

## Berechnung optimaler Wege im öffentlichen Verkehr

Jurek Sander<sup>1</sup>

**Abstract:** In dieser Arbeit stellen wir einen neuen Algorithmus zur Berechnung optimaler Wege in öffentlichen Verkehrsnetzen vor, der auf dem Round-Based Public Transit Routing (RAPTOR) Algorithmus von Delling et al. aus [Delling et al., Transportation Science, 2015] basiert. Im Gegensatz zu den meisten bestehenden Arbeiten wählen wir als Optimalitätskriterium nicht die planmäßige, sondern die erwartete Ankunftszeit. Wir berücksichtigen somit mögliche Verspätungen durch die Definition eines geeigneten Wahrscheinlichkeitsmodells und erreichen dadurch eine deutlich höhere Planungssicherheit. Unser Algorithmus ist darüber hinaus in der Lage, weitere Kriterien wie beispielsweise die maximale Anzahl der Umstiege in der Berechnung optimaler Routen zu berücksichtigen und ist deshalb flexibler einsetzbar als der einzige bereits bekannte Algorithmus dieser Art, der in [Dibbelt et al., ACM J Exp. Alg. 23, 2018] vorgestellt wurde.

**Keywords:** Fahrplan; Öffentlicher Verkehr; Erwartete Ankunftszeit; Verspätung; Entscheidungsgraph

### 1 Einleitung

Bei der Benutzung des öffentlichen Regional- und Fernverkehrs spielt die Routenplanung eine entscheidende Rolle. In der klassischen Routenberechnung werden Reisen beispielsweise mit dem Connection Scan Algorithm (CSA) oder RAPTOR-Algorithmus auf der Basis der geplanten Verbindungen hinsichtlich ihrer Fahrtzeiten optimiert. In der Realität kann es jedoch zu Verspätungen der Verkehrsmittel kommen, die sich auf Reisen der Passagiere auswirken. Daher wollen wir sie unter Berücksichtigung möglicher Verspätungen optimieren. In diesem Bereich gibt es in der Forschung bisher wenige Ansätze, da lediglich für die Art der CSAs durch Dibbelt et al. (2018) in [Di18] bereits eine Optimierung der erwarteten Ankunftszeiten durchgeführt wurde. Für die RAPTOR-Algorithmen gibt es noch keine Lösung für diese Problemstellung und es wurde noch nicht untersucht, inwiefern weitere Optimalitätskriterien im Zusammenhang mit den erwarteten Ankunftszeiten berücksichtigt werden können.

Im Bereich der Routenplanung innerhalb von öffentlichen Verkehrsnetzen wurden bereits unterschiedliche Herangehensweisen erforscht und für diese Algorithmen entwickelt. Verkehrsnetze können als Graphen dargestellt werden, wodurch es möglich ist, die Berechnungen mit einem modifizierten Dijkstra-Algorithmus aus [Di59] durchzuführen. Dieser ist jedoch deutlich unflexibler und ineffizienter als Algorithmen, die konkret für öffentliche Verkehrsnetze entwickelt werden. In [BGM10] verbessern Berger et al. (2010) mit ihrer

---

<sup>1</sup> Universität Stuttgart, Institut für Formale Methoden der Informatik, Universitätsstraße 38, 70569 Stuttgart, Deutschland, st162270@stud.uni-stuttgart.de

Methode die Verwendung des Dijkstra-Algorithmus, indem sie untere Grenzen verwenden. Delling et al. (2012) haben in [DKP12] ihren Self-Pruning Connection-Setting (SPCS) Algorithmus eingeführt. Auch dieser basiert auf der Darstellung der öffentlichen Verkehrsnetze als Graphen, aber er sucht nicht ausschließlich nach der schnellstmöglichen Route für Passagiere, sondern gibt unterschiedliche Möglichkeiten innerhalb eines gegebenen Zeitintervalls an. Der Algorithmus verwendet Vereinfachungsmethoden und besitzt zudem eine parallelisierbare Variante. Um Reisen innerhalb eines öffentlichen Verkehrsnetzes nach mehr als nur 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 Multi-Label-Correcting (MLC) Algorithmus von Pygra et al. (2008), der in [Py08] und [DW09] vorgestellt wurde, basiert ebenfalls auf dem Dijkstra-Algorithmus, aber ermöglicht die Optimierung von mehr als zwei Kriterien. Er wurde von Disser et al. (2008) in [DMS08] verbessert. Die CSAs, die von Dibbelt et al. (2018) in [Di18] vorgestellt wurden, basieren nicht auf dem Dijkstra-Algorithmus. Für sie werden aus den Fahrplandaten jeweils für zwei aufeinanderfolgend angefahrenen Haltestellen der Verkehrsmittel Verbindungen erstellt. Die Algorithmen haben gemeinsam, dass es während ihrer Ausführung ausreicht, einmalig über die Menge der Verbindungen zu iterieren. Mit ihnen können Problemstellungen der Routenberechnungen innerhalb von öffentlichen Verkehrsnetzen, die auf den geplanten Ankunftszeiten basieren, gelöst werden. Zudem wurde für die CSAs eine Variante entwickelt, mit der mögliche Verspätungen über erwartete Ankunftszeiten bei der Optimierung der Routen berücksichtigt werden können. Deren Ergebnisse werden über Entscheidungsgraphen visualisiert, die dem Passagier alternative Reisen anzeigen, falls er durch Verspätungen von seiner ursprünglichen Route abweichen muss. Auch die RAPTOR-Algorithmen, die von Delling et al. (2015) in [DPW15] entwickelt wurden, basieren nicht auf dem Konzept des Dijkstra-Algorithmus. Im Gegensatz zu den CSAs sind die Ausführungszeiten der RAPTOR-Algorithmen etwas langsamer, aber mit ihnen ist es deutlich einfacher umsetzbar zusätzliche Optimalitätskriterien zu verwenden. Zudem sind diese im Gegensatz zu den CSAs leicht parallelisierbar.

