Karlsruhe Institute of Technology

Optimale zeitabhängige Pausenplanung für LKW-Fahrer mit integrierter Parkplatzwahl

Bachelorarbeit
von

Christian Bräuer

An der Fakultät für Informatik
Institut für theoretische Informatik

Erstgutachter:	Prof. Dr. Dorothea Wagner
Zweitgutachter:	Prof. Dr. Peter Sanders
Betreuende Mitarbeiter:	Moritz Baum
	Valentin Buchhold
	Alexander Kleff
	Dr. Frank Schulz

Bearbeitungszeit: 15. Juli 2016 – 9. November 2016

Eigenständigkeitserklärung

Ich versichere wahrheitsgemäß, die Arbeit selbstständig verfasst, alle benutzten Hilfsmittel vollständig und angegeben und alles kenntlich gemacht zu haben, was 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, 9. November 2016

Abstract

Truck drivers have to obey rules about driving and break time to avoid accidents because of fatigue. To plan a tour for a truck driver, one has to consider the need for sufficiently large parking lots due to the vehicle size for taking a break. Also, forseeable impediments because of the rush hour may lead to a delay of the trip, which in turn can cause a violation of the regulations mentioned. Therefore, we present an algorithm based on *Contraction Hierarchies* which finds every parking lot within a given driving time from a starting node in a time-dependent graph. We speed up the parking lot-search by using the approximation algorithm of Imai and Iri, although this results in inaccuracies.

Moreover, we present and implement an algorithm for the regulations of the EU, which schedules breaks on parking lots to complete a given truck tour without violating the hours of service of the driver. We test the algorithm with an up-to-date road graph of Germany together with realistic locations of parking lots. We compare the feasibility of tours for time-dependent and time-independent driving times on several test cases. Furthermore, we describe an algorithm for the regulations of the USA, which differ from those of the EU.

Deutsche Zusammenfassung

LKW-Fahrer müssen in vielen Ländern Lenk- und Ruhezeiten einhalten, um Unfälle aufgrund von Übermüdung zu vermeiden. Bei der Planung einer Tour für LKW-Fahrer muss nicht nur beachtet werden, dass für eine Pause aufgrund der Fahrzeuggröße dafür vorgesehene LKW-Parkplätze nötig sind. Auch vorhersehbare Verkehrsbehinderungen wie Berufsverkehr können zu einer Verzögerung der Fahrt führen, die in einem Verstoß gegen die Lenk- und Ruhezeiten mündet. Daher werden wir einen Algorithmus vorstellen, der auf einem zeitabhängigen Straßennetz mittels *Contraction Hierarchies* alle Parkplätze findet, die von einem gegebenen Standort innerhalb einer gegebenen Fahrzeit erreichbar sind. Wir werden diese Parkplatzsuche auf Kosten von exakten Ergebnissen mit dem Approximationsalgorithmus von Imai und Iri beschleunigen.

Zusätzlich wird für die Lenk- und Ruhezeiten der EU ein Algorithmus erläutert und implementiert, der zu einer gegebenen LKW-Tour Pausenzeiten auf Parkplätzen einplant, um die Tour ohne Verstöße gegen die Lenk- und Ruhezeiten zu beenden. Wir werden den Algorithmus mit einem aktuellen zeitabhängigen Straßengraphen von Deutschland und realistischen Parkplatzorten testen. Für verschiedene Testfälle wird die Erfüllbarkeit von Touren für zeitabhängige und zeitunabhängige Fahrzeiten verglichen. Weiterhin wird ein Algorithmus beschrieben, der die von den EU-Regelungen abweichenden Lenk- und Ruhezeiten der USA berücksichtigt.

Inhaltsverzeichnis

1. Einleitung

Gütertransporte per Lastkraftwagen (LKW) stellen einen wichtigen Grundstein unserer heutigen Versorgung dar. Für die Transportunternehmen bedeutet dies in der Regel nicht nur, dass einfache Fahrten von einem Be- zum Abladeort durchgeführt werden müssen. Um konkurrenzfähig zu bleiben, müssen die Unternehmen stattdessen auch mehrere Fahrten zu einer Tour zusammenfassen. Dadurch sollen möglichst viele Aufträge durch die LKW-Fahrer abgearbeitet werden können. Diese Touren enthalten mehrere Kunden, die der Fahrer in einer vorgegebenen Reihenfolge anfahren muss, um seine Ladung dort abzuladen. Die Kunden erwarten dabei vom Fahrer eine Ankunft innerhalb eines oder mehrerer Zeitfenster; insbesondere darf die Ladung nicht zu spät ankommen.

Dies übt Druck auf die Fahrer aus, die versuchen, diese Zeitfenster einzuhalten und die Touren rechtzeitig abzuschließen. Nicht selten hat das in der Vergangenheit zu schweren Unfällen, ausgelöst durch übermüdete LKW-Fahrer, geführt. Um solche Unfälle zu vermeiden, haben weltweit Gesetzgeber Bestimmungen erlassen, die die Fahrt- und Pausendauer von LKW-Fahrern regeln. Transportunternehmen und Fahrer müssen diese Pausenzeiten bei der Planung ihrer Touren berücksichtigen. Erfolgt die Planung mit zu kurzen zeitlichen Reserven, dann können vorgeschriebene Pausen oder Liefertermine unter Umständen nicht eingehalten werden, was Strafzahlungen an den Gesetzgeber und wirtschaftliche Nachteile nach sich zieht. Andererseits kann eine Planung, bei der zu lange Pufferzeiten eingeplant werden, auch zu einer mangelnden Auslastung der Fahrer führen, die so ihre Lenkzeiten nicht voll nutzen, was Nachteile gegenüber der Konkurrenz zur Folge hat.

Die Schwierigkeit bezüglich der Wahl der Pausen liegt darin, dass diese nur auf geeigneten LKW-Parkplätzen eingelegt werden können. In der Praxis werden Touren ohne Kenntnis über die Orte von Parkplätzen geplant, entsprechend obliegt dem Fahrer die Parkplatzwahl. Wir wollen daher im Folgenden einen Algorithmus darlegen, der für gegebene Touren Pausenorte sowie Abfahrts- und Pausenzeiten optimal berechnet.

1.1 Problembeschreibung

Im Rahmen dieser Arbeit nehmen wir Bezug auf die *Verordnung (EG) 561/2006* der Europäischen Union [Eur06] und auf die *Hours of Service of Drivers* der Federal Motor Carrier Safety Administration [Fed11]. Diese Verordnungen legen für eine Weiterfahrt im gewerblichen Güterverkehr auf der Straße unter anderem fest:

1. In der Europäischen Union eine mindestens 45-minütige Pause nach spätestens 4,5 Stunden *gefahrener* Zeit seit der letzten Pause. Außerdem muss eine mindestens

elfstündige Pause nach spätestens neun Stunden Fahrzeit oder nach spätestens 13 Stunden seit dem Ende der letzten elfstündigen Pause eingelegt werden.

2. In den USA eine mindestens 30-minütige Pause nach spätestens acht Stunden *vergangener* Zeit seit der letzten Pause. Außerdem muss eine mindestens zehnstündige Pause nach spätestens elf Stunden Fahrzeit oder nach spätestens 14 Stunden seit dem Ende der letzten zehnstündigen Pause eingelegt werden.

Ein Halt zählt dabei nur als Pause, wenn der Fahrer frei über seine Tätigkeit bestimmen kann und die Zeit ausschließlich zur Erholung genutzt wird. In beiden Verordnungen werden noch einige Ausnahmen, insbesondere in den EU-Regeln das Aufteilen der 45-minütigen Pause in zwei kürzere, gemacht.

Es sei nun eine Folge von Kunden $k_1, \ldots, k_n$ gegeben. Das Be- und Entladen (der *Service*) mit Dauer S_i am Kunden k_i könne nur innerhalb von Servicefenster $W_i = [b_{1,i}, e_{1,i}] \cup \cdots \cup [b_{m,i}, e_{m,i}]$ begonnen werden. Zusätzlich sei ein Straßengraph $G = (V, E, \omega)$ mit Knoten V, Kanten E und statischen zeitabhängigen Fahrzeiten ω, sowie eine Menge $P \subseteq V$ von Parkplätzen gegeben. Außerdem sei t_0 der Zeitpunkt, zu dem der Fahrer die letzte Tätigkeit der vorherigen Tour (Service leisten oder fahren) beendet hat. Das Tupel $(t_0, (k_1, \ldots, k_n), (W_1, \ldots, W_n), (S_1, \ldots, S_n), G)$ sei die Beschreibung einer *Tour*. Wir gehen davon aus, dass der LKW-Fahrer sich zusammen mit seinem LKW beim ersten Kunden k_1 befindet und bisher eine Dauer von c seit der letzten Pause gefahren (Vorgaben der EU) bzw. seit der letzten Pause vergangen ist (Vorgaben der USA). Ein *Zeitplan* legt die Abfahrtszeiten von Kunden k_i und Parkplätzen $p \in P$, sowie den Beginn der Servicezeiten an einem Kunden fest, vgl. [Kin16]. Ein Zeitplan ist *gültig*, wenn

1. die Kunden in ihrer gegebenen Reihenfolge mit den Abfahrtszeiten des Zeitplans angefahren werden können,
2. der Beginn der Servicezeiten innerhalb der entsprechenden Servicefenster liegt,
3. der Service am Kunden vor Abfahrt komplett durchgeführt werden kann und
4. die entsprechenden Pausenregelungen eingehalten werden.

Touren, zu denen ein gültiger Zeitplan existiert, heißen *erfüllbar*. Ziel der Arbeit ist es, einen Algorithmus zu beschreiben, der das folgende Entscheidungsproblem zum LKW-Fahrer-Scheduling löst: Ist eine gegebene Tour erfüllbar?

Wir werden dazu einen Algorithmus für die Vorgaben der EU und für die der USA angeben, der das frühestmögliche Ende einer gegebenen Tour berechnet, falls diese erfüllbar ist. Da uns keine Parkplatzdaten für die USA vorliegen, werden wir den Fokus auf die Vorgaben der EU legen. Entsprechend wird auch nur der Algorithmus für die Vorgaben der EU implementiert und evaluiert.

1.1.1 Einschränkungen und Annahmen

Grundsätzlich betrachten wir den Fahrer und dessen LKW als eine Einheit. Der Fahrer trennt sich somit nicht von seinem LKW, und wir werden entsprechend die Begriffe „LKW“, „LKW-Fahrer“ und „Fahrer“ synonym verwenden.

Beide oben genannten Vorgaben beziehen sich auf zwei verschiedene Pausentypen, zum einen auf die tägliche Ruhezeit (zehn- und elfstündige Pause) und zum anderen auf kürzere Pausen (30 und 45-minütige Pausen). Wir betrachten allerdings nur die Planung der 45- bzw. 30-Minuten-Pausen. Dies ermöglicht eine Tagesplanung, die auf die Einhaltung der täglichen Gesamtfahr- und -arbeitszeit überprüft werden kann. Außerdem vermeiden wir dadurch, dass die gefahrene und die vergangene Zeit seit der letzten Pause gleichzeitig betrachtet werden müssen. So können wir für die Fahrt zu jedem Zeitpunkt und jeden

Kunden die bisherige optimale Abfahrts- und Pausenzeit festlegen, wie wir später sehen werden. Betrachten wir beide Kriterien gleichzeitig, so können wir dies nicht, wie folgendes Beispiel zeigt:

Ein Fahrer beginnt seine Tour nach einer Pause zum Zeitpunkt t_0 und ist daher seit der letzten Pause nicht gefahren, und es ist auch bisher keine Zeit seit der letzten Pause vergangen. Wir betrachten den Zeitpunkt t, an dem sich der Fahrer am nächsten Kunden befinden soll. Wir zeigen, dass sich nicht immer ein eindeutig bestes Tupel mit kleinsten Werten der gefahrenen und vergangenen Zeit zum Zeitpunkt t am nächsten Kunden finden lässt:

Der Fahrer kann um t_0 sofort zum nächsten Kunden fahren, und dort nach der Ankunft um $t_0 + d_1$ warten, bis t erreicht ist. Damit ist d_1 die gefahrene Zeit und $t - t_0$ die vergangene Zeit seit der letzten Pause. Alternativ kann der Fahrer aber auch erst später losfahren um damit die Wartezeit am nächsten Kunden zu vermeiden. Er kann also so fahren, dass er direkt zum Zeitpunkt t am nächsten Kunden ankommt. Aufgrund zeitabhängiger Fahrzeiten sei die Fahrtdauer nun länger als im ersten Fall, nämlich $d_2 > d_1$. Die vergangene Zeit ist ebenfalls d_2, denn er muss am nächsten Kunden nicht warten. Die Wartezeit vor der Abfahrt zählt nicht als vergangene Zeit seit der letzten Pause, denn der Fahrer kann seine (bis ursprünglich t_0) gemachte Pause verlängern. Da er später als im ersten Fall losfährt, ist $d_2 < t - t_0$. Mit der späteren Abfahrt ist der Fahrer also seit Ende der letzten Pause länger gefahren, allerdings ist weniger Zeit seit dem Ende der letzten Pause vergangen.

Zusätzlich werden wir nur betrachten, dass zwischen je zwei Kunden nur maximal eine Pause auf einem Parkplatz gemacht wird. Ebenfalls werden keine Wartezeiten auf Parkplätzen eingeplant, die kürzer als die Pausendauer sind. Auch die oben angesprochene mögliche Pausenteilung, wie es die Regelungen der EU erlaubn, werden wir nicht berücksichtigen. Wir betrachten also nur die folgenden Pausenregeln: In der EU darf ein Fahrer nur dann weiterfahren, wenn nach einer Gesamtfahrzeit von $R^{driven} = 4{,}5\,\mathrm{h} = 16200\,\mathrm{s}$ seit der letzten Pause eine Pause von $R^{break} = 45\,\mathrm{min} = 2700\,\mathrm{s}$ eingelegt wird. In den USA darf ein Fahrer nur dann weiterfahren, wenn nach einer vergangenen Zeit von $R^{elapsed} = 8\,\mathrm{h} = 28800\,\mathrm{s}$ seit der letzten Pause eine Pause von $R^{break} = 30\,\mathrm{min} = 1800\,\mathrm{s}$ eingelegt wird. Eine Pause ist ein ununterbrochener Halt mit einer Dauer von mindestens R^{break} auf einem Parkplatz oder bei einem Kunden, sofern zeitgleich kein Service geleistet wird. Diese Einschränkungen sorgen einerseits für eine Vereinfachung des Problems, führen allerdings immer noch zu Zeitplänen, die praktisch genutzt werden können.

Anschließend soll noch angemerkt werden, dass wir davon ausgehen, dass der Fahrer sich zu Beginn der Tour am ersten Kunden k_1 befindet. Ist dies nicht der Fall und soll die Fahrt vom Standort des Fahrers zum ersten Kunden auch geplant werden, so kann ein Kunde k_0 künstlich vor k_1 eingefügt werden, der sich am Ort des Fahrers befindet. Kunde k_0 erhält ein ab t_0 immer geöffnetes Servicefenster $W_0 = [t_0, \infty)$ sowie eine Servicedauer $S_0 = 0$. So plant der Schedulingalgorithmus auch die Fahrt zum ersten Kunden k_1 mit in die Tour ein.

1.2 Stand der Forschung

Zur Routenplanung sind verschiedene Algorithmen bekannt. Dijkstras Algorithmus [Dij59] wurde bereits 1959 beschrieben. Heute gibt es bereits viele weitere Algorithmen, die eine Route um Größenordnungen schneller berechnen können, als Dijkstras ursprünglicher Algorithmus, siehe dazu den Überblick von Bast et al. [BDG+15]. Insbesondere auf Straßengraphen können Hierarchien ausgenutzt werden, da einige Straßen „wichtiger“ sind als andere. Als Beispiel sei der Vergleich von Autobahnen und Feldwegen genannt. Eine Möglichkeit der Nutzung dieser Hierarchien bieten *Contraction Hierarchies*, die von Geisberger et al. vorgeschlagen wurden [GSSD08]. Diese können außerdem auf statische zeitabhängige Fahrzeiten erweitert werden, wie Batz et al. zeigen [BGSV13].

Das Problem der Pausenplanung (Scheduling) für LKW-Fahrer wurde in ähnlicher Fassung in verschiedenen Ausprägungen von Goel betrachtet. Dieser gibt für die Regelungen der EU [Goe10] einen Algorithmus an, der einen gültigen Zeitplan finden, sofern dieser existiert. Auch für die Regelungen der USA haben Goel und Kok [GK12] einen Algorithmus veröffentlicht, der das Problem mit einem Zeitfenster pro Kunde mit einer Laufzeit von $\mathcal{O}(n^2)$ (n Anzahl der Kunden) löst. Sowohl für die Regelungen der EU, als auch für die der USA werden die in dieser Arbeit ausgeschlossenen weitergehenden Vorgaben bezüglich täglicher elfstündiger bzw. zehnstündiger Pausen berücksichtigt. Angenommen wird dort allerdings, dass Pausen überall möglich und die Fahrzeiten konstant sind. Außerdem gibt es von Goel weitere Veröffentlichungen zu den Regelungen in Kanada [Goe12a] und in Australien [Goe12b], sowie zu den Regelungen von Australien von Goel et al. [GAS12].

Weiterhin wurde das Problem von Kleff betrachtet [Kle16]. Zusätzlich wurde dort die Berechnung eines Zeitplans mit minimaler Dauer aufgegriffen. Kleff konnte zeigen, dass das von ihm als „EU-Day Minimum Duration Truck Driver Scheduling Problem“ bezeichnete Problem für eine Tagesplanung im zeitunabhängigen Fall mit den Regelungen der EU in $\mathcal{O}(n^2 \cdot m)$ (n Anzahl der Kunden, m Gesamtanzahl der Zeitfenster) lösbar ist. Umgesetzt wurde dies in der Masterarbeit von Kinz [Kin16]. Betrachtet wurde auch eine Erweiterung des vorgestellten Algorithmus, die örtlich beliebig platzierte Parkplätze optimal wählen kann.

Koç et al. erweitern das Schedulingproblem um eine Kostenbetrachtung beim Einlegen einer Pause [KJL16], um den Strom- oder Kraftstoffverbrauch des Fahrers für dessen Aktivitäten während der Pause zu berücksichtigen. Parkplätze können nur eingeplant werden, wenn diese auf dem direkten Weg zwischen zwei Kunden liegen.

Zeitabhängig wurde das Problem bereits durch Kok et al. betrachtet [KHS11]. Die Autoren verwenden ein ganzzahliges lineares Programm, um die EU-Regelungen, inklusive der vorgeschriebenen elfstündigen Pausen, umzusetzen. Pausen können allerdings nur an Kunden durchgeführt werden. Getestet wurde der vorgestellte Algorithmus mit synthetischen Fahrzeitfunktionen.

Einen weitergehenden Überblick über den Stand der Forschung zum LKW-Fahrer-Scheduling bietet die Publikation von Koubâa et al. [KDDEM16]. Dort finden sich auch Veröffentlichungen zu Schedulingproblemen mit mehreren Fahrern für einen LKW und zu Regelungen weiterer Staaten.

Neu an unserem Ansatz ist die gemeinsame Betrachtung des Schedulings und zeitabhängiger Routenplanung. Parkplätze werden durch unseren Algorithmus automatisch aus einer größeren Menge ausgewählt und nur bei Bedarf angefahren. Zusätzlich führen wir Tests auf einem aktuellen zeitabhängigen Straßengraphen von Deutschland durch, dem realistische Fahrzeiten und die Orte echter Parkplätze zugrunde liegen.

1.3 Gliederung

Der Aufbau der Arbeit ist wie folgt: Kapitel 2 gibt einen Überblick über Definitionen und Algorithmen, die im weiteren Verlauf der Arbeit benötigt werden. In Kapitel 3.1 werden notwendige Erweiterungen der in Kapitel 2 besprochenen Algorithmen für die Routenplanung vorgestellt. Daraufhin beschäftigen wir uns in Kapitel 3.2 mit dem Algorithmus zur Lösung des in Kapitel 1.1 definierten Schedulingproblems. Kapitel 4 enthält die experimentelle Evaluierung der Routenplanungsalgorithmen und des Schedulingalgorithmus im Bezug auf die Laufzeit und die Qualität des Schedulings im Vergleich zu einem Scheduling mit konstanten Fahrzeiten. Zum Abschluss wird in Kapitel 5 ein Fazit mit möglichen zukünftigen Erweiterungen des Problems gezogen.

