

Following general observations can be made:

1. $d \geq d' \implies \text{minBreakTime}(d) \geq \text{minBreakTime}(d')$
2. $c^b + \text{minBreakTime}(d) \geq \text{minBreakTime}(d + c^d)$
3. $c^d \geq \text{breakDist}(l') \geq \text{breakDist}(l) + \text{len}(u, v) \geq \text{breakDist}(l)$
   (Line 2 in RelaxEdge in Figure 5.6)
4. $\text{len}(u, v) - \text{chPot}_t(u) + \text{chPot}_t(v) \geq 0$ (feasibility of the CH-Potentials)
5. $\text{len}(u, v) + \text{chPot}_t(v) \geq \text{chPot}_t(u)$

We show that $\text{travelTime}(l') + \text{pot}_t(l', v) - \text{travelTime}(l) - \text{pot}_t(l, u) \geq 0$.

*Case 1: Not parking at $v$.* In this case, $\text{travelTime}(l') = \text{travelTime}(l) + \text{len}(u, v)$ and $\text{breakDist}(l') = \text{breakDist}(l) + \text{len}(u, v)$.

$$
\begin{aligned}
&\text{travelTime}(l') - \text{travelTime}(l) - \text{pot}_t(l,u) + \text{pot}_t(l',v) \\
&= \text{travelTime}(l) + \text{len}(u,v) - \text{travelTime}(l) \\
&\quad - (\text{minBreakTime}(\text{breakDist}(l) + \text{chPot}_t(u)) + \text{chPot}_t(u)) \\
&\quad + \text{minBreakTime}(\text{breakDist}(l') + \text{chPot}_t(v)) + \text{chPot}_t(v) \\
&= \text{len}(u,v) + \text{minBreakTime}(\text{breakDist}(l') + \text{chPot}_t(v)) \\
&\quad - \text{minBreakTime}(\text{breakDist}(l) + \text{chPot}_t(u)) - \text{chPot}_t(u) + \text{chPot}_t(v) \\
&= \text{len}(u,v) + \text{minBreakTime}(\text{breakDist}(l) + \text{len}(u,v) + \text{chPot}_t(v)) \qquad (5.6) \\
&\quad - \text{minBreakTime}(\text{breakDist}(l) + \text{chPot}_t(u)) - \text{chPot}_t(u) + \text{chPot}_t(v) \\
&\overset{\text{(1. and 5.)}}{\geq} \text{len}(u,v) + \text{minBreakTime}(\text{breakDist}(l) + \text{chPot}_t(u)) \\
&\quad - \text{minBreakTime}(\text{breakDist}(l) + \text{chPot}_t(u)) - \text{chPot}_t(u) + \text{chPot}_t(v) \\
&= \text{len}(u,v) - \text{chPot}_t(u) + \text{chPot}_t(v) \\
&\overset{\text{(4.)}}{\geq} 0
\end{aligned}
$$

*Case 2: Parking at $v$.* In this case, $\text{travelTime}(l') = \text{travelTime}(l) + \text{len}(u,v) + c^b$ and $\text{breakDist}(l') = 0$.

$$
\begin{aligned}
&\text{travelTime}(l') - \text{travelTime}(l) - \text{pot}_t(l,u) + \text{pot}_t(l',v) \\
&= \text{travelTime}(l) + \text{len}(u,v) + c^b - \text{travelTime}(l) \\
&\quad - (\text{minBreakTime}(\text{breakDist}(l) + \text{chPot}_t(u)) + \text{chPot}_t(u)) \\
&\quad + \text{minBreakTime}(\text{chPot}_t(v)) + \text{chPot}_t(v) \\
&= \text{len}(u,v) + c^b + \text{minBreakTime}(\text{chPot}_t(v)) \\
&\quad - \text{minBreakTime}(\text{breakDist}(l) + \text{chPot}_t(u)) - \text{chPot}_t(u) + \text{chPot}_t(v) \\
&\overset{\text{(2.)}}{\geq} \text{len}(u,v) + \text{minBreakTime}(c^d + \text{chPot}_t(v)) \\
&\quad - \text{minBreakTime}(\text{breakDist}(l) + \text{chPot}_t(u)) - \text{chPot}_t(u) + \text{chPot}_t(v) \qquad (5.7) \\
&\overset{\text{(1. and 3.)}}{\geq} \text{len}(u,v) + \text{minBreakTime}(\text{breakDist}(l) + \text{len}(u,v) + \text{chPot}_t(v)) \\
&\quad - \text{minBreakTime}(\text{breakDist}(l) + \text{chPot}_t(u)) - \text{chPot}_t(u) + \text{chPot}_t(v) \\
&\overset{\text{(1. and 4.)}}{\geq} \text{len}(u,v) + \text{minBreakTime}(\text{breakDist}(l) + \text{chPot}_t(u)) \\
&\quad - \text{minBreakTime}(\text{breakDist}(l) + \text{chPot}_t(u)) - \text{chPot}_t(u) + \text{chPot}_t(v) \\
&= \text{len}(u,v) - \text{chPot}_t(u) + \text{chPot}_t(v) \\
&\overset{\text{(4.)}}{\geq} 0
\end{aligned}
$$

□

**Lemma 5.4.** *The sum* $\text{travelTime}(l) + \text{pot}_t(l,v)$ *of a label* $l$ *at a node* $v$ *is a lower bound for the travel time from* $s$ *to* $t$ *of the route using* $l$*.*

*Proof.* Let $r$ be a route between $s$ and $t$ with the path $p = \langle s = v_0, v_1, \ldots, t = v_k \rangle$. The route is represented by labels $l_i$ at nodes $v_i$. With Lemma 5.3 and $\text{pot}_t(l_k, v_k) = 0$ follows

$$\begin{aligned}\text{travelTime}(l_i) + \text{pot}_t(l_i, v_i) &\leq \text{travelTime}(l_{i+1}) + \text{pot}_t(l_{i+1}, v_{i+1}) \\ &\leq \cdots \leq \text{travelTime}(l_k) + \text{pot}_t(l_k, v_k) \\ &= \text{travelTime}(l_k) = \text{travelTime}(r)\end{aligned}$$

□

**Theorem 5.5.** *The search can be stopped when the first label at $t$ is removed from the queue.*

*Proof.* Let $l_t$ be the first label at $t$ which is removed from the queue during an $s$-$t$ query. It represents a route $r$ from $s$ to $t$ with a travel time of $\text{travelTime}(r) = \text{travelTime}(l_t)$. When the label $l_t$ is removed from the queue, all remaining labels $l$ at nodes $v$ in the queue fulfill $\text{travelTime}(l) + \text{pot}_t(l, v) \geq \text{travelTime}(l_t) + \text{pot}_t(l_t, t)$. The same holds for all labels which will be inserted into the queue at a later point in time (Lemma 5.3). Assume for contradiction that $r$ is not the shortest possible route from $s$ to $t$. Then, a shorter route $r'$ exists which uses at least one unsettled label $l \in L(v)$ at a node $v$. The label $l$ is eventually propagated to $t$ where a label $l'_t \in L(t)$ is created to represent the route $r'$. With Lemmas 5.3, 5.4, and $\text{pot}_t(l_t, t) = 0$ follows

$$\begin{aligned}\text{travelTime}(r) = \text{travelTime}(l_t) &= \text{travelTime}(l_t) + \text{pot}_t(l_t, t) \\ &\overset{(5.3)}{\leq} \text{travelTime}(l) + \text{pot}_t(l, v) \\ &\overset{(5.4)}{\leq} \text{travelTime}(l'_t) = \text{travelTime}(r')\end{aligned} \tag{5.8}$$

which contradicts the assumption that $r'$ yields a shorter $s$-$t$ route than $r$. Therefore, it must be $\text{travelTime}(r) = \mu_{tt}(s, t)$ when $l_t$ was removed from the queue. The search can be stopped when the first label at $t$ is removed from the queue. □

## 5.3. Multiple Driving Time Constraints

Dijkstra's algorithm for the Truck Driver Routing PRoblem with one driving time constraint (TDRP-1DTC) can be adapted to handle multiple driving time constraints $c_i$ (TDRP-mDTC). With a number of $|C|$ driving time constraints, a label $l$ now contains the total travel time $\text{travelTime}(l)$ and $|C|$ distances $\text{breakDist}_1(l), \ldots, \text{breakDist}_{|C|}(l)$. Each value $\text{breakDist}_i(l)$ represents the distance since the last break at a node $v$ with break time $\text{breakTime}(v) \geq c_i^b$. Pausing at a node occurs with one of the available break times $c_i^b$ of a driving time constraint $c_i \in C$. Pausing with an arbitrary break time is permitted but yields longer travel times and no advantage and is therefore ignored. When a route breaks at $v$ for a time $c_i^b$, the corresponding label $l \in L(v)$ has $\text{breakDist}(l) = 0$ for all $0 < j \leq i$ since the breaks with shorter break times are included in the longer break. In the following, we redefine the fundamental concepts of Section 5.1 for multiple driving time constraints.

**Label Propagation** Label propagation simply extends the component-wise addition of the edge weight. 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 $\text{travelTime}(l') = \text{travelTime}(l) + \text{len}(e)$, $\text{breakDist}_i(l') = \text{breakDist}_i(l) + \text{len}(e)\ \forall\ 1 \leq i \leq |C|$, and $\text{pred}(l') = l$.

**Label Pruning** The pruning rule for driving time constraints is generalized in a similar way. A label is discarded if $\text{breakDist}_i(l) > c_i^d$ for any $i$ with $0 < i \leq |C|$.

**Label Dominance** Label dominance can be generalized to multiple driving time constraints as follows.

**Definition 5.6** (Label Dominance)**.** *A label $l \in L(v)$ dominates another label $l' \in L(v)$ if* $\text{travelTime}(l') \geq \text{travelTime}(l)$ *and* $\text{breakDist}_i(l') \geq \text{breakDist}_i(l)\ \forall\ 1 \leq i \leq |C|$.

### 5.3.1. Potential for Multiple Driving Time Constraints

TODO: Rewrite for only two dtc?

In Section 5.2.1 we defined the potential $\text{pot}_t(l, v)$ to extend Dijkstra's algorithm with one driving time constraint to a goal-directed search using the A* algorithm. We will now generalize $\text{pot}_t$ for the use with an arbitrary number of driving time constraints.