Uns ist noch kein Algorithmus der RAPTOR-Familie bekannt, der mögliche Verspätungen der Verkehrsmittel berücksichtigt. Um die Vorteile dieser Art an Algorithmen mit diesem Optimalitätskriterium zu verbinden, führt unser neu entwickelter Algorithmus eine Optimierung der erwarteten Ankunftszeiten durch. Mit ihm ist es möglich weitere Kriterien, wie beispielsweise die maximale Anzahl an Umstiegen zu berücksichtigen, die ein Passagier auf seiner Reise zwischen Start- und Zielpunkt vornehmen muss. Dies liefert alternative Ergebnisse, die komfortablere Reisen darstellen können, aber trotzdem mögliche Verspätungen über die erwarteten Ankunftszeiten berücksichtigen.

## 2 Grundlagen

Als Grundlage für unseren neuen Algorithmus benötigen wir einige Definitionen, die Problembeschreibungen und den grundlegenden Aufbau der RAPTOR-Algorithmen.

## 2.1 Definitionen

Die nachfolgenden Definitionen verwenden wir analog zu Delling et al. (2015) in [DPW15]. Innerhalb eines öffentlichen Verkehrsnetzes stellen Haltestellen Knotenpunkte dar. An diesen Stopps können Passagiere in Verkehrsmittel einsteigen, aussteigen oder zwischen ihnen umsteigen. Verkehrsmittel fahren auf fest definierten Routen. Eine Route stellt dabei eine Teilmenge der Menge der Stopps dar. Für diese ist innerhalb der Route eine feste Reihenfolge definiert, in der sie abgefahren werden. Für jede Route existieren Trips. Diese definieren die Ankunfts- und Abfahrtszeiten von Verkehrsmitteln an den Stopps der Route. Somit besitzt jeder Trip  $t'$  einer Route  $r$  an jedem Stopp  $p$  von  $r$  eine Ankunftszeit  $\tau_{\text{arr}}(t', p)$  und Abfahrtszeit  $\tau_{\text{dep}}(t', p)$ . Für diese gilt  $\tau_{\text{arr}}(t', p) \leq \tau_{\text{dep}}(t', p)$ . 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. Um Wege von Passagieren im Verkehrsnetz beschreiben zu können, benötigen wir Streckenabschnitte. Jeder Streckenabschnitt  $l$  ist einem Trip  $l_{\text{trip}}$  zugeordnet und definiert sich über den Einstiegspunkt  $l_{\text{dep\_stop}}$  und Ausstiegspunkt  $l_{\text{arr\_stop}}$  des Passagiers und deren Zeiten innerhalb des Trips. Die Reise  $j$  eines Passagiers zwischen dem Startstopp  $s$  und dem Zielstopp  $t$  innerhalb des Verkehrsnetzes kann nun über die Sequenz an Streckenabschnitten  $j = l^0, l^1, \dots, l^k$  mit  $l^0_{\text{dep\_stop}} = s$  und  $l^k_{\text{arr\_stop}} = t$  definiert werden. Sie besitzt die Abfahrtszeit  $j_{\text{dep\_time}}$  an  $s$  und Ankunftszeit  $j_{\text{arr\_time}}$  an  $t$ . Dabei muss  $l^i_{\text{arr\_stop}} = l^{i+1}_{\text{dep\_stop}}$  und  $l^i_{\text{arr\_time}} \leq l^{i+1}_{\text{dep\_time}}$  für alle  $i = 0, \dots, k - 1$  gelten, um einen Umstieg zwischen den Trips zweier aufeinanderfolgender Streckenabschnitte zu ermöglichen. Die Reise  $j$  besteht dabei aus  $k + 1$  Trips, zwischen denen der Passagier  $k$ -Mal umsteigen muss. 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.