2. Grundlagen

In diesem Kapitel soll ein Überblick über die in dieser Arbeit verwendeten Begriffe und Definitionen gegeben werden. Außerdem werden grundlegende Algorithmen zur Routenplanung und zum Scheduling besprochen, die in den folgenden Kapiteln als Grundlage genutzt werden.

2.1 Definitionen

Für diese Ausarbeitung sei $G = (V, E, \omega)$ ein *gerichteter* Graph, der ein Straßennetz modelliert. Knoten $v \in V$ stellen dabei Kreuzungen, Kunden und Parkplätze dar, Kanten $e \in E$ repräsentieren Straßensegmente. Wir betrachten den *zeitabhängigen* Fall, das heißt, jeder Kante $(u, v) \in E$ wird eine stückweise lineare Funktion $\omega_{u,v}$ als Kantengewicht zugeordnet. Die Kantengewichte $\omega_{u,v}$ werden *Fahrzeitfunktionen* genannt. Eine Fahrzeitfunktion $\omega_{u,v} : \mathbb{R} \to \mathbb{R}_{>0}$ bildet den Abfahrtszeitpunkt t (in Sekunden) auf die Dauer der Fahrt $\omega_{u,v}(t)$ (in Sekunden) von u nach v ab. Sind keine anderen Einheiten angegeben, so sind die angegebenen Werte in Sekunden. Der Abfahrtszeitpunkt $t = 0$ gibt eine Abfahrt um 0 Uhr nachts an. Alle Fahrzeitfunktionen $\omega_{u,v}$ sind für unsere Zwecke periodisch mit Periodendauer $\Pi = 24\,\mathrm{h} = 86400\,\mathrm{s}$, das heißt $\omega_{u,v}(x) = \omega_{u,v}(x + \Pi)$. Wir gehen davon aus, dass eine Fahrt bei früherer Abfahrt immer auch früher endet als bei späterer Abfahrt. Formal setzen wir also folgendes voraus: $\forall \varepsilon > 0, t \in \mathbb{R} : \omega_{u,v}(t) < \varepsilon + \omega_{u,v}(t + \varepsilon)$. Diese Eigenschaft nennen wir *FIFO** (First-In First-Out). Dies ist eine stärkere Einschränkung als die in der Literatur mit *FIFO* [BGSV13] bezeichneten Eigenschaft. Durch diese Einschränkung können wir eine Fahrtgeschwindigkeit des LKW von Null ausschließen, um so die Pausen tatsächlich optimal setzen zu können. Alle zeitabhängigen Funktionen werden wir mit griechischen Buchstaben bezeichnen. Für den *zeitunabhängigen* Fall betrachten wir konstante Fahrzeitfunktionen. Es sei dazu Φ_c die Funktion mit $\Phi_c \equiv c$. Außerdem definieren wir $\omega_{u,v} \coloneqq \Phi_\infty$, falls $(u, v) \notin E$ sowohl für den zeitabhängigen, als auch für den zeitunabhängigen Fall. Wenn wir zeitunabhängige Kantengewichte betrachten, so stellen diese immer den Fall dar, dass keine Staus und Verzögerungen betrachtet werden (Freeflow).

Für zeitabhängige Funktionen φ und ψ sei min die Merge-Operation mit $(\min(\varphi, \psi))(t) = \min(\varphi(t), \psi(t))$. Zusätzlich sei $\min \varphi \coloneqq \min_{t \in \mathbb{R}} \varphi(t)$, analog für $\max \varphi$. Weiterhin sei $(\varphi + \psi)(t) \coloneqq \varphi(t) + \psi(t)$ und $(\varphi \odot \psi)(t) \coloneqq \varphi(t) + \psi(t + \varphi(t))$. Wir nennen $\odot$ die Link-Operation. Ein Tupel $(v_1, \ldots, v_n)$ mit Knoten $v_i \in V$ heißt Pfad, falls $(v_i, v_{i+1}) \in E$ für $i \in \{1, \ldots, n-1\}$. Die Länge eines Pfades $(v_1, \ldots, v_n)$ ist definiert als $((\omega_{v_1,v_2} \odot \omega_{v_2,v_3}) \odot \ldots) \odot \omega_{v_{n-1},v_n}$. Es sei $\Psi_{u,v}$ das Minimum (wie oben definiert) der Länge aller Pfade mit

Startknoten u und Zielknoten v. Wir bezeichnen die Fahrzeitfunktion $\Psi_{u,v}$ als Distanz bzw. als (Fahrzeit-)Profil von u nach v. Die Funktion $\Psi_{u,v}$ bildet wie auch die Kantengewichte ω *Abfahrts*zeitpunkte t bei Knoten u auf Fahrzeiten $\Psi_{u,v}(t)$ ab. Wir definieren zusätzlich $\overline{\Psi}_{u,v}(t + \Psi_{u,v}(t)) \coloneqq \Psi_{u,v}(t)$, was aufgrund der FIFO*-Eigenschaft wohldefiniert ist. Die Funktion $\overline{\Psi}_{u,v}$ bildet *Ankunfts*zeitpunkte t bei Knoten v auf Fahrzeiten $\overline{\Psi}_{u,v}(t)$ ab. Während $\Psi_{u,v}$ die FIFO*-Eigenschaft erfüllt, tut $\overline{\Psi}_{u,v}$ dies nicht notwendigerweise.

Für den Schedulingalgorithmus werden zeitabhängige Funktionen benötigt, die nicht zu jedem Zeitpunkt definiert sind. Ist eine Funktion φ zu einem Zeitpunkt t nicht definiert, so schreiben wir $\varphi(t) = \bot$. Ist $\varphi(t) = \bot$ für alle $t < t_0$ so sei $\alpha(\varphi) \coloneqq \min\{t \mid \varphi(t) \neq \bot\}$ der früheste Zeitpunkt, zu dem φ definiert ist. Für $\varphi \equiv \bot$ sei $\alpha(\varphi) = \bot$. Fahrzeitfunktionen sind, wie oben beschrieben, zu jedem Zeitpunkt definiert. Für $c \in \mathbb{R}$ gilt $\min(c, \bot) = c$, $c + \bot = \bot$ und $\min_{t \in \emptyset} \varphi(t) = \bot$.

2.2 Algorithmen zur Routenplanung

Da wir das Scheduling in Verbindung mit der Routenplanung betrachten, werden im Folgenden Algorithmen zur Routenplanung dargelegt, auf denen wir in Kapitel 3 aufsetzen werden. Zur Berechnung von $\Psi_{u,v}$ für einen gegebenen Graphen $G = (V, E, \omega)$ mit zeitunabhängigen Kantengewichten kann der seit 1959 bekannte Algorithmus von Dijkstra [Dij59] verwendet werden. Eine Variante für zeitabhängige Kantengewichte ist in Algorithmus 2.1 beschrieben [BGSV13]. Der Algorithmus speichert vorläufige Profile von s zu bisher gefundenen Knoten und aktualisiert diese, bis alle Knoten das endgültige Profil erhalten haben. Falls nicht die Fahrzeitfunktionen zu allen Knoten im Graphen berechnet werden sollen, sondern nur die Fahrzeit zu einem Knoten $t \in V$, kann der Algorithmus nach Zeile 6 schon abgebrochen werden, falls $\max \Psi_{s,t} < \min \Psi_{s,u}$.

Algorithmus 2.1: Dijkstras Algorithmus (zeitabhängig)

Eingabe: Graph $G = (V, E, \omega)$, Startknoten s
Daten: Prioritätswarteschlange Q
Ausgabe: Fahrzeitfunktionen $\Psi_{s,v}$ für alle $v \in V$

```
1  forall v ∈ V do
2      Ψ_{s,v} ← Φ_∞
3  Q.insert(s, 0)
4  Ψ_{s,s} ← Φ_0
5  while Q is not empty do
6      u ← Q.deleteMin()
7      forall (u, v) ∈ E do
8          if ∃t : (Ψ_{s,u} ⊙ ω_{u,v})(t) < Ψ_{s,v}(t) then
9              Ψ_{s,v} ← min{Ψ_{s,u} ⊙ ω_{u,v}, Ψ_{s,v}}
10             if Q.contains(v) then
11                 Q.decreaseKey(v, min Ψ_{s,v})
12             else
13                 Q.insert(v, min Ψ_{s,v})
```

Zur Beschleunigung des Algorithmus von Dijkstra können verschiedene Techniken [BDG+15] angewandt werden. Im Rahmen dieser Arbeit werden *Contraction Hierarchies* verwendet, vgl. [GSSD08] für zeitunabhängige Kantengewichte und [BGSV13] für zeitabhängige Kantengewichte. Diese nutzen eine Hierarchie, um „wenig relevante“ Knoten bei der Suche

zu überspringen. Dazu wird eine Knotenreihenfolge festgelegt, die diese „Relevanz“ repräsentiert. Durch Einfügen von Abkürzungen (Shortcuts) können diese Knoten in ihrer Reihenfolge entfernt werden, ohne Distanzen zweier beliebiger verbleibender Knoten im resultierenden Graphen zu ändern. Dieser Prozess wird Kontraktion (Contraction) genannt. Im Folgenden soll nun die algorithmische Funktionsweise der Kontraktion und der Berechnung der Distanz eines Knotens zu einem anderen erläutert werden.

Wie oben erwähnt, werden die Knoten des Graphen nacheinander kontrahiert. Sei v der Knoten, der im nächsten Schritt kontrahiert werden soll. Durch Entfernen von v aus dem Graphen ändert sich die Distanz zwischen zwei beliebigen Knoten s und t ($t \neq v \neq s$) genau dann, wenn v zu einem beliebigen Zeitpunkt auf allen kürzesten Pfaden von s nach t liegt. Daher muss für alle Knoten u und w, für die (u, v, w) ein Pfad ist und für die v zu einem Zeitpunkt auf allen kürzesten Pfaden von u nach w liegt, eine Abkürzung zwischen u und w mit Fahrzeitfunktion $\omega_{u,w} = \omega_{u,v} \odot \omega_{v,w}$ eingefügt werden. Existiert bereits eine Kante zwischen u und w, so setze $\omega_{u,w}$ auf $\min\{\omega_{u,w}, \omega_{u,v} \odot \omega_{v,w}\}$. Algorithmus 2.2 stellt die Kontraktion in Pseudocode dar, Abbildung 2.1 zeigt beispielhaft einen Graphen vor und nach der Kontraktion des Knotens 4.

Algorithmus 2.2: KONTRAKTION

Eingabe: Graph $G = (V, E, \omega)$, Knoten v
Ausgabe: G ohne Knoten v

```
1 forall (u, v) ∈ E do
2     forall (v, w) ∈ E do
3         if ∃t: length of (u, v, w) at t is equal to Ψ_{u,w}(t) then
4             ω(u, w) ← min{ω(u, w), ω(u, v) ⊙ ω(v, w)}
5             E ← E ∪ {(u, w)}
6 V ← V \ v
7 E ← {(x, y) ∈ E | x, y ≠ v}
8 return G
```

Abbildung 2.1: Beispielgraph vor (links) und nach (rechts) der Kontraktion von Knoten 4. (Φ_c gibt die jeweiligen Kantengewichte an.)

Ein wesentlicher Teil der Kontraktion ist das Finden aller kürzesten Wege, die zu einem Zeitpunkt über v führen. Dazu wird eine Variante von Dijkstras Algorithmus, genannt Zeugensuche (Witness Search), ausgeführt. Die Zeugensuche wird für jeden Knoten u mit $(u, v) \in E$ als Startknoten ausgeführt und abgebrochen, wenn für alle w mit $(v, w) \in E$ gilt: $\max \Psi_{u,w} < \min \Psi_{u,x}$ für den nächsten Knoten x, der aus der Warteschlange Q entnommen wird. Ist die Länge von (u, v, w) zu einem Zeitpunkt t gleich $\Psi_{u,w}(t)$, dann wird eine Abkürzung in den entstehenden Graphen eingefügt.

Sei eine Ordnung $<$ auf der Knotenmenge bereits gegeben. Dann ergibt sich Algorithmus 2.3 zur Berechnung der Contraction Hierarchy. Die Knoten werden dazu aufsteigend nach ihrer Reihenfolge kontrahiert und zwei Teilgraphen $G_\uparrow$ und $G_\downarrow$ zurückgegeben. Der Teilgraph $G_\uparrow$ enthält neben allen Knoten auch alle Kanten, die von einem niedrig zu einem höher in

der Ordnung $<$ eingestuften Knoten führen. Der Teilgraph $G_{\downarrow}$ enthält entsprechend neben allen Knoten die verbleibenden Kanten, also jene, die von einem höher zu einem niedriger in der Ordnung $<$ eingestuften Knoten führen.

Wieviele Abkürzungen in die Contraction Hierarchy eingefügt werden müssen, ist abhängig von der gewählten Ordnung $<$. Das Finden einer Knotenreihenfolge, bei der eine minimale Anzahl von Abkürzungen eingefügt werden, ist $\mathcal{NP}$-schwer [BCK+10]. In der Regel begnügt man sich daher mit Heuristiken zur Findung einer Knotenreihenfolge. Zusätzlich gibt es die Möglichkeit, die Ordnung erst während der endgültigen Kontraktion festzulegen.

Algorithmus 2.3: Contraction Hierarchy Vorberechnung

Eingabe: Graph $G = (V, E, \omega)$, Knotenreihenfolge $<$
Ausgabe: Contraction Hierarchy $\overline{G} = (G_{\uparrow}, G_{\downarrow})$

```
1 G' = (V', E', ω) ← G
2 forall v ∈ V in ascending order by < do
3 |   G' ← Contract(G', v)
4 |   E ← E ∪ E'
5 G_↑ ← (V, {(a, b) ∈ E | a < b}, ω)
6 G_↓ ← (V, (a, b) ∈ E | b < a}, ω)
7 return (G_↑, G_↓)
```

Zur Berechnung eines Profils von s nach t (st-Anfrage) muss nun Dijkstras Algorithmus zum einen auf dem Graphen $G_{\uparrow}$ mit Startknoten s und zum anderen rückwärts auf dem Graphen $G_{\downarrow}$ mit Startknoten t gestartet werden. Alle Knoten $v \in V$, die durch beide gestarteten Dijkstra-Suchen betrachtet wurden, erhalten dadurch die Distanzen $\Psi_{s,v}$ und $\Psi_{v,t}$. Das gesuchte Profil ist dann das Minimum über alle Profile $\Psi_{s,v} \odot \Psi_{v,t}$.

2.3 Algorithmen zur LKW-Fahrer-Scheduling

Für das Scheduling setzen wir auf die Algorithmen zu ähnlichen zeitunabhängigen Schedulingproblemen von Kleff [Kle16] auf. Dort werden verschiedene Phasen, wie das Fahren, das Warten auf den Beginn eines Zeitfenster, das Leisten des Service usw., definiert. Jeder Kunde erhält für jede der Phasen ein Tupel aus mehreren stückweise linearen Funktionen χ. Die Funktionswerte $\chi(t)$ geben an, wie lange seit der letzten Pause gefahren wurde, bzw. wieviel Zeit zu einem Zeitpunkt t seit dem Ende der letzten Pause vergangen ist. Von Phase zu Phase werden die genannten Funktionen sukzessive aus der jeweils vorherigen berechnet (so werden die Funktionen beispielsweise für Zeitpunkte direkt nach einer Pause auf den Wert 0 gesetzt), um am Ende angeben zu können, wann eine Tour frühestens beendet werden kann. Diese Algorithmen für den zeitunabhängigen Fall wurden von Kinz im Rahmen einer Masterarbeit [Kin16] implementiert und werden in dieser Arbeit auf den zeitabhängigen Fall erweitert.

3. Unser Ansatz

Im Folgenden werden in Kapitel 3.1 die nötigen Routenplanungsalgorithmen dargelegt, um in Kapitel 3.2 den Algorithmus zum Scheduling für die Vorgaben der EU zu präsentieren. Anschließend wird eine Beispielberechnung Schritt für Schritt anhand des vorher gezeigten Algorithmus durchgeführt. Zuletzt wird der Algorithmus für die Vorgaben der USA angegeben und die Unterschiede zum Algorithmus für die EU-Vorgaben verdeutlicht.

3.1 Routenplanung

Für die Routenberechnung werden, wie im vorherigen Kapitel beschrieben, Contraction Hierarchies verwendet. Für die Berechnung eines Profils von einem Kunden k_i zum nächsten Kunden k_{i+1} werden dazu die in Kapitel 2.2 vorgestellten Algorithmen verwendet. Da das Scheduling auch Parkplätze einplanen soll, wird ein Algorithmus benötigt, der die Menge aller zwischen zwei Kunden k_i und k_{i+1} erreichbaren Parkplätze $P' \subseteq P$ findet und die Fahrzeitfunktionen $\Psi_{i,p}$ von Kunde k_i zu Parkplatz $p \in P'$ sowie die Fahrzeitfunktion $\Psi_{p,i+1}$ von Parkplatz p zu Kunde k_{i+1} berechnet. Ein Parkplatz p ist zwischen zwei Kunden k_i und k_{i+1} erreichbar, wenn $\Psi_{i,p}(t') \leq R^{driven}$ für einen Zeitpunkt t' und $\Psi_{p,i+1}(t'') \leq R^{driven}$ für einen weiteren Zeitpunkt t''. Umgangssprachlich ist ein Parkplatz p zwischen zwei Kunden also dann erreichbar, wenn folgende Fahrten eines vollständig erholten Fahrers zu einem beliebigen Zeitpunkt ohne Verstoß gegen die betrachteten Lenk- und Ruhezeiten möglich sind: Die direkte Fahrt vom Kunden k_i aus zu Parkplatz p, sowie nach Erreichen des Parkplatzes und Einlegen einer Pause mit Mindestlänge R^{break} die Fahrt von Parkplatz p zum Kunden k_{i+1}. Die Lösung dieses Problems wird im Folgenden beschrieben.

3.1.1 Finden aller erreichbaren Parkplätze

Wir wollen nun zu zwei Kunden k_i und k_{i+1} alle dazwischen erreichbaren Parkplätze und die dazugehörigen Fahrzeitfunktionen finden. Ein einfacher Ansatz ist die Berechnung der Profile von k_i aus zu allen Parkplätzen $p \in P$ und von allen Parkplätzen p zu k_{i+1} einzeln. Profile zu nicht erreichbaren Parkplätzen können anschließend wieder verworfen werden. Im Rahmen dieser Arbeit wurde ein anderer Ansatz gewählt, bei dem alle erreichbaren Parkplätze mit nur zwei Anfragen gefunden werden können. Die Profile Ψ, die der folgende Algorithmus berechnet, können an Stellen mit tatsächlichen Fahrzeiten von mehr als R^{driven} nach oben abweichen. Das ist an dieser Stelle nicht weiter relevant, da diese Stellen vom späteren Schedulingalgorithmus ohnehin nicht berücksichtigt werden.

Für unsere Parkplatzsuche wählen wir für die Berechnung der Contraction Hierarchy eine Knotenreihenfolge $<$, für die alle Parkplätze $p \in P$ erst nach allen anderen Knoten kontrahiert werden. Es gilt dann also $\forall p \in P, v \in V \setminus P : p > v$. Durch diese Restriktion der Knotenreihenfolge ist es möglich, dass bei der Berechnung der Contraction Hierarchy die Anzahl einzufügender Abkürzungen gegenüber der optimalen oder einer „guten“ Knotenreihenfolge ansteigt, und damit die Laufzeit für einfache Profilanfragen erhöht wird. Daher kann es sinnvoll sein, nur folgendes zu fordern: Wähle C mit $P \subseteq C \subseteq V$ und wähle eine Knotenreihenfolge $<$, so dass $\forall c \in C, v \in V \setminus C : c > v$. Wir nennen C den *Kern* (Core). Die Vorberechnung mit Kern C und der modifizierten Knotenreihenfolge $<$ wird wie in Algorithmus 3.1 durchgeführt. Zu beachten ist, dass die Kontraktion gestoppt wird, sobald alle Knoten aus $V \setminus C$ kontrahiert wurden.