In Equation 5.1 we used the distance $\mu(v, t)$ without regard for pausing from $v$ to $t$ and the distance $\text{breakDist}(l)$ since the last break on the route to calculate a lower bound for the amount of necessary breaks until we reach the target node. We now have to calculate the lower bound with respect to all driving time constraints. How many breaks of which duration do we need at least to comply with all driving time constraints $c_i$? For longer driving time constraints, we will always need a greater or equal amount of breaks than for shorter driving time constraints since they have a longer maximum allowed driving time $c_i^d$. At the same time, a break of length $c_i^b$ will also include breaks of lengths $c_j^b$ with $j < i$. We start with calculating the amount of necessary breaks $\text{minBreaks}_i(d)$, given a driving time $d$, for all constraints $c_i$ independently.

$$\text{minBreaks}_i(d) = \begin{cases} \left\lceil \frac{d}{c_i^d} \right\rceil - 1 & d > 0 \\ 0 & else \end{cases} \tag{5.9}$$

Consider the example graph in Figure 5.2 with two driving time constraints with permitted driving times of 4 and 9. Since the distance $\mu(s, t)$ is 10, a route must have at least one long and two short breaks. If the long break is made at $u$, only one additional short break must be made at $w$. The long break made one shorter break obsolete. To obtain a lower bound for the amount of breaks for a constraint $c_i$, we therefore must subtract the minimum amount of longer breaks being made on the route.

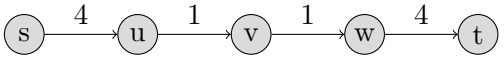



Figure 5.2.: Example graph where a long break at can render a short break obsolete.

This is an optimistic assumption since not in all cases a longer break spares a shorter break. Revisit the example graph of Figure 5.2 with permitted driving times of 4 and 5. We still need one long and two short breaks, but the long break now must take place at $v$ while the short breaks must take place at $u$ and $w$. The long break did not spare a short break. Since we are searching for a lower bound for the amount of breaks, optimistic assumptions are necessary. Given a label $l \in L(v)$, the lower bound estimate for the remaining number of breaks of length $c_i^b$ on the route to $t$ then becomes

$$\text{breakEstimate}_i(l, v) = \begin{cases} \text{minBreaks}_i(\text{breakDist}_i(l) + \text{chPot}_t(v)) & 0 < i < |C| \\ \quad - \sum_{j=i+1}^{|C|} \text{breakEstimate}_j(l, v) & \\ \text{minBreaks}_{|C|}(\text{breakDist}_{|C|}(l) + \text{chPot}_t(v)) & i = |C| \end{cases} \tag{5.10}$$

Since we subtract all break estimates for break times greater than $c_i^b$ to obtain the estimate for $c_i^b$, we can just use

$$\text{breakEstimate}_i(l,v) = \begin{cases} \text{minBreaks}_i(\text{breakDist}_i(l) + \text{chPot}_t(v)) & 0 < i < |C| \\ \quad - \text{minBreaks}_{i+1}(\text{breakDist}_{i+1}(l) + \text{chPot}_t(v)) & \\ \text{minBreaks}_{|C|}(\text{breakDist}_{|C|}(l) + \text{chPot}_t(v)) & i = |C| \end{cases} \tag{5.11}$$

We now can calculate a lower bound estimate for the remaining necessary break time on the route to $t$ for two driving time constraints.

$$\text{remBreakTime}(l,v) = \sum_{i=1}^{|C|} \text{breakEstimate}_i(l,v) \cdot c_i^b \tag{5.12}$$

Finally, the lower bound potential for a label $l \in L(v)$ and a target node $t$ becomes

$$\text{pot}_t(l,v) = \text{remBreakTime}(l,v) + \text{chPot}_t(v,t) \tag{5.13}$$

If queue keys still cannot decrease when propagating labels, Lemma 5.4 and theorem 5.5 follow as a consequence. We follow the outLine of the proof of Lemma 5.3 and therefore revisit the procedure RelaxEdge at an edge $(u,v) \in E$ with a label $l \in L(u)$ and a new label $l' \in L(v)$.

**Lemma 5.7.** *Lemma 5.3 still holds for two driving time constraints.*

*Proof.* There now are three cases to differentiate: not parking at $v$, short break at $v$, and long break at $v$.

Following general observations can be made in an addition to the proof of Lemma 5.3:

6. $d \geq d' \implies \text{minBreaks}_i(d) \geq \text{minBreaks}_i(d')$ (adaption of 1.)
7. $1 + \text{minBreaks}_i(d) \geq \text{minBreaks}_i(d + c_i^d)$ (adaption of 2.)
8. $c_i^d \geq \text{breakDist}_i(l') \geq \text{breakDist}_i(l) + \text{len}(u,v) \geq \text{breakDist}_i(l)$ (adaption of 3.)

*Case 1: Not parking at $v$.* In this case, $\text{travelTime}(l') = \text{travelTime}(l) + \text{len}(u,v)$ and $\text{breakDist}_i(l') = \text{breakDist}_i(l) + \text{len}(u,v)$.

$$
\begin{aligned}
&\text{travelTime}(l') - \text{travelTime}(l) - \text{pot}_t(l,u) + \text{pot}_t(l',v) \\
&= \text{travelTime}(l) + \text{len}(u,v) - \text{travelTime}(l) \\
&\qquad - (\text{remBreakTime}(l,u) + \text{chPot}_t(u)) + (\text{remBreakTime}(l',v) + \text{chPot}_t(v)) \\
&= \text{len}(u,v) - \text{remBreakTime}(l,u) + \text{remBreakTime}(l',v) - \text{chPot}_t(u) + \text{chPot}_t(v) \\
&= \text{len}(u,v) + \text{breakEstimate}_2(l',v) \cdot c_2^b + \text{breakEstimate}_1(l',v) \cdot c_1^b \\
&\qquad - \text{remBreakTime}(l,u) - \text{chPot}_t(u) + \text{chPot}_t(v) \\
&= \text{len}(u,v) + \text{minBreaks}_2(\text{breakDist}_2(l) + \text{len}(u,v) + \text{chPot}_t(v)) \cdot c_2^b \\
&\qquad + (\text{minBreaks}_1(\text{breakDist}_1(l) + \text{len}(u,v) + \text{chPot}_t(v)) \\
&\qquad - \text{minBreaks}_2(\text{breakDist}_2(l) + \text{len}(u,v) + \text{chPot}_t(v))) \cdot c_1^b \\
&\qquad - \text{remBreakTime}(l,u) - \text{chPot}_t(u) + \text{chPot}_t(v) \\
&\overset{\text{(5. and 6.)}}{\geq} \text{len}(u,v) + \text{minBreaks}_2(\text{breakDist}_2(l) + \text{chPot}_t(u)) \cdot c_2^b \\
&\qquad + (\text{minBreaks}_1(\text{breakDist}_1(l) + \text{chPot}_t(u)) \\
&\qquad - \text{minBreaks}_2(\text{breakDist}_2(l) + \text{chPot}_t(u))) \cdot c_1^b \\
&\qquad - \text{remBreakTime}(l,u) - \text{chPot}_t(u) + \text{chPot}_t(v) \\
&= \text{len}(u,v) + \text{remBreakTime}(l,u) - \text{remBreakTime}(l,u) - \text{chPot}_t(u) + \text{chPot}_t(v) \\
&= \text{len}(u,v) - \text{chPot}_t(u) + \text{chPot}_t(v) \\
&\overset{\text{(4.)}}{\geq} 0
\end{aligned}
\tag{5.14}
$$

*Case 2: Short break at $v$.* In this case, $\text{travelTime}(l') = \text{travelTime}(l) + \text{len}(u,v) + c^b$ and $\text{breakDist}_1(l') = 0$ and $\text{breakDist}_2(l') = \text{breakDist}_2(l) + \text{len}(u,v)$.

$$
\begin{aligned}
&\text{travelTime}(l') - \text{travelTime}(l) - \text{pot}_t(l,u) + \text{pot}_t(l',v) \\
&= \text{travelTime}(l) + \text{len}(u,v) + c_1^b - \text{travelTime}(l) \\
&\quad - (\text{remBreakTime}(l,u) + \text{chPot}_t(u)) + (\text{remBreakTime}(l',v) + \text{chPot}_t(v)) \\
&= \text{len}(u,v) + c_1^b + \text{remBreakTime}(l',v) \\
&\quad - \text{remBreakTime}(l,u) - \text{chPot}_t(u) + \text{chPot}_t(v) \\
&= \text{len}(u,v) + c_1^b + \text{breakEstimate}_2(l',v) \cdot c_2^b + \text{breakEstimate}_1(l',v) \cdot c_1^b \\
&\quad - \text{remBreakTime}(l,u) - \text{chPot}_t(u) + \text{chPot}_t(v) \\
&= \text{len}(u,v) + c_1^b + \text{minBreaks}_2(\text{breakDist}_2(l) + \text{len}(u,v) + \text{chPot}_t(v)) \cdot c_2^b \\
&\quad + (\text{minBreaks}_1(0 + \text{chPot}_t(v)) \\
&\quad - \text{minBreaks}_2(\text{breakDist}_2(l) + \text{len}(u,v) + \text{chPot}_t(v))) \cdot c_1^b \\
&\quad - \text{remBreakTime}(l,u) - \text{chPot}_t(u) + \text{chPot}_t(v) \\
&\overset{(7.)}{\geq} \text{len}(u,v) + \text{minBreaks}_2(\text{breakDist}_2(l) + \text{len}(u,v) + \text{chPot}_t(v)) \cdot c_2^b \\
&\quad + (\text{minBreaks}_1(r_1^d + \text{chPot}_t(v)) \\
&\quad - \text{minBreaks}_2(\text{breakDist}_2(l) + \text{len}(u,v) + \text{chPot}_t(v))) \cdot c_1^b \\
&\quad - \text{remBreakTime}(l,u) - \text{chPot}_t(u) + \text{chPot}_t(v) \\
&\overset{(6.\text{ and }8.)}{\geq} \text{len}(u,v) + \text{minBreaks}_2(\text{breakDist}_2(l) + \text{len}(u,v) + \text{chPot}_t(v)) \cdot c_2^b \\
&\quad + (\text{minBreaks}_1(\text{breakDist}_1(l) + \text{len}(u,v) + \text{chPot}_t(v)) \\
&\quad - \text{minBreaks}_2(\text{breakDist}_2(l) + \text{len}(u,v) + \text{chPot}_t(v))) \cdot c_1^b \\
&\quad - \text{remBreakTime}(l,u) - \text{chPot}_t(u) + \text{chPot}_t(v) \\
&\overset{(5.\text{ and }6.)}{\geq} \text{len}(u,v) + \text{minBreaks}_2(\text{breakDist}_2(l) + \text{chPot}_t(u)) \cdot c_2^b \\
&\quad + (\text{minBreaks}_1(\text{breakDist}_1(l) + \text{chPot}_t(u)) \\
&\quad - \text{minBreaks}_2(\text{breakDist}_2(l) + \text{chPot}_t(u))) \cdot c_1^b \\
&\quad - \text{remBreakTime}(l,u) - \text{chPot}_t(u) + \text{chPot}_t(v) \\
&= \text{len}(u,v) + \text{remBreakTime}(l,u) - \text{remBreakTime}(l,u) - \text{chPot}_t(u) + \text{chPot}_t(v) \\
&= \text{len}(u,v) - \text{chPot}_t(u) + \text{chPot}_t(v) \\
&\overset{(4.)}{\geq} 0
\end{aligned}
\tag{5.15}
$$