### 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 führen wir analog zu Dijkstra et al. in [Di18] ein. Dabei definieren wir für jeden Trip  $t'$  an jedem seiner Stopps  $p$  eine Zufallsvariable  $D(t', p)$ , die die Verspätung des Trips an diesem Stopp angibt. Für diese Zufallsvariable definiert die Verteilungsfunktion  $f(x)$  die Wahrscheinlichkeit  $P[D(t', p) \leq x]$  mit  $x$  als Verspätung in Minuten. Der zugehörige Erwartungswert, der die erwartete Verspätung des Trips  $t'$  an  $p$  darstellt, sei  $E[D(t', p)]$ . Damit wir erwartete Ankunftszeiten berechnen können, müssen wir zusätzlich zu dem Verspätungsmodell Annahmen treffen. Die erste besagt, dass jede Zufallsvariable  $D(t', p)$  eine maximale Verspätung  $\max D(t', p)$  besitzt. Zudem nehmen wir an, dass alle Zufallsvariablen unabhängig voneinander sind

und dass die Verkehrsmittel aller Trips pünktlich abfahren. Verspätungen werden somit nur bei ihren Ankünften betrachtet. Diese Annahmen werden analog zu [Di18] getroffen. Hier wurde gezeigt, dass das Modell realistisch ist, solange keine massiven Verspätungen auftreten.

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

Erwartete Ankunftszeiten definieren wir analog zu [Di18] über  $(s, t, \tau_s)$ -Entscheidungsgraphen. Diese umfassen Reisen zwischen dem Startstopp  $s$  und dem Zielstopp  $t$  mit einer frühesten Abfahrtszeit  $\tau_s$ . 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  $l$ . Die erwartete Ankunftszeit 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. Falls  $l$  zum Zielstopp  $t$  führt, setzt sie sich aus der geplanten Ankunftszeit des zugehörigen Trips und seiner erwarteten Verspätung zusammen. Andernfalls wird die erwartete Ankunftszeit über die ausgehenden Streckenabschnitte des Knotens  $l_{\text{arr\_stop}}$  definiert. Sie stellt dabei die gewichtete Aufsummierung der erwarteten Ankunftszeiten der ausgehenden Trips dar, in die der Passagier umsteigen kann. Die Gewichtung der Streckenabschnitte erfolgt dabei jeweils über die Wahrscheinlichkeit, zu der der Passagier abhängig von der Verspätung seines eingehenden Verkehrsmittels in den zugehörigen Trip umsteigt. Die erwartete Ankunftszeit des Entscheidungsgraphen  $G$  entspricht der erwarteten Ankunftszeit des ersten Streckenabschnitts am Startstopp  $s$ . Die späteste Ankunftszeit des Graphen wird mit  $G_{\text{max\_arr}}$  notiert. Für den Nutzer ergibt sich die Reise, die er dem Graph zufolge nehmen sollte, jeweils über den ersten Streckenabschnitt, den er an einem 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 Abb. 1 ist das Beispiel eines Entscheidungsgraphen dargestellt.

## 2.2 Problembeschreibungen

Wir beschäftigen uns mit unterschiedlichen Optimalitätskriterien für Reisen zwischen zwei Stopps und definieren hierfür die Probleme analog zu [Di18] und [DPW15]. Das Problem der frühesten Ankunftszeit (Earliest Arrival Time (EAT)) gibt auf der Eingabe

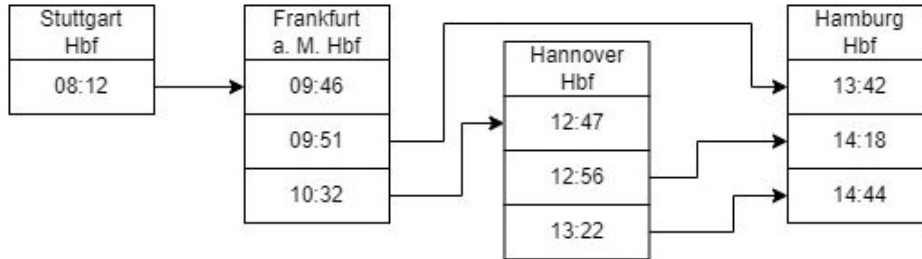


Abb. 1: Ein Entscheidungsgraph zwischen den Stopps Stuttgart Hbf und Hamburg Hbf.