Algorithmus 3.1: Contraction Hierarchy Vorberechnung (mit Kern)

Eingabe: Graph $G = (V, E, \omega)$, Kern C, Knotenreihenfolge $<$
Ausgabe: Contraction Hierarchy $\overline{G} = (G_\uparrow, G_\downarrow)$

```
1 G' = (V', E', ω) ← G
2 forall v ∈ V in ascending order by < do
3     if v ∈ C then
4         break
5     G' ← Contract(G', v)
6     E ← E ∪ E'
7 G↑ ← (V, {(a,b) ∈ E | a < b ∨ (a,b) ∈ C × C}, ω)
8 G↓ ← (V, {(a,b) ∈ E | b < a ∨ (a,b) ∈ C × C}, ω)
9 return (G↑, G↓)
```

Um die Profile vom Kunden k_i zu allen Parkplätzen $p \in P$ zu berechnen, wird Dijkstras Algorithmus mit Startknoten k_i auf $G_\uparrow$ gestartet. Abgebrochen werden kann diese Suche, wenn für den nächsten im Algorithmus betrachteten Knoten eine Mindestfahrzeit von R^{driven} erreicht wurde. Nach der Ausführung enthält $\Psi_{k_i,p}$ das Profil von k_i nach p für alle p, die erreichbar sind. Durch eine Rückwärtssuche ausgehend von k_{i+1} auf $G_\downarrow$ können die Profile von allen erreichbaren Parkplätzen zum nächsten Kunden gefunden werden. In der Schnittmenge der gefundenen Parkplätze beider Suchen können sich Parkplätze befinden, die nicht erreichbar sind. Diese werden anschließend verworfen.

3.1.2 Graph für *st*- und Parkplatzanfragen

Da der Kern zwar für die Parkplatzanfragen nötig ist, aber die Laufzeit für *st*-Anfragen erhöht und für beide Anfragen nicht zwei verschiedene Graphen (einer mit Kern, einer ohne Kern) verwendet werden sollen, nutzen wir folgendes: Wir wählen für den Graphen $G = (V, E)$ wieder einen Kern C und eine entsprechende Knotenreihenfolge $<$ und kontrahieren wie im Algorithmus 2.3 ursprünglich angegeben. Wir erhalten so die Contraction Hierarchy $(G_\uparrow = (V, E_\uparrow, \omega), G_\downarrow = (V, E_\downarrow, \omega))$. Die Menge aller während der Vorberechnung in den Aufwärtsgraphen $G_\uparrow$ eingefügten Abkürzungen innerhalb des Kerns $(u, v) \in E_\uparrow$ mit $u \in C$ und $v \in C$ bezeichnen wir mit $A_\uparrow$, analog für $A_\downarrow$. Außerdem sei $E_c = \{(u, v) \in E \mid (u, v) \in C \times C\}$ die Menge aller Kanten im Kern des ursprünglichen Graphen G vor der Vorberechnung. Wir setzen $E'_\uparrow = E_\uparrow \cup E_c$ und entsprechend $E'_\downarrow = E_\downarrow \cup E_c$.

Für *st*- und Parkplatzanfragen nutzen wir den Graphen $(V, E'_\uparrow, \omega)$ für die Vorwärtssuchen und $(V, E'_\downarrow, \omega)$ für die Rückwärtssuchen. Bei einer *st*-Anfrage werden nur Kanten aus $E'_\uparrow \setminus E_c$, bzw. $E'_\downarrow \setminus E_c$ berücksichtigt. Bei einer Parkplatzanfrage berücksichtigen wir nur Kanten aus $E'_\uparrow \setminus A_\uparrow$, bzw. $E'_\downarrow \setminus A_\downarrow$.

3.1.3 Beschleunigung der Profilberechnung durch Approximation

Sei K die Gesamtanzahl der Stützpunkte der Fahrzeitfunktionen aller Kanten im Graphen. Foschini et al. zeigen, dass ein Profil von einem Knoten s zu einem anderen Knoten t bis zu $K \cdot |V|^{\Theta(\log |V|)}$ Stützpunkte haben kann [FHS14]. Um die Laufzeit der Profilberechnung zu verbessern, können wir die Gesamtanzahl K der Stützpunkte im Graphen verringern. Wir approximieren dazu die Fahrzeitfunktionen des Graphen innerhalb gesetzter Fehlergrenzen, um für jede Funktion eine möglichst kleine Anzahl von Stützstellen zu erhalten. Zwar verlieren wir dadurch die Exaktheit der berechneten Profile, wir werden später allerdings experimentell zeigen, dass diese Ungenauigkeit bei entsprechend gewählten Fehlergrenzen der Approximation tolerierbar ist. Zugleich bringt dieses Vorgehen einen signifikanten Zuwachs der Berechnungsgeschwindigkeit mit sich. Sei dazu φ die zu approximierende Funktion. Sei weiterhin $\varepsilon \in [0, 1]$ der maximale relative Approximationsfehler. Gesucht ist dann eine Funktion ψ mit minimaler Anzahl Stützstellen, für die gilt: $\forall x : \varphi(x)/(1+\varepsilon) \leq \psi(x) \leq (1+\varepsilon) \cdot \varphi(x)$ [BGNS10]. Dazu wird im Rahmen dieser Arbeit der Algorithmus von Imai und Iri [II87] verwendet. Ist $\varepsilon = 0$, so werden die Funktionen nicht approximiert. Wir approximieren alle Fahrzeitfunktionen ω des Graphen G nach der Berechnung der Contraction Hierarchy und berechnen die gesuchten Profile dann mit den approximierten Funktionen.

3.2 LKW-Fahrer-Scheduling

Im Folgenden wird der Algorithmus zur Berechnung des frühesten Endes eines Zeitplans für die EU-Regelung erläutert. Der Algorithmus wird zuerst beschrieben und in Pseudocode dargelegt, und dann mit einem konkreten Beispiel verdeutlicht. Weiterhin wird erklärt, wie dieser Algorithmus erweitert werden kann, um nicht nur das Ende eines optimalen Zeitplans zu finden, sondern auch einen solchen Zeitplan selbst. Danach werden wir in Kapitel 3.2.4 kurz auf den Algorithmus für die Vorgaben der USA eingehen.

Der folgende Algorithmus für das Scheduling innerhalb der EU betrachtet ausschließlich, dass nach spätestens $R^{driven} = 4,5\,\text{h}$ eine Pause von mindestens $R^{break} = 45\,\text{min}$ eingelegt wird. Außerdem wird zwischen je zwei Kunden immer nur maximal ein Parkplatz eingeplant, an dem eine Pause mit einer Dauer von mindestens R^{break} eingelegt wird.

Der Algorithmus besteht aus zwei Schritten, die für alle Kunden nacheinander wiederholt werden. Im ersten Schritt befindet sich der LKW bei dem aktuellen Kunden k_i und wartet, bis sich ein Zeitfenster dieses Kunden öffnet, um dann den Service durchzuführen. Im zweiten Schritt wird dann vom aktuellen zum nächsten Kunden k_{i+1} gefahren. Die Fahrt führt entweder direkt von Kunde k_i zu k_{i+1} oder über einen Parkplatz, an dem immer mindestens eine volle Pause eingelegt wird.

Um zu jeder Zeit zu wissen, wie lange der LKW-Fahrer seit der letzten Pause gefahren ist, nutzen wir für jeden Schritt eine zeitabhängige Funktion $\chi_i^j, i \in \{1, \ldots, n\}, j \in \{\textit{on arrival}, \textit{after service}\}$. Die Indizes i und j geben an, dass der LKW-Fahrer sich gerade bei Kunde k_i befindet und auf den Beginn des Service wartet ($j = \textit{on arrival}$) oder den Service schon beendet hat ($j = \textit{after service}$). Aus $\chi_i^{\textit{on arrival}}$ berechnet der Algorithmus $\chi_i^{\textit{after service}}$ und aus $\chi_i^{\textit{after service}}$ wird $\chi_{i+1}^{\textit{on arrival}}$ berechnet. Der Wert $\chi_i^j(t)$ gibt an, wie lange der LKW-Fahrer seit der letzten Pause mindestens gefahren sein muss, um zum Zeitpunkt t an Kunde k_i bei Schritt j zu sein. Ist $\chi_i^j(t) = \bot$, so bedeutet dies, dass der Fahrer aufgrund der Fahr-, Pausen- und Servicezeiten sowie der Kundenzeitfenster nicht zum Zeitpunkt t am Kunden k_i bei Schritt j sein kann. Am Ende des Algorithmus gibt die erste definierte Stelle der letzten Funktion $\alpha(\chi_n^{\textit{after service}})$ an, wann der Fahrer die Tour frühestens beenden kann.

Algorithmus 3.2: EU-SCHEDULING für einen Planungshorizont von einem Tag mit maximal einem Parkplatz zwischen zwei Kunden

Daten: $\Psi_{x,y}(t)$ Fahrzeit von x nach y bei einer Abfahrt zum Zeitpunkt t
$\overline{\Psi}_{x,y}(t)$ Fahrzeit von x nach y bei einer Ankunft zum Zeitpunkt t
n Anzahl Kunden
t_0 Ende der letzten Tätigkeit (Service leisten oder fahren) der vorherigen Tour des Fahrers
c gefahrene Zeit zum Zeitpunkt t_0 seit dem Ende der letzten Pause
W_i Zeitfenster für den Beginn des Service bei Kunde k_i
S_i Servicedauer bei Kunde k_i
P Menge aller Parkplätze (ohne Kunden)
R^{driven} maximale akkumulierte Fahrzeit zwischen zwei Pausen
R^{break} minimale Pausenzeit

Ausgabe: frühestes Ende der Tour oder $\perp$, falls die Tour nicht erfüllbar ist

1 $\chi_1^{on\ arrival}(t) \leftarrow \begin{cases} \perp, & t < t_0 \\ c, & \text{otherwise} \end{cases}$
2 **for** $i \leftarrow 1$ **to** n **do**
3 $\quad \chi_i^{on\ arrival}(t) \leftarrow \begin{cases} \chi_i^{on\ arrival}(t), & t < \alpha(\chi_i^{on\ arrival}) + R^{break} \\ 0, & \text{otherwise} \end{cases}$
$\quad$ `// Wait and perform service`
4 $\quad t_{begin} \leftarrow \min\{t \in W_i \mid t \geq \alpha(\chi_i^{on\ arrival})\}$
5 $\quad \tau(t) \leftarrow \begin{cases} \perp, & t < t_{begin} + S_i \\ \max\{t' \in W_i \mid t_{begin} \leq t' \leq t - S_i\}, & \text{otherwise} \end{cases}$
6 $\quad \chi_i^{after\ service}(t) \leftarrow \chi_i^{on\ arrival}(\tau(t))$
7 $\quad$ **if** $i = n$ **then**
8 $\quad\quad$ **return** $\alpha(\chi_n^{after\ service})$
9 $\quad \chi_i^{after\ service}(t) \leftarrow \begin{cases} \chi_i^{after\ service}(t), & t < \alpha(\chi_i^{after\ service}) + R^{break} \\ 0, & \text{otherwise} \end{cases}$
$\quad$ `// Drive and take break`
10 $\quad dep_i \leftarrow \min\{t \geq \alpha(\chi_i^{after\ service}) \mid \chi_i^{after\ service}(t) + \Psi_{i,i+1}(t) \leq R^{driven}\}$
11 $\quad arr_{i+1} \leftarrow dep_i + \Psi_{i,i+1}(dep_i)$
12 $\quad \chi_{i+1}^{on\ arrival}(t) \leftarrow \min_{arr_{i+1} \leq t' \leq t} \chi_i^{after\ service}(t' - \overline{\Psi}_{i,i+1}(t')) + \overline{\Psi}_{i,i+1}(t')$
13 $\quad$ **for** $p \in P$ **do**
14 $\quad\quad dep_i \leftarrow \min\{t \geq \alpha(\chi_i^{after\ service}) \mid \chi_i^{after\ service}(t) + \Psi_{i,p}(t) \leq R^{driven}\}$
15 $\quad\quad arr_p \leftarrow dep_i + \Psi_{i,p}(dep_i)$
16 $\quad\quad dep_p \leftarrow \min\{t \geq arr_p + R^{break} \mid \Psi_{p,i+1}(t) \leq R^{driven}\}$
17 $\quad\quad arr_{i+1} \leftarrow dep_p + \Psi_{p,i+1}(dep_p)$
18 $\quad\quad \chi_{i+1}^{on\ arrival}(t) \leftarrow \min\{\chi_{i+1}^{on\ arrival}(t), \min_{arr_{i+1} \leq t' \leq t} \overline{\Psi}_{p,i+1}(t')\}$

Die Beschreibung erfolgt nun anhand von Algorithmus 3.2. Ganz zu Anfang des Algorithmus in Zeile 1 setzen wir $\chi_1^{on\ arrival}(t) = \perp$ für $t < t_0$, da der LKW-Fahrer erst zum Zeitpunkt t_0 seine letzte Tätigkeit der vorherigen Tour beendet hat, und somit dann bereit für die aktuelle Tour ist. Der Fahrer ist zum Zeitpunkt t_0 bereits c seit Ende der letzten Pause gefahren. Für $t \geq t_0$ setzen wir daher $\chi_1^{on\ arrival}(t) = c$. Für jeden Kunden k_i wird nun nacheinander folgendes durchgeführt:

Der Fahrer befindet sich bei Kunde k_i. Vor der Durchführung des Service kann der Fahrer eine Pause einlegen, siehe in Zeile 3 im Algorithmus. Beginnen kann diese zum Zeitpunkt $\alpha(\chi_i^{on\ arrival})$, beendet ist sie dann um $\alpha(\chi_i^{on\ arrival}) + R^{break}$, also die Pausendauer später. Für $t \geq \alpha(\chi_i^{on\ arrival}) + R^{break}$ ist dann $\chi_i^{on\ arrival}(t) = 0$, da gerade eine Pause eingelegt und seitdem nicht gefahren wurde.

Wir werden nun die Berechnung von $\chi_i^{after\ service}$ aus $\chi_i^{on\ arrival}$, also das Warten auf das Öffnen eines Zeitfensters und das Durchführen des Service mit Dauer S_i, herleiten. Zuerst berechnen wir den frühesten Zeitpunkt t_{begin}, zu dem der Service begonnen werden kann. Anschließend berechnen wir die Funktionswerte der Funktion $\chi_i^{after\ service}$.

Der Service kann nur beginnen, wenn der LKW am aktuellen Kunden angekommen ist, also für Zeitpunkte $t \geq \alpha(\chi_i^{on\ arrival})$. Außerdem kann der Service nur dann beginnen, wenn ein Zeitfenster geöffnet ist, also wenn $t \in W_i$. Insgesamt ergibt sich der Zeitpunkt t_{begin}, zu dem der Service frühestens beginnen kann, folgendermaßen: $t_{begin} = \min\{t \in W_i \mid t \geq \alpha(\chi_i^{on\ arrival})\}$. Das frühestmögliche *Ende* des Service ist entsprechend um die Servicezeit S_i später.

Wir setzen $\chi_i^{after\ service}(t) = \perp$ für $t < t_{begin} + S_i$, denn für die Zeitpunkte t kann der Service noch nicht beendet worden sein. Der Funktionswert $\chi_i^{after\ service}(t)$ mit $t \geq t_{begin} + S_i$ ergibt sich nun folgendermaßen aus $\chi_i^{on\ arrival}$: Da während des Service nicht gefahren wird, handelt sich nur um eine Verschiebung der Form $\chi_i^{after\ service}(t) = \chi_i^{on\ arrival}(\tau(t))$. Die Funktion τ bildet dabei die Endzeitpunkte des Service t auf den optimalen Zeitpunkt für den Beginn des Service $\tau(t)$ ab. Der Service muss spätestens zum Zeitpunkt $t - S_i$ begonnen werden, sonst wäre der Service zum Zeitpunkt t noch nicht beendet. Außerdem kann der Service frühestens zum Zeitpunkt t_{begin} beginnen. Zusätzlich kann der Service aber auch nur dann beginnen, wenn auch ein Servicefenster geöffnet ist, also insgesamt nur zu den Zeitpunkten in $\{t' \in W_i \mid t_{begin} \leq t' \leq t - S_i\}$. Der letztmögliche Zeitpunkt $\underline{t} = \max\{t' \in W_i \mid t_{begin} \leq t' \leq t - S_i\}$ für den Beginn des Service ist dabei derjenige mit der kürzesten gefahrenen Zeit seit der letzten Pause.

Wäre für ein Zeitpunkt t' mit $t_{begin} \leq t' < \underline{t}$ die gefahrene Zeit seit der letzten Pause kürzer als zum Zeitpunkt $\underline{t}$, also $\chi_i^{on\ arrival}(t') < \chi_i^{on\ arrival}(\underline{t})$, dann hätte der Fahrer auch so fahren können, dass er am Kunden k_i zum Zeitpunkt t' ankommt, um ab dann bis $\underline{t}$ zu warten, wodurch $\chi_i^{on\ arrival}(t') \geq \chi_i^{on\ arrival}(\underline{t})$ wäre, was einen Widerspruch darstellt.

Es ergibt sich also: $\chi_i^{after\ service}(t) \coloneqq \chi_i^{on\ arrival}(\underline{t})$. Im Algorithmus ist dies in den Zeilen 5 – 6 umgesetzt. Sind wir jetzt bereits am Ende der Tour, also bei Kunde k_n, so ist das frühestmögliche Ende des Zeitplans gleich dem frühestmöglichen Ende des Service beim letzten Kunden, also $t_{end} = \alpha(\chi_n^{after\ service})$. Sind wir noch nicht am Ende der Tour, so kann an dieser Stelle ebenfalls wieder eine Pause eingelegt werden, siehe die obige Beschreibung der Pause und Zeile 9 des Algorithmus.

Im nächsten Schritt muss der Fahrer zum nächsten Kunden k_{i+1} fahren. Dazu berechnen wir aus $\chi_i^{after\ service}$ die Funktion $\chi_{i+1}^{on\ arrival}$. Zuerst betrachten wir dazu den direkten Weg von Kunde k_i zu Kunde k_{i+1} und erhalten so eine vorläufige Funktion $\chi_{i+1}^{on\ arrival}$ (siehe Zeilen 10 – 12), die dann durch die Fahrt von k_i nach k_{i+1} über Parkplätze $p \in P$ aktualisiert wird (Zeilen 13 – 18). Nach wie vor werden vom Algorithmus zwischen zwei Kunden allerdings nur Fahrten betrachtet, die maximal über einen Parkplatz führen.

Für den direkten Weg berechnen wir die frühestmögliche Ankunftszeit arr_{i+1} am Kunden k_{i+1}, dann berechnen wir die vorläufigen Werte der Funktion $\chi_{i+1}^{on\ arrival}$. Für die Fahrzeitfunktion $\Psi_{i,i+1}$ von k_i nach k_{i+1} berechnen wir den ersten Zeitpunkt, zu dem wir abfahren können, ohne die Pausenregel zu verletzen. Dieser Zeitpunkt $dep_i \geq \alpha(\chi_i^{after\ service})$ ist der erste Zeitpunkt t, zu dem die bisher gefahrene Zeit seit der letzten Pause $\chi_i^{after\ service}(t)$ zusammen mit der Fahrzeit zum nächsten Kunden $\Psi_{i,i+1}(t)$ kleiner ist, als die maximale Fahrzeit ohne Pause R^{driven}. Es ergibt sich also $dep_i = \min\{t \geq \alpha(\chi_i^{after\ service}) \mid \chi_i^{after\ service}(t) + \Psi_{i,i+1}(t) \leq R^{driven}\}$. Wir gehen zunächst davon aus, dass ein solcher Zeitpunkt existiert. Den Fall $dep_i = \bot$ betrachten wir später. Ist also eine frühestmögliche Abfahrt dep_i gegeben, dann berechnet sich die frühestmögliche Ankunft arr_{i+1} durch $arr_{i+1} = dep_i + \Psi_{i,i+1}(dep_i)$, denn eine spätere Abfahrt nach dep_i führt aufgrund der FIFO*-Eigenschaft von $\Psi_{i,i+1}$ nicht zu einer früheren Ankunft.

