Institut für Formale Methoden der Informatik

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

Berechnung optimaler Wege im öffentlichen Verkehr

Jurek Sander

Studiengang:	Informatik
Prüfer/in:	Prof. Dr. Stefan Funke
Betreuer/in:	Claudius Proissl, M.Sc.
Beginn am:	21. September 2021
Beendet am:	03. Februar 2022

Kurzfassung

Diese Bachelorarbeit beschäftigt sich mit der Berechnung optimaler Routen innerhalb von öffentlichen Verkehrsnetzen. Wir betrachten dabei zunächst das Problem der frühesten Ankunftszeit. Dabei wird auf der Eingabe eines Startstopps, eines Zielstopps und der frühesten Abfahrtszeit die Route mit der minimalen Ankunftszeit am Zielstopp berechnet. Für das Problem existieren bereits der Connection Scan Algorithm (CSA) und Round-Based Public Transit Routing (RAPTOR) Algorithmus, die es mit einer unterschiedlichen Herangehensweise lösen. Da in einer realen Anwenungssituation die Reisen mit öffentlichen Verkehrsmitteln von Verspätungen betroffen sind, liefern erwartete Ankunftszeiten bessere Reiserouten für deren Passagiere. Erwartete Ankunftszeiten geben den Erwartungswert der Ankunftszeit von Reisen zwischen zwei Stopps an. Dabei wird die Wahrscheinlichkeit einer Ankunftszeit über die möglichen Verspätungen von Verkehrsmitteln ermittelt. Diesen liegt ein Modell zugrunde, das für jede Verspätung eine Wahrscheinlichkeit definiert. Für die Art der CSAs gibt es bereits eine Lösung zur Minimierung von erwarteten Ankunftszeiten. Wir entwickeln einen neuen Algorithmus, der auf den RAPTOR Algorithmen basiert, um zusätzlich zu der Lösung des Problems der minimalen erwarteten Ankunftszeiten Optimierungen hinsichtlich der Anzahl der Umstiege zu ermöglichen. Die Ausgabe der Ergebnisse der minimalen erwarteten Ankunftszeiten erfolgt über Entscheidungsgraphen, die dem Nutzer Alternativrouten anzeigen, falls er beim Umsteigen durch Verspätungen Anschlusszüge verpasst. Die Eingabe der Anfragen und die anschließende Visualisierung der Ergebnisse ermöglichen wir über eine Weboberfläche. Abschließend vergleichen wir die unterschiedlichen Herangehensweisen und führen Benchmarktests durch. Für die Tests verwenden wir das komplette Netz des Schienenregionalverkehrs und -fernverkehrs in Deutschland.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Algorithmen

Abkürzungsverzeichnis

CSA Connection Scan Algorithm. 3

EAT Earliest Arrival Time. 22

EATP Earliest Arrival Time Profile. 23

ESAT Earliest Safe Arrival Time. 22

ExpAT Expected Arrival Time. 23

GTFS Google Transit Feed Specification. 23

MEAT Minimum Expected Arrival Time. 23

RAPTOR Round-Based Public Transit Routing. 3

TL Transfer Limitation. 48

TO Transfer Optimisation. 47

1 Motivation

Bei der Benutzung des öffentlichen Regional- und Fernverkehrs spielt die Routenplanung eine entscheidende Rolle. Für ihre Nutzer ist es wichtig, bei gezielten Anfragen die optimalen Verbindungen zwischen von ihnen ausgewählten Start- und Zielstopps zu gewünschten Abfahrtszeiten angezeigt zu bekommen. Die Optimalität der Routen kann sich je nach Situation über unterschiedlichste Kriterien definieren. Dies sind unter anderem die früheste Ankunftszeit oder die Anzahl an Umstiegen auf der Reise. In der klassischen Routenberechnung wird die Optimierung üblicherweise auf den planmäßigen Verbindungen des verwendeten Verkehrsnetzes angewandt. Die Berechnung erfolgt dabei unter anderem mit dem CSA und RAPTOR Algorithmus. In der Realität kann es zu Verspätungen der Verkehrsmittel kommen, die sich auf Reisen der Passagiere auswirken. Da diese bei der Routenberechnung, die ausschließlich auf den geplanten Ankunftszeiten basiert, nicht berücksichtigt werden, entspricht die berechnete Optimalität nicht der Optimalität der Realität. Daher ist es unsere Motivation, dass wir schon bei der Routenplanung mögliche Verspätungen berücksichtigen wollen und somit die Optimierung nicht ausschließlich auf den geplanten Verbindungen durchführen. In diesem Bereich gibt es in der Forschung bisher wenige Ansätze. Lediglich für die Art der CSAs wurde durch Dibbelt et al. (2018) in [DPSW18] bereits eine Optimierung der erwarteten Ankunftszeit unter Berücksichtigung von Verspätungen durchgeführt. Für die RAPTOR Algorithmen gibt es noch keine Lösung für diese Problemstellung. Zudem wurde noch nicht untersucht inwiefern weitere Optimalitätskriterien im Zusammenhang mit den erwarteten Ankunftszeiten berücksichtigt werden können und wie sich die Vorteile der Optimierung der tatsächlichen Ankunftszeiten auf die Reisen von Passagieren auswirken. Somit gibt es in diesem Bereich dringenden Forschungsbedarf.

2 Grundlagen

Um optimale Wege in öffentlichen Verkehrsnetzen berechnen zu können, benötigen wir zunächst einige Definitionen. Mit diesen können wir anschließend die Optimalitätsprobleme aufstellen. Für deren Lösung verwenden wir als Grundlage den Datensatz eines öffentlichen Verkehrsnetzes, den wir an unsere Definitionen anpassen müssen.

2.1 Definitionen

Die nachfolgenden Definitionen verwenden wir analog zu Delling et al. (2015) in [DPW15] und Dibbelt et al. (2018) in [DPSW18].

Definition 2.1.1
Innerhalb des Verkehrsnetzes stellen Haltestellen Knotenpunkte dar. An diesen Stopps (Stops) können Passagiere in Verkehrsmittel einsteigen, aussteigen oder zwischen ihnen umsteigen. Die Menge aller Stopps innerhalb eines Netzes bezeichnen wir mit S.

Definition 2.1.2
Verkehrsmittel fahren auf fest definierten Routen (Routes) R. Eine Route stellt dabei eine Teilmenge der Menge der Stopps S dar. Für diese ist innerhalb der Route eine feste Reihenfolge definiert, in der sie abgefahren werden.

In der Abbildung 2.1 wird ein beispielhaftes Verkehrsnetz dargestellt. Dabei repräsentieren die Punkte Stopps zwischen denen die Routen r_1 bis r_4 verlaufen. Verkehrsmittel dieser Routen müssen die zugehörigen Stopps in der dargestellten Reihenfolge abfahren.

Definition 2.1.3
Für jede Route existieren Trips (Trips). Diese definieren die Ankunfts- und Abfahrtszeiten von Verkehrsmitteln an den Stopps der Route. Somit besitzt jeder Trip t' einer Route $r \in R$ an jedem Stopp p von r eine Ankunftszeit $\tau_{arr}(t', p)$ und Abfahrtszeit $\tau_{dep}(t', p)$. Für diese gilt $\tau_{arr}(t', p) \leq \tau_{dep}(t', p)$. Die Ankunftszeit des ersten Stopps einer Route und die Abfahrtszeit des letzten Stopps sind undefiniert. Für die Trips einer Route gilt zudem, dass sie sich gegenseitig nicht überholen dürfen. Alle Trips sind in der Menge T gespeichert.

Definition 2.1.4
Für die CSAs werden zusätzlich auf der Basis der Trips Verbindungen (Connections) C definiert. Jede Verbindung $c \in C$ besteht aus einem Abfahrtsstopp c_{dep_stop}, Ankunftsstopp c_{arr_stop}, einer Abfahrtszeit c_{dep_time}, Ankunftszeit c_{arr_time} und dem zugehörigen Trip c_{trip}. Für die Abfahrts- und Ankunftszeiten muss $c_{dep\_time} < c_{arr\_time}$ gelten. Eine Verbindung ist jeweils nur zwischen zwei Stopps definiert, die innerhalb der Route des Trips direkt aufeinanderfolgen.

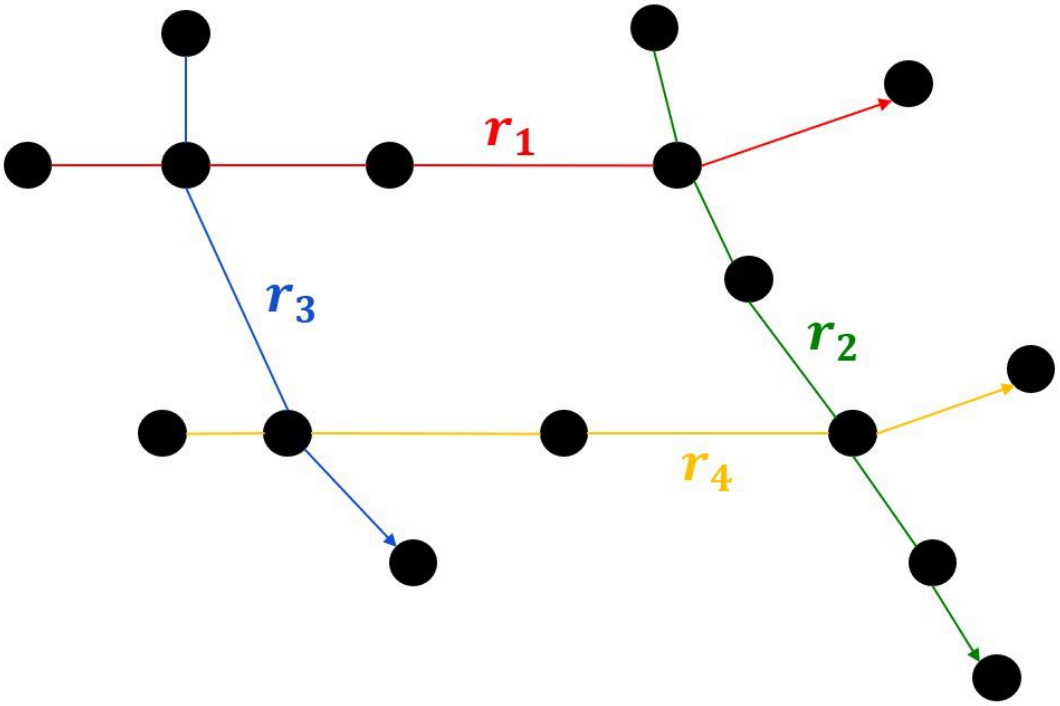

Abbildung 2.1: Ein beispielhaftes Verkehrsnetz. Stopps sind als Punkte dargestellt zwischen denen die Routen r_1 bis r_4 verlaufen.

Definition 2.1.5
Um Wege von Passagieren im Verkehrsnetz beschreiben zu können, benötigen wir Streckenabschnitte (Legs). Jeder Streckenabschnitt l ist einem Trip l_{trip} zugeordnet und definiert sich über den Einstiegspunkt l_{dep_stop} und Ausstiegspunkt l_{arr_stop} des Passagiers und deren Zeiten innerhalb des Trips. Somit ist l über das Tupel $l = (l_{dep\_stop}, l_{arr\_stop}, l_{dep\_time}, l_{arr\_time}, l_{trip})$ gegeben.

Definition 2.1.6
Die Reise (Journey) j eines Passagiers zwischen dem Startstopp s und dem Zielstopp t innerhalb des Verkehrsnetzes kann nun über die Sequenz an Streckenabschnitten $j = l^0, l^1, ..., l^k$ mit $l^0_{dep\_stop} = s$ und $l^k_{arr\_stop} = t$ definiert werden. Sie besitzt die Abfahrtszeit j_{dep_time} an s und Ankunftszeit j_{arr_time} an t. Dabei muss $l^i_{arr\_stop} = l^{i+1}_{dep\_stop}$ und $l^i_{arr\_time} \leq l^{i+1}_{dep\_time}$ für alle $i = 0, ..., k-1$ gelten. Dies gewährleistet, dass ein Umstieg zwischen den Trips zweier benachbarter Streckenabschnitte der Reise möglich ist. Die Reise j besteht dabei aus $k+1$ Trips zwischen denen der Passagier k-Mal umsteigen muss. Jede Reise enthält mindestens einen Streckenabschnitt. Für Reisen können Mengen an Optimierungskriterien definiert werden. Dabei dominiert eine Reise j_1 die Reise j_2, falls j_1 in keinem Kriterium schlechter als j_2 ist. Die Dominanz wird mit $j_1 \leq j_2$ angegeben. Die Pareto-Optimalität der Reise j in einer Menge J an Reisen lässt sich so definieren, dass in J keine andere Reise existiert, die j dominiert.

In der Abbildung 2.2 kann eine mögliche Reise zwischen den Stopps s und t über die Streckenabschnitte l^0 bis l^2 angegeben werden. Die Streckenabschnitte werden dabei jeweils über einen Trip der zugehörigen Route und deren Ein- und Ausstiegsstopps definiert. Dabei entspricht der Abfahrtsstopp von l^0 dem Startstopp s der Reise und der Ankunftsstopp von l^2 dem Zielstopp t.

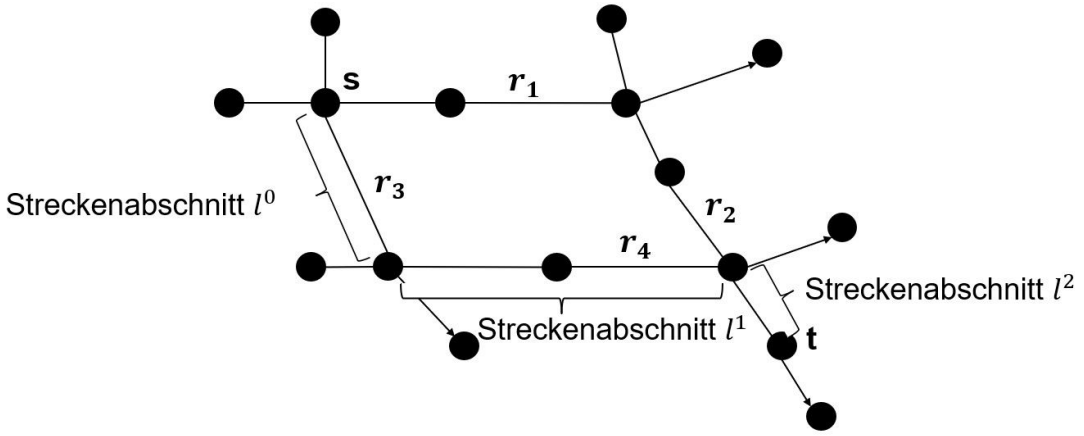

Abbildung 2.2: Eine mögliche Reise zwischen den Stopps s und t kann über die Streckenabschnitte l^0 bis l^2 angegeben werden.

2.1.1 Verspätungen

Für die erwarteten Ankunftszeiten müssen zusätzlich Verspätungen definiert werden. Da wir für unsere Datensätze keine historischen Daten zur Verfügung haben, benötigen wir ein Modell, das uns die Wahrscheinlichkeiten von Verspätungen angibt. Dieses definieren wir analog zu [DPSW18]. Wir notieren dabei alle Zufallsvariablen mit Großbuchstaben und definieren $P[X \leq x]$ als die Wahrscheinlichkeit, dass die Zufallsvariable X kleiner oder gleich groß wie die Konstante x ist. Den Erwartungswert von X notieren wir mit $E[X]$. Für jede Ankunftszeit eines Trips t' an einem Stopp p definieren wir eine maximale Verspätungszeit $\max D(t', p) \geq 0$ in Minuten. Die Wahrscheinlichkeit für mögliche Verspätungen des Trips an diesem Stopp wird über Verteilungsfunktionen f und $\tilde{f}$ mit $f(x) = \begin{cases} 0 & \text{if } x < 0 \\ \tilde{f}(x) & \text{if } 0 \leq x < \max D(t', p) \\ 1 & \text{if } x \geq \max D(t', p) \end{cases}$ definiert. Da es sich um Wahrscheinlichkeiten handelt, muss für jedes x mit $0 \leq x < \max D(t', p)$ Folgendes gelten: $0 \leq \tilde{f}(x) \leq 1$. Dabei gibt x die Verspätung in Minuten an und $f(x)$ die Wahrscheinlichkeit, dass der eingehende Trip eine Verspätung kleiner oder gleich x hat. Wir notieren die Verspätung des Trips t' am Stopp p mit $D(t', p)$ und ihren Erwartungswert mit $E[D(t', p)]$.

Damit wir erwartete Ankunftszeiten berechnen können, müssen wir zusätzlich zu dem Verspätungsmodell Annahmen treffen. Die Erste besagt, dass alle Zufallsvariablen unabhängig voneinander sind. Dies benötigen wir, um sie multiplizieren zu können. In der Realität ist die Unabhängigkeit nicht gewährleistet, da es hier zu gegenseitigen Beeinflussungen von Verspätungen zwischen Verkehrsmitteln kommen kann. Diese haben dann jedoch so große Auswirkungen auf die Reise von Passagieren, dass es kaum möglich ist über ein Modell alle potenziellen Auswirkungen von Verspätungen inklusive der dann benötigten Ausweichpläne zu modellieren. Sollte es somit zu Ereignissen (z.B. großflächige Sperrungen von Bahnstrecken) kommen, aus denen sehr viele Verspätungen resultieren, ist es für Nutzer am besten, abhängig von diesen neue Routen zu planen. Somit nehmen wir bei unserem Modell an, dass es nur für übliche Verspätungen robust ist. Eine weitere Annahme, die wir in unserem Modell treffen ist, dass die Verkehrsmittel aller Trips pünktlich abfahren und Verspätungen

somit nur bei ihren Ankünften betrachtet werden. Abschließend verwenden wir, dass Verspätungen innerhalb eines Trips unabhängig sind. Dies hat auf unser Modell jedoch eine geringe Auswirkung, da es kaum optimale Reisen gibt, bei denen man in einen Trip mehr als einmal einsteigen muss.

Definition 2.1.7
Mit dem Verspätungsmodell können sichere Reisen definiert werden. Eine sichere Reise garantiert, dass der Passagier trotz möglicher Verspätungen der genutzten Trips keinen Trip der Reise verpasst. Somit muss die Umsteigezeit an jedem Umstiegspunkt der Reise mindestens so groß wie die maximale Verspätung des eingehenden Trips an diesem Stopp sein.

2.1.2 Entscheidungsgraphen

Um erwartete Ankunftszeiten zu definieren, verwenden wir (s, t, τ_s)-Entscheidungsgraphen analog zu [DPSW18]. Diese bestehen aus Reisen zwischen dem Startstopp s und dem Zielstopp t mit einer frühesten Abfahrtszeit τ_s. Ein Entscheidungsgraph $G = (V, E)$ besteht aus einer Menge an Stopps als Knoten V und einer Menge an Streckenabschnitten als Kanten E. Die Kanten verlaufen dabei jeweils zwischen $l_{\text{dep\_stop}}$ und $l_{\text{arr\_stop}}$ der zugehörigen Streckenabschnitte l. Falls mehrere Kanten zwischen zwei Knoten exisitieren, müssen die zugehörigen Streckenabschnitte unterschiedliche Abfahrts- und Ankunftszeiten besitzen. Es muss gewährleistet sein, dass jeder Streckenabschnitt $l \in E$ für einen Passagier erreichbar ist. Daher muss für l eine Reise zwischen s und $l_{\text{dep\_stop}}$ mit einer Abfahrtszeit von frühestens τ_s existieren. Die Ankunftszeit dieser Reise muss vor der Abfahrtszeit $l_{\text{dep\_time}}$ liegen. Zusätzlich benötigt jeder Streckenabschnitt l eine sichere Reise, die von $l_{\text{arr\_stop}}$ zu t führt und nach $l_{\text{arr\_time}} + \max D(l_{\text{trip}}, l_{\text{arr\_stop}})$ abfährt. Der Graph G enthält abgesehen von s und t ausschließlich Knoten, die mindestens eine eingehende und ausgehende Kante besitzen. Die erwartete Ankunftszeit $\text{expAT}(G)$ des Graphen G kann über die erwarteten Ankunftszeiten seiner Streckenabschnitte bestimmt werden. Die erwartete Ankunftszeit eines Streckenabschnitts $l \in E$ wird rekursiv per Fallunterscheidung definiert und mit $\text{expAT}(l)$ notiert. Falls $l_{\text{arr\_stop}} = t$ gilt, beträgt die erwartete Ankunftszeit $\text{expAT}(l) = l_{\text{arr\_time}} + E[D(l_{\text{trip}}, l_{\text{arr\_stop}})]$. Sie setzt sich somit aus der geplanten Ankunftszeit des zugehörigen Trips und seiner erwarteten Verspätung zusammen. Falls l nicht zu dem Zielstopp t führt, wird die erwartete Ankunftszeit über die ausgehenden Streckenabschnitte des Knotens $l_{\text{arr\_stop}}$ definiert. Sei dazu $l_1, l_2, ..., l_n$ die nach den Abfahrtszeiten sortierte Sequenz an ausgehenden Streckenabschnitten an $l_{\text{arr\_stop}}$, die nach $l_{\text{arr\_time}}$ abfahren. Seien $d_1, d_2, ..., d_n$ die zu den Streckenabschnitten zugehörigen Abfahrtszeiten und $d_0 = l_{\text{arr\_time}}$. Die erwartete Ankunftszeit von l wird nun über $\text{expAT}(l) = \sum_{i \in \{1,...,n\}} P[d_{i-1} < l_{\text{arr\_time}} + D(l_{\text{trip}}, l_{\text{arr\_stop}}) \leq d_i] \cdot \text{expAT}(l_i)$ mit $D(l_{\text{trip}}, l_{\text{arr\_stop}})$ als Verspätung von dem Trip l_{trip} an $l_{\text{arr\_stop}}$ definiert. In diesem Fall stellt die erwartete Ankunftszeit die gewichtete Aufsummierung der erwarteten Ankunftszeiten der ausgehenden Trips dar, in die der Passagier umsteigen kann. Die Gewichtung der Streckenabschnitte l_i erfolgt dabei jeweils über die Wahrscheinlichkeit, dass der Passagier in den zugehörigen Trip $l_{i\,\text{trip}}$ umsteigt. Sie summiert sich zu eins auf, da laut Definition für jeden eingehenden Streckenabschnitt l am Stopp $l_{\text{arr\_stop}}$ eine sichere Reise zu t existiert. Den Trip des ersten Streckenabschnitts dieser Reise kann der Nutzer somit in jedem Fall an dem Stopp $l_{\text{arr\_stop}}$ nehmen. Sei G^{first} der Streckenabschnitt aus E mit der frühesten Abfahrtszeit. Diesen muss der Nutzer initial am Stopp s wählen. Die erwartete Ankunftszeit $\text{expAT}(G)$ des Entscheidungsgraphen G kann nun als $\text{expAT}(G^{\text{first}})$ angegeben werden. Zusätzlich wird die späteste Ankunftszeit des Graphen mit $G_{\text{max\_arr}}$ notiert und definiert sich über die maximale Ankunftszeit $l_{\text{arr\_time}}$ aller $l \in E$. Für den Nutzer ergibt sich die Reise, die er dem Graph zufolge nehmen sollte, jeweils über den ersten Streckenabschnitt, den er an einem

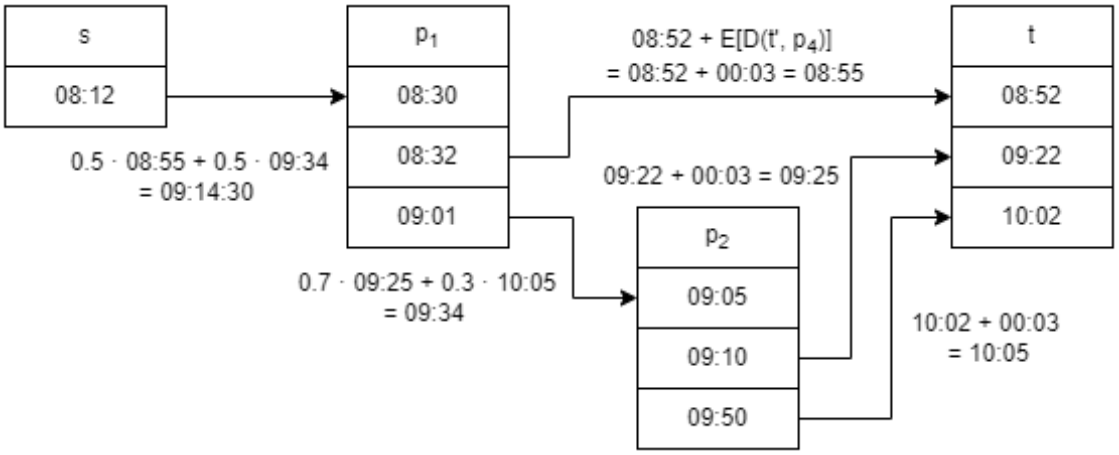

Abbildung 2.3: Ein Entscheidungsgraph zwischen den Stopps s und t inklusive der erwarteten Ankunftszeiten seiner Streckenabschnitte. Für die Berechnung der erwarteten Ankunftszeiten wurden beispielhafte Werte für die Wahrscheinlichkeiten und Erwartungswerte der Verspätungen gewählt.

Umstiegspunkt nach seiner Ankunft erreicht. Dabei bezeichnen wir als Alternativreisen alle Reisen, die er nur deswegen nimmt, da er durch eine Verspätung des eingehenden Trips die erste Möglichkeit des Graphen verpasst.

In Abbildung 2.3 ist das Beispiel eines (s, t, τ_s)-Entscheidungsgraphen G mit einer minimalen Abfahrtszeit τ_s von 8:12 Uhr inklusive der Berechnungen der erwarteten Ankunftszeiten seiner Streckenabschnitte dargestellt. Dazu wurden beispielhafte Werte für die Wahrscheinlichkeiten und Erwartungswerte von Verspätungen verwendet. Als maximale Verspätung wählen wir für jeden Trip eine halbe Stunde. Der Entscheidungsgraph kann so interpretiert werden, dass ein Passagier, der von s nach t reisen will, zunächst in den Trip an Stopp s einsteigt, der um 8:12 Uhr abfährt und um 8:30 Uhr an Stopp p_1 ankommt. An diesem Stopp steigt er aus und wählt je nach Verspätung des eingehenden Verkehrsmittels den ersten Streckenabschnitt des Entscheidungsgraphen, dessen Trip er erreichen kann. Sollte er also vor 8:32 Uhr ankommen, kann er direkt zu Stopp t fahren. Andernfalls muss er auf den Trip, der um 9:01 Uhr abfährt warten, und über p_2 zu t reisen. Entscheidungsgraphen sind in der Realität deutlich komplexer als in diesem Beispiel und können auch mehr als nur eine Alternativreise an Umstiegspunkten enthalten. Die erwarteten Ankunftszeiten der Streckenabschnitte des Beispiels berechnen sich über die definierte Fallunterscheidung. Für die drei Streckenabschnitte, die zum Zielstopp t führen, ergibt sie sich über die Summe aus der geplanten Ankunftszeit und der erwarteten Verspätung. Diese beträgt in unserem Beispiel für jeden Trip drei Minuten. An den beiden anderen Kanten wird die erwartete Ankunftszeit über die gewichtete Aufsummierung der Werte der ausgehenden Streckenabschnitte an ihren Ankunftsstopps berechnet. Für die Gewichtung werden die beispielhaften Wahrscheinlichkeiten verwendet. Die erwartete Ankunftszeit des Entscheidungsgraphen G entspricht nun der erwarteten Ankunftszeit seines ersten Streckenabschnitts und beträgt somit 9:14:30 Uhr. Die späteste Ankunftszeit $G_{\text{max\_arr}}$ ergibt sich über die Ankunftszeit von 10:02 Uhr am Zielstopp t.