eines Startstopps  $s$ , Zielstopps  $t$  und einer Startzeit  $\tau_s$  die minimale Ankunftszeit aller Reisen an, die an Stopp  $s$  nach  $\tau_s$  abfahren und an Stopp  $t$  ankommen. Eine Abwandlung des EAT-Problems ist das Problem der frühesten sicheren Ankunftszeit (Earliest Safe Arrival Time (ESAT)), das lediglich sichere Reisen berücksichtigt. Das EAT-Problem kann auf das Problem der frühesten Ankunftszeit im Intervall (Earliest Arrival Time Profile (EATP)) erweitert werden. Dabei ist als Eingabe ein Startstopp  $s$ , Zielstopp  $t$  und ein Intervall an Abfahrtszeiten  $\Delta$  gegeben. Gesucht sind alle Pareto-optimalen Reisen zwischen  $s$  und  $t$ , die innerhalb des Intervalls an  $s$  abfahren. Die Pareto-Optimalität wird bezüglich der Abfahrts- und Ankunftszeit der Reisen definiert. Das Problem der minimalen erwarteten Ankunftszeit (Minimum Expected Arrival Time (MEAT)) besitzt als Eingabe einen Startstopp  $s$ , Zielstopp  $t$  und eine minimale Abfahrtszeit  $\tau_s$ . Wie in [Di18] verwenden wir die  $\alpha$ -beschränkte Variante des Problems, wobei die maximale Ankunftszeit auf  $G_{\max\_arr} \leq \tau_s + \alpha \cdot (\text{ESAT}(s, t, \tau_s) - \tau_s)$  mit  $\alpha \geq 1$  gesetzt und daraufhin der Entscheidungsgraph  $G$  mit der minimalen erwarteten Ankunftszeit gesucht wird.

### 2.3 RAPTOR-Algorithmen

Die nachfolgenden RAPTOR-Algorithmen wurden von Delling et al. (2015) in [DPW15] vorgestellt. RAPTOR-Algorithmen berechnen die frühesten Ankunftszeiten jeweils rundenbasiert. So erhält man nach  $k$  Runden die frühesten Ankunftszeiten der Reisen, die maximal  $k$  Streckenabschnitte und somit  $k - 1$  Umstiege umfassen.

In der Standardvariante des RAPTOR-Algorithmus zur Lösung des EAT-Problems wird für jeden Stopp  $p$  in  $\tau^*(p)$  die bisherige frühestmögliche Ankunftszeit und in  $\tau_k(p)$  die minimale Ankunftszeit nach  $k$  Runden gespeichert. In einer Runde werden jeweils alle Stopps  $p$  markiert, für die ein neuer Wert in  $\tau^*(p)$  gesetzt wurde. In jeder Runde des Algorithmus betrachten wir alle Routen, die einen markierten Stopp der letzten Runde enthalten und ermitteln ihren frühesten Trip, den der Passagier erreichen kann. Dessen Ankunftszeiten können nun verwendet werden, um die frühesten Ankunftszeiten der Stopps der Route in ihrer definierten Reihenfolge zu aktualisieren. An jedem Stopp wird zusätzlich überprüft, ob ausgehend von der minimalen Ankunftszeit der letzten Runde ein früherer

Trip erreicht werden kann. Der Algorithmus terminiert, sobald keine Stopps in einer Runde markiert werden. Die Reise der frühesten Ankunftszeit am Ziel  $t$ , die in  $\tau^*(t)$  gespeichert ist, können wir über zusätzliche Zeiger ermitteln.

Der davon abgeleitete McRAPTOR-Algorithmus kann dazu genutzt werden, das EATP-Problem zu lösen. Dazu speichern wir Labels einer Runde  $k$  für Stopps  $p$  in Beuteln  $B_k(p)$ . Jedes Label ist einem Trip zugeordnet und enthält die Abfahrtszeit am Startstopp  $s$  und Ankunftszeit am Stopp  $p$ . Beutel sind jeweils hinsichtlich dieser Parameter Pareto-optimal. Der Routenbeutel  $B_r$  enthält die Labels der aktuellen Route  $r$ . Während des Algorithmus betrachten wir in jeder Runde  $k$  jede Route  $r$ . Dabei führen wir für jeden ihrer Stopps  $p$  in ihrer definierten Reihenfolge drei Schritte durch. Zunächst aktualisieren wir die Ankunftszeiten aller Labels des  $B_r$  mit den Ankunftszeiten der zugehörigen Trips an  $p$ . Dann fügen wir unter Berücksichtigung der Dominanzregeln die Labels aus  $B_r$  in  $B_k(p)$  ein und wiederholen dies für  $B_{k-1}(p)$  und  $B_r$ . Bei dem letzten Schritt weisen wir den neuen Labels in  $B_r$  jeweils den ersten erreichbaren Trip von  $r$  zu. Zusätzliche Beutel  $B^*(p)$  für Stopps  $p$  können verwendet werden, um alle nicht dominierten Labels zu speichern und somit den Algorithmus zu optimieren.