Wir wollen nun für $t \geq arr_{i+1}$ einen vorläufigen Wert $\chi_{i+1}^{on\ arrival}(t)$ aus $\chi_i^{after\ service}$ berechnen, der durch eine direkte Fahrt ohne Pause von k_i nach k_{i+1} entsteht. Dazu verwenden wir die Fahrzeitfunktion $\Psi_{i,i+1}$, welche die Abfahrtszeiten von Kunde k_i auf die Fahrtdauer zu Kunde k_{i+1} abbildet, und die Fahrzeitfunktion $\overline{\Psi}_{i,i+1}$, welche die Ankunftszeiten bei Kunde k_{i+1} auf die Fahrtdauer abbildet. Um bei Kunde k_{i+1} zum Zeitpunkt t zu sein, muss die Fahrt zu einem Zeitpunkt $t' \in [dep_i, t - \overline{\Psi}_{i,i+1}(t)]$ begonnen werden. Kommt der Fahrer zeitlich vor t an, so kann er die restliche Zeit warten. Daraus ergeben sich die möglichen Ankunftszeiten $\bar{t} \in [arr_{i+1}, t]$. Die bisher gefahrene Zeit seit der letzten Pause beträgt bei einer Ankunft zum Zeitpunkt $\bar{t}$ dann $\chi_i^{after\ service}(\bar{t} - \overline{\Psi}_{i,i+1}(\bar{t})) + \overline{\Psi}_{i,i+1}(\bar{t})$. Dies berechnet sich aus der bisher gefahrenen Zeit seit der letzten Pause bei Abfahrt um $\bar{t} - \overline{\Psi}_{i,i+1}(\bar{t})$ und der Fahrzeit zum Kunden k_{i+1}. Aus allen möglichen Ankunftszeiten $\bar{t}$ wählen wir diejenige, die zur kürzesten Fahrzeit seit der letzten Pause führt. Damit setzen wir vorläufig $\chi_{i+1}^{on\ arrival}(t) = \min_{arr_{i+1} \leq \bar{t} \leq t} \chi_i^{after\ service}(\bar{t} - \overline{\Psi}_{i,i+1}(\bar{t})) + \overline{\Psi}_{i,i+1}(\bar{t})$.

Sollte für eine Ankunftszeit $\widetilde{t} \in [arr_{i+1}, t]$ gelten, dass $\chi_i^{after\ service}(\widetilde{t} - \overline{\Psi}_{i,i+1}(\widetilde{t})) + \overline{\Psi}_{i,i+1}(\widetilde{t}) > R^{driven}$, überschreitet also die gefahrene Zeit die maximal erlaubte Fahrzeit, so ist dies an dieser Stelle nicht relevant, denn dann wird aufgrund der Bildung des Minimums der Wert einer Fahrt mit einer früheren Ankunftszeit $t'' \in [arr_{i+1}, \widetilde{t}]$ gewählt, der die Pausenregeln nicht verletzt. Ein solcher Zeitpunkt t'' existiert, denn $\chi_i^{after\ service}(arr_{i+1} - \overline{\Psi}_{i,i+1}(arr_{i+1})) + \overline{\Psi}_{i,i+1}(arr_{i+1}) \leq R^{driven}$.

Ist $dep_i = \bot$ (ist also eine direkte Fahrt von Kunde k_i zu Kunde k_{i+1} durch die Lenk- und Ruhezeiten zu keiner Zeit möglich), so folgt auch $arr_{i+1} = \bot$. Damit ist $\chi_{i+1}^{on\ arrival}(t) = \min_{\bar{t} \in \emptyset} \chi_i^{after\ service}(\bar{t} - \overline{\Psi}_{i,i+1}(\bar{t})) + \overline{\Psi}_{i,i+1}(\bar{t}) = \bot$.

Zur Berechnung der abschließenden Werte $\chi_{i+1}^{on\ arrival}(t)$ wird nun für eine Fahrt über die Parkplätze $p \in P$ der früheste Ankunftszeitpunkt am Parkplatz p sowie am Kunden k_{i+1} berechnet, um dann die Funktionswerte $\chi_{i+1}^{on\ arrival}(t)$ zu aktualisieren. Sei also $p \in P$ der aktuell betrachtete Parkplatz. Die früheste Ankunftszeit arr_p an p berechnet sich analog zur Ankunftszeit an k_{i+1} bei einer direkten Fahrt ohne Pause und soll daher an dieser Stelle nicht näher betrachtet werden. Kommt der LKW-Fahrer zum Zeitpunkt arr_p am Parkplatz p an, so macht er dort eine Pause, die zum Zeitpunkt $arr_p + R^{break}$ beendet ist. Nach dieser Pause kann der LKW-Fahrer wieder die volle Zeit R^{driven} ohne eine weitere Pause fahren. Für den frühesten Abfahrtszeitpunkt dep_p vom Parkplatz p suchen wir also den Zeitpunkt, bei dem die Fahrzeit zum nächsten Kunden kleiner ist als R^{driven}, also $dep_p = \min\{t \geq arr_p + R^{break} \mid R^{driven} \geq \Psi_{p,i+1}(t)\}$. Für eine Fahrt über den Parkplatz p ergibt sich bei Ankunftszeiten $\bar{t} \geq arr_{i+1} = dep_p + \Psi(dep_p)$ bei Kunde k_{i+1} somit eine gefahrene Zeit seit der letzten Pause von $\overline{\Psi}_{p,i+1}(\bar{t})$. Wir setzen $\chi_i^{after\ service}$ somit auf das Minimum der bisherigen Funktion $\chi_i^{after\ service}$ und der Funktion $\min_{arr_p \leq \bar{t} \leq t} \overline{\Psi}_{p,i+1}(\bar{t})$, da

der Fahrer für eine Ankunft zum Zeitpunkt t früher als nötig fahren und ab dann bis t warten kann.

Ist eine Fahrt von Kunde k_i zu einem Parkplatz p aufgrund der Lenk- und Ruhezeiten zu keinem Zeitpunkt möglich, so ist analog zu einer direkten Fahrt von Kunde k_i zum nächsten $dep_i = \bot$. Ist eine Fahrt von einem Parkplatz p zu einem Kunden k_{i+1} nicht möglich, so ist $dep_p = \bot$. In beiden Fällen folgt $\min_{\bar{t} \in \emptyset} \overline{\Psi}_{p,i+1}(\bar{t}) = \bot$ und damit bleibt Zeile 18 im Algorithmus ohne Effekt.

3.2.1 Beispielberechnung

Zur Verdeutlichung des Algorithmus 3.2 folgt an dieser Stelle ein Beispiel für eine Tour mit zwei Kunden und genau einem Parkplatz $p \in P$. Das Ende der vorherigen Tour sei zum Zeitpunkt $t_0 = 0$, die zum Zeitpunkt t_0 gefahrene Zeit seit der letzten Pause sei $c = 0$. Außerdem nehmen wir

1. bei Kunde k_1 eine Servicezeit $S_1 = 500$ und Zeitfenster $W_1 = [500, 1000] \cup [1500, 2500]$ und
2. bei Kunde k_2 eine Servicezeit $S_2 = 2000$ und Zeitfenster $W_2 = [17000, 20000]$ an.

Die für dieses Beispiel relevanten Bereiche der Fahrzeitfunktionen $\Psi_{1,2}$, $\Psi_{1,p}$ und $\Psi_{p,2}$ sind in Abbildung 3.4, 3.6 und 3.7 dargestellt. Den frühesten definierten Zeitpunkt einer Funktion χ stellen wir in den Schaubildern durch $\times$ dar. Die Werte der Koordinatenachsen sind in Sekunden.

Wir setzen also $\chi_1^{on\ arrival}(t) = 0$ für $t \geq t_0$, denn wir gehen davon aus, dass der Fahrer vollständig erholt und seit der letzten Pause nicht gefahren ist (da $c = 0$). Das Schaubild der Funktion $\chi_1^{on\ arrival}$ befindet sich in Abbildung 3.1.

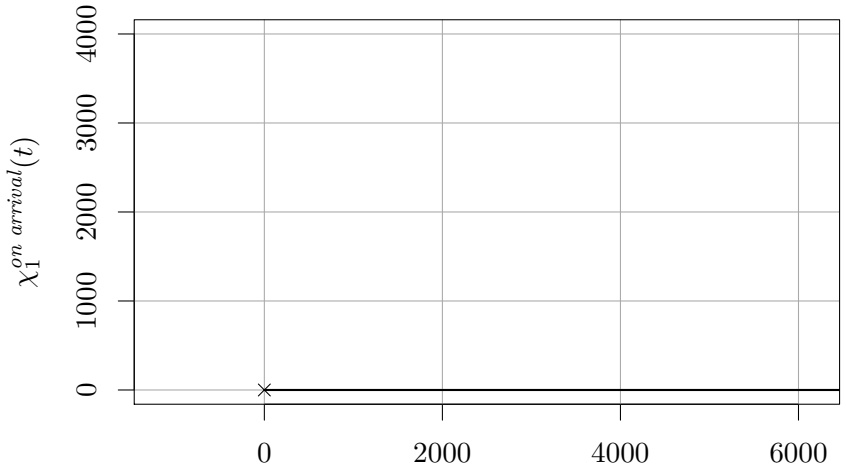

Abbildung 3.1: Schaubild der Funktion $\chi_1^{on\ arrival}$.

Wir betrachten nun die Schleife des Algorithmus in Zeile 2 für $i = 1$. Da der Fahrer bereits vollständig erholt und noch nicht gefahren ist, hat Zeile 3 zur Durchführung einer Pause keinen Effekt. Kunde k_1 hat die Zeitfenster $W_1 = [500, 1000] \cup [1500, 2500]$, in denen der Service mit Dauer $S_1 = 500$ beginnen kann. Daraus ergibt sich die Funktion τ wie in Abbildung 3.2. Damit kann die Funktion $\chi_1^{after\ service}$ berechnet werden. Die Funktionen $\chi_1^{after\ service}$ und $\chi_1^{on\ arrival}$ unterscheiden sich nur dadurch, dass $\chi_1^{on\ arrival}$ bereits ab dem Zeitpunkt $t_0 = 0$, und die Funktion $\chi_i^{after\ service}$ erst ab $\alpha(\tau) = 1000$ definiert ist, siehe Abbildung 3.3. Dies ist ein Resultat der Servicedauer $S_1 = 500$ und des Zeitfensters, durch

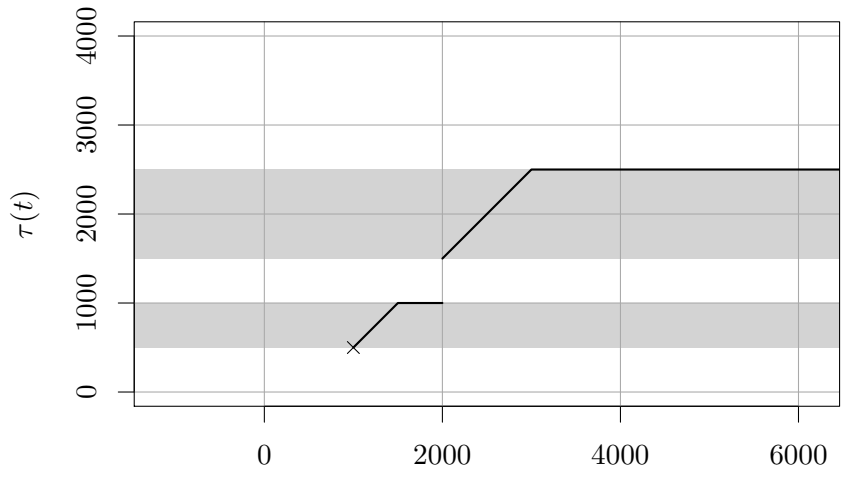

Abbildung 3.2: Schaubild der Funktion τ bei Kunde k_1. (Die Zeitfenster W_1 des Kunden k_1 sind grau markiert.)

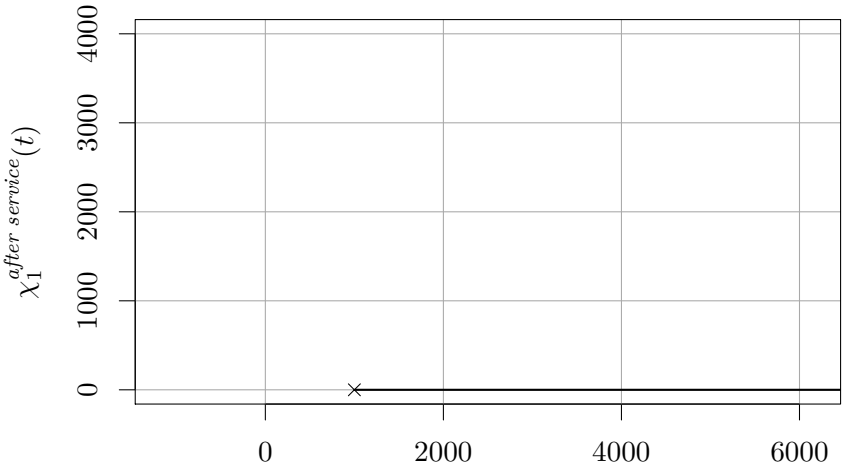

Abbildung 3.3: Schaubild der Funktion $\chi_1^{after\ service}$.

das der Service erst um 500 beginnen, und entsprechend erst um 1000 fertiggestellt sein kann. Die Pause, die der Algorithmus an dieser Stelle einfügt, hat wie schon zuvor keinen Effekt.

Wir berechnen nun, wie im Algorithmus angegeben, zuerst eine vorläufige Funktion $\chi_2^{on\ arrival}$. Dazu nehmen wir eine Fahrzeitfunktion $\Psi_{1,2}$ von Kunde k_1 zu Kunde k_2 wie in Abbildung 3.4 an. Nach Zeile 10 des Algorithmus ergibt sich der früheste Abfahrtszeitpunkt von Kunde k_1 zu $dep_1 = 2000$. Früher kann nicht gefahren werden, weil sonst länger als $R^{driven} = 16200\,\mathrm{s} = 4{,}5\,\mathrm{h}$ gefahren werden müsste, was aufgrund der Pausenregeln verboten ist. Daraus berechnet sich der früheste Ankunftszeitpunkt arr_2 an Kunde k_2 wie im Algorithmus, also $arr_2 = dep_1 + \Psi_{1,2}(dep_1) = 2000 + 16200 = 18200$. Für $\chi_2^{on\ arrival}$ ergeben sich dann vorläufig für $t \geq arr_2$ folgende Funktionswerte: $\chi_2^{on\ arrival}(t) = \min_{arr_2 \leq t' \leq t} \chi_1^{after\ service}(t' - \overline{\Psi}_{1,2}(t')) + \overline{\Psi}_{1,2}(t') = \min_{arr_2 \leq t' \leq t} \overline{\Psi}_{1,2}(t')$. Sowohl $\overline{\Psi}_{1,2}(t')$ als auch die vorläufigen Werte $\chi_2^{on\ arrival}(t) = \min_{arr_2 \leq t' \leq t} \overline{\Psi}_{1,2}(t')$ sind in Abbildung 3.5 dargestellt.

Nun wollen wir die Fahrt über den Parkplatz $p \in P$ betrachten. Die Fahrzeit $\Psi_{1,p}$ ist in Abbildung 3.6 dargestellt. Da bisher noch nicht gefahren wurde und für $t = \alpha(\chi_1^{after\ service})$ der Wert $\chi_1^{after\ service}(t) + \Psi_{1,p}(t) = \Psi_{1,p}(t)$ kleiner ist als die maximale Fahrzeit $R^{driven} =$

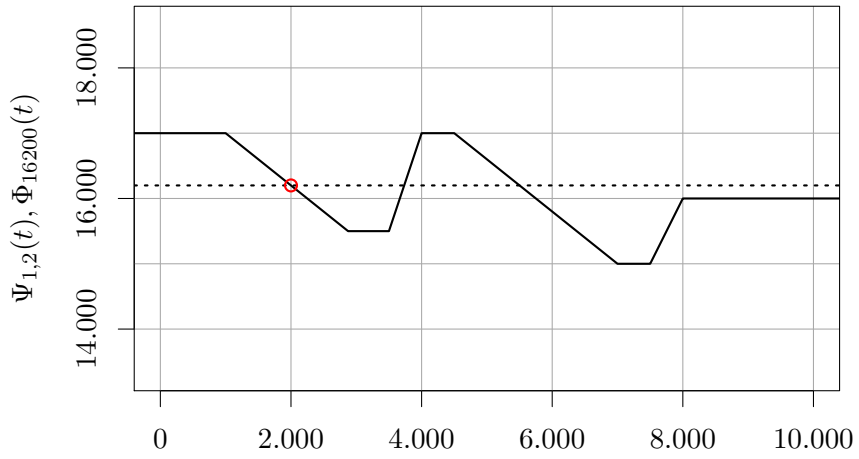

Abbildung 3.4: Schaubild der Fahrzeitfunktionen $\Psi_{1,2}$, der Funktion Φ_{16200} (gepunktet) und der früheste Schnittpunkt beider Funktionen (rot).

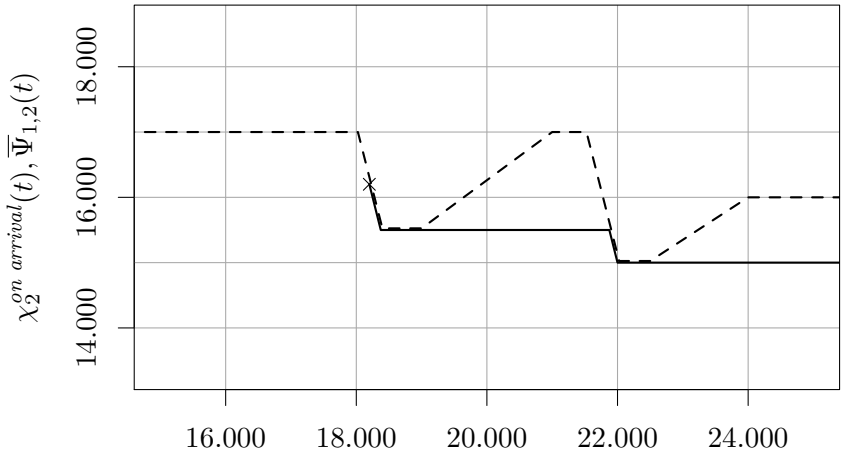

Abbildung 3.5: Schaubild der vorläufigen Funktion $\chi_2^{on\ arrival}$ sowie der Fahrzeitfunktion $\overline{\Psi}_{1,2}$ (gestrichelt).

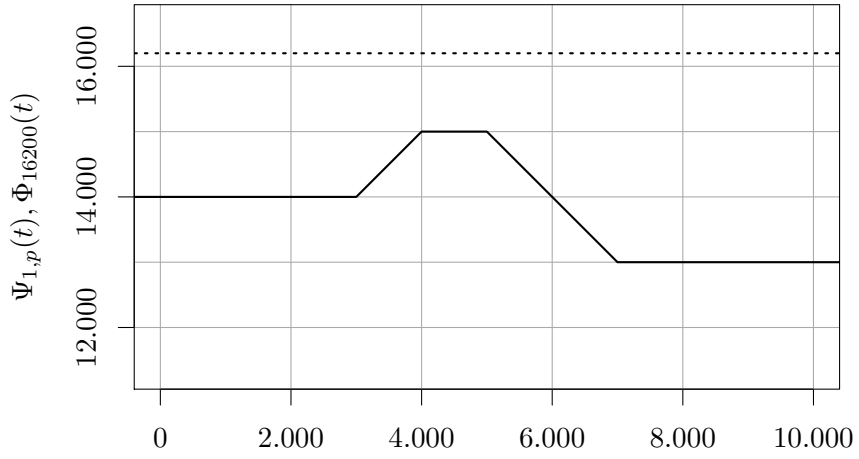

Abbildung 3.6: Schaubild der Fahrzeitfunktion $\Psi_{1,p}$ und der Funktion Φ_{16200} (gepunktet).

