Karlsruhe Institute of Technology

Routenplanung mit temporären Straßensperrungen und ortsabhängigen Wartekosten

Bachelorarbeit
von

Jakob Wagenblatt

An der Fakultät für Informatik
Institut für Theoretische Informatik

Erstgutachter:	Prof. Dr. Dorothea Wagner
Zweitgutachter:	Prof. Dr. Peter Sanders
Betreuende Mitarbeiter:	Christian Bräuer, M. Sc.
	Dr. Alexander Kleff
	Dr. Frank Schulz
	Tim Zeitz, M. Sc.

Bearbeitungszeit: 1. Juli 2019 – 31. Oktober 2019

Eidesstattliche Erklärung

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, 31. Oktober 2019

Zusammenfassung

In dieser Arbeit untersuchen wir das Problem der Routenplanung auf einem Straßennetzwerk mit temporären Sperrungen. In vielen Ländern ist es Lastkraftwagen-Fahrern verboten, bestimmte Straßen in einem spezifizierten Zeitraum zu befahren. Zusätzlich existieren regional beschränkte Sperrungen, wie zum Beispiel Nachtfahrverbote in Städten. Aufgrund dieser Einschränkungen können Fahrer sich gezwungen sehen, auf Parkplätzen zu warten. Wir definieren in dieser Arbeit ein Modell, welches das Finden von geeigneten Parkplätzen bereits in der Routenplanung berücksichtigt. Außerdem priorisieren wir das Warten an Parkplätzen guter Qualität, um den Reisekomfort des Fahrers zu verbessern. Weiterhin minimieren wir sowohl die Ankunftszeit als auch die Fahrzeit. Da die beiden letztgenannten Kriterien jedoch nicht immer vereinbar sind, bieten wir gegebenenfalls mehrere mögliche Routen an. Wir entwerfen einen Algorithmus, der diese Routen effizient berechnet. Anschließend analysieren wir die Eigenschaften und Komplexität des Modells wie auch des Algorithmus. Die Laufzeit des Algorithmus verbessern wir mithilfe einer Kombination aus bewährten und neuen Beschleunigungstechniken. Dies führt zu einer durchschnittlichen Laufzeit von unter einer Sekunde für europaweite Anfragen.

Abstract

In this thesis we investigate the problem of route planning with temporary road-closures. In many countries, truck drivers are prohibited from driving on certain roads for specified periods of time. In addition, there are regionally limited closures, such as night driving bans in cities. Due to these restrictions, drivers may be forced to wait in parking lots. We define a model which already takes the search for suitable parking lots into account during route planning. Additionally, we prioritize waiting on high-quality parking lots to improve the truck driver's comfort. We minimize both the arrival time and the travel time. However, since these two criteria are not always compatible, we may offer several possible routes. We design an algorithm that calculates these routes efficiently. Afterwards, we analyze the properties and complexity of the model as well as the algorithm. We improve the running time of the algorithm using a combination of proven and new speed-up techniques. This leads to an average runtime of less than one second per query on the European road network.

Inhaltsverzeichnis

1 Einleitung **1**
1.1 Literaturübersicht . . . 2
1.2 Überblick . . . 3

2 Grundlagen **5**
2.1 Graph mit Straßensperrungen und Parkplätzen . . . 5
2.2 Routen . . . 6
2.3 Dijkstra-Algorithmus . . . 6
2.4 Pareto-Optimalität . . . 6
2.5 Optimierungskriterium . . . 7

3 Algorithmus **9**
3.1 Beschreibung . . . 9
3.2 Korrektheit . . . 11
3.3 Komplexität . . . 12
3.4 Varianten . . . 18

4 Problemkomplexität **21**

5 Beschleunigungstechniken **25**
5.1 Pruning . . . 25
5.2 A* . . . 26

6 Experimentelle Evaluation **29**
6.1 Versuchsaufbau . . . 29
6.2 Laufzeit . . . 33
6.3 Lösungsqualität . . . 39

7 Fazit **45**

Literaturverzeichnis **47**

1. Einleitung

Der Straßengüterverkehr ist in vielen Ländern Europas beschränkt. Abhängig von der betroffenen Region ist es Lastkraftwagen (Lkw)-Fahrern nicht gestattet, bestimmte Straßen in einem spezifizierten Zeitrahmen zu befahren. Diese Fahrverbote variieren sowohl saisonal als auch regional. Beispielsweise herrscht in Deutschland an Sonn- und Feiertagen für Lkw ein Fahrverbot zwischen 0:00 Uhr und 22:00 Uhr. In der Schweiz, Liechtenstein und Österreich unterliegen die Fahrtzeiten von Lkw-Fahrern noch strengeren Gesetzen. Dort gilt ein tägliches Fahrverbot zwischen 22:00 Uhr und 5:00 Uhr. Zusätzlich ist es ihnen in zahlreichen, auch deutschen Städten nicht erlaubt, nachts zu fahren, um die Nachtruhe der Anwohner zu gewährleisten und die Luftverschmutzung zu reduzieren. Dies hat zur Folge, dass Lkw-Fahrer gezwungen sind, das Ende der Sperrungen entweder auf Parkplätzen abzuwarten oder möglicherweise große Umwege in Kauf zu nehmen. Es ist also im Interesse von Speditionen, bei der Berechnung von schnellsten Routen Straßensperrungen zu berücksichtigen und geeignete Parkplätze zu finden.

Obwohl diese Problematik weit verbreitet ist, ist uns nur eine einzige Publikation [Brä18] bekannt, die das Finden von Parkplätzen bereits in der Routenplanung berücksichtigt. Um schnelle Ergebnisse zu erzielen, wird stattdessen laut [vdTdWB18] derzeit dieses Problem in zwei Teilprobleme aufgeteilt. Diese beiden Teilaspekte können dann effizient sequentiell gelöst werden.
Im ersten Schritt wird unter Berücksichtigung der Sperrungen der kürzeste Pfad gefunden. Anschließend wird im zweiten Schritt, ausgehend von dem initial berechneten Warteort, nach einem möglichen Parkplatz gesucht, falls der Fahrer auf der berechneten Route vor einer Sperrung warten muss. Hierbei ist jedoch zu beachten, dass das verwendete Verfahren eventuell keinen erreichbaren Parkplatz findet oder der gefundene Parkplatz gegebenenfalls nur über einen langen Umweg angefahren werden kann.

In dieser Arbeit stellen wir ein Modell vor, welches das Finden von optimalen Parkplätzen bereits bei der Routenplanung berücksichtigt. Der Fahrer darf hierbei theoretisch überall warten. Die suboptimalen Wartepunkte werden jedoch bei der Evaluierung der Route durch hohe abstrakte Kosten bestraft und daher bestmöglich vermieden. Zusätzlich priorisieren wir das Warten an größeren Parkplätzen, da wir annehmen, dass mit mehr Lkw-Stellplätzen meist ein erhöhter Komfort für die Fahrer einher geht, zum Beispiel durch Restaurants oder Einkaufsmöglichkeiten.
Als Ergebnis einer Anfrage werden dann möglicherweise mehrere Routen angeboten. Diese minimieren die Ankunftszeit und Fahrtzeit, beziehungsweise maximieren den Reisekomfort. Wir entwerfen einen Algorithmus, welcher das modellierte Problem löst und analysieren

dessen Komplexität und Eigenschaften. Schließlich werten wir die Performance und Qualität des Algorithmus experimentell auf dem europäischen Straßennetzwerk aus.

1.1 Literaturübersicht

Das Finden von kürzesten Pfaden auf einem Straßengraphen ist ein intensiv erforschtes Themengebiet. Aufbauend auf dem Algorithmus von Dijkstra [Dij59] wurden viele Beschleunigungstechniken entwickelt. Beispielsweise erlaubt der A^*-Algorithmus [HNR68] zielgerichtetes Suchen. Er verwendet eine Potentialfunktion, die für alle Knoten eine untere Schranke der Distanz zum Ziel angibt. Der ALT-Algorithmus [GH05] ist eine Spezialisierung des A^*-Algorithmus. Die Distanz zum Ziel wird hierbei durch den Einsatz von vorberechneten Distanzen zu Landmarken und unter Ausnutzung der Dreiecks-Ungleichung approximiert.
Eine andere Herangehensweise verwenden Contraction Hierarchies [GH05], welche die inhärente Hierarchie eines Straßennetzwerks ausnutzen. In der Vorberechnung werden Knoten in der Reihenfolge ihrer Priorität aus dem Graph kontrahiert und, falls notwendig, Abkürzungen in den Graphen eingefügt, um kürzeste Distanzen zu erhalten. Durch eine bidirektionale Suche, die nur Kanten zu Knoten höherer Priorität exploriert, können dann kürzeste Pfade berechnet werden. Mithilfe von Contraction Hierarchies kann effizient die Distanz von jedem Knoten zum Ziel bestimmt werden. Diese Eigenschaft wird von Strasser und Zeitz [SZ19] ausgenutzt, um eine perfekte Potentialfunktion für A^* zu berechnen.
Für einen ausführlicheren Überblick über Routing-Algorithmen und Beschleunigungstechniken verweisen wir auf Bast et al. [BDG+16].

Das Finden von Pfaden, die zwei Kriterien, wie zum Beispiel Kosten und CO_2-Emissionen, optimieren, ist allgemein schwierig [Han80]. Wenn die beiden Kriterien unabhängig voneinander sind, können immer Beispiele erstellt werden, deren Lösungsmenge exponentiell mit der Größe des Graphen wächst.
Wenn auf einem Graphen mit Sperrungen das Warten nicht überall erlaubt ist, können Anfragen generiert werden, sodass der Fahrer bei einer früheren Ankunft gezwungen ist einen Umweg in Kauf zu nehmen, um eine Sperrung zu umfahren. Somit kann eine spätere Abfahrt auf diesem Graphen zu einer früheren Ankunft führen. Dies bricht die sogenannte First In - First Out (FIFO)-Eigenschaft. [OR90] zeigten, dass das Routing auf diesen Netzwerken $\mathcal{NP}$-schwer ist.

Das von uns untersuchte Problem ist ebenfalls mit dem Finden von kürzesten Wegen unter Berücksichtigung von Zeitfenstern verwandt. Hierbei wird jedem Knoten oder auch Kanten ein oder mehrere Zeitintervalle zugewiesen, zu denen er erreichbar ist. [PG13] liefert eine ausführliche Übersicht über Ansätze und Algorithmen zur Lösung dieses Problems.

[vdTdWB18] berücksichtigen die gesetzlich vorgeschriebenen Lenk- und Ruhezeiten für Lkw bereits während der Routenplanung. Sie optimieren die Berechnung von Pausen, indem sie Straßensperrungen ausnutzen. Die von ihnen vorgestellte Modellierung erlaubt ebenfalls das Warten an Kanten. Das Finden von Parkplätzen wird jedoch erst in einem zweiten separaten Schritt betrachtet. Weiterhin liefert ihr Algorithmus nur die Route mit der frühesten Ankunftszeit. Eine Reduzierung der Fahrzeit wird in diesem Kontext nicht betrachtet.

Diese Arbeit baut auf der Masterarbeit von Bräuer [Brä18] auf, welche das gleiche Problem untersuchte. Die dort verwendete Modellierung erlaubte jedoch nur das Warten an Parkplätzen und berücksichtigte nicht deren Qualität. Die so berechneten Routen optimierten die Reise- und Fahrzeit, wobei die Reisezeit hierbei der Summe aus Fahrzeit und Wartezeit entspricht. Aufgrund der hohen Komplexität seines Modells und der nicht praxistauglichen Laufzeit seines Algorithmus explorieren wir in dieser Arbeit einen neue Modellierung, um effizient optimale Routen auf Straßengraphen mit temporären Sperrungen zu finden.

1.2 Überblick

In Kapitel 2 definieren wir unser Problemmodell. Weiterhin werden grundlegende Definitionen und die verwendete Notation eingeführt und der Dijkstra-Algorithmus rekapituliert. In Kapitel 3 stellen wir den verwendeten Algorithmus vor, beweisen seine Korrektheit und untersuchen seine Eigenschaften. Wir analysieren anschließend in Kapitel 4 die Komplexität des Problems, unabhängig von der Wahl des Algorithmus. In Kapitel 5 zeigen wir, wie die Berechnung durch Pruning und den Einsatz des A^*-Algorithmus mit einer perfekten Potentialfunktion beschleunigt werden kann. In Kapitel 6 evaluieren wir experimentell die Qualität und Performance des Algorithmus. Schließlich fassen wir unserer Ergebnisse in Kapitel 7 zusammen und geben einen Ausblick auf mögliche zukünftige Forschung.

2. Grundlagen

In den folgenden Abschnitten werden elementare Konzepte und die verwendete Notation eingeführt. Insbesondere definieren wir Graphen mit zeitabhängigen Sperrungen und ortsabhängigen Reisekosten und Routen. Anschließend rekapitulieren wir den Algorithmus von Dijkstra und die Pareto-Optimalität. Darauf aufbauend definieren wir dann das Optimierungskriterium für unser Modell.

2.1 Graph mit Straßensperrungen und Parkplätzen

Ein gerichteter Graph besteht aus einer Menge $\mathcal{V}$ von n *Knoten* und einer Menge $\mathcal{E} \subseteq \{(u, v) \mid u, v \in \mathcal{V}\}$ von m *Kanten*, welche jeweils zwei Knoten verbinden. Wir erweitern den Graph um die Abbildung $d : \mathcal{E} \to \mathbb{N}_+$, welche jeder Kante $e \in \mathcal{E}$ ihre *Fahrtzeit* d_e zuweist, und eine Menge von *Parkplätzen* $\mathcal{P} \subseteq \mathcal{V}$. Diese Parkplatzmenge $\mathcal{P}$ partitionieren wir weiter in q disjunkte Parkplatzmengen $\mathcal{P}_1, \ldots, \mathcal{P}_q$. Die Partitionierung erfolgt gemäß einer vorgegebenen Bewertung der Parkplatzqualität. Alle Parkplätze in $\mathcal{P}_i$ mit $i \in \{1, \ldots, q\}$ sind in der gleichen *Parkplatzkategorie* i. Im Rahmen unserer Arbeit sei das Warten an allen Knoten und auch auf Kanten erlaubt, es wird allerdings je nach Warteort unterschiedlich bestraft. Wir unterscheiden in diesem Modell nicht zwischen einem Warten auf Kanten und an Knoten, die keine Parkplätze sind. Deshalb bezeichnen wir diese Knoten, wie auch alle Kanten, im Folgenden als *Nicht-Parkplätze*.
Jeder Kante e wird eine Intervall-Menge $\mathcal{W}_e = \{[A_1, \Omega_1), \ldots, [A_{w_e}, \Omega_{w_e})\}$ mit $A_i, \Omega_i \in \mathbb{N}_0, A_i < \Omega_i$ für alle $i \in \{1, \ldots, w_e\}$ und $\Omega_j < A_{j+1}$ für alle $j \in \{1, \ldots, w_e - 1\}$ von *Sperrungen* zugewiesen, während derer die Kante nicht traversierbar ist. Wir nennen $\mathcal{W} := \cup_{e \in \mathcal{E}} \mathcal{W}_e$ die Menge aller Sperrungen. Hier sei w_e die Anzahl der Sperrungen dieser Kante und wir verwenden $w_{\mathcal{E}} := \sum_{e \in \mathcal{E}} w_e$ als Kurzschreibweise für die Gesamtzahl der Sperrungen. Wenn eine Kante zu jedem Zeitpunkt traversierbar ist, gilt $\mathcal{W}_e = \emptyset$. Die Dauer einer Sperrung wird durch $|[A_i, \Omega_i)| := \Omega_i - A_i$ beschrieben. Eine Kante darf auch dann befahren werden, wenn eine vollständige Traversierung vor Schließung nicht möglich ist. In diesem Fall muss jedoch auf der Kante bis zur erneuten Öffnung der Kante gewartet werden.
Wir interpretieren $\mathcal{W}_e$ sowohl als Intervall-Menge als auch als Multi-Intervall. Dies ermöglicht die Schreibweise $t \in \mathcal{W}_e$ für einen Zeitpunkt t, falls eine Sperrung $[A_i, \Omega_i)$ für $i \in \{1, \ldots, w_e\}$ existiert mit $A_i \leq t < \Omega_i$.
Wir nennen $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{P}, d, \mathcal{W})$ einen *gerichteten Graph mit Straßensperrungen und Parkplätzen*.

Aus dem Graphen lässt sich die *Reisezeitfunktion* $R_e : \mathbb{N}_+ \to \mathbb{N}_+$ für eine Kante $e = (u, v) \in \mathcal{E}$ ableiten. Diese bildet von der Ankunftszeit t an v auf die Dauer der Traversierung

der Kante ab. Das heißt $t' := t - R_e(t)$ ist der späteste Abfahrtszeitpunkt von u, so dass $|[t', t)| - \sum_{i=i}^{w_e} |[t, t') \cap [A_i, \Omega_i)| \geq d_e$ mit $[A_i, \Omega_i) \in \mathcal{W}_e$ gilt. Es gilt $R_e(t) \geq d_e$ für alle t. Die Wartezeit auf der Kante ergibt sich aus $R_e(t) - d_e$.

2.2 Routen

Ein *Pfad* ist eine Folge von Knoten, die durch Kanten verbunden sind. Sei $(v_1, \ldots, v_x)$ eine solche Folge, dann gilt $(v_i, v_{i+1}) \in \mathcal{E}$ für alle $i \in \{1, \ldots, x-1\}$.
Wir nennen einen Pfad, der um einen Startzeitpunkt $t \in \mathbb{N}_0$ und Wartezeiten erweitert wird, eine *Route* $(t, ((v_1, \tau_v(1)), (e_1, \tau_e(1)), \ldots, (e_{x-1}, \tau_e(x-1)), (v_x, \tau_v(x))))$ oder eine v_1-v_x-t-Route. Hierbei ist zu beachten, dass Routen nicht elementar sein müssen. Es ist somit möglich, dass eine Kante in einer Route mehrfach traversiert wird. Die Funktionen $\tau_v : \{1, \ldots, x\} \to \mathbb{N}_0$ und $\tau_e : \{1, \ldots, x-1\} \to \mathbb{N}_0$ weisen hier jedem Knoten seine und jeder Kante ihre Wartezeit zu. An einem Knoten oder auf einer Kante wird gewartet, wenn die respektive Wartezeit echt größer 0 ist. Aus der Summe der Fahrt- und Wartezeiten kann zusammen mit der Startzeit der Route die *Ankunftszeit* t_a am Endknoten v_x berechnet werden. Es gilt $t_a = t + \sum_{i=1}^{x-1} (d_{e_i} + \tau_e(i) + \tau_v(i)$.
Sei t_i die Ankunftszeit am Knoten v_i auf der v_1-v_x-t-Route mit $i \in \{1, \ldots, x\}$. Wir nennen diese Route zulässig, wenn die Wartezeit $\tau_e(i)$ auf der Kante mindestens der Dauer der Sperrungen $\mathcal{W}_{e_i}$ dieser Kante im Zeitintervall $[t_i + \tau_v(i), t_{i+1})$ entspricht. Es muss also

$$\tau_e(i) - \sum_{j=1}^{w_{e_i}} |[t_i + \tau_v(i), t_{i+1}) \cap [A_j, \Omega_j)| \geq 0$$

gelten.