### 3 RAPTOR-MEAT

Nachdem das MEAT-Problem bereits mit dem Ansatz der CSAs gelöst wurde, entwickeln wir nun einen Algorithmus, bei dem dies auch über die rundenbasierte Herangehensweise des McRAPTOR-Algorithmus möglich ist. Der RAPTOR-MEAT-Algorithmus kann so erweitert werden, dass zusätzliche Optimalitätskriterien berücksichtigt werden.

Um minimale erwartete Ankunftszeiten berechnen zu können, verändern wir das Konzept des McRAPTOR-Algorithmus in der Hinsicht, dass wir ausgehend vom Zielstopp  $t$  die erwarteten Ankunftszeiten bis hin zum Startstopp  $s$  berechnen. Dazu benötigen wir ein Intervall an Ankunftszeiten als Eingabe. Über einen modifizierten CSA erhalten wir die früheste sichere Ankunftszeit  $\text{ESAT}(s, t, \tau_s)$  mit der Eingabe des Startstopps  $s$ , der minimalen Abfahrtszeit  $\tau_s$  und dem Zielstopp  $t$ . Als obere Grenze des Intervalls ergibt sich daraus die maximale Ankunftszeit  $\tau_t$ :  $\tau_t = \tau_s + \alpha \cdot (\text{ESAT}(s, t, \tau_s) - \tau_s)$ . Zudem benötigen wir für unseren Algorithmus die frühesten Ankunftszeiten an jedem Stopp ausgehend von dem Startstopp  $s$  und der minimalen Abfahrtszeit  $\tau_s$ . Dazu führen wir einen 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 unseren Algorithmus.

Im Gegensatz zu der Standardvariante des McRAPTOR-Algorithmus optimiert unsere Variante die minimalen erwarteten Ankunftszeiten. Die Labels eines Stopps  $p$  speichern somit Tupel bestehend aus der Abfahrtszeit an  $p$  und der erwarteten Ankunftszeit am Zielstopp  $t$ . 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 den Abfahrtszeiten sortiert sind. Der RAPTOR-MEAT-Algorithmus speichert für jeden Stopp  $p$  einen Beutel  $B^*(p)$  mit allen nicht dominierten Labels der bisher ausgeführten Runden. Unser Algorithmus arbeitet wie der McRAPTOR-Algorithmus rundenbasiert. In jeder Runde benötigen wir drei Arrays, die danach wieder zurückgesetzt werden können. Im ersten Array LD 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  $\tau_t$  hinzu. In dem Array Q speichern wir alle Routen-Stopp-Paare, die wir in der aktuellen Runde betrachten. Im letzten Array sichern wir für jeden Stopp  $p$  einen Beutel  $B_c(p)$  mit den Labels, die in der aktuellen Runde hinzugefügt werden. Initial wird der Stopp  $t$  markiert und der Rundenzähler  $k$  auf 0 gesetzt.

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

Zu Beginn jeder Runde inkrementieren wir den Rundenzähler  $k$  und bestimmen alle Routen, die mindestens einen markierten Stopp der letzten Runde enthalten und somit in der aktuellen Runde betrachtet werden müssen. Die zugehörigen Routen-Stopp-Paare  $(r, p)$  werden in Q gespeichert. Dabei entspricht  $p$  dem letzten markierten Stopp der Route  $r$ .

Die darauffolgende Schleife führen wir für jeden Eintrag  $(r, p)$  von Q durch. Zunächst initialisieren wir den Routenbeutel  $B_r$ , der alle nicht dominierten Label von  $r$  speichert, mit einem leeren Array. Während der Behandlung einer Route  $r$  führen wir für jeden Stopp  $p'$  ab dem Stopp  $p$  drei Schritte analog zu dem McRAPTOR durch. Dabei werden die Stopps innerhalb der Route von hinten nach vorne abgearbeitet. Zunächst wird der Routenbeutel mit den Abfahrtszeiten am aktuellen Stopp aktualisiert, dann wird er mit dem Beutel der Labels von  $p'$  der aktuellen Runde zusammengefügt und abschließend werden neue Labels in den Routenbeutel hinzugefügt.