*Case 3: Long break at $v$.* In this case, $\text{travelTime}(l') = \text{travelTime}(l) + \text{len}(u,v) + c^b$ and $\text{breakDist}_i(l') = 0$.

$$
\begin{aligned}
&\text{travelTime}(l') - \text{travelTime}(l) - \text{pot}_t(l,u) + \text{pot}_t(l',v) \\
&= \text{travelTime}(l) + \text{len}(u,v) + c_2^b - \text{travelTime}(l) \\
&\quad - (\text{remBreakTime}(l,u) + \text{chPot}_t(u)) + (\text{remBreakTime}(l',v) + \text{chPot}_t(v)) \\
&= \text{len}(u,v) + c_2^b + \text{remBreakTime}(l',v) \\
&\quad - \text{remBreakTime}(l,u) - \text{chPot}_t(u) + \text{chPot}_t(v) \\
&= \text{len}(u,v) + c_2^b + \text{breakEstimate}_2(l',v) \cdot c_2^b + \text{breakEstimate}_1(l',v) \cdot c_1^b \\
&\quad - \text{remBreakTime}(l,u) - \text{chPot}_t(u) + \text{chPot}_t(v) \\
&= \text{len}(u,v) + c_2^b + \text{minBreaks}_2(0 + \text{chPot}_t(v)) \cdot c_2^b \\
&\quad + (\text{minBreaks}_1(0 + \text{chPot}_t(v)) - \text{minBreaks}_2(0 + \text{chPot}_t(v))) \cdot c_1^b \\
&\quad - \text{remBreakTime}(l,u) - \text{chPot}_t(u) + \text{chPot}_t(v) \\
&\overset{(7.)}{\geq} \text{len}(u,v) + \text{minBreaks}_2(c_2^d + \text{chPot}_t(v)) \cdot c_2^b \\
&\quad + (\text{minBreaks}_1(c_1^d + \text{chPot}_t(v)) - \text{minBreaks}_2(c_2^d + \text{chPot}_t(v))) \cdot c_1^b \\
&\quad - \text{remBreakTime}(l,u) - \text{chPot}_t(u) + \text{chPot}_t(v) \\
&\overset{(6.\text{ and }8.)}{\geq} \text{len}(u,v) + \text{minBreaks}_2(\text{breakDist}_2(l) + \text{len}(u,v) + \text{chPot}_t(v)) \cdot c_2^b \\
&\quad + (\text{minBreaks}_1(\text{breakDist}_1(l) + \text{len}(u,v) + \text{chPot}_t(v)) \\
&\quad - \text{minBreaks}_2(\text{breakDist}_2(l) + \text{len}(u,v) + \text{chPot}_t(v))) \cdot c_1^b \\
&\quad - \text{remBreakTime}(l,u) - \text{chPot}_t(u) + \text{chPot}_t(v) \\
&\overset{(5.\text{ and }6.)}{\geq} \text{len}(u,v) + \text{minBreaks}_2(\text{breakDist}_2(l) + \text{chPot}_t(u)) \cdot c_2^b \\
&\quad + (\text{minBreaks}_1(\text{breakDist}_1(l) + \text{chPot}_t(u)) \\
&\quad - \text{minBreaks}_2(\text{breakDist}_2(l) + \text{chPot}_t(u))) \cdot c_1^b \\
&\quad - \text{remBreakTime}(l,u) - \text{chPot}_t(u) + \text{chPot}_t(v) \\
&= \text{len}(u,v) + \text{remBreakTime}(l,u) - \text{remBreakTime}(l,u) - \text{chPot}_t(u) + \text{chPot}_t(v) \\
&= \text{len}(u,v) - \text{chPot}_t(u) + \text{chPot}_t(v) \\
&\overset{(4.)}{\geq} 0
\end{aligned}
\tag{5.16}
$$

□

## 5.4. Bidirectional Goal-Directed Search with Multiple Driving Time Constraints

We now extend the goal-directed approach of Section 5.3.1 to a bidirectional approach similar to the bidirectional search described in Section 3. Our algorithm will consist of a forward search from $s$ in $\overrightarrow{G}$ and a backward search from $t$ in $\overleftarrow{G}$. The distances of the forward and backward search are combined at nodes were which were settled by both searches. We therefore introduce a concept to merge the label sets of backward and forward search to find the best currently known feasible route using information of both searches. Since we aim to stop the search as early as possible, we have to decide on a stopping criterion which allows the search to stop way before the forward search settles the target node $t$ or the backward search settles $s$. The correctness of the stopping criterion is closely tied to the potential of Section 5.3.1.

The input of the search remains a graph $G = (V, E, \text{len})$, a set of parking nodes $P \subseteq V$, a set of driving time constraints $C$, and start and target nodes $s, t \in V$. There are two potentials $\overrightarrow{\text{pot}_t}$ and $\overleftarrow{\text{pot}_s}$ which we call the forward and the backward potential. The forward potential yields lower bounds for the remaining travel time of a label to $t$ in $\overrightarrow{G} = G$. The backward potential yields lower bounds for the remaining travel time of a label to $s$ in $\overleftarrow{G}$. The forward search then constitutes a normal A* search on $\overrightarrow{G}$ with start node $s$ and target node $t$ and the backward search is a normal A* search on $\overleftarrow{G}$ with start node $t$ and target node $s$. Each search owns a queue of labels $\overrightarrow{Q}$ and $\overleftarrow{Q}$ and a label set $\overrightarrow{L}(v)$, respectively $\overleftarrow{L}(v)$ for each $v \in V$.

During the search, forward and backward search alternately settle nodes until the stopping criterion is met, one search completed the search by itself, or the queues ran empty. We hold the tentative value for $\mu_{tt}(s, t)$ in a variable $tent(s, t)$ which we initialize with $\infty$ before settling the first node. When forward or backward search settles a node $v$, they additionally check if the label set of the other search at $v$ contains any settled labels. If this is the case, forward and backward search met at this node. We then search for the combination of labels which yields the shortest feasible route between $s$ and $t$ via $v$. In other words, we want to find the labels $l \in \overrightarrow{L}(v)$ and $m \in \overleftarrow{L}(v)$ which minimize $\text{travelTime}(l) + \text{travelTime}(m)$ and for which $\text{breakDist}_1(l) + \text{breakDist}_1(m) < c_1^d$ and $\text{breakDist}_1(l) + \text{breakDist}_1(m) < c_2^d$. If the resulting distance for an $s$-$t$ path via $v$ is smaller than the previously known minimum tentative distance $tent(s, t)$, we update $tent(s, t)$ accordingly. We stop the forward search if the minimum key of $\overrightarrow{Q}$ is greater than $tent(s, t)$ and stop the backward search when the minimum key of $\overleftarrow{Q}$ is greater than $tent(s, t)$.

**Theorem 5.8.** *At the point in time when forward and backward search have stopped, $tent(s, t) = \mu_{tt}(s, t)$. In other words, when the search stops, $tent(s, t)$ equals the minimum travel time from $s$ to $t$ which complies with the driving time constraints $C$.*

*Proof.* We show that when the search stops, all feasible $s$-$t$ routes $q$ which comply with the driving time constraints $C$ and which were not found yet yield a larger travel time $\text{travelTime}(q)$ than the current $tent(s, t)$. This also implies that if $tent(s, t) = \infty$ at the point in time when the search stops, there exist no paths from $s$ to $t$.

Since the search stops when the minimum keys of $\overrightarrow{Q}$ and $\overleftarrow{Q}$ are both greater than $tent(s, t)$, all labels $l$ which will be settled by continuing the search have a greater distance $\text{travelTime}(l)$. Therefore, if any new connection between forward and backward search which complies with the driving time constraints will be found, its distance will be greater than $tent(s, t)$.

The shortest path with travel time $\mu_{tt}(s, t)$ consists of two subpaths of forward and backward search which were connected at a node $v$. There exist label $l \in \overrightarrow{L}$ and $m \in \overleftarrow{L}$ with $\text{travelTime}(l) + \text{travelTime}(m) = \mu_{tt}(s, t)$. Each label is the result of a unidirectional search from $s$, respectively from $t$. In Section 5.1.4 we proved that a unidirectional search can be stopped when the first label at its target node was removed from the queue. Since $l$ and $m$ are both smaller or equal to $tent(s, t)$ and both queue keys of forward and backward queue are greater, $l$ and $m$ where already removed from the respective queue. Therefore, we know that $\overrightarrow{L}(v)$ and $\overleftarrow{L}(v)$ contain the labels with the shortest distance from $s$ to $v$, respectively from $t$ to $v$. Consequently, when the second of both labels was settled at $v$, $\mu_{tt}(s, t)$ was updated with the value $\text{travelTime}(l) + \text{travelTime}(m)$. □

## 5.5. Goal-Directed Core Contraction Hierarchy Search

Given a graph $G = (V, E, \text{len})$ and a parking set $P$, we construct a core contraction hierarchy $G^* = (V^*, E^*, \text{len}^*)$ as described in Section 3.2 in which the core contains all the parking

nodes and no other nodes. The graph $G^+ = (V^+, E^+, \text{len}^+)$ with nodes $V^+ = V^* \setminus P$ and edges $E^+ = \{(u, v) \in E^* \mid u, v \in V^+\}$ therefore is a valid contraction hierarchy which does not contain any parking nodes. The core graph $G_C = (P, E^* \setminus E^+, \text{len}^*)$ contains only parking nodes and (shortcut) edges between them.