2.3 Dijkstra-Algorithmus

Sei $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ ein Graph und $d : \mathcal{E} \to \mathbb{N}_+$ die Fahrtzeitabbildung. Gegeben einen Startknoten s berechnet der berechnet der Algorithmus von Dijkstra [Dij59] dann die kürzeste Fahrtdauer von s zu allen erreichbaren Knoten.
Der Algorithmus verwaltet alle bereits erreichten Knoten - deren kürzeste Fahrtzeit noch nicht eindeutig ist - in einer Prioritätswarteschlange, die nach der tentativen Distanz zu diesen Knoten sortiert ist. Initial wird nur der Knoten s in die Warteschlange eingefügt. Wenn ein Knoten $u \in \mathcal{V}$ aus der Warteschlange entfernt wird, ist die berechnete Distanz $dist_u$ zu ihm minimal und kann im Laufe des Algorithmus nicht verbessert werden. Wir nennen diese Eigenschaft *label-setting*. Anschließend wird jede von u ausgehende Kante $e = (u, v)$ relaxiert, indem die Distanz $dist_u + d_e$ via u nach v berechnet wird. Falls dieser Pfad eine Verbesserung darstellt, wird die tentative Distanz $dist_v$ und der korrespondierende Warteschlangeneintrag aktualisiert beziehungsweise v wird in die Warteschlange eingefügt, falls der Knoten bisher noch nicht besucht war. Der Algorithmus terminiert, wenn die Prioritätswarteschlange leer ist. Falls die Anfrage einen Zielknoten $z \in \mathcal{V}$ beinhaltet, wird die Berechnung nach der Entfernung dieses Knotens aus der Warteschlange abgebrochen, da die berechnete Distanz vom Startknoten s zum Zielknoten d dann bereits minimal ist. Wir bezeichnen die Anzahl der besuchten Knoten als *Suchraum* der Anfrage. Einem Knoten v wird der *Dijkstra-Rank* x zugeteilt, wenn er als x-ter Knoten aus der Warteschlange entfernt wird.

2.4 Pareto-Optimalität

Um unser Optimierungskriterium zu beschreiben, führen wir die Begriffe der *Pareto-Dominanz* und der *Pareto-Fronten* ein. Wir definieren diese als:

Definition 2.1 (Pareto-Dominanz)**.** *Seien $x = (x_1, \ldots, x_z)$ und $y = (y_1, \ldots, y_z)$ zwei z-Tupel.*
Wir sagen 'x dominiert y', falls x in allen Werten gleich gut oder kleiner und in mindestens einem Wert echt kleiner als y ist. Wenn also gilt:

$$\forall i \in \{1, \ldots, z\} : x_i \leq y_i \wedge \exists j \in \{1, \ldots, z\} : x_i < y_i$$

Definition 2.2 (Pareto-Front)**.** *Sei X eine Menge von z-Tupeln und $x \in X$. Dann ist x Pareto-optimal, wenn kein Element in X existiert, das x dominiert.*
Eine Teilmenge $L \subseteq X$ wird Pareto-Front genannt, wenn sie alle Pareto-optimalen Elemente von X enthält.

2.5 Optimierungskriterium

Wir nennen den Zeitraum zwischen der frühesten Abfahrtszeit $t_{begin} \in \mathbb{N}_0$ am *Startknoten* $s \in \mathcal{V}$ und der spätesten *Ankunftszeit* $t_{end} \in \mathbb{N}_0$ am Zielknoten $z \in \mathcal{V}$ den *Planungshorizont* $\mathcal{H} = [t_{begin}, t_{end}]$.
Die *Kostenparameter* beschreiben jeweils die Kosten pro Zeiteinheit für das Warten an einem Knoten γ_v oder an einer Kante γ_e und für das Fahren δ mit $\gamma_v, \gamma_e, \delta \in \mathbb{N}_0$. Hierbei weisen Knoten der gleichen Parkplatzkategorie die gleichen Kosten auf. Das Warten an allen Kanten wird mit den Kosten γ_e belegt. Da wir nicht zwischen dem Warten auf einer Kante $e \in \mathcal{E}$ und an einem Knoten $v \in \mathcal{V} \setminus \mathcal{P}$, der kein Parkplatz ist, unterscheiden, setzen wir die jeweiligen Wartekosten gleich und schreiben im Folgenden γ_{NP} für diese. Seien $p_1, \ldots, p_q \in \mathcal{P}$ Parkplätze, wobei p_i ein Parkplatz der Kategorie i ist. Die Parameterbelegung muss die Bedingung $0 \leq \gamma_{p_q} < \cdots < \gamma_{p_1} < \gamma_{\text{NP}} \leq \delta$ erfüllen.
Gegeben einen *gerichteten Graph mit Straßensperrungen und Parkplätzen* $\mathcal{G}$ besteht eine *Anfrage* aus einem Planungshorizont $\mathcal{H} = [t_{begin}, t_{end}]$, einem Startknoten s, Zielknoten z und Kostenparametern.

Sei $(t, ((v_1, \tau_v(1)), (e_1, \tau_e(1)), \ldots, (e_{x-1}, \tau_e(x-1)), (v_x, \tau_v(x))))$ eine Route. Wir definieren die *Fahrtkosten* dieser Route als $\sum_{i=1}^{x-1} d_{e_i} \cdot \delta$ und die *Wartekosten* als $\sum_{i=1}^{x-1} \tau_e(i) \cdot \gamma_{\text{NP}} + \sum_{i=1}^{x} \tau_v(i) \cdot \gamma_{v_i}$. Die *Reisekosten* für diese Route ergeben sich aus der Summe der Fahrt- und Wartekosten.

Problemdefinition

Gegeben eine Anfrage suchen wir die Menge der Routen zwischen dem Start- und Zielknoten, die eine Pareto-Front aus Ankunftszeit und Reisekosten aufspannen. Alle gefundenen Routen müssen im Planungshorizont liegen.

Das Ziel dieser Arbeit ist das effiziente Finden dieser Pareto-Front.

3. Algorithmus

In diesem Kapitel stellen wir die verwendeten Datenstrukturen und den Algorithmus vor, der das gegebene Problem löst. Anschließend beweisen wir die Korrektheit des Algorithmus und untersuchen seine Eigenschaften und Komplexität. Aus diesen Erkenntnissen ziehen wir Rückschlüsse auf die Laufzeit des Algorithmus. Schließlich beschreiben wir eine Variante des Algorithmus, der das Problem mit zeitabhängigen Fahrtkosten löst und erklären, warum eine bidirektionale Suche nicht möglich ist.

3.1 Beschreibung

Wir nennen eine Funktion, welche die Kosten zur Ankunft bezogen auf einen impliziten Startknoten und eine implizite früheste Startzeit an einem Knoten $v \in \mathcal{V}$ zu jedem Zeitpunkt angibt, ein *Kostenprofil* $C_v : \mathbb{N}_0 \to \mathbb{N}_0$ des Knotens v. Es ist eine stückweise lineare Funktion, da das Warten an Knoten und das Fahren mit linearen Kosten belegt ist. Jeder Sprung in der Funktion signalisiert eine Änderung des Pfades oder eines Warteortes. Falls v zu einem Zeitpunkt noch nicht erreichbar ist, ist das Kostenprofil zu diesem Zeitpunkt undefiniert. Wir verwenden im Folgenden $\perp$, um diesen Zustand darzustellen. Bei Vergleichen wird $\perp$ als ∞ ausgewertet.
Wir bezeichnen die linearen Abschnitte des Kostenprofils C_v als *Segmente* $g = (b, c, a)$ mit $b, c, a \in \mathbb{N}_0$. Hierbei beschreibt b den Zeitpunkt, ab dem dieses Segment gültig ist, c die Kosten um v zum Zeitpunkt b zu erreichen und a die Steigung dieses Segments.

Der Algorithmus ist in Algorithmus 3.1 dargestellt. Gegeben einen Startknoten s wird eine Pareto-Front $\mathcal{L}$ aus Tupeln von Ankunftszeit und Reisekosten am Zielknoten z berechnet. Aus diesen Tupeln und den Kostenprofilen der anderen Knoten können die Routen rekonstruiert werden, die z zu diesen Bedingungen erreichen.
Dieser Algorithmus baut auf dem Dijkstra-Algorithmus auf. Anders als dieser hat unser Algorithmus jedoch nicht die label-setting-Eigenschaft. Das heißt, es ist möglich, dass ein Knoten erneut in die Warteschlange hinzugefügt wird, nachdem er bereits aus dieser entfernt wurde.
Das Kostenprofil eines Knotens wird intern als ein Array von Segmenten verwaltet, um eine schnelle Iteration über ein Kostenprofil oder den Vergleich zweier Profile mit einem Sweepline-Algorithmus zu ermöglichen.
Das Kostenprofil C_s am Start wird für alle t im Planungshorizont $\mathcal{H}$ auf 0 gesetzt, ansonsten ist es undefiniert. Der Fahrer kann zu jedem Zeitpunkt innerhalb des Planungshorizonts von s abfahren. Warten am Startknoten wird in keiner Weise bestraft, es gilt also $\gamma_s = 0$. Der

Algorithm 3.1: Berechnung der Pareto-Front

Input: Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{P}, d, \mathcal{W})$, source node s, target node z, planning horizon $\mathcal{H} = [t_{begin}, t_{end}]$, driving cost δ, waiting costs $\gamma_v \ \forall v \in \mathcal{V}$, edge waiting cost γ_{NP}

Data: Priority queue Q

Output: Pareto-Set of arrival time and cost at z

```
// Initialization
forall v ∈ V do
    forall t ∈ ℕ_0 do
        C_v(t) ← ⊥
forall t ∈ H do
    C_s(t) ← 0
Q.INSERT(s, t_begin)
// Main loop
while Q is not empty do
    (u, t_min) ← Q.DELETEMIN()
    forall (u, v) ∈ E do
        C''_v ← LINK(u,v,t_min)
        t_update ←MERGE(C_v, C''_v)
        if t_update ≠ ⊥ and t_update ≤ t_end then
            if Q.CONTAINS(v) then
                Q.DECREASEKEY(v, t_update)
            else
                Q.INSERT(v, t_update)
return {(t, C_z(t)) | t ∈ H, C_z(t) ≠ ⊥, ∄t' ∈ H : t' < t ∧ C_z(t') ≤ C_z(t)}
```

Fahrer muss aber auch z innerhalb des Planungshorizonts erreichen. Die Kostenprofile aller anderen Knoten sind initial für alle Zeitpunkte undefiniert. Die Prioritätswarteschlange besteht aus (v, t) Paaren, wobei v einen Knoten und t einen Zeitpunkt beschreibt. Zu Beginn des Algorithmus wird das Tupel (s, t_{begin}) in die Warteschlange eingefügt. Die Warteschlange ist lexikographisch sortiert nach der Zeit t, dem Wert des Kostenprofils zum Zeitpunkt t und der Steigung des Kostenprofils zu diesem Zeitpunkt.
In der Hauptschleife wird der kleinste Eintrag (u, t_{min}) aus der Warteschlange genommen und jede ausgehende Kante $e = (u, v)$ dieses Knotens relaxiert. Sei C_u das Kostenprofil des Knoten u. Wir berechnen ein partielles Kostenprofil C_v an v durch das Linken des Kostenprofils C_u mit der Kante e in zwei Schritten und das Mergen des resultierenden Kostenprofils mit dem bisher berechneten Kostenprofil an v.

Linken

Schritt 1 (Fahren): Wir berechnen in diesem Schritt die Kosten für das Erreichen des Knotens v via e, indem wir das Kostenprofil C_u mit dieser Kante verknüpfen. Wir setzen

$$C'_v(t) := C_u(t - R_e(t)) + \delta \cdot d_e + \gamma_{\text{NP}} \cdot (R_e(t) - d_e) \tag{3.1}$$

für alle Ankunftszeiten $t \in \mathcal{H}$, für die $t - R_e(t) \geq t_{min}$ gilt. Für alle anderen Ankunftszeiten t setzen wir $C'_v(t) := \bot$. Die Reisezeitfunktion R_e (eingeführt in Abschnitt 2.1) gibt die minimale Zeit an, die es dauert, um Kante e zu traversieren und zum Zeitpunkt t das Ende der Kante zu erreichen.

Schritt 2 (Warten): In diesem Schritt überprüfen wir, ob das Warten am Knoten u einer spätere Ankunft vorzuziehen ist. Es seien γ_v die Kosten für das Warten an v. Wir setzen

$$C''_v(t) := \min\{C'_v(t') + \gamma_v \cdot (t - t') \mid t' \leq t, C'_v(t') \neq \bot\}$$

für alle Ankunftszeiten $t \in \mathcal{H}$ mit $t \geq \alpha(C'_v)$. Sonst setzen wir $C''_v(t) := \bot$. Es sei $\alpha(C'_v)$ der früheste Zeitpunkt, zu dem C'_v definiert ist.
Der erste Schritt des Linkens kann durch einen parallelen Sweepline-Algorithmus über die Segmente des Kostenprofils und die Sperrungen der Kante umgesetzt werden und ist linear in der Summe der beiden Größen. Das Warten kann durch einen einfachen Sweepline-Algorithmus über das Fahrtprofil C'_v realisiert werden. Dies resultiert in einer Laufzeit, die linear in der Größe des Fahrtprofils ist.

Mergen

Falls der Knoten v noch nicht erreicht ist und das Kostenprofil C_v somit für alle t undefiniert ist, setzen wir $C_v := C''_v$. Ansonsten vereinigen wir die beiden Profile anhand folgender Gleichung:

$$C_v(t) := \min\{C_v(t), C''_v(t)\}$$

für alle Ankunftszeiten $t \in \mathcal{H}$. Für alle anderen Ankunftszeiten t setzen wir $C_v(t) := \bot$.
Beim Mergen wird der erste Zeitpunkt t_{update} zurückgegeben, an dem $C_v(t_{update}) > C''_v(t_{update})$ gilt oder $\alpha(C''_v)$, falls $\alpha(C''_v) < \alpha(C_v)$. Falls das neu berechnete Kostenprofil nie dominiert, gilt $t_{update} = \bot$. Wenn das Profil verbessert wird, wird entweder das Schlüssel-Wert-Paar (v,t_{update}) in die Warteschlange eingefügt oder der Warteschlangeneintrag des Knotens v aktualisiert. Das Mergen kann durch einen parallelen Sweepline-Algorithmus über die beiden Kostenprofile realisiert werden. Die Laufzeit dieses Algorithmus ist linear in der Summe der Anzahl der Segmente der Profile.
Der Algorithmus terminiert, wenn die Warteschlange keine Einträge mehr enthält. Die Pareto-Front aus Ankunftszeit und Reisekosten wird aus dem Kostenprofil C_z des Ziels z berechnet und zurückgegeben. Die korrespondierenden Routen können dann folgendermaßen berechnet werden: In jedem Segment ist ein Verweis auf den vorhergehenden Knoten gespeichert. Falls dieses Segment durch Warten an diesem Knoten induziert ist, zeigt dieser auf den Knoten selbst. Zusammen mit dem Beginn des Segments und der Reisezeitfunktion, kann dann das Vorgängersegment gefunden, und somit die gesamte Route rekonstruiert werden.

Um die Segmente eines Kostenprofils und somit auch die Routen zu einem Knoten $v \in \mathcal{V}$ eindeutig festzulegen, wird das Verhalten bei gleichen Kosten definiert.

Definition 3.1 (Dominanz bei gleichen Kosten)**.** *Seien zwei Segmente gegeben, $g = (b, c, a)$ ein Segment des Kostenprofils C_v und $g'' = (b'', c'', a'')$ ein Segment des Kostenprofils C''_v, die zu einem Zeitpunkt $t \geq \max\{b, b''\}$ die gleichen Kosten aufweisen. Zu diesem Zeitpunkt gilt somit $c + a \cdot (t - b) = c'' + a'' \cdot (t - b'')$. Dann dominiert zum Zeitpunkt t das Segment, das früher definiert ist. Wenn jedoch zudem $b = b''$ gilt, dominiert das Segment mit der geringeren Steigung.*

3.2 Korrektheit

Lemma 3.2. *Sei (v, t_v) mit $v \in \mathcal{V}$ der erste Eintrag der Prioritätswarteschlange und $w \in \mathcal{V}$ ein beliebiger Knoten. Dann gilt für alle Zeitpunkte $t \leq t_v$ mit $t \in \mathbb{N}_0$: $C_w(t)$ sind die minimalen Kosten, um w zum Zeitpunkt t zu erreichen.*

Beweis. Angenommen, es gibt eine Möglichkeit w zum Zeitpunkt t mit Kosten $c < C_w(t)$ über die Kante $e = (u, w) \in \mathcal{E}$ zu erreichen.
Dann muss die Kante ausgehend von u spätestens zum Zeitpunkt $t - d_e$ traversiert werden. Da diese besseren Kosten noch nicht in C_w vermerkt sind, existiert ein Eintrag (u, t_u) in der Warteschlange mit $t_u \leq t - d_e \leq t_v - d_e$. Dies stellt allerdings einen Widerspruch zu der in Abschnitt 3.1 beschriebenen Sortierung der Warteschlange dar, weil $t_u < t_v$ gilt.
Somit kann w zum Zeitpunkt t nicht mit geringeren Kosten als $C_w(t)$ erreicht werden. □

Lemma 3.3. *Sei C_v das Kostenprofil eines beliebigen Knotens $v \in \mathcal{V}$. Dann gilt nach Abschluss des Algorithmus, dass für alle $t \in \mathcal{H}$ $C_v(t)$ die minimalen Kosten sind um v von s aus zum Zeitpunkt t zu erreichen.*

Beweis. Sei (w, t_w) der letzte Eintrag, der aus der Warteschlange entfernt wurde. In Lemma 3.2 wurde gezeigt, dass das Profil C_v für alle $t \leq t_w$ minimal ist. Wir betrachten also nun $t > t_w$.
Angenommen, es gibt eine Möglichkeit v zum Zeitpunkt t mit geringeren Kosten $c < C_v(t)$ über die Kante $e = (u, v)$ zu erreichen. Da diese Verbesserung jedoch noch nicht propagiert wurde, existiert ein Warteschlangeneintrag (u, t_u) mit $t_u \leq t - d_e$. Dies steht jedoch im Widerspruch zur Terminierung des Algorithmus. □

Der Algorithmus berechnet somit für alle Knoten und somit auch für den Zielknoten ein minimales Kostenprofil. Aus diesem kann die Pareto-optimale Lösungsmenge einfach berechnet werden.

3.3 Komplexität

Um ein unerwünschtes und möglicherweise verkehrswidriges Verhalten, wie zum Beispiel das mehrfache Umfahren eines Kreisverkehrs, zu verhindern, haben wir in Abschnitt 2.5 die Bedingung

$$\forall v \in \mathcal{V} : \gamma_v \leq \delta \tag{3.2}$$

aufgestellt. In Kombination mit der in Definition 3.1 beschriebenen Dominanz bei gleichen Kosten ist es damit immer sinnvoller zu warten als Kreise zu fahren.