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. Im nächsten Schritt werden die Labels des Routenbeutels dem Beutel  $B_c(p')$  des Stopps  $p'$  hinzugefügt. Wir verwenden für den Vorgang des Zusammenfügens, dass sowohl der Beutel  $B_r$  als auch der Beutel  $B_c(p')$  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 danach als 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. Da wir in diesem Schritt zusätzlich die erwarteten Ankunftszeiten der neuen Labels berechnen müssen, wird hier das Verfahren des McRAPTOR angepasst. Dabei wählen wir zunächst alle Trips der Route  $r$  aus, deren Ankunftszeit zwischen der frühestmöglichen Ankunftszeit und der spätesten Abfahrtszeit der letzten Runde an diesem Stopp liegt. Die frühestmögliche 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 Array LD, 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. Für alle Trips, die innerhalb des definierten Intervalls zwischen frühester Ankunftszeit und spätester Abfahrtszeit an  $p'$  ankommen, erzeugen wir ein neues Label. Bei der Berechnung der erwarteten Ankunftszeiten der neuen Labels führen wir analog zu ihrer Definition 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. Im anderen Fall ergibt 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. Wir verwenden für diese Berechnung die Labels, die in dem Beutel  $B^*(p')$  für  $p'$  gespeichert sind. Dieser umfasst alle nicht dominierten Labels der vorangegangenen Runden. Falls die erwartete Ankunftszeit berechnet werden kann, setzen wir die Abfahrtszeit des neuen Labels auf die Abfahrtszeit des Trips an dem Stopp  $p'$ . Die neuen Labels fügen wir in den Routenbeutel  $B_r$  ein.

Bei der klassischen Variante des McRAPTOR werden die Beutel  $B^*$  während der vorherigen Schleife um die neuen Labels ergänzt. Da diese Beutel bei uns zur Berechnung der erwarteten Ankunftszeiten verwendet werden, erfolgt dies beim RAPTOR-MEAT erst, nachdem alle Routen einer Runde behandelt wurden. Dabei wird zunächst das Array LD der spätesten Abfahrtszeiten geleert. Anschließend fügen wir für alle Stopps  $p$  die Beutel  $B_c(p)$  und  $B^*(p)$  zusammen und verwenden das Ergebnis als neuen Beutel  $B^*(p)$ . Dazu werden nur nicht dominierte Labels berücksichtigt, und falls mindestens eines der Labels aus der aktuellen Runde darunter ist, markieren wir den Stopp  $p$ . Zusätzlich wird die späteste Abfahrtszeit der neuen Labels, die in den Beutel  $B^*(p)$  eingefügt wurden, am Stopp  $p$  in dem Array der spätesten Abfahrtszeiten gespeichert.

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. Sobald der Algorithmus terminiert ist, können die Labels analog zu dem Vorgehen beim CSA-MEAT in [Di18] verwendet werden, um den Entscheidungsgraphen der minimalen erwarteten Ankunftszeit zu erstellen.

Die Behandlung der einzelnen Routen und das abschließende Aktualisieren der Beutel  $B^*$  kann innerhalb einer Runde jeweils parallelisiert werden, da es unabhängig für die Routen beziehungsweise Stopps erfolgt. Analog zu [DPW15] können dabei Konfliktgraphen verwendet werden, um Speicherungs-Konflikte zu vermeiden. Die Korrektheit des Algorithmus basiert auf der Invariante, dass zu Beginn der Runde  $k$  die minimalen erwarteten Ankunftszeiten der Reisen, die aus maximal  $k - 1$  Streckenabschnitten bestehen, in  $B^*$  gespeichert sind. Diese können in der Runde  $k$  dazu verwendet werden, gemäß der Definition



der erwarteten Ankunftszeiten diese für die Reisen bestehend aus  $k$  Streckenabschnitten zu berechnen. Somit werden die bisher betrachteten Reisen rundenbasiert jeweils um einen Umstieg bzw. Streckenabschnitt verlängert. Die Invariante kann dazu verwendet werden, den RAPTOR-MEAT-Algorithmus in der Hinsicht zu erweitern, dass dem Nutzer alternative Ergebnisse 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 einer Reise verbessern, da sie seltener zwischen Trips umsteigen müssen, aber eine annähernd gleich gute Ankunftszeit erwarten können. Die entstehende Variante des Algorithmus bezeichnen wir mit RAPTOR-MEAT-TO (Transfer Optimisation). 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 der  $B^*$  Beutel. 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 Parameter definieren wir hierfür die Zeit pro verringerter Anzahl an Umstiegen, um die sich die erwartete Ankunftszeit von der minimalen erwarteten Ankunftszeit unterscheiden darf. Über die Ergebnisse, die in den Beuteln  $B^*$  gespeichert wurden, können wir die erste Runde bestimmen, in der das Kriterium erfüllt ist. Der zugehörige Entscheidungsgraph stellt das Resultat des RAPTOR-MEAT-TO dar. Eine ähnlich einfache Umsetzung der Variante ist aus unserer Sicht mit dem bestehenden CSA-MEAT aus [Di18] nicht möglich.