We implement the goal-directed core CH query by reusing the bidirectional, goal-directed approach of Section 5.4 on a core CH $G^*$ which is split into a forward graph $\overrightarrow{G^*}$ and a backward graph $\overleftarrow{G^*}$. The forward graph $\overrightarrow{G^*}$ contains all nodes $V^*$ of the core CH, all upward edges of the contracted nodes $V^+$ and all edges of the core graph. The backward core CH graph also contains all nodes $V^*$ of the core CH and all edges of the core graph. Additionally, the backward graph contains all reversed downward edges of the contracted nodes.

The bidirectional query in $G^*$ consequently consists of a forward search from $s$ in $\overrightarrow{G^*}$ and backward search from $t$ in $\overleftarrow{G^*}$. Each search consists of two parts, an upward search in $G^+$ and a search in $G_C$. A search may begin at a core node skip the upward part, it also may not reach the core graph at all and only perform the CH upward search. This behavior is a result of the modelling of the core CH with a backward and forward graph and does not require any change of the bidirectional algorithm.

The stopping criterion of the bidirectional search must take into account that the graph $G^*$ contains the contraction hierarchy $G^+$. The common stopping criterion for $s$-$t$ queries in a CH is to stop the search if the minimum queue key of both queues $\overrightarrow{Q}$ and $\overleftarrow{Q}$ is greater or equal to the tentative minimum distance $\mu(s, t)$ [GSSV12]. Since this criterion is a valid stopping criterion for the bidirectional A* algorithm, we can use it for our combination of CH and bidirectional A*.

**Correctness**

*Proof.* We derive the correctness of the core contraction hierarchy search from the correctness of a CH search and the correctness of the bidirectional A* algorithm. The label set of a node $v \in V^+$ can never contain more than one label for the forward search and one label for the backward search. Let $C'$ be the set of nodes which are reachable from any core node, i.e. $C' = C \cup \{v \mid (u, v) \in E^* \wedge u \in C\}$.

*Case 1: Neither forward nor backward search settle a node in $C'$.* No parking is involved since $P = C$ and $C \subseteq C'$. The query constitutes a simple CH query without labels, extended by pruning with the driving time constraints and goal-direction. The correctness directly follows from the correctness of a CH query and the correctness the bidirectional A* algorithm of Section 5.3 which ensures correct pruning with the driving time constraints, correct concatenation of forward and backward paths, and the correctness of the stopping criterion.

*Case 2: Forward or backward search settle a node in $C'$, but not both.* No feasible $s$-$t$ route which uses a node $v \in C'$ exists per definition of $C'$. The correctness of the query is trivial since forward and backward search either cannot settle a common node or the query constitutes case 1 which we already have proven to be correct.

*Case 3: Both forward and backward search settle a node in $C'$.* The query continues in the core as a bidirectional goal-directed search as described in Section 5.3. If the two searches settle a common node in the core, the correctness of the bidirectional A* algorithm ensures correctness. It still may occur that the searches meet during their upward searches in the contracted part of the core CH after one or both searches have settled core nodes. In this case, the correctness follows from the proof of case 1. □

## 5.6. Improvements and Implementation

In this section, we describe modifications to the algorithm described in Section 5.5 which enable further improvements of running time.

**Label Queue** In Section 5.1, we introduced our basic approach as an algorithm which maintains one queue $Q$ of labels in contrast to Dijkstra's algorithm, which maintains one queue of nodes. In practice, we realize this by maintaining a queue of nodes $Q$ and a priority queue of labels per node $L(v)$ instead of a label set. The key of a node $v$ in $Q$ is the best known travel time of a label in $L(v)$. When settling a label, we remove a node $v$ from $Q$ and the best label $l$ at $v$ from $L(v)$. If $L(v)$ is not empty now, we insert the node $v$ into $Q$ again with the travel time of the new best label $l' \in L(v)$ as the key.

**Bidirectional Backward Pruning** When running the bidirectional A* algorithm in combination with CH-Potentials, the progress and the potential of the backward search can be used to prune the forward search and vice versa. This pruning was introduced by [SZ18] for node potentials. We will adapt the concept for the use with labels and the potential as defined in Section 5.3.1.

A label $l$ at a node $v$ which was propagated along an edge $(u, v)$ can be discarded if we can proove that all routes using the label are longer or equal to the tentative travel time $tent(s, t)$, i.e., the shortest currently known travel time from $s$ to $t$. The travel time travelTime($l$) of the label $l$ at $v$ is already known and it remains to find a lower bound for the remaining distance to $t$. We will describe the pruning of the forward search, the backward search can be pruned accordingly.

The backwards queue $\overleftarrow{Q}$ contains labels $l' \in \overleftarrow{L}(v)$ with travelTime($l'$) + $\text{pot}_s(l', v)$ as the queue key. Labels are removed from the queue with an increasing key. If a label $l'$ was not yet removed from the backwards queue, we know that travelTime($l'$) + $\text{pot}_s(l', v) \geq \text{minKey}(\overleftarrow{Q})$ holds which gives us a lower bound for the travel time of the label travelTime($l'$) $\geq$ minKey($\overleftarrow{Q}$) − $\text{pot}_s(l', v)$. We can only compute $\text{pot}_s(l', v)$ if the label $l'$ was already inserted into the queue, otherwise $l'$ does not exist yet. We need to resort to a pure node potential which is not a function of labels for the case when $l'$ was not yet inserted into the queue. Such a node potential is the potential pot′ of Section 5.2.1 where we assessed it to be infeasible for the use in the A* algorithm. The potential $\text{pot}'_s(v)$ is a lower bound for $\text{pot}_t(l, v)$ with $l \in \overrightarrow{L}(v)$, but we need an upper bound for the potential to obtain a lower bound for travelTime($l'$). Revisit the observations (2.) and (3.) from the proof of Lemma 5.3:

2. $c^b + \text{minBreakTime}(d) \geq \text{minBreakTime}(d + c^d)$
3. $c^d \geq \text{breakDist}(l') \geq \text{breakDist}(l) + \text{len}(u, v) \geq \text{breakDist}(l)$.

Since the distance since the last break of a label can never be greater than the maximum allowed driving time of the driving time constraint, we can give a tight upper bound of the potential $\text{pot}_t$ for the case of one driving time constraint:

$$\text{pot}_s(l, v) \overset{\text{(2. and 3.)}}{\leq} \text{pot}'_s(v) + c^b \tag{5.17}$$

In the case where $|C| > 1$, we need to add the largest break time of all driving time constraints. We use the adaptions of observations (2.) and (3.) for the case of multiple driving time constraints from the proof of Lemma 5.3.

$$\text{pot}_s(l, v) \overset{\text{(7. and 8.)}}{\leq} \text{pot}'_s(v) + c^b_{|C|} \tag{5.18}$$

We now can to compute a lower bound for the travel time of a label $l' \in \overleftarrow{L(t)}$ from $t$ to any node $v$. This is a lower bound for the remaining distance of a label $l \in \overrightarrow{L}(v)$ to $t$. The label $l$ contains distances since the last break of zero or greater and the label $l'$ started at t with distances since the last break of zero and thus cannot exceed the remaining travel time of $l$ from $v$ to $t$. This gives us a lower bound for the remaining travel time of the label $l$ to $t$ of

$$\text{travelTime}(l) + \text{minKey}(\overleftarrow{Q}) - \text{pot}'_s(v) + c^b_{|C|} \tag{5.19}$$

We do not insert the label $l$ into the label set $\overrightarrow{v}$ if our lower bound exceeds the shortest currently known travel time $tent(s, t)$ between $s$ and $t$ because the label $l$ cannot lead to an improvement. Our final backward pruning is defined below as Algorithm 5.7.

**Algorithm 5.7:** Pruning the forward search before inserting a label $l$ into the forward label set of $v$.

```
1 Procedure PruneBackward(v,l):
2     if L←(v) has a settled label l' then
3         prune if travelTime(l) + travelTime(l') ≥ tent(s,t)
4     else if Q← contains a label l' at v then
5         prune if travelTime(l) + Q←.minKey() − pot_s(l',v) ≥ tent(s,t)
6     else
7         prune if travelTime(l) + Q←.minKey() − pot'_s(v) + c^b_|C| ≥ tent(s,t)
```

**Core Contraction Hierarchy Stopping Criteria** A core CH $G^*$ can be separated into the fully contracted CH $G^+$ and the core graph $G_C$. Only a subset of nodes of a core CH is potentially reachable from a core node. This includes all the core nodes and the top layer of nodes of the CH, i.e., all nodes $v \in V^+$ which are reachable from nodes $v \notin V^+$ via outgoing edges of an uncontracted core node. If the forward search stops without touching any of the nodes which are reachable from the core, the forward search can only connect with the backwards search within the fully constructed CH $G^+$ of the core CH. If the backwards search has no nodes $v \in V^+$ left in its queue, we can be sure that forward and backward search will not connect anymore and that no path will be found. Therefore, we can terminate the search. Equivalently, we can terminate the search if the backward search terminates early without touching any nodes which are reachable from the core and the forward search has no CH nodes $v \in V^+$ left in its queue.