Lemma 3.4. *Eine Route, auf der ohne Warten an einem Parkplatz in einem Kreis gefahren wird, kann nicht Pareto-optimal sein.*

Beweis. Sei $g = (b, c, a)$ das Segmente des Kostenprofils eines Knotens $v \in \mathcal{V}$. Sei $t_{circle} > 0$ die Zeit, die benötigt wird, um beginnend an v zum Zeitpunkt b einen Kreis zu fahren. Die Kosten für das Warten an v bis $b + t_{circle}$ betragen $t_{circle} \cdot \gamma_v$, für das Durchfahren des Kreises fallen Kosten von $t_{circle} \cdot \delta$ an. Das Warten weist somit für $\gamma_v < \delta$ geringere Kosten auf. Wenn die beiden Kosten gleich sind, dominiert dennoch das Warten, da $b < b + t_{circle}$.
Eine Route, auf der ohne Warten an einem Parkplatz im Kreis gefahren wird, ist somit nie Teil eines Kostenprofils und kann deshalb auch nicht Pareto-optimal sein. □

Theorem 3.5. *Wenn das Warten auf Nicht-Parkplätzen nicht mit den gleichen Kosten wie das Fahren belegt wird, kann die Anzahl der Segmente eines Kostenprofils exponentiell mit der Größe des Graphen wachsen.*

Beweis. Zunächst beweisen wir, dass die Wahl $\delta > \gamma_{\text{NP}}$ der Kostenparameter zu einem exponentiellen Wachstum der Größe der Kostenprofile führen kann. Betrachte hierzu den Graphen in Abbildung 3.1. Keiner der Knoten ist ein Parkplatz, die Wartekosten aller

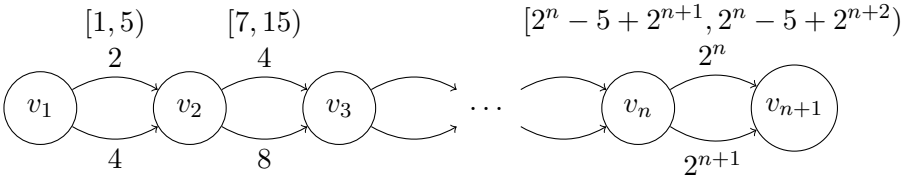

Abbildung 3.1: Beispielgraph mit Sperrungen. Die oberen Kanten sind jeweils im angegebenen Intervall gesperrt und somit nicht traversierbar.

Knoten und Kanten sind somit gleich γ_{NP}. Gestartet wird am Knoten v_1 zum Zeitpunkt 0 mit Kosten 0.
Der Graph ist so konstruiert, dass es immer möglich ist, den nächsten Knoten früher über die untere Kante zu erreichen. Dies geht jedoch mit einer höheren Fahrtzeit einher. Das Traversieren der oberen Kante hingegen ist immer mit Warten auf der Kante verbunden. Wir zeigen nun, dass sich durch die Wahl der Sperrungen und Fahrtdauern die Anzahl der Segmente bei jedem durchlaufenen Knoten für alle Parameterbelegungen, welche die Bedingung $\delta > \gamma_{\text{NP}}$ erfüllen, verdoppeln.

Der Knoten v_2 kann entweder ohne Warten zum Zeitpunkt 4 über die untere Kante mit Kosten $4 \cdot \delta$ oder zum Zeitpunkt 6 mit Warten auf der gesperrten oberen Kante und Kosten von $2 \cdot \delta + 4 \cdot \gamma_{\text{NP}}$ erreicht werden. Dies sind die einzigen optimalen Lösungen.
Das Kostenprofil des Knotens v_3 besteht somit aus Segmenten mit Startzeitpunkten 12, 14, 16 und 18 zu jeweils Kosten $12 \cdot \delta, 10 \cdot \delta + 4 \cdot \gamma_{\text{NP}}, 8 \cdot \delta + 8 \cdot \gamma_{\text{NP}}$ und $6 \cdot \delta + 12 \cdot \gamma_{\text{NP}}$.
Betrachte ein Segment des Knotens v_i mit Startzeit t und initialen Kosten c. Ausgehend von diesem Segment kann der Knoten v_{i+1} entweder mit Warten auf der oberen Kante zum Zeitpunkt $t + 2^i + 2^{i+1}$ mit Kosten $c + 2^i \cdot \delta + 2^{i+1} \cdot \gamma_{\text{NP}}$ oder über die untere Kante zum Zeitpunkt $t + 2^{i+1}$ und Kosten $c + 2^{i+1} \cdot \delta$ erreicht werden.
Aus den 2^{i-1} Segmenten - hier als Tupel aus Startzeit und initialen Kosten dargestellt - an v_i

$$\{(t_i, t_i \cdot \delta), (t_i + 2, (t_i - 2) \cdot \delta + 4 \cdot \gamma_{\text{NP}}), \ldots, (t_i + \frac{t_i}{2}, \frac{t_i}{2} \cdot \delta + t_i \cdot \gamma_{\text{NP}})\}$$

mit $t_i = 2^{i+1} - 4$ werden somit folgende 2^i Segmente an v_{i+1} generiert:

$$\{(t_{i+1}, t_{i+1} \cdot \delta), (t_{i+1} + 2, (t_{i+1} - 2) \cdot \delta + 4 \cdot \gamma_{\text{NP}}), \ldots, (t_{i+1} + \frac{t_{i+1}}{2}, \frac{t_{i+1}}{2} \cdot \delta + t_{i+1} \cdot \gamma_{\text{NP}})\}$$

mit $t_{i+1} = 2^{i+2} - 4$. Hierbei ist die Steigung aller Segmente gleich γ_{NP}.
Wenn eine spätere Ankunft am Knoten mit niedrigeren Kosten einhergeht, dann wächst die Anzahl der Segmente der Kostenprofile exponentiell. Im Falle von v_2 unter der Bedingung $\delta > \gamma_{\text{NP}}$ gilt somit:

$$\delta > \gamma_{\text{NP}} \Rightarrow 4 \cdot \delta + 2 \cdot \gamma_{\text{NP}} > 2 \cdot \delta + 4 \cdot \gamma_{\text{NP}}$$

Das zweite Segment wird in diesem Fall nicht vom ersten dominiert. Dies stößt nach Konstruktion das exponentielle Wachstum der Größe der Kostenprofile an.

Wir beweisen nun, dass die Wahl $\delta < \gamma_{\text{NP}}$ der Kosten ebenfalls zu einer exponentiell großen Anzahl an Segmente führen kann. Betrachte hierzu wiederum den Graphen in Abbildung 3.1, jedoch in diesem Fall ohne Sperrungen. Dann kann v_2 zu den Zeitpunkten 2 oder 4 mit Kosten $2 \cdot \delta$ beziehungsweise $4 \cdot \delta$ erreicht werden.
Aus den 2^{i-1} Segmenten an v_i

$$\{(t_i, t_i \cdot \delta), (t_i + 2, (t_i + 2) \cdot \delta), \ldots, (2 \cdot t_i, 2 \cdot t_i \cdot \delta)\}$$

mit $t_i = 2^i - 2$ werden folgende 2^i Segmente an v_{i+1} generiert:

$$\{(t_{i+1}, t_{i+1} \cdot \delta), (t_{i+1} + 2, (t_{i+1} + 2) \cdot \delta), \ldots, (2 \cdot t_{i+1}, 2 \cdot t_{i+1} \cdot \delta)\}$$

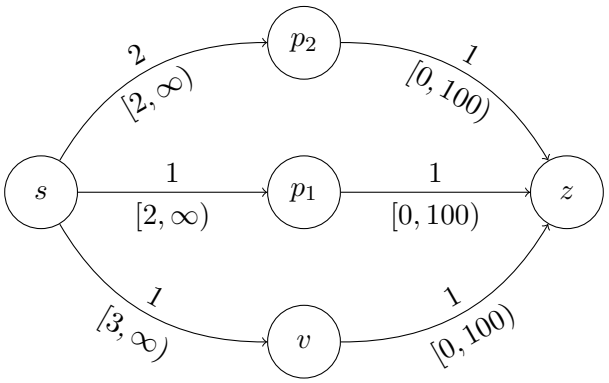

Abbildung 3.2: Beispielgraph, um die Korrelation von Warte- und Fahrtkosten zu verdeutlichen. An s kann ab dem Zeitpunkt 0 gestartet werden, Ziel ist der Knoten z, das Warten an s ist kostenlos. p_1 und p_2 sind zwei Parkplätze, mit $\gamma_{p_1} > \gamma_{p_2}$, der Knoten v ist kein Parkplatz. Die Kanten sind in den angegebenen Intervallen gesperrt.

mit $t_{i+1} = 2^{i+1} - 2$. Die Steigung aller Segmente beträgt wiederum γ_{NP}.
Auch in diesem Fall gilt, dass eine spätere Ankunft mit niedrigeren Kosten zu einem exponentiellen Wachstum der Segmente führt. Am Knoten v_2 ist für $\gamma_{\text{NP}} > \delta$ folgende Ungleichung erfüllt:

$$\gamma_{\text{NP}} > \delta \Rightarrow 2 \cdot \delta + 2 \cdot \gamma_{\text{NP}} > 4 \cdot \delta$$

Aus dem exponentiellen Wachstum der Größe der Kostenprofile für sowohl $\delta > \gamma_{\text{NP}}$ als auch $\delta < \gamma_{\text{NP}}$ folgt die Behauptung. □

Um ein exponentielles Wachstum des benötigten Speichers und somit auch der Laufzeit zu verhindern, dürfen sich die Fahrtkosten somit nicht von den Wartekosten an Nicht-Parkplätzen unterscheiden. Wir setzen deshalb im Folgenden

$$\delta = \gamma_{\text{NP}}. \tag{3.3}$$

Auch wenn diese Bedingung erfüllt ist, kann eine freie Wahl der Kostenparametern unter Berücksichtigung der weiteren Voraussetzungen (siehe 2.5) zu unerwünschten Ergebnissen führen. Schauen wir uns als Beispiel hierfür den Graphen in Abbildung 3.2 an. Der Startknoten s kann ab dem Zeitpunkt 0 verlassen werden, Ziel der Anfrage ist der Knoten z. Der Parkplatz p_2 ist von besserer Qualität als p_1, es gilt $\gamma_{p_2} < \gamma_{p_1}$. Der Knoten v ist kein Parkplatz. Der Graph ist so konstruiert, dass alle Routen z zum Zeitpunkt 101 erreichen. Die Kosten betragen $3 \cdot \delta + 98 \cdot \gamma_{p_2}$ via p_2, $2 \cdot \delta + 98 \cdot \gamma_{p_1}$ via p_1 und $2 \cdot \delta + 97 \cdot \gamma_{\text{NP}}$ via v. Bei einer Kostenbelegung von $\delta = 100, \gamma_{p_1} = 99$ und $\gamma_{p_2} = 98$ dominiert die Route über v, die aufgrund des Wartens auf einem Nicht-Parkplatz unerwünscht ist. Wenn hingegen eine Kostenbelegung von $\delta = 100, \gamma_{p_1} = 2$ und $\gamma_{p_2} = 1$ gewählt wird, ist die Route über den schlechteren Parkplatz p_1 optimal.

Wir lernen daraus, dass ein zu groß gewähltes Verhältnis von Warte- zu Fahrtkosten eine Differenzierung der Parkplatzkategorien erschwert. In diesem Fall wird das Warten an Parkplätzen besserer Qualität nicht ausreichend priorisiert, da nur kurze Umwege in Kauf genommen werden. Sollte dieses Verhältnis andererseits zu klein gewählt sein, wird das Warten an Nicht-Parkplätzen einem Umweg oder einem früheren Startzeitpunkt und dem Warten an einem Parkplatz vorgezogen.

Definition 3.6 (Korrespondierende Routen)**.** *Sei $g = (b, c, a)$ ein Segment des Kostenprofils eines Knotens $v_x \in \mathcal{V}$ und $(t, ((s, \tau_v(1)), (e_1, \tau_e(1)), (v_2, \tau_v(2)), \ldots, (v_x, \tau_v(x))))$ eine Route.*

Wir nennen diese Route eine zu g korrespondierende Route*, wenn die Bedingungen* $t + \sum_{i=1}^{x-1}(d_{e_i} + \tau_e(i) + \tau_v(i)) + \tau_v(x) = b$ *und* $\sum_{i=1}^{x-1}(d_{e_i} \cdot \delta + \tau_v(i) \cdot \gamma_{v_i} + \tau_e(i) \cdot \gamma_{\mathrm{NP}}) + \tau_v(x) \cdot \gamma_{v_x} = c$ *erfüllt sind. Hierbei ist es möglich, dass auf dieser Route an* v_x *gewartet wird. Die Funktion* $K(g)$ *liefert eine beliebige Route, die diese Eigenschaft erfüllt.*

Lemma 3.7. *Seien ein Knoten* $u \in \mathcal{V}$ *und ein Zeitpunkt* t *gegeben. Dann existiert im optimalen Kostenprofil eines Knotens* $v \in \mathcal{V}$ *maximal ein Segment* g*, sodass sich ein Fahrer auf der Route* $K(g)$ *zum Zeitpunkt* t *an* u *befindet und ab diesem Zeitpunkt an keinem Parkplatz mehr wartet.*

Beweis. Angenommen, es gibt zwei solche Segmente $g_1 := (b_1, c_1, a_1)$, $g_2 := (b_2, c_2, a_2)$ mit $b_1 \neq b_2$ im Kostenprofil von v.
Seien $K(g_i)$ zwei beliebige korrespondierende Routen der beiden Profile, welche die Anforderung erfüllen. Seien weiterhin t_i^{drive} die Fahrtzeiten und t_i^{wait} die Wartezeiten an Nicht-Parkplätzen der beiden Routen ab dem Zeitpunkt t, mit $i \in \{1, 2\}$. Die Kosten, um u zum Zeitpunkt t zu erreichen, ergeben sich aus dem Wert des Kostenprofils C_u zu diesem Zeitpunkt.
In Gleichung 3.3 fordern wir die Gleichheit von γ_{NP} und δ.
Aus

$$c_i = C_u(t) + t_i^{wait} \cdot \gamma_{\mathrm{NP}} + t_i^{drive} \cdot \delta = (t_i^{wait} + t_i^{drive}) \cdot \delta = (b_i - t) \cdot \delta$$

folgt $c_i \sim b_i$, da δ konstant und t fest gewählt ist.
Sei o.B.d.A. $b_1 < b_2$. Dann folgt aus obiger Gleichung $c_1 < c_2$.
Die Kosten für die Ankunft zum Zeitpunkt b_1 und anschließendes Warten bis zum Ankunftszeitpunkt des zweiten Segments betragen

$$c_1 + (b_2 - b_1) \cdot \gamma_v \leq c_1 + (b_2 - b_1) \cdot \delta = c_2$$

wegen der Bedingung $\gamma_v \leq \delta$ in Gleichung 3.2.
Aufgrund des Verhaltens bei gleichen Kosten (siehe Definition 3.1) wird das zweite Segment vom ersten dominiert und kann somit nicht Teil des Kostenprofils sein. □

Lemma 3.8. *Die Steigung der Segmente des Kostenprofils* C_v *eines Knotens* $v \in \mathcal{V}$ *können maximal* $q + 2$ *viele verschiedene Werte annehmen.*

Beweis. Bei der Berechnung des Fahrtprofils wird ein gegebenes Segment nur um einen Faktor verschoben. Wenn das Warten an v optimal ist, ist es möglich, dass das Warteprofil um die Steigung γ_v ergänzt wird. Alle anderen Segmente dieses Kostenprofils werden vom Fahrtprofil übernommen.
Das Mergen zweier Profile berechnet das Minimum zweier stückweise linearer Funktionen. Das Resultat hieraus ist wiederum eine stückweise lineare Funktion. In dieser Funktion treten nur Steigungen auf, die bereits in einer der beiden ursprünglichen Funktionen vertreten sind.
Die Steigung eines Segments repräsentiert somit das Warten an einem Knoten $u \in \mathcal{V}$ mit $\gamma_u = a$. Da nur das Warten am Start, an Nicht-Parkplätzen und an den q Parkplatzkategorien unterschiedlich bewertet wird, kann γ_u maximal $q + 2$ viele Werte annehmen. □

Definition 3.9 (Schnitte von Segmenten)**.** *Seien* $g_1 = (b_1, c_1, a_1)$*,* $g_2 = (b_2, c_2, a_2)$ *zwei aufeinanderfolgende Segmente eines Kostenprofils. Wenn die Gleichungen*

$$c_1 + (b_2 - b_1) \cdot a_1 > c_2 \tag{3.4}$$

$$c_1 + ((b_2 - \epsilon) - b_1) \cdot a_1 < c_2 - \epsilon \cdot a_2 \tag{3.5}$$

für alle $\epsilon > 0$ *erfüllt sind, nennen wir* g_2 *durch den Schnitt mit* g_1 *induziert.*

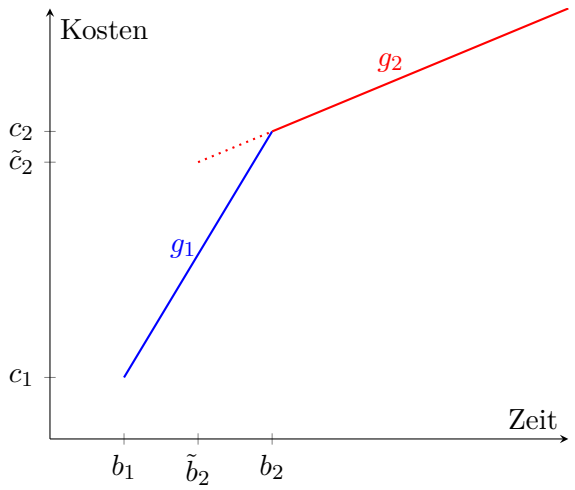

Abbildung 3.3: Beispiel eines Kostenprofils eines Knotens. Das Segment $\tilde{g}_2$ ist ab dem Zeitpunkt $\tilde{b}_2$ mit Kosten $\tilde{c}_2$ definiert. Da es allerdings bis b_2 von g_1 dominiert wird, ist stattdessen das Segment g_2 mit Startzeit b_2 Teil des Kostenprofils. Zu diesem Zeitpunkt gilt $c_1 + (b_2 - b_1) \cdot a_1 > c_2$ Das Segment g_2 ist somit nach Definition 3.9 durch den Schnitt mit g_1 induziert.

In Abbildung 3.3 ist ein Beispiel für ein durch einen Schnitt induziertes Segment dargestellt.

Lemma 3.10. *Auf ein Segment eines Kostenprofils können maximal $q + 1$ viele durch Schnitte induzierte Segmente folgen.*

Beweis. Seien $g_1 = (b_1, c_1, a_1), g_2 = (b_2, c_2, a_2)$ zwei Segmente, g_2 durch den Schnitt mit g_1 induziert.
Setze c_2 aus Gleichung 3.4 in Gleichung 3.5 ein, um

$$c_1 + (b_2 - b_1) \cdot a_1 - \epsilon \cdot a_1 < c_1 + (b_2 - b_1) \cdot a_1 - \epsilon \cdot a_2 \Leftrightarrow a_1 > a_2$$

zu erhalten.
Wenn mehrere Segmente nacheinander durch Schnitte induziert werden, muss somit $a_1 > a_2 > ... > a_{q+2}$ gelten. Aus Lemma 3.8 folgt die Behauptung. □

Definition 3.11 (Pareto-Optimalität von Routen und Segmenten)**.** *Seien t die Ankunftszeit am Ziel und c die Kosten einer Route. Diese Route ist Pareto-optimal, wenn das Tupel (t, c) Pareto-optimal ist. Es existiert also kein Tupel $(t', C_z(t'))$ mit $t' \in \mathcal{H}$, sodass $t' < t$ und $c = C_z(t') \leq C_z(t)$ gilt.*
Ein Segment $g = (b, c, a)$ des Kostenprofils C_z wird Pareto-optimal genannt, wenn das Tupel (b, c) Pareto-optimal ist.

Lemma 3.12. *Sei r eine Pareto-optimale s-z-t-Route und $p \in \mathcal{P}$ ein Parkplatz, an dem auf r bis zum Zeitpunkt t_p gewartet wird.*
Dann existiert eine Kante $e = (u, v) \in \mathcal{E}$ auf r, die im Intervall $[A, \Omega)$, $\Omega \geq t_p$ gesperrt ist und von u ausgehend zum Zeitpunkt Ω traversiert wird.