## 4 Experimente

Wir führen nun Benchmark-Tests und weitere Experimente für den neuen RAPTOR-MEAT-Algorithmus und seine Variante aus dem Kapitel 3 durch. Als Grundlage für die Tests nutzen wir die Datensätze des Schienenregional- und Schienenfernverkehrs in Deutschland. Diese umfassen alle S-Bahnen, Regionalbahnen, ICs und ICEs. Wir führen alle Tests auf denselben 1000 zufällig ausgewählten Anfragen aus. Als Verspätungsmodell nutzen wir das Modell von Disser et al. (2008) in [DMS08]. Es basiert auf der Funktion  $\text{rel} : \tau \mapsto s - e^{\ln(1-a) - \frac{\tau}{b}}$ , die wir als Verteilungsfunktion  $f$  mit  $s = 1$  verwenden. 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. Die für die Tests verwendete Hardware besitzt einen Intel Core i5 mit 6 Kernen, eine 500 GB NVMe Festplatte und 64 GB RAM. Auf ihr läuft Ubuntu 20.04. Bei dem RAPTOR-MEAT-Algorithmus verwenden wir die Single-Core Variante ohne Parallelisierung.

#### 4.1 Benchmark-Tests

Bei den Benchmark-Tests vergleichen wir die Ausführungszeiten des neuen RAPTOR-MEAT-Algorithmus mit denen des bestehenden CSA-MEAT. Dabei führen wir die Tests mit den Beschränkungsparametern von  $\alpha = 1$ ,  $\alpha = 2$  und  $\alpha = 3$  durch, um mögliche Einflüsse der Größe des initialen Intervalls an Verbindungen und Trips, die genutzt werden können, zu erkennen. Wie man in Abb. 2 sehen kann, ist der RAPTOR-MEAT-Algorithmus für  $\alpha = 1$  mit einer Zeit von 182 ms schneller als der CSA-MEAT, der 214 ms benötigt. Dies ändert sich für größere Werte von  $\alpha$ , da hier die Ausführung des RAPTOR-MEAT-Algorithmus länger dauert. Das lässt sich damit erklären, dass einerseits dann insgesamt mehr Trips berücksichtigt werden, die wir durch das rundenbasierte Vorgehen potenziell mehrfach betrachten und zudem längere Reisen möglich sind. Daraus resultieren mehr Runden, die während des Algorithmus ausgeführt werden müssen. Im Gegensatz dazu steigt bei den CSAs lediglich die Anzahl der betrachteten Verbindungen linear, da bei ihnen gesichert ist, dass jede Verbindung maximal einmal betrachtet wird. Dennoch lohnt es sich den RAPTOR-MEAT-Algorithmus auszuführen, da nur mit ihm eine einfache Optimierung der maximalen Anzahl an Umstiegen und eine Parallelisierung möglich ist. Außerdem verbessern sich in unseren Experimenten die erwarteten Ankunftszeiten durchschnittlich nur um sechs Minuten, wenn man  $\alpha = 2$  statt  $\alpha = 1$  wählt. Die Differenz von  $\alpha = 2$  zu  $\alpha = 3$  liegt im Durchschnitt sogar nur bei 0,03 s.

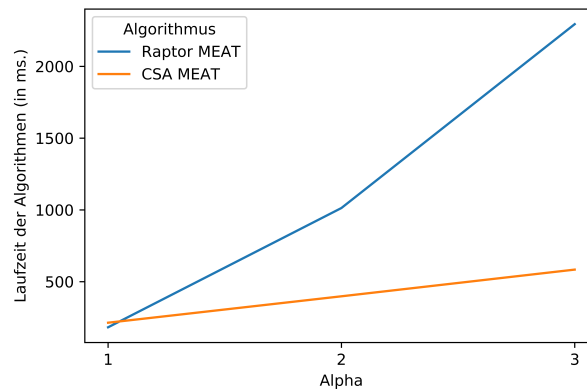


Abb. 2: Die Gesamtlaufzeiten des CSA-MEAT und RAPTOR-MEAT-Algorithmus für unterschiedliche Werte von  $\alpha$ .

#### 4.2 Relevanztests

Um bestimmen zu können, welche Relevanz die Berechnung der minimalen erwarteten Ankunftszeiten hat, vergleichen wir sie mit den erwarteten Ankunftszeiten der Reisen, deren Optimierung ausschließlich hinsichtlich der frühesten geplanten Ankunftszeit erfolgt. Um diese Werte zu bestimmen, haben wir eine Variante der CSAs entwickelt, die auch

als Alternativreisen jeweils die Reisen mit den minimalen 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 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 den frühesten Ankunftszeiten der klassischen Routenberechnung in öffentlichen Verkehrsnetzen richtet. Im Durchschnitt beträgt die Differenz der Ergebnisse der beiden Algorithmen für  $\alpha = 2$  etwa 1200 s bzw. 20 Minuten. Ein Passagier muss somit annehmen, um diese Zeitspanne später am Ziel anzukommen, wenn er sich nicht nach den minimalen erwarteten, sondern frühesten Ankunftszeiten richtet. Im Extremfall beträgt die Differenz absolut sogar über 8 Stunden und relativ gesehen 62% zur minimalen erwarteten Fahrtzeit.