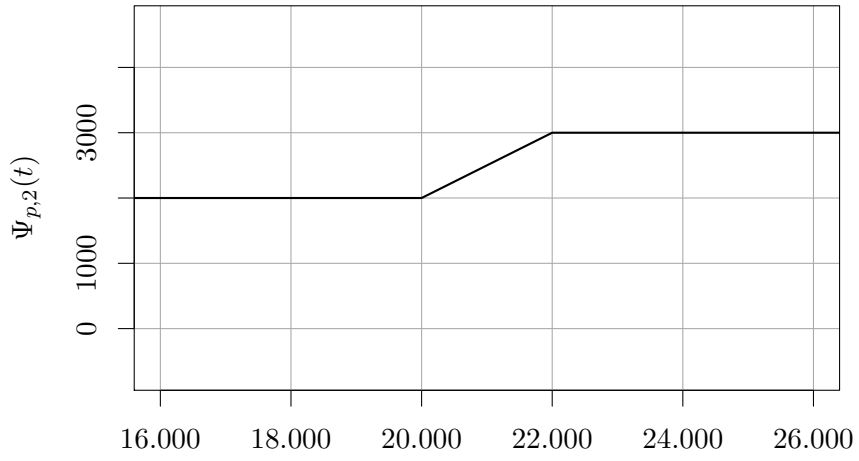

Abbildung 3.7: Schaubild der Fahrzeitfunktion $\Psi_{p,2}$.

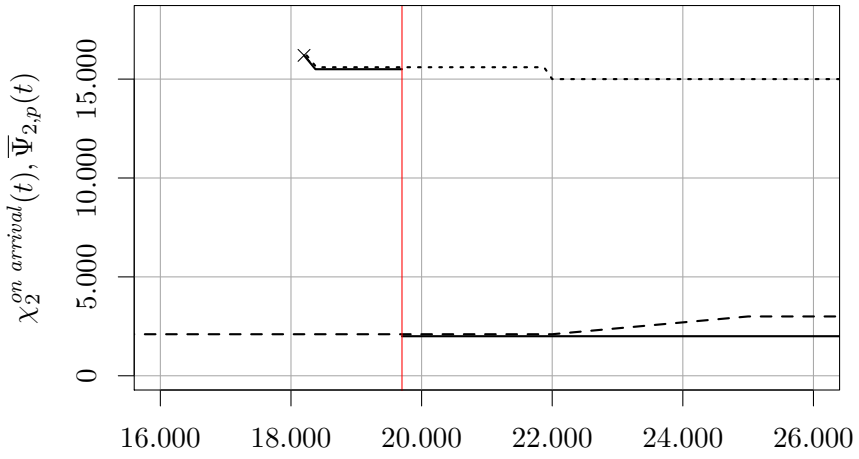

Abbildung 3.8: Schaubild der Funktion $\chi_2^{on\ arrival}$, der vorherigen vorläufigen Funktion $\chi_2^{on\ arrival}$ (gepunktet), der Fahrzeitfunktion $\overline{\Psi}_{2,p}$ (gestrichelt) und der Ankunftszeit arr_2 bei einer Fahrt über Parkplatz p (rot).

16200 s = 4,5 h, kann zum Parkplatz p sofort um $dep_1 = \alpha(\chi_1^{after\ service}) = 1000$ losgefahren werden. Der früheste Ankunftszeitpunkt arr_p an Parkplatz p errechnet sich dann durch $arr_p = dep_1 + \Psi_{1,p}(dep_1) = 1000 + 14000 = 15000$. Nach der Pause auf Parkplatz p mit einer Dauer von $R^{break} = 2700\,\text{s} = 45\,\text{min}$ kann zu Kunde k_2 gefahren werden.

Die entsprechende Fahrzeitfunktion $\Psi_{p,2}$ ist in Abbildung 3.7 dargestellt. Da auch die Funktionswerte $\Psi_{p,2}(t)$ alle kleiner als R^{driven} sind, kann sofort nach der Pause, also um $dep_p = arr_p + R^{break} = 15000 + 2700 = 17700$ zum Kunden k_2 weitergefahren werden. Der früheste Ankunftszeitpunkt bei Kunde k_2, falls über Parkplatz p gefahren wird, beträgt damit $arr_2 = dep_p + \Psi_{p,2}(dep_p) = 17700 + 2000 = 19700$. Damit ergeben sich die neuen Funktionswerte $\chi_2^{on\ arrival}(t)$ aus dem Minimum der bisherigen Werte und $\min_{arr_2 \le t' \le t} \overline{\Psi}_{p,2}(t')$ wie in Abbildung 3.8. Zusätzlich sind in der Abbildung die vorherigen Funktionswerte von $\chi_2^{on\ arrival}$ sowie die Fahrzeitfunktion $\overline{\Psi}_{p,2}$ dargestellt.

Nachdem wir alle möglichen Fahrwege von Kunde k_1 zu Kunde k_2 betrachtet haben, betrachten wir nun die Schleife des Algorithmus für $i = 2$. Zuerst wird eine mögliche Pause bei Kunde k_2 eingefügt, die wir direkt nach der frühesten Ankunft dort beginnen können.

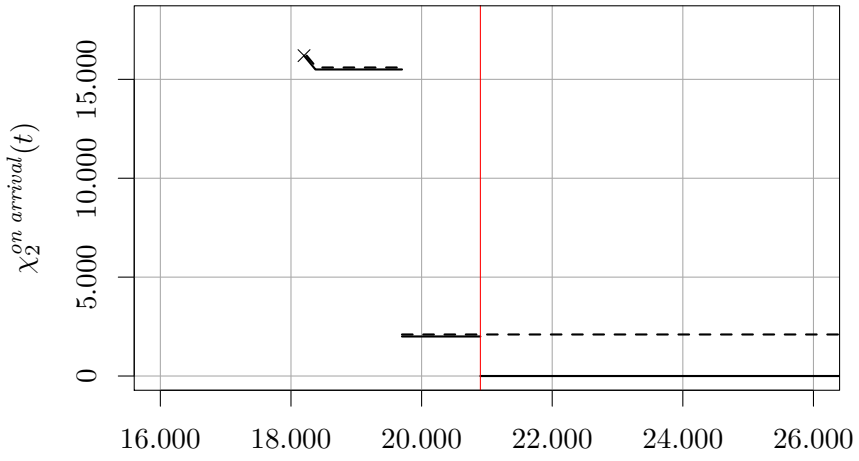

Abbildung 3.9: Schaubild der Funktion $\chi_2^{on\ arrival}$, der vorherigen Funktion $\chi_2^{on\ arrival}$ (gestrichelt) und des Endes der frühesten Pause an Kunde k_2 (rot).

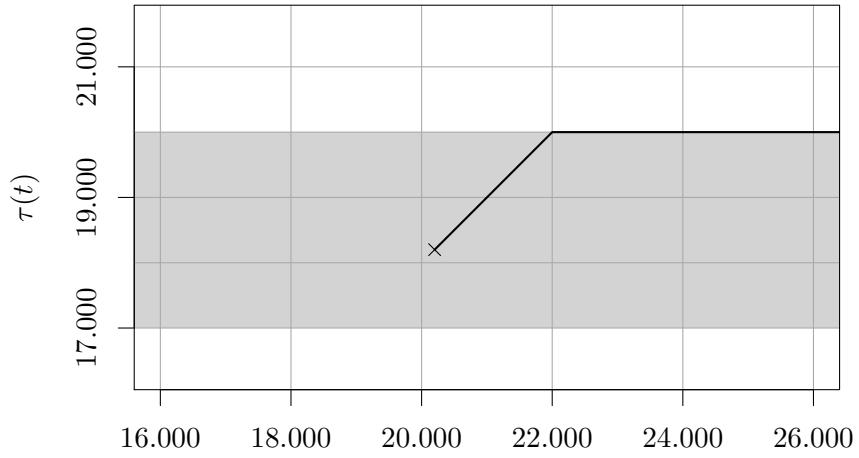

Abbildung 3.10: Schaubild der Funktion τ bei Kunde k_2. (Das Zeitfenster W_2 des Kunden k_2 ist grau markiert.)

Dann ergibt sich für $t \geq \alpha(\chi_2^{on\ arrival}) + R^{break}$ eine gefahrene Zeit seit der letzten Pause von $\chi_2^{on\ arrival}(t) = 0$, siehe Abbildung 3.9.

Anschließend muss nur noch der Service mit Dauer $S_2 = 2000$ innerhalb des Servicefensters $W_2 = [17000, 20000]$ bei Kunde k_2 durchgeführt werden. Dazu berechnen wir erneut die Funktion τ (siehe Abbildung 3.10) und daraus $\chi_2^{after\ service}(t) = \chi_2^{on\ arrival}(\tau(t))$ (siehe Abbildung 3.11). Das früheste Ende des Service $\alpha(\chi_2^{after\ service}) = 20200\,\text{s} \approx 5\,\text{h}\,37\,\text{min}$ ist somit auch das früheste Ende der Tour und damit der Rückgabewert des Algorithmus.

3.2.2 Reduzierung des Parkplatz-Suchaufwandes

Wie im Algorithmus beschrieben, werden immer alle Parkplätze des Straßengraphs betrachtet. Dies ist allerdings nicht nötig, da es Parkplätze gibt, die nichts zu einer besseren Lösung beitragen können. Zum einen sind dies Parkplätze $p \in P$, die zwischen den Kunden k_i und k_{i+1} nicht erreichbar sind. Unter Einhaltung der Lenk- und Ruhezeiten können diese somit ohnehin nicht angefahren werden. Im Algorithmus äußert sich dies durch eine Abfahrtszeit $dep_i = \perp$ von Kunde k_i zu Parkplatz p oder durch eine Abfahrtszeit $dep_p = \perp$ von Parkplatz p zu Kunde k_{i+1}. Wir können also, wie auch in Kapitel 3.1.1 beschrieben, die Suchweite auf R^{driven} beschränken.

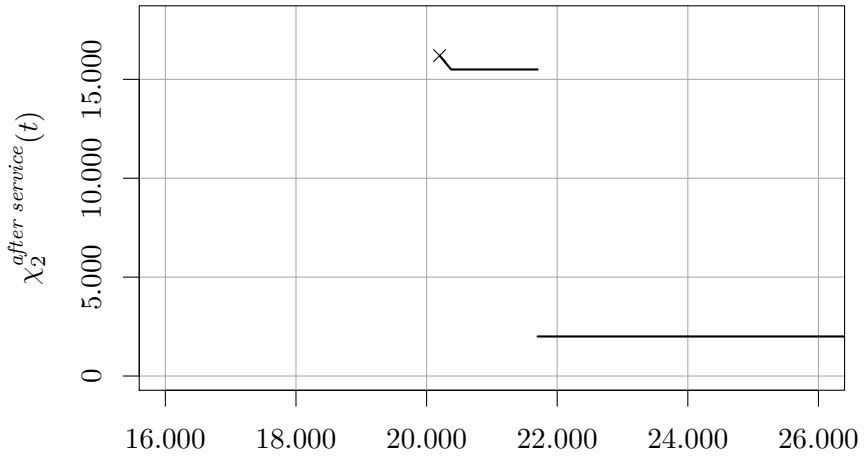

Abbildung 3.11: Schaubild der Funktion $\chi_2^{after\ service}$.

Außerdem gilt: Dauert die Fahrt von Kunde k_i zu Kunde k_{i+1} zu keinem Zeitpunkt länger als R^{driven}, und ist diese Fahrtdauer zu jedem Zeitpunkt kürzer als die Fahrtdauer zu einem Parkplatz p, so trägt auch dieser Parkplatz nichts zu einer optimalen Lösung bei. Eine Pause am Kunden k_{i+1} ist dann in jedem Fall vorzuziehen, denn der Fahrer kann bei einer direkten Fahrt am Kunden k_{i+1} ankommen, bevor er einen solchen Parkplatz p erreicht. Dadurch kann bei k_{i+1} eine Pause abgeschlossen werden, bevor der Fahrer die Pause auf Parkplatz p abschließen kann.

3.2.2.1 Sonderfall für zeitunabhängige Kantengewichte

Falls es sich im Straßengraph ausschließlich um zeitunabhängige, also konstante, Kantengewichte handelt, so kann zusätzlich zu den Überlegungen oben die Suche nach Parkplätzen unter gewissen Umständen vollständig unterlassen werden. Dies ist der Fall, wenn direkt nach dem Service zum nächsten Kunden gefahren werden kann, also falls $dep_i = \alpha(\chi_i^{after\ service})$. Dann gilt, dass nach keinen Parkplätzen gesucht werden muss, denn eine Pause am nächsten Kunden k_{i+1} ist immer vorzuziehen.

Die Fahrt von Kunde k_i zu einem Parkplatz p, und von dort zu Kunde k_{i+1} kann nicht kürzer sein, als die direkte Fahrt von Kunde k_i zu k_{i+1}. Damit kann eine Pause am Kunden k_{i+1} immer spätestens beendet werden, wenn der Fahrer von k_i nach p gefahren ist, dort eine Pause eingelegt hat, und dann von p nach k_{i+1} gefahren ist.

3.2.3 Finden eines optimalen Zeitplans

Ausgehend vom frühestmöglichen Ende einer Tour und den berechneten Funktionen χ_i^j kann rekonstruiert werden, zu welchen Zeitpunkten gefahren und Service geleistet werden muss, um das berechnete früheste Ende einer Tour zu erreichen. Tritt im rekonstruierten Zeitplan eine Wartezeit von R^{break} oder länger auf, so bedeutet dies, dass eine Pause eingelegt werden muss.

Um den Zeitplan berechnen zu können, muss zusätzlich protokolliert werden, wie der Wert $\chi_i^{on\ arrival}$ zustande gekommen ist. Die Notation $M(\chi_i^{on\ arrival}, t) = x$ gibt an, ob entweder direkt ($x = k_{i+1}$) oder über den Parkplatz p ($x = p$) gefahren werden muss, um den minimal möglichen Wert $\chi_i^{on\ arrival}(t)$ zu erreichen. Führen mehrere Werte von x zum minimalen Wert $\chi_i^{on\ arrival}(t)$, so kann ein beliebiger aus diesen ausgewählt werden.

Absteigend für alle $i \in \{n, \ldots, 2\}$ wird nacheinander zu Kunde k_i der Servicebeginn b_i und zu Kunde k_{i-1} der Abfahrtszeitpunkt t_{i-1} berechnet. Falls eine Pause auf einem Parkplatz

p gemacht wird, so wird ebenfalls der Abfahrtszeitpunkt t_p von Parkplatz p berechnet. Gegeben sei nun $\chi_i^{after\ service}$, $\chi_i^{on\ arrival}$ sowie der Abfahrtszeitpunkt t_i am Kunden k_i. Ist $i = n$ so wählen wir das früheste Ende der Tour als Abfahrtszeitpunkt t_n.

Wir wollen nun zunächst den Beginn des Service b_i bestimmen. Ist $t_i \geq \alpha(\chi_i^{after\ service}) + R^{break}$, dann kann zwischen dem Ende des Service und dem Abfahrtszeitpunkt t_i eine Pause eingelegt werden. In diesem Fall setzen wir daher $b_i = \alpha(\chi_i^{after\ service}) - S_i$. Ist $t_i < \alpha(\chi_i^{after\ service}) + R^{break}$, so setzen wir den Servicebeginn auf den spätestmöglichen Zeitpunkt, zu dem der Service noch bis zur Abfahrt um t_i beendet werden kann, also $b_i = \max\{t' \in W_i \mid \alpha(\chi_i^{on\ arrival}) \leq t' \leq t_i - S_i\}$.

Wir wollen nun die Fahrt von Kunde k_{i-1} zu Kunde k_i betrachten. Ist $M(\chi_i^{on\ arrival}, b_i) = k_{i-1}$, so führt die Fahrt direkt von Kunde k_{i-1} zu Kunde k_i. Ist $b_i \geq \alpha(\chi_i^{on\ arrival}) + R^{break}$, dann kann vor dem Servicebeginn b_i eine Pause eingelegt werden. Wir setzen daher $t_{i-1} = \alpha(\chi_i^{on\ arrival}) - \overline{\Psi}_{i-1,i}(\alpha(\chi_i^{on\ arrival}))$. Andernfalls muss der Abfahrtszeitpunkt t_{i-1} am Kunden k_{i-1} zeitlich im Intervall $I_{i-1} = [\alpha(\chi_{i-1}^{after\ service}), b_i - \overline{\Psi}_{i-1,i}(b_i)]$ liegen. Früher ist nicht möglich, da der Fahrer sonst den Service am Kunden k_{i-1} nicht vollenden könnte. Später ist ebenfalls nicht möglich, da der Fahrer sonst nicht rechtzeitig zum Zeitpunkt b_i zum Service am Kunden k_i ankommen würde. Wir wählen ein $t_{i-1} \in I_{i-1}$ so, dass $\chi_{i-1}^{after\ service}(t_{i-1}) + \Psi_{i-1,i}(t_{i-1})$ minimal ist.

Ist $M(\chi_i^{on\ arrival}, b_i) = p$ für ein $p \in P$, so führt die Fahrt von Kunde k_{i-1} über Parkplatz p zu Kunde k_i. Wir berechnen zuerst die Abfahrt bei Parkplatz p, dann bei Kunde k_{i-1}. Ist $b_i \geq \alpha(\chi_i^{on\ arrival}) + R^{break}$, dann wird analog zur direkten Fahrt von k_{i-1} zu k_i die Abfahrt bei Parkplatz p so früh wie möglich gesetzt. Wir erhalten $t_p = \alpha(\chi_i^{on\ arrival}) - \overline{\Psi}_{p,i}(\alpha(\chi_i^{on\ arrival}))$. Damit kann an Kunde k_i vor dem Beginn des Service eine Pause gemacht werden. Ansonsten kann die Abfahrt bei Parkplatz p nur für Zeitpunkte im Intervall $I_p = [arr_p, b_i - \overline{\Psi}_{p,i-1}(b_i)]$ erfolgen. Wir wählen ein $t_p \in I_p$ so, dass $\Psi_{p,i}(t_i)$ minimal ist. Zur Berechnung der Abfahrtszeit an Kunde k_{i-1} gehen wir analog vor. Setze das Intervall I_{i-1} der möglichen Abfahrtszeitpunkte von Kunde k_{i-1} zu Parkplatz p wie folgt: $I_{i-1} = [\alpha(\chi_{i-1}^{after\ service}), t_p - R^{break} - \overline{\Psi}_{i-1,p}(t_p - R^{break})]$. Da nach der Pause auf dem Parkplatz p die gefahrene Zeit seit der letzten Pause bei Null liegt, kann der Abfahrtszeitpunkt t_{i-1} beliebig aus $\{t \in I_{i-1} \mid \chi_{i-1}^{after\ service}(t) + \Psi_{i-1,p}(t) \leq R^{driven}\}$ gewählt werden. Dadurch erhalten wir einen Abfahrtszeitpunkt am Kunden k_{i-1}.

Wir führen das Verfahren für die übrigen Kunden durch, bis der Abfahrtszeitpunkt für den ersten Kunden k_1 berechnet wurde. Der Beginn des Service b_1 für den ersten Kunden k_1 kann dann wie oben berechnet werden. Dadurch erhalten wir alle Abfahrts- und Servicezeitpunkte für einen Zeitplan, der zum frühestmöglichen Zeitpunkt endet.

3.2.4 Vorgaben der USA

Im Unterschied zu den Vorgaben der EU beziehen sich die der USA auf die *vergangene*, und nicht die *gefahrene* Zeit seit der letzten Pause. Zu beachten ist, dass der Service von der Pausenregel nicht betroffen ist. Konkret bedeutet das, dass der Service zu jeder Zeit, auch bei einer letzten Pause vor mehr als $R^{elapsed} = 28800\,\mathrm{s} = 8\,\mathrm{h}$, begonnen und weitergeführt werden darf. Der entsprechende Schedulingalgorithmus für die USA ist in Algorithmus 3.3 dargestellt. Im Folgenden sollen kurz die Unterschiede zwischen dem Algorithmus für die Regelungen der USA und dem Algorithmus für die Regelungen der EU herausgestellt werden.