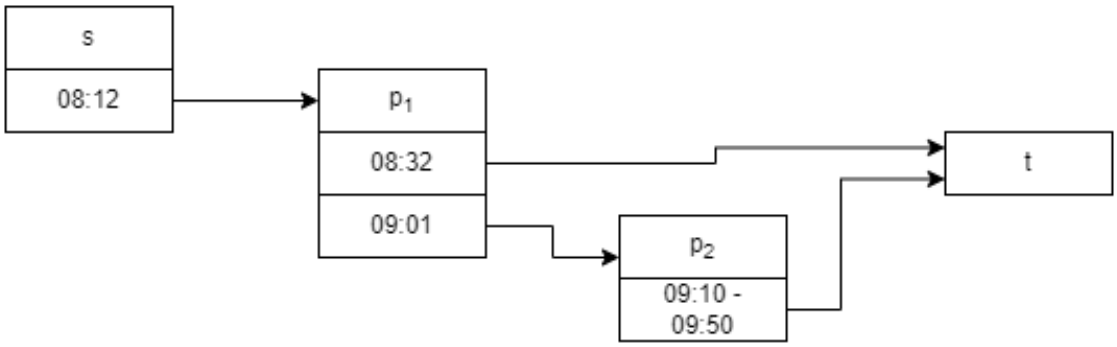

Abbildung 2.4: Ein beispielhafter kompakter Entscheidungsgraph.

Entscheidungsgraphen können auf unterschiedliche Arten visualisiert werden. Wir verwenden die erweiterte und die kompakte Darstellung der Graphen. Bei beiden Varianten werden alle Stopps als Knoten dargestellt und mit dem Namen der zugehörigen Haltestelle beschriftet. Bei der erweiterten Darstellung sind die Knoten in Zeilen unterteilt, die jeweils die Ankunftszeit oder Abfahrtszeit eines Streckenabschnitts enthalten, der zu dem Stopp hinführt oder von ihm abfährt. Die Zeilen sind dabei aufsteigend nach ihren Zeiten sortiert. Streckenabschnitte werden als Pfeile zwischen der Abfahrtszeit an ihrem Ausgangsstopp und der Ankunftszeit an ihrem Ankunftsstopp dargestellt. In der kompakten Darstellung wird nur ein Teil der Informationen der erweiterten Variante angezeigt. In den Zeilen der Stopps sind lediglich die Abfahrtszeiten seiner ausgehenden Streckenabschnitte enthalten. Falls mehrere Streckenabschnitte eines Stopps den gleichen Ankunftsstopp besitzen, können ihre Abfahrtszeiten in einer Zeile zusammengefasst werden. In diese wird lediglich das Intervall zwischen der frühesten und spätesten Abfahrtszeit geschrieben. Streckenabschnitte werden als Pfeile zwischen ihrer Abfahrtszeit und dem Ankunftsstopp dargestellt. Für Intervalle an Abfahrtszeiten, wird nur ein Pfeil zu dem gemeinsamen Ankunftsstopp benötigt.

In Abbildung 2.4 ist der Entscheidungsgraph aus Abbildung 2.3 in seiner kompakten Version dargestellt. Dazu wurden die beiden Kanten, die von p_2 zu t führen, zusammengefasst und die Ankunftstzeiten entfernt.

2.2 Problembeschreibung

Wir beschäftigen uns mit unterschiedlichen Optimalitätskriterien für Reisen zwischen zwei Stopps und definieren hierfür die Probleme analog zu Dibbelt et al. (2018) in [DPSW18].

Das erste Problem ist das Problem der frühesten Ankunftszeiten (Earliest Arrival Time (EAT)). Dieses gibt auf der Eingabe eines Startstopps $s \in S$, Zielstopps $t \in S$ und einer Startzeit τ_s die minimale Ankunftszeit aller Reisen an, die an Stopp s nach τ_s abfahren und an Stopp t ankommen.

Eine Abwandlung des EAT Problems ist das Problem der frühesten sicheren Ankunftszeit (Earliest Safe Arrival Time (ESAT)), das die gleiche Eingabe besitzt. Hier wird die minimale Ankunftszeit aller sicheren Reisen gesucht, die an Stopp s nach τ_s abfahren und an Stopp t ankommen.

Das EAT Problem kann auf das Problem der frühesten Ankunftszeit im Interval (Earliest Arrival Time Profile (EATP)) erweitert werden. Dabei ist als Eingabe ein Startstopp s, Zielstopp t, eine minimale Abfahrtszeit τ_s und eine maximale Ankunftszeit τ_t gegeben. Gesucht sind als Ausgabe alle Paare $(j_{\text{dep\_time}}, j_{\text{arr\_time}})$ der Reisen j für die Folgendes gilt: j fährt nach τ_s an Stopp s ab und kommt vor τ_t an Stopp t an. Zudem muss das Paar $(j_{\text{dep\_time}}, j_{\text{arr\_time}})$ unter Berücksichtigung aller Reisen Pareto-Optimal bezüglich Abfahrtszeit und Ankunftszeit sein.

Das Problem der erwarteten Ankunftszeiten (Expected Arrival Time (ExpAT)) sucht auf der Eingabe eines Startstopps s, eines Zielstopps t, einer minimalen Abfahrtszeit τ_s und einer maximalen Ankunftszeit τ_t die erwartete Ankunftszeit des Entscheidungsgraphen der Reise, die das zugehörige EAT Problem löst. An den Umstiegspunkten des Graphen werden die Alternativreisen jeweils nach der minimalen Ankunftszeit ausgewählt. Die Pareto-Optimalität ist über die Abfahrtszeit und Ankunftszeit der Reisen definiert. Somit spannen wir bei diesem Problem einen Entscheidungsgraphen zwischen den Reisen auf, die wir in dem Intervall des EATP Problems als Reisen mit den frühesten Ankunftszeiten erhalten haben. Mit unserer Definition der Entscheidungsgraphen und ihren zugehörigen erwarteten Ankunftszeiten ist es dementsprechend möglich die erwarteten Ankunftszeiten von den schnellsten Reisen zu berechnen.

Zuletzt definieren wir das Problem der minimalen erwarteten Ankunftszeiten (Minimum Expected Arrival Time (MEAT)) in drei Varianten. Als Eingabe steht dabei jeweils der Startstopp s, Zielstopp t und die minimale Abfahrtszeit τ_s. In der unbeschränkten Variante ist der (s, t, τ_s)-Entscheidungsgraph G mit der minimalen erwarteten Ankunftszeit $\text{expAT}(G)$ gesucht. Die beschränkte MEAT Variante minimiert zusätzlich die maximale Ankunftszeit $G_{\text{max\_arr}}$. Als Kompromiss fungiert die α-beschränkte Variante, da hier die maximale Ankunftszeit auf $G_{\text{max\_arr}} \leq \tau_s + \alpha \cdot (\text{ESAT}(s, t, \tau_s) - \tau_s)$ mit $\alpha \geq 1$ gesetzt und daraufhin die erwartete Ankunftszeit von G minimiert wird. Für $\alpha = 1$ entspricht die α-beschränkte Variante der beschränkten Variante, da durch die Definition der Entscheidungsgraphen $G_{\text{max\_arr}} \geq \text{ESAT}(s, t, \tau_s)$ gilt. Wie wir später zeigen werden, liefert lediglich die unbeschränkte Variante des MEAT Problems den optimalen Entscheidungsgraphen hinsichtlich der minimalen erwarteten Ankunftszeit. Jedoch müssen hierbei alle Trips betrachtet werden, die nach der minimalen Abfahrtszeit für Passagiere erreichbar sind. Da die von uns verwendeten Datensätze eine unendliche Anzahl an Trips enthalten, die potenziell erreichbar sind, können wir den Algorithmus nicht in seiner unbeschränkten Version durchführen. Die α-beschränkte Variante des Algorithmus garantiert, dass wir lediglich eine endliche Anzahl an Trips berücksichtigen müssen und er somit ausführbar ist. Die Anzahl der Trips kann über den Parameter α beliebig gesteuert werden. Den Einfluss, den dieser Parameter auf die Laufzeit und die Ergebnisse des Algorithmus hat, analysieren wir in Kapitel 5.

2.3 Datensatz

Als Datengrundlage für unsere Algorithmen nutzen wir die kompletten Datensätze des Schienenregionalverkehrs und -fernverkehrs in Deutschland. Die Daten liegen im Google Transit Feed Specification (GTFS) Format vor. Damit sie von unseren Algorithmen genutzt werden können, müssen die Daten so angepasst werden, dass sie den Definitionen aus dem Kapitel 2.1 entsprechen.

2.3.1 GTFS Format

Das GTFS Format besteht aus mehreren Textdateien, die jeweils Tabellen mit den Werten der zugehörigen Felder enthalten. Entscheidend für unsere Algorithmen sind die Tabellen Stops, Routes, Trips, Stop_Times und Calendar.

In der Tabelle Stops ist für jeden Stopp eine ID über die er referenziert werden kann, sein Name als String und seine Koordinaten gespeichert. Die Tabelle Routes enthält die IDs der Routen, ihre Namen und Typen. Über die Tabelle Trips erfolgt die Zuweisung der Trips zu den Routen, da für jeden Trip dessen ID und die zugehörige Routen ID gespeichert ist. Zusätzlich befindet sich in der Trip Tabelle der Service ID Eintrag. Dieser verweist auf die Calendar Tabelle. Jeder Eintrag dieser Tabelle besitzt eine eindeutige Service ID und gibt über Booleans für die Tage Montag bis Sonntag an, an welchen Tagen ein bestimmter Trip verfügbar ist. Dies ist entscheidend für unsere Algorithmen, da für sie die Information benötigt wird, an welchen Wochentagen Trips und somit ihre zugehörigen Verkehrsmittel genutzt werden können. Die Tabelle Stop_Times definiert die Abfahrtszeiten der Trips an den zugehörigen Stopps. Somit besitzt diese Tabelle die Felder Arrival_Time und Departure_Time zur Angabe der Zeiten und Stop_ID und Trip_ID, um auf die zugehörigen Trips und Stopps zu verweisen. Der Eintrag Sequence gibt an als wievielter Stopp der zugehörige Stopp innerhalb des Trips angefahren wird.

Die Tabellen importieren wir jeweils so, dass alle Einträge in einem zugehörigen Array gespeichert werden. Die Position innerhalb des Arrays stimmt dabei jeweils mit der ID des zugehörigen Eintrages überein, um einen direkten Zugriff zu ermöglichen.

Zusätzlich sind in weiteren Tabellen der GTFS Dateien Informationen über die Agenturen der Verkehrsmittel des Netzes gespeichert. Diese haben jedoch für unsere Algorithmen keine weitere Bedeutung und müssen nicht importiert werden.

2.3.2 Anpassungen

Für die Algorithmen werden zusätzlich zu den Daten weitere Datenstrukturen benötigt, die aus den Tabellen des GTFS Formats generiert werden können. Diese sind notwendig, um die Algorithmen effizient zu gestalten und müssen einmalig nach dem Einlesen eines GTFS Datensatzes erstellt werden. Zudem passen wir die importierten Datenstrukturen so an, dass sie die Bedingungen unserer Definitionen aus dem Kapitel 2.1 erfüllen.

Zunächst ist für die Routen des GTFS Formats nicht gesichert, dass alle zugehörigen Trips die exakt gleichen Stopps in derselben Reihenfolge abfahren. Da dies durch unsere Definitionen vorgegeben ist und für die RAPTOR Algorithmen benötigt wird, müssen wir basierend auf den Trips neue Routen so erstellen, dass die Bedingung gesichert ist. Zusätzlich füllen wir in dem gleichen Schritt vier Zeiger Arrays, die uns einen schnellen Zugriff auf die Daten ermöglichen. Hier speichern wir für jede Routen ID die IDs der zugehörigen Stopps in der Reihenfolge, in der sie in der Route abgefahren werden. Andersherum benötigen wir zudem für jede Stopp ID die IDs der Routen, in denen der Stopp enthalten ist. Für die Routen müssen wir zusätzlich alle IDs der Trips speichern, die zu der Route gehören. Zuletzt benötigen wir für jeden Trip die Position seines ersten Stoppzeit-Eintrages, um effizient auf alle seine Abfahrts- und Ankunftszeiten innerhalb des Arrays zugreifen zu können.

Damit wir die neuen Routen und die zusätzlichen Zeiger Arrays erstellen können, sortieren wir zunächst das Array der Stoppzeiten nach ihren Trips und innerhalb eines Trips nach der Sequenz seiner Stopps. Die Zeiger Arrays initialisieren wir jeweils mit einem leeren Array. Anschließend iterieren wir einmal über das Stoppzeit-Array. Dabei speichern wir so lange die zugehörigen Stopps bis sich die Trip ID der betrachteten Einträge ändert. An diesem Punkt überprüfen wir, ob wir bereits eine neue Route mit der Reihenfolge dieser Stopps erstellt haben. Ist dies der Fall, fügen wir die Trip ID in das Array, das die Trips einer Route speichert, für diese Route hinzu. Andernfalls erstellen wir eine neue Route mit zugeordneter Route ID. Dazu kopieren wir die Routen Informationen der zugehörigen Route aus der GTFS Tabelle und ersetzten deren ID mit der neuen ID. Für diese speichern wir die Reihenfolge der zugehörigen Stopps und die Routen ID für jeden der Stopps. Zusätzlich speichern wir die Trip ID für diese Route. Abschließend aktualisieren wir in beiden Fällen den Eintrag der Routen ID in dem Array der Trips, die aus den GTFS Daten importiert wurden. Für den darauffolgenden ersten Stoppzeit-Eintrag des nächsten Trips speichern wir dessen Position innerhalb des Stoppzeiten-Arrays für den zugehörigen Trip.

Nach der Erstellung der neuen Routen, nutzen wir die Abfahrtszeiten der Trips, um das Zeiger Array, das alle Trips einer Route speichert, zu sortieren. Damit erreichen wir, dass alle Trips einer Route aufsteigend bezüglich der Abfahrtszeit der Trips an dem ersten Stopp gespeichert sind. Dies können wir somit in den Algorithmen als Annahme verwenden.

Wie in Kapitel 2.1 bereits geschildert, benötigen wir für die Art der CSAs Verbindungen. Diese können auch aus den Stoppzeit-Einträgen erstellt werden.

Dazu nutzen wir, dass die Stoppzeit-Einträge schon nach der Trip ID und Sequenz der Stopps sortiert sind. So reicht es aus, einmal über das Array zu iterieren. Dabei erstellen wir jeweils für zwei aufeinanderfolgende Stoppzeiten des gleichen Trips eine neue Verbindung. In diese wird der Abfahrtsstopp und die Abfahrtszeit der ersten Stoppzeit und der Ankunftsstopp und die Ankunftszeit des zweiten Eintrages gespeichert. Zudem fügen wir die gemeinsame Trip ID der beiden Stoppzeiten hinzu. Nach dem Erstellen aller Verbindungen sortieren wir sie nach ihren Abfahrtszeiten, damit wir dies für die CSAs verwenden können.

3 Bestehende Algorithmen

Es existieren bereits unterschiedliche Ansätze, um Reisen innerhalb eines öffentlichen Verkehrsnetzes zu planen. Da Verkehrsnetze durch Graphen dargestellt werden können, wobei die Stopps ihren Knoten und die Kanten den geplanten Routen der Verkehrsmitteln entsprechen, ist es möglich, zur Lösung des EAT Problems eine angepasste Version des Dijkstra Algorithmus in [Dij59] durchzuführen. In [BGM10] verbessern Berger et al. (2010) mit ihrem SUBITO Algorithmus die Variante des Dijkstra Algorithmus, indem sie untere Grenzen verwenden. Delling et al. (2012) haben in [DKP12] den SPCS Algorithmus eingeführt, um das EATP Problem zu lösen. Für ihren Algorithmus gibt es zusätzlich eine parallele Variante. Die Herangehensweise des TB Algorithmus in [Wit15] basiert darauf, dass in seiner Vorbereitungsphase alle möglichen Transfers zwischen Trips berechnet werden. Diese dauert damit aber deutlich länger als bei den anderen vorgestellten Algorithmen.

Um Reisen innerhalb des Verkehrsnetzes nach mehr als einem Kriterium zu optimieren, wurde der Layered Dijkstra Algorithmus von Brodal et al. (2004) in [BJ04] eingeführt. Dieser ermöglicht die Verwendung eines zweiten Kriteriums auf der Basis des Dijkstra Algorithmus. Der MLC Algorithmus von Pygra et al. (2008), der in [PSWZ08] und [DW09] vorgestellt wurde, basiert auch auf dem Dijkstra Algorithmus, aber ermöglicht eine Verwendung von mehr als zwei Kriterien. Er wurde von Disser et al. (2008) in [DMS08] optimiert.

Unsere Algorithmen zur Lösung der ExpAT und MEAT Probleme basieren auf den Konzepten der CSAs und RAPTOR Algorithmen. Diese existieren jeweils in zwei unterschiedlichen Varianten, um das EAT und EATP Problem zu lösen. Für die CSAs gibt es bereits zusätzlich einen Algorithmus für das MEAT Problems.

3.1 CSAs

Die nachfolgenden Algorithmen, die von Dibbelt et al. (2018) in [DPSW18] vorgestellt wurden, basieren auf dem grundlegenden Konzept des CSA. Sie benötigen lediglich die Verbindungen, die aus den Stoppzeiten der Trips erstellt wurden und verwenden somit nicht die Informationen zu welchen Routen deren Trips zugeordnet sind. Bei den CSAs wird angenommen, dass die Verbindungen nach ihren Abfahrtszeiten sortiert sind. Die Verwendung der Verbindungen in dieser Reihenfolge garantiert, dass jede Verbindung während der Durchführung des Algorithmus maximal einmal genutzt wird. Wir betrachten drei Varianten der CSAs. Die erste berechnet früheste Ankunftszeiten und die zweite die schnellsten Reisen, die in einem bestimmten Intervall am Zielstopp ankommen. Diese Profile CSA Variante kann so angepasst werden, dass sie Lösungen für das MEAT Problem liefert.

3.1.1 CSA

Mit der Standardvariante des CSA können minimale Ankunftszeiten berechnet und die zugehörigen Reisen ermittelt werden. Als Eingabe dient dabei der Startstopp s, der Zielstopp t und eine minimale Abfahrtszeit τ_s. Der Algorithmus liefert somit die Lösung für das EAT Problem.

Der CSA benötigt drei Arrays. In dem S-Array wird für jeden Stopp die früheste Ankunftszeit gespeichert, zu der er ausgehend von dem Startstopp s und der Startzeit τ_s erreicht werden kann. In dem Array T wird für jeden Trip die erste Verbindung gespeichert, über die es für einen Passagier möglich ist, in das dem Trip zugehörige Verkehrsmittel einzusteigen. Um die zugehörige Reise nach der Berechnung der frühesten Ankunftszeit extrahieren zu können, speichern wir in dem Array J für jeden Stopp einen Reise Zeiger. Dieser verweist auf den Trip, der den Passagier zu dem Stopp führt, für den der Zeiger erstellt wurde. Als Zeiger speichern wir dafür die erste erreichbare Verbindung des Trips und die Verbindung des Trips, die den Stopp des Zeigers als Ankunftsstopp besitzt. Über diese beiden Verbindungen ist definiert, zu welchen Zeitpunkten und an welchen Stopps der Passagier in den Trip ein- und aussteigt.

In der Initialisierungsphase des Algorithmus speichern wir für jeden Stopp p abgesehen von dem Startstopp s in S $S[p] \leftarrow \infty$. Für s setzen wir $S[s] \leftarrow \tau_s$. Die Einträge der Arrays T und J initialisieren wir für alle Trips und Stopps jeweils undefiniert. Zudem ermitteln wir mit einer binären Suche die erste Verbindung mit einer Abfahrtszeit, die mindestens τ_s beträgt. Dies ist die erste Verbindung, die von Passagieren genutzt werden kann.

Der CSA betrachtet alle Verbindungen, die hinsichtlich ihrer Abfahrtszeit auf diese erste Verbindung folgen. Dabei werden die folgenden Schritte für jede Verbindung c durchgeführt: Zunächst überprüfen wir, ob die Abfahrtszeit $c_{\text{dep\_time}}$ größer als die früheste bekannte Ankunftszeit des Zielstopps t ist. Da die Verbindungen nach ihren Abfahrtszeit sortiert sind, kann ab dieser Verbindung keine kleinere Ankunftszeit an t gefunden werden. Somit können wir an dieser Stelle den Algorithmus terminieren. Ist die Terminierungsbedingung nicht erfüllt, überprüfen wir, ob die Verbindung erreichbar ist. Dies ist möglich, falls der Trip c_{trip} der Verbindung bereits erreicht wurde oder an dem Stopp $c_{\text{dep\_stop}}$ der Zustieg in den Trip möglich ist. Ist der Eintrag $T[c_{\text{trip}}]$ bereits durch eine Verbindung definiert, kann der zugehörige Trip schon durch eine frühere Verbindung erreicht werden. Andernfalls überprüfen wir, ob $S[c_{\text{dep\_stop}}] \leq c_{\text{dep\_time}}$ gilt und es somit möglich ist an diesem Stopp in den Trip einzusteigen. Ist dies der Fall, speichern wir die Verbindung c in $T[c_{\text{trip}}]$ ab. Falls eine der beiden Möglichkeiten erfüllt ist, setzen wir die Ankunftszeit der Verbindung in $S[c_{\text{arr\_stop}}]$ auf $c_{\text{arr\_time}}$, wenn $c_{\text{arr\_time}}$ kleiner als der bisherige Wert in $S[c_{\text{arr\_stop}}]$ ist. Sollten wir so die früheste Ankunftszeit an diesem Stopp aktualisiert haben, speichern wir als zugehörigen Reise Zeiger das Tupel $(T[c_{\text{trip}}], c)$ für den Stopp $c_{\text{arr\_stop}}$ in dem Array J.

Sobald der Algorithmus terminiert ist, steht die früheste Ankunftszeit am Zielstopp t in $S[t]$. Die zugehörige Reise extrahieren wir ausgehend von t. Dazu starten wir mit dem Reise Zeiger, der in $J[t]$ gespeichert ist. Aus diesem können wir den Streckenabschnitt, der zwischen den beiden darin enthaltenen Verbindungen verläuft, erstellen. Für den nächsten Streckenabschnitt verwenden wir den Reise Zeiger an dem Abfahrtsstopp des ersten Streckenabschnitts. Dies wiederholen wir so lange bis wir den Startstopp s erreicht haben. Die umgekehrte Reihenfolge der erstellten Streckenabschnitte stellt die Reise dar.

3.1.2 Profile CSA

In der Profile CSA Variante wird der CSA so erweitert, dass das EATP Problem gelöst werden kann. Gesucht sind somit mehrere Reisen, die innerhalb eines definierten Intervalls am Zielstopp ankommen. Als Eingabe erhält der Algorithmus den Startstopp s, Zielstopp t, eine minimale Abfahrtszeit τ_s und eine maximale Ankunftszeit τ_t.

Bei dem Profile CSA werden die Verbindungen in ihrer umgekehrten Reihenfolge von der Verbindung mit der spätesten Abfahrtszeit bis hin zu der Verbindung mit der frühesten Abfahrtszeit behandelt. Dabei wird zu Grunde gelegt, dass ein Passagier, der sich in einer Verbindung mit zugehörigem Trip befindet, drei Möglichkeiten besitzt, wie er seine Reise fortsetzen kann. Entweder führt die Verbindung zu dem Zielstopp und er steigt dort aus. Alternativ bleibt er in dem Trip sitzen oder der Passagier steigt an dem Ankunftsstopp der Verbindung in einen anderen Trip um.

Der Algorithmus benötigt zwei Arrays. In dem S-Array wird für jeden Stopp ein Tupel bestehend aus der Abfahrtszeit, Ankunftszeit und der Austrittsverbindung des Trips, den der Passagier ausgehend von diesem Stopp nimmt, gespeichert. Die Ankunftszeit stellt dabei die Zeit dar, zu der ein Passagier am Zielstopp ankommt, wenn er zu der zugehörigen Abfahrtszeit an diesem Stopp abfährt. Das Array S besitzt für jeden Stopp ein Array aus Tupeln. Sie sind jeweils nach ihrer Abfahrtszeit aufsteigend sortiert. Da die einzelnen Tupel des Arrays jeweils Pareto-Optimal hinsichtlich ihrer Abfahrts- und Ankunftszeiten sind, ist zudem garantiert, dass die Ankunftszeit eines Tupels von keinem Tupel mit größerer Abfahrtszeit dominiert wird. Die Pareto-Optimalität der Tupel muss während dem Algorithmus bei ihrem Einfügen beachtet werden. Initial wird für jeden Stopp ein Tupel mit ∞ als Abfahrts- und Ankunktszeit gespeichert. In dem T-Array speichern wir für jeden Trip die Ankunftszeit am Zielstopp t und die Austrittsverbindung des Trips. Zu Beginn des Algorithmus wird auch hier als Ankunftszeit ∞ für jeden Trip gespeichert. Zudem suchen wir mit einer binären Suche die Verbindung mit der spätesten Abfahrtszeit vor der maximalen Ankunftszeit τ_t. Diese stellt die erste Verbindung dar, die durch den Algorithmus betrachtet wird. Mit einer weiteren binären Suche erhalten wir die Verbindung mit der frühesten Abfahrtszeit, die mindestens so groß ist wie τ_s. Abschließend müssen wir in der Initialisierungsphase noch eine One-To-All-Variante des Standard-CSAs durchführen, um für jeden Stopp die frühest mögliche Ankunftszeit ausgehend von s und τ_s zu bestimmen.

Der Profile CSA führt für alle Verbindungen c, die zwischen der maximalen Ankunftszeit τ_t und der frühesten Abfahrtszeit τ_s abfahren, zwei Schritte durch. Zuerst wird die Ankunftszeit τ_c am Zielstopp berechnet und sie anschließend in T und S integriert. Dies wird lediglich für Verbindungen durchgeführt, die ausgehend von s und τ_s erreichbar sind.

Zur Berechnung von τ_c müssen die drei Möglichkeiten beachtetet werden, die ein Passagier hat, wenn er sich in dem Trip c_{trip} der Verbindung c befindet. Wir bestimmen alle drei zugehörigen Ankunftszeiten τ_1, τ_2 und τ_3 und verwenden deren Minimum als τ_c.

Falls die Verbindung c zu dem Zielstopp führt und somit $c_{\text{arr\_stop}} = t$ gilt, setzen wir τ_1 auf $c_{\text{arr\_time}}$. Andernfalls kann der Zielstopp nicht direkt von der Verbindung aus erreicht werden und wir erhalten $\tau_1 = \infty$. Der zweite Fall stellt die Ankunftszeit dar, falls der Passagier in dem Trip von c sitzen bleibt. Da die Verbindungen von hinten nach vorne behandelt werden, ist diese schon bekannt und als Paar zusammen mit der zugehörigen Austrittsverbindung in dem T-Array gespeichert. Es gilt somit $\tau_2 = T[c_{\text{trip}}]_{\text{arr\_time}}$. Um für τ_3 den Trip zu finden, in den der Passagier umsteigt, iterieren wir

über das S-Array von $c_{\text{arr\_stop}}$. Wir wählen dabei das erste Tupel mit einer Abfahrtszeit größer oder gleich der Ankunftszeit $c_{\text{arr\_time}}$ aus. Als τ_3 setzen wir dabei die Ankunftszeit des ausgewählten Tupels.