Beweis. Angenommen, eine solche Kante existiert nicht.
Dann ist es immer optimal, p zu einem früheren Zeitpunkt zu verlassen, um früher mit geringeren Kosten zu z zu gelangen, da es nach Konstruktion keine Sperrung gibt, die dies verhindert. □

Auf einer Pareto-optimalen Route kann somit jedem Parkplatz, auf dem der Fahrer wartet, eine Kante $e = (u, v) \in \mathcal{E}$ zugewiesen werden, die bei Ankunft an u noch geschlossen wäre, wenn auf diesem Parkplatz kürzer gewartet worden wäre.

Lemma 3.13. *Jedes Segment g des Kostenprofils eines Knoten $v \in \mathcal{V}$, auf dessen korrespondierender Route $K(g)$ an einem Parkplatz gewartet wird und das nicht aus einem Schnitt hervorgeht, kann auf eine Sperrung zurückgeführt werden. Jede Sperrung kann hierbei maximal ein Segment in diesem Kostenprofil induzieren.*

Beweis. Betrachte eine beliebige zu einem Segment g korrespondierende Route $K(g)$.
Sei p der Parkplatz auf $K(g)$, an dem zuletzt gewartet wird und der zum Zeitpunkt t_p verlassen wird. Sei $e = (u, w) \in \mathcal{E}$ die Kante, bei der eine Sperrung $[A, \Omega)$, $\Omega \geq t_p$ existiert, die von u ausgehend zum Zeitpunkt Ω - also genau zur Öffnung der Kante - durchfahren wird. Die Existenz einer solchen Kante und Sperrung wurde in Lemma 3.12 gezeigt. Wenn mehrere Sperrungen existieren, die diese Eigenschaft erfüllen, wählen wir die erste.
Aus Lemma 3.7 folgt, dass nur maximal ein Segment auf diese Sperrung zurückzuführen ist. Diese Eigenschaft ist für alle möglichen korrespondierenden Routen erfüllt, die zugewiesene Sperrung kann jedoch variieren. □

Mit diesen Bausteinen kann nun eine obere Schranke für die Anzahl der Segmente eines Kostenprofils eingeführt und bewiesen werden.

Theorem 3.14 (Obere Schranke für Anzahl Segmente)**.** *Sei C_v das Kostenprofil eines Knotens $v \in \mathcal{V}$ und $w_{\mathcal{E}} := \sum_{e \in \mathcal{E}} w_e$ die Anzahl aller Sperrungen. Dann gilt*

$$|C_v| \leq (q + 2) \cdot (w_{\mathcal{E}} + 1) \in \mathcal{O}(q \cdot w_{\mathcal{E}}) \subseteq \mathcal{O}(n \cdot w_{\mathcal{E}})$$

Beweis. Aus Lemma 3.13 folgt, dass es maximal $w_{\mathcal{E}}$ viele durch Sperrungen induzierte Segmente geben kann. Lemma 3.7 zeigt, dass höchstens ein weiteres nicht durch Schnitte oder Sperrungen induziertes Segment existieren kann. Jedes dieser Segmente kann nach Lemma 3.10 höchstens $q + 1$-mal geschnitten werden, woraus die Behauptung folgt. □

Laufzeit des Algorithmus

Lemma 3.15. *Jeder Knoten wird maximal $\mathcal{O}(q \cdot w_{\mathcal{E}})$-mal in die Prioritätswarteschlange eingefügt.*

Beweis. Sei t_1 der Zeitpunkt, an dem der Knoten $v \in \mathcal{V}$ das erste mal aus der Warteschlange genommen wird und $g_1 = (t_1, c_1, a_1)$ das korrespondierende Segment. In Lemma 3.2 wurde gezeigt, dass c_1 die minimalen Kosten sind, um v zum Zeitpunkt t_1 zu erreichen.
Damit v mit dem Eintrag (v, t_2), $t_2 > t_1$ erneut in die Prioritätswarteschlange eingefügt wird, muss das Kostenprofil nach t_1 aktualisiert werden, sodass es möglich ist, v zum Zeitpunkt t_2 billiger oder mit einer geringeren Steigung zu erreichen. Es gibt somit ein Segment $g_2 = (t_2, c_2, a_2)$, das zum Zeitpunkt t_2 optimal ist.
Jedes Einfügen des Knotens ist also auf ein neues Segment zurückzuführen. Die Behauptung folgt aus dem Theorem 3.14. □

Mithilfe dieser Lemmata kann die Laufzeit des Algorithmus hergeleitet und bewiesen werden.

Theorem 3.16. *Die Laufzeit des Algorithmus, dominiert von Sweepline-Algorithmen zum Linken und Mergen und Warteschlangenoperationen, liegt in $\mathcal{O}((q \cdot w_{\mathcal{E}} + 1) \cdot (n \log n + m \cdot (q \cdot w_{\mathcal{E}} + 1)))$.*

Beweis. In Lemma 3.15 wurde gezeigt, dass jeder Knoten maximal $\mathcal{O}(q \cdot w_{\mathcal{E}})$-mal in die Prioritätswarteschlange eingefügt und dementsprechend auch entfernt wird.
In jedem Schleifendurchlauf werden alle ausgehenden Kanten des aktuellen Knotens relaxiert, jede Kante wird insgesamt $\mathcal{O}(q \cdot w_{\mathcal{E}})$-mal relaxiert.
Die Sweepline-Algorithmen zum Linken und Mergen, die bei jeder Kantenrelaxion aufgerufen werden, sind linear in der Größe des Kostenprofils. Das Verringern des Schlüssels erfolgt in konstanter Zeit. In Lemma 3.14 wurde gezeigt, dass die Anzahl der Segmente eines Kostenprofils durch $\mathcal{O}(q \cdot w_{\mathcal{E}})$ beschränkt ist.
Insgesamt ergibt sich eine Laufzeit in $\mathcal{O}((q \cdot w_{\mathcal{E}} + 1) \cdot (n \log n + m))$ für die Prioritätswarteschlangenoperationen und in $\mathcal{O}((q^2 \cdot w_{\mathcal{E}}^2 + 1) \cdot m)$ für das Linken und Mergen, wodurch sich die Gesamtlaufzeit ergibt. □

3.4 Varianten

Zeitabhängige Variante

Mit einer kleinen Modifizierung der Reisezeitfunktion ist es möglich, die Pareto-Front aus Ankunftszeit und Reisekosten mit dem angegebenen Algorithmus ebenfalls auf einem Graphen mit zeitabhängigen Fahrtzeiten zu berechnen.
Sei $d'_e : \mathbb{N}_0 \to \mathbb{N}_+$ in diesem Fall die Funktion, die gegebenen einen Startzeitpunkt t an u die Zeit zur Traversierung der Kante $e = (u, v)$ angibt. Wir fordern, dass d'_e die FIFO-Eigenschaft erfüllt. Für alle $\epsilon > 0$ und alle $t \in \mathbb{N}_0$ muss somit $d'_e(t) \leq d'_e(t + \epsilon) + \epsilon$ gelten. Wir definieren die Reisezeitfunktion als $R_e(t) := \max\{d'_e(t') \mid t' + d'_e(t') = t\}$. Durch Umformen der Gleichung 3.1 erhalten wir

$$C'_v(t) = C_u(t - R_e(t)) + \delta \cdot R_e(t),$$

da die Fahrkosten δ gleich den Wartekosten an Nicht-Parkplätzen γ_{NP} sind.

Wir vermuten, dass die in Abschnitt 3.3 untersuchte Komplexität des allgemeinen Ansatzes auf diese Variante übertragbar ist, haben dies jedoch nicht weiter untersucht. Eine experimentelle Auswertung dieser Variante ist nicht Teil dieser Arbeit.

Bidirektionale Suche

Eine bidirektionale Suche mit diesem Algorithmus zum Berechnen der Pareto-Front ist nicht möglich. In der Rückwärtssuche ist es möglich, dass Pareto-optimale Lösungen verworfen werden. Im Graphen in Abbildung 3.4 ist hierfür ein Beispiel gegeben.
Gestartet wird am Knoten s, Ziel der Anfrage ist der Knoten z. Die Kosten für das Warten am Parkplatz p_1 betragen $\frac{\delta}{2}$. Sowohl die Route, die z zum Zeitpunkt 40 mit Kosten $21 \cdot \delta$, als auch die Route mit Ankunftszeit 41 und Kosten $14 \cdot \delta + 8 \cdot \gamma_{p_1} = 18 \cdot \delta$ sind Pareto-optimal. Bei der Rückwärtssuche wird jedoch die Route mit Ankunftszeit 40 verworfen.

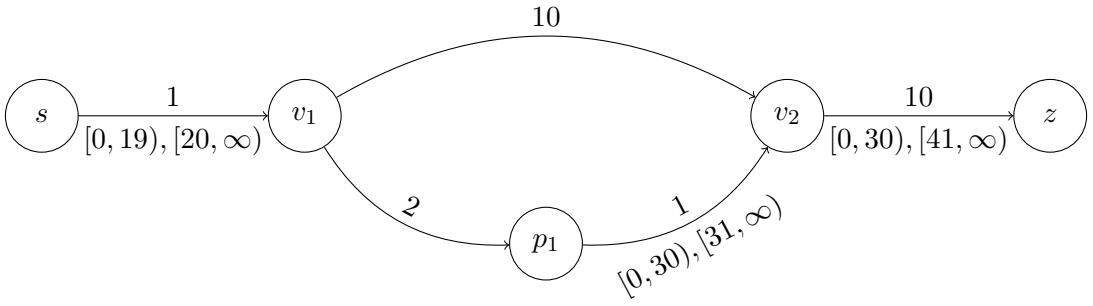

Abbildung 3.4: Beispielgraph, um das Problem der bidirektionalen Suche darzustellen. Es wird an Knoten s gestartet, das Ziel der Anfrage ist der Knoten z. Der Knoten p_1 ist ein Parkplatz mit Wartekosten $\frac{\delta}{2}$.

4. Problemkomplexität

In diesem Kapitel untersuchen wir die Komplexität dieses Modells, unabhängig von der Wahl des Algorithmus. Wir leiten eine obere Schranke für die Kardinalität der Lösungsmenge her und untersuchen das Verhältnis der Kostenparameter.

Lemma 4.1 (Obere Schranke für Anzahl Pareto-optimaler Lösungen)**.** *Unter der Voraussetzung $\gamma_{\mathrm{NP}} = \delta$ ist die Kardinalität der Lösungsmenge $\mathcal{L}$ durch die Gesamtanzahl aller Sperrungen $w_{\mathcal{E}}$ beschränkt.*
Es gilt

$$|\mathcal{L}| \leq w_{\mathcal{E}} + 1 \in \mathcal{O}(w_{\mathcal{E}})$$

Dies ist die kleinste obere Schranke.

Beweis. Die Schranke ist minimal: Betrachte den Graphen in Abbildung 4.1. Der Knoten s ist der Startknoten, an welchem kostenlos gewartet werden kann. Das Ziel der Anfrage ist der Knoten z. Der Planungshorizont beginnt zum Zeitpunkt 0.
Die Sperrungen sind so gewählt, dass eine spätere Ankunft an z mit einer geringeren Fahrtzeit und demnach geringeren Kosten einhergeht. Hierbei reduziert sich die Fahrtzeit für jede spätere Route um jeweils eine Zeiteinheit.
Auf der ersten Route, auf welcher der Fahrer z zum Zeitpunkt $w_{\mathcal{E}}+1$ mit Kosten $(w_{\mathcal{E}}+1)\cdot\delta$ erreicht, wird nicht an s gewartet. Für jede weitere Route erhöht sich die Wartezeit an s um 2 Zeiteinheiten. Zusammen mit der um 1 reduzierten Fahrtzeit erfolgt die Ankunft an z eine Zeiteinheit später mit um δ verringerten Kosten.
Es gibt somit $w_{\mathcal{E}} + 1$ Pareto-optimale Routen.

Der Graph in Abbildung 4.2 verdeutlicht, dass die Kardinalität der Lösung nicht durch die Anzahl der Kanten, sondern durch die der Sperrungen beschränkt ist.
Der Knoten s ist der Startknoten, das Ziel ist der Knoten z. Das Warten an s wird nicht mit Kosten belegt, es kann ab dem Zeitpunkt 0 an s gestartet werden. Auf dem Nicht-Parkplatz v kann zu Kosten γ_{NP} gewartet werden. Beide Kanten sind in den gleichen $x := \frac{w_{\mathcal{E}}}{2}$ Intervallen gesperrt und können in einer Zeiteinheit traversiert werden.
Der Graph ist so konstruiert, dass eine spätere Ankunft an z mit einer geringeren Wartezeit an v und demnach geringeren Kosten einhergeht. Hierbei reduziert sich die Wartezeit für jede spätere Route um jeweils eine Zeiteinheit. Die i-te Sperrung einer Kante ist im Intervall $[i + \sum_{k=x-i+2}^{x} k, i + \sum_{k=x-i+1}^{x} k)$ gesperrt.
Auf der frühesten Route, die z zum Zeitpunkt $x+2$ und Kosten $x\cdot\gamma_{\mathrm{NP}} + 2\cdot\delta = (x+2)\cdot\gamma_{\mathrm{NP}}$

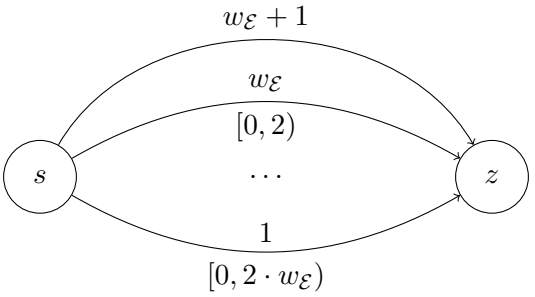

Abbildung 4.1: Beispielgraph, auf dem $w_\mathcal{E} + 1$ Pareto-optimale Routen existieren. Das Warten am Startknoten s ist kostenlos, das Ziel ist der Knoten z. Die Kanten sind in den angegebenen Intervallen gesperrt.

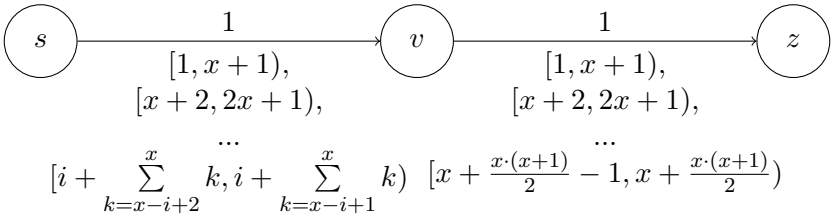

Abbildung 4.2: Beispielgraph, auf dem $\frac{w_\mathcal{E}}{2} + 1$ Pareto-optimale Routen existieren. In dieser Anfrage ist s der Startknoten, der Knoten z ist das Ziel und der Planungshorizont beginnt zum Zeitpunkt 0. Am Startknoten kann kostenlos gewartet werden. Der Knoten v ist ein Nicht-Parkplatz. Das Traversieren beider Kanten dauert eine Zeiteinheit und sie sind jeweils in den angegebenen Intervallen gesperrt. Die Variable x ist definiert als $\frac{w_\mathcal{E}}{2}$.

erreicht, werden x Zeiteinheiten gewartet. Die nächste mögliche Route mit Ankunftszeit $2x + 2$, wartet nur noch $x - 1$ Zeiteinheiten und kostet $(x + 1) \cdot \gamma_{\text{NP}}$. Die späteste optimale Route erreicht z ohne Warten an v und mit Kosten von $2 \cdot \gamma_{\text{NP}}$ zum Zeitpunkt $x + \frac{x(x+1)}{2} + 2$. Es gibt somit $x + 1 = \frac{w_\mathcal{E}}{2} + 1$ Pareto-optimale Routen.

$|\mathcal{L}| \leq w_\mathcal{E} + 1$: In Theorem 3.14 wurde gezeigt, dass ein Kostenprofil aus maximal $(q + 2) \cdot (w_\mathcal{E} + 1)$ vielen Segmenten bestehen kann. Hierbei ist der Faktor $q + 2$ auf durch Schnitte induzierte Segmente zurückzuführen. Diese können jedoch nicht Pareto-optimal sein, da sie jeweils vom vorherigen Segment dominiert werden.
Seien $g_1 = (b_1, c_1, a_1)$, $g_2 = (b_2, c_2, a_2)$ zwei Segmente eines Kostenprofils, g_2 durch den Schnitt mit g_1 induziert. Dann folgt aus der Definition des Schnitts (siehe Definition 3.9), dass $b_2 > b_1$ gilt.
Betrachte Gleichung 3.5, die für alle $\epsilon > 0$ gilt. Setze $\epsilon = b_2 - b_1$, dann folgt

$$c_1 + ((b_2 - \epsilon) - b_1) \cdot a_1 = c_1 < c_2 - \epsilon \cdot a_2 < c_2$$

Das zweite Segment ist somit nicht Pareto-optimal und nicht Teil der Lösung.

Im ersten Teil des Beweises wurden ein Graph und eine Anfrage konstruiert, die eine Lösung aus $w_\mathcal{E} + 1$ Pareto-optimalen Routen liefern. Im zweiten Teil wurde gezeigt, dass die Anzahl der Pareto-optimalen Routen durch $w_\mathcal{E} + 1$ beschränkt ist. Die angegebene obere Schranke ist somit minimal. □

Lemma 4.2. *Wenn die Kostenparameter die Gleichung $\delta > 2 \cdot \gamma_{\text{NP}}$ erfüllen, das Fahren also mehr als doppelt so teuer als das Warten an Nicht-Parkplätzen ist, kann die Größe der Lösungsmenge exponentiell mit der Größe des Graphen wachsen.*

Beweis. Betrachte den Graphen in Abbildung 3.1. Im Beweis zum Theorem 3.5 wurde gezeigt, dass das Kostenprofil des Knoten v_i die Einträge

$$\{(t_i, t_i \cdot \delta), (t_i + 2, (t_i - 2) \cdot \delta + 4 \cdot \gamma_{\text{NP}}), \ldots, (t_i + \frac{t_i}{2}, \frac{t_i}{2} \cdot \delta + t_i \cdot \gamma_{\text{NP}})\}$$

mit $t_i = 2^{i+1} - 4$ und Steigung γ_{NP} enthält. Sei v_i das Ziel der Anfrage. Dann sind alle 2^{i-1} Einträge unter der Voraussetzung $\delta > 2 \cdot \gamma_{\text{NP}}$ pareto-optimal, da

$$t_i \cdot \delta > (t_i - 2) \cdot \delta + 4 \cdot \gamma_{\text{NP}} > \cdots > \frac{t_i}{2} \cdot \delta + t_i \cdot \gamma_{\text{NP}}$$

gilt. □

Betrachte folgendes Entscheidungsproblem:

Definition 4.3 (Routing-Entscheidungsproblem auf einem Graphen mit Straßensperrungen, Reisekosten und Zeitfenstern(RGSRZDEC))**.**

Gegeben: *Eine Anfrage bestehend aus einem Planungshorizont $\mathcal{H}$, einem Startknoten s und Zielknoten z und Kostenparametern auf einem Graphen $\mathcal{G}$ mit zeitabhängigen Sperrungen und ortsabhängigen Reisekosten.*

Frage: *Existiert für ein $k \in \mathbb{N}_0$ eine zulässige Route von s zu z auf $\mathcal{G}$ im Planungshorizont mit Kosten $c \leq k$?*

Lemma 4.4. *Wenn die Kosten für das Fahren kleiner sind als die Kosten für das Warten auf Nicht-Parkplätzen, die Ungleichung $\delta < \gamma_{\text{NP}}$ also erfüllt ist, dann ist* RGSRZDEC *$\mathcal{NP}$-vollständig*

Beweis. Gegeben eine Route auf $\mathcal{G}$. Dann ist es in Linearzeit in der Länge der Route möglich zu überprüfen, ob diese Route eine gültige Lösung ist. Das Entscheidungsproblem ist somit in $\mathcal{NP}$.
Um zu zeigen, dass dieses Problem $\mathcal{NP}$-schwer ist, geben wir eine polynomiale Transformation von PARTITION, das schwach $\mathcal{NP}$-schwer ist [Kar72], zu diesem Entscheidungsproblem an. PARTITION ist definiert als:

Definition 4.5 (PARTITION)**.**

Gegeben: *Eine endliche Menge $\mathcal{M}$ von natürlichen Zahlen.*

Frage: *Existiert eine Teilmenge $\mathcal{M}' \subset \mathcal{M}$ mit*

$$\sum_{m \in \mathcal{M}'} m = \sum_{m \in \mathcal{M} \setminus \mathcal{M}'} m?$$

Sei eine Instanz von PARTITION mit $\mathcal{M} = \{m_1, m_2 \ldots, m_x\}$ gegeben. Diese kann in eine Instanz unseres Entscheidungsproblems überführt werden, indem für alle $i \in \{1, \ldots, x\}$ die Zahl m_i durch einen Knoten v_i repräsentiert wird, der kein Parkplatz ist. Wir erweitern den Graphen um den Startknoten s, Zielknoten z und Hilfsknoten v_{x+1}, der ebenfalls kein Parkplatz ist. Der Planungshorizont beginnt zum Zeitpunkt 0. Zwischen dem Knoten s und v_1 wird eine Kante mit Reisezeit 1 eingefügt, die nur zu Beginn des Planungshorizonts geöffnet ist. Dies verhindert das kostenlose Warten am Start. Der Knoten v_{x+1} ist durch eine Kante mit Reisezeit 1 mit dem Zielknoten verbunden. Die Traversierung dieser Kante ist nur zum Zeitpunkt $a := \sum_{m \in \mathcal{M}} \frac{m}{2} + 1$ möglich. Für alle $i \in \{1, \ldots, x\}$ sind die Knotenpaare (v_i, v_{i+1}) durch zwei Kanten mit Reisezeit 0 beziehungsweise m_i verbunden. In Abbildung 4.3 ist die Transformation visualisiert. Dieser Graph kann in Polynomialzeit ($\mathcal{O}(|\mathcal{M}|)$) konstruiert werden. Wir setzen die maximalen Kosten k auf $(a + 1) \cdot \delta$.