Die Funktion χ_i^j gibt die vergangene Zeit seit der letzten Pause an. Für die Berechnung von $\chi_i^{after\ service}(t)$ aus $\chi_i^{on\ arrival}$ ist zusätzlich die Zeit $t - \tau(t)$ vom Servicebeginn bis zum aktuellen Zeitpunkt t relevant, da diese die vergangene Zeit seit der letzten Pause ebenfalls erhöht. Ist eine Fahrt zum nächsten Kunden möglich, so wird nicht die bisher gefundene

Algorithmus 3.3: US-SCHEDULING für einen Planungshorizont von einem Tag mit maximal einem Parkplatz zwischen zwei Kunden

Daten: $\Psi_{x,y}(t)$ Fahrzeit von x nach y bei einer Abfahrt zum Zeitpunkt t
$\overline{\Psi}_{x,y}(t)$ Fahrzeit von x nach y bei einer Ankunft zum Zeitpunkt t
n Anzahl Kunden
t_0 Ende der letzten Tätigkeit (Service leisten oder fahren) der vorherigen Tour des Fahrers
c vergangene Zeit zum Zeitpunkt t_0 seit dem Ende der letzten Pause
W_i Zeitfenster für den Beginn des Service bei Kunde k_i
S_i Servicedauer bei Kunde k_i
P Menge aller Parkplätze (ohne Kunden)
R^{driven} erlaubte Maximalzeit zwischen zwei Pausen
R^{break} minimale Pausenzeit

Ausgabe: frühestes Ende der Tour oder $\perp$, falls die Tour nicht erfüllbar ist

1 **begin**

2 $\quad \chi_1^{on\ arrival}(t) \leftarrow \begin{cases} \perp, & t < t_0 \\ c + (t - t_0), & t \geq t_0 \wedge c \neq 0 \\ 0, & \text{otherwise} \end{cases}$

3 $\quad$ **for** $i \leftarrow 1$ **to** n **do**

4 $\quad\quad \chi_i^{on\ arrival}(t) \leftarrow \begin{cases} \chi_i^{on\ arrival}(t), & t < \alpha(\chi_i^{on\ arrival}) + R^{break} \\ 0, & t \geq \alpha(\chi_i^{on\ arrival}) + R^{break} \end{cases}$

`// Wait and perform service`

5 $\quad\quad t_{begin} \leftarrow \min([\alpha(\chi_i^{on\ arrival}), \infty) \cap W_i)$

6 $\quad\quad \tau(t) \leftarrow \begin{cases} \perp, & t < t_{begin} + S_i \\ \max((t_{begin}, t - S_i] \cap W_i), & \text{otherwise} \end{cases}$

7 $\quad\quad \chi_i^{after\ service}(t) \leftarrow \chi_i^{on\ arrival}(\tau(t)) + t - \tau(t)$

8 $\quad\quad$ **if** $i = n$ **then**

9 $\quad\quad\quad$ **return** $\alpha(\chi_n^{after\ service})$

10 $\quad\quad \chi_i^{after\ service}(t) \leftarrow \begin{cases} \chi_i^{after\ service}(t), & t < \alpha(\chi_i^{after\ service}) + R^{break} \\ 0, & t \geq \alpha(\chi_i^{after\ service}) + R^{break} \end{cases}$

`// Drive and take break`

11 $\quad\quad dep_i \leftarrow \min\{t \geq \alpha(\chi_i^{after\ service}) \mid \chi_i^{after\ service}(t) + \Psi_{i,i+1}(t) \leq R^{elapsed}\}$

12 $\quad\quad arr_{i+1} \leftarrow dep_i + \Psi_{i,i+1}(dep_i)$

13 $\quad\quad \chi_{i+1}^{on\ arrival}(t) \leftarrow \begin{cases} \perp, & t < arr_{i+1} \\ \chi_i^{after\ service}(t - \overline{\Psi}(t)) + \overline{\Psi}(t), & \text{otherwise} \end{cases}$

14 $\quad\quad$ **for** $p \in P$ **do**

15 $\quad\quad\quad dep_i \leftarrow \min\{t \geq \alpha(\chi_i^{after\ service}) \mid \chi_i^{after\ service}(t) + \Psi_{i,p}(t) \leq R^{elapsed}\}$

16 $\quad\quad\quad arr_p \leftarrow dep_i + \Psi_{i,p}(dep_i)$

17 $\quad\quad\quad dep_p \leftarrow \min\{t \geq arr_p + R^{break} \mid \Psi_{p,i+1}(t) \leq R^{elapsed}\}$

18 $\quad\quad\quad arr_{i+1} \leftarrow dep_p + \Psi_{p,i+1}(dep_p)$

19 $\quad\quad\quad \chi_{i+1}^{on\ arrival}(t) \leftarrow \min\{\chi_{i+1}^{on\ arrival}(t), \begin{cases} \perp, & t < arr_{i+1} \\ \overline{\Psi}_{p,i+1}(t), & \text{otherwise} \end{cases}\}$

20 $\quad\quad \tau(t) \leftarrow \max\{t' \leq t \mid \chi_{i+1}^{on\ arrival}(t') \leq R^{elapsed}\}$

21 $\quad\quad \chi_{i+1}^{on\ arrival}(t) \leftarrow \chi_{i+1}^{on\ arrival}(\tau(t)) + t - \tau(t)$

minimale Fahrzeit gewählt, sondern für jeden Zeitpunkt so spät wie möglich gefahren. Früheres Fahren würde nur dazu führen, dass eine einmal möglicherweise eingelegte Pause früher als nötig abgebrochen wird, obwohl diese auch länger als die Mindestzeit R^{break} andauern könnte. Durch den früheren Abbruch der letzten Pause würde dadurch die zum aktuellen Zeitpunkt t vergangene Zeit seit der letzten Pause steigen.

Ist eine Fahrt zum nächsten Kunden zu bestimmten Fahrzeiten nicht möglich (aufgrund einer Fahrzeit, die über der noch verbleibenden erlaubten Fahrzeit des Fahrers liegt), so werden ungültige Fahrzeiten durch eine frühere Fahrt (mit einer noch erlaubten Fahrzeit) und anschließendes Warten ersetzt. Dazu bildet τ in Zeile 20 den Zeitpunkt t auf den spätestmöglichen Zeitpunkt ab, an dem eine Ankunft noch ohne Verstoß gegen die Lenk- und Ruhezeiten möglich ist.

4. Experimentelle Evaluierung

In diesem Kapitel werden die erarbeiteten Routingalgorithmen und der EU-Schedulingalgorithmus auf ihre Laufzeit untersucht. Außerdem wird der Einfluss von zeitabhängigen Fahrzeiten und die Anzahl der zur Verfügung stehenden Parkplätzen auf die Erfüllbarkeit von Touren analysiert. Wir stellen zuerst das Testsystem und dann die Testfälle und Ergebnisse für die Algorithmen zur Routenplanung vor. Anschließend betrachten wir die Testfälle für den EU-Schedulingalgorithmus und dessen Ergebnisse.

4.1 Testumgebung und verwendete Straßengraphen

Alle Tests wurden auf einem VMware ESX-Cluster mit Ubuntu 16.04 LTS durchgeführt. Zur Verfügung standen vier Kerne eines Intel Xeon E5-2698 v4 mit 2,20 GHz und 64 GB RAM. Da alle getesteten Algorithmen sequentiell sind, wurden diese nur auf einem Kern ausgeführt. Als Compiler wurde GCC 5.3 verwendet.

Die vorgestellten Routenplanungsalgorithmen bauen auf der zeitabhängigen Contraction Hierarchy-Implementierung von Batz [Bat15] auf, die um die Berechnung von Profilen und um die Parkplatzsuche (siehe Kapitel 3.1.1) erweitert wurden. Die Implementierung enthält außerdem eine Sonderfallbehandlung für konstante Kantengewichte. Für den Algorithmus von Imai und Iri verwenden wir die Implementierung von Neubauer [Neu09].

Die Tests wurden auf einem zeitabhängigen Graphen von Deutschland aus dem Jahr 2016 durchgeführt, und stellen die LKW-Fahrzeiten an einem Freitag dar („Deu 2016“). Weiterhin enthält dieser Graph nur die von LKW befahrbaren Straßensegmente. Einige Tests wurden mit einem zeitabhängigen Graphen von 2006 mit PKW-Fahrzeiten eines Dienstags bzw. Donnerstags durchgeführt („Deu 2006“). Dieser Graph wurde auch in einigen anderen Publikationen zum Thema Routenplanung zur Laufzeitmessung verwendet [BGNS10, BGSV13, Str16, Neu09]. Beide Graphen wurden durch die PTV Group bereitgestellt.

Für die Parkmöglichkeiten für LKW wurden die Daten der App „Truck Parking Europe“ der PTV Group genutzt. In dieser App kann nach einer Registrierung jeder Nutzer die Orte LKW-gerechter Parkplätze angeben. Da dort viele Parkplätze lediglich Stellplätze ohne beispielsweise sanitäre Anlagen darstellen, wurden zusätzlich aus allen Parkplätzen in Deutschland 8 % zufällig gleichverteilt ausgewählt, die für weitere Tests genutzt werden („Deu 2016 red“). Dadurch können Laufzeitveränderungen aufgezeigt und Änderungen bezüglich der Erfüllbarkeit von Touren bewertet werden, falls nur eine geringere Parkplatzanzahl mit besserer Ausstattung betrachtet werden soll.

Tabelle 4.1: Eigenschaften der Straßengraphen.

	# Knoten	# Kanten	td-Kanten [%]	# Parkplätze	Kerngröße [%]
Deu 2016	7.233.592	24.405.343	28,5	5449	0,20
Deu 2016 red	7.233.592	24.093.648	28,5	484	0,05
Deu 2006	5.137.910	12.471.300	3,7	5327	0,20

Tabelle 4.1 zeigt für Deu 2016, Deu 2016 red und Deu 2006 die Anzahl der Knoten, die Anzahl der Kanten, die relative Anzahl der Kanten mit nicht-konstanten Fahrzeiten („td-Kanten“) und die Anzahl der Parkplätze. Zusätzlich wird die Anzahl der Knoten im Kern relativ zur Gesamtanzahl der Knoten dargestellt („Kerngröße“). Da der Schedulingalgorithmus jeweils eine *st*- und eine Parkplatzanfrage pro Fahrt zwischen zwei Kunden durchführt, wurde die Kerngröße experimentell bestimmt, um eine minimale durchschnittliche Laufzeit für beide Anfragetypen zusammen zu erreichen. Durch die unterschiedliche Wahl des Kerns für Deu 2016 und Deu 2016 red ergibt sich die abweichende Anzahl Kanten im Graph.

Die unterschiedliche Parkplatzanzahl für Deu 2016 und Deu 2006 resultiert aus der Zuordnung der Parkplätze zu Knoten. Ein Knoten wird als Parkplatz markiert, wenn der euklidische Abstand zwischen Knoten und Parkplatz unter allen Knoten am kürzesten ist. Ist zu einem Parkplatz kein Knoten innerhalb von 200 Metern gelegen, so wird der Parkplatz verworfen. Für Deu 2006 wurde für 122 Parkplätze kein Knoten gefunden, der innerhalb des 200-Meter-Umkreises lag.

Zusätzlich wurden für jeden Graphen die entstandenen Fahrzeitfunktionen mit dem Algorithmus von Imai und Iri mit maximalem relativen Approximationsfehler ε approximiert, siehe Tabelle 4.2. Ein Approximationsfehler $\varepsilon = 0\,\%$ gibt an, dass die Funktionen nicht approximiert wurden. Konstante Fahrzeiten ($\varepsilon = \mathrm{C}$) ergeben sich aus dem Minimum der ursprünglichen Kantengewichten. Für konstante Fahrzeiten werden also keine Staus berücksichtigt (Freeflow). Aufgrund der unterschiedlichen Anzahl Kanten stimmt auch die Anzahl der Stützstellen von Deu 2016 und Deu 2016 red nicht überein.

Wie zu erwarten war, sinkt die Anzahl der Stützstellen bei steigendem maximalen Approximationsfehler ε. Wir sehen allerdings, dass eine Erhöhung des Approximationsfehlers für Deu 2016 und Deu 2016 red von $\varepsilon = 2\,\%$ auf $\varepsilon = 5\,\%$ die Anzahl der Stützstellen ungefähr halbiert, während für Deu 2006 die Anzahl der Stützstellen nur auf ungefähr 70 % zurückgeht. Im Vergleich zur exakten Variante mit $\varepsilon = 0\,\%$ sinkt die Anzahl der Stützpunkte für Deu 2016 und Deu 2016 red mit $\varepsilon = 5\,\%$ auf unter 3,5 %, während für Deu 2006 die Anzahl der Stützpunkte nur auf ungefähr 17,1 % sinkt. Das Verhältnis der Anzahl Stützstellen bei einem Approximationsfehler ε zur Gesamtanzahl Stützstellen bei exakten Fahrzeiten ist in Abbildung 4.1 dargestellt.

4.2 Routing

In diesem Abschnitt werden die Laufzeiten der Routingalgorithmen evaluiert. Im Folgenden wird die Laufzeit und die Genauigkeit von *st*-Anfragen für verschiedene Approximationsfehler ε angegeben. Für Parkplatzanfragen wird die Laufzeit gemessen. Ausgeführt werden die Tests für *st*- und Parkplatzanfragen auf einem Graphen wie in Kapitel 3.1.2 angegeben.

4.2.1 *st*-Anfrage

Eine *st*-Anfrage berechnet zu zwei gegebenen Knoten s und t das Profil $\Psi_{s,t}$. Es wurden jeweils 100 *st*-Anfragen mit zufällig gleichverteilten Start- und Endknoten durchgeführt.

Tabelle 4.2: Anzahl Stützstellen in Abhängigkeit vom Approximationsfehler.

	ε [%]	# Stützstellen [10^3]
Deu 2016	0,00	2.865.247
	0,05	818.735
	0,10	639.188
	0,50	359.817
	1,00	268.398
	2,00	160.971
	5,00	72.969
	C	24.405
Deu 2016 red	0,00	2.298.066
	0,05	762.388
	0,10	609.069
	0,50	353.689
	1,00	265.291
	2,00	159.127
	5,00	71.820
	C	24.094
Deu 2006	0,00	134.132
	0,05	83.203
	0,10	70.703
	0,50	45.752
	1,00	38.417
	2,00	32.534
	5,00	23.081

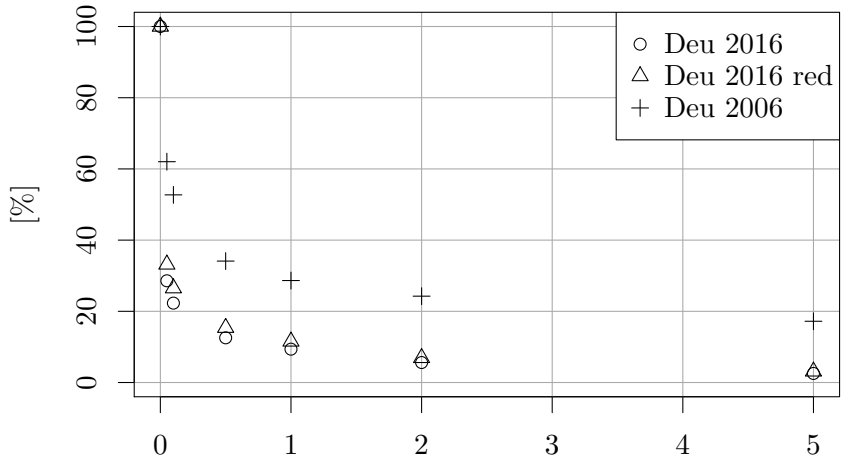

Abbildung 4.1: Anzahl Stützstellen für einen Approximationsfehler ε (in Prozent) prozentual zur Gesamtanzahl Stützstellen für exakte Fahrzeitfunktionen.

Tabelle 4.3: Laufzeiten der *st*-Anfragen.

	ε [%]	Laufzeit [ms]	# Knoten	# Kanten	# Punkte [10^3]
Deu 2016	0,00	14.083	1978	14.159	193.558
	0,05	2461	1979	14.175	33.959
	0,10	1605	1985	14.213	21.791
	0,50	545	2008	14.387	7393
	1,00	360	2137	15.412	4969
	2,00	194	1953	13.918	2361
	5,00	252	2385	18.406	3011
	C	1,71	491	2239	2,5
Deu 2016 red	0,00	16.346	2047	13.782	228.465
	0,05	2856	2058	13.863	40.047
	0,10	1830	2062	13.892	25.712
	0,50	666	2237	15.100	9050
	1,00	495	2803	18.788	6616
	2,00	288	2716	18.435	3666
	5,00	179	2076	14.286	2166
	C	1,16	461	1958	2,2
Deu 2006	0,00	81,80	526	2704	1178
	0,05	41,49	527	2711	469
	0,10	35,54	527	2715	351
	0,50	29,60	529	2729	150
	1,00	27,18	531	2743	99
	2,00	23,81	538	2799	67
	5,00	23,80	578	3085	46

Zuerst wird die Laufzeit auf Deu 2016, Deu 2016 red und Deu 2006 mit verschiedenen Approximationsfehlern miteinander verglichen. Danach wird für Deu 2016 die Abweichung der berechneten approximierten Profile vom exakten Profil mit $\varepsilon = 0$ betracht.

4.2.1.1 Laufzeit

In Tabelle 4.3 sind für Deu 2016, Deu 2016 red und Deu 2006 sowie unterschiedlichen Approximationsfehlern ε folgende Werte angegeben:

maximaler Approximationsfehler ε: Maximaler relativer Approximationsfehler der Kantengewichte in Prozent, $\varepsilon = 0\,\%$ bedeutet, dass exakte Kantengewichte verwendet wurden, $\varepsilon = \mathrm{C}$ bedeutet, dass konstante Kantengewichte ohne Berücksichtigung von Staus und Verzögerungen betrachtet wurden.

Laufzeit: Durchschnittliche Dauer pro Anfrage in Millisekunden.

 # Knoten: Durchschnittliche Anzahl Knoten pro Anfrage, die insgesamt aus der Queue `Q` (siehe Dijkstras Algorithmus 2.1) entnommen werden.

 # Kanten: Durchschnittliche Anzahl Kanten pro Anfrage, die durch den Algorithmus berücksichtigt werden.

 # Punkte: Durchschnittliche Gesamtanzahl Stützpunkte pro Anfrage in 10^3, die insgesamt durch die Link- und Mergeoperationen verarbeitet wurden.

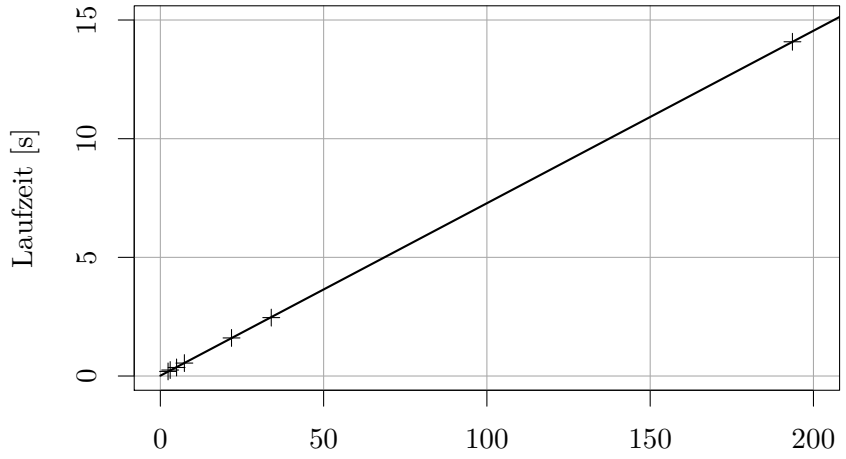

Abbildung 4.2: Laufzeit in Abhängigkeit der Anzahl der verarbeiteten Stützpunkte (in 10^6) für Deu 2016.

Betrachten wir zuerst die Laufzeit von *st*-Anfragen für Deu 2016. Wir sehen, dass die Laufzeit mit steigendem Approximationsfehler ε sinkt. In Abbildung 4.2 ist die Laufzeit in Abhängigkeit von der Anzahl der verarbeiteten Stützpunkte (in 10^6) für die Approximationsfehler in der Tabelle (außer für $\varepsilon = \mathrm{C}$) aufgetragen, sowie die daraus resultierende lineare Regressionsgerade. Man erkennt, dass die Laufzeit ungefähr proportional zu der Anzahl der verarbeiteten Stützpunkte ist. Lediglich für die Laufzeit bei konstanten Fahrzeitfunktionen trifft dieser Zusammenhang nicht zu. Ausgelöst wird das vermutlich dadurch, dass die Verarbeitung der Stützpunkte (von denen es bei konstanten Fahrzeitfunktionen nur einen pro Kante gibt), nicht mehr die Laufzeit dominiert. Außerdem sieht man, dass die Anzahl der betrachteten Knoten und Kanten über unterschiedliche Approximationsfehler $\varepsilon \neq \mathrm{C}$ wenig schwankt, während diese für konstante Fahrzeiten sinkt. Dieser Effekt kann dadurch erklärt werden, dass bei zeitabhängigen Fahrzeiten mehrere Pfade zu unterschiedlichen Zeiten zum Profil $\Psi_{s,t}$ beitragen, während bei konstanten Fahrzeiten nur ein Pfad entscheidend ist. Dass die Anzahl der verarbeiteten Stützpunkte für $\varepsilon = 2\,\%$ niedriger ist als für $\varepsilon = 5\,\%$, scheint ein Ausreißer zu sein.