Um herauszufinden, welche Vorteile die RAPTOR-MEAT-TO-Variante liefern kann, vergleichen wir ihre erwarteten Ankunftszeiten und 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 vornehmen muss, kleiner oder gleich groß wie die des RAPTOR-MEAT-Algorithmus. Für  $\alpha = 2$  und fünf Minuten als Parameter der Variante ist die erwartete Ankunftszeit der Transferoptimierungsvariante im Durchschnitt ca. 100 s später als die minimale erwartete Ankunftszeit. Dafür ist die maximale Anzahl an Umstiegen des Passagiers durchschnittlich um einen Umstieg geringer. 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. Bei den Maximalwerten lohnt es sich besonders das Ergebnis der RAPTOR-MEAT-TO-Variante zu verwenden, da hier die Anzahl an Umstiegen um 10 reduziert werden konnte. Jedoch muss man beachten, dass die erwartete Ankunftszeit um bis zu 30 Minuten erhöht wurde.

## 5 Fazit

Wir haben basierend auf den bestehenden RAPTOR-Algorithmen einen neuen Algorithmus zur Lösung des MEAT-Problems entwickelt. Mit dem RAPTOR-MEAT-Algorithmus haben wir eine parallelisierbare Variante zur Berechnung der minimalen erwarteten Ankunftszeiten geschaffen. Für kleine Werte von  $\alpha$  resultiert er dabei in besseren Ausführungszeiten als der CSA-MEAT. Zudem liefert er den Vorteil, dass wir über das rundenbasierte Vorgehen die maximale Anzahl an Umstiegen innerhalb der Reisen einfach optimieren können. Dies ist mit dem CSA-MEAT nur schwer möglich, kann aber einem Passagier Vorteile bieten. Aus unseren Relevanztests 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 und somit für mehr Planungssicherheit sorgt. Ein weiteres Ergebnis der Tests ist, dass über die Transferoptimierungsvariante des RAPTOR-MEAT-Algorithmus die maximale Anzahl der Umstiege verringert werden kann, ohne die minimale erwartete Ankunftszeit erheblich zu verschlechtern. Dies resultiert in alternativen Ergebnissen, die für Passagiere potenziell einfachere Reisen ermöglichen und trotzdem mögliche Verspätungen berücksichtigen.

## Literatur

- [BGM10] Berger, A.; Grimmer, M.; Müller-Hannemann, M.: Fully Dynamic Speed-Up Techniques for Multi-criteria Shortest Path Searches in Time-Dependent Networks. In: SEA. 2010.
- [BJ04] Brodal, G.; Jakob, R.: Time-dependent Networks as Models to Achieve Fast Exact Time-table Queries. Proceedings of the 3rd Workshop on Algorithmic Methods and Models for Optimization of Railways (ATMOS'03), Electronic Notes in Theoretical Computer Science/, 2004.
- [Di18] Dibbelt, J.; Pajor, T.; Strasser, B.; Wagner, D.: Connection Scan Algorithm. ACM J. Exp. Algorithmics 23/, Okt. 2018, ISSN: 1084-6654, URL: <https://doi.org/10.1145/3274661>.
- [Di59] Dijkstra, E. W.: A Note on Two Problems in Connexion with Graphs. Numerische Mathematik 1/, S. 269–271, 1959.
- [DKP12] Delling, D.; Katz, B.; Pajor, T.: Parallel Computation of Best Connections in Public Transportation Networks. ACM J. Exp. Algorithmics 17/, Okt. 2012, ISSN: 1084-6654, URL: <https://doi.org/10.1145/2133803.2345678>.
- [DMS08] Disser, Y.; Müller-Hannemann, M.; Schnee, M.: Multi-Criteria Shortest Path in Time Dependent Train Networks. Proceedings of the 7th Workshop on Experimental Algorithms (WEA'08), Lecture Notes in Computer Science/, S. 347–361, 2008.
- [DPW15] Delling, D.; Pajor, T.; Werneck, R. F.: Round-based public transit routing. Transportation Science/, S. 591–604, 2015.
- [DW09] Delling, D.; Wagner, D.: Time-Dependent Route Planning. In (Ahuja, R. K.; Möhring, R. H.; Zaroliagis, C. D., Hrsg.): Robust and Online Large-Scale Optimization: Models and Techniques for Transportation Systems. Springer Berlin Heidelberg, Berlin, Heidelberg, S. 207–230, 2009, URL: [https://doi.org/10.1007/978-3-642-05465-5\\_8](https://doi.org/10.1007/978-3-642-05465-5_8).
- [Py08] Pyrga, E.; Schulz, F.; Wagner, D.; Zaroliagis, C.: Efficient models for timetable information in public transportation systems. ACM Journal of Experimental Algorithmics/, 2008.