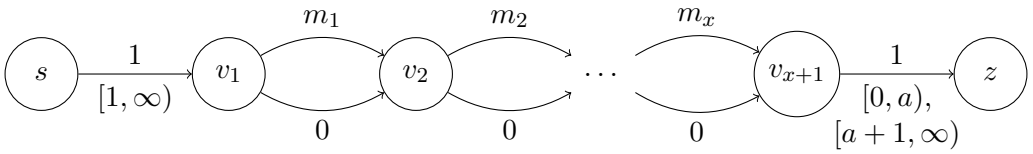

Abbildung 4.3: Visualisierung der Transformation einer Partition Probleminstanz in eine Instanz des hier untersuchten Entscheidungsproblems.

Der Zielknoten kann somit nur zum Zeitpunkt $a + 1$ erreicht werden. Da das Warten am Start nicht möglich ist und das Fahren mit geringeren Kosten als das Warten an Nicht-Parkplätzen belegt ist, sind $(a + 1) \cdot \delta = k$ die minimalen Kosten um z zu erreichen.

Wir zeigen nun, dass Partition erfüllbar ist, genau dann, wenn RgsrzDec erfüllbar ist. Zuerst zeigen wir, dass eine Ja-Instanz von RgsrzDec eine Ja-Instanz von Partition induziert. Sei eine JA-Instanz von RgsrzDec gegeben. Dann existiert eine zulässige Route von s nach z im Planungshorizont mit Kosten $c \leq k$. Wir haben gezeigt, dass die minimalen Kosten dann $(a+1) \cdot \delta$ betragen. Die Summe der Fahrtzeiten auf dieser Route beträgt somit $a+1 = \sum_{m \in \mathcal{M}} \frac{m}{2} + 2$. Wenn die Kanten vom Start und zum Ziel außer Acht gelassen werden, beträgt die Fahrtzeit auf den verbleibenden Kanten somit genau $\sum_{m \in \mathcal{M}} \frac{m}{2}$. Betrachte für alle $i \in \{1, \ldots, x\}$ die gewählte Kante zwischen v_i und v_{i+1}. Das Element m_i der Menge $\mathcal{M}$ wird zu $\mathcal{M}'$ hinzugefügt, wenn die obere Kante mit Fahrtzeit m_i traversiert wird. Dann gilt:

$$\sum_{m \in \mathcal{M}'} m = a - 1 = \sum_{m \in \mathcal{M}} \frac{m}{2} = \sum_{m \in \mathcal{M} \setminus \mathcal{M}'} m$$

Dies ist eine gültige Lösung für Partition. Die konstruierte Partition-Instanz ist somit auch eine JA-Instanz.

Jetzt zeigen wir, dass eine gültige Lösung für das Partition-Problem eine Route mit Kosten k induziert. Betrachte eine gültige Lösung $\mathcal{M}' \subset \mathcal{M}$ von Partition. Dies induziert in $\mathcal{G}$ eine Route, auf der für alle $i \in \{1, \ldots, x\}$ die Kante zwischen v_i und v_{i+1} mit Reisezeit m_i genau dann gewählt wird, wenn $m_i \in \mathcal{M}'$. Die Kante (s, v_1) kann zum Zeitpunkt 0 traversiert werden. Die Fahrtzeit zum Knoten v_{x+1} beträgt somit $1 + \sum_{m \in \mathcal{M}'} m = 1 + \sum_{m \in \mathcal{M}} \frac{m}{2} = a$. Die Kante (v_{x+1}, d) kann somit traversiert werden und d wird zum Zeitpunkt $a + 1$ mit Kosten $(a+1) \cdot \delta \leq k$ erreicht. Dies ist eine gültige Lösung für RgsrzDec. Die konstruierte RgsrzDec-Instanz ist also ebenfalls eine JA-Instanz.

Das vorgestellte Entscheidungsproblem ist somit sowohl in $\mathcal{NP}$ als auch $\mathcal{NP}$-schwer, woraus die Behauptung folgt. □

5. Beschleunigungstechniken

In diesem Kapitel stellen wir die verwendeten Beschleunigungstechniken vor. Im ersten Abschnitt 5.1 werden Techniken zur Einschränkung des Suchraums und Reduzierung von Link- und Merge-Operationen eingeführt, die eine Verbesserung der Laufzeit nach sich ziehen. Im zweiten Abschnitt 5.2 wird kurz das Verfahren A^* rekapituliert, um dann genauer auf die verwendete *Potentialfunktion* und das Verfahren *ALT* einzugehen.

5.1 Pruning

Die folgende Methode beschleunigt die Berechnung der Pareto-Front, indem die Größe des Suchraums reduziert wird. Pfade, die keine optimale Lösung darstellen können, werden in diesem Fall nicht weiter untersucht.

Target-Pruning

Sei $v \in \mathcal{V}$ ein Knoten und t_v der Zeitpunkt, mit dem er in der Warteschlange hinterlegt ist. Wenn die Ungleichung

$$\forall t \in [t_v, t_{end} - \pi(v)] : C_v(t) + \pi(v) \cdot \delta > C_z(t + \pi(v))$$

erfüllt ist, muss v nicht weiter betrachtet werden, da keiner der Einträge in C_v ab dem Zeitpunkt t_v zur Pareto-Front beitragen kann. In diesem Fall wird v aus der Warteschlange entfernt, ohne seine ausgehenden Kanten zu relaxieren. Es ist jedoch möglich, dass C_v durch ein Update zu einem späteren Zeitpunkt wieder relevant wird.

Zusätzlich zur Reduzierung der besuchten Knoten ist es möglich den Algorithmus weiter zu beschleunigen, indem Kanten nicht relaxiert werden, die nicht zu einer optimalen Lösung führen können. Dies ist von Vorteil, da das Linken und Mergen von Profilen aufwendige Operationen sind. Im Folgenden stellen wir zwei Verfahren vor, welche die Überprüfung dieser Eigenschaft effizient umsetzen.

Pruning mit Schranken

Seien $upper(v) := \max\{C_v(t) \mid t \in \mathcal{H}, C_v(t) \neq \bot\}$ die obere und $lower(v) := \min\{C_v(t) \mid t \in \mathcal{H}, C_v(t) \neq \bot\}$ die untere Schranke des Kostenprofils eines Knotens $v \in \mathcal{V}$. Wenn ein Kostenprofil eines Knotens v leer ist, gilt $upper(v) = \infty$. Diese Schranken können mithilfe

eines Sweepline-Algorithmus bei der Berechnung des Kostenprofils erstellt werden. Die Kante $e = (u, v) \in \mathcal{E}$ muss nicht relaxiert werden, wenn folgende Bedingungen erfüllt sind:

$$lower(u) + d_e \cdot \delta > upper(v) \text{ und } \alpha(C_u) + d_e > \alpha(C_v)$$

Hierbei beschreibt $\alpha(C_v)$ den ersten Zeitpunkt, ab dem das Kostenprofil definiert ist.
In diesem Fall kann eine Relaxierung der Kante e keine Verbesserung des Kostenprofils C_v nach sich ziehen, da die Summe aus den Fahrtkosten und dem minimalen Kostenwert des Profils C_u größer ist als die maximalen Kosten des Profils C_v.

Hopping-Reduction

Sei $v \in \mathcal{V}$ ein Knoten. Betrachte die optimalen Routen, die zum Knoten v führen. Wenn der Vorgänger aller Routen übereinstimmt, nennen wir diesen Knoten $parent(v)$. Wenn der Knoten $parent(v)$ kein Parkplatz ist, kann die Kante $(v, parent(v))$ ignoriert werden. Dies ist ebenfalls mit einem Sweepline-Algorithmus über das Kostenprofil umsetzbar, indem jedem Segment ein Verweis auf seinen Vorgängerknoten hinzugefügt wird.

5.2 A*

Der A^*-Algorithmus ist eine Erweiterung des Dijkstra-Algorithmus, welche 1968 veröffentlicht wurde [HNR68]. Ziel dieses Algorithmus ist es den Suchraum zu verkleinern, indem die Reihenfolge angepasst wird, in der die Knoten besucht werden.
Um dies zu erreichen, wird jedem Knoten $v \in \mathcal{V}$ ein Potential $\pi_z(v) \in \mathbb{N}_0$ zugewiesen, das die Distanz, beziehungsweise in unserem Fall die Fahrtzeit, von diesem Knoten zum Ziel z abschätzt. Als Schlüssel der Warteschlange wird die Summe aus der tentativen Distanz zu diesem Knoten und dem Potential des Knotens verwendet. Eine Anforderung an eine Potentialfunktion ist hierbei, dass sie die Bedingung

$$\forall e = (u, v) \in \mathcal{E} : d_e - \pi_z(u) + \pi_z(v) \geq 0$$

erfüllt [HNR68]. Das Resultat ist eine zielgerichtete Suche, die das Finden eines ersten Kostenprofils am Zielknoten - je nach Güte der Abschätzung - beschleunigt. Mithilfe des Target-Pruning kann dann die Berechnung möglicherweise frühzeitig abgebrochen werden, was zu einer Reduktion der besuchten Knoten führt.
Sei $(u, v) \in \mathcal{E}$ eine Kante und C''_v das Kostenprofil, welches sich durch das Linken des Kostenprofils des Knotens u mit der Kante ergibt (siehe Kapitel 3). Wir verwenden in unserem Algorithmus anstelle der tentativen Distanz den Zeitpunkt t_{update}, an dem das erste Mal $C''_v(t_{update}) < C_v(t_{update})$ gilt. Wenn $t_{update} + \pi_z(v) > t_{end}$ gilt, wird der Knoten v nicht in die Warteschlange eingefügt, da das Ziel von v zum Zeitpunkt t_{update} ausgehend nicht mehr vor Ende des Planungshorizonts erreichbar ist.

Potentialfunktion

Die verwendete perfekte Potentialfunktion [SZ19] liefert die genaue Reisezeit zum Ziel auf dem Graphen ohne Sperrungen. Dies effizient zu berechnen wird durch den Einsatz von *Contraction Hierarchies* [GSSV12] ermöglicht, welche die inhärente Hierarchie eines Straßengraphen ausnutzen. Jedem Knoten wird eine Priorität zugewiesen, dann werden die Knoten nach aufsteigender Priorität kontrahiert. Dies erfolgt, indem der zu kontrahierende Knoten aus dem Graphen entfernt wird und möglicherweise Abkürzungskanten zwischen Nachbarknoten hinzugefügt werden um kürzeste Distanzen zu erhalten. Die genaue Konstruktion von und das Routen auf *Contraction Hierarchies* wird in [GSSV12] beschrieben. Angelehnt an dieses Paper teilen wir den resultierenden Graphen in einen Aufwärtsgraphen $\mathcal{G}_\uparrow = (\mathcal{V}, \mathcal{E}_\uparrow)$ mit $E_\uparrow = \{(u, v) \in \mathcal{E} : u < v\}$ und einen Abwärtsgraphen $\mathcal{G}_\downarrow = (\mathcal{V}, \mathcal{E}_\downarrow)$ mit

Algorithm 5.1: Berechnung des Potentials

```
1 Function Potential(Node v)
2     if v ∉ S then
3         forall e = (v, w) ∈ E↑ do
4             if Π[v] > d_e + Potential(w) then
5                 Π[v] = d_e + Potential(w)
6         S = S ∪ {v}
7     return Π[v]
```

$E_{\downarrow} = \{(u, v) \in \mathcal{E} : u > v\}$. Hierbei wird für die Vergleichsoperation die Priorität der Knoten ausgewertet. Wir verwenden die Implementierung von *Contraction Hierarchies* von [Str19].

Die vorberechnete *Contraction Hierarchy* wird genutzt, indem bei jeder Anfrage initial der Rückwärtssuchraum von z auf $\mathcal{G}_{\downarrow}$ durchlaufen wird und die minimale Distanz jedes Knotens in diesem Suchraum zu z im Potential-Array Π gespeichert wird. Dies entspricht einer Dijkstra-Anfrage auf dem Rückwärtsgraphen $\mathcal{G}_{\downarrow}^{R}$ mit invertierten Kanten und Startknoten z. Für alle anderen Knoten v gilt $\Pi[v] = +\infty$. Wir setzen $S = \emptyset$, wobei S eine Menge von Knoten ist.
Nach dieser Initialisierung kann A^* mit der in Algorithmus 5.1 beschriebenen rekursiven Funktion zur Berechnung des Potentials ausgeführt werden. Wenn ein Knoten $v \in \mathcal{V}$ bereits in S enthalten ist, ist die minimale Distanz von v zum Ziel bereits berechnet und wird zurückgegeben. Wenn dies nicht der Fall ist, werden alle von v im Aufwärtsgraphen $G_{\uparrow}$ ausgehenden Kanten $e = (v, w) \in \mathcal{E}_{\uparrow}$ relaxiert, indem die Potentialfunktion auf w aufgerufen wird. Wenn der Pfad via w eine Verbesserung der Distanz zu z darstellt, wird das Potential $\Pi[v]$ des Knoten aktualisiert. Anschließend wird v zu S hinzugefügt und die berechnete Distanz zurückgegeben.

ALT

Der ALT-Algorithmus [GH05] stellt eine weitere Möglichkeit dar, Potential für den A^*-Algorithmus zu berechnen. Die Knotenpotentiale werden hierbei anhand von Landmarken $L \subseteq \mathcal{V}$ und unter Ausnutzung der Dreiecks-Ungleichung berechnet. Die Landmarken werden hierbei in einem Vorberechnungsschritt ausgewählt und die Distanzen von und zu allen Landmarken für jeden Knoten gespeichert. Aufgrund dieses großen Speicher-Overheads ist die Menge der Landmarken klein, meist gilt $|L| \leq 32$. Sei $v \in \mathcal{V}$ ein Knoten und $l \in L$ eine Landmarke. Wir schreiben $d_{(v,l)}$ beziehungsweise $d_{(l,v)}$ für die minimale Distanz von v zu l beziehungsweise von l zu v.
Das Potential des Knotens v gegeben einen Zielknoten $z \in \mathcal{V}$ berechnet sich aus

$$\pi_z(v) = \max_{l \in L}\{\max\{d_{(l,z)} - d_{(l,v)}, d_{(v,l)} - d_{(z,l)}\}\}.$$

Die Güte der Potentialfunktion hängt somit stark mit der Wahl der Landmarken zusammen. Diese sollten am Rande des Graphen liegen um eine gute Approximation der Distanzen zu gewährleisten. In [GW05] sind verschiedene Heuristiken zur Wahl der Landmarken vorgestellt. Die von uns verwendeten Landmarken sind mit dem Avoid-Algorithmus berechnet. Hierbei wird in jeder Iteration eine Landmarke in einem Gebiet des Graphen hinzugefügt, welches noch nicht ausreichend von existierenden Landmarken abgedeckt ist.

6. Experimentelle Evaluation

In diesem Kapitel evaluieren wir die Performance und Lösungsqualität des vorgestellten Algorithmus bei verschiedenen Parameterbelegungen experimentell.
In Abschnitt 6.1 stellen wir die Rahmenbedingungen der verwendeten Experimente vor. Mit diesen Experimenten werten wir in Abschnitt 6.2 dann die Laufzeit des Algorithmus bei Variation der Parameter aus. Schließlich untersuchen wir in Abschnitt 6.3, welchen Einfluss unterschiedliche Kostenbelegungen und Parkplatzkategorisierungen auf die gefundenen Lösungen ausüben.

6.1 Versuchsaufbau

Eine Anfrage auf einem gegebenen Graphen zeichnet sich durch einen Start- und Zielknoten, den Planungshorizont, die Zuweisung der Parkplätze auf Parkplatzkategorien und die Kostenbelegung für das Fahren sowie das Warten aus.
In diesem Abschnitt stellen wir das Straßennetzwerk und die evaluierten Belegungen für diese Parameter vor. Eine Zeiteinheit entspricht einer Sekunde.

Straßennetzwerk

Der auf TomTom-Daten basierende verwendete Graph modelliert das Straßennetzwerk Deutschlands, Frankreichs, Italiens, Liechtensteins, Luxemburgs, Österreichs und der Schweiz und wurde von der PTV AG zur Verfügung gestellt. Der Graph besteht aus 30.069.809 Knoten - von denen 15.315 Parkplätze sind - und 47.635.384 Kanten. Die Anzahl der Knoten verringert sich nach der Entfernung von Knoten ohne ein- und ausgehende Kanten auf 21.903.924. In Abbildung 6.1 ist ein Histogramm der Parkplatzstellplätze dargestellt, das angibt, wie viele Parkplätze eine gegebene Anzahl an Stellplätzen zur Verfügung stellen.
Die Fahrverbote an Wochenenden und nachts für Sattelzüge mit einem Gesamtgewicht von 40 Tonnen sind in Tabelle 6.1 dargestellt. Alle Zeitangaben beziehen sich auf die Mitteleuropäische Sommerzeit.