Für die Werte von Deu 2016 red ergibt sich ein ähnliches Bild, lediglich die Laufzeiten liegen etwas höher als bei Deu 2016. Dies lässt sich damit erklären, dass eine Knotenreihenfolge des Kerns entsprechend gewählt wurde, die auch für Parkplatzanfragen genutzt wird. So sind die *st*-Anfragen von Deu 2016 red zwar etwas langsamer als die von Deu 2016, die Parkplatzanfragen sind allerdings deutlich schneller, wie wir später sehen werden. Da Deu 2006 auch für den gleichen Approximationsfehler eine niedrigere Anzahl Stützpunkte als Deu 2016 hat, ist die Laufzeit der *st*-Anfragen für $\varepsilon = 0\,\%$ zirka 170-mal schneller als die von Deu 2016. Ebenso ist die Anzahl der verarbeiteten Stützstellen für Deu 2016 ungefähr 165-mal so hoch, wie die von Deu 2006. Wir sehen allerdings, dass die Laufzeit für Deu 2006 bei einer Erhöhung des maximalen Approximationsfehlers von $\varepsilon = 2\,\%$ auf $\varepsilon = 5\,\%$ um weniger als $1\,\%$ sinkt. Dafür verantwortlich ist die Gesamtanzahl der Stützstellen für $\varepsilon = 2\,\%$ und $\varepsilon = 5\,\%$., siehe Tabelle 4.2, Abbildung 4.1 und die dortigen Hinweise. Zuletzt bemerken wir noch, dass für Deu 2006 kein linearer Zusammenhang zwischen der Laufzeit und der Anzahl der verarbeiteten Stützpunkte besteht, weil hier die Laufzeit wie auch bei konstanten Kantengewichten nicht durch die Link- und Mergeoperationen dominiert wird.

4.2.1.2 Genauigkeit

In Tabelle 4.4 sind zur Betrachtung der Genauigkeit der *st*-Anfragen bei unterschiedlichen ε-Werten für Deu 2016 folgende Werte angegeben:

Tabelle 4.4: Genauigkeit der *st*-Anfragen.

	ε [%]	∅-Fehler [10^{-6}]	max-Fehler [10^{-6}]
Deu 2016	0,00	0	0
	0,05	185	1481
	0,10	378	3182
	0,50	2409	48.970
	1,00	4341	38.847
	2,00	8596	40.768
	5,00	21.189	66.638
	C	19.281	148.859

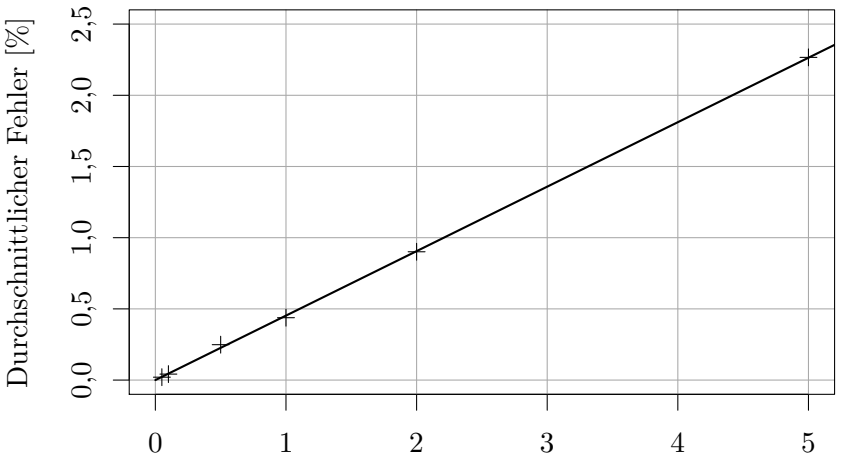

Abbildung 4.3: Durchschnittlicher Fehler in Abhängigkeit vom Approximationsfehler in Prozent für Deu 2016.

∅-Fehler: Durchschnitt über die durchschnittliche relative Abweichung des berechneten Profils in 10^{-6} zum Referenzergebnis. Das Referenzergebnis ist das Profil der entsprechenden Anfrage mit $\varepsilon = 0$.

max-Fehler: Maximale relative Abweichung in 10^{-6} zum Referenzergebnis über alle Anfragen und jeden Zeitpunkt. Das Referenzergebnis ist das Profil der entsprechenden Anfrage mit $\varepsilon = 0$.

Wie zu erwarten war, sinkt die Genauigkeit der Profile bei steigendem Approximationsfehler, sowohl für den durchschnittlichen, als auch für den maximalen Fehler. Ein annähernd linearer Verlauf des durchschnittlichen Fehlers in Abhängigkeit des Approximationsfehlers ε kann erkannt werden. In Abbildung 4.3 ist dazu der durchschnittliche Fehler in Abhängigkeit vom Approximationsfehler ε in Prozent dargestellt, sowie die daraus resultierende lineare Regressionsgerade.

Der maximale Fehler fällt mit 15 % für konstante Fahrzeiten höher aus, als bei allen anderen Approximationsfehlern ε. Der durchschnittliche Fehler für konstante Fahrzeiten ist hingegen mit ungefähr 1,9 % etwas niedriger als der durchschnittliche Fehler von 2,1 % für einen Approximationsfehler von $\varepsilon = 5\,\%$. Zur Erklärung betrachten wir für zwei beispielhaft gewählte Knoten s und t das Profil $\Psi_{s,t}$, siehe Abbildung 4.4. Wie zu sehen ist, liegt die Fahrzeit ohne Approximation bei über sechs Stunden (über 22500 s). Während das konstante Profil für Abfahrtszeiten nachts sehr genau mit dem exakten Profil ($\varepsilon = 0$) übereinstimmt, liegt das Profil für $\varepsilon = 5$ durchgängig unter dem exakten

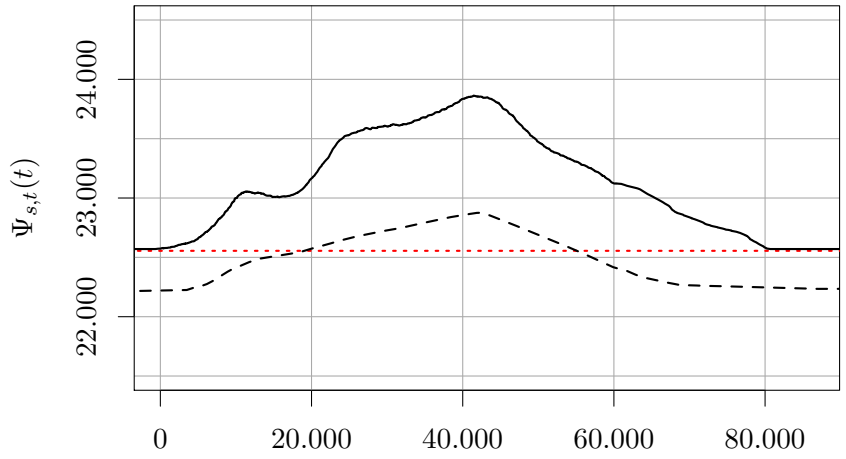

Abbildung 4.4: Beispielschaubild eines Profils für $\varepsilon = 0$, $\varepsilon = 5$ (gestrichelt) und für konstante Fahrzeiten (gepunktet, rot).

Profil und zum Teil sogar auch unter dem Profil, das mit konstanten Fahrzeiten berechnet wurde. Möglicherweise bessere Ergebnisse würde eine Approximation mit anderen relativen Fehlergrenzen liefern, bei der der Approximationsfehler nach unten stärker begrenzt wird, als nach oben. Gesucht ist dann zum oberen relativen Approximationsfehler $\varepsilon_\uparrow$ und dem unteren relativen Approximationsfehler $\varepsilon_\downarrow < \varepsilon_\uparrow$ und einer Fahrzeitfunktion φ eine Funktion ψ mit $\varphi(x)/(1+\varepsilon_\downarrow) \leq \psi(x) \leq (1+\varepsilon_\uparrow) \cdot \varphi(x)$. Tests mit dieser Art der Approximation wurden allerdings im Rahmen dieser Arbeit nicht durchgeführt.

Ein anderer Ansatz, der gute Ergebnisse verspricht, wurde von Strasser für Earliest-Arrival-Anfragen beschrieben [Str16], die zu einem Abfahrtszeitpunkt t_0 und Start- und Zielknoten s und t die Dauer der Fahrt $\Psi_{s,t}(t)$ berechnen. Strassers Verfahren funktioniert grob umrissen wie folgt:

Zu einem gegebenen zeitabhängigen Graphen werden innerhalb gewählter Zeitfenster die Fahrzeiten gemittelt und daraus mehrere Graphen mit konstanten Fahrzeiten erzeugt. Für eine Earliest-Arrival-Anfrage wird dann auf den zeitunabhängigen Graphen je ein oder mehrere kürzeste Pfade gesucht. Diese Pfade werden zu einem weiteren Graphen mit den ursprünglichen zeitabhängigen Kantengewichten zusammengesetzt. Auf dem entstandenen Graphen wird eine zeitabhängige Earliest-Arrival-Anfrage gestartet. Dieser Ansatz bietet eine bessere Beschleunigung als die Approximation der Fahrzeitfunktionen durch den Algorithmus von Imai und Iri. Es soll außerdem angemerkt werden, dass durch diesen Algorithmus die Fahrzeiten keinesfalls unter-, sondern höchstens überschätzt werden. Die Tauglichkeit des Verfahrens für Profilanfragen muss jedoch noch geprüft werden.

4.2.2 Parkplatzanfrage

Eine Parkplatzanfrage berechnet für zwei gegebene Knoten s und t die Profile $\Psi_{s,p}$ und $\Psi_{p,t}$ für alle Parkplätze p , für die folgende Bedingungen erfüllt sind: Die Fahrt von s nach p dauert zu einem Zeitpunkt weniger als 4,5 Stunden und die Fahrt von p nach t dauert zu einem Zeitpunkt weniger als 4,5 Stunden. Für Deu 2016, Deu 2016 red und Deu 2006 und jeden getesteten Approximationsfehler ε wurden 100 Parkplatzanfragen mit zufällig gleichverteilten Start- und Endknoten durchgeführt. Die Start- und Zielknoten der Anfragen sind identisch mit denen aus Kapitel 4.2.1 für die *st*-Anfragen. Im Folgenden werden die durchschnittlichen Laufzeiten von Deu 2016, Deu 2016 red und Deu 2006 miteinander verglichen.

Tabelle 4.5: Laufzeiten der Parkplatzanfragen.

	ε [%]	Laufzeit [ms]	# Knoten	# Kanten	# Punkte [10^3]
Deu 2016	0,00	350.489	13.932	537.645	4.883.960
	0,05	83.422	13.941	538.033	1.154.110
	0,10	49.648	13.949	538.360	687.609
	0,50	14.472	14.016	540.807	196.849
	1,00	8842	14.096	544.029	116.461
	2,00	5763	14.229	550.408	73.398
	5,00	4114	14.508	566.281	49.680
	C	106	13.291	303.254	327
Deu 2016 red	0,00	97.173	3754	152.266	1.342.970
	0,05	13.489	3756	152.365	187.620
	0,10	8115	3758	152.446	112.637
	0,50	2530	3774	153.121	33.892
	1,00	1588	3795	154.072	20.669
	2,00	1030	3827	155.936	12.861
	5,00	733	3868	160.267	8558
	C	31	3641	97.763	107
Deu 2006	0,00	31.888	15.187	647.922	472.798
	0,05	21.910	15.213	649.159	318.669
	0,10	16.104	15.199	648.628	232.014
	0,50	6953	15.282	654.353	91.271
	1,00	4297	15.301	655.176	56.869
	2,00	2936	15.421	662.671	36.493
	5,00	2523	16.010	690.928	29.178

4.2.2.1 Laufzeit

Die in Tabelle 4.5 angegeben Werte haben dieselbe Bedeutung wie die von Tabelle 4.3 in Kapitel 4.2.1.1. Wir werden zunächst für Deu 2016 die Laufzeit analysieren und daraufhin Deu 2016 red und Deu 2006 betrachten.

Wir sehen, dass für Deu 2016 die Laufzeit für eine Parkplatzanfrage im Schnitt 25- bis 30-mal länger ist als für eine *st*-Anfrage (je nach Approximationsfehler). Der Grund liegt in der Größe des Kerns. Für eine *st*-Anfrage wird lediglich entlang von Kanten gesucht, die zu einem bezüglich der Knotenreihenfolge $<$ weiter oben eingeordneten Knoten führen. Für eine Parkplatzanfrage wird hingegen innerhalb des Kern von einem Knoten aus in beide Richtung (aufwärts als auch abwärts) gesucht, was die Anzahl der verarbeiteten Knoten, Kanten und Stützpunkte erhöht.

Der lineare Zusammenhang zwischen der Anzahl der verarbeiteten Stützpunkte und der Laufzeit kann ebenfalls wieder beobachtet werden. Dazu sehen wir in Abbildung 4.5 die lineare Regressionsgerade zu den entsprechenden Datenpunkten für Deu 2016(außer für $\varepsilon = \mathrm{C}$).

Die Laufzeit für eine Parkplatzanfrage für Deu 2016 red ist ungefähr 3,5 bis 6,5-mal so schnell wie eine Parkplatzanfrage für Deu 2016. Das ist bedingt durch eine kleinere Kerngröße von nur 0,05 % für Deu 2016 red im Gegensatz zu einer Kerngröße von 0,2 % für Deu 2016. Durch die geringere Kerngröße findet ein kleinerer Teil der Suche innerhalb des Kerns statt. Da innerhalb des Kerns bezüglich der Knotenreihenfolge $<$ sowohl aufwärts,

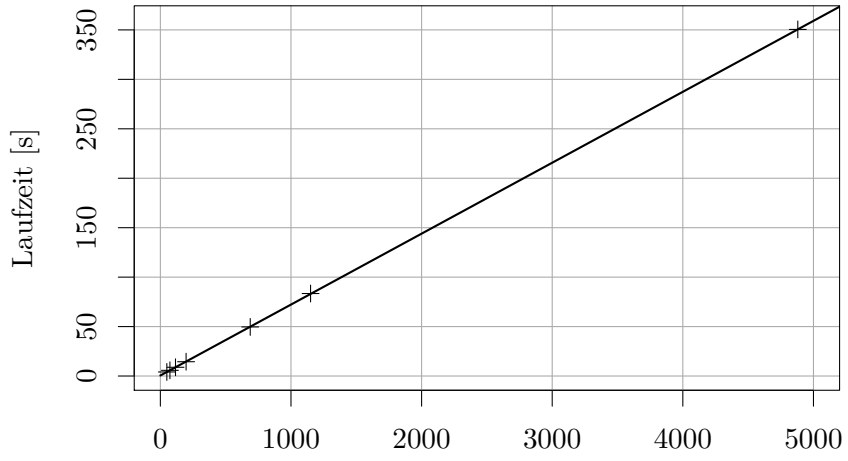

Abbildung 4.5: Laufzeit in Abhängigkeit der Anzahl der verarbeiteten Stützpunkte (in 10^6) für Deu 2016.

als auch abwärts gelegene Knoten betrachtet werden müssen, sind Parkplatzanfragen für Deu 2016 red schneller als für Deu 2016.

Die Laufzeit des Algorithmus für Deu 2006 ist verglichen mit der Laufzeit für Deu 2016 kleiner. Der Grund liegt wie auch bei den *st*-Anfragen in der geringen Anzahl verarbeiteter Stützpunkte, wie es Tabelle 4.5 zu entnehmen ist. Allerdings zeigt sich hier erneut, dass Deu 2006 auf eine Erhöhung des maximalen Approximationsfehlers ε nicht mit der gleichen Beschleunigung der Laufzeit reagiert, wie Deu 2016 und Deu 2016 red. Der Grund liegt wie auch bei den Laufzeiten der *st*-Anfragen in der Gesamtanzahl der Stützpunkte (siehe Tabelle 4.2 und Abbildung 4.1).

4.3 Scheduling

Wir werden nun die Testfälle zur Evaluation des Schedulingalgorithmus für die EU-Regelungen darlegen. Danach wird der Algorithmus bezüglich Laufzeit und Genauigkeit bei verschiedenen Approximationsfehlern für Deu 2016 evaluiert. Anschließend wird die Erfüllbarkeit der generierten Touren bei verschiedenen Rahmenbedingungen und Fahrzeiten geprüft. Zuletzt werden die Ergebnisse von Deu 2016 red mit denen von Deu 2016 verglichen.

4.3.1 Testfälle

Tests wurden nur für die EU-Regelungen durchgeführt, also für $R^{driven} = 16200\,\text{s} = 4{,}5\,\text{h}$ und $R^{break} = 2700\,\text{s} = 45\,\text{min}$. Der Algorithmus wurde wie in Kapitel 3.2 umgesetzt, die Optimierungen zur Reduzierung des Parkplatz-Suchaufwandes aus Kapitel 3.2.2 wurden ebenfalls berücksichtigt.

Die Testfälle wurden folgendermaßen generiert: Eine Tour hat $N \in \{6, 12\}$ Kunden und beginnt zum Zeitpunkt t_0 zufällig gleichverteilt zwischen 5 und 7 Uhr. Wir gehen davon aus, dass der Fahrer zum Zeitpunkt t_0 noch nicht seit der letzten Pause gefahren ist ($c = 0$). Der erste Kunde wird zufällig gleichverteilt aus der Knotenmenge des Graphen gewählt. Vom aktuellen Kunden aus werden alle Knoten gesucht, die auf dem Graphen mit konstanten Fahrzeiten (ohne Beachtung von Staus) mit einer Fahrzeit von $0{,}95 \cdot 2 \cdot R^{driven}/(N-1)$ bis $2 \cdot R^{driven}/(N-1)$ erreichbar sind. Der nächste Kunde wird aus dieser Knotenmenge zufällig gleichverteilt gewählt. Dies wird $N-1$-mal wiederholt, um alle Kundenorte festzulegen.

Jeder Kunde erhält eine Servicezeit $S_i = 0$ und ein Servicefenster $W_i = [t_{begin}, t_{end}]$, das sich öffnet, sobald der Fahrer auf dem zugrundeliegenden Graphen mit konstanten Fahrzeiten am Kunden frühestens ankommen kann, ohne Pausenzeiten zu beachten. Das heißt, $t_{begin} = (\Psi_{1,2} \odot \cdots \odot \Psi_{2,i})(t_0)$ (man beachte, dass die Fahrzeitfunktionen Ψ konstant sind). Das Zeitfenster endet nach der Pausenzeit R^{break} und einer zusätzlichen Pufferzeit $B \in \{0, 15, 45, 60, 90, 120\}$ (in Minuten), nachdem das Zeitfenster sich geöffnet hat, also $t_{end} = t_{begin} + R^{break} + B$.

Jeder Testfall hat also mit konstanten Kantengewichten eine reine Fahrzeit von $0{,}95 \cdot 2 \cdot R^{driven}$ bis $2 \cdot R^{driven}$. Damit sind alle Touren für $B = 0$ erfüllbar, wenn Pausen an jeder beliebigen Position eingelegt werden können (insbesondere kann die Pause nach der Hälfte der Fahrzeit eingelegt werden, womit die gefahrene Zeit seit der letzten Pause R^{driven} nie überschreitet). Hintergrund dieser Art der Testfallerzeugung ist, dass die bisherige Tourenberechnung der PTV Group auf einer zeitunabhängigen Berechnung mit Pausen an beliebigen Stellen arbeitet. Außerdem wird in vielen Publikationen, wie in Kapitel 1.2 zum Stand der Forschung angegeben, ebenfalls die Annahme getroffen, dass Pausen überall möglich sind.