Von diesen drei möglichen Ankunftszeiten wählen wir die minimale als τ_c aus und aktualisieren die Arrays T und S, wenn $\tau_c \leq \tau_t$ gilt. Falls τ_c kleiner als die Ankunftszeit in $T[c_{\text{trip}}]$ ist, ersetzen wir den bisherigen Eintrag in $T[c_{\text{trip}}]$ durch das Paar (τ_c, c). Um das neue Tupel $p = (c_{\text{dep\_time}}, \tau_c, T[c_{\text{trip}}]_{\text{exit}})$ mit $T[c_{\text{trip}}]_{\text{exit}}$ als Austrittsverbindung des Trips c_{trip} in das Array S einzufügen, benötigen wir den ersten Eintrag q von $S[c_{\text{dep\_stop}}]$. Falls p nicht von q hinsichtlich ihrer Abfahrts- und erwarteter Ankunftszeit dominiert wird, fügen wir p in $S[c_{\text{dep\_stop}}]$ ein. Da die Tupels nach ihren Abfahrtszeiten sortiert sind, müssen wir dazu eine Fallunterscheidung durchführen. Falls die Abfahrtszeit von q größer als die von p ist, können wir p als neues erstes Element von $S[c_{\text{dep\_stop}}]$ einfügen. Andernfalls gilt $p_{\text{dep\_time}} = q_{\text{dep\_time}}$, da wir die Verbindungen in der Reihenfolge ihrer Abfahrtszeiten abarbeiten. In diesem Fall müssen wir q durch p in $S[c_{\text{dep\_stop}}]$ ersetzen. Durch dieses Vorgehen beim Einfügen von Tupeln bleibt die Pareto-Optimalität der Einträge des Arrays erhalten.

Sobald wir alle Verbindungen in dem Intervall zwischen τ_s und τ_t betrachtet haben, stellen die Einträge in $S[s]$ die Ankunftszeiten der Reisen in diesem Intervall dar. Aus den Austrittsverbindungen der Tupel in dem S-Array können die Streckenabschnitte der Reisen ausgehend von dem Startstopp bis zu dem Zielstopp erstellt werden.

3.1.3 CSA MEAT

Der Profile CSA kann so angepasst werden, dass mit ihm die minimalen erwarteten Ankunftszeiten berechnet werden können. Aus dem daraus resultierenden S-Array wird der zugehörige Entscheidungsgraph extrahiert.

Als Eingabe erhalten wir bei dem MEAT Problem lediglich den Startstopp s, den Zielstopp t und die minimale Abfahrtszeit τ_s. Da der Profile CSA zusätzlich die maximale Ankunftszeit τ_t benötigt, müssen wir diese zunächst berechnen. Wir verwenden dabei die α-beschränkte Variante des MEAT Problems. Es gilt somit $\tau_t = \tau_s + \alpha \cdot (\text{ESAT}(s, t, \tau_s) - \tau_s)$. Um die früheste sichere Ankunftszeit $\text{ESAT}(s, t, \tau_s)$ ausgehend von s und τ_s zu erhalten, verwenden wir eine leicht modifizierte Standardvariante des CSA.

Da wir die minimale erwartete Ankunftszeit statt den frühesten Ankunftszeiten berechnen wollen, ersetzen wir die Ankunftszeitfelder in den Tupeln des S- und T-Arrays durch erwartete Ankunftszeiten. Dementsprechend müssen wir in τ_1, τ_2, τ_3 und daher auch in τ_c erwartete Ankunftszeiten berechnen.

Für τ_1 erfolgt die Anpassung dabei so, dass wir die Ankunftszeit der Verbindung mit der erwarteten Verspätung des Trips addieren, falls die Verbindung zu dem Zielstopp führt. Somit gilt nun $\tau_1 = c_{\text{arr\_time}} + E[D(c_{\text{trip}}, c_{\text{dep\_stop}})]$. Da τ_2 lediglich die erwartete Ankunftszeit des zugehörigen Trips übernimmt, ändert sich die Berechnung für die erwarteten Ankunftszeiten nicht. Die größte Anpassung muss bei der Berechnung von τ_3 gemacht werden. Da der ankommende Trip c_{trip} möglicherweise verspätet ist, muss die neue erwartete Ankunftszeit aus den erwarteten Ankunftszeiten der Trips, in die der Passagier potenziell umsteigt, berechnet

werden. Dabei werden diese mit der Wahrscheinlichkeit, dass der Passagier in sie umsteigt, gewichtet und anschließend addiert. Die benötigten erwarteten Ankunftszeiten sind in $S[c_{\text{arr\_stop}}]$ gespeichert und nach ihren Abfahrtszeiten aufsteigend sortiert. Seien $p^1, ..., p^k$ die zugehörigen Tupel aus dem S-Array mit einer Abfahrtszeit von mindestens $c_{\text{arr\_time}}$, $d_1, ..., d_k$ ihre Abfahrtszeiten und $d_0 = c_{\text{arr\_time}}$. Die erwartete Ankunftszeit von dem neuen Tupel p kann nun über $p_{\text{exp\_arr\_time}} = \sum_{i \in \{1,...,n\}} P[d_{i-1} < c_{\text{arr\_time}} + D(c_{\text{trip}}, c_{\text{arr\_stop}}) \leq d_i] \cdot p^i_{\text{exp\_arr\_time}}$ berechnet werden. Diese erwartete Ankunftszeit stellt den neuen Wert von τ_3 dar.

Als τ_c wird wieder der minimale Wert von τ_1, τ_2 und τ_3 gewählt. Die Aktualisierung des T-Arrays erfolgt analog zu dem ursprünglichen Algorithmus. Bei dem Einfügen des neuen Tupels p in das S-Array überprüfen wir die Dominanzregeln gemäß der erwarteten Ankunfts- und Abfahrtszeiten der Tupel. Anschließend verwenden wir die gleiche Fallunterscheidung wie bei dem Profile CSA.

Nachdem wir diesen modifizierten Profile CSA durchgeführt haben, steht die minimale erwartete Ankunftszeit in dem ersten Tupel von $S[s]$. Die Zeiger der Tupel des S-Arrays können wir nun verwenden, um den zugehörigen (s, τ_s, t)-Entscheidungsgraphen $G = (V, E)$ zu extrahieren. Dazu erstellen wir mit Hilfe der Verbindungen, die die Ausstiegspunkte der Trips definieren, die Streckenabschnitte des Graphen. Aus diesen kann die Knotenmenge V abgeleitet werden.

Zunächst benötigen wir eine Prioritätswarteschlange, die Tupel des S-Arrays aufsteigend nach ihren Abfahrtszeiten sortiert enthält. Solange die Warteschlange nicht leer ist, entnehmen wir das erste Element und erstellen einen Streckenabschnitt aus der Abfahrtszeit des zugehörigen Trips an diesem Stopp und der Austrittsverbindung des Tupels. Anschließend müssen wir die Einträge des S-Arrays in die Warteschlange einfügen, in deren zugehörigen Trips der Passagier potenziell nach dem Verlassen des neu erstellten Streckenabschnitts umsteigt. Falls dieser Streckenabschnitt zu dem Zielstopp führt, müssen wir keine neuen Einträge in die Warteschlange hinzufügen. Seien $l_{\text{arr\_time}}$ die Ankunftszeit, $l_{\text{arr\_stop}}$ der Ankunftsstopp und l_{trip} der Trip des neu erstellten Streckenabschnitts. Falls $l_{\text{arr\_stop}} \neq t$, fügen wir alle Tupel des Arrays $S[l_{\text{arr\_stop}}]$ in die Warteschlange ein, deren Abfahrtszeit zwischen $l_{\text{arr\_time}}$ und $l_{\text{arr\_time}} + \max D(l_{\text{trip}}, l_{\text{arr\_stop}})$ liegt. Zusätzlich fügen wir das erste Element ein, dessen Trip nach $l_{\text{arr\_time}} + \max D(l_{\text{trip}}, l_{\text{arr\_stop}})$ abfährt, da dieser den ersten sicheren Trip für den Passagier darstellt. Sobald die Prioritätswarteschlange leer ist, haben wir alle Streckenabschnitte erhalten, die die Kanten des Entscheidungsgraphen darstellen. Aus deren zugehörigen Abfahrts- und Ankunftsstopps können wir die Knoten des Graphen ableiten. Um die kompakte Version des Graphen zu erstellen, müssen wir jeweils über alle ausgehenden Kanten eines Knotens iterieren und die Kanten zusammenfassen, die zu dem gleichen Stopp führen.

3.2 RAPTOR Algorithmen

Die nachfolgenden RAPTOR Algorithmen wurden von Delling et al. (2015) in [DPW15] vorgestellt. Im Gegensatz zu den CSAs benötigen die RAPTOR Algorithmen zusätzlich die Information zu welcher Route Trips gehören. Dafür verwenden die Algorithmen nicht die Verbindungen. Die RAPTOR Algorithmen berechnen die frühesten Ankunftszeiten jeweils rundenbasiert. So erhält man nach k Runden die frühesten Ankunftszeiten, deren Reisen maximal k Streckenabschnitte und somit $k - 1$ Umstiege besitzen. Wie bei den CSAs betrachten wir zunächst die Standardvariante des RAPTOR Algorithmus mit der wir das EAT Problem lösen können und anschließend die McRAPTOR Variante zur Lösung des EATP Problems. Für die RAPTOR Algorithmen existiert noch keine Lösung des Problems der minimalen erwarteten Ankunftszeiten.

3.2.1 RAPTOR

Der Standard RAPTOR Algorithmus zur Lösung des EAT Problems berechnet die frühesten Ankunftszeiten für jeden Stopp ausgehend von s und τ_s rundenbasiert.

Dazu speichern wir für jeden Stopp p die Labels $(\tau_0(p), \tau_1(p), ...)$, die für jede Runde die minimale Ankunftszeit am Stopp p enthalten. Dabei besitzen die Reisen, die den frühesten Ankunftszeiten in Runde k zugeordnet werden können, maximal k Streckenabschnitte und somit $k-1$ Umstiege. Um den Algorithmus effizienter zu gestalten, verwenden wir die lokale Vereinfachungsmethode (local pruning). Dazu speichern wir für jeden Stopp p die minimale bisher bekannte Ankunftszeit in $\tau^*(p)$. Da unser Ziel lediglich darin besteht, die früheste Ankunftszeit zu ermitteln, müssen wir in den Labels $\tau_i(p)$ mit $i \geq 1$ keine Zeiten abspeichern, die größer oder gleich groß wie die bisher bekannte minimale Ankunftszeit $\tau^*(p)$ sind. Initial speichern wir für alle Stopps und Runden sowohl in τ_i als auch $\tau^* \infty$ ab. Lediglich für den Startstopp s gilt $\tau_0(s) \leftarrow \tau_s$ und $\tau^*(s) \leftarrow \tau_s$. Wir verwenden in jeder Runde ein Array, in dem wir die markierten Stopps der aktuellen Runde speichern. In dieses fügen wir vor der ersten Runde den Startstopp s ein. Um die Reise nach der Berechnung der minimalen Ankunftszeit extrahieren zu können, verwenden wir ein Reise Zeiger Array J. In diesem ist für jeden Stopp ein Paar bestehend aus dem Trip mit dem man zu dem Stopp gelangt und der Einstiegspunkt dieses Trips gespeichert.

In jeder Runde k mit $k \geq 1$ führen wir zwei Schritte durch. Dabei ermitteln wir zunächst alle Routen, die mindestens einen markierten Stopp der letzten Runde enthalten. Über diese Routen iterieren wir im zweiten Teil und aktualisieren die Ankunftszeiten ihrer Stopps. In jeder Runde benötigen wir das Array Q, das wir zu Beginn der Runde leeren. In diesem sind alle Routen-Stopp-Paare gespeichert, die wir in der aktuellen Runde k betrachten. Für jede Route r darf sich nur ein Eintrag in Q befinden.

Um im ersten Schritt der Runde k alle benötigten Routen zu ermitteln und mit ihnen Q zu füllen, iterieren wir über alle markierten Stopps p der vorherigen Runde $k-1$. Für jeden Stopp p bestimmen wir alle Routen r, in denen p enthalten ist. Falls bereits ein Eintrag (r, p') für die Route r in Q gespeichert ist, ersetzen wir diesen durch das Routen-Stopp-Paar (r, p), wenn sich p vor p' innerhalb der Route r befindet. Falls für r noch kein Paar in Q gespeichert ist, kann (r, p) direkt eingefügt werden. Abschließend entfernen wir die Markierung von p.

Im zweiten Teil des Algorithmus iterieren wir über alle Routen-Stopp-Paare (r, p) in Q. Dabei bestimmen wir zunächst den ersten Trip t' von r, den wir an Stopp p erreichen können. Anschließend betrachten wir alle Stopps p_i, die in der Route r auf p folgen. Für diese überprüfen wir, ob die Ankunftszeit $\tau_{\text{arr}}(t', p_i)$ des Trips t' an p_i kleiner als die bisher bekannte minimale Ankunftszeit $\tau^*(p_i)$ und die früheste Ankunftszeit $\tau^*(t)$ des Zielstopps t ist. Diesen Vergleich der Ankunftszeit mit der von t, der als globale Vereinfachung (target pruning) bezeichnet wird, führen wir durch, da wir als Ergebnis unseres Algorithmus lediglich die minimale Ankunftszeit des Zielstopps berechnen wollen. Falls die Ankunftszeit $\tau_{\text{arr}}(t', p_i)$ beide Bedingungen erfüllt, verwenden wir sie als neuen Wert von $\tau_k(p_i)$ und $\tau^*(p_i)$. In diesem Fall markieren wir p_i. Zusätzlich speichern wir dann in $J[p_i]$ den Trip t' und den Stopp an dem wir t' als frühesten Trip der Route r ausgewählt haben. Anschließend wird getestet, ob an p_i ein früherer Trip der Route r als der bisherige Trip t' erreicht werden kann. Dies ist möglich, falls die früheste Ankunftszeit $\tau_{k-1}(p_i)$ der letzten Runde kleiner als die Abfahrtszeit $\tau_{\text{dep}}(t', p_i)$ des aktuellen Trips ist. Ist dies erfüllt, ermitteln wir wie bei dem ersten betrachteten Stopp der Route den frühesten Trip, den wir an p_i erreichen können und verwenden ihn anstatt von t' für die folgenden Stopps.

Am Ende jeder Runde überprüfen wir, ob in der aktuellen Runde neue Stopps markiert wurden. Ist dies nicht der Fall, terminiert der Algorithmus. Nach der Terminierung des Algorithmus steht die minimale Ankunftszeit in $\tau^*(t)$. Die zugehörige Reise kann ausgehend von dem Zielstopp t erstellt werden. Dazu wird der erste Streckenabschnitt zwischen dem in $J[t]$ gespeicherten Einstiegsstopp des zugehörigen Trips und t erstellt. Ausgehend von diesem Einstiegsstopp wird daraufhin der nächste Streckenabschnitt der Reise extrahiert. Dies wird solange wiederholt, bis wir den Startstopp s erreichen. Die resultierende Reise besteht dann aus den erstellten Streckenabschnitten in ihrer umgekehrten Reihenfolge.

3.2.2 McRAPTOR

Die Standardvariante des RAPTOR Algorithmus können wir so anpassen, dass das Ergebnis nach mehr als einem Kriterium optimiert wird. Der daraus entstehende Algorithmus wird als McRAPTOR bezeichnet. Wählen wir als Kriterien des Algorithmus die Ankunftszeit und die Abfahrtszeit, können wir das EATP Problem lösen. Da wir nun für jeden Stopp mehr als einen Wert speichern müssen, verwenden wir für jeden Stopp p Beutel (Bags), in denen Labels enthalten sind. Diese Labels bestehen aus der Abfahrtszeit am Startstopp s, der Ankunftszeit an dem zugehörigen Stopp p und dem Trip, der den Passagier zu dieser Zeit zu p führt. Um später die Reisen extrahieren zu können, enthält das Label zusätzlich den Einstiegsstopp des Trips. Wir speichern dabei für jeden Stopp p und jede Runde $k \geq 1$ einen Beutel $B_k(p)$ mit Labels. Um analog zu der Standardvariante des Algorithmus lokale und globale Vereinfachungen (local and target pruning) anwenden zu können, verwenden wir für jeden Stopp p einen Beutel $B^*(p)$, der alle nicht dominierten Labels der früheren Runden umfasst. Entscheident für die Dominanzregeln sind die Abfahrts- und Ankunftszeiten.

Im Gegensatz zu unserer Definition des EATP Problems, hat der McRAPTOR Algorithmus als Eingabe den Startstopp s, Zielstopp t, eine minimale Abfahrtszeit $\tau_{\min\_s}$ und eine maximale Abfahrtszeit $\tau_{\max\_s}$. Die in unserer Definition fehlende maximale Abfahrtszeit können wir aus der maximalen Ankunftszeit τ_t berechnen. Mit Hilfe der umgekehrten Version des Standard CSA kann auf Eingabe von s, t und τ_t die späteste Abfahrtszeit berechnet werden, deren schnellste Reise keine Ankunftszeit nach τ_t besitzt. Diese Zeit können wir als $\tau_{\max\_s}$ verwenden, da alle späteren Abfahrtszeiten in Ankunftszeiten nach τ_t resultieren. Zunächst erstellen wir alle Beutel $B^*(p)$ und $B_k(p)$ für die Stopps p und Runden k leer. Initial fügen wir lediglich in $B_0(s)$ und $B^*(s)$ jeden Trip, der in dem Intervall zwischen $\tau_{\min\_s}$ und $\tau_{\max\_s}$ abfährt, ein Label mit der zugehörigen Abfahrtszeit am Startstopp s hinzu und markieren den Stopp s.

In jeder Runde des McRAPTOR Algorithmus führen wir wie bei dem RAPTOR Algorithmus zwei Schritte durch. Dabei verändern wir die Methode des Auswählens der Routen-Stopp-Paare nicht. Im zweiten Schritt iterieren wir über alle ausgewählten Routen-Stopp-Paare. Das Aktualisieren der Beutel erfolgt dabei in drei Schritten. Während wir die Stopps einer Route r betrachten, verwenden wir einen Routenbeutel (Route Bag) B_r in dem wir alle aktuellen Labels speichern. Vor der Bearbeitung einer Route in der aktuellen Runde leeren wir den Routenbeutel. Für jedes Paar (r, p) aus Q starten wir mit dem Stopp p. Die drei Schritte führen wir für alle Stopps p_i der Route r ausgehend von p durch. Im ersten Bestandteil aktualisieren wir die Ankunftszeiten der Labels des Routenbeutels. Dazu verwenden wir die Stoppzeiten der zugehörigen Trips an dem Stopp p_i. Im zweiten Teil führen wir B_r und $B_k(p_i)$ zusammen. Dabei fügen wir lediglich Labels aus B_r hinzu, die nicht durch die Labels in $B^*(p_i)$, $B^*(t)$ oder $B_k(p_i)$ dominiert werden. Nachdem wir alle nicht dominierten Labels in $B_k(p_i)$ und $B^*(p_i)$ eingefügt haben, entfernen wir aus ihnen alle Labels,

die durch die Neuen dominiert werden. Falls mindestens ein neues Label in den Beutel $B^*(p_i)$ hinzugefügt wurde, markieren wir p_i. Im letzten Schritt fügen wir $B_{k-1}(p_i)$ und B_r zusammen und verwenden dies als neuen Routenbeutel. Dabei weisen wir den neuen Labels jeweils den ersten Trip der Route r zu, der nach ihrer Ankunftszeit am Stopp p_i erreicht werden kann. Als Einstiegspunkt des Trips, der in dem Label gespeichert wird, wählen wir p_i.

Am Ende jeder Runde überprüfen wir, ob Stopps markiert wurden. Ist dies nicht der Fall, terminiert der Algorithmus. Nach der Terminierung des Algorithmus befinden sich in $B^*(t)$ alle Labels mit den Ankunftszeiten der Reisen, die ausgehend von dem Startstopp s und dem Intervall an Abfahrtszeiten zu dem Stopp t führen, und gemäß der Abfahrtszeit und Ankunftszeit Pareto-Optimal sind. Wie bei der Standardvariante können die Streckenabschnitte der zugehörigen Reisen über die Einstiegspunkte der Trips, die in den Labels gespeichert sind, extrahiert werden.

4 Neue Algorithmen

Auf der Basis der bereits bestehenden Algorithmen entwickeln wir nun zwei neue. Der erste Algorithmus basiert auf dem Profile CSA und löst das ExpAT Problem, bei dem die erwarteten Ankunftszeiten berechnet werden, falls man Reisen ausschließlich nach ihrer frühesten Ankunftszeit auswählt. Der zweite Algorithmus besitzt, alternativ zu dem CSA MEAT, als Grundlage den McRAPTOR Algorithmus und berechnet ebenfalls Lösungen für das MEAT Problem.

4.1 CSA ExpAT

Um das Problem der ExpATs lösen zu können, passen wir den Standard Profile CSA so an, dass wir zusätzlich zu den frühesten Ankunftszeiten auch die erwarteten Ankunftszeiten berechnen. Aus den dabei erstellten Tupeln extrahieren wir analog zu dem CSA MEAT Entscheidungsgraphen. Als Eingabe erhalten wir einen Startstopp s, einen Zielstopp t und eine minimale Abfahrtszeit τ_s. Der zugehörige Pseudocode befindet sich in Algorithmus 4.1.

Bei dem Profile CSA haben wir für jeden Stopp in dem S-Array ein Tupel bestehend aus der Abfahrtszeit, der Ankunftszeit am Zielstopp und der Austrittsverbindung des zugehörigen Trips gespeichert. In unserer abgewandelten Variante speichern wir zusätzlich die erwartete Ankunftszeit am Zielstopp. Entscheidend für die Überprüfung der Dominanz von Tupeln des S-Arrays sind aber weiterhin die Abfahrts- und Ankunftszeiten und das Array enthält ausschließlich Elemente, die hinsichtlich dieser Parameter Pareto-Optimal sind. Dies wird benötigt, da laut der Definition des Problems die Auswahl der Trips des Passagiers ausschließlich über die frühest mögliche Ankunftszeit erfolgt. Als initialen Wert speichern wir für jede erwartete Ankunftszeit ∞. Auch die Abfahrts- und Ankunftszeit wird in dem ersten Tupel von jedem Stopp auf ∞ gesetzt (Z. 6-8 im Pseudocode). Das T-Array speichert für jeden Trip nun ein Tupel aus Ankunftszeit, erwarteter Ankunftszeit und der Austrittsverbindung des Trips. In der Initialisierungsphase fügen wir für jeden Trip ein Tupel mit ∞ als Wert für die Ankunftszeit und die erwartete Ankunftszeit ein (Z. 9-11). Um einen vergleichbaren Wert zu der minimalen erwarteten Ankunftszeit des MEAT Problems zu erhalten, verwenden wir dieselbe maximale Ankunftszeit $\tau_t = \tau_s + \alpha \cdot (\text{ESAT}(s, t, \tau_s) - \tau_s)$. Zusätzlich benötigen wir für den Algorithmus die frühesten Ankunftszeiten an allen Stopps ausgehend von s und τ_s. Dazu führen wir einen One-To-All CSA durch.

Der CSA ExpAT führt wie die Standard Variante des Profile CSA zwei Schritte für jede Verbindung zwischen der maximalen Ankunftszeit und der minimalen Abfahrtszeit durch. Die Betrachtung der Verbindungen erfolgt dabei auch in der umgekehrten Reihenfolge ihrer Abfahrtszeiten. Wir behandeln lediglich Verbindungen, die ausgehend von s und τ_s erreichbar sind und deren Ankunftszeit vor der maximalen Ankunftszeit liegt (Z. 17-19). Im ersten Schritt jeder Verbindung c wird die früheste Ankunftszeit τ_c am Zielstopp berechnet. Als Erweiterung bestimmen wir nun dabei zusätzlich die zugehörige erwartete Ankunftszeit $\tau_{\text{c\_exp}}$. Mit diesen Werten und den daraus entstehenden Tupeln können wir die Arrays S und T aktualisieren.

Für die Berechnung von τ_c und $\tau_{\text{c\_exp}}$ der aktuell betrachteten Verbindung c bestimmen wir zunächst analog zu dem Profile Algorithmus die frühesten Ankunftszeiten τ_1, τ_2 und τ_3 (Z. 21-35). Von diesen wählen wir das Minimum als τ_c aus (Z. 37). Je nachdem welche der drei Varianten gewählt wurde, berechnen wir die zugehörige erwartete Ankunftszeit analog zu dem CSA MEAT. τ_1 setzen wir auf die Ankunftszeit $c_{\text{arr\_time}}$, falls die Verbindung zu dem Zielstopp führt. Deren zugehörige erwartete Ankunftszeit τ_{c_exp} berechnen wir, indem wir die erwartete Verspätung $E[D(c_{\text{trip}}, c_{\text{arr\_stop}})]$ des Trips c_{trip} der Verbindung mit seiner Ankunftszeit $c_{\text{arr\_time}}$ addieren (Z. 39-40). Für τ_2 können wir die Ankunftszeit verwenden, die in dem T-Array für den Trip c_{trip} gespeichert ist. Deren zugeordnete erwartete Ankunftszeit ist ebenfalls in dem T-Array für c_{trip} enthalten (Z. 41-42). Als τ_3 erhalten wir die Ankunftszeit des ersten Trips, in den wir umsteigen können. Aus den Dominanzregeln folgt, dass dies die frühest mögliche Ankunftszeit darstellt, die durch einen Umstieg an diesem Stopp zu der Ankunftszeit der Verbindung erreicht werden kann. Die zugehörige erwartete Ankunftszeit berechnen wir analog zu dem CSA MEAT über die gewichtete Addition der Tupel in $S[c_{\text{arr\_stop}}]$. Dabei sichern die Dominanzregeln der Tupel, dass jeweils die Reisen mit den minimalen Ankunftszeiten als Alternativreisen ausgewählt werden. Zusätzlich kann die erwartete Ankunftszeit nur berechnet werden, falls in $S[c_{\text{arr\_stop}}]$ ein Tupel mit einer Abfahrtszeit nach der maximalen Verspätung des Trips c_{trip} existiert, da es ansonsten keine sichere Alternativreise für das neue Tupel geben würde. Wir können auch dann keine erwartete Ankunftszeit berechnen, falls eine der Alternativreisen ∞ als erwartete Ankunftszeit besitzt. Sollte die erwartete Ankunftszeit nicht berechnet werden können, setzen wir $\tau_{\text{c\_exp}} = \infty$ (Z. 43-56).

Im zweiten Schritt aktualisieren wir zunächst die Ankunftszeiten und erwartete Ankunftszeiten von $T[c_{\text{trip}}]$, falls τ_c kleiner als die bisher für diesen Trip gespeicherte Ankunftszeit ist. Dann setzen wir die Ankunftszeit des Trips auf τ_c, seine erwartete Ankunftszeit auf $\tau_{\text{c\_exp}}$ und die Austrittsverbindung auf c (Z. 58-60). Anschließend fügen wir das neue Tupel p, das aus τ_c, $\tau_{\text{c\_exp}}$, $c_{\text{dep\_time}}$ und der Austrittsverbindung von $T[c_{\text{trip}}]$ besteht, in $S[c_{\text{dep\_stop}}]$ ein, falls es nicht durch deren ersten Eintrag q dominiert wird. Analog zu dem Profile CSA führen wir dabei eine Fallunterscheidung durch und ersetzen entweder q oder fügen p an den Anfang von $S[c_{\text{dep\_stop}}]$ ein (Z. 62-70).

Sobald wir alle Verbindungen betrachtet haben, terminiert der Algorithmus. Anschließend erhalten wir die erwartete Ankunftszeit über den ersten Eintrag in $S[s]$. Mit den Austrittsverbindungen der Tupel in dem S-Array können wir analog zu dem CSA MEAT den zugehörigen Entscheidungsgraphen erstellen. Im Gegensatz zu seiner MEAT Variante ist bei dem CSA ExpAT nicht gesichert, dass ein Entscheidungsgraph und somit eine erwartete Ankunftszeit existiert, falls es eine sichere Reise zu dem Zielstopp gibt. Die Existenz eines Entscheidungsgraphen, wenn eine sichere Reise von s nach t vorliegt, wurde in dem Lemma 2.1 von Dibbelt et al. (2018) in [DPSW18] bewiesen. Im Gegensatz zu dem CSA MEAT können wir es für die CSA ExpAT Variante nicht nutzen, da wir hier auch Tupel mit einer erwarteten Ankunftszeit von ∞ in die Einträge des S-Arrays einfügen. Für deren zugehörige Streckenabschnitte existieren keine sicheren Alternativreisen zu dem Zielstopp. Da dies eine Bedingung der Entscheidungsgraphen ist, ist nicht gewährleistet, dass für jede Anfrage eine Lösung des ExpAT Problems berechnet werden kann.