Ort und Planungshorizont

Um eine Aussage über die Performance und Qualität des vorgestellten Algorithmus treffen zu können, ist es wichtig, dass die kürzeste Route einer Anfrage durch ein gesperrtes Gebiet führt und somit die Suche nach Alternativrouten angestoßen wird. Wir erreichen

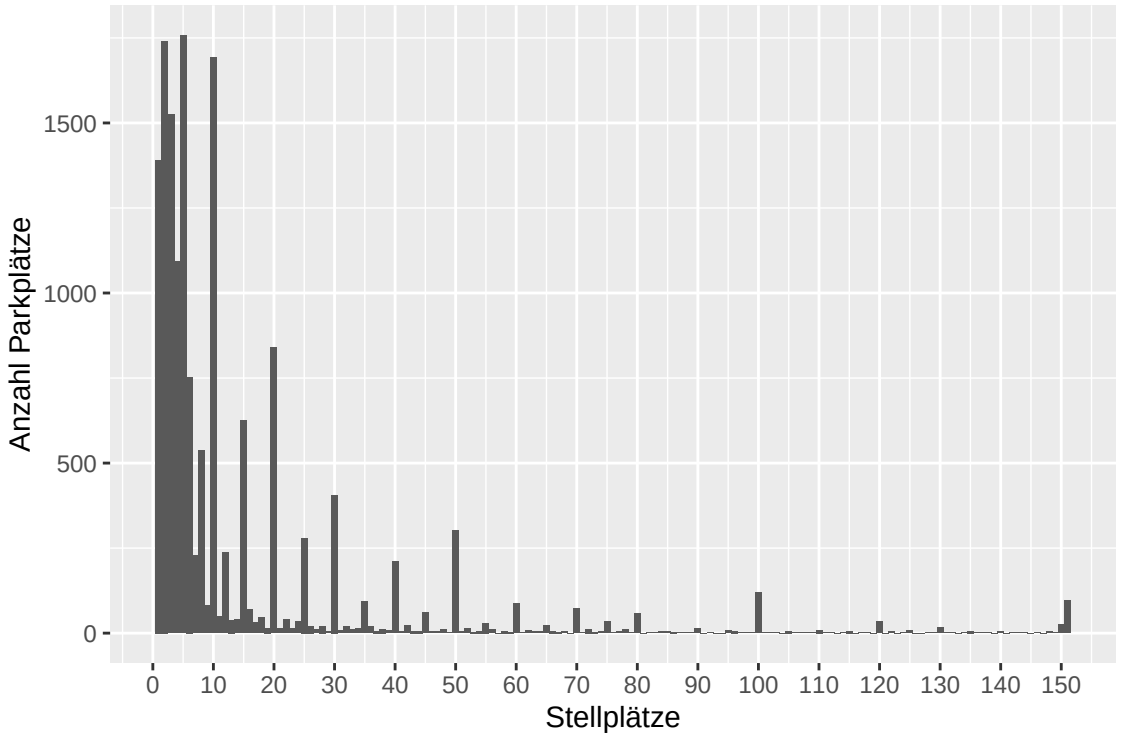

Abbildung 6.1: Histogramm der Parkplatzstellplätze. Zur Übersicht sind alle Parkplätze mit mehr als 150 Stellplätzen in der letzten Klasse repräsentiert.

dies, indem wir das Nachtfahrverbot der Schweiz und Österreichs ausnutzen und Anfragen generieren, deren Routen durch diese beiden Länder führen.
Die Anfragen, bestehend aus Start- und Zielknoten sowie dem Beginn des Planungshorizonts, sind in vier Testmengen aufgeteilt. In den ersten drei Testmengen untersuchen wir den Effekt, den eine Veränderung des Planungshorizonts auf die Laufzeit und Qualität der Lösung hat. Deshalb stimmen die Knotenpaare dieser Mengen überein. Die vierte Testmenge enthält Knotenpaare, die eine Durchquerung der Schweiz und Österreichs forcieren. Betrachte Abbildung 6.2, welche die beiden Gebiete darstellt, in denen jeweils einer der beiden Knoten der Anfragen dieser Menge liegen.
Die Testmengen 1 bis 3 enthalten jeweils 100 Anfragen für die Dijkstra-Ranks 2^i für alle $i \in \{12, \ldots, 24\}$. Testmenge 4 enthält 200 Anfragen, 2 mit Dijkstra-Rank 2^{20}, 9 mit Dijkstra-Rank 2^{21}, 65 mit Dijkstra-Rank 2^{22}, 97 mit Dijkstra-Rank 2^{23} und 27 Anfragen mit Dijkstra-Rank 2^{24}.
Ein Überblick über die Startzeitpunkte ist in Tabelle 6.2 gegeben. Bei Testmenge 2 und 3 ist zu beachten, dass in Italien aufgrund des erhöhten Verkehrsaufkommen in den Sommermonaten am Samstag, den 7. Juli 2018, ein Fahrverbot von 08:00-16:00 Uhr besteht. Wir betrachten Planungshorizonte von einem und zwei Tagen.
Dieser Versuchsaufbau wurde von [Brä18] übernommen. Die untersuchten Planungshorizonte unterscheiden sich um eine Stunde, da wir die lokale Zeitzone und nicht UTC+1 betrachten.

Einteilung in Parkplatzkategorien

Wir betrachten drei Kategorisierungen von Parkplätzen. In der ersten Kategorisierung unterscheiden wir nicht zwischen unterschiedlichen Parkplätzen, allen Parkplätzen wird die gleiche Kategorie zugewiesen.
In den beiden anderen Kategorisierungen weisen wir Parkplätzen ihre Kategorie aufgrund

Abbildung 6.2: Alle Start- und Zielknotenpaare in der Testmenge 4 befinden sich in den Gebieten A und B, jeweils einer der beiden Knoten in einem Gebiet. A erstreckt sich von 49° N 4° E an der linken oberen Ecke bis zu 47° N 18° E in der rechten unteren Ecke. Die nordwestliche Spitze des Gebiets B liegt bei 46° N 4° E, das südöstliche Ende ist an 42° N 18° E (Quelle: [Brä18]).

Tabelle 6.1: Überblick der Wochenend- und Nachtfahrverbote für Sattelzüge mit 40 Tonnen Gesamtgewicht in 2018 (Quelle: [Brä18]).

Land	Samstagsfahrverbot	Sonntagsfahrverbot	Nachtfahrverbot
Österreich[1]	15.00 – 24.00	00.00 – 22.00	22.00 – 05.00
Frankreich[2]	22.00 – 24.00	00.00 – 22.00	–
Deutschland[3]	–	00.00 – 22.00	–
Italien[4]	–	07.00 – 22.00 (Jun. – Sep.) 09.00 – 22.00 (Okt. – Mai)	–
Liechtenstein[5]	–	00.00 – 24.00	22.00 – 05.00
Luxemburg[6]	21.30 – 24.00 oder[7] 23.30 – 24.00	00.00 – 21.45	–
Schweiz[8]	–	00.00 – 24.00	22.00 – 05.00

[1] Österreichische Straßenverkehrsordnung: StVO §42(1)+(6), 12. Juli 2018
[2] Amtsblatt der Republik Frankreich: Journal officiel de la République française: n°0302 du 28 décembre 2017, texte n°120
[3] Deutsche Straßenverkehrsordnung: StVO §30(3), 06. Oktober 2017
[4] Ministerium für Infrastruktur und Verkehr Italien: Decreto Ministeriale numero 571 del 19-12-2017, Art. 1, 1a)+b)
[5] Liechtensteinische Straßenverkehrsordnung: VRV Art. 89 1)+2), 01. Juli 2018
[6] Luxemburgische Straßenverkehrsordnung: Réglement grand-ducal modifié du 19 juillet 1997 relatif aux limitations de la circulation des poids lourds pendant les dimanches et jours fériés Art. 1
[7] 21.30 – 24.00 Richtung Frankreich oder 23.30 – 24.00 Richtung Deutschland
[8] Schweizer Straßenverkehrsordnung: SVG Art. 2(2), 01. Januar 2018

Tabelle 6.2: Früheste Abfahrtszeit für jede Testmenge. Diese sind angelehnt an [Brä18], die Planungshorizonte beginnen jedoch eine Stunde später.

Testmenge	Startzeit
1	Mo, 2. Juli 2018, 06.00
2	Fr, 6. Juli 2018, 06.00
3	Fr, 6. Juli 2018, 18.00
4	Mo, 2. Juli 2018, 18.00

der Anzahl ihrer Stellplätze zu. Wir nehmen an, dass mit einem größeren Parkplatz meist ein erhöhter Komfort für die Fahrer einher geht, zum Beispiel durch Duschen oder Restaurants. In Kategorisierung 2 unterteilen wir die Parkplätze in 5 Kategorien. Ein Überblick über die Einteilung sowie die Anzahl von Parkplätzen pro Kategorie ist in Tabelle 6.3 gegeben. Kategorisierung 3 umfasst 500 Kategorien. Hierbei stimmt die Kategorie für alle Parkplätze mit weniger als 500 Stellplätzen mit der Anzahl der Stellplätze überein. Die 10 Parkplätze mit mehr als 500 Stellplätzen werden in der Kategorie 500 zusammengefasst.

Kostenbelegung

Wir belegen das Fahren bei Kategorisierung 1 mit Kosten 100 und das Warten an Parkplätzen mit Kosten 1. Im Falle von Kategorisierung 2 unterscheiden wir zwischen drei Belegungen:

- linear: Die Kosten für das Warten an Parkplätzen geringerer Qualität steigen linear mit der Qualität. Die Kosten sind so gewählt, dass ein Umweg von 4 Minuten bei einer Stunde Wartezeit für einen Wechsel zu einem um eine Kategorie besseren Parkplatz in Kauf genommen wird.

Tabelle 6.3: Zuweisung der Parkplätze in Kategorisierung 2

Kategorie	1	2	3	4	5
Anzahl Stellplätze	< 5	< 15	< 40	< 80	≥ 80
Anzahl Parkplätze absolut	5748	5418	2664	997	488
Anzahl Parkplätze prozentual	37,5	35,4	17,4	6,5	3,2

Tabelle 6.4: Überblick über die Kostenbelegungen der Parkplatzkategorisierung 2.

Kostenbelegung	$\delta = \gamma_{\text{NP}}$	γ_{p_1}	γ_{p_2}	γ_{p_3}	γ_{p_4}	γ_{p_5}	γ_s
linear	14	7	6	5	4	3	0
exponentiell	64	16	8	4	2	1	0
logarithmisch	32	31	30	28	24	16	0

- exponentiell: Die Kosten für das Warten an Parkplätzen geringerer Qualität steigen exponentiell mit der Qualität. Dies hat zur Folge, dass das Warten an einem Parkplatz zwar priorisiert wird, die Qualität des Parkplatzes jedoch eine untergeordnete Rolle spielt.

- logarithmisch: Die Zunahme der Kosten für das Warten an Parkplätzen geringerer Qualität ist logarithmisch. In diesem Fall wird das Warten an Parkplätzen höherer Qualität oder am Start priorisiert. Wie in Abschnitt 3.2 gezeigt, kann die geringe Differenz von Fahrt- und Wartekosten an Parkplätzen niedriger Qualität jedoch vermehrt die Pareto-Optimalität von Routen mit Warten auf Nicht-Parkplätzen nach sich ziehen.

In Tabelle 6.4 ist ein Überblick über die Kostenbelegungen der Kategorisierung 2 gegeben. Die Tabellen 6.5-6.7 zeigen für alle Belegungen die maximale zusätzliche Fahrtzeit in Minuten an, die für das Warten an einem Parkplatz höherer Qualität bei einer Stunde Wartezeit in Kauf genommen wird. Sie geben somit die maximale Fahrtzeit an, sodass die Summe aus Wartekosten an dem besseren Parkplatz und Fahrtkosten kleiner oder gleich den Wartekosten am schlechter bewerteten Parkplatz sind.
In der ersten Spalte sind die Parkplatzkategorien abgebildet, auf denen gewartet wird, hierbei steht NP für Nicht-Parkplätze. In der ersten Reihe sind die Parkplatzkategorien und der Startknoten repräsentiert, zu denen gewechselt wird.

Für Kategorisierung 3 setzen wir $\delta = 1000$ und $\gamma_{p_i} = 501 - i$ für einen Parkplatz der Kategorie $i \in \{1, \ldots, 500\}$.

6.2 Laufzeit

In diesem Abschnitt untersuchen wir die Laufzeit des Algorithmus, insbesondere welchen Einfluss eine Änderung des Planungshorizonts, der Parkplatzkategorisierung und der Kostenbelegung auf diese haben. Wir analysieren jedoch zuerst die Beschleunigung durch A^* und ALT (siehe Abschnitt 5.2) und vergleichen unsere Performance mit Bräuer [Brä18].

Der Algorithmus ist in C++ 14 implementiert und mit vc14 kompiliert. Die Durchführung der Experimente erfolgte auf einem Rechner mit einem 4 Kern Intel i7-7600 Prozessor mit einer Taktrate von 3,4GHz und 32 GB DDR4 Arbeitsspeicher. Das verwendete Betriebssystem ist Windows 10 Pro. Der Algorithmus hat keine parallelen Komponenten.

Tabelle 6.5: Zusätzliche Fahrtzeit in Minuten pro Stunde Warten bei der linearen Kostenbelegung.

	p_1	p_2	p_3	p_4	p_5	s
NP	30	34,28	38,57	42,85	47,14	60
p_1		4,28	8,57	12,85	17,14	30
p_2			4,28	8,57	12,85	25,71
p_3				4,28	8,57	21,42
p_4					4,28	17,14
p_5						12,85

Tabelle 6.6: Zusätzliche Fahrtzeit in Minuten pro Stunde Warten bei der exponentiellen Kostenbelegung.

	p_1	p_2	p_3	p_4	p_5	s
NP	45	52,5	56,25	58,12	59,06	60
p_1		7,5	11,25	13,12	14,06	15
p_2			3,75	5,62	6,56	7,5
p_3				1,87	2,81	3,75
p_4					0,93	1,87
p_5						0,93

Tabelle 6.7: Zusätzliche Fahrtzeit in Minuten pro Stunde Warten bei der logarithmischen Kostenbelegung.

	p_1	p_2	p_3	p_4	p_5	s
NP	1,87	3,75	7,5	15	30	60
p_1		1,87	5,62	13,12	28,12	58,12
p_2			3,75	11,25	26,25	56,25
p_3				7,5	22,5	52,5
p_4					15	45
p_5						30

Tabelle 6.8: Vergleich der Dauer der Vorberechnung, der durchschnittlichen Berechnungszeit einer Anfrage der Testmenge 4 und des zusätzlichen Speicherverbrauchs ohne Verwendung einer Potentialfunktion, mit A^* mit perfektem Potential und mit ALT. Die Länge des Planungshorizonts beträgt einen Tag und wir betrachten die lineare Kostenbelegung.

Potentialfunktion	Vorberechnung [mm:ss]	∅ Laufzeit	Speicher [MB]
Keine	00:00	16,9 s	0
Perfekt (CH)	07:48	529 ms	1587
ALT	23:40	1,14 s	4198

Evaluation der Beschleunigungstechniken

In Tabelle 6.8 ist eine Übersicht über die Performance der drei Potentialfunktionen gegeben. Wir verwenden ALT mit 16 Landmarken, die alle aktiv sind.
Bei einem Speicher-Overhead von knapp 4,2 GB beschleunigt ALT die Berechnung um mehr als eine Größenordnung verglichen mit der Laufzeit ohne A^*. Die auf *Contraction Hierarchies* basierende perfekte Potentialfunktion weist jedoch eine weitere Beschleunigung um den Faktor 2 bei geringerem Speicherverbrauch und schnellerer Vorberechnung auf, weshalb wir im Folgenden diese verwenden.

Vergleich mit Bräuer [Brä18]

Wir vergleichen die Laufzeit von den in [Brä18] vorgestellten Algorithmen mit der Laufzeit des in dieser Arbeit vorgestellten Algorithmus bei Verwendung der Testmenge 4 mit einer Länge des Planungshorizonts von einem Tag und der linearen Kostenbelegung.
Die durchschnittliche Laufzeit einer Anfrage unseres Algorithmus liegt bei circa 529 ms. Die Berechnung Bräuers wird in 93 der 200 Anfragen nach einer Laufzeit einer halben Stunde abgebrochen. Die durchschnittliche Laufzeit der erfolgreich berechneten Anfragen liegt bei knapp 97 Sekunden, mindestens 2 Größenordnungen langsamer als unsere Berechnung.

Variation des Planungshorizonts

Wir vergleichen die Laufzeiten der Testmengen unter Verwendung der Parkplatzkategorisierung 2 und der linearen Kostenbelegung, um den Einfluss von unterschiedlichen Planungshorizonten auf die Performance des Algorithmus zu untersuchen.

In Abbildung 6.3 sind die Laufzeiten der Testmengen 1 bis 3 bei einer Dauer des Planungshorizonts von einem Tag auf einer logarithmischen Skala dargestellt. Die drei Ausreißer mit einer Laufzeit von circa einer Sekunde bei Dijkstra-Rank 2^{15} sind auf eine Anfrage zurückzuführen, deren Ziel in einem gesperrten Gebiet liegt, weshalb ein Großteil des Graphen erfolglos abgesucht wird.
Der sprunghafte Anstieg der Laufzeit der Anfragen in Testmenge 3 ab einem Dijkstra-Rank von 2^{23}, sowie der Ausreißer in dieser Testmenge bei geringerem Rank und in den anderen beiden Testmengen ist dem Nachtfahrverbot in der Schweiz und Österreich zuzuschreiben. Diese Sperrung hat zur Folge, dass sich aufgrund der Suche nach Alternativrouten der Suchradius des Algorithmus stark vergrößert und die Anzahl der Segmente der Kostenprofile steigt. Die Testmenge 3 ist hiervon wegen der späteren Abfahrtszeit besonders stark betroffen. Die Ankunftszeit der Anfragen geringerer Länge und der anderen Testmengen liegen meist vor 22:00 Uhr, weshalb dieses Verbot keinen Einfluss auf sie hat. Ein weiterer Faktor, der die Berechnung der Routen der Testmenge 3 verlangsamt, ist das Samstagsfahrverbot in Italien an diesem Wochenende, weshalb einige der Ziele nicht innerhalb eines Tages erreichbar sind.

Die niedrigen Laufzeiten der Ausreißer bei den Dijkstra-Ranks von 2^{22} und höher treten auf, da die Fahrtzeiten auf dem Graphen ohne Sperrungen vom Start- zum Zielknoten dieser Anfragen die Länge des Planungshorizonts überschreiten. In diesem Fall reduziert sich die Laufzeit auf die Berechnung des Potentials des Startknotens.