**Constructing the Core Contraction Hierarchy** In Chapter 5.5 we introduced a core contraction hierarchy algorithm with a core of uncontracted nodes $C$ which contains all the parking nodes $P$. It remains to determine if $C = P$ is the optimal choice for C or if it is beneficial to include more nodes in the core. In general, the parking nodes are not the most important nodes in the graph according to the node order of the CH. Because we declare them as core nodes and contract everything but those nodes, we label the parking nodes the most important nodes of the graph, possibly contradicting the computed node

order. This can lead to a CH of lower quality. The idea is to include the most important nodes according to the node order of the CH in the core to obtain a CH of higher quality, i.e. with lower node degrees. An additional advantage are shorter build times for the core CH. TODO finish

# 6. Evaluation

In this section, we evaluate the running time and behavior of our algorithms of Chapter 5. Our machine runs openSUSE Leap 15.3, has 128 GB (8x16 GB) of 2133 MHz DDR4 RAM, and a 4-core Intel Xeon E5-1630v3 CPU which runs at 3.7 GHz. The code is written in Rust and compiled with cargo 1.64.0-nightly using the release profile with `lto = true` and `codegen-units = 1`.

**Data.** Our data is a road network of Europe[1] and of Germany[2] from Open Street Map (OSM). We extract the routing graph and parking nodes from the OSM data using a custom extension[3] of RoutingKit[4]. The obtained routing graph of Europe has 81.5 million nodes and 190 million edges. If not stated otherwise, our set $P$ of parking nodes in the European routing graph consists of 6796 nodes which were selected according to their OSM attributes. Equivalently, the routing graph of Germany has 12.5 million nodes, 29.5 million edges, and we selected 3222 nodes as parking nodes. RoutingKit constructs the routing graph by filtering the OSM data using a rich set of attributes to determine which of the OSM objects can be used for driving with a car. The OSM objects types of importance are OSM nodes and OSM ways. An OSM node is a location with associated geographic coordinates and an OSM way is a polyline consisting of multiple OSM nodes. RoutingKit removes OSM nodes which are only used for modelling of the shape of roads to obtain the set of nodes for the routing graph. It then uses additional attributes to classify the OSM ways into categories with different assumed average speeds. The categorized OSM ways and their spatial length are used to determine driving times between the routing nodes which can then be used as the length function len of the routing graph. In our custom extension, we additionally extract parking nodes from the OSM data. The extraction again is based on attributes which indicate a designated parking location for heavy goods vehicles (HGV), i.e., if `hgv = "yes"` or `hgv = "designated"` or `access = "hgv"` is true. We consider OSM nodes and OSM ways as potential parking areas. If an OSM node is found which is marked as HGV parking, we simply flag it as a routing node so that RoutingKit does not remove it from the graph even if it would consider it a modelling node only. Additionally, we add a parking flag for all parking nodes. Some parking areas are modelled as an area instead of a node. In this case, an OSM way which encloses the parking area is flagged as HGV parking. We need to define parking nodes ourselves which we then can add to the routing graph.

[1] `https://download.geofabrik.de/europe-latest.osm.pbf` of March 22, 2022
[2] `https://download.geofabrik.de/europe/germany-latest.osm.pbf` of March 22, 2022
[3] `https://github.com/maxoe/RoutingKit`
[4] `https://github.com/RoutingKit/RoutingKit`

Additionally, we have to connect these nodes to the rest of the routing graph in order to allow routing from and to them. As a pragmatic solution, we search for OSM nodes on the OSM way modelling the parking area that are part of a second OSM way which is marked as suitable for driving with a car. We flag all of these nodes as routing nodes and parking nodes. RoutingKit will include them in the routing graph. As a result, a parking area which is modelled using an OSM way can result in multiple parking nodes in the routing graph, each of which can be viewed as an entry or exit node. In the routing algorithm, this does not lead to noteworthy overhead since the label which took a break at the exit node of the route through the parking area will always dominate labels which took a break at any entry node. The amount of labels being propagated therefore does not increase.

**Methodology.** All experiments are run sequentially. We conduct experiments regarding the preprocessing time of the core CH and the running time of queries on the extracted routing graph. We average preprocessing running times over 10 runs and running times of $s$-$t$ queries over 1000 queries with $s$ and $t$ independently chosen uniformly at random for each query. If not stated otherwise, we use $c_1$ with $c_1^d = 4.5\,\text{h}$ and $c_1^b = 0.75\,\text{h}$ and $c_2$ with $c_2^d = 9\,\text{h}$ and $c_2^b = 11\,\text{h}$ to approximate the regulations of the EU.

## 6.1. Algorithms

We evaluate the different algorithms of Chapter 5 and the backward pruning of Section 5.6.

First, we compare the running times of queries of the algorithms of Chapter 5 on a German and European road network. We scale the experiment down from the European road network because some variants of the algorithms in this experiment cannot keep up with the performance of the goal-directed core CH algorithm and would render the experiment slow and impracticable.

As Table 6.1 shows, on the German road network, the baseline Dijkstra's algorithm with our amendments for driving time constraints averages at about 30 s of running time. Its bidirectional counterpart without goal-direction almost cuts this running time in half. The goal-directed, unidirectional algorithm on the other hand performs three orders of magnitude better than the baseline, the bidirectional variant of the goal-directed searches even manages to improve the running time by another two orders of magnitude. The goal-directed core CH exhibits the best running times of all algorithms, improving the baseline by a factor of about 10 000 and even slightly outperforming the bidirectional goal-directed algorithm. The non-goal-directed core CH variant falls back significantly behind its goal-directed counterpart with running times still significantly better than the unidirectional goal-directed algorithm.

On the European road network, we omitted the baseline and the bidirectional goal-directed variant since they are too slow or suffer from outliers and would render the experiment impracticable. It appears that the goal-directed variant of the algorithm does not scale very well with longer routes which need an increasing amount of necessary breaks. Most of the performance gain of the goal-directed search in comparison to the baseline originates from the very tight lower-bound given by the CH potentials. If the shortest travel time between two nodes $s$ and $t$ is much larger than $\mu(s, t)$ due to necessary breaks and even detours to parking nodes on the route, then the performance of the goal-directed search degrades. The bidirectional variant can mitigate this disadvantage on the German road network since it connects two routes which each have fewer breaks on the route. On the European network, the bidirectional goal-directed algorithm fails to do so and only multiplies the problem of its unidirectional counterpart.

Both core CH algorithms scale better. The simple core CH algorithm scales the best of all algorithms and the goal-directed core CH variant again shows the best result. It also shows robustness against adding a second driving time constraint on both road networks.

| | | | Germany [ms] | | Europe [ms] | |
|---|---|---|---|---|---|---|
| Goal-Directed | Bidirectional | Core CH | 1DTC | 2DTC | 1DTC | 2DTC |
| ✗ | ✗ | ✗ | 29946.22 | 35205.21 | - | - |
| ✗ | ✓ | ✗ | 16109.46 | 10095.23 | - | - |
| ✓ | ✗ | ✗ | 133.06 | 2.14 | 74051.06 | - |
| ✓ | ✓ | ✗ | 4.23 | 3.60 | 15672.69 | - |
| ✗ | ✓ | ✓ | 114.44 | 156.69 | 11.07 | 20.89 |
| ✓ | ✓ | ✓ | 3.10 | 3.34 | 10.83 | 15.85 |

Table 6.1.: Average running times of random queries on a German and European road network with one or two driving time constraints.

We also provide the median running times in Table 6.2. While the running times of non-goal-directed variants improves a little, the median of the running times of goal-directed variants is much smaller than the average, except for cases in which the average already is very small. The goal-directed variants, especially those without a core CH suffer from outliers. The core CH shrinks the maximal possible search space from the entire graph to much fewer nodes which are reachable via upward edges or are core nodes. Therefore, the outliers are not as bad as for the variants without a core CH.

| | | | Germany [ms] | | Europe [ms] | |
|---|---|---|---|---|---|---|
| Goal-Directed | Bidirectional | Core CH | 1DTC | 2DTC | 1DTC | 2DTC |
| ✗ | ✗ | ✗ | 21755.89 | 25274.51 | - | - |
| ✗ | ✓ | ✗ | 8686.98 | 4319.55 | - | - |
| ✓ | ✗ | ✗ | 1.53 | 1.51 | 836.66 | - |
| ✓ | ✓ | ✗ | 2.35 | 2.00 | 1259.19 | - |
| ✗ | ✓ | ✓ | 98.57 | 137.57 | 11.78 | 22.85 |
| ✓ | ✓ | ✓ | 2.91 | 2.93 | 9.66 | 10.81 |

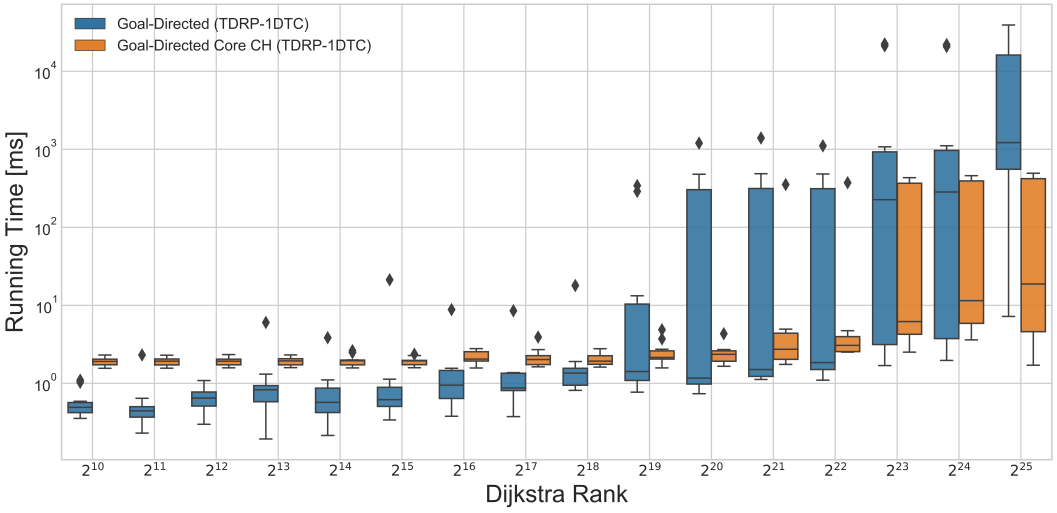
Table 6.2.: Median running times of random queries on a German and European road network with one or two driving time constraints.

We investigate the impact of an increased route length and amount of necessary breaks on a route further. For this, we plot the running times of queries to target nodes of increasing Dijkstra rank on the European road network. The Dijkstra rank is obtained from the sequence in which a standard Dijkstra search without driving time constraints settles its nodes. The Dijkstra rank of a node is the position of the node in that sequence. We plot running times to nodes of rank $2^{10}$, $2^{11}$,...,$2^{\log(|V|)}$ where $|V|$ is the number of nodes in the graph. The plot is shown in Figure 6.1 with a side-by-side comparison of the goal-directed and the goal-directed core CH algorithms.