Algorithmus 4.1 CSA ExpAT

```
1:  procedure CSAExpAT(s: Stop, t: Stop, τ_s: Time)
2:      earliestSafeArrTime ← EarliestSafeArrTime(s, t, τ_s)        // use one-to-one csa
3:      τ_t ← τ_s + α · (earliestSafeArrTime − τ_s)                 // alpha-bounded version
4:      earliestArrTimes ← EarliestArrTimes(s, ·, τ_s)              // use one-to-all csa
5:
6:      for all stops p do
7:          S[p] ← [{arr_time: ∞, dep_time: ∞, exp_arr_time: ∞}]
8:      end for
9:      for all trips t′ do
10:         T[t′] ← [{arr_time: ∞, exp_arr_time: ∞}]
11:     end for
12:
13:     for connections c decreasing by c_dep_time and starting with c_dep_time = τ_t do
14:         if c_dep_time < τ_s then
15:             stop
16:         end if
17:         if c_dep_time < earliestArrTimes[c_dep_stop] ∨ c_arr_time > τ_t then
18:             continue
19:         end if
20:
21:         if c_arr_stop = t then
22:             τ_1 ← c_arr_time
23:         else
24:             τ_1 ← ∞
25:         end if
26:         τ_2 ← T[c_trip]_arr_time
27:         p ←⊥
28:                                                  // labels in S are sorted by departure time
29:         for all labels l ∈ S[c_arr_stop] do
30:             p ← l
31:             if p_dep_time ≥ c_arr_time then
32:                 break
33:             end if
34:         end for
35:         τ_3 ← p_arr_time
36:                                                  // get minimum arrival time
37:         τ_c ← min{τ_1, τ_2, τ_3}
38:                                                  // get related expected arrival time
39:         if τ_c = τ_1 then
40:             τ_c_exp ← c_arr_time + E[D(c_trip, c_arr_stop)]
41:         else if τ_c = τ_2 then
42:             τ_c_exp ← T[c_trip]_exp_arr_time
43:         else if τ_c = τ_3 then
44:             τ_c_exp ← 0
45:                                                  // labels in S are sorted by departure time
```

```
46:             for all labels l ∈ S[c_arr_stop] do
47:                 if l_dep_time < c_arr_time then
48:                     continue
49:                 else
50:                     τ_c_exp += (l_exp_arr_time · P[take l])
51:                     if l_dep_time ≥ c_arr_time + max D(c_trip, c_arr_stop) then
52:                         break
53:                     end if
54:                 end if
55:             end for
56:         end if
57:                                                          // update T Array
58:         if τ_c < T[c_trip]_arr_time then
59:             T[c_trip] ← {arr_time: τ_c, exp_arr_time: τ_c_exp}
60:         end if
61:                                                          // update S Array
62:         p ← {arr_time: τ_c, dep_time: c_dep_time, exp_arr_time: τ_c_exp}
63:         q ← earliest pair of S[c_dep_stop]
64:         if q does not dominate p then
65:             if q_dep_time = p_dep_time then
66:                 Replace q as the earliest pair of S[c_dep_stop] with p
67:             else
68:                 Insert p at the front of S[c_dep_stop]
69:             end if
70:         end if
71:     end for
72: end procedure
```

4.2 RAPTOR MEAT

Nachdem das MEAT Problem bereits mit dem Ansatz des Profile CSA gelöst wurde, entwickeln wir nun einen Algorithmus bei dem dies auch über die rundenbasierte Herangehensweise des McRAPTOR Algorithmus möglich ist. Anschließend beweisen wir dessen Korrektheit, analysieren die Komplexität des neuen Algorithmus und zeigen welche Vorteile er zusätzlich bietet.

4.2.1 Beschreibung

Der RAPTOR MEAT Algorithmus basiert auf dem McRAPTOR Algorithmus. Bei diesem werden die frühesten Ankunftszeiten für ein Intervall an Abfahrtszeiten am Startstopp berechnet. Dieses Konzept verändern wir bei dem MEAT Algorithmus in der Hinsicht, dass wir ausgehend vom Zielstopp t die erwarteten Ankunftszeiten bis hin zum Startstopp s berechnen. Dafür benötigen wir ein Intervall an Ankunftszeiten als Eingabe. Wie bei dem CSA MEAT können wir das über den leicht modifizierten Standard CSA berechnen. Durch diesen erhalten wir die früheste sichere Ankunftszeit $\mathrm{ESAT}(s, t, \tau_s)$ mit der Eingabe des Startstopps s, der minimalen Abfahrtszeit τ_s und dem Zielstopp t. Wir verwenden analog zu dem CSA MEAT die α-beschränkte Variante des

MEAT Problems. Als obere Grenze des Intervalls ergibt sich somit die maximale Ankunftszeit τ_t: $\tau_t = \tau_s + \alpha \cdot (\text{ESAT}(s, t, \tau_s) - \tau_s)$. Zudem benötigen wir für unseren Algorithmus die minimalen Ankunftszeiten an jedem Stopp ausgehend von dem Startstopp s und der minimalen Abfahrtszeit τ_s. Dazu führen wir den One-To-All CSA zur Lösung des $\text{EAT}(s, \cdot, \tau_s)$ Problems durch. Aus dessen Ergebnis erhalten wir auch die früheste Ankunftszeit an dem Zielstopp t. Diese ist die untere Grenze des Intervalls für unsere Variante des McRAPTOR Algorithmus. Der zu dem RAPTOR MEAT zugehörige Pseudocode befindet sich in Algorithmus 4.2.

Im Gegensatz zu der Standardvariante des McRAPTOR Algorithmus optimiert unser Algorithmus die minimalen erwarteten Ankunftszeiten. Die Labels speichern somit Tupel bestehend aus der Abfahrtszeit und der erwarteten Ankunftszeit am Zielstopp t. Ein Label L_1 dominiert das Label L_2, falls $\tau_{\text{dep\_time}}(L_1) \geq \tau_{\text{dep\_time}}(L_2)$ und $\tau_{\text{exp\_arr\_time}}(L_1) \leq \tau_{\text{exp\_arr\_time}}(L_2)$ gilt. Um aus den Labels Entscheidungsgraphen extrahieren zu können, speichern wir zusätzlich für jedes Label die Runde, in der es erzeugt wurde, den zugehörigen Trip und den Ausstiegspunkt des Trips. Dieser entspricht dem Stopp, an dem der Passagier aus dem Verkehrsmittel des Trips aussteigen muss. Labels desselben Stopps werden wie bei dem McRAPTOR Algorithmus in Beuteln abgespeichert. Sie sind so aufgebaut, dass sie ausschließlich Pareto-Optimale Labels hinsichtlich der Abfahrts- und erwarteten Ankunftszeit enthalten und ihre Einträge zudem nach ihren Abfahrtszeiten sortiert sind. Der RAPTOR MEAT Algorithmus besitzt das Array expectedArrivalTimes, das für jeden Stopp p einen Beutel $B^*(p)$ mit allen nicht dominierten Labels der bisher ausgeführten Runden enthält. Initialisiert wird es für jeden Stopp mit dem Label $(\infty, \infty, 0, \perp, \perp)$ (Z. 5-7 im Pseudocode). Unser RAPTOR MEAT Algorithmus arbeitet wie der McRAPTOR Algorithmus rundenbasiert. In jeder Runde benötigen wir vier Arrays, die danach wieder zurück gesetzt werden können. Zunächst müssen wir alle markierten Stopps speichern. Initial wird hier lediglich der Zielstopp t hinzugefügt (Z.9). Im zweiten Array latestDepartureTimesOfLastRound speichern wir für jeden Stopp die späteste Abfahrtszeit der Labels, die in der vorherigen Runde für diesen Stopp hinzugefügt wurden. Vor dem Start der ersten Runde fügen wir nur für den Zielstopp t die maximale Ankunftszeit τ_t hinzu (Z. 8). In dem Array Q, das auch in den anderen Versionen der RAPTOR Algorithmen existert, speichern wir alle Routen-Stopp-Paare, die wir in der aktuellen Runde betrachten. Vor der ersten Runde wird es somit leer erzeugt. Das letzte Array expectedArrivalTimesOfCurrentRound speichert für jeden Stopp p einen Beutel $B_c(p)$ mit den Labels, die in der aktuellen Runde hinzugefügt werden und ist somit initial auch leer. An dieser Stelle verändern wir die Herangehensweise des Standard McRAPTOR Algorithmus in der Hinsicht, dass wir nicht für jede Runde alle Beutel mit den neu erstellten Labels speichern. Diese benötigen wir im weiteren Verlauf des Algorithmus nicht und wir vermeiden somit die Nutzung des zugehörigen Speicherplatzes. Den Rundenzähler k, der angibt in welcher Runde der Algorithmus sich befindet, initialisieren wir mit 0.

Nach der Initialisierung führen wir Folgendes in jeder Runde des Algorithmus durch:

Zu Beginn jeder Runde inkrementieren wir den Rundenzähler k und initialisieren die Beutel B_c und das Array Q jeweils mit einem leeren Array (Z. 11-12). Der nächste Schritt entspricht dem Bestimmen von allen Routen, die in der Runde betrachtet werden müssen. Das Vorgehen ist dabei dasselbe wie bei der Standardversion des RAPTOR Algorithmus. Wir passen es lediglich so an, dass wir das Routen-Stopp-Paar (r, p) für den Stopp p speichern, der sich innerhalb der Route r am nächsten zu ihrem Ende befindet, falls mehr als ein Stopp der Route in der vorherigen Runde markiert wurde. Dies ist notwendig, da wir durch die veränderte Richtung des Algorithmus auch

über die Stopps einer Route in ihrer umgekehrten Richtung iterieren müssen. Dabei speichern wir die Routen-Stopp-Paare in dem Array Q und entfernen die Markierungen der Stopps der letzten Runde (Z. 13-22).

Die darauffolgende Schleife führen wir für jeden Eintrag (r, p) von Q durch. Das entspricht jeder Route, die mindestens einen markierten Stopp der letzten Runde enthält. Zunächst initialisieren wir den Routenbeutel B_r mit einem leeren Array (Z. 25). Der Routenbeutel speichert alle nicht dominierten Labels einer Route in der aktuellen Runde. Während der Behandlung einer Route r führen wir für jeden Stopp p' ab dem Stopp p drei Schritte durch. Dabei werden die Stopps innerhalb der Route von hinten nach vorne abgearbeitet. Zunächst wird der Routenbeutel mit den Abfahrtszeiten am aktuellen Stopp aktualisiert (Update-Operation), dann wird er mit dem Beutel der erwarteten Ankunftszeiten der aktuellen Route zusammengefügt (Merge-Operation) und abschließend werden neue Labels in den Routenbeutel hinzugefügt (Add-Operation).

An Stopp p' innerhalb der Route r erfolgt das Anpassen der Labels im ersten Schritt über die Abfahrtszeiten an diesem Stopp. Jedem Label des Routenbeutels ist ein Trip zugeordnet. Von diesem können wir die Abfahrtszeit an Stopp p' ermitteln und die bisherige Abfahrtszeit des Labels durch sie ersetzen (Z. 28-30). Im nächsten Schritt werden die Labels des Routenbeutels in den expectedArrivalTimesOfCurrentRound Beutel $B_c(p')$ des Stopps p' hinzugefügt (Z.32). Ist dieser noch leer, können wir die Elemente des Routenbeutels direkt in ihrer vorsortieren Reihenfolge einfügen. Andernfalls verwenden wir für den Vorgang des Zusammenfügens, dass sowohl der B_r als auch der $B_c(p')$ Beutel nach den Abfahrtszeiten ihrer Labels sortiert sind. Daher reicht es aus, in einer Iteration über beide Beutel von der spätesten zur frühesten Abfahrtszeit zu gehen und alle nicht dominierten Labels in einen neuen Beutel einzufügen. Dieser wird nach der Merge-Operation als neuer Beutel $B_c(p')$ für den Stopp p' verwendet. Im dritten Schritt werden neue Labels in den Routenbeutel hinzugefügt, falls es möglich ist an diesem Stopp umzusteigen. Dazu wählen wir alle Trips der Route r aus, deren Ankunftszeit zwischen der frühest möglichen Ankunftszeit und der spätesten Abfahrtszeit der letzten Runde an diesem Stopp liegt (Z. 38). Die frühest mögliche Ankunftszeit erhalten wir über das Ergebnis des One-To-All CSA vom Startstopp s ausgehend, den wir während der Initialisierung durchgeführt haben. Die späteste Abfahrtszeit befindet sich in dem latestDepartureTimesOfLastRound Array, das uns diesen Wert für jeden Stopp speichert. Ist dieser Wert nicht definiert, wurde der Stopp in der letzten Runde nicht markiert. Dann kann der Schritt übersprungen werden, da die dadurch erzeugten Labels keine Veränderungen der letzten Runde beinhalten würden (Z. 34). Sie würden uns somit im Vergleich zu den Labels, die in den vorherigen Runden bereits erstellt wurden, keine neuen Informationen liefern. Falls das Intervall zwischen frühester Ankunftszeit und spätester Abfahrtszeit definiert ist und wir alle Trips erhalten haben, die innerhalb dessen an Stopp p' ankommen, erzeugen wir für jeden Trip ein neues Label. Bei der Berechnung der erwarteten Ankunftszeiten der neuen Labels führen wir eine Fallunterscheidung durch. Falls p' der Zielstopp t ist, wird die erwartete Ankunftszeit über die Summe aus der Ankunftszeit des Trips an Stopp p' und seiner erwarteten Verspätung berechnet (Z. 41-42). Im anderen Fall berechnet sich die erwartete Ankunftszeit über die gewichtete Addition der erwarteten Ankunftszeiten der Streckenabschnitte, in die der Umstieg an dem Stopp p' erfolgen kann. Die Gewichtung entspricht dabei jeweils der Wahrscheinlichkeit mit der der Trip, der dem Streckenabschnitt zugeordnet ist, von dem Nutzer genommen wird und summiert sich an jedem Umstieg zu eins auf. Wir verwenden für diese Berechnung die Labels, die in dem Beutel $B^*(p')$ für p' in dem expectedArrivalTimes Array gespeichert sind. Dieser umfasst alle nicht dominierten Labels der vorangegangenen Runden. Bei der Rechnung berücksichtigen wir jeweils alle Labels deren Abfahrtszeiten zwischen der Ankunftszeit des Trips und der maximalen Verspätung des Trips

liegen. Zudem wird das erste Label miteinbezogen, dessen zugehöriger Trip nach der maximalen Verspätung abfährt (Z. 43-54). Falls kein solches Label existiert, erstellen wir kein neues Label, da für den zugehörigen Streckenabschnitt eine Bedingung der Entscheidungsgraphen nicht erfüllt wäre. So muss jeder Streckenabschnitt eines Graphen eine sichere Reise zu dem Zielstopp besitzen, die vor der maximalen Ankunftszeit ankommt und nach der maximalen Verspätung des zugehörigen Trips an dem Ankunftsstopp des Streckenabschnitts abfährt. Falls die erwartete Ankunftszeit berechnet werden kann, setzen wir die Abfahrtszeit des neuen Labels auf die Abfahrtszeit des Trips an dem Stopp p'. Da die Trips nach ihren Abfahrtszeiten sortiert sind, erhalten wir die neuen Labels auch in der Reihenfolge ihrer Abfahrtszeiten (Z. 56-57). Somit können die neuen Labels analog zu der Merge-Operation in den Routenbeutel B_r eingefügt werden (Z. 59).

Nachdem alle Routen der aktuellen Runde behandelt wurden, werden die Beutel B^* der erwarteten Ankunftszeiten aktualisiert und neue Stopps markiert. Dabei wird zunächst das Array latestDepartureTimesOfLastRound der spätesten Abfahrtszeiten geleert (Z. 64). Da für alle Stopps p' für die in der aktuellen Runde mindestens ein Label gespeichert wurde gilt, dass sowohl der Beutel $B^*(p')$ in expectedArrivalTimes für p' als auch der Beutel $B_c(p')$ in expectedArrivalTimesOfCurrentRound nach den Abfahrtszeiten ihrer Labels sortiert sind, kann der gleiche Vorgang des Zusammenfügens wie in der vorherigen Schleife zum Einfügen der neuen Labels verwendet werden (Z. 69). Sollte es dabei dazu kommen, dass ein Label aus $B^*(p')$ die gleiche Abfahrts- und erwartete Ankunftszeit wie ein Label aus $B_c(p')$ besitzt, wird das Label aus den vorherigen Runden verwendet. Dieses befindet sich in $B^*(p')$. Falls mindestens eines der Labels aus der aktuellen Runde in $B^*(p')$ hinzugefügt wird, wird der Stopp p' markiert. Zusätzlich wird die späteste Abfahrtszeit der neuen Labels, die in den expectedArrivalTimes Beutel eingefügt wurden, am Stopp p' in dem Array der spätesten Abfahrtszeiten gespeichert (Z. 70-74).

Bevor wir die nächste Runde des Algorithmus durchführen, überprüfen wir ob in der aktuellen Runde Stopps markiert wurden. Ist dies nicht der Fall, terminiert der Algorithmus (Z. 76-78).

Sobald der Algorithmus terminiert ist, können die Labels analog zu dem Vorgehen beim CSA MEAT in einen Entscheidungsgraphen umgewandelt werden.

Algorithmus 4.2 RAPTOR MEAT

```
1:  procedure RaptorMeat(s: Stop, t: Stop, τ_s: Time)
2:      earliestSafeArrTime ← EarliestSafeArrTime(s, t, τ_s)          // use one-to-one csa
3:      τ_t ← τ_s + α · (earliestSafeArrTime − τ_s)                   // alpha-bounded version
4:      earliestArrTimes ← EarliestArrTimes(s, ·, τ_s)                // use one-to-all csa
5:      for all stops p ∈ S do
6:          expArrTimes[p] ← [{exp_arr_time: ∞, dep_time: ∞, round: 0}]
7:      end for
8:      latestDepTimeOfLastRound[t] ← τ_t
9:      mark t
10:     for all k ← 1, 2, ... do
11:         clear expArrTimesOfCurrRound
12:         clear Q
13:         for all marked stops p do
14:             for all routes r serving p do
15:                 if (r, p′) ∈ Q for some stop p′ then
16:                     substitute (r, p) ∈ Q if p comes after p′ in r
17:                 else
18:                     add (r, p) to Q
19:                 end if
20:             end for
21:             unmark p
22:         end for
23:                                                                   // traverse routes
24:         for all pairs (r, p) ∈ Q do
25:             initialize B_r
26:             for all stops p_i beginning with p do                 // from p to first stop
27:                                                                   // update labels
28:                 for all labels l ∈ B_r do
29:                     l_dep_time ← GetDepTimeByTripAndStop(l_trip, p_i)
30:                 end for
31:                                                                   // merge labels
32:                 expArrTimesOfCurrRound[p_i] ← Merge(B_r, expArrTimesOfCurrRound[p_i])
33:                                                                   // add labels
34:                 if latestDepTimeOfLastRound[p_i] ≠⊥ then
35:                     newLabels ← []
36:                     m ← earliestArrTimes[p_i]
37:                     n ← latestDepTimeOfLastRound[p_i]
38:                     newTrips ← GetAllTripsOfInterval(p_i, r, m, n)
39:                     for all trips t′ ∈ newTrips do
40:                         expArrTime ← 0
41:                         if p_i = t then
42:                             expArrTime ← τ_arr_time(t′, p_i) + E[D(t′, p_i)]
43:                         else
44:                                                                   // labels are sorted by departure time
45:                             for all labels l ∈ expArrTimes[p_i] do
```

46: **if** $\tau_{\text{arr\_time}}(t', p_i) > l_{\text{dep\_time}}$ **then**
47: **continue**
48: **else**
49: expArrTime += ($l_{\text{exp\_arr\_time}} \cdot P[\text{take } l]$)
50: **if** $l_{\text{dep\_time}} \geq \tau_{\text{arr\_time}}(t', p_i) + \max D(t', p_i)$ **then**
51: **break**
52: **end if**
53: **end if**
54: **end for**
55: **end if**
56: newLabel ← {exp_arr_time: expArrTime, dep_time: $\tau_{\text{dep\_time}}(t', p_i)$, round: k}
57: Insert newLabel into newLabels
58: **end for**
59: $B_r \leftarrow$ MERGE(B_r, newLabels)
60: **end if**
61: **end for**
62: **end for**
63: // update expected arrival times
64: clear latestDepTimesOfLastRound
65: **for all** stops $p \in S$ **do**
66: **if** expArrTimesOfCurrRound[p] contains at least one element **then**
67: bag1 ← expArrTimes[p]
68: bag2 ← expArrTimesOfCurrRound[p]
69: expArrTimes[p] ← MERGE(bag1, bag2)
70: **if** at least one element of expArrTimesOfCurrRound[p] was merged **then**
71: mark p
72: set latestDepTimesOfLastRound[p]
73: **end if**
74: **end if**
75: **end for**
76: **if** no stops are marked **then**
77: **stop**
78: **end if**
79: **end for**
80: **end procedure**

4.2.2 Korrektheitsbeweis

Als Voraussetzung für den Korrektheitsbeweis benötigen wir das Lemma 6.3 aus dem CSA MEAT in [DPSW18]. Dieses besagt, dass für jeden Startstopp s und jede Startzeit τ_s ein optimaler Entscheidungsgraph existiert, falls es einen Entscheidungsgraphen gibt. Die Optimalität ist so definiert, dass für jeden Streckenabschnitt l von G die Eintrittsverbindung l^{enter} am Stopp $l^{\text{enter}}_{\text{dep\_stop}}$ nicht dominiert wird. Auf die Definitionen des RAPTOR MEAT Algorithmus kann das so übertragen werden, dass für jeden Streckenabschnitt l von G das zugehörige Label L am Stopp $l_{\text{dep\_stop}}$ nicht dominiert wird. Diese Zuweisung ist möglich, da jeder Streckenabschnitt durch genau ein Label definiert wird.

Zudem ist es notwendig, dass auch Labels in Beutel eingefügt werden, obwohl es in ihnen bereits Labels mit einer kleineren erwarteten Ankunftszeit aber früheren Abfahrtszeit gibt, da nicht gewährleistet ist, dass die Streckenabschnitte der früheren Labels durch Passagiere erreicht werden können. Um somit optimale Alternativreisen berechnen zu können, werden alle nicht dominierten Labels benötigt.

Lemma 4.2.1
Für den RAPTOR MEAT Algorithmus gilt die folgende Invariante für jede Runde mit $k \geq 1$: Der Algorithmus berechnet für jeden Stopp p die minimalen erwarteten Ankunftszeiten am Zielstopp t, deren zugehörige Reisen maximal $k-1$ Umstiege besitzen. Um die größte Anzahl an Umstiegen zu bestimmen, die ein Passagier maximal vornehmen muss, falls er sich für ein Label mit zugehöriger erwarteter Ankunftszeit entscheidet, ordnen wir jedem Label Reisen zu. Dabei werden einem Label alle Reisen zugewiesen, über die der Nutzer möglicherweise zum Zielstopp fährt, falls er die Reise ausgehend von dem Label antritt. Somit sind alle möglichen Alternativreisen der ursprünglichen Reise mit inbegriffen. Die maximale Anzahl an Umstiegen dieser Reisen entspricht der maximalen Anzahl an Umstiegen der erwarteten Ankunftszeit des Labels. Die Invariante gilt für jede mögliche Abfahrtszeit in dem Intervall zwischen der minimalen Ankunftszeit an diesem Stopp p ausgehend vom Startstopp s und der maximalen Abfahrtszeit, sodass die Ankunft an t noch vor der maximalen Ankunftszeit τ_t liegt. Die Zuordnung von erwarteten Ankunftszeiten zu Abfahrtszeiten erfolgt dabei so, dass alle Abfahrtszeiten, die kein Label und somit keine erwartete Ankunftszeit besitzen, dem Label mit der nächstgrößeren Abfahrtszeit zugeordnet werden. Zusätzlich müssen zur Erfüllung der Invariante an allen Umstiegspunkten sichere Alternativreisen zum Zielstopp existieren, die vor der maximalen Ankunftszeit ankommen. Durch die Einschränkung der Anzahl der Umstiege beinhalten die Reisen der minimalen erwarteten Ankunftszeiten maximal k Streckenabschnitte und Trips, die der Nutzer auf seiner Fahrt zum Zielstopp nimmt.

Beweis. Die Invariante kann über einen Induktionsbeweis bewiesen werden.

In der ersten Runde des Algorithmus werden alle Trips ausgewählt, die in dem Intervall zwischen der frühesten Ankunftszeit am Zielstopp t und der maximalen Ankunftszeit τ_t ankommen. Davon ausgehend werden die erwarteten Ankunftszeiten der Trips berechnet. Da es in der ersten Runde noch zu keinen Umstiegen kommen kann, besteht diese aus der Summe der Ankunftszeit des Trips und seiner erwarteten Verspätung (Z. 41-42 im Pseudocode). Dies entspricht der Definition für die erwartete Ankunftszeit von Streckenabschnitten, deren Ankunftsstopp der Zielstopp ist. Alle Stopps, die auf den ausgewählten Trips vor dem Zielstopp angefahren werden, erhalten dementsprechend ein Label mit dessen erwarteter Ankunftszeit, falls dieses nicht von einem anderen Label dominiert wird. Dadurch werden für alle Stopps, die innerhalb des Intervalls Reisen ohne Umstieg zu t besitzen, die minimalen erwarteten Ankunftszeiten für $k = 1$ gespeichert. Da alle erstellten Streckenabschnitte direkt zu dem Zielstopp führen, ist die Bedingung der sicheren Reisen erfüllt.

Für ein beliebiges aber festes k gelte die Invariante.

In der Runde $k + 1$ werden die neuen erwarteten Ankunftszeiten für jeden Stopp p mit Hilfe der erwarteten Ankunftszeiten aus den vorherigen Runden wie folgt berechnet: Für p sind im ersten Schritt alle Trips von Routen relevant, in denen p vor dem Stopp, der in der letzten Runde markiert wurde, angefahren wird. Dabei müssen nur Trips berücksichtigt werden, die an dem markierten Stopp vor der spätesten Abfahrtszeit der Labels, die in der letzten Runde hinzugefügt wurden, ankommen. Bei späteren Trips würden nach dem Umstieg nur Labels ausgewählt werden, deren zugehörigen Reisen weniger als $k - 1$ Umstiege besitzen. Die zugehörigen neuen erwarteten Ankunftszeiten

wurden somit schon spätestens in der $k-1$-ten Runde berechnet und den Beuteln hinzugefügt. Eine Berechnung in Runde $k+1$ würde die gleichen Labels erneut erzeugen. Zusätzlich werden nur Trips ausgewählt, die nicht vor der frühest möglichen Ankunftszeit an dem markierten Stopp ankommen, da alle anderen für den Nutzer nicht erreichbar sind und somit keine Relevanz für ihn besitzen (Z. 36-38). Anschließend berechnet sich die erwartete Ankunftszeit der ausgewählten Trips über die gewichtete Addition der erwarteten Ankunftszeiten der Reisen, die bei einer möglichen Verspätung des ankommenden Trips ausgewählt werden können. Die Berechnung entspricht dabei der Definition aus Kapitel 2.1.2 für die erwartete Ankunftszeit von Streckenabschnitten, die nicht den Zielstopp als Ankunftsstopp besitzen. Die verwendeten erwarteten Ankunftszeiten sind in den Labels, die spätestens in der k-ten Runde für den Ankunftsstopp des neuen Labels hinzugefügt wurden, gespeichert und somit nach der Invariante für Reisen mit maximal $k-1$ Umstiegen und k Streckenabschnitten minimal. Für mögliche neue Labels, für die kein Label mit einer Abfahrtszeit nach ihrer maximalen Verspätung existiert, kann die erwartete Ankunftszeit nicht berechnet werden und sie werden somit nicht berücksichtigt (Z. 43-54). Dies erfüllt die Bedingung, dass für jeden Streckenabschnitt an seinem Ankunftsstopp eine sichere Reise zum Zielstopp existieren muss. Für alle anderen ausgewählten Trips werden die erwarteten Ankunftszeiten berechnet und die zugehörigen Labels für den Stopp p erstellt. Dabei werden alle Labels in das Array von p hinzugefügt, die nicht durch ein Label mit einer späteren Abfahrtszeit dominiert werden. Somit gilt für jede mögliche Abfahrtszeit an Stopp p: Falls eine erwartete Ankunftszeit existiert, deren zugehörige Reisen maximal k Umstiege besitzen, ist die minimale erwartete Ankunftszeit entweder in einem Label gespeichert, das in dieser Runde hinzugefügt wurde. Die zugehörige längste Reise besitzt somit k Umstiege. Andernfalls wurde das Label bereits in einer der vorherigen Runden hinzugefügt und die zugehörigen Reisen haben somit weniger als k Umstiege. In beiden Fällen ist die erwartete Ankunftszeit durch die vorangegangene Berechnung minimal und die zugehörigen Reisen bestehen aus maximal $k+1$ Streckenabschnitten und k Umstiegen. Dies erfüllt die Invariante.