In Abbildung 6.4 sind die Laufzeiten der Testmengen 3 und 4 bei einer Dauer des Planungshorizonts von einem und zwei Tagen für die Dijkstra-Ranks 2^i für alle $i \in \{21, \ldots, 24\}$ visualisiert.
Im Falle der vierten Testmengen ist nur eine Steigerung der Laufzeit von durchschnittlich $22,5\%$ bei einer Verdopplung der Länge des Planungshorizonts zu erkennen. Die Pareto-Fronten der Anfragen dieser Menge beinhalten bei einem Tag Planungshorizont meist bereits die optimale Route, die dem kürzesten Pfad auf dem Graphen ohne Sperrungen entspricht und auf welcher nicht oder nur am Start gewartet wird. Mithilfe des Target-Pruning (siehe 5.1) wird gewährleistet, dass in diesem Fall der Suchraum nicht ausgeweitet wird. Die Berechnung dieser Anfragen ist häufig dennoch langsamer, da mit der Länge des Planungshorizonts auch die Anzahl der Sperrungen und somit die Anzahl der Segmente der Kostenprofile steigen.
Das gleiche Verhalten ist bei der Testmenge 3 für Anfragen mit einem Dijkstra-Rank kleiner als 2^{24} zu beobachten. Mit zunehmender Länge des Pfades auf dem Graphen ohne Sperrungen steigt jedoch die Wahrscheinlichkeit, dass das Ziel aufgrund von Sperrungen nicht erreichbar ist. Dies ist besonders bei diesen Anfragen erkennbar, da zusätzlich zu den Nachtfahrverboten der Schweiz und Österreich die Samstagsfahrverbote Italiens und die Sonntagsfahrverbote allgemein zum Tragen kommen.
Wenn das Ziel nicht erreichbar ist, kann der Suchraum nicht eingeschränkt werden. Wie auch bei dem Dijkstra-Algorithmus kann dieser dann als Kreis mit dem Startknoten als Mittelpunkt interpretiert werden, weshalb der Suchraum quadratisch mit zunehmender Länge des Planungshorizonts wächst.
Auffällig ist dies insbesondere bei der langsamsten Anfrage, deren Berechnung circa 136 Sekunden benötigt, 5 Größenordnungen langsamer als der Median der Anfragen mit Dijkstra-Rank 2^{23} von 3,3 ms. Der Startknoten dieser Anfrage liegt bei Ingolstadt, das Ziel im Stadtkern von Belluno im Nordosten Italiens. Dieses ist im gegebenen Zeitraum nicht erreichbar. Infolge des zentralen Startknotens kann fast jeder Knoten des Graphen in zwei Tagen erreicht werden, so werden mehr als 20,5 Millionen oder $93,7\%$ der Knoten exploriert. Hierbei wird jeder Knoten durchschnittlich 4,6-mal aus der Warteschlange genommen. Die Kostenprofile der Knoten bestehen aus durchschnittlich 11,8 Segmenten, die durchschnittliche Anzahl Segmente aller Anfragen dieses Testmenge mit einem Dijkstra-Rank von 2^{23} liegt bei 6,2.

Die Wahl des Planungshorizont hat also einen großen Einfluss auf die Laufzeit des Algorithmus. Wenn aufgrund von Sperrungen die Ankunft am Ziel erst zu einem späten Zeitpunkt oder in einem gegebenen Zeitraum überhaupt nicht möglich ist, fällt die Performance dieses Algorithmus massiv ab, da der Suchraum nicht ausreichend eingeschränkt werden kann.

Variation der Parkplatzkategorisierung

Wir analysieren in diesem Absatz den Einfluss unterschiedlicher Parkplatzkategorisierungen auf die Laufzeit des Algorithmus. Hierzu werten wir die Performance der Testmenge 4 bei Verwendung der drei Kategorisierungen (siehe 6.1) und einem Planungshorizont von einem Tag aus. Bei der Aufteilung in 5 Kategorien verwenden wir die lineare Kostenbelegung.

Die durchschnittliche Laufzeit des Algorithmus bei der Parkplatzkategorisierung ohne Differenzierung von Parkplätzen liegt mit durchschnittlich 349 ms pro Anfrage unter den Laufzeiten der anderen beiden Kategorisierungen mit 529 ms bei der Aufteilung in 5 Klassen und 651 ms im Falle der 500 Parkplatzkategorien. Dies ist darauf zurückzuführen, dass in der

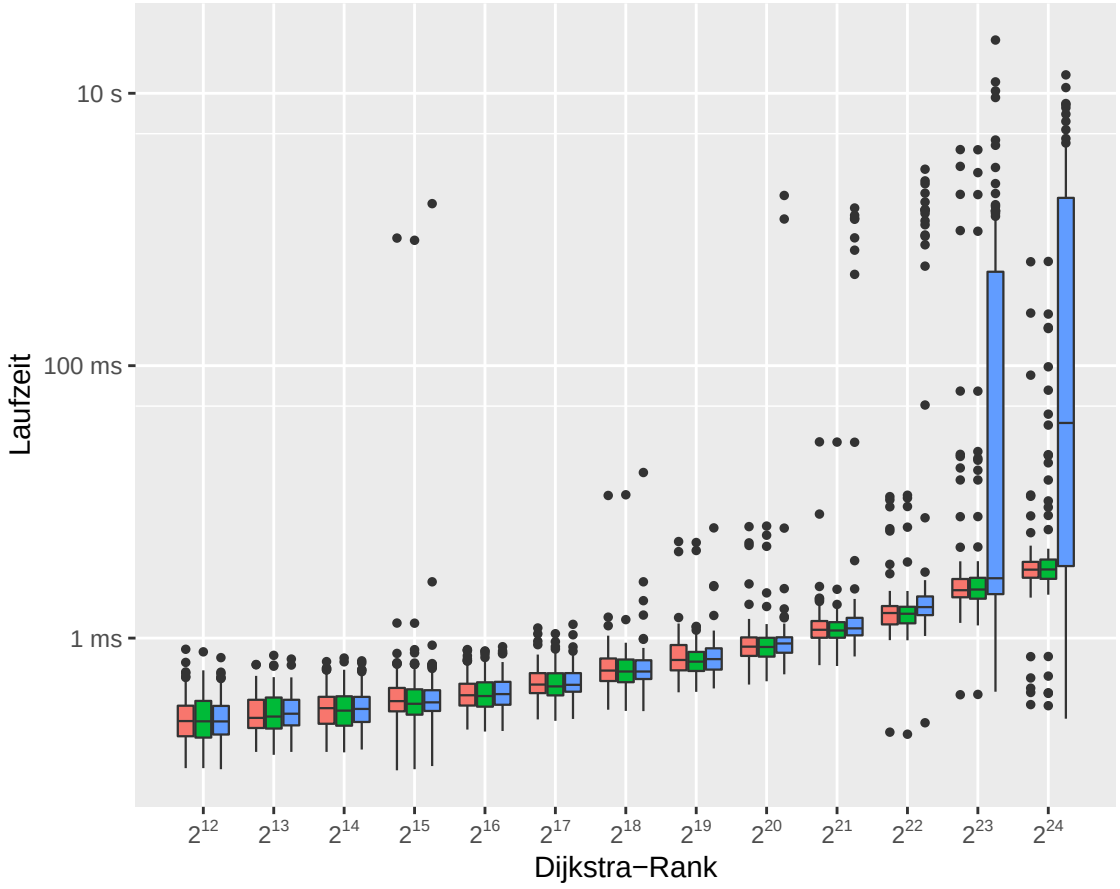

Abbildung 6.3: Vergleich der Laufzeit der Testmengen 1-3. Testmenge 1 ist in rot, 2 in grün und die Testmenge 3 in blau dargestellt.

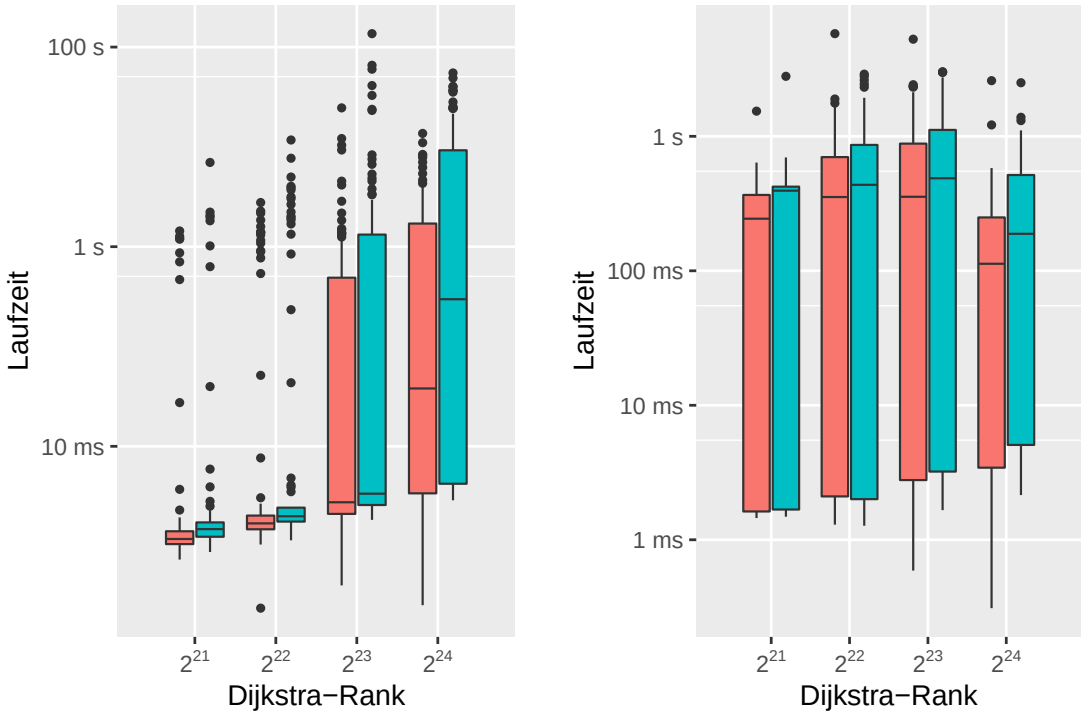

Abbildung 6.4: Vergleich der Laufzeiten bei einer Dauer des Planungshorizonts von einem (in rot) beziehungsweise zwei (in türkis) Tagen. Das linke Diagramm vergleicht die Laufzeiten der Testmenge 3, das rechte die von Testmenge 4. Beachte die unterschiedliche Skalierung der y-Achsen.

ersten Kategorisierungen bei Sperrungen einfach der nächstgelegene Parkplatz angefahren wird, während in den anderen beiden Fällen möglicherweise Umwege untersucht werden, um mit geringeren Kosten warten zu können. Dies spiegelt sich auch in den gesammelten Daten wider. Im ersten Fall werden durchschnittlich knapp $8 \cdot 10^5$ Knoten aus der Warteschlange genommen, bei den anderen Kategorisierungen beläuft sich diese Zahl auf über $1,1 \cdot 10^6$. Aufgrund der 500 Parkplatzkategorien kommt es bei der dritten Kategorisierung zu vielen Schnitten von Segmenten, welche vergleichsweise langsam zu berechnen sind. Zusammen mit einer leicht erhöhten durchschnittlichen Anzahl von Segmenten pro Kostenprofil - 2,94 im Vergleich zu 2,61 bei 5 Kategorien - erklärt dies die schlechtere Laufzeit dieser Kategorisierung.

Die Anzahl der Parkplatzkategorien hat somit einen Einfluss auf die Laufzeit des Algorithmus. Dieser ist jedoch verglichen mit dem Einfluss der Wahl des Planungshorizonts sehr gering.

Variation der Kostenbelegung

In diesem Absatz untersuchen wir, wie unterschiedliche Kostenbelegungen die Laufzeit des Algorithmus beeinflussen. Wir verwenden bei allen Tests die Testmenge 4 bei einer Länge des Planungshorizonts von einem Tag und der Unterteilung der Parkplätze in 5 Kategorien. Wir analysieren die drei in 6.1 vorgestellten Kostenbelegungen.

In Tabelle 6.9 ist für die drei Kostenbelegungen eine Übersicht über die durchschnittliche Anzahl der Segmente eines Kostenprofils gegeben. Darüber hinaus zeigt sie an, wie viele Knoten durchschnittlich aus der Warteschlange genommen wurden, die durchschnittliche Anzahl besuchter Knoten und die durchschnittliche Laufzeit der Berechnung.
Die kürzere Laufzeit der exponentiellen Belegung ergibt sich aus der niedrigen Priorität, die

Tabelle 6.9: Vergleich der durchschnittlichen Anzahl der Segmente eines Kostenprofils, der DeleteMin-Operationen, der explorierten Knoten und der durchschnittlichen Laufzeit für die drei Kostenbelegungen.

Belegung	# Segmente	del. min Ops. [10^3]	expl. Knoten [10^3]	Laufzeit [ms]
linear	2,61	1.110	818	529
exponentiell	2,27	885	761	418
logarithmisch	3,02	1.201	844	742

Parkplätze höherer Qualität bei dieser Belegung haben. So werden nur sehr kurze Umwege für diese Parkplätze in Kauf genommen, da die Untersuchung längerer, nicht optimaler Wege durch das Target-Pruning unterbunden wird. Dementsprechend ergibt sich mit der stärkeren Priorisierung besserer Parkplätze bei linearen Kosten eine schlechtere Laufzeit und ein größerer Suchraum. Auch wächst die Größe der Kostenprofile und diese werden häufiger korrigiert, was durch eine Erhöhung der DeleteMin-Operationen signalisiert wird. Dieser Trend setzt sich bei der logarithmischen Kostenbelegung fort, weshalb diese mit 742 ms die schlechteste durchschnittliche Laufzeit aufweist. Eine Variation der Kostenbelegung kann somit zu einer Veränderung der Performance führen. Die Differenz der Laufzeiten unterschiedlicher Belegungen bewegt sich hierbei in einem vergleichbaren Bereich wie die, welche bei der Veränderung der Parkplatzzuteilung gemessen wird.

6.3 Lösungsqualität

Wir betrachten beispielhaft eine Anfrage der Testmenge 4 mit der linearen Kostenbelegung und eine Dauer des Planungshorizonts von einem Tag. Der Start liegt im Nordwesten Österreichs, das Ziel befindet sich südlich von Mailand in Italien. Die Pareto-Front dieser Anfrage besteht aus 8 Routen, aus Gründen der Übersicht betrachten wir jedoch nur drei davon. Diese sind in der Abbildung 6.5 visualisiert. Die Tabelle 6.10 enthält die Kenndaten dieser Routen.
Der Fahrer startet auf der grünen und schwarzen Route fast zeitgleich und verlässt Österreich Richtung Deutschland um das Nachtfahrverbot zu vermeiden. Auf der grünen Route fährt er südlich von Rosenheim von der Autobahn ab, um auf einem Parkplatz der Kategorie 5 zu warten. Die beiden Routen trennen sich nördlich von Bregenz. Auf der schwarzen Route fährt der Fahrer einen großen Umweg, um die Fahrtzeit in der Schweiz zu verringern. Auf beiden Routen wird die Grenze zu Österreich beziehungsweise der Schweiz zum Ende des Fahrverbotes um 05:00 Uhr überschritten. Sie treffen nördlich von Bellinzona kurz vor der italienischen Grenze wieder aufeinander. Aufgrund der verringerten Fahrtzeit durch die Schweiz erreicht der Fahrer auf der schwarzen Route zuerst das Ziel, muss dafür jedoch gute 3 Stunden länger fahren.
Auf der blauen Route startet der Fahrer 10 Minuten früher, fährt jedoch - abgesehen von einem Umweg einer Minute zu einem Parkplatz der Kategorie 4 - den direkten Weg zum Ziel. Da er jedoch Österreich nicht verlässt, muss er das 7-stündige Fahrverbot an diesem Parkplatz abwarten. Deshalb erreicht der Fahrer auf dieser Route das Ziel eine gute Stunde später, aber mit einer deutlich kürzeren Fahrtzeit als auf den anderen beiden Routen.

Wir verwenden die Testmenge 4 mit einer Länge des Planungshorizonts von einem Tag, um die Veränderung der Pareto-Front einer Anfrage bei einer Variation der Kostenbelegung beziehungsweise Parkplatzkategorisierung zu untersuchen.
Bei allen betrachteten Belegungen existieren 127 identische Routen, auf denen nie, auch nicht am Startknoten, gewartet wird. Ebenfalls stimmen 13 Routen überein, die nicht gültig sind, da der Fahrer auf ihnen an einem Nicht-Parkplatz wartet. Diese ungültigen

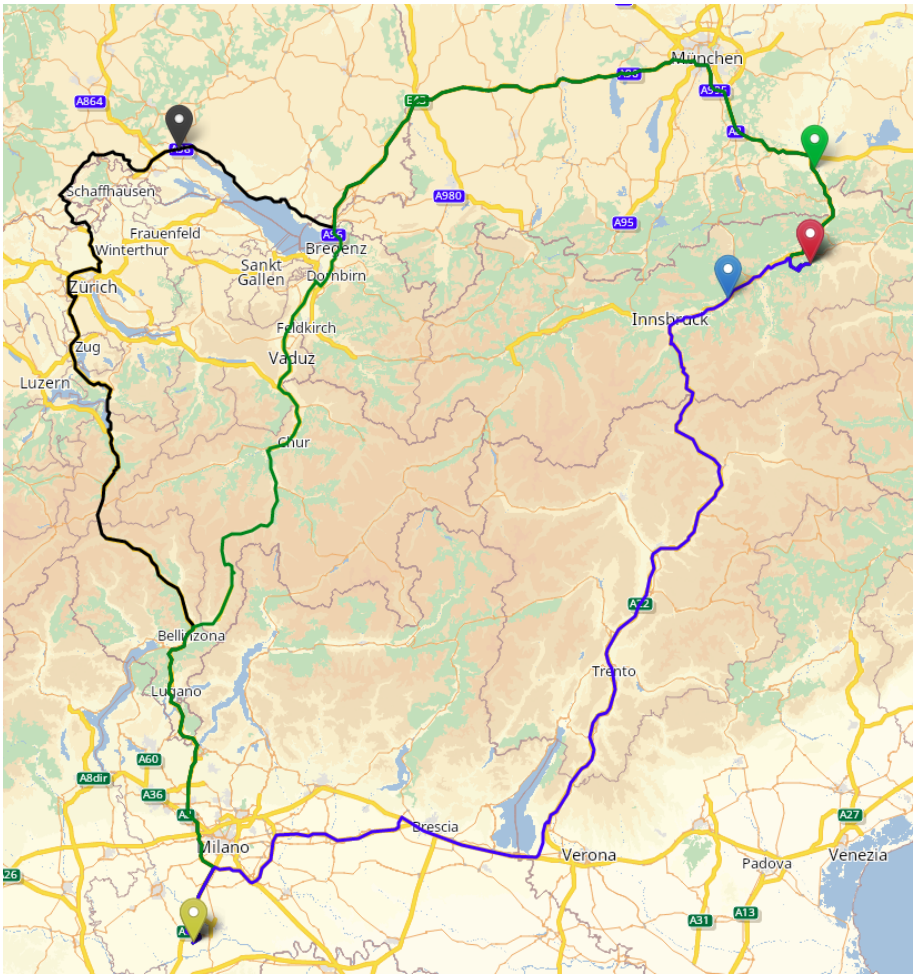

Abbildung 6.5: Ausgewählte Pareto-optimale Routen einer Anfragen. Der Start ist mit einer roten und das Ziel mit einer gelben Markierung versehen. Der Parkplatz, auf welchem auf der jeweiligen Route gewartet wird, ist in der Farbe der Route markiert.

Tabelle 6.10: Ankunfts-, Fahrt- und Wartezeiten der Routen in Abbildung 6.5 sowie die Kategorie des Parkplatzes, an welchem gewartet wird. Alle Zeitangaben in hh:mm:ss.

Route	Ankunftszeit	Fahrtzeit	Wartezeit Start	Wartezeit Parkplatz	Kategorie
schwarz	10:46:36	15:57:59	00:46:57	00:02:36	3
grün	10:52:35	12:50:10	00:46:01	03:16:24	5
blau	12:02:28	10:26:23	00:36:05	07:00:00	4

Tabelle 6.11: Fahrt- und Wartezeiten der Routen. Alle Angaben in hh:mm:ss.

Belegung	Fahrtzeit	Wartezeit Start	Wartezeit Parkplatz
linear	16:48:42	02:41:57	03:25:09
exponentiell	16:20:41	00:56:05	05:39:02
logarithmisch	17:05:21	03:30:45	02:19:42

Lösungen treten auf, wenn eine Umfahrung eines gesperrten Gebiets - meist die Schweiz oder Österreich - langsamer als der direkte Weg mit unerlaubtem Warten auf Nicht-Parkplätzen ist. Wenn das Ziel innerhalb des Planungshorizonts erreichbar ist, existiert jedoch immer mindestens eine weitere gültige Lösung. Bei 9 Anfragen wird keine Route berechnet, da ihr jeweiliges Ziel nicht innerhalb eines Tages erreicht werden kann.