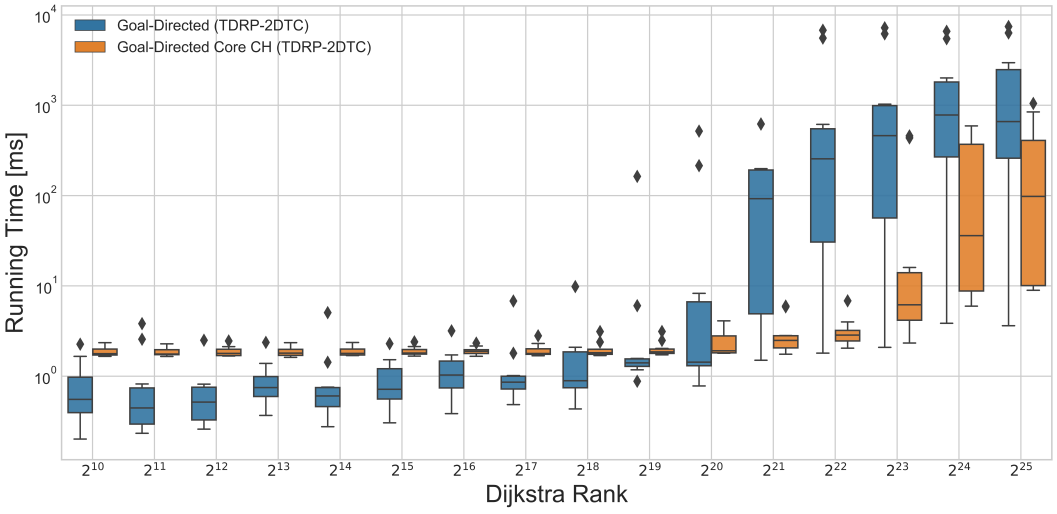
The goal-directed algorithm shows an increase of multiple orders of magnitude for large Dijkstra ranks. Starting at Dijkstra rank $2^{20}$, a significant amount of very slow queries occurs which stretches the interquartile range. The median remains low, almost at the first quartile, an indication that the stretching of the IQR is caused by a few very slow queries. The median of the running time jumps from about 2 ms at Dijkstra rank $2^{22}$ to about 220 ms at Dijkstra rank $2^{23}$. This coincides with the median travel time of the route crossing the mark of 4.5 h which is the maximum allowed driving time without a break of

the shorter driving time constraint of the EU rules being used. Therefore, most queries to target nodes with a Dijkstra rank of $2^{23}$ need at least one break while most queries to target nodes of Dijkstra rank $2^{22}$ do not need to pause on the route yet.

The goal-directed core CH variant scales better with longer queries. It falls slightly behind the goal-directed algorithm for lower Dijkstra ranks due to a larger overhead. While the algorithm suffers from the same general problem of an increasing running time with an increasing route length and number of breaks, the median of its running time increases only by one order of magnitude for the longest queries. Up to a Dijkstra rank of $2^{23}$, the median of the running time does not rise higher than 5 ms and it never exceeds 12 ms. TODO




(a) Goal-Directed and Goal-Directed Core CH Algorithms (TDRP-1DTC)




(b) Goal-Directed and Goal-Directed Core CH Algorithms (TDRP-2DTC)

Figure 6.1.: Running times of queries to target nodes of increasing Dijkstra rank, logarithmic scales. The box represents the interquartile range (IQR) from the first quartile Q1 to the third quartile Q3. The horizontal line within the IQR is the median. The whiskers represent the range from $Q1 - IQR \cdot 1.5$ to $Q3 + IQR \cdot 1.5$ which contains 99.3% of the data points.

The running times of Figure 6.1 to increasingly distant target nodes support the median running times of Table 6.2, but they cannot explain the high average running times of

Table 6.1. The reason for this is that many of the outliers are queries for which no route is found. In fact, when exemplary investigating the 10% of the slowest queries of the goal-directed algorithm, we find that two thirds of those did not find a route. These queries are naturally left out when plotting queries to nodes for which a Dijkstra rank exists. The remaining queries are long queries with a high number of breaks on the route. To complete the evaluation of the different variants of the algorithm, we provide the running times of the queries for which no route was found in Table 6.3.

| | | | Mean [ms] | | Median [ms] | |
|---|---|---|---|---|---|---|
| Goal-Directed | Bidirectional | Core CH | 1-DTC | 2-DTC | 1-DTC | 2-DTC |
| ✓ | ✗ | ✗ | 167739.69 | 1327.55 | 9758.52 | 1.46 |
| ✓ | ✓ | ✗ | 14168.37 | 2454.10 | 7532.67 | 4.60 |
| ✗ | ✓ | ✓ | 321.22 | 631.09 | 346.76 | 571.64 |
| ✓ | ✓ | ✓ | 220.41 | 455.96 | 245.75 | 627.32 |

Table 6.3.: Comparison of running times of queries which failed to find a feasible route.

Finally, we evaluate the backward pruning of as defined in Algorithm 5.7 in use with the goal-directed core CH algorithm on the European road network. Table 6.4 shows that the pruning leads to a significant improvement of running time. TODO weird sample, increase n

| | Mean [ms] | | Median [ms] | |
|---|---|---|---|---|
| Backward Pruning | 1DTC | 2DTC | 1DTC | 2DTC |
| ✗ | 9.68 | 14.72 | 8.48 | 9.89 |
| ✓ | 10.03 | 15.19 | 8.98 | 10.61 |

Table 6.4.: Comparison of running times of the goal-directed core CH algorithm with and without the backward pruning of Section5.6.

## 6.2. Influence of Parameters and Data

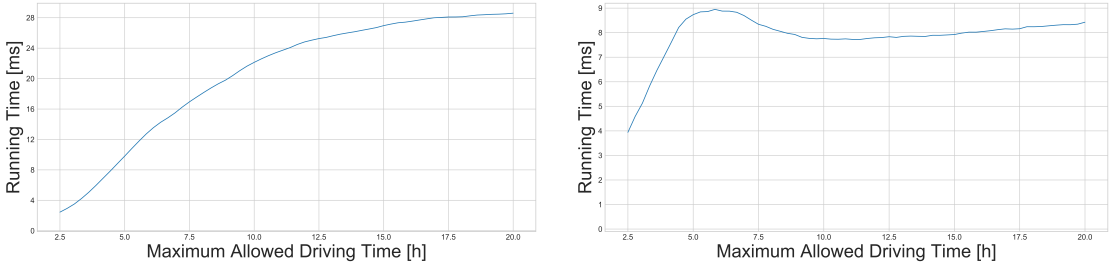
In this section, we investigate how varying parameters and changes in the road network influence the running time of our algorithms.

### 6.2.1. Driving Time Constraints

We investigate how different driving time limits and break times influence the running time of the goal-directed core CH and non-goal-directed core CH algorithms. We use only one driving time constraint $c$ (1DTC) to be able to change one parameter at a time and observe the consequences. Figure 6.2 shows the running times for an increasing maximum allowed driving time $c^d$. We plot the median of 1000 random queries for one value of $c^d$ and increase the driving time limit stepwise. Additionally, the plot is smoothed using a rolling window mean to increase readability.

The goal-directed core CH variant behaves differently compared to the core CH variant, except for very small driving time limits when both variants fail to find a path and terminate early. When increasing the driving time limit, the search radius first increases equally for both. This is because the goal-directed variant presents no advantage over the non-goal-directed variant if a route from $s$ to $t$ is not found because of the driving time constraint. The goal-directed algorithm can exclude nodes $v$ from the search only if the target node $t$ is not reachable from the $v$ at all in the graph, since the CH-Potentials

yield an infinite distance value in this case. It fails to do so if $t$ is reachable from $s$ in the graph, but the driving time constraint prohibits finding a feasible route. In this case, both algorithms settled the same labels, just in a different order. This behavior changes when the driving time limit increased enough that a route can be found and the advantage of the goal-directed algorithm becomes apparent. In the optimal case, which is if a route from a node $v$ to $t$ without the need for a break exists, the CH-Potentials yield the exact distant values $\mu(v, t)$. Therefore, the goal-directed algorithm knows the shortest path to $t$ and does not search aside the shortest route.




(a) Core CH Algorithm

(b) Goal-Directed Core CH Algorithm

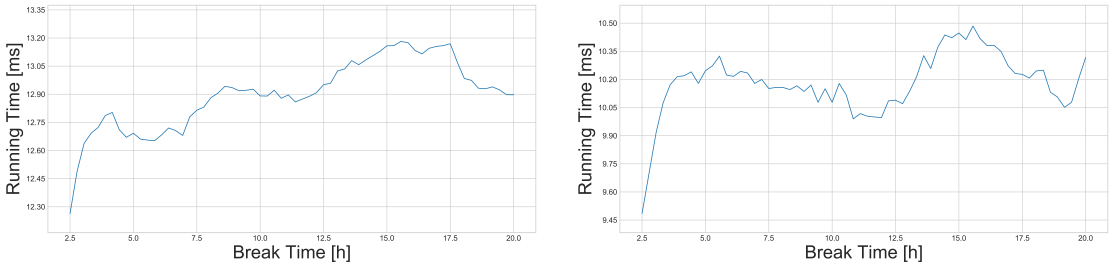
Figure 6.2.: Running times of the two core CH algorithms with increasing maximum allowed driving time.

Second, we vary the break time $c^b$ of the driving time constraint in the same way. Both plots of Figure 6.3 have the same scaling of the y-axis to enable an easy comparison pf the increase in running time. The running time increases when the break time starts to increase from zero. If the break time is exactly zero, there never are multiple labels at a parking node. When duplicating a label at a parking node to represent the two options of taking a break and not taking a break, the label $l$ which took the break will always dominate the other label $l'$ since it resets the distance since the last break to zero without adding any break time to its travel time. With a non-zero break time, the two cases non-goal-directed and goal-directed differ.