Wir werden nun die Laufzeit des Schedulingalgorithmus, die Auswirkung des Approximationsfehlers ε auf die Genauigkeit des berechneten Tourendes und die Auswirkungen der Pufferzeit B auf die Erfüllbarkeit der generierten Touren untersuchen. Sämtliche Tests wurden mit jeweils 100 Testfällen durchgeführt.

4.3.2 Laufzeit

Wir betrachten nun die Laufzeit für Deu 2016 und Deu 2016 red. Wir setzen die Pufferzeit $B = 120\,\text{min} = 7200\,\text{s}$, damit zunächst jede Tour erfüllbar ist. Die Spalten von Tabelle 4.6 geben folgendes an:

 # Kunden *N*: Anzahl Kunden der getesteten Touren, siehe Kapitel 4.3.1 zur Testfallgenerierung.

maximaler Approximationsfehler ε: Maximaler relativer Approximationsfehler der Kantengewichte in Prozent, $\varepsilon = 0\,\%$ bedeutet, dass exakte Kantengewichte verwendet wurden, $\varepsilon = \text{C}$ bedeutet, dass konstante Fahrzeiten ohne Berücksichtigung von Staus und Verzögerungen betrachtet wurden.

Laufzeit: Durchschnittliche Laufzeit des Algorithmus über alle 100 Testfälle in Millisekunden.

 # Parkplätze: Durchschnittliche Anzahl Parkplätze, die zwischen je zwei Kunden gefunden wurden.

Betrachten wir zunächst Deu 2016 mit $N = 6$ Kunden. Für exakte Fahrzeitfunktionen mit $\varepsilon = 0\,\%$ erhalten wir Laufzeiten von durchschnittlich über einer halben Stunde. Die Laufzeit sinkt bereits bei einem maximalen Approximationsfehler von $\varepsilon = 0,5\,\%$ auf unter 75 Sekunden. Für konstante Fahrzeitfunktionen wird eine durchschnittliche Laufzeit von unter 200 Millisekunden erreicht. Zu beachten ist, dass dazu auch die Optimierung für konstante Fahrzeitfunktionen genutzt wurde. Durch diese muss eine Suche nach Parkplätzen nur dann durchgeführt werden, wenn eine direkte Fahrt zum nächsten Kunden nicht sofort nach Beenden des Service möglich ist, siehe Kapitel 3.2.2.1. Diese Optimierung spiegelt sich auch in der Anzahl der zwischen je zwei Kunden gefundenen Parkplätzen wieder, denn für konstante Fahrzeiten muss nach Konstruktion der Testfälle nur zwischen dem dritten und vierten Kunden nach Parkplätzen gesucht werden. Für alle anderen Fahrten wird nicht nach Parkplätzen gesucht, und somit auch keine gefunden, was die durchschnittlich nur 630 gefundenen Parkplätze für konstante Fahrzeiten erklärt.

Tabelle 4.6: Laufzeiten des Schedulingalgorithmus.

	# Kunden N	ε [%]	Laufzeit [ms]	# Parkplätze
Deu 2016	6	0,00	1.915.040	2113
		0,50	74.672	2127
		1,00	44.755	2141
		2,00	29.006	2166
		C	188	630
	12	2,00	61.621	1092
		C	157	340
Deu 2016 red	6	2,00	5873	208
		C	68	77

Für $N = 12$ Kunden, sowie für Deu 2016 red, wurden aus zeitlichen Gründen keine Tests mit exakten Fahrzeiten durchgeführt, da schon bei $N = 6$ Kunden für Deu 2016 die Laufzeit für die Berechnung des frühesten Endes einer Tour mit exakten Fahrzeiten über eine halbe Stunde dauert. Trotzdem kann für $N = 12$ Kunden eine ungefähr doppelt so lange Laufzeit festgestellt werden, wie für $N = 6$, wenn zeitabhängige Fahrzeiten betrachtet werden. Für konstante Fahrzeiten ist die Laufzeit mit 157 Millisekunden gegenüber 188 Millisekunden für $N = 6$ Kunden sogar kürzer. Für $N = 12$ Kunden und konstante Fahrzeiten wird ebenfalls nur einmal nach Parkplätzen gesucht. Gleichzeitig ist die Suchweite aufgrund näher beieinander liegender Kunden kürzer ist als bei $N = 6$ Kunden. Somit ist die Laufzeit für konstante Fahrzeiten und $N = 12$ Kunden kürzer als für $N = 6$ Kunden. Für Deu 2016 red und $N = 6$ Kunden ist die Laufzeit gegenüber der Instanz Deu 2016 mehr als viermal kürzer. Ein ähnlicher Faktor war schon bei einem Vergleich der Laufzeiten der Parkplatzanfrage zu beobachten und war daher zu erwarten.

4.3.3 Genauigkeit der Approximation

In Tabelle 4.7 ist die durchschnittliche Tourdauer für eine Pufferzeit B von 120 Minuten in Abhängigkeit vom Approximationsfehler ε angegeben.

Bis einschließlich einem Approximationsfehler von $\varepsilon = 1\,\%$ ist die Abweichung des Tourendes im Vergleich zu exakt berechneten Tourenden kleiner als eine Minute. Für $\varepsilon = 2\,\%$ weicht das Tourende weniger als 80 Sekunden im Vergleich zum exakten Ergebnis ab. Um die Tourdauer für praktische Anwendungen nicht zu unterschätzen, können andere Verfahren zur Approximation gewählt werden. Zum einen bietet sich eine Approximation der Fahrzeitfunktionen mit getrennten maximalen Approximationsfehler nach oben und nach unten an (siehe die Erklärungen zur Genauigkeit der Fahrzeitprofile in Kapitel 4.2.1.2),

Tabelle 4.7: Berechnete Tourdauern mit $N = 6$ Kunden.

Instanz	ε [%]	Tourdauer [s]
Deu 2016	0,00	36.140
	0,50	36.122
	1,00	36.108
	2,00	35.961
	C	34.536

zum anderen kann auch Strassers Ansatz verwendet werden, der ebenfalls in Kapitel 4.2.1.2 erwähnt wurde. Nutzt man konstante Fahrzeiten, so beträgt die Abweichung zu exakt berechneten Tourenden über 25 Minuten.

4.3.4 Auswirkung des Puffers B

In diesem Abschnitt werden wir unterschiedliche Pufferzeiten B betrachten. Wir versuchen mit den Pufferzeiten, trotz zeitabhängiger Fahrzeiten und dedizierter Pausenorte die generierten Touren erfüllbar zu machen. Aufgrund der Laufzeit von über einer halben Stunde pro getesteter Tour für exakte Fahrzeitfunktionen werden in der folgenden Tabelle nur Ergebnisse für einen maximalen Approximationsfehler von $\varepsilon = 2\,\%$ sowie für konstante Fahrzeitfunktionen angegeben. In Tabelle 4.8 ist die Erfüllbarkeit von Touren mit unterschiedlichen Pufferzeiten angegeben. Grafisch wird dies für zeitabhängige Kantengewichte mit einem Approximationsfehler von $\varepsilon = 2\,\%$ in Abbildung 4.6 dargestellt. Die Spalten der Tabelle geben folgendes an:

maximaler Approximationsfehler ε: Maximaler relativer Approximationsfehler der Kantengewichte in Prozent, $\varepsilon = \mathrm{C}$ bedeutet, dass konstante Fahrzeiten ohne Berücksichtigung von Staus und Verzögerungen betrachtet wurden.

Puffer B: Pufferzeit B in Minuten für das Ende der Zeitfenster.

erfüllbare Touren: Prozentsatz der Touren, die mit der gesetzten Pufferzeit B erfüllbar sind, jeweils für Touren mit $N = 6$ und $N = 12$ Kunden.

Aufgrund zeitabhängiger Fahrzeiten und dedizierter Pausenorte sind ohne Puffer keine der generierten Touren erfüllbar. Für konstante Fahrzeiten, die keine Staus oder Verzögerungen berücksichtigen, sind für Deu 2016 nur 30 – 35 % der Touren erfüllbar, für Deu 2016 red lediglich 15 %. Während für zeitabhängige Fahrzeiten der Prozentsatz der erfüllbaren Touren für $N = 12$ Kunden erst bei einer Pufferzeit von 90 Minuten auf über die Hälfte wächst, ist dies für $N = 6$ Kunden schon für eine Pufferzeit von 45 Minuten der Fall. Angemerkt werden muss, dass für zeitabhängige Fahrzeiten und eine Pufferzeit von 45 Minuten nur eine Pause eingeplant werden kann, denn ein Teil des 45-minütigen Puffers wird von Staus und Verzögerungen aufgebraucht. Erst ab 60 Minuten Puffer ist eine zweite Pause daher überhaupt denkbar.

Die folgenden Beschreibungen beziehen sich nur auf die Tabelleneinträge mit einem Approximationsfehler mit $\varepsilon = 2\,\%$. Für Deu 2016 und $N = 6$ Kunden sind ab 60 Minuten Pufferzeit alle Touren erfüllbar, während für Deu 2016 red nur 81 % erfüllbar sind. Selbst für 90 Minuten sind noch 1 % der Touren nicht erfüllbar. Für Deu 2016 und $N = 12$ Kunden sind mit einer Pufferzeit von 90 Minuten nur 69 % der Touren erfüllbar. Für eine Tour mit mehreren Kunden ist also mit einer höheren Verzögerung zu rechnen, als für Touren mit weniger Kunden.

Bei $N = 6$ Kunden für Deu 2016 sind 94 % der Touren auch mit einem Puffer von 45 Minuten erfüllbar. Für Deu 2016 red sind es noch 79 %. Es bietet sich also aus Gründen der Laufzeit an, eine Vorauswahl aus den Parkplätzen zu treffen. Gleichzeitig bleiben 79 von 94 Touren mit dieser eingeschränkten Auswahl erfüllbar.

Tabelle 4.8: Erfüllbarkeit der Touren für verschiedene Pufferzeiten B.

	B [min]	ε [%]	erfüllbare Touren [%] $N = 6$	$N = 12$
Deu 2016	0	2,00	0	0
		C	30	35
	15	2,00	5	0
		C	93	76
	45	2,00	94	24
		C	100	100
	60	2,00	100	27
		C	100	100
	90	2,00	100	69
		C	100	100
	120	2,00	100	100
		C	100	100
Deu 2016 red	0	2,00	0	–
		C	15	–
	15	2,00	2	–
		C	89	–
	45	2,00	79	–
		C	100	–
	60	2,00	81	–
		C	100	–
	90	2,00	99	–
		C	100	–
	120	2,00	100	–
		C	100	–

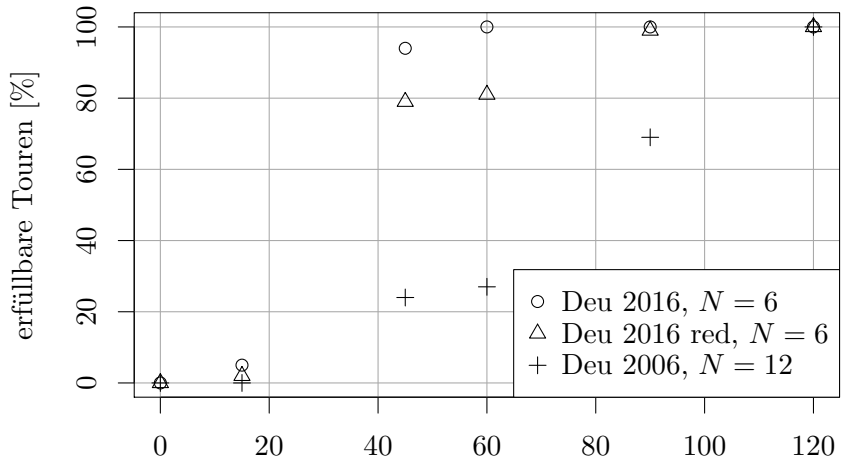

Abbildung 4.6: Erfüllbarkeit der generierten Touren in Abhängigkeit von der Pufferzeit B in Minuten für einen Approximationsfehler von $\varepsilon = 2\,\%$.

5. Fazit und Ausblick

Wir haben gesehen, dass zeitabhängiges LKW-Fahrer-Scheduling mit dedizierten Pausenorten umsetzbar ist. Eine Laufzeit für exakte Fahrzeiten von über einer halben Stunde, die von der Parkplatzsuche dominiert wird, ist allerdings zu hoch für den praktischen Einsatz. Sind keine exakten Ergebnisse gefordert, so kann die Laufzeit auf unter eine Minute pro Tour gesenkt werden. Es hat sich außerdem gezeigt, dass die Betrachtung von einer Teilmenge aller Parkplätze die Laufzeit senkt. Es kann versucht werden, eine Tour nur mit besser ausgestatteten Parkplätzen durchzuführen, um dem Fahrer mehr Komfort auf seiner Fahrt zu bieten. Nur wenn die Tour mit der gewählten Parkplatzteilmenge nicht erfüllbar ist, muss auf weniger gut ausgestattete Parkplätze zurückgegriffen werden.

Werden Zeitpläne nur unter Beachtung zeitunabhängiger Fahrzeiten erstellt, und wird davon ausgegangen, dass Pausen überall möglich sind, dann können diese Zeitpläne in der Praxis nicht durch einen Fahrer eingehalten werden. Insofern besteht der Bedarf eines Schedulingalgorithmus, der zeitabhängige Fahrzeiten und dedizierte Pausenorte berücksichtigt. Im Rahmen dieser Arbeit wurden die zeitabhängigen Fahrzeiten als statisch angenommen. In der Praxis können Straßensperrungen und Verzögerungen bedingt durch Unfälle auftreten. Ein Schedulingalgorithmus, der solche Informationen in Echtzeit verarbeitet, kann die Arbeit des LKW-Fahrers vereinfachen. Gleiches gilt für Informationen über Parkplätze, wie zum Beispiel, ob bei der Ankunft noch ein Stellplatz für den LKW frei ist.

In dieser Arbeit wurde der Approximationsalgorithmus von Imai und Iri zur Beschleunigung der Routenplanungsalgorithmen verwendet. Weitere Tests mit anderen Algorithmen zur Approximation können unter Umständen zu noch besseren Ergebnissen führen. Insbesondere sollte im Sinne eines Zeitplans, der auch in der Praxis umsetzbar sein soll, die Fahrzeit eher über- als unterschätzt werden.

Der in dieser Arbeit beschriebene Schedulingalgorithmus kann auf vielerlei Arten erweitert werden: Die Rekonstruktion des berechneten Zeitplans mit Abfahrts-, Service- und Pausenzeiten wurde noch nicht implementiert. Die Einschränkung, dass zwischen zwei Kunden nur maximal ein Parkplatz eingeplant wird, kann durch die Erweiterung des Algorithmus aufgehoben werden. Außerdem sind von der EU, als auch von den USA weitere Regelungen erlassen worden, die mehrtägige Touren betreffen. Diese können, zusammen mit der Möglichkeit der Pausenteilung in der EU, im Algorithmus umgesetzt werden. Nicht zuletzt bietet es sich an, den Kostenfaktor bedingt durch den Kraftstoffverbrauch und die Maut, oder den CO_2-Austoß des Fahrzeugs durch eine geschickte Wahl des Zeitplans zu senken.

Literaturverzeichnis

[Bat15] Gernot Veit Batz: *KaTCH – Karlsruhe Time-Dependent Contraction Hierarchies*, Juli 2015. `https://github.com/GVeitBatz/KaTCH`, Verwendete Implementierung vom 26. März 2016.

[BCK+10] Reinhard Bauer, Tobias Columbus, Bastian Katz, Marcus Krug und Dorothea Wagner: *Preprocessing Speed-Up Techniques is Hard.* In: *Proceedings of the 7th Conference on Algorithms and Complexity (CIAC'10)*, Band 6078 der Reihe *Lecture Notes in Computer Science*, Seiten 359–370. Springer, 2010.

[BDG+15] Hannah Bast, Daniel Delling, Andrew Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner und Renato F. Werneck: *Route planning in transportation networks.* Technischer Bericht, ArXiv e-prints, 2015.

[BGNS10] Gernot Veit Batz, Robert Geisberger, Sabine Neubauer und Peter Sanders: *Time-Dependent Contraction Hierarchies and Approximation.* In: *Proceedings of the 9th International Symposium on Experimental Algorithms (SEA'10)*, Band 6049 der Reihe *Lecture Notes in Computer Science*, Seiten 166–177. Springer, May 2010. `http://www.springerlink.com/content/u787292691813526/`.

[BGSV13] Gernot Veit Batz, Robert Geisberger, Peter Sanders und Christian Vetter: *Minimum Time-Dependent Travel Times with Contraction Hierarchies.* ACM Journal of Experimental Algorithmics, 18(1.4):1–43, April 2013.

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

[Eur06] Europäisches Parlament und der Rat der Europäischen Union 2006: *Verordnung (EG) Nr. 561/2006 des Europäischen Parlaments und des Rates vom 15. März 2006 zur Harmonisierung bestimmter Sozialvorschriften im Straßenverkehr und zur Änderung der Verordnungen (EWG) Nr. 3821/85 und (EG) Nr. 2135/98 des Rates sowie zur Aufhebung der Verordnung (EWG) Nr. 3820/85 des Rates*, April 2006. In: *Amtsblatt der Europäischen Union*, L102, S. 1–14.

[Fed11] Federal Motor Carrier Safety Administration: *Hours of Service of Drivers*, 2011. Federal Register 76, Nr. 248.

[FHS14] Luca Foschini, John Hershberger und Subhash Suri: *On the complexity of time-dependent shortest paths.* Algorithmica, 68(4):1075–1097, April 2014.

[GAS12] Asvin Goel, Claudia Archetti und Martin Savelsbergh: *Truck Driver Scheduling in Australia.* Computers & Operations Research, 39(5):1122–1132, 2012, ISSN 03050548.

[GK12] Asvin Goel und Leendert Kok: *Truck Driver Scheduling in the United States.* Transportation Science, 46(3):317–326, 2012, ISSN 0041-1655.

[Goe10] Asvin Goel: *Truck Driver Scheduling in the European Union.* Transportation Science, 44(4):429–441, 2010, ISSN 0041-1655.

[Goe12a] Asvin Goel: *The Canadian Minimum Duration Truck Driver Scheduling Problem.* Computers & Operations Research, 39(10):2359–2367, 2012, ISSN 03050548.

[Goe12b] Asvin Goel: *A Mixed Integer Programming Formulation and Effective Cuts for Minimising Schedule Durations of Australian Truck Drivers.* Journal of Scheduling, 15(6):733–741, 2012, ISSN 1094-6136.

[GSSD08] Robert Geisberger, Peter Sanders, Dominik Schultes und Daniel Delling: *Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks.* In: *Proceedings of the 7th Workshop on Experimental Algorithms (WEA'08)*, Band 5038 der Reihe *Lecture Notes in Computer Science*, Seiten 319–333. Springer, June 2008.

[II87] Hiroshi Imai und Masao Iri: *An Optimal Algorithm for Approximating a Piecewise Linear Function.* Journal of information processing, 9(3):159–162, 1987.

[KDDEM16] Mayssa Koubâa, Souhail Dhouib, Diala Dhouib und Abderrahman El Mhamedi: *Truck Driver Scheduling Problem: Literature Review.* IFAC-PapersOnLine, 49(12):1950–1955, 2016.

[KHS11] Adrianus Leendert Kok, Erwin Hans und Johannes Marius Jacobus Schutten: *Optimizing departure times in vehicle routes.* European Journal of Operational Research, 210(3):579–587, 2011.

[Kin16] Monika Kinz: *Optimale Pausenplanung von LKW-Fahrern mit integrierter Parkplatzwahl.* Masterarbeit, Technische Universität Darmstadt, 2016.

[KJL16] Çağri Koç, Ola Jabali und Gilbert Laporte: *Long-haul vehicle routing and scheduling with idling options.* Technischer Bericht, CIRRELT, 2016.

[Kle16] Alexander Kleff: *Truck Driver Scheduling.* Dissertation, 2016. Unveröffentlicht.

[Neu09] Sabine Neubauer: *Space Efficient Approximation of Piecewise Linear Functions.* Studienarbeit, Universität Karlsruhe (TH), Fakultät für Informatik, 2009. `http://algo2.iti.kit.edu/download/neubauer_sa.pdf`.

[Str16] Ben Strasser: *Intriguingly Simple and Efficient Time-Dependent Routing in Road Networks.* CoRR, abs/1606.06636, 2016. `http://arxiv.org/abs/1606.06636`.