Lemma 4.2.2
Aus Lemma 4.2.1 lässt sich schließen, dass der RAPTOR MEAT Algorithmus die minimale erwartete Ankunftszeit und den zugehörigen Entscheidungsgraphen korrekt berechnet, falls für das gegebene MEAT Problem eine Lösung existiert.

Beweis. Falls für das gegebene MEAT Problem eine Lösung existiert, besitzt eine der Reisen, die der minimalen erwarteten Ankunftszeit zugeordnet werden können, eine maximale Anzahl an Streckenabschnitten $k_{\max}$. Führt man nun mindestens $k_{\max}$ Runden des Algorithmus durch, garantiert die Invariante, dass das Label des Startstopps mit der geringsten erwarteten Ankunftszeit die Lösung des Problems darstellt. Dies ist das Label mit der kleinsten Abfahrtszeit des Graphen. Da für das Label alle zugehörigen Reisen inklusive ihrer Alternativreisen bekannt sind und die Bedingung der sicheren Alternativreisen erfüllt ist, kann ein Entscheidungsgraph ausgehend von diesem Label extrahiert werden. Aus Lemma 6.3 des CSA MEAT in [DPSW18] folgt, dass dann auch ein optimaler Entscheidungsgraph für das Label mit der minimalen Ankunftszeit existiert. Ist $k_{\max}$ bereits vor der Ausführung des Algorithmus bekannt, kann er bereits nach $k_{\max}$ Runden abgebrochen werden. Andernfalls terminiert er erst nachdem in einer Runde keine neuen Stopps markiert wurden. Da jede Reise, deren erwartete Ankunftszeit berechnet wird, eine endliche Anzahl an Umstiegen besitzt, terminiert der Algorithmus in jedem Fall. Die endliche Anzahl an Umstiegen ist für jede Reise gesichert, da für jeden Stopp über die früheste Ankunftszeit von dem Startstopp ausgehend eine minimale Abfahrtszeit der zugehörigen Streckenabschnitte definiert ist. Da jeder

Streckenabschnitt l eine Fahrtzeit von $l_{\text{arr\_time}} - l_{\text{dep\_time}} > 0$ hat, erreicht jede Reise, die vor der maximalen Ankunftszeit am Zielstopp ankommt, zwangsläufig die untere Grenze bevor sie eine unendliche Anzahl an Streckenabschnitten besitzt.

4.2.3 Komplexitätsanalyse

Um die Komplexität des RAPTOR MEAT Algorithmus zu bestimmen, berechnen wir zunächst die Komplexitäten der drei Bestandteile einer Runde.

Beim Auswählen der Routen, die in der aktuellen Runde behandelt werden, wird über alle markierten Stopps der letzten Runde iteriert. Im schlechtesten Fall sind dies alle $|S|$ Stopps. Für jeden markierten Stopp p werden alle Routen $R(p)$ ausgewählt, die p enthalten, und in dem Array Q gespeichert. Für den Stopp p werden somit $|R(p)|$ Routen betrachtet (Z. 13-22 im Pseudocode). Insgesamt ergibt sich für den ersten Schritt der Runde eine Komplexität im schlechtesten Fall von $O(\sum_{p \in S} |R(p)|)$.

Die Iteration über die Routen (Z. 24-62) besteht aus drei Schritten. Diese werden für alle ausgewählten Routen durchgeführt. Im schlechtesten Fall entspricht dies allen $|R|$ Routen. Für jede behandelte Route r wird über alle Stopps $S(r)$ von r iteriert. Dabei führen wir für jeden Stopp $p \in S(r)$ die Update-, Merge- und Add-Operation durch. Bei der Update-Operation wird jedes Label der aktuellen Route r mit der Abfahrtszeit des Stopps p aktualisiert (Z. 28-30). Da die Labels jeweils einem Trip der Route zugeordnet sind und es durch die Dominanzregeln für jeden der Trips der Route maximal ein Label gibt, müssen im schlechtesten Fall $|T(r)|$ Labels aktualisiert werden. Das Aktualisieren eines Labels erfolgt in konstanter Zeit. Die Komplexität der Update-Operation des Stopps p der Route r liegt somit in $O(|T(r)|)$. Bei der Merge-Operation (Z. 32) können wir nutzen, dass der Routenbeutel B_r und der Beutel $B_c(p)$ mit den Labels der aktuellen Runde für den Stopp p jeweils nach den Abfahrtszeiten ihrer Labels sortiert sind. Die Merge-Operation kann daher linear in der Größe der beiden Beutel erfolgen. Der Routenbeutel besitzt wie bei der Update-Operation maximal $|T(r)|$ Labels. In $B_c(p)$ ist für alle Trips $T(p)$, in denen p enthalten ist, jeweils maximal ein Label gespeichert. Die Komplexität der Merge-Operation liegt daher in $O(|T(r)| + |T(p)|)$. Die Add-Operation (Z. 34-62), bei der neue Labels dem Routenbeutel hinzugefügt werden, muss nur für markierte Stopps der letzten Runde ausgeführt werden. Im schlechtesten Fall ist das für jeden Stopp p der Route r notwendig. Innerhalb der Operation wird über alle Trips $T(r)$ der Route r iteriert und daraus die neuen Labels erzeugt. Beim Erzeugen der Labels müssen deren erwartete Ankunftszeiten berechnet werden. Dafür werden potenziell alle Labels aus den vorherigen Runden benötigt, die für diesen Stopp in $B^*(p)$ gespeichert sind. Im schlechtesten Fall sind dies $|T(p)|$ Label. Das Erstellen aller neuen Labels für den Stopp p der Route r liegt somit in $O(|T(r)| \cdot |T(p)|)$. Da die Trips nach ihren Abfahrtszeiten sortiert sind, kann der anschließende Vorgang des Einfügens der neuen Labels in den Routenbeutel wieder linear erfolgen (Z. 59). Beide beteiligten Beutel beinhalten im schlechtesten Fall $|T(r)|$ Labels, was dazu führt, dass die Add-Operation des Stopps p eine Komplexität von $O(|T(r)| \cdot |T(p)| + 2 \cdot |T(r)|) = O(|T(r)| \cdot |T(p)|)$ besitzt. Insgesamt beträgt die Komplexität des zweiten Teils unseres Algorithmus $O(\sum_{r \in R} \sum_{p \in S(r)} (2 \cdot |T(r)| + |T(p)| + |T(p)| \cdot |T(r)|)) = O(\sum_{r \in R} \sum_{p \in S(r)} (|T(p)| \cdot |T(r)|))$.

Der dritte Bestandteil jeder Runde ist das Zusammenfügen der Beutel B_c bestehend aus den Labels der aktuellen Runde und der Beutel B^*, die die Labels der vorherigen Runden enthalten (Z. 64-75). Dies muss für jeden Stopp erfolgen, für den in der aktuellen Runde mindestens ein neues Label erstellt wurde. Im schlechtesten Fall sind dies alle $|S|$ Stopps. Für jeden Stopp p sind beide

Beutel wiederum nach der Abfahrtszeit ihrer Labels sortiert. Die Operation des Zusammenfügens kann also in linearer Zeit erfolgen. In beiden Beuteln sind im schlechtesten Fall $|T(p)|$ Labels enthalten. Für die gesamte Aktualisierung der nicht dominierten Labels beträgt die Komplexität daher $O(\sum_{p \in S} |T(p)|)$.

Da alle drei Bestandteile in jeder Runde ausgeführt werden, beträgt die Komplexität des RAPTOR MEAT Algorithmus für K Runden insgesamt $O(K \cdot (\sum_{p \in S}(|R(p)|) + \sum_{r \in R} \sum_{p \in S(r)}(|T(p)| \cdot |T(r)|) + \sum_{p \in S} |T(p)|))$.

Jeder Route ist mindestens ein Trip zugeordnet, daher gilt für alle $p \in S$: $|R(p)| \leq |T(p)|$. Somit kann die Komplexität vereinfacht werden, indem der erste Bestandteil jeder Runde entfernt wird, da er durch den Teil der Aktualisierung der Beutel B^* dominiert wird. Da zusätzlich jeder Stopp, der in mindestens einem Trip enthalten, auch mindestens einer Route zugeordnet ist, kann auch der dritte Bestandteil der Komplexität entfernt werden. Dieser wird durch die Komplexität der Iteration über die Routen dominiert. Als Komplexität ergibt sich somit $O(K \cdot \sum_{r \in R} \sum_{p \in S(r)}(|T(p)| \cdot |T(r)|))$.

4.2.4 Transferoptimierung

Wie in Kapitel 4.2.2 gezeigt, liefert die rundenbasierte Berechnung des RAPTOR MEAT Algorithmus die Information, wie viele Umstiege maximal genommen werden müssen, um ausgehend von einem Label zu dem Zielstopp zu gelangen. Somit garantiert der Entscheidungsgraph, der auf der Basis der Labels in der k-ten Runde mit $k > 0$ erstellt werden kann, dass er die minimale erwartete Ankunftszeit aller Graphen hat, die Reisen mit maximal $k - 1$ Umstiegen zwischen Start- und Zielstopp besitzen. Mit diesem Wissen wollen wir den RAPTOR MEAT Algorithmus aus 4.2.1 in der Hinsicht erweitern, dass alternative Ergebnisse dem Nutzer vorgeschlagen werden, die Vorteile hinsichtlich der maximalen Anzahl an Umstiegen besitzen. Dabei sollen kleinere Verschlechterungen der erwarteten Ankunftszeit durch Verbesserungen der maximalen Anzahl an Umstiegen aufgewogen werden. Dies kann für Passagiere den Komfort der Reise verbessern, da sie seltener zwischen Trips umsteigen müssen aber eine annähernd gute Ankunftszeit erwarten können. Die entstehende Variante des Algorithmus bezeichnen wir mit RAPTOR MEAT Transfer Optimisation (TO).

Dazu müssen wir die Standardvariante des RAPTOR MEAT Algorithmus anpassen. Da wir nach der Ausführung des Algorithmus potenziell Entscheidungsgraphen basierend auf den Labels aus bestimmten vorherigen Runden erstellen müssen, sichern wir in jeder Runde den Zustand des expectedArrivalTimes Arrays, das die Beutel B^* mit allen nicht dominierten Labels für jeden Stopp speichert. Dabei ist es ausreichend jeweils die Beutel von den Stopps zu speichern, für die in der aktuellen Runde neue Labels hinzugefügt oder bestehende Labels durch neue ersetzt wurden. Auf der Basis dieser gespeicherten Beutel können Entscheidungsgraphen aus allen durchgeführten Runden erstellt werden.

Abgesehen von dieser Anpassung führen wir die Standardvariante des RAPTOR MEAT Algorithmus durch. Nach seiner Ausführung müssen wir die erste Runde bestimmen, in der das Verhältnis aus der maximalen Anzahl an Umstiegen zu der erwarteten Ankunftszeit passend ist. Als Paramter müssen wir hierfür die Zeit pro verringerter Anzahl an Umstiegen definieren, um die sich die erwartete Ankunftszeit von der minimalen erwarteten Ankunftszeit unterscheiden darf. Wir haben diesen Wert initial auf fünf Minuten gesetzt. Somit kann sich pro Umstieg, um den sich die maximale

Anzahl an Umstiegen des Ergebnisses von dem Graphen der minimalen erwarteten Ankunftszeit abweicht, die zugehörige erwartete Ankunftszeit um fünf Minuten erhöhen. Dieser Parameter kann jedoch je nach Präferenz des Nutzers beliebig angepasst werden.

Um die Anforderung umzusetzen, bestimmen wir zunächst die Runde k_{res} in der das Label mit der minimalen erwarteten Ankunftszeit des RAPTOR MEAT Algorithmus in den expectedArrivalTimes Beutel $B^*(s)$ des Startstopps hinzugefügt wurde. Die maximale Anzahl an Umstiegen, die die Reisen des zugehörigen Entscheidungsgraphen umfassen, ist somit $k_{\text{res}} - 1$. Anschließend iteriert der Algorithmus jeweils über das erste Element der gespeicherten expectedArrivalTimes Beutel des Startstopps s von der ersten bis zur $k_{\text{res}} - 1$-ten Runde. Das erste Element ist dabei jeweils das Label mit der kleinsten Abfahrtszeit und somit auch der kleinsten erwarteten Ankunftszeit. Für die Labels überprüfen wir jeweils, ob ihre verringerte Anzahl an maximalen Umstiegen die spätere erwartete Ankunftszeit im Vergleich zu dem Label der minimalen erwarteten Ankunftszeit ausgleicht. Sobald dies erfüllt ist, stoppt der Algorithmus und erstellt den Entscheidungsgraphen basierend auf dem Label aus dieser Runde. An jedem Label L, das einen Umstiegspunkt innerhalb des Graphen darstellt, werden die nachfolgenden Labels aus dem expectedArrivalTimes Beutel ausgewählt, der in der Runde vor L erstellt wurde. Somit wird garantiert, dass die maximale Anzahl an Umstiegen, die durch das ausgewählte erste Label definiert ist, auf den Reisen zwischen Start- und Zielstopp nicht überschritten wird.

Sollte kein Label aus vorherigen Runden gefunden werden, das die Bedingungen hinsichtlich der Anzahl an Umstiegen und der erwarteten Ankunftszeit erfüllt, verwenden wir das Label mit der minimalen erwarteten Ankunftszeit des RAPTOR MEAT Algorithmus.

Bei dem CSA MEAT ist eine ähnliche Optimierung bezüglich der maximalen Anzahl an Umstiegen nur schwer möglich, da wir während der Durchführung des Algorithmus keine Möglichkeit haben auf die Anzahl der Umstiege zuzugreifen. Somit können wir aus dem Ergebnis des Algorithmus auch keine Entscheidungsgraphen mit einer bestimmten oberen Grenze als maximale Anzahl an Umstiegen erstellen.

4.2.5 Transferbeschränkung

Eine weitere Anwendungsmöglichkeit des rundenbasierten Vorgehens des RAPTOR MEAT Algorithmus ist die Beschränkung der maximalen Anzahl an Umstiegen. So kann es für Passagiere von Interesse sein, dass sie während ihrer Reise selbst im schlechtesten Fall eine maximale Anzahl an Umstiegen nicht überschreiten.

Der RAPTOR MEAT Algorithmus kann dazu verwendet werden, die minimale erwartete Ankunftszeit und den zugehörigen Entscheidungsgraphen zu berechnen, bei denen alle enthaltenen Reisen eine maximale Anzahl an Umstiegen besitzen. Dazu muss dem Algorithmus zusätzlich zu unserer bisherigen Eingabe dieses Maximum k_{max} als Parameter übergeben werden.

Anschließend werden $k_{\text{max}} + 1$ Runden des RAPTOR MEAT Algorithmus ausgeführt. Falls eine erwartete Ankunftszeit für diese maximale Anzahl an Umstiegen berechnet werden kann, befindet sich die minimale erwartete Ankunftszeit in dem ersten Label von $B^*(s)$. Um den zugehörigen Entscheidungsgraphen erstellen zu können, benötigen wir wie in der Transferoptimierungsvariante die Zustände der Beutel B^* aller durchgeführten Runden. Diese Variante des Algorithmus bezeichnen wir mit RAPTOR MEAT Transfer Limitation (TL).

4.2.6 Parallelisierung

Ein weiterer Vorteil, den die RAPTOR Algorithmen im Vergleich zu den CSAs besitzen, ist die Möglichkeit große Bestandteile des Algorithmus zu parallelisieren. Da unser RAPTOR MEAT Algorithmus auf dem McRAPTOR Algorithmus basiert, ist es möglich auch diesen zu parallelisieren. Dies ist in jeder Runde bei der Iteration über die Routen und in der Schleife, in der die Labels der erwarteten Ankunftszeiten aller Stopps aktualisiert werden, möglich, da die einzelnen Elemente jeweils unabhängig voneinander betrachtet werden können.

Bei der Iteration über alle ausgewählten Routen einer Runde hat die Reihenfolge in der sie angeschaut werden keine Bedeutung. Somit können die Routen auf die CPU Kerne verteilt werden. Diese berechnen für die zugehörigen Trips und Stopps ihre erwartete Ankunftszeiten und erstellen daraus neue Labels. Bei der Aufteilung der Routen auf die Threads spielt es eine Rolle, dass alle für einen Stopp p' in der aktuellen Runde erstellten Labels in dem zugehörigen ExpectedArrivalTimesOfCurrentRound Beutel $B_c(p')$ gespeichert werden. Da Stopps in mehreren Routen enthalten sein können, die von unterschiedlichen Threads bearbeitet werden, kann es hier zu Konflikten kommen. Um diese möglichst effizient zu lösen, ergeben sich zwei Optionen. Die Option der Konfliktgraphen wurde bereits für den McRAPTOR Algorithmus in [DPW15] eingeführt und kann auch in unserer Variante verwendet werden. Hierbei erstellen wir einen ungerichteten Graphen, der alle Routen des Datensatzes als Knoten besitzt. Kanten werden als Verbindung zwischen zwei Knoten hinzugefügt, falls die zwei zugehörigen Routen mindestens einen gemeinsamen Stopp beinhalten. Anschließend färben wir den Graphen mit einem Greedy Algorithmus so, dass keine adjazenten Knoten die gleiche Farbe besitzen. Wir verteilen die Routen nun in jeder Runde so auf die CPU Kerne, dass jeder Kern lediglich Routen einer Farbe des Graphen besitzt. Damit ist gesichert, dass kein B_c Beutel eines Stopps in einer Runde von zwei Threads bearbeitet wird. Bei der zweiten Option können die Routen beliebig auf die CPU Kerne verteilt werden. Um trotzdem Konflikte zu verhindern, führt jeder Thread einen eigenen B_c Beutel für jeden der Stopps, die in seinen Routen behandelt werden. Innerhalb dieser müssen die Dominanzregeln bezüglich der erwarteten Ankunftszeiten erfüllt sein. Die Labels der Beutel sind nach ihren Abfahrtszeiten sortiert. Bei der Variante muss zusätzlich das Aktualisieren der erwarteten Ankunftszeiten in der dritten Schleife jeder Runde angepasst werden. Hierbei erfolgt die Merge-Operation für jeden Stopp, für den neue Labels erstellt wurden, nun zwischen den Labels der vorherigen Runden und allen B_c Beuteln der Threads, die in dieser Runde neue Labels für den Stopp erstellt haben. Die zweite Option hat den Vorteil, dass keine Vorauswahl bei der Aufteilung der Stopps erfolgen muss. Sie besitzt aber den Nachteil, dass bei dem abschließenden Vorgang des Zusammenfügens potenziell mehr Labels berücksichtigt werden müssen, da durch die fehlende Synchronisation der Threads während der Behandlung der Routen weniger dominierte Labels frühzeitig entfernt werden.

Bei der RAPTOR MEAT Variante des McRAPTOR Algorithmus kann zudem das Aktualisieren der erwarteten Ankunftszeiten am Ende jeder Runde parallelisiert werden, da dies unabhängig voneinander für alle Stopps, für die in der aktuellen Runde neue Labels erstellt wurden, erfolgen kann. Da es hier keine Schreibkonflikte zwischen den einzelnen Threads gibt, erfolgt die Aufteilung der Stopps auf die Threads beliebig.

5 Experimente

Wir führen nun Benchmarktests für die neuen Algorithmen aus dem Kapitel 4 durch. Dabei vergleichen wir die Ausführungszeiten des CSA ExpAT und RAPTOR MEAT Algorithmus mit denen des CSA MEAT. Zusätzlich analysieren wir die Größe der entstehenden Entscheidungsgraphen und prüfen die Korrektheit und Relevanz der berechneten minimalen erwarteten Ankunftszeiten.

5.1 Voraussetzungen

Als Grundlage für die Tests nutzen wir die Datensätze des Schienenregional- und Schienenfernverkehrs in Deutschland. Diese umfassen alle S-Bahnen, Regionalbahnen, ICs und ICEs. Zusammengesetzt besteht der Datensatz aus insgesamt 7609 Stopps, 7663 Routen, 50438 Trips und 711496 Stoppzeiten. Aus diesen werden 661198 Verbindungen für die CSAs erstellt.

Wir führen alle Tests auf denselben 1000 zufällig ausgewählten Anfragen aus und berechnen jeweils die Durchschnitts- und Maximalwerte der Resultate. Als Anfrage wählen wir zufällig zwei Stopps, eine minimale Abfahrtszeit und ein Datum innerhalb der nächsten Woche aus. Dies ist erforderlich, da je nach Wochentag unterschiedliche Trips verfügbar sind. Zudem verwenden wir ausschließlich Reisen, deren früheste sichere Ankunftszeit maximal 24 Stunden nach der minimalen Abfahrtszeit liegt.

Als Verspätungsmodell nutzen wir das Modell von Disser et al. (2008) in [DMS08]. Es basiert auf der Funktion $\text{rel} : \tau \mapsto 1 - e^{\ln(1-a)-\frac{\tau}{b}}$. Diese beschränkt exponentielle Funktion kann über die Parameter a und b gesteuert werden, da $\text{rel}(0) = a$ gilt und über b die Steigung der Funktion definiert ist. Für die Trips des Schienenfernverkehrs wählen wir dabei andere Parameter aus als bei den Trips des Schienenregionalverkehrs, da wir annehmen, dass es durch die deutlich längeren Fahrtzeiten des Fernverkehrs wahrscheinlicher zu längeren Verspätungen kommt. Wir wählen als maximale Verspätung des Fernverkehrs 30 Minuten und 15 Minuten für den Regionalverkehr. Als Parameter setzen wir dementsprechend $a = 0.5$, $b = 7$ im Fernverkehr und $a = 0.65$, $b = 3.5$ im Regionalverkehr. Dies sorgt dafür, dass die erwartete Verspätung im Fernverkehr ca. drei Minuten und im Regionalverkehr etwa eine Minute beträgt. Auch hier nehmen wir an, dass es im Fernverkehr zu längeren Verspätungen kommt. Die Wahrscheinlichkeit, dass Züge keine Verspätung besitzen, beträgt mit diesen Werten im Fernverkehr 50% und im Regionalverkehr 65%. Die darauf folgenden Werte $\text{rel}(\tau)$ für die möglichen Verspätungszeiten $\tau > 0$ geben jeweils an, mit welcher Wahrscheinlichkeit die Verspätung des Trips kleiner oder gleich τ ist. Je größer die Verspätungszeiten werden, desto unwahrscheinlicher ist es, dass sie auftreten. Um die Werte effizienter berechnen zu können, diskretisieren wir das Intervall der möglichen Verspätungen in 1-Minuten-Schritten. Die zugehörigen Wahrscheinlichkeiten können wir schon beim Start des Programms berechnen und die wiederholt benötigten Wahrscheinlichkeiten müssen somit nicht jedes Mal neu ermittelt werden. Die Testergebnisse des ersten Verspätungsmodells kennzeichnen wir mit DM1 (Delay Model 1).

Um den Einfluss des Verspätungsmodells auf unsere Tests erkennen zu können, führen wir sie zusätzlich für ein zweites Modell durch. Dabei setzen wir die maximale Verspätung für alle Trips wie in den Tests des CSA MEAT in [DPSW18] auf eine Stunde fest. Wir wählen die Parameter $a = 0.6$ und $b = 7$. Somit beträgt die Wahrscheinlichkeit, dass es zu keiner Verspätung kommt 60% und der Erwartungswert der Verspätung eines Trips etwa drei Minuten. Die Testergebnisse des zweiten Verspätungsmodells kennzeichnen wir mit DM2 (Delay Model 2).

Für die MEAT und ExpAT Algorithmen verwenden wir die α-beschränkte Variante der Probleme. Dabei führen wir alle Tests für $\alpha = 1$, $\alpha = 2$ und $\alpha = 3$ durch, um mögliche Einflüsse der Größe des initialen Intervalls an Verbindungen, die genutzt werden können, zu erkennen.

Die für die Tests verwendete Hardware besitzt einen Intel Core i5 mit 6 Kernen, eine 500GB NVMe Festplatte und 64GB RAM. Auf ihr läuft Ubuntu 20.04. Bei dem RAPTOR MEAT Algorithmus verwenden wir die Single-Core Variante ohne Parallelisierung.

5.2 Benchmarktests

Bei den Benchmarktests messen wir die Ausführungszeiten der beiden neu entwickelten Algorithmen RAPTOR MEAT und CSA ExpAT. Als Vergleichswerte verwenden wir die Laufzeiten des CSA MEAT. Um die Werte genauer analysieren zu können, bestimmen wir jeweils die Zeiten der kompletten Ausführung und diese aufgeteilt auf die drei Bestandteile der Algorithmen. Dabei erfolgt die Aufteilung jeweils auf die Initialisierung, den eigentlichen Algorithmus, der die erwarteten Ankunftszeiten berechnet, und die Erstellung des Entscheidungsgraphen. Die zugehörigen Zeiten sind in der Tabelle 5.1 für das erste Verspätungsmodell und in der Tabelle 5.2 für das zweite jeweils in Millisekunden angegeben. Dabei umfassen sie jeweils die durchschnittlichen und maximalen Zeiten. Die Werte sind für die drei ausgewählten Optionen von α aufgeführt.