In the non-goal-directed case, the labels in the label queue are sorted using their travel time. A label $l$ at a parking node $v$ which takes the break is inserted into the queue at a position further back than the label $l'$ that does not take a break. The search does not settle $l$ at all if the break time which has been added to its travel time increased its travel time over the travel time of the shortest route from $s$ to $t$. If the label $l$ is part of the shortest route, the algorithm spends unnecessary time with propagating labels which have taken fewer breaks and therefore yield a smaller travel time, but will not reach $t$ because they exceed the driving time limit before. The longer the break time, the more of such labels are settled before $l$ which increases the running time. The running time stops to increase significantly when there are no more labels with fewer breaks which the algorithm can propagate before $l$. In this case, the algorithm first propagates all labels which take zero breaks, then all labels which take one break and so on until a feasible route to $t$ can be found.

In the goal-directed case, the labels are sorted in the queue using the sum of their travel time and the potential towards the target node. If a label is positioned further back in the queue, it takes a longer detour from the route which the CH-Potentials have determined to be the shortest route without regard for driving time constraints to $t$. Detours arise from the need to deviate from the direct route to reach a parking node because the direct route exceeded the driving time limit. As a consequence, the algorithm starts to propagate labels along the direct route until either $t$ is found or the driving time limit is exceeded.

It then continues to do so with the label that took the smallest detour. If a label takes a break it only gets positioned further back in the label queue if the break is not of use, i.e., if the break does not spare a break on the remaining route. Such labels cannot be part of a shortest route. TODO dominating, data looks weird




(a) Core CH Algorithm

(b) Goal-Directed Core CH Algorithm

Figure 6.3.: Running times of the core CH and the goal-directed core CH algorithms with an increasing break time.

### 6.2.2. Parking Set of the Road Network

### 6.2.3. Car and Truck Speeds

RoutingKit assumes speeds for different kinds of OSM ways reaching from 5 km/h for walking speed to 130 km/h on highways. The assumed highway speeds are faster than the possible or allowed speed of HGV. In this experiment, we measure the running time of the goal-directed core CH algorithm in multiple iterations in which we cap the maximum speed at the values 130 km/h (standard RoutingKit setting), 100 km/h, 80 km/h, 50 km/h, 30 km/h, 15 km/h, and 5 km/h and observe the change in running time. TODO

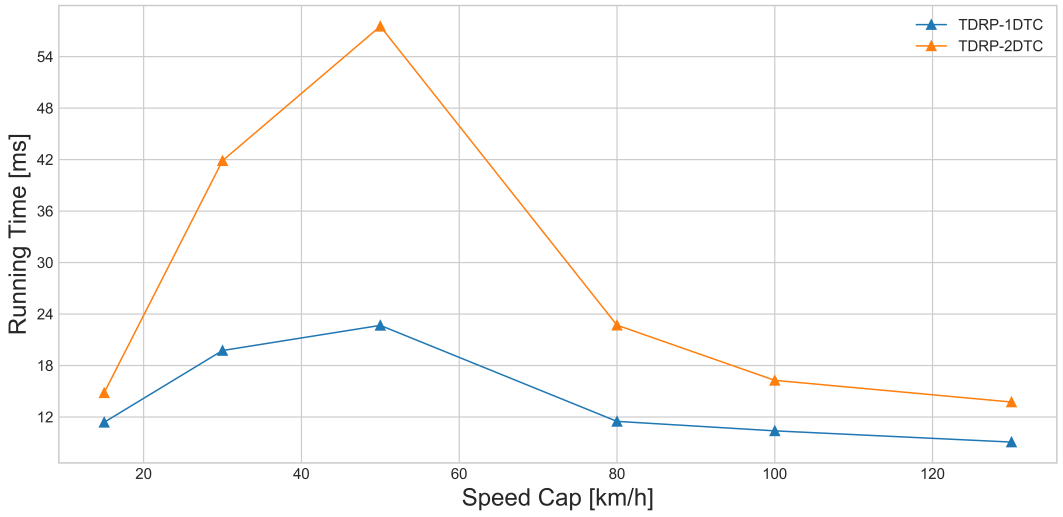



Figure 6.4.: The running time of goal-directed core CH queries increases with a decreasing speed cap, i.e., a decreasing assumed maximum speed of the vehicle.

## 6.3. Core Contraction Hierarchy

In Section 5.6 we mentioned that in general, it is not the best option to choose the set of core nodes as the set of parking nodes. Parking nodes are not necessarily important nodes

according to the computed node order of the core CH, but we define them as the most important nodes of the road network regardless. This impairs the quality of the core CH. It is beneficial to also include the most important nodes according to the node order in the core. In the following, we will analyze the impact of different core sizes on the running time of queries of the goal-directed core CH algorithm on the European road network. Since the choice of the core size also significantly impacts the construction time of the core CH which we also present the construction times in Figure 6.3. TODO

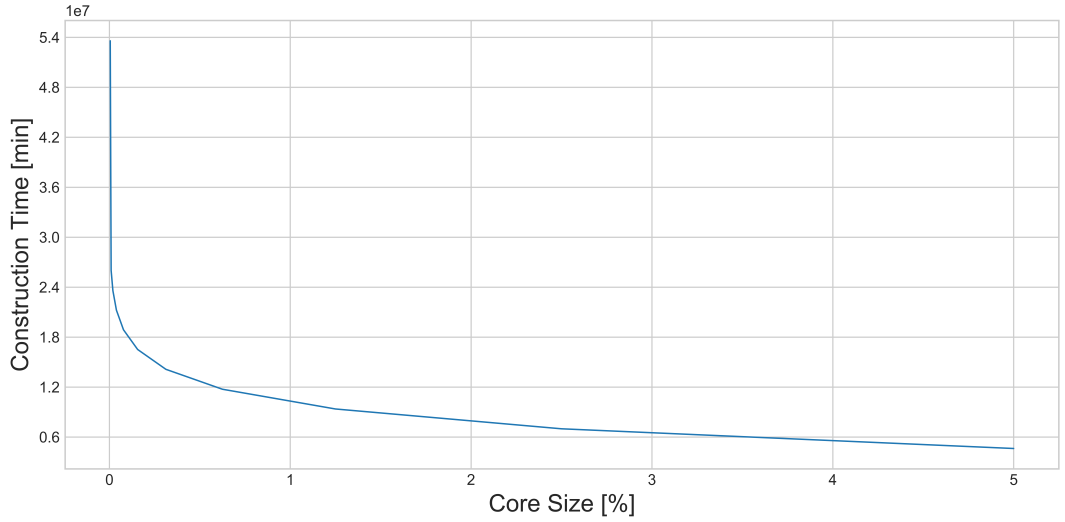



Figure 6.5.: Construction times of the core CH decrease exponentially with an increasing core size relative to the number of nodes in the graph.

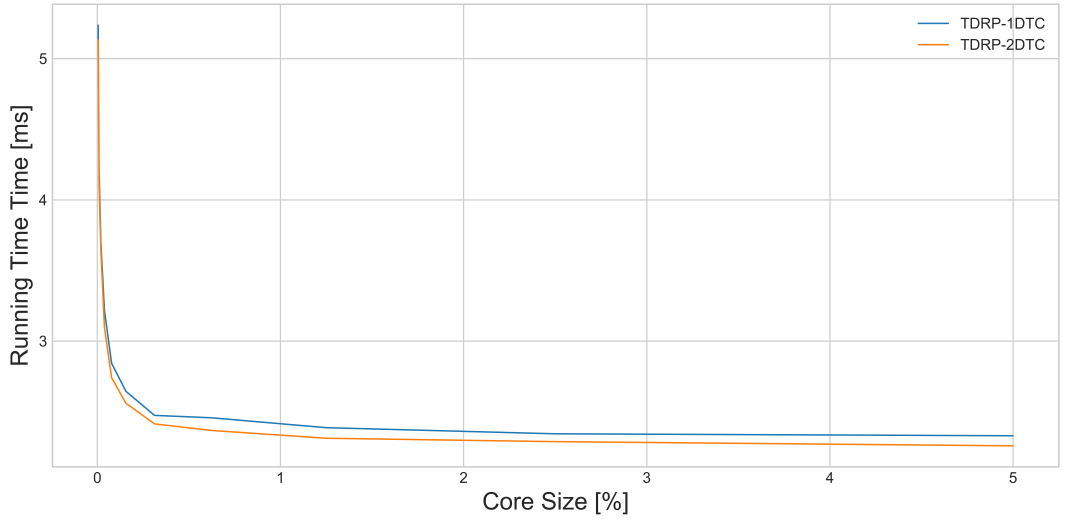



Figure 6.6.: Running time of goal-directed core CH queries for varying core sizes relative to the number of nodes in the graph.

# 7. Conclusion

conclusion - motivtion recap: long-haul - abstraction tdrp - algorithm baseline and goal-directed and core ch extensions - evaluation shows ... results

outlook - theoretical analysis of n dtc - exact model of dtc - merging with time-dependent for road closures and driving bans

# Bibliography


[ADGW12] Ittai Abraham, Daniel Delling, Andrew V. Goldberg, and Renato F. Werneck. Hierarchical Hub Labelings for Shortest Paths. In *Proceedings of the 20th Annual European Symposium on Algorithms (Esa'12), Volume 7501 of Lecture Notes in Computer Science*, pages 24–35. Springer, 2012.

[AG03] Julie Adams-Guppy and Andrew Guppy. Truck driver fatigue risk assessment and management: A multinational survey. *Ergonomics*, 46(8):763–779, June 2003.

[BDG+15] Hannah Bast, Daniel Delling, Andrew Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F. Werneck. Route Planning in Transportation Networks, April 2015.

[BDSV09] G. Veit Batz, Daniel Delling, Peter Sanders, and Christian Vetter. Time-Dependent Contraction Hierarchies. In Irene Finocchi and John Hershberger, editors, *2009 Proceedings of the Eleventh Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 97–105. Society for Industrial and Applied Mathematics, Philadelphia, PA, January 2009.

[BGSV13] G. Veit Batz, Robert Geisberger, Peter Sanders, and Christian Vetter. Minimum time-dependent travel times with contraction hierarchies. *Journal of Experimental Algorithmics (JEA)*, April 2013.

[Bom20] Stefan Bomsdorf. Exact and Heuristic Solution of the Time-Dependent Truck Driver Scheduling and Routing Problem with Two Types of Breaks. Master's thesis, RWTH Aachen, Aachen, 2020.