Variation der Kostenbelegung

Wir betrachten den Effekt, den unterschiedliche Kostenbelegungen auf die Qualität der Lösung bei einer Aufteilung der Parkplätze in 5 Kategorien haben. In Abbildung 6.6 ist für alle betrachteten Belegungen die Größe der Pareto-Front für die 200 Anfragen der Testmenge dargestellt.
Auffällig ist hierbei, dass mit zunehmender Priorisierung von Parkplätzen höherer Qualität eine Abnahme der Summe der berechneten Routen einhergeht. So werden bei der exponentiellen Kostenbelegung insgesamt 602 Routen berechnet, im Falle der linearen Belegung sind es noch 572 und bei der logarithmischen Belegung fällt diese Zahl auf 542 Routen. Hierbei stimmen 536 Routen der linearen und logarithmischen Belegung überein, bei der logarithmischen und exponentiellen Kostenbelegung sind es hingegen nur 499 Routen.
Dies ist darauf zurückzuführen, dass die Parkplätze geringer Qualität überwiegen. Somit ergeben sich im exponentiellen Fall mehr Möglichkeiten diese ohne große Umwege anzufahren, was zu einer größeren Pareto-Front führt. Diese Routen sind für die anderen beiden Belegungen jedoch nicht Pareto-optimal, da das Warten an diesen Parkplätzen mit vergleichsweise höheren Kosten belegt ist. Die lineare und logarithmische Belegung generiert hingegen Routen, die weite Umwege in Kauf nehmen, um zu besseren Parkplätzen zu gelangen oder am Start länger warten zu können. Der Abfall der generierten Routen ist auch zwischen diesen beiden Belegungen erkennbar, fällt allerdings deutlich geringer aus, da bereits im linearen Fall meist an Parkplätzen hoher Qualität gewartet wird.

Wir betrachten als Beispiel eine Anfrage von Weikersdorf im Norden Österreichs nach Fonte Vivola in Italien. In den Abbildungen 6.7 und 6.8 sind drei Routen abgebildet, je eine Route für alle drei Kostenbelegungen. Alle Routen erreichen das Ziel am 03. Juli um 16:55:58 Uhr und verlassen den Parkplatz der Kategorie 5 an der deutsch-österreichischen Grenze um 04:42:50 Uhr desselben Tages, um die Grenze exakt zum Ende des Nachtfahrverbots zu überqueren. Sie unterscheiden sich jedoch in ihrer Fahrtzeit sowie den Wartezeiten. In Tabelle 6.11 sind diese dargestellt.
Die Route, die aus der exponentiellen Kostenbelegung hervorgeht, startet nach nur einer knappen Stunde Wartezeit am Start um den direkten Weg zum Parkplatz durch Österreich wahrzunehmen. Auf den Routen der anderen beiden Belegungen wird das Warten am Start jedoch nach Konstruktion stärker priorisiert, weshalb der Fahrer deutlich später startet und deshalb Umwege in Kauf nehmen muss, um das Nachtfahrverbot Österreichs zu umfahren.

Variation der Parkplatzkategorisierung

Wir betrachten im Folgenden, welchen Einfluss unterschiedliche Parkplatzkategorisierungen auf die Qualität der Lösung haben. Bei einer Aufteilung der Parkplätze in 5 Kategorien

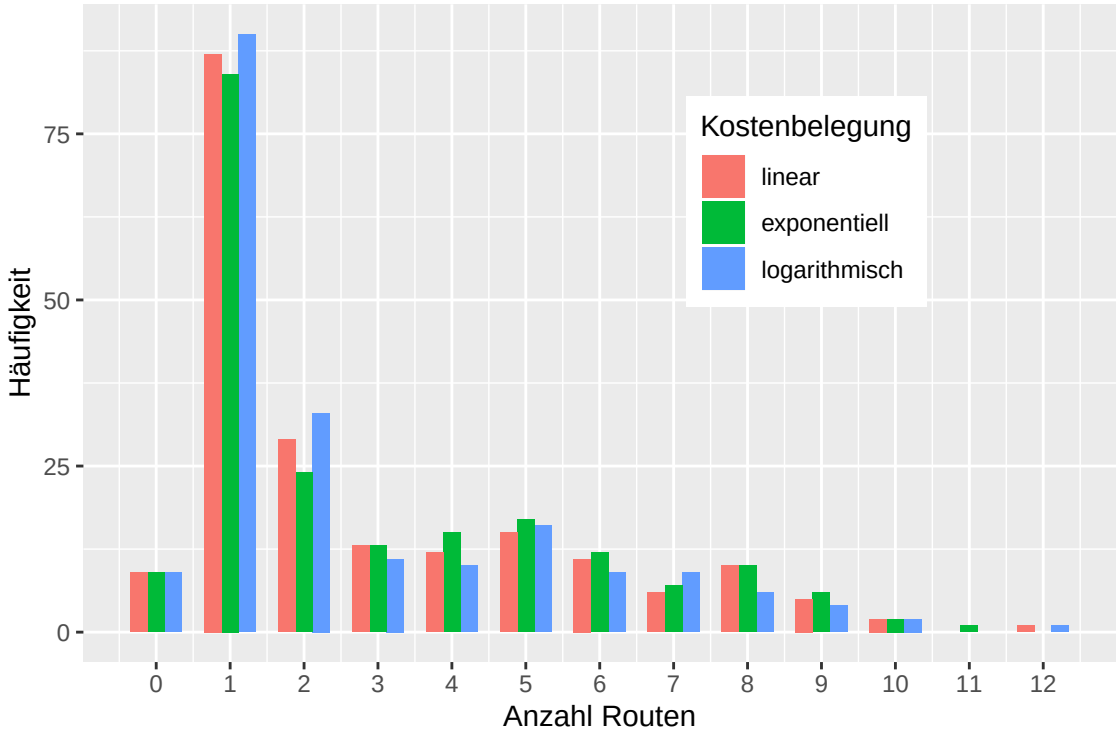

Abbildung 6.6: Vergleich der Größe der Pareto-Fronten der Anfragen der Testmenge 4 bei einer linearen, exponentiellen und logarithmischen Kostenbelegung.

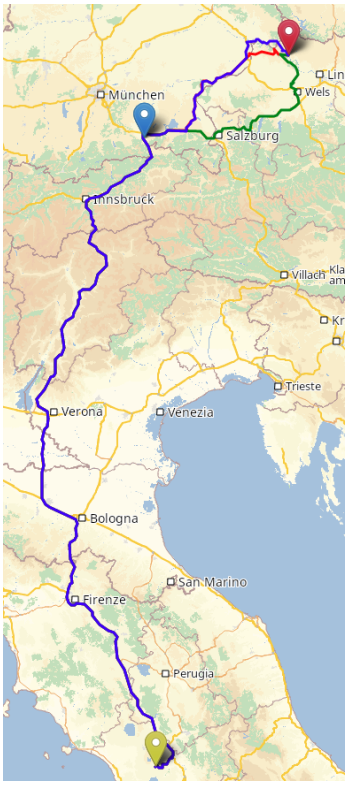

Abbildung 6.7: Berechnete Routen für die lineare (rot), exponentielle (grün) und logarithmische (blau) Kostenbelegung mit einer Ankunftszeit am Ziel um 16:55:58 Uhr am 03. Juli. Der Start ist mit einer roten, der Parkplatz mit einer blauen und das Ziel mit einem gelben Markierung versehen.

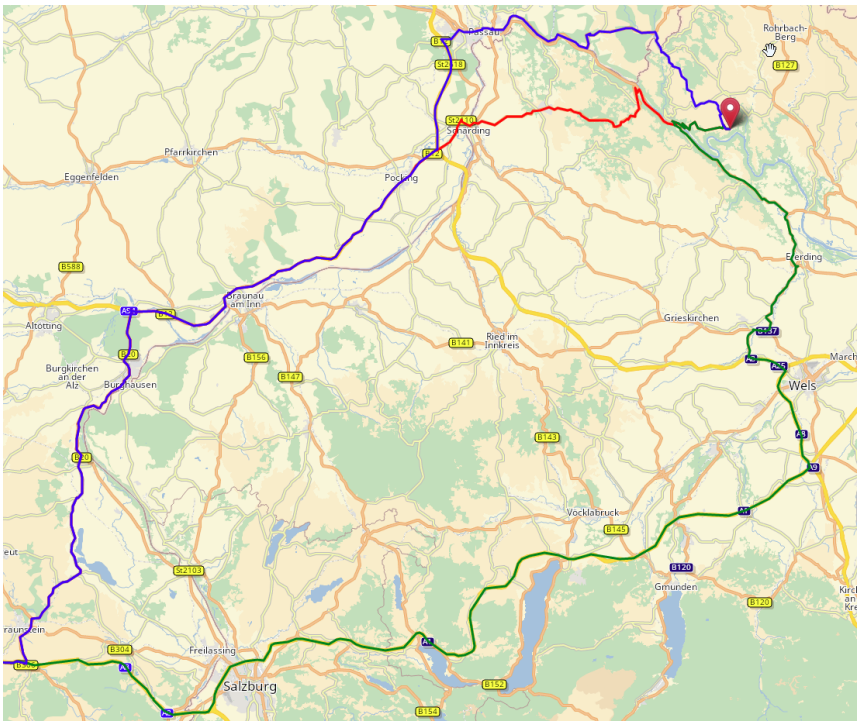

Abbildung 6.8: Nahaufnahme der Unterschiede der drei Routen in Abbildung 6.7.

verwenden wir die lineare Kostenbelegung. In Abbildung 6.9 ist für die drei betrachteten Kategorisierungen die Größe der Pareto-Front der 200 Anfragen der Testmenge 4 dargestellt. Die Anzahl der berechneten Routen fällt von 593 ohne Aufteilung der Parkplätze in mehrere Klassen auf 572 und 563 Routen bei einer Kategorisierung in 5 beziehungsweise 500 Klassen. Dieser Effekt ist analog zur Variation der Kostenbelegung zu betrachten. Bei einer Aufteilung in mehrere Kategorien werden Parkplätze besserer Qualität stärker priorisiert, von denen es jedoch nur wenige gibt, weshalb die Anzahl der generierten Routen fällt.

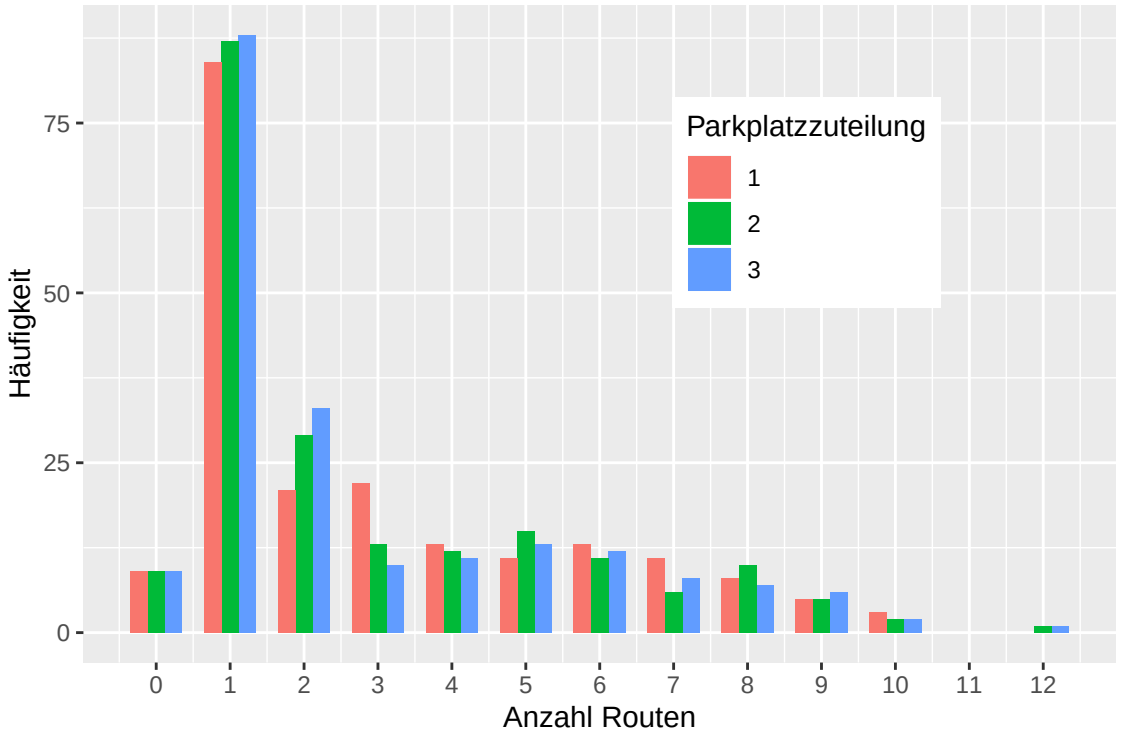

Abbildung 6.9: Vergleich der Größe der Pareto-Fronten der Anfragen der Testmenge 4 bei einer PARTITION der Parkplatzmenge in 1 (rot), 5 (grün) und 500 (blau) Kategorien.

7. Fazit

In dieser Arbeit haben wir eine Lösung zur folgenden Frage vorgestellt und untersucht: Wie kann man effizient sinnvolle Routen auf einem Straßengraphen finden, dessen Kanten aufgrund von Fahrverboten temporär gesperrt sind? Hierbei soll das Finden von geeigneten Parkplätzen priorisiert werden.

Um dieses Problem effizient zu lösen, haben wir in Kapitel 2 ein Modell aufgestellt, bei dem Fahren und Warten mit abstrakten Kosten belegt wird. Die Wartekosten hängen hierbei von der Art des Aufenthaltsortes ab. Die Pareto-Front aus Ankunftszeit und Kosten liefert dann korrespondierende Routen, die sinnvolle Lösungen dieses Problems auf einem realen Straßengraphen darstellen.

In Kapitel 3 haben wir einen Algorithmus vorgestellt, der diese Modellierung umsetzt und die Pareto-Front und die korrespondierenden Routen berechnet. Anschließend haben wir gezeigt, dass - um eine polynomielle Laufzeit dieses Algorithmus zu gewährleisten - die Kosten für das Warten an Nicht-Parkplätzen und dem Fahren gleich gesetzt werden müssen. In Kapitel 4 haben wir bewiesen, dass eine obere Schranke für die Anzahl der optimalen Routen durch die Anzahl der Sperrungen gegeben ist.

In Kapitel 5 haben wir Beschleunigungstechniken für den Algorithmus vorgestellt. Mithilfe des A^*-Algorithmus mit einer perfekten Potentialfunktion durch Contraction Hierarchies konnten wir zielgerichtet suchen und schnell eine erste mögliche Lösung finden. Diese konnte dann im Target-Pruning verwendet werden, um den Suchraum einzuschränken.

Wir haben schließlich in Kapitel 6 die Laufzeit und Qualität unseres Ansatzes auf einem Straßengraph Europas untersucht. Durch die Visualisierung der berechneten Route konnte die Plausibilität des Modells bestätigt werden. Die durchschnittliche Laufzeit des Algorithmus von unter einer Sekunde pro Anfrage wies die Praxistauglichkeit unseres Ansatzes nach. Es gab jedoch Ausreißer, deren Laufzeit teilweise mehrere Größenordnungen langsamer waren, da das Ziel im gegebenen Planungshorizont nicht oder nur zu einem späten Zeitpunkt erreichbar war.

Ausblick

Die beschriebenen Ausreißer können möglicherweise verhindert werden, wenn eine bidirektionale Suche verwendet wird, um eine erste mögliche Lösung zu finden. Damit könnte der Suchraum eingeschränkt werden oder schneller festgestellt werden, dass das Ziel im gegebenen Planungshorizont nicht erreichbar ist. Falls das Ziel erreichbar ist, könnte

dann der beschriebene unidirektionale Algorithmus eingesetzt werden, um die vollständige Pareto-Front zu berechnen.

Ebenfalls kann die Berechnungszeit potentiell weiter durch die Verwendung von Beschleunigungstechniken, wie Contraction Hierarchies oder einem Multi-Level-Ansatz, verbessert werden.

Weiterhin ist eine genauere Untersuchung der in Abschnitt 3.4 vorgestellten Variante dieses Algorithmus mit einer zeitabhängigen Reisezeitfunktion von Interesse. Mithilfe dieser Modifikation können Änderungen der Fahrtzeit einer Kante aufgrund der Verkehrslage berücksichtigt werden.

Schließlich kann dieses Modell erweitert werden, um eine Berücksichtigung der Lenk- und Ruhezeiten von Lkw-Fahrern zur Laufzeit zu gewährleisten. Dies ist von Vorteil, da das Warten auf das Ende einer Sperrung als Pause interpretiert werden kann.

Literaturverzeichnis

[BDG+16] Hannah Bast, Daniel Delling, Andrew Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner und Renato F. Werneck: *Route Planning in Transportation Networks*, Seiten 19–80. Springer International Publishing, Cham, 2016, ISBN 978-3-319-49487-6. https://doi.org/10.1007/978-3-319-49487-6_2.

[Brä18] Christian Bräuer: *Route Planning with Temporary Road Closures.* Master Thesis, Karlsruhe Institute of Technology, 2018.

[Dij59] Edsger W. Dijkstra: *A Note on Two Problems in Connexion with Graphs.* Numerische Mathematik, 1:269–271, 1959.

[GH05] Andrew V. Goldberg und Chris Harrelson: *Computing the Shortest Path: A* Search Meets Graph Theory.* In: *Proceedings of the 16th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA'05)*, Seiten 156–165. SIAM, 2005.

[GSSV12] Robert Geisberger, Peter Sanders, Dominik Schultes und Christian Vetter: *Exact Routing in Large Road Networks Using Contraction Hierarchies.* Transportation Science, 46(3):388–404, August 2012.

[GW05] Andrew V. Goldberg und Renato F. Werneck: *Computing Point-to-Point Shortest Paths from External Memory.* In: *Proceedings of the 7th Workshop on Algorithm Engineering and Experiments (ALENEX'05)*, Seiten 26–40. SIAM, 2005.

[Han80] Pierre Hansen: *Bicriterion Path Problems.* In: Günter Fandel und Tomas Gal (Herausgeber): *Multiple Criteria Decision Making Theory and Application*, Seiten 109–127, Berlin, Heidelberg, 1980. Springer Berlin Heidelberg, ISBN 978-3-642-48782-8.

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

[Kar72] Richard M. Karp: *Reducibility among Combinatorial Problems.* In: *Complexity of Computer Computations*, Seiten 85–103. Plenum Press, 1972.

[OR90] Ariel Orda und Raphael Rom: *Shortest-Path and Minimum Delay Algorithms in Networks with Time-Dependent Edge-Length.* Journal of the ACM, 37(3):607–625, 1990.

[PG13] Luigi Di Puglia Pugliese und Francesca Guerriero: *A survey of resource constrained shortest path problems: Exact solution approaches.* Networks, 62(3):183–200, 2013. https://onlinelibrary.wiley.com/doi/abs/10.1002/net.21511.

[Str19] Ben Strasser: *RoutingKit*, 2019. https://github.com/RoutingKit/RoutingKit, besucht am 06.09.2019.

[SZ19] Ben Strasser und Tim Zeitz: *A* with Perfect Potentials*, 2019. `https://arxiv.org/abs/1910.12526`.

[vdTdWB18] Marieke van der Tuin, Mathijs de Weerdt und G. Batz: *Route Planning with Breaks and Truck Driving Bans Using Time-Dependent Contraction Hierarchies*, 2018. `https://www.aaai.org/ocs/index.php/ICAPS/ICAPS18/paper/view/17745/16955`.