Im ersten Verspätungsmodell lässt sich für $\alpha = 1$ erkennen, dass der neu entwickelte RAPTOR MEAT Algorithmus geringere Ausführungszeiten als der bereits bestehende CSA MEAT besitzt. Dies zeigt sich durch die 25ms statt 57ms, die er im Durchschnitt zur Berechnung der minimalen erwarteten Ankunftszeiten in dem entscheidenden Bestandteil des Algorithmus benötigt. Auch die durchschnittliche Dauer des gesamten Algorithmus beträgt 182ms statt 214ms. Dies lässt sich damit erklären, dass bei dem CSA MEAT alle Verbindungen betrachtet werden, die ausgehend von dem Startstopp nach der minimalen Abfahrtszeit erreicht werden können. Als obere Grenze ist hier lediglich die maximale Ankunftszeit definiert. Somit kommt es zu keiner Überprüfung, ob der Zielstopp von der Verbindung aus erreichbar ist. Im Gegensatz dazu startet der RAPTOR MEAT Algorithmus ausschließlich mit Trips, die zu dem Zielstopp führen. Auch in den folgenden Runden werden lediglich Stoppzeiten von Trips beachtet, bei denen man mit Umstiegen zum Zielstopp vor der maximalen Ankunftszeit gelangt. Dies sorgt dafür, dass einige irrelevante Trips und somit Verbindungen aus dem CSA MEAT hier nicht betrachtet werden müssen. Der Nachteil des RAPTOR MEAT Algorithmus ist jedoch, dass durch die rundenbasierte Struktur die gleichen Stoppzeiten eines Trips potenziell in mehreren Runden verwendet werden. Bei dem CSA MEAT werden alle Verbindungen in der Reihenfolge ihrer Abfahrtszeiten betrachtet. Somit ist hier gesichert, dass jede Verbindung maximal einmal angeschaut wird. Die Maximalzeiten für $\alpha = 1$ zeigen schon, dass dieser Vorteil des CSA MEAT gegenüber dem Vorteil des RAPTOR MEAT Algorithmus überwiegen kann. So beträgt sie beim CSA MEAT lediglich 558ms und im Vergleich dazu bei dem RAPTOR MEAT Algorithmus 714ms. Auch die Zeiten für $\alpha = 2$ und $\alpha = 3$ weisen dies

auf, da hier sogar bei den Durchschnittszeiten der CSA MEAT schneller als der RAPTOR MEAT Algorithmus ist. Dies lässt sich damit begründen, dass bei späteren maximalen Ankunftszeiten mehr Trips im initialen Schritt am Zielstopp ausgewählt werden. Somit werden potenziell mehr Stopps markiert und auch an ihnen in größeren Intervallen nach Trips gesucht, in die Passagiere umsteigen können. Dies hat insgesamt zur Folge, dass in jeder Runde deutlich mehr Stoppzeiten betrachtet und auch potenziell mehr Stoppzeiten in unterschiedlichen Runden mehrfach verwendet werden. Zudem können durch die größere Differenz zwischen der minimalen Abfahrtszeit und maximalen Ankunftszeit längere Reisen, die aus mehr Streckenabschnitten bestehen, zwischen s und t potenziell gefunden werden. Dies sorgt dafür, dass mehr Runden des RAPTOR MEAT Algorithmus ausgeführt werden müssen. Dahingegen steigt der Aufwand des CSA MEAT linear, da sich hier lediglich die Anzahl der betrachteten Verbindungen um den Unterschied der Intervallgröße zwischen minimaler Abfahrtszeit und maximaler Ankunftszeit verändert. Dieses Verhalten lässt sich an der Abbildung 5.1 erkennen, in der jeweils die Laufzeiten der gesamten Algorithmen für die drei Werte von α dargestellt sind. Für $\alpha = 1$ weisen die RAPTOR MEAT Algorithmen noch geringere Zeiten auf, aber danach verlaufen die Zeiten der CSAs linear. Im Gegensatz dazu steigen die Laufzeiten der RAPTOR MEAT Algorithmen deutlich an.

An den maximalen Laufzeiten kann man erkennen, dass sich die maximale Ausführungszeit bei dem CSA MEAT jeweils deutlich weniger von der zugehörigen Durchschnittszeit als bei dem RAPTOR MEAT Algorithmus unterscheidet. Dies lässt sich damit begründen, dass bei dem RAPTOR MEAT Algorithmus die Anzahl der betrachteten Stoppzeiten zusätzlich von der Anbindung des Zielstopps im Verkehrsnetz abhängt. Je besser dieser angebunden ist, desto mehr Trips exisitieren, die zu ihm führen. Dadurch werden potenziell mehr Stoppzeiten behandelt. Da die Start- und Zielstopps unserer Anfragen zufällig ausgewählt werden, kann es zu stark unterschiedlich angebundenen Zielstopps und somit zu großen Unterschieden bei den Ausführungszeiten in den Tests kommen. Im Gegensatz dazu hängt die Anzahl der betrachteten Verbindungen bei dem CSA MEAT nicht von der Anbindung des Zielstopps ab. Dies sorgt dafür, dass es insgesamt weniger Ausreißer gibt und sich die Maximalzeiten weniger von den Durchschnittswerten unterscheiden.

Da bei allen Algorithmen die gleichen Werte in der Initialisierungsphase berechnet werden müssen, dauert diese jeweils ungefähr gleich lang. Wir führen dabei CSAs zur Berechnung der frühesten Ankunftszeiten durch und profitieren hier davon, dass deren Ausführungszeiten für größere Intervalle annähernd linear steigen. Im Vergleich zu den beiden anderen Bestandteilen weist die Erstellung der Entscheidungsgraphen den kleinsten Anteil auf. Deren durchschnittliche Dauer wird zudem durch die unterschiedlichen Beschränkungen kaum beeinflusst.

Die Ausführungszeiten von CSA MEAT und CSA ExpAT stimmen in allen Fällen fast überein. Der Unterschied der beiden Algorithmen liegt darin, dass die Dominanzregeln bei dem CSA MEAT durch die erwarteten Ankunftszeiten und die des CSA ExpAT von den geplanten Ankunftszeiten bestimmt werden. Dies hat jedoch keinen Einfluss auf die Anzahl der Verbindungen, die betrachtet werden müssen. Daraus resultieren die ähnlichen Ausführungszeiten der beiden Algorithmen.

Der Vergleich zu dem zweiten Verspätungsmodell in 5.2 zeigt, dass sich bei diesem die Laufzeiten ähnlich verhalten. Auch hier ist die RAPTOR MEAT Variante für $\alpha = 1$ schneller als die Version des CSA MEAT. Jedoch steigen die Ausführungszeiten der CSAs linear und sind somit für größere α deutlich geringer als die des RAPTOR MEAT Algorithmus. Insgesamt sind alle Laufzeiten des zweiten Verspätungsmodells größer als die des ersten Modells. Das zeigt auch die Abbildung 5.1. Die größeren maximalen Verspätungen der Trips in dem zweiten Modell sorgen für spätere

		$\alpha = 1$				$\alpha = 2$				$\alpha = 3$			
		Complete	Init	Algorithm	Graph	Complete	Init	Algorithm	Graph	Complete	Init	Algorithm	Graph
RAPTOR	Avg	182	156	25	1.0	1013	249	762	1.1	2294	334	1958	1.2
MEAT	Max	714	493	390	7.8	5890	625	5399	2.7	11750	914	11065	8.0
CSA	Avg	214	156	57	1.1	398	249	148	1.1	584	335	248	1.2
MEAT	Max	558	416	194	4.2	1013	663	418	3.4	1491	861	729	5.4
CSA	Avg	214	157	57	1.0	391	251	139	1.1	565	336	228	1.1
ExpAT	Max	557	415	188	1.9	976	634	427	2.8	1417	863	650	4.5

Tabelle 5.1: Die Ausführungszeiten (in ms) der RAPTOR MEAT, CSA MEAT und CSA ExpAT Algorithmen und ihrer Bestandteile für alle drei Werte von α. Dabei erfolgt die Aufteilung in Initialisierung, den eigentlichen Algorithmus und die Erstellung der Graphen. Verwendet wurde das erste Verspätungsmodell mit maximal 15 bzw. 30 Minuten Verspätung.

		$\alpha = 1$				$\alpha = 2$				$\alpha = 3$			
		Complete	Init	Algorithm	Graph	Complete	Init	Algorithm	Graph	Complete	Init	Algorithm	Graph
RAPTOR	Avg	239	193	44	1.4	1806	300	1503	3.3	5550	414	5133	4.1
MEAT	Max	1036	553	694	8.6	11475	780	10886	94	70003	986	69141	121
CSA	Avg	264	192	71	1.3	488	301	185	2.7	766	415	348	3.2
MEAT	Max	563	421	204	7.0	1115	649	492	97	1778	927	891	163.5
CSA	Avg	264	193	71	1.0	480	302	176	1.3	710	418	291	1.3
ExpAT	Max	563	422	221	1.6	1024	646	426	5.4	1532	926	699	6.6

Tabelle 5.2: Die Ausführungszeiten (in ms) der RAPTOR MEAT, CSA MEAT und CSA ExpAT Algorithmen und ihrer Bestandteile für alle drei Werte von α. Dabei erfolgt die Aufteilung in Initialisierung, den eigentlichen Algorithmus und die Erstellung der Graphen. Verwendet wurde das zweite Verspätungsmodell mit maximal 60 Minuten Verspätung.

minimale sichere Ankunftszeiten. Somit sind die Intervalle der Trips und Verbindungen, die in den Algorithmen betrachtet werden müssen, größer. Dieser zusätzliche Aufwand resultiert in den längeren Ausführungszeiten.

Für den RAPTOR MEAT Algorithmus betrachten wir zusätzlich die Ausführungszeiten der drei Schleifen, die in jeder Runde ausgeführt werden und sich zu der Ausführungszeit des eigentlichen Algorithmus aus der Tabelle 5.1 zusammensetzen. Die Zeiten der Schleifen sind in der Tabelle 5.3 für das erste Verspätungsmodell jeweils in Millisekunden für die drei Werte von α dargestellt. Analog dazu befinden sich die Zeiten der Bestandteile des eigentlichen Algorithmus aus der Tabelle 5.2 in der Tabelle 5.4.

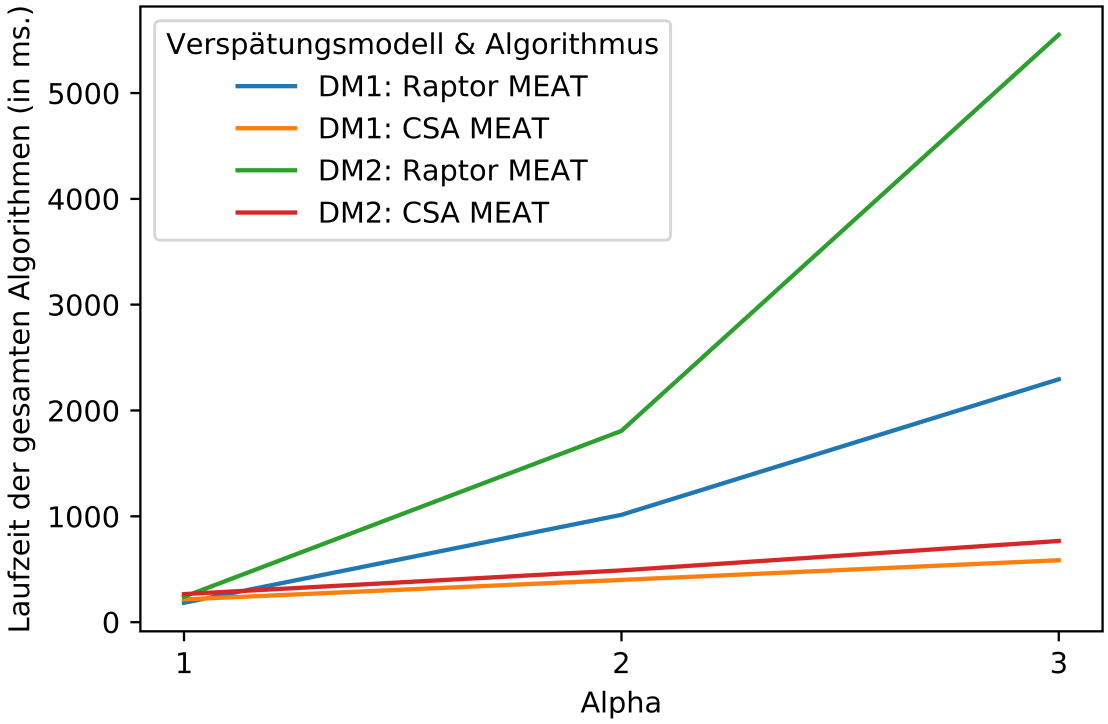

Abbildung 5.1: Die Gesamtlaufzeiten der CSA und RAPTOR MEAT Algorithmen. Die Laufzeiten der Algorithmen inklusive Initialisierungsphase und Erstellung der Graphen für alle drei Werte von α und die zwei unterschiedlichen Verspätungsmodelle.

Für beide Verspätungsmodelle kann man dabei erkennen, dass die Iteration über die Routen den längsten Teil des Algorithmus darstellt. In diesem Bestandteil werden für alle Trips, die mindestens einen markierten Stopp der letzten Runde enthalten, neue Labels erstellt und deren erwartete Ankunftszeiten berechnet. Bei allen drei Schleifen ist auch hier der Einfluss der vergrößerten Intervalle sichtbar, da sie für $\alpha = 2$ und $\alpha = 3$ länger benötigen. Besonders deutlich wird das bei der dritten Schleife, bei der für jeden Stopp die neuen Labels in den Beutel der Labels aus den vorherigen Runden hinzugefügt werden. Auch dies liegt daran, dass durch die größere maximale Ankunftszeit mehr Trips betrachtet und somit deutlich mehr Stopps markiert und neue Labels erstellt werden. Dies sorgt dafür, dass die Operation des Zusammenfügens öfter und mit größeren Beuteln durchgeführt werden muss. Der Aufwand der Initialisierungsschleife, in der die Routen für die aktuelle Runde ausgewählt werden, steigt ebenso, da hier die Ausführungszeit von der Anzahl der markierten Stopps der letzten Runde abhängt. Sie steigt somit bei wachsender maximalen Ankunftszeit an, jedoch nicht in dem Maße wie die der beiden anderen Schleifen. Bei allen drei Schleifen zeigt sich zudem der Einfluss der maximalen Verspätung der Trips, da das zweite Verspätungsmodell jeweils längere Laufzeiten aufweist.

Ein Vorteil des RAPTOR MEAT Algorithmus liegt wie in 4.2 beschrieben darin, dass Teile des Algorithmus parallelisiert werden können, da die Berechnungen hier unabhängig voneinander ablaufen. Dies umfasst die zweite und dritte Schleife. Bei der Schleife zur Iteration über die Routen können einzelne Routen unabhängig voneinander betrachtet werden. Bei der Schleife zum Aktualisieren der Labels mit den minimalen erwarteten Ankunftszeiten können die einzelnen

	$\alpha = 1$	$\alpha = 2$	$\alpha = 3$
Init Loop	1.5	23	39
Traverse Routes Loop	22	553	1340
Update MEATs Loop	1.0	186	579

Tabelle 5.3: Die Ausführungszeiten (in ms) der drei Schleifen des RAPTOR MEAT Algorithmus für alle drei Werte von α. Verwendet wurde das erste Verspätungsmodell mit maximal 15 bzw. 30 Minuten Verspätung.

	$\alpha = 1$	$\alpha = 2$	$\alpha = 3$
Init Loop	2.3	39	79
Traverse Routes Loop	40	997	3379
Update MEATs Loop	2.1	467	1674

Tabelle 5.4: Die Ausführungszeiten (in ms) der drei Schleifen des RAPTOR MEAT Algorithmus für alle drei Werte von α. Verwendet wurde das zweite Verspätungsmodell mit maximal 60 Minuten Verspätung.

Threads die Operation des Zusammenfügens der Beutel unabhängig für die Stopps durchführen. Somit ist auch dieser Teil parallelisierbar. Da diese beiden Bestandteile den größten Teil der Ausführungszeit des RAPTOR MEAT Algorithmus ausmachen, kann die Parallelisierung die gesamte Laufzeit des Algorithmus deutlich verringern. Im Vergleich hierzu ist dies bei dem CSA MEAT nicht möglich, da über die Verbindungen in einer festen Reihenfolge iteriert werden muss und sie daher nicht in unabhängige Teilmengen aufgeteilt werden können. Bei der Standardvariante des McRAPTOR Algorithmus, auf dem unser RAPTOR MEAT Algorithmus basiert, konnte in [DPW15] durch eine Parallelisierung mit acht Kernen die Ausführungszeit um einen Faktor von bis zu fünf verkleinert werden. Sollte dies für unseren Algorithmus auch erreicht werden, würde er für $\alpha = 2$ in dem ersten Verspätungsmodell ähnlich lange wie seine CSA Alternative benötigen. Im zweiten Verspätungsmodell würde sich seine Laufzeit der des CSA annähern. Bei beiden Modellen könnte man zusätzlich beobachten, dass die Laufzeit für $\alpha = 1$ des RAPTOR Algorithmus noch deutlich geringer als die des CSA ist.

5.3 Qualitätstests

Nach der Durchführung der Benchmarktests analysieren wir zusätzlich die Qualität der berechneten minimalen erwarteten Ankunftszeiten. Zunächst betrachten wir die Unterschiede der Ergebnisse für die veränderten α-Werte zur Beschränkung der berücksichtigten Verbindungen und Trips. Anschließend vergleichen wir die Ergebnisse der MEAT Algorithmen mit den erwarteten Ankunftszeiten, falls wir uns bei jeder Entscheidung für die schnellste Reise entscheiden. Zudem nähern wir die minimalen erwarteten Ankunftszeiten an, indem wir Fahrten, die die Lösung der Entscheidungsgraphen berücksichtigen, inklusive möglicher Verspätungen simulieren. Wir vergleichen daraufhin das Ergebnis der Entscheidungsgraphen mit der Berechnung kürzester Ankunftszeiten, wenn Verspätungen schon vor dem Antritt der Fahrt bekannt sind. Abschließend analysieren wir mögliche Vorteile, die uns die Transferoptimierungs- und Transferbeschränkungsvarianten des RAPTOR MEAT Algorithmus liefern.

Diese Tests führen wir für beide Verspätungsmodelle aus dem Kapitel 5.1 durch, um mögliche Einflüsse der maximalen Verspätung von Trips analysieren zu können.

5.3.1 Beschränkungstests

Bei den Benchmarktests im Kapitel 5.2 haben wir bereits den Einfluss der Beschränkungen der maximalen Ankunftszeit auf die Laufzeiten der Algorithmen gesehen. In diesem Abschnitt analysieren wir nun, inwiefern größere Werte für α die Genauigkeit der berechneten minimalen erwarteten Ankunftszeiten verbessern. Eine Verbesserung der minimalen erwarteten Ankunftszeit ist möglich, da durch größere maximale Ankunftszeiten mehr Verbindungen bzw. Trips betrachtet werden, die potenziell in Alternativreisen verwendet werden können. Deren Existenz kann dazu führen, dass die erwartete Ankunftszeit von Reisen, die bisher ohne vollständige Alternative unberücksichtigt blieben, berechnet werden kann. Die Nichtberücksichtigung von Reisen ohne vollständige Alternative wird durch die Definition der Entscheidungsgraphen vorgegeben. So besagt diese, dass jeder Streckenabschnitt des Graphen eine sichere Reise zum Zielstopp benötigt, über die ein Passagier dort vor der maximalen Ankunftszeit ankommen kann. Dies ist für die Streckenabschnitte einer Reise ohne vollständige Alternative nicht gegeben. Da die erwarteten Ankunftszeiten der bisherigen nicht beachteten Reisen kleiner sein kann als die aktuellen minimalen erwarteten Ankunftszeiten, ist es möglich, dass für größere Beschränkungen noch kleinere erwartete Ankunftszeiten gefunden werden. Dies kann so lange mit immer größeren Beschränkungen fortgeführt werden, bis alle Verbindungen, die auf die minimale Abfahrtszeit folgen, überprüft werden. Da dies in unserem Fall eine unbeschränkte Anzahl an Verbindungen wäre, benötigen wir in der Praxis eine obere Schranke und somit einen Wert für α.

Wir betrachten zunächst die Ergebnisse für das erste Verspätungsmodell. In der Tabelle 5.5 sind die Differenzen der minimalen erwarteten Ankunftszeiten für $\alpha = 1$ im Vergleich zu den minimalen erwarteten Ankunftszeiten der Berechnungen für $\alpha = 2$ und $\alpha = 3$ jeweils in Sekunden dargestellt. Die relativen Differenzen beziehen sich auf die Zeitspanne zwischen frühester Abfahrtszeit und der minimalen erwarteten Ankunftszeit für $\alpha = 1$.

An den absoluten Differenzen lässt sich erkennen, dass die minimale erwartete Ankunftszeit für $\alpha = 2$ im Durchschnitt ungefähr 6 Minuten kleiner als die minimale erwartete Ankunftszeit für $\alpha = 1$ ist. Somit könnte es sich für viele Nutzer des Algorithmus lohnen, trotz der etwas längeren Ausführungszeiten die Variante dieser Beschränkung zu nutzen. Vor allem im schlechtesten Fall ist das gegeben, da die maximale absolute Differenz etwa zehn Stunden beträgt.

Dagegen ist der Vorteil, den man erhält, wenn man als Schranke $\alpha = 3$ statt $\alpha = 2$ wählt, deutlich geringer. So beträgt hier die durchschnittliche absolute Differenz in der Tabelle 5.6 lediglich 0.03s. Dies lässt darauf schließen, dass in den meisten Fällen schon für die geringere maximale Ankunftszeit alle relevanten Trips bzw. Verbindungen bekannt sind. Da sogar im schlechtesten Fall unserer Tests die absolute Differenz lediglich bei 30s liegt, lohnt es sich in den wenigsten Fällen den Algorithmus mit $\alpha = 3$ auszuführen.

Relativ gesehen ist selbst bei den Differenzen zu $\alpha = 1$ in Tabelle 5.5 der Einfluss gering. Die relative Differenz beträgt hier durchschnittlich lediglich 1.15% der minimalen erwarteten Fahrtzeit für $\alpha = 1$ und weist somit auf die gesamte benötigte Zeit keine allzu große Bedeutung auf.

		$\alpha = 2$	$\alpha = 3$
Absolute	Avg	369	369
	Max	36967	36967
Relative	Avg	1.15%	1.15%
	Max	42.9%	42.9%

Tabelle 5.5: Die Unterschiede (in s) von den MEAT-Ergebnissen mit einer Beschränkung von $\alpha = 1$ und den Ergebnissen für $\alpha = 2$ und $\alpha = 3$. Verwendet wurde das erste Verspätungsmodell mit maximal 15 bzw. 30 Minuten Verspätung.

		$\alpha = 3$
Absolute	Avg	0.03
	Max	30
Relative	Avg	0.00009%
	Max	0.2%

Tabelle 5.6: Die Unterschiede (in s) von den MEAT-Ergebnissen mit einer Beschränkung von $\alpha = 2$ und den Ergebnissen für $\alpha = 3$. Verwendet wurde das erste Verspätungsmodell mit maximal 15 bzw. 30 Minuten Verspätung.

In den Tabellen 5.7 und 5.8 sind die jeweiligen Differenzen für das zweite Verspätungsmodell dargestellt. Auch hier kann man erkennen, dass vor allem die absolute Differenz der erwarteten Ankunftszeiten für $\alpha = 2$ und $\alpha = 3$ von den Werten für $\alpha = 1$ abweicht.

5.3.2 Relevanztests

Um bestimmen zu können, welche Relevanz die Berechnung der minimalen erwarteten Ankunftszeiten hat, vergleichen wir sie mit den erwarteten Ankunftszeiten der schnellsten Reisen. Diese Werte berechnen wir mit dem CSA ExpAT, der auch als Alternativreisen jeweils die Reisen mit den frühesten Ankunftszeiten auswählt und daraus die erwartete Ankunftszeit für den entstehenden Entscheidungsgraphen berechnet. Dieser Wert stellt die Ankunftszeit dar, die ein Passagier unter Berücksichtigung unseres Verspätungsmodells erwarten muss, wenn er sich bei jeder Entscheidung lediglich nach der minimalen Ankunftszeit richtet. Im Gegensatz dazu liefert der RAPTOR MEAT Algorithmus eine mindestens genauso gute erwartete Ankunftszeit, da er sie minimiert. Somit sind

		$\alpha = 2$	$\alpha = 3$
Absolute	Avg	454	454
	Max	37181	37181
Relative	Avg	1.4%	1.4%
	Max	60.7%	60.7%

Tabelle 5.7: Die Unterschiede (in s) von den MEAT-Ergebnissen mit einer Beschränkung von $\alpha = 1$ und den Ergebnissen für $\alpha = 2$ und $\alpha = 3$. Verwendet wurde das zweite Verspätungsmodell mit maximal 60 Minuten Verspätung.

		$\alpha = 3$
Absolute	Avg	0.33
	Max	284
Relative	Avg	0.001%
	Max	1.4%

Tabelle 5.8: Die Unterschiede (in s) von den MEAT-Ergebnissen mit einer Beschränkung von $\alpha = 2$ und den Ergebnissen für $\alpha = 3$. Verwendet wurde das zweite Verspätungsmodell mit maximal 60 Minuten Verspätung.

zwar hier potenziell die schnellsten Reisen in dem Entscheidungsgraphen nicht enthalten, aber der Passagier kann unter Annahme unseres Verspätungsmodells erwarten, dass er mindestens genauso früh ankommt wie ein Passagier, der sich nach dem CSA ExpAT Ergebnis richtet.

Die Differenz aus den beiden Möglichkeiten zeigt nun, inwiefern sich die erwartete Ankunftszeit des CSA ExpAT von der minimalen erwarteten Ankunftszeit unterscheidet. Wir analysieren dabei zunächst die Ergebnisse des ersten Verspätungsmodells. Deren Differenzen sind in der Tabelle 5.11 in Sekunden angegeben. Die relative Differenz bezieht sich auf die Zeitspanne zwischen der frühesten Abfahrtszeit am Startstopp und der minimalen erwarteten Ankunftszeit am Zielstopp.

Für die berechneten erwarteten Ankunftszeiten und den zugehörigen Entscheidungsgraphen des CSA ExpAT muss beachtet werden, dass durch die oberen Schranken der Ankunftszeit nicht garantiert ist, dass es zu einer vollständigen Abdeckung der Reisen des Graphen durch Alternativreisen kommt. Dies ist lediglich für die Graphen der minimalen erwarteten Ankunftszeiten gewährleistet. Die erwartete Ankunftszeit kann für unvollständige Graphen nicht berechnet werden. Wie viele der 1000 Anfragen jeweils in unvollständigen Graphen resultiert sind, zeigt die Tabelle 5.12. Wie man erkennen kann, liefert der Vergleich der erwarteten Ankunftszeiten für $\alpha = 1$ wenig aussagekräftige Resultate, da etwa 85% aller Anfragen unvollständige Entscheidungsgraphen als Ergebnis besitzen. Für $\alpha = 2$ und $\alpha = 3$ sinken diese Werte deutlich, da durch die erhöhten Schranken mehr Trips und die daraus entstehenden potenziellen Alternativreisen berücksichtigt werden. Somit sind die Differenzen, die für diese Werte von α gebildet werden, aussagekräftiger und der Vergleich der Ergebnisse der beiden Algorithmen möglich.

Hierbei fällt auf, dass für $\alpha = 2$ und $\alpha = 3$ die absolute Differenz im Durchschnitt ungefähr 1200s beträgt. Dies entspricht 20 Minuten und stellt für Passagiere die Zeit dar, die sie erwarten müssen durchschnittlich später am Ziel anzukommen, falls sie ihre Reise ausschließlich unter Berücksichtigung der frühesten Ankunftszeiten planen. Auch im Hinblick auf die gesamte Fahrtzeit stellt es einen Vorteil dar, wenn sie sich stattdessen nach den berechneten minimalen erwarteten Ankunftszeiten richten. So beträgt die relative Differenz zu der minimalen erwarteten Fahrtzeit ca. 4%, was vor allem für die erwarteten Ankunftszeiten längerer Reisen eine Auswirkung darstellt.

Der absolute Maximalwert der Differenz für $\alpha = 2$ und $\alpha = 3$ von jeweils über 8 Stunden zeigt das Verhalten im schlechtesten Fall der Verwendung des CSA ExpAT. Insgesamt lässt sich aus den Relevanztests somit schließen, dass es für Passagiere einen deutlichen Vorteil bietet, wenn sie sich bei ihrer Routenplanung an den Ergebnissen der MEAT Algorithmen und nicht ausschließlich an den frühesten Ankunftszeiten orientieren.