[Brä16] Christian Bräuer. *Optimale zeitabhängige Pausenplanung für LKW-Fahrer mit integrierter Parkplatzwahl.* Bachelor's thesis, Karlsruhe Institute of Technology, 2016.

[CKMR01] P. Cummings, T. Koepsell, J. Moffat, and F. Rivara. Drowsiness, countermeasures to drowsiness, and the risk of a motor vehicle crash. *Injury prevention : journal of the International Society for Child and Adolescent Injury Prevention*, 2001.

[Cou85] Council of the European Communities. Council Regulation (EEC) No 3820/85 of 20 December 1985 on the harmonization of certain social legislation relating to road transport, 1985.

[CWNJ01] J. Connor, G. Whitlock, R. Norton, and R. Jackson. The role of driver sleepiness in car crashes: A systematic review of epidemiological studies. *Accident; Analysis and Prevention*, 33(1):31–41, January 2001.

[DGNW11] Daniel Delling, Andrew V. Goldberg, Andreas Nowatzyk, and Renato F. Werneck. PHAST: Hardware-Accelerated Shortest Path Trees. In *2011 IEEE International Parallel & Distributed Processing Symposium*, pages 921–931, May 2011.

[DGPW14] Daniel Delling, Andrew V Goldberg, Thomas Pajor, and Renato F. Werneck. Robust Exact Distance Queries on Massive Networks. In *Proceedings of the 22nd Annual European Symposium on Algorithms (ESA'14), Volume 8737 of Lecture Notes in Computer Science*, pages 321–333. Springer, 2014.

[Dij59] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, December 1959.

[DSW16] Julian Dibbelt, Ben Strasser, and Dorothea Wagner. Customizable Contraction Hierarchies. *ACM Journal of Experimental Algorithmics*, 21:1.5:1–1.5:49, April 2016.

[EA05] Claudia Evers and Kerstin Auerbach. Verhaltensbezogene Ursachen schwerer Lkw-Unfälle. BASt Report M 174, Bundesanstalt für Straßenwesen, 2005.

[Eur02] European Parliament. Directive 2002/15/EC of the European Parliament and of the Council of 11 March 2002 on the organisation of the working time of persons performing mobile road transport activities, 2002.

[Eur06] European Parliament. 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., 2006.

[Fed00] Federal Motor Carrier Safety Administration (FMCSA). Federal Register, Volume 65 Issue 85. *Federal Register*, 65(85), 2000.

[Fed06] Federal Motor Carrier Safety Administration (FMCSA). Report to Congress on the Large Truck Crash Causation Study. Report to Congress, 2006.

[Fed11] Federal Motor Carrier Safety Administration (FMCSA). Federal Register, Volume 76 Issue 248. *Federal Register*, 76(248), 2011.

[GH05] Andrew V. Goldberg and Chris Harrelson. Computing the shortest path: A search meets graph theory. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '05, pages 156–165, USA, January 2005. Society for Industrial and Applied Mathematics.

[GK12] Asvin Goel and Leendert Kok. Truck Driver Scheduling in the United States. *Transportation Science*, 46(3):317–326, 2012.

[Goe09] Asvin Goel. Vehicle Scheduling and Routing with Drivers' Working Hours. *Transportation Science*, 43(1):17–26, 2009.

[Goe12] Asvin Goel. The minimum duration truck driver scheduling problem. *EURO Journal on Transportation and Logistics*, 1(4):285–306, December 2012.

[GSSV12] Robert Geisberger, Peter Sanders, Dominik Schultes, and Christian Vetter. Exact Routing in Large Road Networks Using Contraction Hierarchies. *Transportation science*, 46(3):388, 2012.

[GV11] Robert Geisberger and Christian Vetter. Efficient Routing in Road Networks with Turn Costs. In Panos M. Pardalos and Steffen Rebennack, editors, *Experimental Algorithms*, Lecture Notes in Computer Science, pages 100–111, Berlin, Heidelberg, 2011. Springer.

[HNR68] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, July 1968.

[KBS+17] Alexander Kleff, Christian Bräuer, Frank Schulz, Valentin Buchhold, Moritz Baum, and Dorothea Wagner. Time-Dependent Route Planning for Truck Drivers. In Tolga Bektaş, Stefano Coniglio, Antonio Martinez-Sykora, and Stefan Voß, editors, *Computational Logistics*, Lecture Notes in Computer Science, pages 110–126, Cham, 2017. Springer International Publishing.

[Kle19] Alexander Kleff. *Scheduling and Routing of Truck Drivers Considering Regulations on Drivers' Working Hours.* PhD thesis, Karlsruhe Institute of Technology, 2019.

[KSWZ20] Alexander Kleff, Frank Schulz, Jakob Wagenblatt, and Tim Zeitz. Efficient Route Planning with Temporary Driving Bans, Road Closures, and Rated Parking Areas. In *18th International Symposium on Experimental Algorithms (SEA 2020)*, volume 160 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 17:1–17:13, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

[MDGCNR20] Sérgio Fernando Mayerle, Daiane Maria De Genaro Chiroli, João Neiva de Figueiredo, and Hidelbrando Ferreira Rodrigues. The long-haul full-load vehicle routing and truck driver scheduling problem with intermediate stops: An economic impact evaluation of Brazilian policy. *Transportation Research Part A: Policy and Practice*, 140:36–51, October 2020.

[MNR19] Ali Moradi, Seyed Saeed Hashemi Nazari, and Khaled Rahmani. Sleepiness and the risk of road traffic accidents: A systematic review and meta-analysis of previous studies. *Transportation Research Part F: Traffic Psychology and Behaviour*, 65:620–629, August 2019.

[Sha08] Vidit Divyang Shah. Time dependent truck routing and driver scheduling problem with hours of service regulations. Master's thesis, Boston, Massachusetts : Northeastern Universit, 2008.

[SSVB21] Carlo Sartori, Pieter Smet, and Greet Vanden Berghe. Scheduling hours of service for truck drivers with interdependent routes. Technical report, KU Leuven, 2021.

[Sta21] Statistisches Bundesamt (Destatis). Verkehrsunfälle - Zeitreihen - 2020, 2021.

[SZ18] Ben Strasser and Tim Zeitz. Using Incremental Many-to-One Queries to Build a Fast and Tight Heuristic for A* in Road Networks, Preprint, 2018.

[SZ21] Ben Strasser and Tim Zeitz. A Fast and Tight Heuristic for A* in Road Networks. In *19th International Symposium on Experimental Algorithms (SEA 2021)*, volume 190 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 6:1–6:16, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[vdTdWB18] Marieke van der Tuin, Mathijs de Weerdt, and G. Batz. Route Planning with Breaks and Truck Driving Bans Using Time-Dependent Contraction Hierarchies. *Proceedings of the International Conference on Automated Planning and Scheduling*, 28:356–364, June 2018.

[WFFS01] A Williamson, A.-M. Feyer, R Friswell, and S Sadural. Driver Fatigue: A Survey of Professional Long Distance Heavy Vehicle Drivers in Australia. Information Paper/ATSB CR 198, 2001.

# Appendix

## A. List of Abbreviations

| | |
|---|---|
| EU | European Union |
| EC | European Commission |
| EEC | European Economic Community |
| US | United States |
| HOS | Hours of Services |
| DTC | Driving Time Constraint |
| | |
| SPP | Shortest Path Problem |
| TDSP | Truck Driver Scheduling Problem |
| MD-TDSP | Minimum Duration Truck Driver Scheduling Problem |
| TDRP | Truck Driver Routing Problem |
| TDRP-1DTC | Restriction of the Truck Driver Routing Problem to one driving time constraint |
| TDRP-2DTC | Restriction of the Truck Driver Routing Problem to two driving time constraints |
| TDRP-mDTC | Truck Driver Routing Problem with an unrestricted number of driving time constraints |
| TDSRP | Truck Driver Scheduling and Routing Problem |
| TD-TDSRP | Time-Dependent Truck Driver Scheduling and Routing Problem |
| TD-TDSRP-2B | Time-Dependent Truck Driver Scheduling and Routing Problem with two types of breaks (two types of driving time restrictions) |
| VRTDSPIS | Truck Driver Scheduling Problem with Intermediate Stops |
| LH-TDRP | Long-Haul Truck Driver Routing Problem |
| | |
| OSM | Open Street Map |
| HGV | Heavy Goods Vehicle |
| CH | Contraction Hierarchy |
| IQR | Interquartile Range |

## B. List of Symbols and Designations

We use Greek symbols for theoretical and abstract concepts such as the shortest distance between two nodes or an arbitrary valid node potential. Concrete functions and procedures,

i.e., those with definitions, have verbose names. Algorithmic functions and procedures for which pseudocode is provided use SmallCaps.

| | |
|---|---|
| $\pi_t(v)$ | A node potential to $t$ |
| $\mu(s,t)$ | Shortest distance between $s$ and $t$ |
| $\mu_{tt}(s,t)$ | Shortest travel time between $s$ and $t$ |
| breakTime$(v)$ | A function which assigns each node of a route the length of the break at the node or zero, if the route does not take a break at the node. |
| $r$ | A route consisting of a path and a function breakTime$(v)$ |
| $c$ | A driving time constraint |
| $c^d$ | Maximum allowed driving time of a driving time constraint $c$ |
| $c^b$ | Minimum pause time of a driving time constraint $c$ |
| $C$ | A set of one or more driving time constraints |
| $L(v)$ | The label set of a node $v$ |
| breakDist$_i(l)$ | Distance since the last break of a label with a duration of at least $c_i^b$ or distance since the last break if the index $i$ is omitted |
| pred$(l)$ | Predecessor label of a label $l$ |
| travelTime$(l)$ | Travel time of a label $l$ |
| pot$(l,v)$ | The potential of a label $l \in L(v)$ as defined in Section 5.3.1 |
| len$((u,v))$ | The weight, respectively length of an edge $(u,v)$, alternatively used for the length len$(p)$ of a path $p$ |
| $G^+$ | A contraction hierarchy with nodes $V^+$ and edges $E^+$ |
| $G^*$ | A core contraction hierarchy with nodes $V^*$ and edges $E^*$ |