		$\alpha = 1$	$\alpha = 2$	$\alpha = 3$
Absolute	Avg	196	1131	1246
	Max	2295	30213	30213
Relative	Avg	0.7%	3.6%	4.0%
	Max	23%	62%	62%

Tabelle 5.9: Die Differenzen (in s) zwischen erwarteter (ExpAT) und minimal erwarteter Ankunftszeit (MEAT) für alle drei Werte von α. Verwendet wurde das erste Verspätungsmodell mit maximal 15 bzw. 30 Minuten Verspätung.

	$\alpha = 1$	$\alpha = 2$	$\alpha = 3$
Absolute	855	104	28
Relative	85.5%	10.4%	2.8%

Tabelle 5.10: Die Anzahl der unvollständigen Entscheidungsgraphen, die aus den 1000 Anfragen an den CSA ExpAT für alle drei Werte von α resultieren. Verwendet wurde das erste Verspätungsmodell mit maximal 15 bzw. 30 Minuten Verspätung.

Bei dem zweiten Verspätungsmodell ergeben sich in den Tabellen 5.11 und 5.12 ähnliche Resultate wie bei dem ersten Modell. Auffällig ist hierbei die maximale relative Differenz für $\alpha = 2$ und $\alpha = 3$. Diese beträgt 97% und somit ist in diesem Fall der Unterschied zwischen der erwarteten und minimalen erwarteten Ankunftszeit fast so groß wie die kürzeste erwartete Fahrtzeit.

5.3.3 Annäherungstests

Das Konzept der erwarteten Ankunftszeiten soll uns die durchschnittliche Ankunftszeit am Zielstopp liefern, die wir erhalten, wenn wir uns mehrmalig zur minimalen Abfahrtszeit am Startstopp befinden und zum Zielstopp reisen. Dabei verwenden wir abhängig von den Verspätungen der ausgewählten Trips die Alternativreisen, die uns durch den Entscheidungsgraphen vorgegeben werden. Um dieses Konzept der erwarteten Ankunftszeiten zu simulieren und somit die erwarteten Ankunftszeiten des zugehörigen Entscheidungsgraphen zu überprüfen, führen wir Annäherungstests durch.

Dabei verwenden wir als Entscheidungsgraphen die Graphen der 1000 Anfragen der bisherigen Tests. Für jeden dieser Graphen simulieren wir 10 Millionen Reisen zwischen seinem Start- und Zielstopp beginnend mit der gegebenen frühesten Abfahrtszeit. Um die Wahrscheinlichkeiten für

		$\alpha = 1$	$\alpha = 2$	$\alpha = 3$
Absolute	Avg	349	1147	1272
	Max	3941	20043	28940
Relative	Avg	1.4%	3.7%	4.1%
	Max	19%	52%	97%

Tabelle 5.11: Die Differenzen (in s) zwischen erwarteter (ExpAT) und minimal erwarteter Ankunftszeit (MEAT) für alle drei Werte von α. Verwendet wurde das zweite Verspätungsmodell mit maximal 60 Minuten Verspätung.

	$\alpha = 1$	$\alpha = 2$	$\alpha = 3$
Absolute	868	149	37
Relative	86.8%	14.9%	3.7%

Tabelle 5.12: Die Anzahl der unvollständigen Entscheidungsgraphen, die aus den 1000 Anfragen an den CSA ExpAT für alle drei Werte von α resultieren. Verwendet wurde das zweite Verspätungsmodell mit maximal 60 Minuten Verspätung.

		$\alpha = 1$	$\alpha = 2$	$\alpha = 3$
Delay Model 1	Abs	0.33	0.33	0.33
	Rel	0.001%	0.001%	0.001%
Delay Model 2	Abs	0.36	0.36	0.35
	Rel	0.0011%	0.0011%	0.0011%

Tabelle 5.13: Die Differenzen (in s) der erwarteten und approximierten Ankunftszeiten für alle drei Werte von α und beide Verspätungsmodelle.

Verspätungen auf der Reise zu erhalten, nutzen wir unsere Modelle aus dem Kapitel 5.1. Somit berechnen wir für jeden genommenen Trip die Verspätung, die er in diesem Durchlauf hat und steigen abhängig davon in die vorgeschlagene Alternativreise um. Aus jedem Durchlauf resultiert eine Ankunftszeit am Zielstopp. Aus diesen Zeiten berechnen wir die durchschnittliche Ankunftszeit der 10 Millionen Reisen.

In der Tabelle 5.13 sind die Beträge der Differenzen der approximierten und berechneten minimalen erwarteten Ankunftszeiten in Sekunden dargestellt. Man kann an ihnen erkennen, dass sich die approximierten Ankunftszeiten für 10 Millionen Durchläufe im Durchschnitt auf ca. 0.3 Sekunden an die berechneten erwarteten Ankunftszeiten angenähert haben. Würde man die Anzahl der Durchläufe noch weiter erhöhen, könnte man über die Approximationen noch genauere Ergebnisse erzielen. Jedoch zeigen auch schon die Resultate für 10 Millionen simulierte Reisen, dass die berechneten minimalen erwarteten Ankunftszeiten die korrekten Erwartungswerte der zugehörigen Entscheidungsgraphen darstellen. Relativ gesehen stellt die Differenz lediglich etwa 0.001% der kürzesten erwarteten Fahrtzeit dar.

5.3.4 Verspätungstests

Bei den Verspätungstests analysieren wir die Ergebnisse des RAPTOR MEAT Algorithmus, wenn die Verspätungen der Trips schon vor dem Antritt der Reise bekannt sind. Dazu berechnen wir mit dem Standard CSA die früheste Ankunftszeit, zu der wir mit diesen Verspätungen den Zielstopp erreichen können. Anschließend verwenden wir die Lösung des MEAT Problems und ermitteln die Ankunftszeit, die wir bei den Verspätungen auf den vorgegebenen Reisen erhalten. Dieser Wert ist gleich groß oder größer als die früheste Ankunftszeit des Standard CSA. In der Realität spielt dieser Anwendungsfall kaum eine Rolle, da in den wenigsten Fällen die genauen Verspätungen der Trips schon vor dem Antritt der Reise bekannt sind. Dennoch liefern die Tests uns einen Anhaltspunkt, wie gut das Konzept der minimalen erwarteten Ankunftszeiten selbst bei bekannten Verspätungen funktioniert.

		$\alpha = 1$	$\alpha = 2$	$\alpha = 3$
Absolute	Avg	453	98	131
	Max	43620	7200	27960
Relative	Avg	1.4%	0.3%	0.4%
	Max	129%	42%	109%

Tabelle 5.14: Die Differenzen (in s) der Ankunftszeiten bei bekannter Verspätung für alle drei Werte von α. Der Vergleich erfolgt zwischen der frühesten Ankunftszeit des CSA und dem Entscheidungsgraphen, der aus dem MEAT Problem resultiert. Verwendet wurde das erste Verspätungsmodell mit maximal 15 bzw. 30 Minuten Verspätung.

	$\alpha = 1$	$\alpha = 2$	$\alpha = 3$
Absolute	855	925	929
Relative	85.5%	92.5%	92.9%

Tabelle 5.15: Die Anzahl und der Anteil der Anfragen bei denen die Ergebnisse des CSA und RAPTOR MEAT Algorithmus bei bekannten Verspätungen übereinstimmen. Verwendet wurde das erste Verspätungsmodell mit maximal 15 bzw. 30 Minuten Verspätung.

In der Tabelle 5.14 sind die Differenzen zwischen der frühest möglichen Ankunftszeit bei bekannten Verspätungen und die Zeit, die man über die Lösung des MEAT Problems erhält, für das erste Verspätungsmodell angegeben. Die relative Differenz bezieht sich auf die kürzeste Fahrtzeit für diese Verspätungen, die mit dem CSA berechnet wurde.

Die durchschnittliche absolute Differenz der Ankunftszeiten bei bekannten Verspätungen liegt für $\alpha = 1$ bei 453s. Wie schon bei den Vergleichen der Ergebnisse für die unterschiedlichen oberen Beschränkungen der Ankunftszeiten in 5.3.1, lässt sich auch hier ein Einfluss der Werte von α erkennen. So liegt die durchschnittliche Differenz für $\alpha = 2$ und $\alpha = 3$ ungefähr bei zwei Minuten. Dies hat auch relativ gesehen mit 0.3% bzw. 0.4% nur einen kleinen Einfluss auf die Ankunftszeit des Passagiers. Somit lässt sich erkennen, dass durchschnittlich gesehen das Ergebnis des RAPTOR MEAT Algorithmus selbst bei bekannten Verspätungen sehr gute Ergebnisse liefert. Auffällig ist hierbei, dass die durchschnittlichen und maximalen Differenzen für $\alpha = 3$ größer als für $\alpha = 2$ sind. Dies ist möglich, da wir mit unserem Algorithmus lediglich die erwartete Ankunftszeit minimieren. Durch die unterschiedlichen Intervalle an Ankunftszeiten für die Werte von α ist zwar gesichert, dass die minimale erwartete Ankunftszeit von $\alpha = 3$ nicht größer als die von $\alpha = 2$ ist, jedoch können sie in unterschiedlichen Entscheidungsgraphen resultieren. Dies kann bei den festen Verspätungen dieses Tests dazu führen, dass wir für $\alpha = 3$ schlechtere Ergebnisse als für $\alpha = 2$ erhalten.

Die maximale absolute Differenz liegt für $\alpha = 1$ bei knapp zehn, für $\alpha = 2$ bei etwa zwei und für $\alpha = 3$ bei sieben Stunden. Dass es sich hierbei lediglich um einige wenige Ausreißer handelt, zeigt die Tabelle 5.15. In ihr ist der Anteil der Testanfragen dargestellt, bei denen die Resultate des CSA und RAPTOR MEAT Algorithmus für die bekannten Verspätungen übereinstimmen. Für $\alpha = 1$ liegt dieser Anteil dabei bei ungefähr 85% und für $\alpha = 2$ und $\alpha = 3$ sogar bei über 90%.

Im Vergleich der Ergebnisse mit den Resultaten des zweiten Verspätungsmodells in den Tabellen 5.16 und 5.17 fällt auf, dass auch hier für $\alpha = 2$ und $\alpha = 3$ die durchschnittliche absolute Differenz bei etwa zwei Minuten liegt. Der Anteil der übereinstimmenden Ergebnisse liegt erneut bei über 90%.

		$\alpha = 1$	$\alpha = 2$	$\alpha = 3$
Absolute	Avg	615	114	98
	Max	43620	27720	7680
Relative	Avg	2.0%	0.6%	0.3%
	Max	190%	101%	34%

Tabelle 5.16: Die Differenzen (in s) der Ankunftszeiten bei bekannter Verspätung für alle drei Werte von α. Der Vergleich erfolgt zwischen der frühesten Ankunftszeit des CSA und dem Entscheidungsgraphen, der aus dem MEAT Problem resultiert. Verwendet wurde das zweite Verspätungsmodell mit maximal 60 Minuten Verspätung.

	$\alpha = 1$	$\alpha = 2$	$\alpha = 3$
Absolute	858	931	922
Relative	85.8%	93.1%	92.2%

Tabelle 5.17: Die Anzahl und der Anteil der Anfragen bei denen die Ergebnisse des CSA und RAPTOR MEAT Algorithmus bei bekannten Verspätungen übereinstimmen. Verwendet wurde das zweite Verspätungsmodell mit maximal 60 Minuten Verspätung.

5.3.5 Transferoptimierungstests

Als nächstes analysieren wir die Größe der Entscheidungsgraphen, die aus der Berechnung der minimalen erwarteten Ankunftszeiten mit dem RAPTOR MEAT Algorithmus resultieren. Dazu zählen wir für jeden Graphen die Anzahl seiner Stopps (Stops) und der Streckenabschnitte (Legs), die die Knoten des erweiterten Entscheidungsgraphen verbinden. Als zusätzlichen Wert bestimmen wir die Anzahl der Kanten (Edges), die die kompakte Variante des Graphen enthält. Von allen drei Werten ermitteln wir jeweils den Durchschnitt und das Maximum aller Entscheidungsgraphen der 1000 zufälligen Anfragen. Wir wollen mit diesen Tests zusätzlich die Ergebnisse des RAPTOR MEAT Algorithmus mit seiner Variante zur Transferoptimierung vergleichen, daher sind in der Tabelle 5.18 auch die jeweiligen Werte der Entscheidungsgraphen des RAPTOR MEAT TO Algorithmus dargestellt. In dieser Tabelle befinden sich die Werte für unser erstes Verspätungsmodell. Zum Vergleich sind die Werte des zweiten Verspätungsmodells in der Tabelle 5.19 dargestellt.

Zunächst kann man an den Durchschnittswerten den Vorteil der kompakten Darstellung des Graphen erkennen, da die Anzahl seiner Kanten jeweils geringer als die Anzahl der Streckenabschnitte des erweiterten Graphen ist. Somit sorgt die kompakte Darstellung dafür, dass sich der Nutzer einen besseren Überblick über seine Reise verschaffen kann. Er sieht zwar nicht die genauen Abfahrtszeiten der Züge, aber kann erkennen zwischen welchen Stopps er reisen muss. Auch während der Reise ist dies für die meisten Passagiere ausreichend, da es bei Umstiegen hauptsächlich darauf ankommt zu wissen, zu welchem Bahnhof der Anschlusszug fahren soll. Die Abfahrtszeit von diesem ergibt sich dann aus der Ankunftszeit des Passagiers an dem Bahnhof des Umstiegs.

Auch bei der Betrachtung der Entscheidungsgraphen kann man einen Einfluss des Beschränkungsparameters α erkennen. Besonders bei den Maximalwerten des RAPTOR MEAT Algorithmus zeigt sich, dass für $\alpha = 2$ und $\alpha = 3$ mehr Trips bzw. Verbindungen betrachtet und längere Reisen zwischen dem Start- und Zielstopp gefunden werden können. Somit besitzen auch potenziell größere Entscheidungsgraphen die minimale erwartete Ankunftszeit.

		$\alpha = 1$			$\alpha = 2$			$\alpha = 3$		
		Stops	Legs	Edges	Stops	Legs	Edges	Stops	Legs	Edges
RAPTOR	Avg	7	12	8	8	16	9	8	15	9
MEAT	Max	21	77	34	27	96	43	27	96	43
RAPTOR	Avg	6	10	6	6	10	6	6	10	6
MEAT TO	Max	17	65	24	22	96	34	22	96	34

Tabelle 5.18: Die Anzahl der Stopps, Streckenabschnitte und Kanten der Entscheidungsgraphen für alle drei Werte von α. Grundlage sind die Entscheidungsgraphen von dem RAPTOR MEAT Algorithmus und seiner TO-Variante. Verwendet wurde das erste Verspätungsmodell mit maximal 15 bzw. 30 Minuten Verspätung.

		$\alpha = 1$			$\alpha = 2$			$\alpha = 3$		
		Stops	Legs	Edges	Stops	Legs	Edges	Stops	Legs	Edges
RAPTOR	Avg	9	29	12	14	95	30	15	111	34
MEAT	Max	51	297	141	84	3272	590	107	3717	590
RAPTOR	Avg	7	19	8	7	21	8	7	21	8
MEAT TO	Max	26	168	43	25	317	54	25	317	54

Tabelle 5.19: Die Anzahl der Stopps, Streckenabschnitte und Kanten der Entscheidungsgraphen für alle drei Werte von α. Grundlage sind die Entscheidungsgraphen von dem RAPTOR MEAT Algorithmus und seiner TO-Variante. Verwendet wurde das zweite Verspätungsmodell mit maximal 60 Minuten Verspätung.

Die Entscheidungsgraphen des RAPTOR MEAT TO Algorithmus besitzen im Durchschnitt für das erste Verspätungsmodell 6 statt 7 bzw. 8 Stopps für die unterschiedlichen Werte von α im Vergleich zu dem RAPTOR MEAT Algorithmus. Diesem Unterschied liegt zugrunde, dass bei der Variante Graphen mit einer geringeren maximalen Anzahl an Umstiegen ausgewählt werden, falls sich die dadurch auftretende Differenz der erwarteten Ankunftszeiten unter einer vorher definierten Grenze befindet. Eine weitere Folge ist, dass dadurch im Durchschnitt auch die Anzahl der Streckenabschnitte und Kanten kleiner ist. Diese Variante liefert somit übersichtlichere Ergebnisse, was auch die Maximalwerte der Entscheidungsgraphen zeigen.

Im Vergleich zu der Tabelle 5.19 des zweiten Verspätungsmodell zeigt sich der Einfluss der maximalen Verspätung von Trips. Da diese im zweiten Modell 60 statt 15 bzw. 30 Minuten beträgt, werden an Umstiegspunkten mehr Trips als Alternative für mögliche Verspätungen benötigt. Dies resultiert in den deutlich größeren Graphen, die in den Durchschnitts- und Maximalwerten sichtbar werden. Auffällig ist hier die maximale Anzahl an Streckenabschnitten. Die zugehörigen Graphen umfassen für $\alpha = 2$ und $\alpha = 3$ bis zu 3272 bzw. 3717 Streckenabschnitte.

Um herauszufinden welche Vorteile die RAPTOR MEAT TO Variante liefern kann, vergleichen wir ihre erwarteten Ankunftszeiten und ihre maximale Anzahl an Umstiegen mit den zugehörigen Werten des RAPTOR MEAT Algorithmus. Dabei ist die erwartete Ankunftszeit des RAPTOR MEAT TO Algorithmus mindestens so groß wie die der klassischen Variante. Im Gegensatz dazu

ist die Anzahl an Umstiegen, die der Nutzer auf seiner Reise zwischen Start- und Zielstopp maximal nehmen muss kleiner oder gleich groß wie die des RAPTOR MEAT Algorithmus. Die absoluten und relativen Differenzen in Bezug auf die minimalen erwarteten Ankunftszeiten sind in Tabelle 5.20 für das erste Verspätungsmodell dargestellt. Die relative Differenz der Anzahl an Umstiegen bezieht sich dabei auf die maximale Anzahl an Umstiegen, die der Passagier bei dem Entscheidungsgraphen des RAPTOR MEAT Ergebnis nehmen muss.

Dabei ist für $\alpha = 2$ und $\alpha = 3$ die erwartete Ankunftszeit der Transferoptimierungsvariante im Durchschnitt ca. 100s später als die minimale erwartete Ankunftszeit. Dafür ist die maximale Anzahl an Umstiegen des Passagiers durchschnittlich um einen Umstieg geringer. Durch die kleineren Optionen an Trips, die für $\alpha = 1$ zur Verfügung stehen, wird hier die Anzahl an Umstiegen im Schnitt lediglich nur um 0.5 verringert. Jedoch liegt hier die durchschnittliche absolute Differenz der erwarteten Ankunftszeiten auch nur bei 43s.

Die relativen Differenzen der Anzahl der Umstiege zeigen, dass für $\alpha = 2$ und $\alpha = 3$ die maximale Anzahl an Umstiegen durch die Variante durchschnittlich um ungefähr 24% gesenkt werden können. Dies kann für Passagiere ein Vorteil sein, wenn sie auf Kosten einer etwas späteren erwarteten Ankunftszeit die Variante mit weniger Umstiegen wählen, da diese für sie eine einfachere Reise darstellt. Auch bei den Maximalwerten lohnt es sich das Ergebnis der RAPTOR MEAT TO Variante zu verwenden, da hier die Anzahl an Umstiegen um 10 bzw. 12 reduziert werden konnte. Jedoch muss man beachten, dass die erwartete Ankunftszeit um bis zu 30 Minuten erhöht wurde.

Die absoluten und relativen Unterschiede sind in der Tabelle 5.21 für das zweite Verspätungsmodell mit maximal 60 Minuten Verspätung dargestellt. Hierbei fällt auf, dass durch die späteren sicheren Ankunftszeiten und somit größeren Intervalle, in denen nach Trips gesucht werden kann, mehr Umstiege durch die Transferoptimierungsvariante vermieden werden können. Für $\alpha = 1$ sind dies im Durchschnitt 1.3 und für $\alpha = 2$ und $\alpha = 3$ sogar knapp 4 Umstiege. Dies entspricht relativ gesehen etwa 50% der maximalen Anzahl an Umstiegen. Jedoch muss der Passagier dafür durchschnittlich um 75s bis 201s spätere Ankunftszeiten erwarten. Maximal beträgt die Differenz der Anzahl der Umstiege sogar 109 und die erwarteten Ankunftszeiten besitzen einen Unterschied von bis zu 4 Stunden.

Insgesamt ist es für Nutzer am sinnvollsten, sowohl das Ergebnis des RAPTOR MEAT Algorithmus als auch den Entscheidungsgraphen der Transferoptmierungsvariante zu berechnen und sich dann je nach Präferenz und Unterschied der Graphen für einen der beiden zu entscheiden.

5.3.6 Transferbeschränkungstests

Die Transferbeschränkungsvariante des RAPTOR MEAT Algorithmus führt lediglich eine feste Anzahl $k_{\max}$ an Runden der Standardvariante des Algorithmus durch. Der zugehörige Entscheidungsgraph basiert auf der zu diesem Zeitpunkt bekannten minimalen erwarteten Ankunftszeit und seine Reisen bestehen aus maximal $k_{\max}$ Streckenabschnitten. Somit muss der Passagier auf seiner Reise vom Startstopp s zum Zielstopp t maximal $k_{\max} - 1$ Mal umsteigen.

In der Tabelle 5.22 ist die Anzahl der Runden dargestellt, die in der Standardvariante des Algorithmus in beiden Verspätungsmodellen durchgeführt wurden. Für $\alpha = 2$ werden dabei durchschnittlich knapp 13 Runden des Algorithmus für das erste Verspätungsmodell ausgeführt. Im zweiten Modell beträgt durch die erhöhte maximale Verspätung der Trips im Durchschnitt die Rundenanzahl 16. Das

		$\alpha = 1$		$\alpha = 2$		$\alpha = 3$	
		MEAT	Transfers	MEAT	Transfers	MEAT	Transfers
Absolute	Avg	43	0.5	99	1.1	99	1.2
	Max	1044	5	1796	10	1796	12
Relative	Avg	0.13%	12.4%	0.31%	24.0%	0.31%	24.0%
	Max	4.8%	62.5%	16.3%	83.3%	16.3%	83.3%

Tabelle 5.20: Die Differenzen zwischen RAPTOR MEAT und seiner TO-Variante für alle drei Werte von α. Der Vergleich erfolgt dabei über die Differenzen der minimalen erwarteten Ankunftszeiten und Anzahl der Umstiege. Verwendet wurde das erste Verspätungsmodell mit maximal 15 bzw. 30 Minuten Verspätung.

		$\alpha = 1$		$\alpha = 2$		$\alpha = 3$	
		MEAT	Transfers	MEAT	Transfers	MEAT	Transfers
Absolute	Avg	75	1.3	200	3.7	201	4.0
	Max	1784	9	15120	109	15120	109
Relative	Avg	0.23%	26.6%	0.63%	51.0%	0.64%	52.9%
	Max	8.82%	75.0%	21.4%	98.0%	21.4%	98.0%

Tabelle 5.21: Die Differenzen zwischen RAPTOR MEAT und seiner TO-Variante für alle drei Werte von α. Der Vergleich erfolgt dabei über die Differenzen der minimalen erwarteten Ankunftszeiten und Anzahl der Umstiege. Verwendet wurde das zweite Verspätungsmodell mit maximal 60 Minuten Verspätung.

endgültige Ergebnis der minimalen erwarteten Ankunftszeit wird dabei aber schon durchschnittlich nach knapp 6 bzw. 8 Runden gefunden. Das Ausführen der darauffolgenden Runden sorgt somit für keine Verbesserung des Ergebnisses. Mit den Tests zu den Beschränkungstests wollen wir nun überprüfen, inwiefern diese Eigenschaft genutzt werden kann.

Die Beschränkungsvariante des RAPTOR MEAT Algorithmus führen wir für alle Parameter $1 \leq k_{\max} \leq 10$ auf allen 1000 Anfragen unserer Tests durch. Da laut den Tests in Kapitel 5.3.1 die minimalen erwarteten Ankunftszeiten in der Praxis für $\alpha = 2$ berechnet werden sollten, analysieren wir die Beschränkungstests für diesen Parameter.

Wir vergleichen dabei zunächst die Ausführungszeiten dieser Variante mit denen des normalen RAPTOR MEAT Algorithmus. Die Zeiten des RAPTOR MEAT Algorithmus sind in den Tabellen 5.1 und 5.2 der Benchmarktests in den Spalten „Algorithm“ aufgelistet. Sie umfassen die Ausführungszeit aller Runden, die innerhalb des RAPTOR MEAT Algorithmus durchgeführt wurden. In der Abbildung 5.2 sind die durchschnittlichen relativen Laufzeiten der eigentlichen Algorithmen der einzelnen Runden des RAPTOR MEAT TL Algorithmus im Vergleich zu der Laufzeit der Standardvariante für beide Verspätungsmodelle dargestellt. Hierbei ist erkennbar, dass jeweils die Laufzeit steigt, umso mehr Runden ausgeführt werden. Jedoch liegt die Laufzeit für alle enthaltenen

		$\alpha = 1$		$\alpha = 2$		$\alpha = 3$	
		Computed Rounds	Rounds of Result	Computed Rounds	Rounds of Result	Computed Rounds	Rounds of Result
Delay Model 1	Avg	6.8	5.2	12.7	5.8	15.0	5.8
	Max	16	11	23	14	23	16
Delay Model 2	Avg	7.4	5.9	16.3	8.2	22.3	8.5
	Max	20	14	114	113	172	113

Tabelle 5.22: Die Anzahl der Runden, die in der Standardvariante des RAPTOR MEAT Algorithmus durchgeführt wurden und nach denen das Ergebnis bekannt war. Dargestellt sind jeweils die durchschnittlichen und maximalen Werte für beide Verspätungsmodelle.

Werte von $k_{\max}$ unter der der Standardvariante. Dies lässt sich damit begründen, dass laut der Tabelle 5.22 in der normalen Variante des RAPTOR MEAT Algorithmus durchschnittlich knapp 13 Runden durchgeführt werden. Da für das zweite Verspätungsmodell durchschnittlich mehr Runden in der Standardvariante des Algorithmus benötigt werden, ist deren relative Ausführungszeit der ersten zehn Runden geringer.

Als nächstes analysieren wir pro Runde des RAPTOR MEAT TL Algorithmus bei wie vielen der 1000 Anfragen wir einen Wert für die minimale erwartete Ankunftszeit am Zielstopp t erhalten. Die zugehörigen relativen Werte sind in der Abbildung 5.4 dargestellt. Erkennbar ist dabei, dass in den ersten Runden nur für einen kleinen Anteil der Reisen ein Ergebnis berechnet werden kann, da die meisten Reisen zwischen Start- und Zielstopp aus mehr Streckenabschnitten bestehen. Jedoch existiert für beide Verspätungsmodelle bereits nach sechs Runden für alle Anfragen unserer Tests eine erwartete Ankunftszeit.

Um die Genauigkeit der berechneten erwarteten Ankunftszeiten nach einer bestimmten Anzahl an Runden zu ermitteln, betrachten wir die Differenz dieser zu der minimalen erwarteten Ankunftszeit des RAPTOR MEAT Algorithmus. Die relativen Differenzen im Bezug auf das MEAT Ergebnis sind in der Abbildung 5.4 dargestellt. Dabei lässt sich erkennen, dass für eine kleinere Anzahl an Runden die durchschnittliche Differenz bis zu 39% der minimalen erwarteten Ankunftszeit beträgt. Hierbei muss jedoch beachtet werden, dass nicht für alle Anfragen in diesen Runden bereits erwartete Ankunftzeiten berechnet werden können. Der Durchschnitt wurde somit in diesen Runden nicht über alle der 1000 Anfragen gebildet. Dementsprechend kann man aus der kleinen relativen Differenz nach einer Runde nicht darauf schließen, dass das für jede Ausführung mit $k_{\max} = 1$ gilt. Die Abbildung 5.4 zeigt, dass bei weniger als 1% der Anfragen in der ersten Runde schon eine erwartete Ankunftszeit berechnet werden konnte. In der sechsten Runde, in der im Gegensatz dazu für alle Anfragen bereits eine erwartete Ankunftszeit existiert, ist die durchschnittliche relative Differenz für beide verwendeten Verspätungsmodelle kleiner als 1%.

Insgesamt lässt sich durch die Tests der Beschränkungsvariante des RAPTOR MEAT Algorithmus erkennen, dass eine Ausführung von sechs Runden des eigentlichen Algorithmus für die meisten Anfragen sehr gute Ergebnisse liefert. Dies bietet den Vorteil, dass die verbleibenden Runden nicht

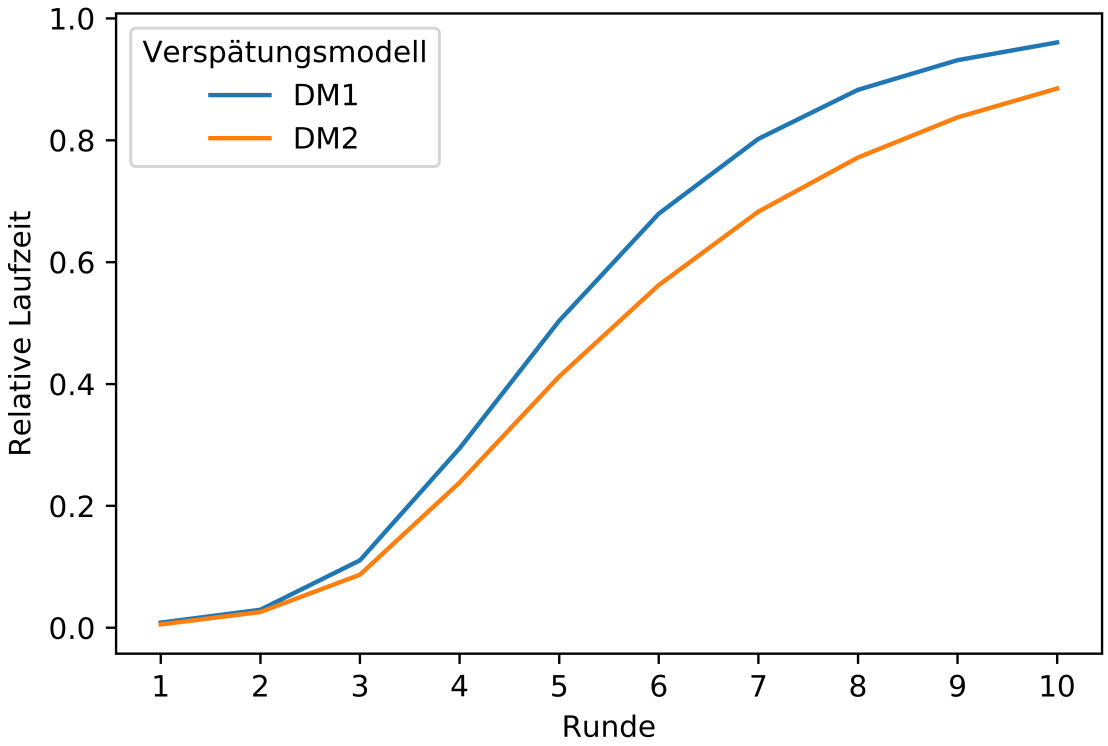

Abbildung 5.2: Die durchschnittlichen relativen Laufzeiten der Runden des eigentlichen Teils des RAPTOR MEAT TL Algorithmus im Vergleich zu seiner Standardvariante.

ausgeführt werden müssen und somit die Laufzeit des Algorithmus durchschnittlich um ungefähr 50% reduziert werden kann. Zudem ist für den Nutzer garantiert, dass er maximal fünf Umstiege auf seiner Reise vornehmen muss.

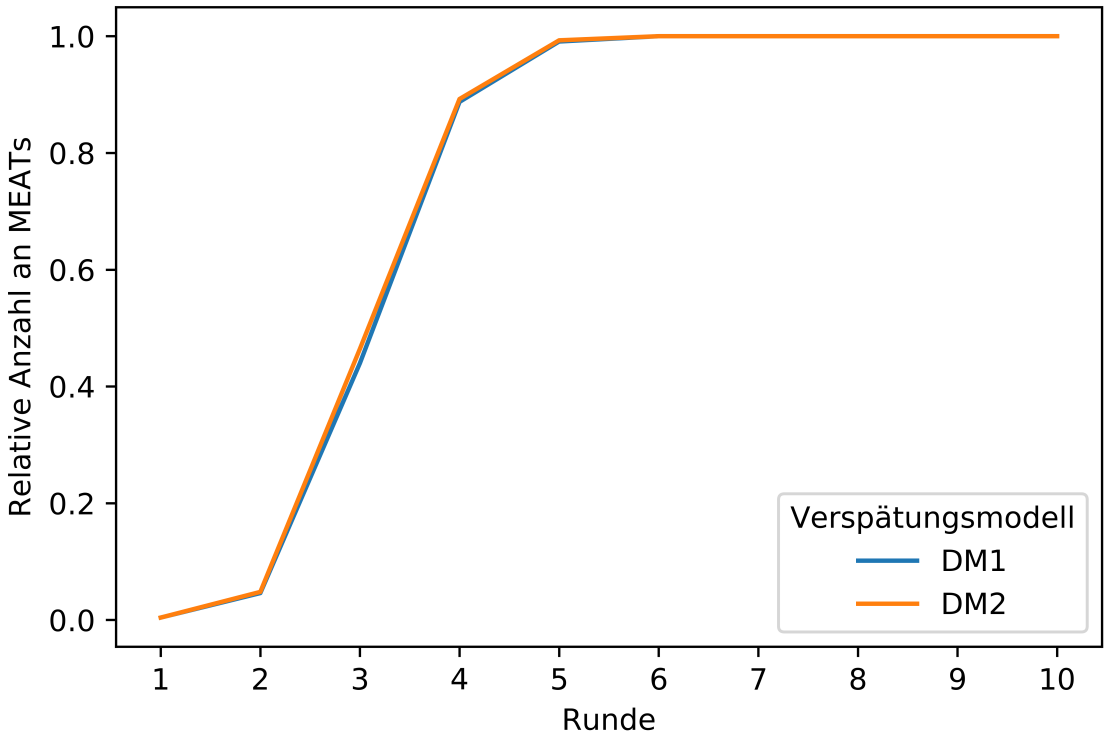

Abbildung 5.3: Die relative Anzahl an Anfragen pro Runde, für die mit dem RAPTOR MEAT TL Algorithmus bereits eine minimale erwartete Ankunftszeit berechnet werden konnte.

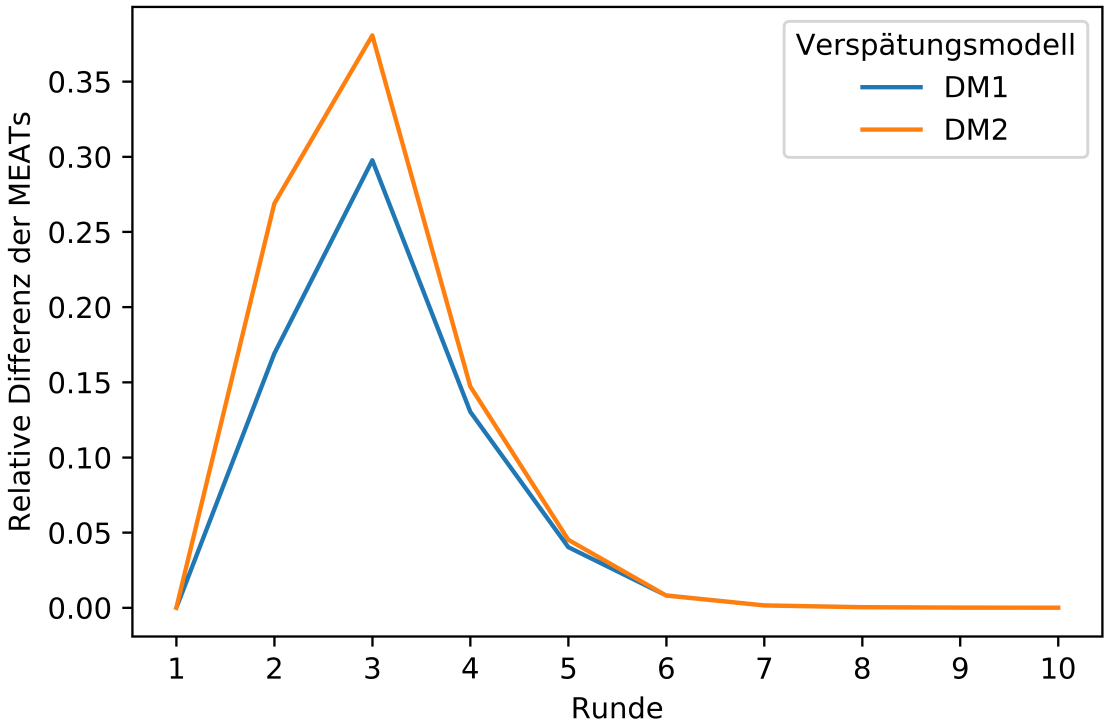

Abbildung 5.4: Die durchschnittliche relative Differenz zwischen der frühesten erwarteten Ankunftszeit einer Runde mit der minimalen erwarteten Ankunftszeit des RAPTOR MEAT Algorithmus im Vergleich zu diesem minimalen Wert. Bei den Werten für kleinere Runden muss beachtet werden, dass nach dieser Anzahl an Runden nicht bei allen Anfragen bereits erwartete Ankunftszeiten berechnet werden konnten.

6 Weboberfläche

Damit die in den Kapiteln 3 und 4 beschriebenen Algorithmen genutzt werden können, entwickeln wir eine Weboberfläche. Über diese können die gewünschten Parameter einer Anfrage eingegeben werden. Nach der Durchführung von einem der Algorithmen werden seine Ergebnisse visualisiert und dem Nutzer alle relevanten Informationen der zugehörigen Reisen angezeigt.

6.1 Benutzung

Wir entwickeln die Weboberfläche getrennt von den Algorithmen zur Routenplanung. Dabei kommuniziert das Frontend der Weboberfläche mit dem Backend, das die Algorithmen enthält, über HTTP Requests. So werden die eingegebenen Parameter ausgehend vom Frontend den Algorithmen im Backend als Eingabe übermittelt. Nach der Berechnung der zugehörigen Reisen und Entscheidungsgraphen durch unsere Algorithmen, werden die Informationen, die zu ihrer Visualisierung benötigt werden, dem Frontend als Antwort zurück gesendet.

Der Programmcode von Backend und Frontend befindet sich in den Git Repositories `https://github.com/sanderjk5/public-transit-backend` und `https://github.com/sanderjk5/public-transit-frontend`. In den darin enthaltenen Readme Dateien ist die Vorgehensweise zur Ausführung der zwei Bestandteile beschrieben. Nach dem Start von Backend und Frontend kann die Weboberfläche über `http://localhost:4200/` erreicht werden.

6.2 Eingabe der Anfragen

Auf der Startseite der Weboberfläche kann der Nutzer die Parameter seiner gewünschten Anfrage eingeben und anschließend die Ausführung eines Algorithmus starten.

Der Nutzer muss dabei zunächst den Startstopp und den Zielstopp seiner Reise eingeben. Dazu liefert die Weboberfläche dem Nutzer während seiner Eingabe Vorschläge zur Vervollständigung der eingegebenen Zeichen. Als weitere Eingabe benötigen die Algorithmen die gewünschte Abfahrtszeit und das Abfahrtsdatum der Reise. Zuletzt kann der Nutzer auswählen, mit welchem Algorithmus das Ergebnis berechnet werden soll. Dazu stehen ihm der Standard CSA und RAPTOR Algorithmus zur Lösung des EAT Problems zur Verfügung. Alternativ kann er zwischen den CSA ExpAT, CSA MEAT und RAPTOR MEAT Algorithmen wählen, um die Entscheidungsgraphen der erwarteten und minimalen erwarteten Ankunftszeiten angezeigt zu bekommen. Als weitere Option steht dem Nutzer die Transferoptimierungsvariante des RAPTOR MEAT Algorithmus zur Verfügung. Mit dieser wird auf der Basis des RAPTOR MEAT Ergebnis der optimale Entscheidungsgraph hinsichtlich der erwarteten Ankunftszeit und der maximalen Anzahl an Umstiegen ausgewählt.

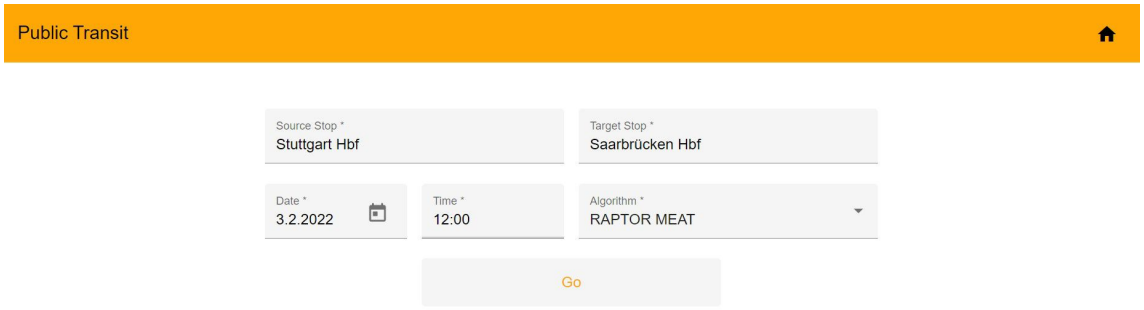

Abbildung 6.1: Die Eingabefelder der Weboberfläche.

Zuletzt liefert die Transferbeschränkungsvariante die minimale erwartete Ankunftszeit nach sechs Runden des RAPTOR MEAT Algorithmus. Nach der Eingabe aller Parameter kann die Berechnung über den Button „Go“ gestartet werden.

Als Verspätungsmodell wird jeweils das erste Modell aus Kapitel 5 verwendet. Der Beschränkungsparameter wird analog zu den Ergebnissen der Tests aus dem Kapitel 5.3.1 auf $\alpha = 2$ gesetzt.

In Abbildung 6.1 wird die Eingabe einer Anfrage dargestellt, bei der der Nutzer am 03.02.2022 um 12 Uhr vom Stuttgarter zum Saarbrückener Hauptbahnhof reisen will. Als Algorithmus ist der RAPTOR MEAT ausgewählt.

6.3 Visualisierung der Resultate

Nach der Durchführung des Algorithmus auf der Eingabe der Parameter werden dem Nutzer die Ergebnisse visualisiert. Diese gliedern wir in zwei Bestandteile, die jeweils in einer Kachel angezeigt werden.

Im ersten Teil werden dem Nutzer die grundlegenden Informationen der ermittelten Reise oder des Entscheidungsgraphen angezeigt. Im Falle der Lösung des EAT Problems umfasst dies den Startstopp, den Zielstopp, die Abfahrtszeit der schnellsten Reise und die minimale Ankunftszeit. Zusätzlich wird die Anzahl der Umstiege und die Verlässlichkeit der Reise unter Berücksichtigung des verwendeten Verspätungsmodells angegeben. Falls der Nutzer einen Algorithmus zu der Lösung des ExpAT oder MEAT Problems ausgewählt hat, werden ihm in dieser Kachel zusätzlich zu dem Startstopp und Zielstopp die minimale Abfahrtszeit und die erwartete Ankunftszeit des Entscheidungsgraphen angezeigt. Um weitere generelle Informationen über die Anfrage zu liefern, werden ihm zudem die minimale und die früheste sichere Ankunftszeit präsentiert.

Die zweite Kachel enthält die spezifischen Informationen der erhaltenen Reise oder des Entscheidungsgraphen. Im Falle der Reise mit der frühesten Ankunftszeit des EAT Problems werden dem Nutzer hier die einzelnen Streckenabschnitte der Reise angezeigt. Über diese erhält er die Information, welche Trips er zwischen den Stopps der Reise nehmen muss. Falls der Nutzer einen

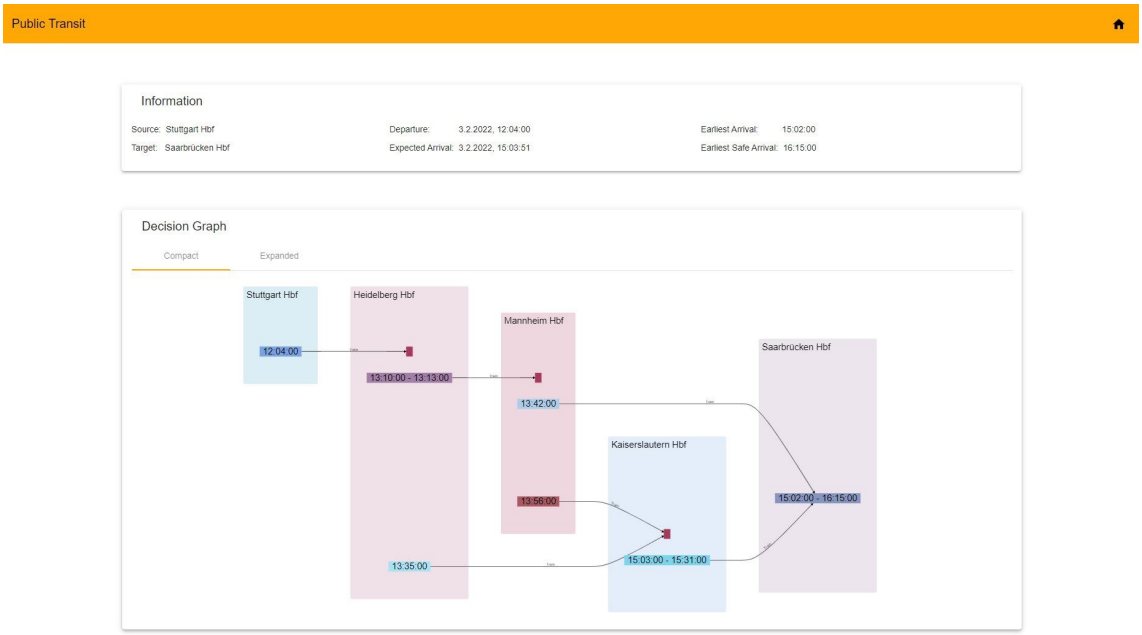

Abbildung 6.2: Die Visualisierung eines kompakten Entscheidungsgraphen in der Weboberfläche.

Algorithmus ausgewählt hat, der einen Entscheidungsgraphen als Lösung besitzt, wird dieser in der zweiten Kachel angezeigt. Dabei kann der Nutzer zwischen den Tabs „Compact“ und „Expanded“ wählen, um sich den kompakten oder erweiterten Entscheidungsgraphen anzeigen zu lassen.

In den Abbildungen 6.2 und 6.3 sind die Ergebnisse der Anfrage aus der Abbildung 6.1 dargestellt. Dabei werden die allgemeinen Informationen, der kompakte und der erweiterte Entscheidungsgraph dargestellt.

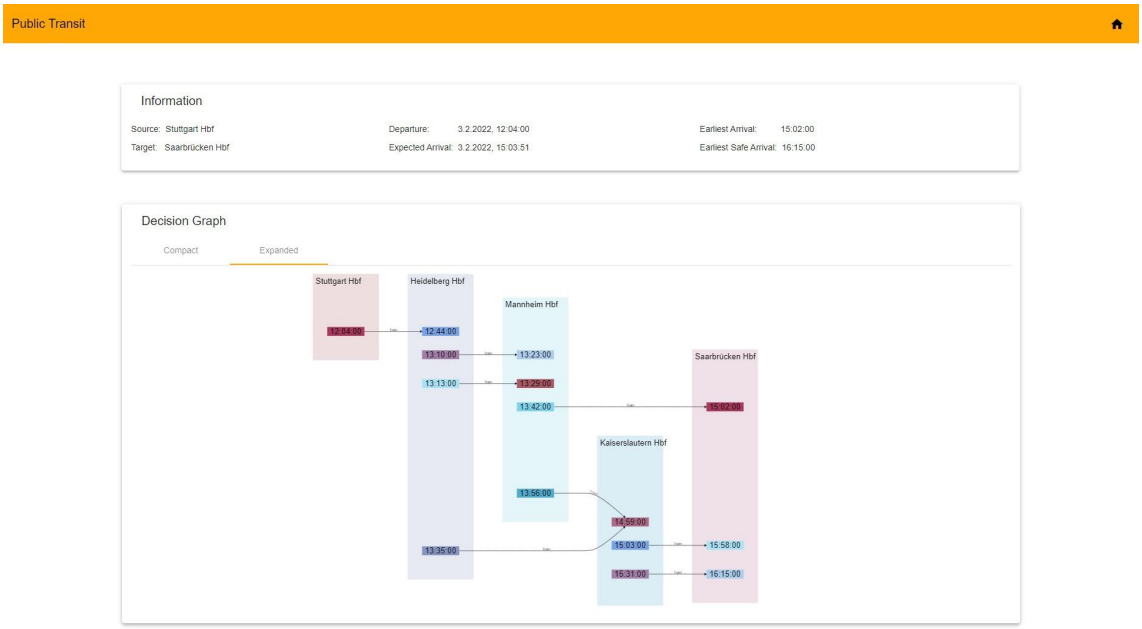

Abbildung 6.3: Die Visualisierung eines erweiterten Entscheidungsgraphen in der Weboberfläche.

7 Fazit

Wir haben basierend auf den bestehenden CSAs und RAPTOR Algorithmen neue Algorithmen zur Lösung der ExpAT und MEAT Probleme entwickelt. Mit dem CSA ExpAT haben wir dabei eine Möglichkeit geschaffen, die erwarteten Ankunftszeiten von den Reisen mit frühesten Ankunftszeiten zu berechnen. Diese können somit die Information liefern, wie verlässlich die schnellstmöglichen Reisen sind. Durch die zugehörige Erstellung eines Entscheidungsgraphen liefert der Algorithmus zusätzlich den Vorteil, dass wir Alternativrouten für diese Reisen bestimmen können. Mit dem RAPTOR MEAT Algorithmus haben wir eine neue Variante zur Berechnung der minimalen erwarteten Ankunftszeiten entwickelt. Für kleine Werte von α resultiert er dabei in besseren Ausführungszeiten. Zudem liefert er den Vorteil, dass wir über das rundenbasierte Vorgehen die maximale Anzahl an Umstiegen innerhalb der Reisen einfach optimieren oder beschränken können. Dies war mit dem bisherigen CSA MEAT nur schwer möglich, kann aber einem Passagier Vorteile bringen.

Aus den Tests zu den ExpAT und MEAT Ergebnissen ergibt sich, dass das Konzept der minimalen erwarteten Ankunftszeiten im Vergleich zu der Auswahl der Reisen ausschließlich über die frühesten Ankunftszeiten in den meisten Fällen hinsichtlich der erwarteten Ankunftszeit deutliche Vorteile liefert. Jedoch zeigt sich, dass das Ergebnis der minimalen erwarteten Ankunftszeit von α und somit der Anzahl der betrachteten Verbindungen und Trips abhängt. Hierbei liefern die Algorithmen für $\alpha = 2$ gute Ergebnisse, aber benötigen mehr Zeit als für kleinere Werte von α. Ein weiteres Ergebnis der Tests ist, dass über die Transferoptimierungsvariante des RAPTOR MEAT Algorithmus die Anzahl der maximalen Umstiege verringert werden kann, ohne die minimale erwartete Ankunftszeit erheblich zu verschlechtern.

Ausblick

Über die von uns entwickelte Weboberfläche haben Nutzer die Möglichkeit sich für die gleiche Anfrage die unterschiedlichen Ergebnisse der ExpAT und MEAT Probleme und zusätzlich der Varianten des RAPTOR MEAT Algorithmus anzeigen zu lassen. Je nach Anfrage und Präferenz des Nutzers kann dabei einer der Entscheidungsgraphen als Grundlage für seine Reise genutzt werden. Für zukünftige Arbeiten wäre es interessant, Möglichkeiten zu finden, wie die Algorithmen kombiniert werden können. Dabei würden auf Eingabe der Anfrage-Parameter alle Algorithmen ausgeführt und einer der daraus resultierenden Entscheidungsgraphen auf der Basis von benutzerdefinierten Optimalitätskriterien ausgewählt werden. Für den Nutzer könnte dies den Vorteil bieten, dass er nicht mehr manuell die unterschiedlichen Algorithmen ausführen und vergleichen müsste.

Eine weitere Ergänzung dieser Arbeit könnte sich dem verwendeten Verspätungsmodell widmen. Hierbei wäre es interessant zu sehen, inwiefern sich die Ergebnisse der Algorithmen bei einem Modell verändern, dass sich näher an echten Daten von Verspätungen orientiert.

Literaturverzeichnis

[BGM10] A. Berger, M. Grimmer, M. Müller-Hannemann. „Fully Dynamic Speed-Up Techniques for Multi-criteria Shortest Path Searches in Time-Dependent Networks“. In: *SEA*. 2010 (zitiert auf S. 27).

[BJ04] G. Brodal, R. Jakob. „Time-dependent Networks as Models to Achieve Fast Exact Time-table Queries“. In: *Proceedings of the 3rd Workshop on Algorithmic Methods and Models for Optimization of Railways (ATMOS'03), Electronic Notes in Theoretical Computer Science* (2004) (zitiert auf S. 27).

[Dij59] E. W. Dijkstra. „A Note on Two Problems in Connexion with Graphs.“ In: *Numberische Mathematik 1* (1959), S. 269–271 (zitiert auf S. 27).

[DKP12] D. Delling, B. Katz, T. Pajor. „Parallel Computation of Best Connections in Public Transportation Networks“. In: *ACM J. Exp. Algorithmics* 17 (Okt. 2012). ISSN: 1084-6654. DOI: 10.1145/2133803.2345678. URL: https://doi.org/10.1145/2133803.2345678 (zitiert auf S. 27).

[DMS08] Y. Disser, M. Müller-Hannemann, M. Schnee. „Multi-Criteria Shortest Path in Time Dependent Train Networks“. In: *Proceedings of the 7th Workshop on Experimental Algorithms (WEA'08), Lecture Notes in Computer Science* (2008), S. 347–361 (zitiert auf S. 27, 51).

[DPSW18] J. Dibbelt, T. Pajor, B. Strasser, D. Wagner. „Connection Scan Algorithm“. In: *ACM J. Exp. Algorithmics* 23 (Okt. 2018). ISSN: 1084-6654. DOI: 10.1145/3274661. URL: https://doi.org/10.1145/3274661 (zitiert auf S. 15, 17, 19, 20, 22, 27, 36, 43, 45, 52).

[DPW15] D. Delling, T. Pajor, R. F. Werneck. „Round-based public transit routing“. In: *Transportation Science* (2015), S. 591–604 (zitiert auf S. 17, 31, 49, 56).

[DW09] D. Delling, D. Wagner. „Time-Dependent Route Planning“. In: *Robust and Online Large-Scale Optimization: Models and Techniques for Transportation Systems*. Hrsg. von R. K. Ahuja, R. H. Möhring, C. D. Zaroliagis. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, S. 207–230. DOI: 10.1007/978-3-642-05465-5_8. URL: https://doi.org/10.1007/978-3-642-05465-5_8 (zitiert auf S. 27).

[PSWZ08] E. Pyrga, F. Schulz, D. Wagner, C. Zaroliagis. „Efficient models for timetable information in public transportation systems.“ In: *ACM Journal of Experimental Algorithmics* (2008) (zitiert auf S. 27).

[Wit15] S. Witt. „Trip-based public transit routing“. In: *Proceedings of the 23rd Annual European Symposium on Algorithms (ESA'15), Lecture Notes in Computer Science* (2015), S. 1025–1036 (zitiert auf S. 27).

Alle URLs wurden zuletzt am 02. 02. 2022 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift