Karlsruhe Institute of Technology

Umlegeverfahren für öffentliche Verkehrsnetze mittels minimal erwarteter Ankunftszeit

Bachelorarbeit
von

Holger Ebhart

An der Fakultät für Informatik
Institut für Theoretische Informatik

Erstgutachter:	Prof. Dr. Dorothea Wagner
Zweitgutachter:	Prof. Dr. rer. nat. Peter Sanders
Betreuende Mitarbeiter:	Dipl.-Inform. Ben Strasser
	M.Sc. Tobias Zündorf

Bearbeitungszeit: 15. Mai 2016 – 14. September 2016

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Bachelorarbeit selbstständig und ohne die Verwendung anderer als der gegebenen Quellen verfasst habe.

Karlsruhe, 12. September 2016 Holger Ebhart

Abstract - Public Transit Assignement based upon Minimum Expected Arrival Time

This thesis presents a new approach for public transit assignement. This means: given a schedule and a request, we want to know how much passengers use which train. Therefore, we first model the public transit network and the request in the network. Later on, this will be the input for the present algorithm.
The algorithm is based upon minimum expected arrival times. It takes the input from above and executes a simulation. Thereby each passenger is moved in the network towards his target. If all passengers have reached their targets, we know which passenger used which train and how much passengers use which train.
To show the effect of the algorithm, we conduct a case study. In our case study, we use an instance of the public transit network of Greater Stuttgart. To analyse the algorithm, we have done several run time measurements. We analyse the quality of the results by checking characteristic parameters. Furthermore we compare our results with an assignement, generated by an existing tool named visum [vis]. As our case study shows a huge improvement in run-time, also the quality of the assignement is pretty well.
To conclude, it is to announce that the presented algorithm solves the given assignement problem with real-world instances fast and precise.

Zusammenfassung - Umlegeverfahren für öffentliche Verkehrsnetze mittels minimal erwarteter Ankunftszeit

Als Verkehrsplaner stellt man sich oft die Frage: Wie viele Personen fahren in welchem Zug? Diese Frage möchte man für eine gegebene Nachfrage und einen Fahrplan lösen. In dieser Arbeit wird dafür ein neuer Algorithmus vorgestellt. Dieser löst das Problem mit einer kürzeren Laufzeit als bisher.
Um die Eingabe zu konstruieren modellieren wir zuerst ein öffentliches Personen-Verkehrsnetzwerk. Außerdem benötigen wir noch ein Nachfrage-Modell.
Der Algorithmus nimmt diese beiden Eingaben und führt darauf eine Simulation durch. Dabei werden einzelne Fahrgäste durch das Netzwerk zu ihrem Ziel geroutet. Als Grundlage dafür dient die minimal erwartete Ankunftszeit. Der präsentierte Algorithmus zeigt als Ergebnis welcher Fahrgast welchen Zug benutzt. Im Vergleich zu schon verfügbaren Tools konnten wir die Laufzeit einer Umlegung verbessern.
Um die Effizienz zu zeigen führen wir eine Fallstudie durch. Als Instanz dient uns ein Fahrplan des Großraum Stuttgarts und die entsprechende Nachfrage. Wie die Experimente zeigen, erreicht der Algorithmus eine gute Laufzeit und erzielt qualitativ hochwertige Ergebnisse. Zur Beurteilung der Ergebnisse schauen wir uns die relevanten Kenngrößen der Umlegung an. Außerdem führen wir einen Vergleich mit der Umlegung des Tools visum [vis] durch.
Wir stellen damit fest, dass der vorgestellte Algorithmus mit realen Instanzen eine schnelle Laufzeit erzielt. Ebenfalls wird eine hohe Qualität der Ergebnisse erreicht.

Inhaltsverzeichnis

1. Einleitung

1.1. Überblick

Diese Arbeit befasst sich mit einem effizienten Algorithmus zu Umlegeverfahren für die Verkehrsplanung in öffentlichen Verkehrsnetzen. Diese beinhalten alle liniengebundenen Verkehrsmittel wie zum Beispiel Zug, Bus und Fußwege. Wir wollen bestimmen, welche Züge eine Menge an Fahrgästen benutzt um jeweils von ihrem Start zu ihrem Ziel zu kommen. Dies wird als Umlegung bezeichnet. Ein Beispiel dazu wäre ein Fahrgast, der um 9:00 Uhr von Karlsruhe nach Stuttgart fahren möchte. Er hat nun die Möglichkeit verschiedene Züge zu nehmen, zum Beispiel den ICE um 9:10 Uhr, den RE um 9:19 Uhr oder einen RE und IC um 9:19 Uhr. Dabei ist ein öffentliches Verkehrsnetz gegeben, sowie die Nachfrage der Fahrgäste in diesem Netz. Die Frage, die sich einem Verkehrsplaner stellt, ist unter anderem: Wie voll wird welcher Zug?
Unser Ziel ist es nun zu bestimmen, welchen Zug ein Fahrgast wie wahrscheinlich wählt. Dazu geben wir in dieser Arbeit einen effizienten Algorithmus an, der dieses Problem für viele Fahrgäste gleichzeitig löst.

Wir geben in Kapitel 2 einen Überblick über die verwendete Modellierung des Netzes und über einen Algorithmus auf den wir aufbauen werden. In Kapitel 3.1 stellen wir unseren Algorithmus vor. Den Algorithmus evaluieren wir anhand realer Daten in Kapitel 4. Abschließend ziehen wir in Kapitel 5 ein Fazit und geben einen Überblick über weiterführende Arbeiten.

1.2. Verwandte Arbeiten

Mit der Verkehrsumlegung im Individual-Verkehr (IV) - vor allem dem Auto - haben sich schon etliche Arbeiten beschäftigt. In [FV05] wird ein Überblick über verschiedene Umlegeverfahren gegeben. In [RB] wird ein Verfahren zur statischen Verkehrsumlegung vorgestellt, in [FV05] zur dynamischen Umlegung.
Die Grundlagen zur Modellierung werden in [LS11] und [Fri] sowohl für den IV als auch den öffentlichen Verkehr (ÖV) beschrieben.
Der Prozess einer Umlegung im ÖV wird in [FF02] beschrieben. Ein Modell für eine komplette Simulation des öffentlichen Verkehrs wurde in [Bri] vorgestellt und implementiert.
Als Grundlage für den hier präsentierten Algorithmus wird der Connection-Scan-Algorithmus (CSA) verwendet. Dabei wurde auf folgende Arbeiten zurückgegriffen: Aus [Wei], [DPSW13] und [DSW14] wurde die Idee des zugrunde liegenden CSA verwendet. Bei diesen Arbeiten ging es nicht um eine Umlegung, sondern um einzelne Anfragen.

1.3. Beitrag

Aktuelle Verfahren zur Verkehrsumlegung im öffentlichen Verkehr benötigen viel Rechenzeit. Diese Arbeit soll einen Weg aufzeigen, wie die Berechnungszeit drastisch verkürzt werden kann. Wir evaluieren dazu ein neues schnelles Verfahren. Dafür testen wir unseren Algorithmus anhand von Verkehrsdaten aus dem Großraum Stuttgart. Als Vergleich dient uns eine Umlegung von Visum ([vis]) auf der selben Datengrundlage.

1.4. Kurzfassung

Wir geben eine kurze Zusammenfassung des vorzustellenden Algorithmus. Dazu betrachten wir ein öffentliches Verkehrsnetzwerk inklusive Fahrplan. Die Anfragedaten stellen die Nachfrage dar, d.h. wie viele Personen zu welchem Zeitpunkt von wo nach wo fahren möchten.
Der vorgestellte Algorithmus berechnet damit wie voll welcher Zug des Fahrplans wird. Wir erhalten also pro Verbindung die Fahrgäste, die diese Verbindung benutzen.
In folgenden vier Schritten führt unser Algorithmus diese Berechnung durch. Eine genauere Beschreibung des Algorithmus findet sich in Kapitel 3. Jeder der nun folgenden Schritte wird für jedes Ziel ausgeführt:

MEAT- und Profilberechnung Zuerst führen wir eine Vorberechnung für die spätere Umlegung durch. Dazu bewerten wir jeden Zug für das aktuelle Ziel. Damit geben wir an, wie günstig es ist mit diesem Zug an das Ziel zu kommen. Für jede Haltestelle nehmen wir die am Besten bewerteten Züge und speichern diese ab.

Zonen-Stopp-Fußwege bestimmen Unsere Nachfragedaten beziehen sich nicht auf Haltestellen, sondern auf größer gefasste Räume (später Zonen). Von diesen Räumen gibt es Fußwege hin zu Haltestellen. In diesem Schritt verteilen wir die Personen aus den Räumen über diese Fußwege auf Start-Haltestellen. Dabei betrachten wir pro Schleifeniteration nur Personen mit dem selben Ziel.

Umlegung Wir betrachten die Fahrgäste an ihren initialen Start-Haltestellen und verteilen sie ausgehend von diesen auf die möglichen Züge. Dazu iterieren wir über alle Züge und benutzen zum einen die Bewertungen der verschiedenen Züge und die bevorzugten Züge von einer Haltestelle aus. Aufgrund dieser Bewertungen entscheiden wir dann welche Fahrgäste wir in welchem Zug mitnehmen und wann sie wieder aussteigen.

Zyklen-Elimination Im letzten Schritt werden unnötige Zyklen aus den Fahrten der Fahrgäste entfernt. Damit garantieren wir, dass kein Passagier unsinnig im Kreis fährt.

In einem Akkumulationsschritt am Ende jeder Iteration summieren wir die Fahrgäste pro Zug auf. Der Pseudo-Code in 1.1 veranschaulicht den Ablauf nochmals.

Algorithmus 1.1: Ablauf des Algorithmus

```
1 forall Ziele do
2     MEAT- und Profilberechnung;
3     Zonen-Stopp-Fußwege bestimmen;
4     Umlegung;
5     Zyklen eliminieren;
6     Ergebnisse akkumulieren;
```

Einige Schwierigkeiten bei der Entwicklung des Algorithmus sind hier beschrieben:
Intuitiv ist klar, dass ein Fahrgast nicht 10 mal umsteigen möchte, wenn er von Karlsruhe

nach Stuttgart fährt um 1 Minute früher anzukommen. Das heißt, wir können nicht einfach die schnellste Verbindung für einen Fahrgast auswählen, sondern wir müssen eine möglichst schnelle Verbindung mit wenigen Umstiegen wählen. Dazu verschlechtern wir die Bewertung einer Verbindung für jeden Umstieg der gemacht werden muss. Wir erreichen dadurch eine Optimierung der ausgewählten Verbindung hinsichtlich Ankunftszeit und Anzahl an Umstiegen.
Problematisch sind auch die Umsteigezeiten an den Stopps. Um bei einer Verbindungsplanung nicht an jedem Stopp explizit die Umsteigezeit zu berücksichtigen, modellieren wir die Umsteigezeit als Fußweg von einem Stopp hin zum selben Stopp. Dadurch ist es uns möglich Umsteigezeiten äquivalent zu Fußwegen zu behandeln.
Diese können wir allerdings nicht wie Verbindungen betrachten (sie können jederzeit begangen werden). Deshalb bauen wir einen Fußweggraph auf und beziehen diesen in die Bewertung der einzelnen Verbindungen mit ein. Bei der eigentlichen Umlegung betrachten wir den Fußweggraph nur noch um Erreichbarkeiten festzustellen. Die Dauer interessiert uns in diesem Schritt nur sekundär (genaueres dazu in Kapitel 3.4).
Betrachten wir Passagiere, die eine längere Zeit an einem Stopp stehen, so stellen wir fest, dass diese Passagiere anstatt zu warten Schleifen fahren. Dies möchten wir in unserer Umlegung vermeiden. Dazu fassen wir Stopps zu Stationen zusammen und eliminieren Zyklen bzw. Schleifen in Fahrten von einzelnen Passagieren. Eine genaue Beschreibung der Lösungsidee findet sich in 3.3.3.

2. Grundlagen

2.1. Modellierung des öffentlichen Verkehrsnetzes

In diesem Kapitel formalisieren wir die Eingabe für eine Umlegung und geben wichtige Definitionen und Begriffserklärungen an. Dazu gehört ein Fahrplan und eine Nachfrage. Der Fahrplan ist aus mehreren Objekten aufgebaut. Dazu gehören Stopps, Zonen, Connections, Trips und Fußwege.

Wir stellen in Unterkapitel 2.1.1 einen vereinfachten Fahrplan vor und erweitern diesen in Unterkapitel 2.1.2 um Fußwege. Die Nachfrage formalisieren wir in Unterkapitel 2.1.3.

2.1.1. Grundlegender Fahrplan

Unser vereinfachter Fahrplan ohne Fußwege beinhaltet diese Objekte:

- Stopps
- Zonen
- Zonen-Stopp-Fußwege
- Connections
- Trips

Stopp

Ein Stopp ist eine geographisch feste Position, an der ein Reisender stehen kann. An Stopps ist es Reisenden möglich das Verkehrsmittel zu wechseln (Ein-, Aus- oder Umstieg). Ein Stopp hat die Attribute:

- Name
- geographische Koordinaten
  - Längengrad
  - Breitengrad
- Mindest-Umsteigezeit

Die Umsteigezeit gibt an wie lange ein Reisender an diesem Stopp mindestens braucht um das Verkehrsmittel zu wechseln.

Beispiel

Der Hauptbahnhof in Karlsruhe ist ein Stopp. Sein Name ist *Karlsruhe Hbf*. Er hat die Koordinaten *8° 39921' östliche Länge, 48° 9903' nördliche Breite*. Die Umsteigezeit beträgt *1 Minute*.

Aufgrund der Umsteigezeit kann ein Reisender der um 7:20 Uhr ankommt nicht mit einem Zug um 7:20 Uhr weiterfahren. Er kann frühestens einen Zug um 7:21 Uhr erreichen.
Es ist zu beachten, dass es mehrere Stopps mit dem selben Namen geben kann: Zum Beispiel einen für Fernverkehrszüge, einen für die S-Bahnen und einen für Busse.

Zone

Mehrere Stopps fassen wir zu einer Zone zusammen, welcher folgende Attribute zugeordnet werden:

- geographische Koordinaten (Zentrum der Zone)
  - Längengrad
  - Breitengrad

Die hier definierten Zonen entsprechen im Allgemeinen nicht den bekannten Tarifwaben / -zonen eines Verkehrsverbundes. Eine Zone bekommt ihre Stopps implizit über die Zonen-Stopp-Fußwege zugewiesen.

Beispiel

Ein kleiner Ort wie *Nussdorf* kann als eine Zone aufgefasst werden. Dabei sind die zugewiesenen Stopps die Bushaltestellen im Ort.

Zonen-Stopp-Fußwege

Ein Zonen-Stopp-Fußweg beschreibt einen Fußweg aus einer Zone heraus zu einem Stopp. Dieser Fußweg kann in beide Richtungen begangen werden. Er hat die Attribute:

- Zone
- Stopp
- Dauer

Die Dauer beschreibt wie lange ein Reisender vom Zentrum der Zone hin zu einem Stopp oder vom Stopp zum Zentrum der Zone läuft. Wird ein solcher Fußweg begangen, bezeichnen wir dies als **Transfer**. Ein Stopp kann in keiner, einer oder mehreren Zonen liegen. Eine Zone kann keinen, einen oder mehrere Zonen-Stopp-Fußwege enthalten.

Beispiel

In Abbildung 2.1 sind beispielhaft Zonen-Stopp-Fußwege zu sehen.

Connection

Wir betrachten nun Verbindungen, die wir von nun an Connections nennen und weisen ihnen folgende Attribute zu:

- Abfahrtsstopp
- Ankunftsstopp

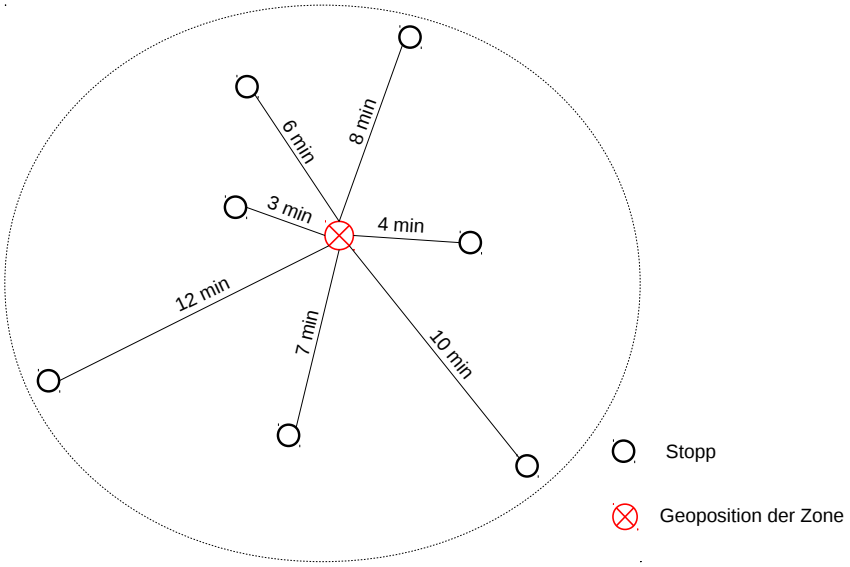


Abbildung 2.1.: Beispiel für Zonen-Stopp-Fußwege mit Dauer der Fußwege

- Abfahrtszeit
- Ankunftszeit
- Trip

In Abbildung 2.2 sind die Attribute visualisiert. Eine Connection ist eine elementare Verbindung ohne Zwischenstopps.

Beispiel

Eine Connection ist die Fahrt des RE 19545 von *Karlsruhe Hbf* nach *Karlsruhe Durlach*. Die Abfahrtszeit ist *17:19 Uhr* und die Ankunftszeit ist *17:23 Uhr*. Der Trip ist die gesamte Zugfahrt des RE 19545 von Karlsruhe Hbf nach Stuttgart Hbf.

Trip

Ein Trip besteht aus mehreren zeitlich sortierten Connections des selben Verkehrsmittels. Die Connections sind in der Abfahrtszeit aufsteigend angeordnet. Die Abfahrtszeit der nächsten Connection ist stets größer oder gleich der Ankunftszeit der vorhergehenden Connection. Außerdem entspricht der Abfahrtsstopp der nächsten Connection dem Ankunftsstopp der vorhergehenden Connection.

Beispiel

Eine Zugfahrt von Stuttgart nach Karlsruhe mit einem IRE ist ein Trip. Damit ist *Stuttgart Hbf* der Startstopp und *Karlsruhe Hbf* der Endstopp. Eine Auflistung aller beteiligten Connections findet sich in Tabelle 2.1.

2.1.2. Erweiterter Fahrplan

In diesem Kapitel werden wir Fußwege zwischen Stopps einführen. Damit wird es möglich an Stopp a auszusteigen, zu Stopp b zu laufen und dort in den nächsten Zug einzusteigen. Wir nennen diese Wege **Transfer-Fußwege**:

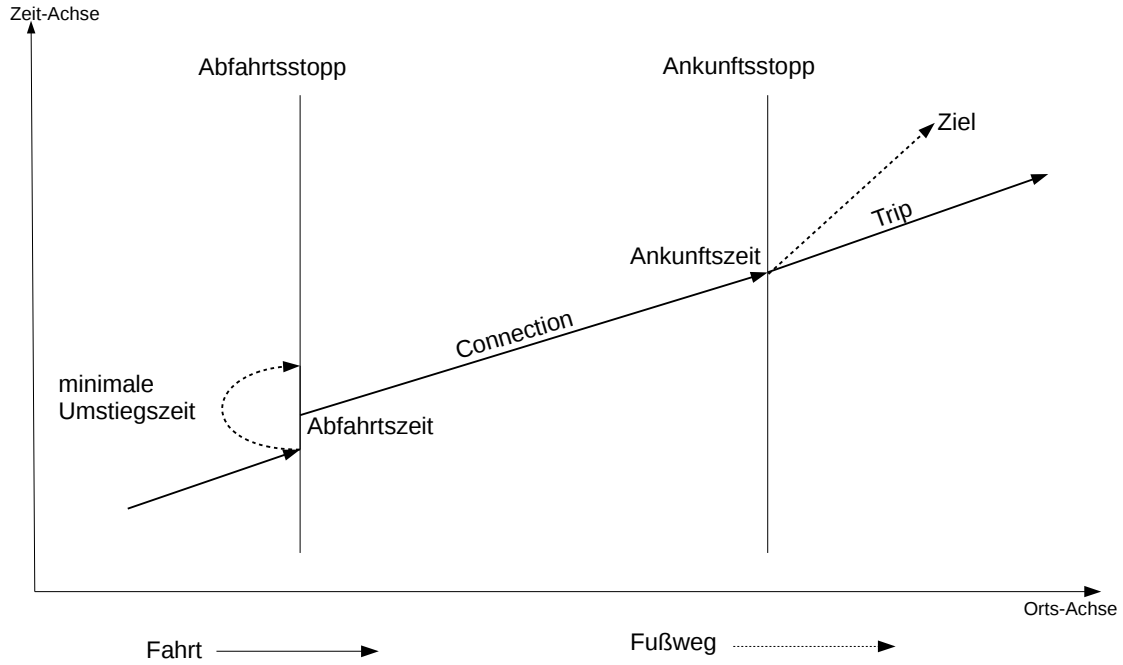


Abbildung 2.2.: Visualisierung der Attribute einer Connection. In x-Richtung sind die Stopps aufgetragen, in y-Richtung die Zeit.

Tabelle 2.1.: Beispiel für einen Trip: IRE 4906 von Stuttgart nach Karlsruhe

Abfahrtsstopp	Abfahrtszeit	Ankunftsstopp	Ankunftszeit	Dauer [min]
Stuttgart Hbf	12:59	Vaihingen/Enz	13:15	16
Vaihingen/Enz	13:15	Mühlacker	13:21	6
Mühlacker	13:21	Pforzheim	13:30	9
Pforzheim	13:30	Karlsruhe Durlach	13:47	17
Karlsruhe Durlach	13:47	Karlsruhe Hbf	13:53	6

Grob gesagt, werden wir die Fußwege als Graph modellieren, wobei die Stopps die Knoten sind und die Fußwege gerichtete Kanten. Die Gewichte der Kanten entsprechen der Dauer des Fußweges. Wir fordern, dass Stopps, die über Fußwege miteinander verbunden sind, Cliquen sind. Eine Clique ist ein Teilgraph in dem jedes Knotenpaar durch eine Kante verbunden ist. Dadurch sind die Fußwege transitiv abgeschlossen. Des weiteren fordern wir, dass die Dreiecksungleichung gilt: $\forall a, b, c$ aus einer Clique gilt: $\Delta t_{a\to b} + \Delta t_{b\to c} \geq \Delta t_{a\to c}$. Einen Spezialfall stellen Umsteigezeiten an Stopps dar. Wir werden sie auch als Fußwege modellieren, indem wir Schleifen einfügen. Das Gewicht einer Schleife ist die Umsteigezeit an dem Stopp.

Beispiel

Ein Beispiel-Graph mit Fußwegen ist in Abbildung 2.3 zu sehen.
Benutzt eine Person einen Fußweg von Stopp a nach Stopp b, so sprechen wir von einem **Transfer von a nach b**. Dabei sei auch $a = b$ zugelassen, was bedeutet dass ein Reisender an einem Stopp umsteigt.

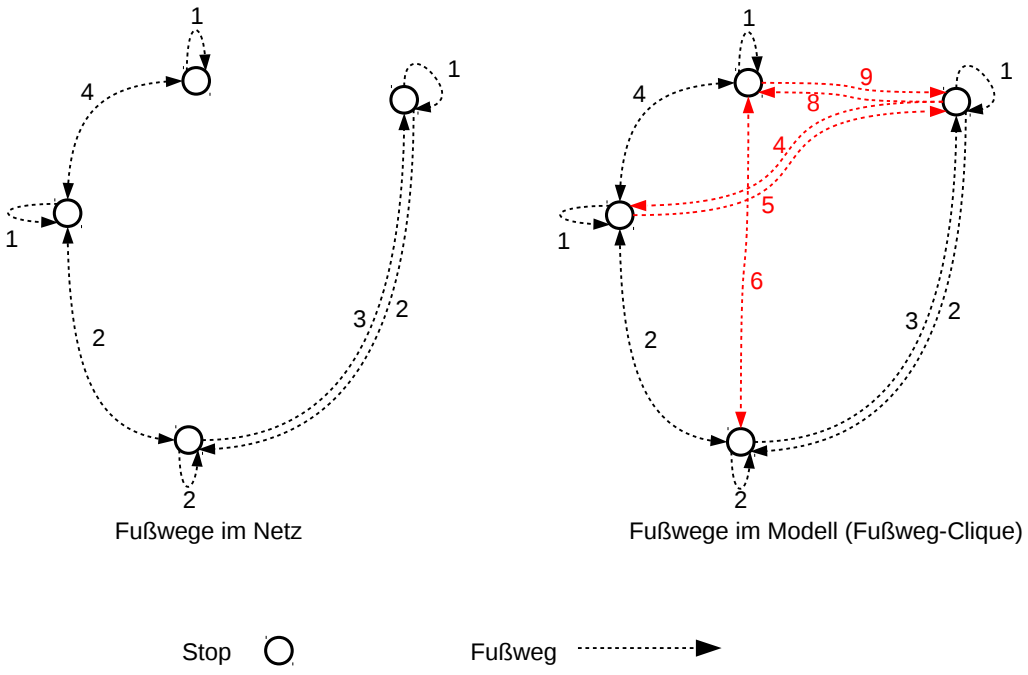


Abbildung 2.3.: Beispiel für eine Fußwege-Clique und deren Abbildung im Modell

Station

Eine Station ist eine Menge per Fußweg verbundener Stopps. Damit sind Cliquen im Fußwegegraph identisch zu Stationen.

Beispiel

In Abbildung 2.3 bilden alle vier abgebildeten Stopps eine Station.

Unser erweiterter Fahrplan enthält genau folgende Objekte:

- Stopps
- Zonen
- Connections
- Trips
- Zonen-Stopp-Fußwege
- Transfer-Fußwege

2.1.3. Nachfrage

Den gewünschten Ortswechsel einer oder mehrerer Personen bezeichnen wir als Nachfrage. Dabei beschreiben wir die Nachfrage durch eine Menge folgender Quadrupel:

- Startzone
- Zielzone
- Startzeit
- Anzahl an Personen

Wir gehen davon aus, dass die Anzahl an Personen stets größer 0 ist.

Tabelle 2.2.: Beispiel für eine Nachfrage bestehend aus mehreren Nachfrage-Quadrupeln

Abfahrtszeit	Abfahrtszone	Zielzone	Anzahl an Personen
07:00	Stuttgart Stadtmitte	Vaihingen/Enz	1
07:10	Nussdorf	Stuttgart Vaihingen	5
07:35	Winnenden Ortsmitte	Waiblingen Bahnhof	50
07:35	Plochingen	Affalterbach	10
08:05	Waldenbuch	Fellbach Mitte	3
12:45	Stuttgart Sillenbuch	Renningen	6

Beispiel

In Tabelle 2.2 ist beispielhaft eine Nachfrage bestehend aus mehreren Nachfrage-Quadrupeln aufgelistet.

2.1.4. Terminologie

Für eine einheitliche Terminologie führen wir noch einige Begrifflichkeiten ein:

Ride Eine Ride bezeichnet mehrere Connections eines Trips, welche direkt hintereinander liegen. Dann ist eine Ride durch eine Einstiegs-Connection und eine Ausstiegs-Connection eindeutig definiert. Die Einstiegs-Connection ist die erste benutzte Connection der Ride und die Ausstiegs-Connection die letzte Benutzte. Einstiegs- und Ausstiegs-Connection sind aus dem selben Trip und die Ausstiegs-Connection liegt zeitlich nach (oder gleich mit) der Einstiegs-Connection.

Journey Eine Journey entspricht einer umgangssprachlichen Reise. Dies ist eine Verknüpfung von Rides, sodass wir von einem Startstopp zu einem Zielstopp kommen. Die Rides einer Journey sind in zeitlich aufsteigender Reihenfolge verknüpft. Dabei ist stets gewährleistet, dass die Anschluss-Ride erreicht wird. Das bedeutet für die Abfahrtszeit der nächsten Ride in der Journey: $\textit{Abfahrtszeit} \geq \textit{Ankunftszeit} + \textit{Mindest-Umsteigezeit}$.

In Abbildung 2.4 verdeutlichen wir diese Begriffe an einem Beispiel.

dominierende Connection Eine dominierende Connection ist die späteste abfahrende Connection um frühest möglichst anzukommen. Connections die früher abfahren und die gleiche oder eine spätere erwartete Ankunftszeit (MEAT) haben, werden von ihnen dominiert.

Pareto-optimal Ein Zustand ist Pareto-optimal, wenn die Verbesserung einer Eigenschaft die Verschlechterung einer anderen Eigenschaft nach sich zieht. Somit sind in einem Pareto-optimalen Zustand alle Eigenschaften optimal.

Um unser Modell weiter der Realität anzupassen, werden wir Verspätungen der Verkehrsmittel berücksichtigen. Dies hat Einfluss auf die spätere Umlegung, wodurch diese präziser wird. Wir begrenzen die Verspätung im von uns modellierten Verkehrsnetz auf eine maximal mögliche Verspätung. Verspätungen von mehr als der maximal Möglichen betrachten wir nicht weiter.

Verkehrsmittel wie Züge oder Busse haben eine begrenzte Kapazität. Es passt nur eine begrenzte Anzahl Personen in einen Bus. Wir wählen in dieser Arbeit die Kapazität beliebig groß. Damit können beliebig viele Personen in einem Verkehrsmittel fahren. In Kapitel 5.1 geben wir einen Ausblick auf die Kapazitätsbeschränkung.

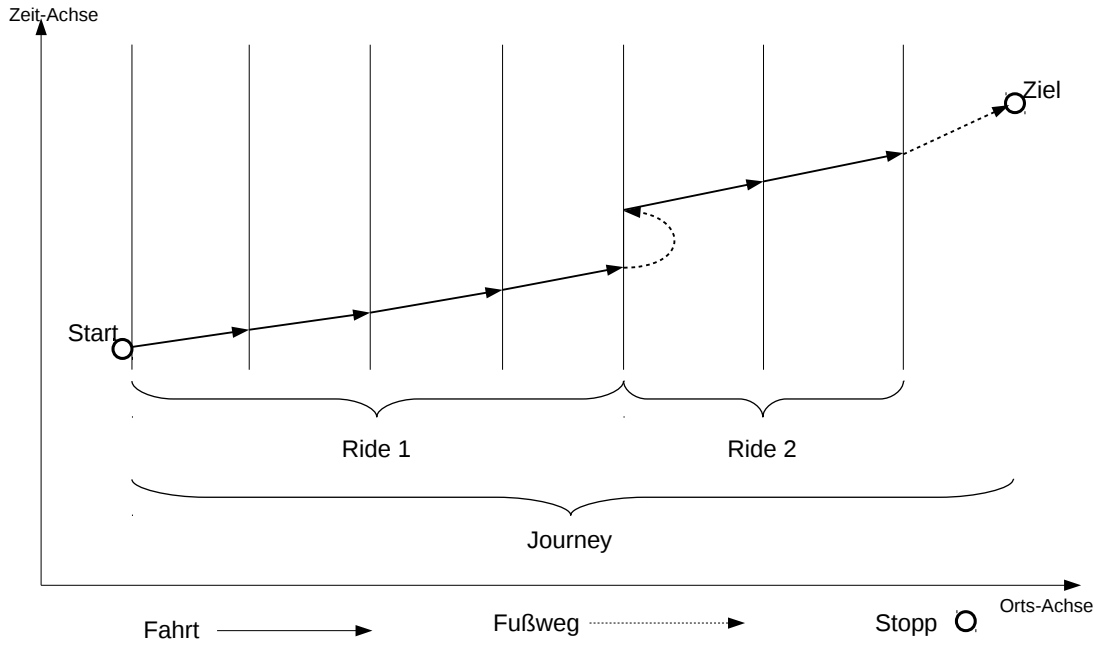


Abbildung 2.4.: Beispiel einer Journey

2.2. Umlegung

In dieser Arbeit verwenden wir ausschließlich eine fahrplanfeine Umlegung (siehe [Fri]). Dies bedeutet, dass wir anhand eines Fahrplans bestimmen, welche Journeys eine Menge an Fahrgästen wählt. Für weitere Umlegungsmodelle sei auf [Fri] verwiesen.
Während unserer Umlegung kann ein Fahrgast die vier folgenden Zustände annehmen:

- Fahrgast befindet sich auf dem Weg zum ersten Abfahrtsstopp
- Fahrgast wartet an einem Stopp auf eine Ride
- Fahrgast sitzt in einer Connection
- Fahrgast ist auf dem finalen Stopp-Zonen Fußweg zum Ziel bzw. am Ziel

Wir gehen von einem nicht perfekten Verhalten der Fahrgäste aus. Dies entspricht intuitiv auch der Realität. Es bedeutet, dass ein Fahrgast nicht immer die für ihn beste Verbindung bezüglich der Ankunftszeit bzw. Umstiege auswählt.

Beispiel

Hat ein Fahrgast die Wahl zwischen zwei Journeys wobei die erste um 9:00 Uhr und die zweite um 9:01 Uhr ankommt, so wird er sich mit einer Wahrscheinlichkeit von ungefähr $\frac{1}{2}$ für die erste und mit einer Wahrscheinlichkeit von ungefähr $\frac{1}{2}$ für die zweite Journey entscheiden. Journey 1 ist objektiv die bessere Wahl, aber der Unterschied ist sehr gering. Damit ist der Nachteil nicht unverhältnismäßig groß und ein Fahrgast wird beide Journeys fast gleich bewerten.

2.3. Minimal Erwartete Ankunftszeit

Unser Algorithmus für die Umlegung basiert auf der sogenannten minimal erwarteten Ankunftszeit (MEAT, engl.: minimal expected arrival time).

Um deren Bedeutung zu verstehen, schauen wir uns an wie wir sie berechnen. Die MEAT bezieht sich immer auf ein Ziel. Aus diesem Grund führen wir für jedes Ziel eine extra Berechnung durch.
Betrachten wir zuerst den Basisfall: Eine Connection, deren Ankunftsstopp dem Ziel entspricht, hat als MEAT ihre Ankunftszeit.
Für die übrigen Connections gilt, dass sich deren MEAT aus dem Minimum der beiden folgenden Werte ergibt:

Trip-MEAT Gibt es im Trip nachfolgende Connections, so entspricht die Trip-MEAT der MEAT der nachfolgenden Connection im Trip. Ist die betrachtete Connection die letzte eines Trips, so ist die Trip-MEAT ∞.

Arrivalstop-MEAT Hierbei werden sämtliche am Ankunftsstopp erreichbaren Connections betrachtet. Jede dieser Connections hat einen MEAT-Wert. Aus diesen wird dann eine gewichtete Summe gebildet, welche den neuen MEAT-Wert darstellt. Die Gewichte werden durch den MEAT-Wert und die Abfahrtszeit bestimmt.

Damit bekommen wir für jede Connection eine MEAT, die einem Erwartungswert entspricht. Die Umsteigezeiten, Anzahl an Transfers, mögliche Verspätungen und nicht optimale Entscheidungen der Fahrgäste fließen in die Berechnung des MEAT-Wertes mit ein.

2.4. Profile Connection-Scan-Algorithmus

Zur Berechnung der MEAT-Werte verwenden wir eine Variante des Connection-Scan-Algorithmus aus [Wei] und [DPSW13]:

Dieser Algorithmus benötigt als Eingabe eine Liste aller Connections. Diese Liste ist aufsteigend nach den Abfahrtszeiten der Connections sortiert. Als Ergebnis liefert der CSA pro Stopp ein Profil mit dominierenden Connections. Dominierende Connections sind immer Pareto-optimal bezüglich Abfahrtszeit und MEAT. Jedes Profil stellt eine nach MEAT-Werten und Abfahrtszeiten aufsteigend sortierte Liste dar.
Der Algorithmus führt für alle Connections absteigend in der Abfahrtszeit zwei Berechnungsschritte durch:

Berechnung des MEAT-Wertes Um den MEAT-Wert der Connection zu berechnen, betrachten wir folgende drei Werte:

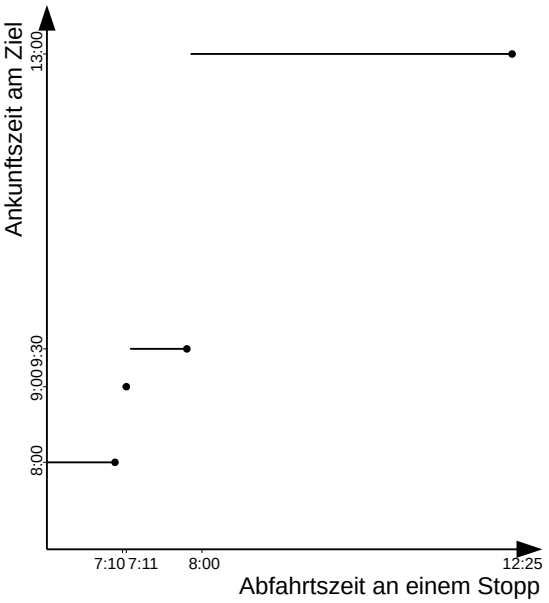
- Den MEAT-Wert der nachfolgenden Connection des selben Trips (∞ falls solch eine Connection nicht existiert).
- Die Ankunftszeit der Connection, falls der Ankunftsstopp dem Zielstopp entspricht (∞ falls nicht).
- Eine gewichtete Summe der MEAT-Werte des Profils am Ankunftsstopp der Connection. Die Gewichtung hängt von der Erreichbarkeit der Connections, deren MEAT-Werten und der Abfahrtszeit ab.

Der MEAT-Wert der Connection ergibt sich aus dem Minimum dieser drei Werte.

Aufbau des Profils Den gerade berechneten MEAT-Wert vergleicht man mit dem MEAT-Wert des letzten Eintrags im Profil am Abfahrtsstopp. Ist er strikt kleiner, so fügt man einen neuen Eintrag mit dem MEAT-Wert und der Abfahrtszeit der Connection in das Profil ein. Falls die Abfahrtszeit der Connection gleich der Abfahrtszeit im letzten Eintrag des Profils ist, so wird der letzte Eintrag durch den Neuen ersetzt. Andernfalls gehen wir zur nächsten Connection weiter.

Abbildung 2.5.: Beispielhaftes Profil an einem Stopp

Abfahrtszeit	MEAT	dominierend
~~07:00~~	~~08:05~~	✗
07:10	08:00	✓
07:11	09:00	✓
~~07:40~~	~~09:30~~	✗
~~08:00~~	~~09:45~~	✗
08:00	9:30	✓
~~12:45~~	~~13:02~~	✗
12:25	13:00	✓



Hat der Algorithmus jede Connection betrachtet, dann liegt pro Stopp ein Profil mit MEAT-Werten vor. Dieses Profil enthält nur sogenannte dominierende Connections. Diese unterteilen die Zeit in disjunkte Intervalle $(a, b]$, wobei a und b Abfahrtszeiten von Connections sind. Jede dominierende Connection stellt eine spätest mögliche Abfahrt für ein Intervall dar, um frühest möglich ans Ziel zu kommen. Dominierte Connections tauchen in einem Profil nicht auf. In Abbildung 2.5 findet sich ein Beispiel eines Profils.

Die Ergebnisse des Algorithmus hängen maßgeblich von der Gewichtung der MEAT-Werte ab. Die MEAT einer Connection wird höher gewichtet, je größer die Umsteigezeit ist. Wir betrachten nur dominierende Connections, daher sind wir stets optimal bezüglich der MEAT.

3. Verkehrsumlegung

In diesem Kapitel stellen wir einen Algorithmus vor, der aus einem Fahrplan und einer Nachfrage für ein öffentliches Verkehrsnetz eine Umlegung berechnet. Diese Umlegung gibt an, welche und wie viele Fahrgäste welche Connections benutzen.

3.1. Lösungsansatz

Wir stellen unseren Ansatz zur Verkehrsumlegung vor. Dieser basiert auf minimal erwarteten Ankunftszeiten. Dabei berechnen wir die MEAT-Werte mittels einer abgewandelte Version des CSA-Algorithmus. Anschließend findet die Umlegung der Passagiere anhand dieser MEAT-Werte statt.

In 3.2 wird der grundlegende Ablauf des Verfahrens dargelegt. Dieser wird dann in 3.3 verfeinert und schließlich in 3.4 um Fußwege erweitert.

3.2. Idee

Die Eingabe unseres Algorithmus ist ein Fahrplan, wie in 2.1.2 definiert, und eine Nachfrage wie in 2.1.3 definiert.
Für jede Zielzone führt der Algorithmus die fünf folgenden Schritte aus:

1. Zuerst findet die MEAT- und Profilberechnung statt. In dieser werden die einzelnen Connections bewertet und die dominierenden Connections pro Stopp markiert.
2. Danach legen wir die Fahrgäste auf die initialen Zonen-Stopp-Fußwege um. Nun stehen die Fahrgäste an ihren Start-Stopps.
3. Wir können jetzt die Umlegung auf einzelne Connections durchführen. Damit wissen wir welche Fahrgäste welche Connection benutzen.
4. Da der Algorithmus keine Zyklenfreiheit der Journeys garantiert, werden wir im letzten Schritt Zyklen aus den Journeys der einzelnen Fahrgäste entfernen.
5. Sind diese Schritte für alle Zielzonen durchgeführt worden, akkumulieren wir noch die Ergebnisse. So erhalten wir eine Umlegung für die gesamte Nachfrage.

Im Folgenden geben wir nochmals einen Überblick über die einzelnen Schritte. Eine detaillierte Erläuterung folgt im Anschluss.

MEAT- und Profilberechnung Aufbauend auf dem Fahrplan und den Fußwegen wird für jede Connection der MEAT-Wert berechnet. Dies ist der Erwartungswert für die früheste Ankunft am Ziel, unter der Voraussetzung, dass man die aktuelle Connection benutzt. Aus diesen MEAT-Werten wird dann ein MEAT-Profil für jeden Stopp erstellt. Es enthält die dominierenden Connections von diesem Stopp aus. Dabei sind sowohl die Abfahrtszeiten der Connections im Profil als auch die MEAT-Werte zeitlich monoton aufsteigend.

Zonen-Stopp-Fußwege bestimmen Die Nachfrage gibt uns für die Fahrgäste nur Startzonen an. Um die Fahrgäste den Stopps zuzuweisen, nutzen wir die Zonen-Stopp-Fußwege und verteilen die Fahrgäste von der Zone über diese Fußwege an die Stopps. Dazu betrachten wir jeweils die Fußwegedauer und das Profil des Stopps.

Umlegung Für die Umlegung iterieren wir aufsteigend über die Connections und entscheiden jeweils drei Fälle:

Einsteigende Fahrgäste Am Abfahrtsstopp der Connection wird entschieden welche der am Stopp wartenden Fahrgäste in diese Connection einsteigen und welche weiterhin am Stopp warten.

Aussteigende Fahrgäste Am Ankunftsstopp der Connection wird entschieden welche der aktuell im Verkehrsmittel mitfahrenden Passagiere aus- bzw. umsteigen.

Fahrgäste am Ziel Ist es möglich vom Ankunftsstopp zu Fuß das Ziel zu erreichen, so bestimmen wir welche der aussteigenden Fahrgäste zu Fuß zum Ziel laufen.

Hat man die einsteigenden Passagiere ausgewählt, so kennt man die Passagiere in der Connection. Diese schreiben wir mit.

Zykleneliminierung Die bisherigen Schritte garantieren keine Zyklenfreiheit in Journeys von Fahrgästen[1]. In der Realität fährt jedoch kaum jemand freiwillig eine Schleife, anstatt zu warten. Daher eliminieren wir die Zyklen im letzten Schritt. Es werden die Passagiere aus den Connections entfernt, die Zyklen bilden.

Die Schritte *MEAT- und Profil-Berechnung*, *Zonen-Stopp-Fußwege bestimmen*, *Umlegung* und *Zykleneliminierung* werden für jede Zielzone durchgeführt. Nach jeder Iteration akkumulieren wir die Ergebnisse und erhalten am Schluss eine komplette fahrplanfeine Umlegung.

3.3. Algorithmus

Wir geben im Folgenden eine detaillierte Beschreibung unseres Algorithmus an. Zu Beginn beschreiben wir den Algorithmus ohne Transfer-Fußwege. Es ist also nicht möglich zwischen zwei Stopps zu Fuß zu wechseln. In Unterkapitel 3.3.1 beginnen wir mit der MEAT- und Profilberechnung, danach folgt in Unterkapitel 3.3.2 die Umlegung auf Zonen-Stopp-Fußwege. Unterkapitel 3.3.3 widmet sich der Umlegung und im letzten Unterkapitel 3.3.4 beschreiben wir die Zyklenelimination. Danach erweitern wir in Kapitel 3.4 den Algorithmus um Transfer-Fußwege.

Wir beschreiben nun die Schritte, die der Algorithmus für jede Zielzone nacheinander durchführt. Sie liegen also in einer Schleife über alle Zielzonen. Nach jeder Iteration werden die einzelnen Ergebnisse akkumuliert. Das heißt wir summieren für jede Connection die zugewiesenen Fahrgäste über alle Zielzonen auf.

[1] Muss ein Fahrgast 20 Minuten an einem Stopp warten, so kann er auch mit z.B. einem Bus zu einem anderen Stopp fahren, dort umsteigen und wieder zurückfahren. Benötigt dies weniger als 20 Minuten, so erreicht er trotzdem noch seinen Anschluss.

3.3.1. MEAT- und Profil-Berechnung

Wir berechnen in diesem Schritt die MEAT-Werte der Connections und die Profile. Dazu iterieren wir absteigend in der Abfahrtszeit über alle Connections. Für jede Connection berechnen wir folgende fünf MEAT-Werte:

Trip-MEAT Gibt an wann Fahrgäste voraussichtlich ankommen, wenn sie am Ankunftsstopp der aktuellen Connection im Verkehrsmittel sitzen bleiben. Sie nutzen also die im Trip nachfolgende Connection. Diesen Wert erhalten wir durch Auslesen des MEAT-Werts der nachfolgenden Connection. Dieser ist ∞ falls der Trip mit der Connection endet.

Target-MEAT Gibt an ob und wann man vom Ankunftsstopp aus zu Fuß am Ziel ankommt. Ist es nicht möglich vom Ankunftsstopp an das Ziel zu laufen, so ist dieser Wert ∞. Wir berechnen diesen Wert indem wir uns die Dauer von einem Zielstopp zum Ziel speichern. Dies ermöglicht eine effiziente Berechnung des Wertes.

Arrivalstop-MEAT Gibt an, wann man voraussichtlich am Ziel ankommt, wenn man die aktuelle Connection nutzt und an deren Ankunftsstopp umsteigt. Dabei werden beim Umstieg nur erreichbare Connections beachtet. Dieser Wert entspricht dem frühesten Eintrag im Profil am Ankunftsstopp. Da es hierbei zu einem Umstieg kommt, rechnen wir eine Transfer-Penalty auf die MEAT. Dies führt zur Verschlechterung von Journeys mit vielen Umstiegen.

Connection-MEAT Ist das Minimum über die drei oberen Werte *Trip-MEAT*, *Target-MEAT* und *Arrivalstop-MEAT*.

Departurestop-MEAT Gibt die voraussichtliche Ankunftszeit an, wenn wir die aktuelle Connection nicht benutzen und an deren Abfahrtsstopp warten. Dazu schreiben wir einfach den frühsten Eintrag nach Abfahrt der Connection im Profil des Abfahrt-Stopps mit.

Eine grafische Übersicht über die MEAT-Werte findet sich in 3.1.

Von den berechneten MEAT-Werten speichern wir die Trip-MEAT, die Arrivalstop-MEAT und die Departurestop-MEAT für jede Connection. Die Connection-MEAT und die Target-MEAT müssen wir nicht speichern. Sie können bei Bedarf wieder effizient berechnet werden. Anhand der Connection-MEAT aktualisieren wir zuletzt das Profil am Abfahrtsstopp. Dazu prüfen wir die MEAT des frühesten Eintrags. Ist die Connection-MEAT kleiner als dieser Wert, so wird ein neuer Eintrag mit der Connection-MEAT und der Abfahrtszeit der Connection angelegt. Damit haben wir eine dominierende Connection, die Pareto-optimal ist. Sind die Abfahrtszeiten gleich, so ersetzen wir den frühesten Eintrag durch den Neuen. Ist es nicht möglich mit einer Connection das Ziel zu erreichen, so setzen wir deren MEAT-Wert auf ∞.

3.3.2. Zonen-Stopp-Fußwege bestimmen

Die Nachfragedaten geben uns für die Fahrgäste nur Startzonen an. Deswegen verteilen wir die Fahrgäste von ihren Startzonen auf die Stopps in der Zone. Wir betrachten dazu alle Stopps in der Zone und berechnen für jeden Stopp einen MEAT-Wert wie folgt: Wir addieren die Dauer des Zonen-Stopp-Fußweges auf die Startzeit und erhalten die Ankunftszeit am Stopp. Am entsprechenden Stopp schlagen wir im Profil den Eintrag für diese Ankunftszeit nach. Er liefert uns für diesen Stopp die MEAT zur Zielzone. Wir vergleichen diese MEATs miteinander und verteilen die Fahrgäste entsprechend auf die Stopps. Das Modell für die Verteilung wird in Unterkapitel 3.3.5 beschrieben. In Abbildung 3.2 ist ein Beispiel gezeigt.

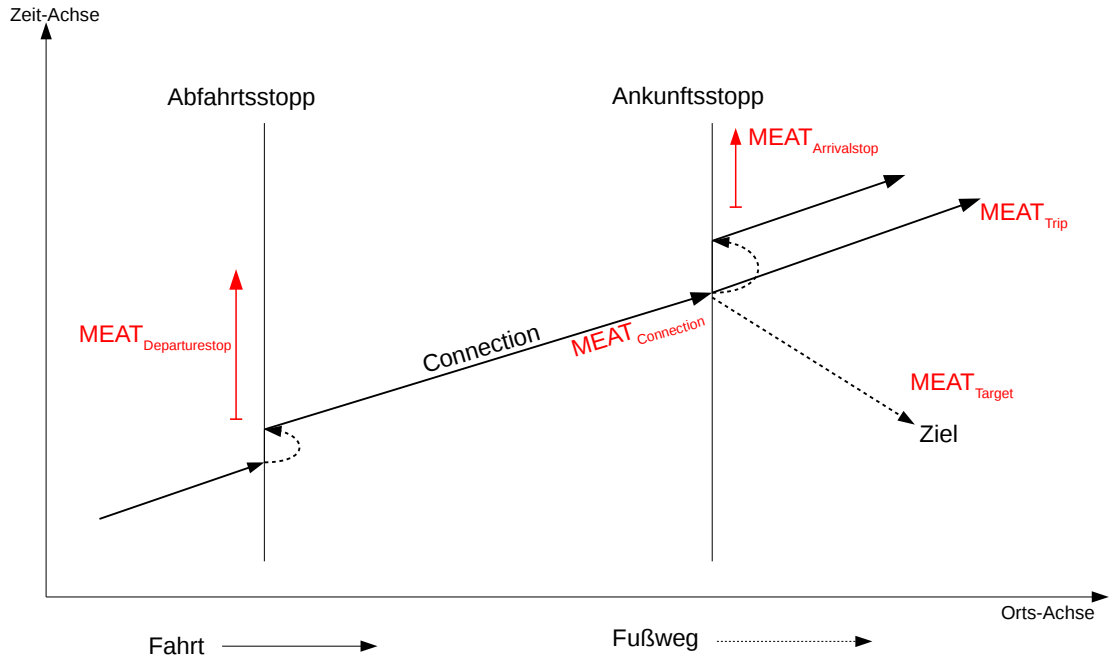


Abbildung 3.1.: Im Algorithmus verwendete MEAT-Werte

3.3.3. Umlegung

Mit den berechneten MEAT-Werten und Profilen beginnen wir jetzt mit der eigentlichen Umlegung. Zuerst schauen wir uns an welche Werte wir während der Umlegung möglichst effizient zwischenspeichern müssen. Wir simulieren in unserer Umlegung das Verhalten der einzelnen Fahrgäste.

Datenstrukturen

Wir beschreiben im Folgenden die Datenstrukturen, die für die Umlegung benötigt werden. Jeder Fahrgast bekommt eine ID zugewiesen, mit der er im Verkehrsnetz simuliert wird. Für jeden Stopp speichern wir welche Fahrgäste dort gerade warten. Für jeden Trip speichern wir welche Fahrgäste gerade in diesem Verkehrsmittel sitzen. Des weiteren verwalten wir für jeden Stopp eine Prioritätswarteschlange mit Passagieren die gerade Umsteigen und am Laufen sind. Die Warteschlange ist aufsteigend nach den Ankunftszeiten der Fahrgäste sortiert. Das heißt, diese Fahrgäste sind zu einer früheren Zeit von einem Stopp (gleicher oder per Fußweg verbundener Stopp) losgelaufen und werden zu einer bestimmten Zeit diesen Stopp erreichen.

Für die Umlegung iterieren wir über alle Connections in zeitlich aufsteigender Reihenfolge der Abfahrtszeiten. Für jede Connection führen wir die folgenden Berechnungen in dieser Reihenfolge durch:

Passagiere generieren Passagiere, die ihre Reise vor Abfahrt dieser Connection am Abfahrtsstopp antreten, werden diesem zugeordnet. Für jeden Stopp halten wir ein Array mit Passagieren, die von diesem Stopp aus ihre Reise antreten. Dieses Array ist nach deren Startzeit sortiert. Wir weisen ihnen hier ihre ID zu.

Umsteigende Passagiere zuweisen Umsteigende Passagiere, die gerade auf dem Weg zum Abfahrtsstopp dieser Connection sind und vor Abfahrt dieser ankommen, werden zu diesem Stopp als wartend hinzugefügt.

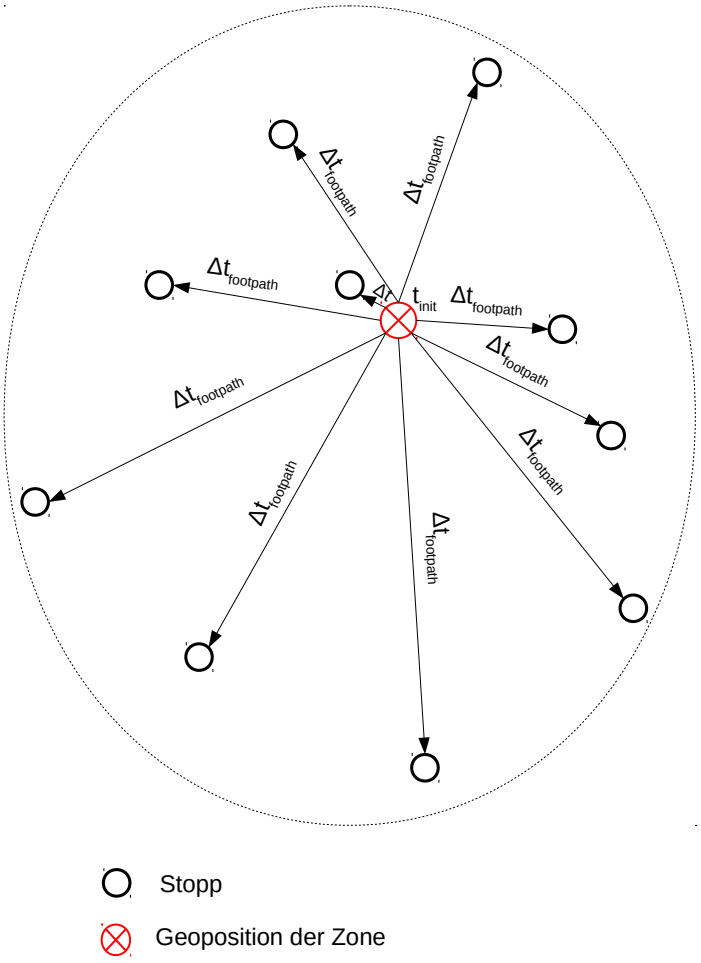


Abbildung 3.2.: Umlegung auf Zonen-Stopp-Fußwege. Die Fahrgäste werden den Start-Stopps in ihrer Startzone zugeteilt.

Einsteigende Passagiere berechnen Wir bestimmen, welche der am Abfahrtsstopp wartenden Passagiere in die Connection einsteigen. Dazu berechnen wir die Connection-MEAT und vergleichen diese mit der Departurestop-MEAT der Connection. Das Verhältnis gibt uns an, wie viele Passagiere in diese Connection einsteigen. Diese Passagiere stehen nun nicht mehr am Stopp, sondern sitzen im Trip zu dem die Connection gehört.

Aussteigende Passagiere berechnen Wir kennen die Passagiere in der Connection (sitzen-gebliebene und neu eingestiegene) und möchten nun wissen welche dieser Passagiere am Ankunftsstopp aussteigen. Dazu vergleichen wir die Trip-MEAT der Connection mit dem Minimum der Arrivalstop-MEAT und der Target-MEAT. Das Minimum über diese Werte gibt uns die erwartete Ankunftszeit am Ziel, wenn die Fahrgäste hier aussteigen. Die damit bestimmten Passagiere werden aus dem Trip entfernt und im nächsten Schritt verwendet.

Umsteigende Passagiere berechnen Im letzten Schritt berechnen wir zuerst mittels der Target-MEAT und der Arrivalstop-MEAT welche der aussteigenden Passagiere einen finalen Fußweg zum Ziel laufen und welche noch Umsteigen. Die Umsteigenden

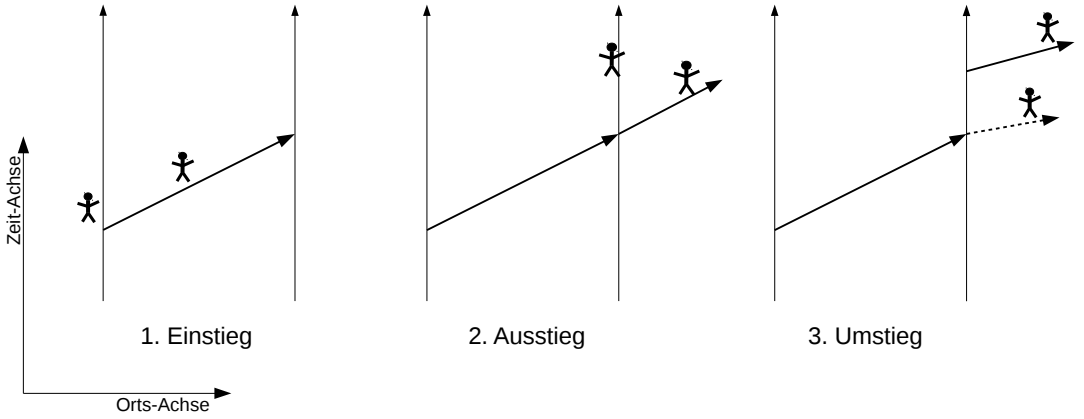


Abbildung 3.3.: Übersicht über die Phasen des Umlegungsschritts

werden dann auf einen Fußweg vom und zum Ankunftsstopp (Schleife) geschickt. Dazu fügen wir die Passagiere in die Prioritätswarteschlange am Ankunftsstopp hinzu. Sie stehen dann nach der minimalen Umsteigezeit wieder am Stopp.

Eine Übersicht über die Verteilung gibt Abbildung 3.3.

Während der Berechnung protokollieren wir für jede Connection nach der Berechnung der *Einsteigenden Passagiere* wie viele Passagiere sich gerade im Trip befinden. Davon werden später noch die Passagiere entfernt die einen Zyklus fahren.

Für die Entscheidungen welche Passagiere ausgewählt werden, benutzen wir hier und im gesamten Algorithmus das selbe Entscheidungsmodell. Dieses wird in Unterkapitel 3.3.5 beschrieben.

3.3.4. Zyklenelimination

Die vorherigen Schritte des Algorithmus verhindern keine Zyklen in Journeys von Fahrgästen. Daher müssen wir diese entfernen.
Wir iterieren über alle Journeys von Fahrgästen und markieren Stopps, die doppelt in einer Journey vorkommen. An diesen Stopps beginnt bzw. endet ein Zyklus. Um den Zyklus zu eliminieren, entfernen wir aus allen zeitlich dazwischen liegenden Connections den betroffenen Fahrgast. Damit erhalten wir eine zyklenfreie Umlegung.

Am Ende jeder Iteration fügen wir für jede Connection die neuen mitfahrenden Fahrgäste hinzu. Nach Beendigung der letzten Iteration erhalten wir die Umlegung.

3.3.5. Entscheidungsmodell

In den beiden Schritten *Zonen-Stopp-Fußwege* und *Umlegung* benötigen wir ein Modell um Fahrgäste anteilig zu verteilen. Dies geschieht durch MEAT-Werte als Gewichte. Wir verwenden stets das selbe Entscheidungsmodell. Dieses entscheidet für zwei Alternativen mit MEAT $meat_1, meat_2$ und eine Liste an Fahrgästen, welche Fahrgäste welche Alternative wählen. Wir benötigen zusätzlich noch ein *offset*. Dieses gibt die maximale Verschlechterung der MEAT an, die wir in Kauf nehmen.
Wir unterscheiden dabei drei Fälle:

$\mathbf{meat_1 + offset < meat_2}$ Der Wert für Alternative zwei ist zu schlecht, als dass sich Fahrgäste dafür entscheiden würden. Wir weisen alle Fahrgäste Alternative eins zu.

$\mathbf{meat_1 > meat_2 + offset}$ Der Wert für Alternative eins ist zu schlecht, als dass sich Fahrgäste dafür entscheiden würden. Wir weisen alle Fahrgäste Alternative zwei zu.

sonst In diesem Fall gilt:

$meat_2 - meat_1 \in [\text{-}offset,\ offset] \Rightarrow \Delta t := meat_2 - meat_1 + offset \in [0, 2 \cdot offset]$

Damit berechnet sich die Wahrscheinlichkeit für Alternative eins zu $w_1 := \frac{\Delta t}{2 \cdot offset}$ und für Alternative zwei zu $w_2 := 1 - w_1$. Für jeden Passagier wird eine Münze geworfen, wobei gilt, dass Zahl mit w_1 eintritt und Kopf mit w_2. Je nach Ergebnis wird der Passagier der entsprechenden Alternative zugeordnet. Die Wahrscheinlichkeit für eine Alternative korreliert hier mit deren MEAT. Je niedriger die MEAT im Vergleich zur MEAT der anderen Alternative, desto mehr Fahrgäste werden zugewiesen. Bei gleichen MEAT-Werten teilen sich die Fahrgäste zu gleichen Teilen auf die Alternativen auf.

3.4. Fußwege

In diesem Kapitel betrachten wir den Algorithmus mit Transfer-Fußwegen. Da ein Fußweg jederzeit begangen werden kann, ist keine zeitliche Sortierung wie bei den Connections möglich. Das bedeutet wir müssen Fußwege anders behandeln. Wir lassen den grundsätzlichen Aufbau des Algorithmus unverändert. In die vier Schritte der Schleife integrieren wir die Berechnungen für die Transfer-Fußwege.

MEAT- und Profil-Berechnung Wir beginnen schon in der Profilberechnung mit der Integration der Transfer-Fußwege. Dazu legen wir pro Stopp zwei Profile an: Das erste Profil speichert die MEAT-Werte ohne Transfers, das zweite Profil die MEAT-Werte mit Transfers. Im zweiten Profil eines Stopps treten deshalb Einträge auf, für die zuerst ein Transfer zu einem anderen Stopp stattfinden muss. Wir benutzen das zweite Profil für die Evaluation eines Umstiegs am Ankunftsstopp einer Connection. Das erste Profil benötigen wir für die *Departurestop-MEAT* und im nächsten Schritt für die Umlegung. Bei den Berechnungen der verschiedenen MEAT-Werte benutzen wir für die *Arrivalstop-MEAT* das Profil mit Transfers. Ansonsten ändern wir daran nichts. Das Profil ohne Transfers bauen wir wie in 3.3.1 auf, das Profil mit Transfers bauen wir folgendermaßen auf:
Wir iterieren über alle adjazenten Stopps des Abfahrt-Stopps. Für jeden dieser Stopps berechnen wir die spätest mögliche Abfahrtszeit (Abfahrtszeit der Connection abzüglich der Dauer des Fußwegs). Den MEAT-Wert kennen wir schon, es ist derselbe wie für das andere Profil. Wir fügen einen neuen Profileintrag in das Profil mit Transfers des adjazenten Stopps ein, falls der neue MEAT-Wert kleiner ist als der schon vorhandene MEAT-Wert im entsprechenden Profileintrag. Damit erhalten wir das Profil mit Berücksichtigung der Umsteigezeiten und der Fußwege. Die Fußwege können auch als Umsteigezeiten zwischen zwei verschiedenen adjazenten Stopps gesehen werden.

Zonen-Stopp-Fußwege bestimmen Dieser Schritt bleibt unverändert. Wir verwenden das Profil ohne Transfers.

Umlegung Wir ändern nur den Teil, in dem Personen aus der Connection aussteigen und dann umsteigen. Dabei gehen wir über alle adjazenten Stopps des Ankunftsstopps. Für jeden dieser Stopps berechnen wir die MEAT: Diese entspricht der MEAT im Profil des Stopps zum Zeitpunkt *Ankunft + Dauer Fußweg*. Dazu müssen wir über die ersten Einträge des Profils iterieren. Einträge mit einer Abfahrtszeit früher der

aktuellen Abfahrtszeit der Connection löschen wir dabei. Wir verschlechtern diese MEAT nun noch in Abhängigkeit der Dauer des Fußweges. Muss der Fahrgast nur warten, so ist die Verschlechterung nicht so hoch, wie wenn er einen Fußweg nehmen muss. Anhand dieser so berechneten MEAT-Werte entscheiden wir, von welchem der adjazenten Stopps welcher Fahrgast abfährt.

Zyklenelimination Die Zyklenelimination findet prinzipiell wie im Algorithmus ohne Fußwege statt. Wir eliminieren hier jedoch Zyklen nach Stationen und nicht nach Stopps[2]. Dabei enthält eine Station alle Stopps, die über Fußwege miteinander verbunden sind.

Das Entscheidungsmodell belassen wir wie in 3.3.5.

3.5. Laufzeit

Wir betrachten in diesem Unterkapitel die Laufzeit des Algorithmus. Dazu analysieren wir jeden Schritt einzeln und zeigen auf warum der Algorithmus mit realen Eingaben sehr schnell ist.

Für die Analyse führen wir einige Größen ein:

- Es sei α die maximale Anzahl an Einträgen in einem Profil. Wir beachten, dass im Profil keine dominierten Connections vorkommen.
- Es sei β die maximale Anzahl an Profileinträgen, die bei der Evaluation eines Profils angeschaut werden.
- Es sei γ die maximale Größe einer Clique im Fußwegegraph.
- Es sei δ die durchschnittliche Anzahl an Connections, die ein Fahrgast benutzt.
- Es sei ζ die maximale Anzahl an Connections, die ein Fahrgast benutzt (inklusive Zyklen).
- Es sei η die maximale Anzahl Zonen-Stopp-Fußwegen einer Zone.

Wir zeigen in Kapitel 4 experimentell, dass α, β, γ, δ und ζ klein sind. In Tabelle 4.5 sind konkrete Werte der Stuttgart-Instanz gegeben.

Zuerst werden wir die Laufzeit für eine einzelne Zielzone analysieren.

MEAT- und Profil-Berechnung Wir iterieren über alle Connections. Dabei führen wir die Berechnung der MEAT-Werte durch und aktualisieren die Profile.

MEAT-Berechnung Zur Berechnung der *Trip-MEAT*, *Target-MEAT*, *Departure-MEAT* und *Connection-MEAT* benötigen wir konstant viel Zeit pro Connection. Für die Arrivalstop-MEAT evaluieren wir das Profil am Ankunftsstopp. Dazu betrachten wir die ersten Einträge bis die Abfahrtszeit später als die Ankunftszeit plus die maximale Verspätung ist. Wir schauen dadurch nur β viele Einträge an und sind in $\mathcal{O}(\beta)$ pro Connection.

Profil aktualisieren Die Aktualisierung des Profils ohne Transfers können wir in konstanter Zeit durchführen. Wir vergleichen dazu nur den ersten Eintrag mit der Connection-MEAT und fügen eventuell einen neuen Eintrag hinzu.
Die Profile mit Transfers aktualisieren wir nur falls die Connection schon im Profil ohne Transfers dominierend ist. Dazu iterieren wir über alle adjazenten Stopps

[2]Wir würden Zyklen, die einen direkten Fußweg zwischen zwei Stopps kaschieren nicht eliminieren.

des Abfahrt-Stopps. Dies sind γ viele Stopps, da wir eine Clique betrachten. Bei Profilen an adjazenten Stopps kann es vorkommen, dass wir den neuen Eintrag nicht direkt am Beginn des Profils eintragen können. In diesem Fall iterieren wir über die schon vorhandene Einträge im Profil, was maximal α viele sind. Ist die aktuelle Connection dominierend, so aktualisieren wir das Profil in konstanter Zeit. Damit erhalten wir eine worst-Case Laufzeit von $\mathcal{O}(\gamma \cdot \alpha)$.

Die Berechnung der MEAT-Werte und Profile liegt damit in $\mathcal{O}(\#\text{Connections} \cdot (\alpha \cdot \gamma + \beta)) = \mathcal{O}(\#\text{Connections} \cdot (\alpha \cdot \gamma))$.

Zonen-Stopp-Fußwege bestimmen Wir iterieren über jedes Nachfrage-Tupel. Dabei betrachten wir für die Startzone alle enthaltenen Stopps. Für jeden dieser Stopps führen wir im Profil eine binäre Suche nach der frühesten erreichbaren Connection durch. Für die Verteilung auf die Stopps wird dann für jeden Fahrgast eine Münze geworfen. Es gilt: $\#\text{Nachfrage} - \text{Tupel} \leq \#\text{Fahrgäste}$. Die binäre Suche in einem Profil benötigt $\mathcal{O}(\#\text{Fahrgäste} \cdot \log \alpha)$ Zeit. Es ergibt sich damit eine Laufzeit von $\mathcal{O}(\eta \cdot \#\text{Fahrgäste} \cdot \log \alpha)$.

Umlegung Während der Umlegung iterieren wir über alle Connections. Wir betrachten die fünf einzelnen Schritte:

1. **Passagiere generieren** Dieser Teil wird für jeden Fahrgast genau einmal aufgerufen und ist pro Fahrgast konstant. Es wird auf ein Array zugegriffen und eine ID vergeben.

2. **Umsteigende Passagiere zuweisen** Wir verwalten eine Prioritätswarteschlange mit umsteigenden Fahrgästen für jeden Stopp. Für jeden umsteigenden Passagier haben wir zwei Mal logarithmischen Aufwand in der Anzahl an umsteigenden Fahrgästen zum Einfügen und Entfernen. Aufgrund der unterschiedlichen Start- und Zielzonen und Zeiten der Fahrgäste ist die Anzahl an gleichzeitig zum gleichen Stopp laufenden Fahrgästen sehr gering. Im worst-Case ergibt sich hier eine Laufzeit von $\mathcal{O}(\log \#\text{Fahrgäste})$.

3. **Einsteigende Passagiere berechnen** Ein Fahrgast benutzt maximal ζ viele Connections, dadurch kann er maximal ζ oft in eine Connection einsteigen. Der Aufwand, den Fahrgast einsteigen zu lassen ist konstant.

4. **Aussteigende Passagiere berechnen** Nur wenn ein Fahrgast in eine Connection eingestiegen ist, kann er auch wieder aussteigen. Wir haben pro Fahrgast damit maximal ζ viele konstante Ausstiege.

5. **Umsteigende Passagiere berechnen** Ebenso ist die Anzahl an Umstiegen durch $\zeta - 1$ begrenzt. Ein Umstieg wird in konstanter Zeit berechnet

Für jede Benutzung einer Connection werfen wir eine Münze. Da deren Anzahl pro Fahrgast durch ζ begrenzt ist, ist die Anzahl an Münzwürfen ebenfalls begrenzt. Der gesamte Schritt kann in $\mathcal{O}(\#\text{Connections} \cdot \#\text{Fahrgäste} \cdot \log(\#\text{Fahrgäste}) \cdot \zeta)$ durchgeführt werden.

Zyklenelimination Für die Zyklenelimination wird jede Journey einzeln bearbeitet. Es gilt: $\#\text{Journeys} = \#\text{Fahrgäste}$. Über jede Journey iterieren wir zwei Mal. In der ersten Iteration suchen wir Zyklen, in der Zweiten entfernen wir diese. Wir sind also beschränkt durch $\mathcal{O}(\#\text{Fahrgäste} \cdot \zeta)$.

Es fehlt uns noch die Laufzeit für das Sortieren der Connections: $\mathcal{O}(\#\text{Connections} \cdot \log(\#\text{Connections}))$. Es ergibt sich damit folgende Gesamtbetrachtung:

$$\mathcal{O}(\#\text{Zonen} \cdot (\#\text{Connections} \cdot (\#\text{Fahrgäste} \cdot \log \#\text{Fahrgäste} \cdot \zeta + (\alpha \cdot \gamma)) + \#\text{Fahrgäste} \cdot \log \alpha) + \eta \cdot \#\text{Fahrgäste} \cdot \log \alpha + \#\text{Connections} \cdot \log \#\text{Connections})$$

Damit ist unsere Laufzeit hauptsächlich abhängig von der Anzahl an Zonen bzw. Stopps, der Anzahl an Connections und der Anzahl an Fahrgästen.

Sämtliche Werte die gespeichert werden, lassen sich in Variablen speichern, deren Größe durch folgende Eingaben begrenzt wird:

- # Connections
- # Trips ($\leq$ # Connections)
- # Stopps
- # Zonen ($\leq$ # Stopps)

Parallelisierung

Der Algorithmus ist sehr gut parallelisierbar. Man führt dazu die Schleife über die Zielzonen parallel aus. Die Akkumulation der Ergebnisse danach sollte sequentiell oder in einem kritischen Abschnitt ausgeführt werden. Wir erhalten damit einen sehr guten Speedup.

4. Evaluierung

In den folgenden Abschnitten werden wir unseren Algorithmus an realen Daten aus dem Großraum Stuttgart evaluieren. Dabei werden wir die schnelle Laufzeit demonstrieren und die Qualität der Ergebnisse analysieren und vergleichen.
Im ersten Kapitel 4.1 erläutern wir, wie wir die Eingabedaten vorberechnet haben. Danach stellen wir unseren Versuchs-Aufbau in Kapitel 4.2 vor und evaluieren damit in Kapitel 4.3 unseren Algorithmus anhand der Daten aus Stuttgart.

4.1. Vorberechnungen

Fußwege

Wie in Kapitel 2 erläutert, benötigen wir die transitiv abgeschlossene Hülle des Fußwegegraphen. Meist geben die Daten von Verkehrsnetzwerken nur einige Fußwege an. Wir bauen daraus und aus den Umsteigezeiten den gewünschten Fußwegegraph. Die Kanten darin entsprechen genau den Fußwegen, die der Algorithmus als Eingabe verlangt.
Die Knoten entsprechen den Stopps. Wir fügen jedem Knoten eine Schleife hinzu. Sie bekommt als Gewicht die minimale Umsteigezeit an diesem Stopp. Als nächstes fügen wir die Fußwege aus den Daten ein. Dabei entsprechen die Gewichte der Kanten der Dauer der Fußwege. Existiert ein Fußweg von a nach b, so muss auch ein Fußweg von b nach a existieren[1]. Jetzt schließen wir den Graphen noch transitiv ab. Dazu führen wir für jeden Knoten Dijkstras Algorithmus aus. Wir erhalten damit die kürzesten Wege von diesem Knoten zu allen anderen erreichbaren Knoten. Damit bilden wir Cliquen. Sie entsprechen den in 2.1.2 eingeführten Stationen. Es ist zu beachten, dass diese Vorberechnung nur für kleine Cliquen effizient funktioniert. Am Ende extrahieren wir alle Fußwege und nutzen diese als Eingabe an den Algorithmus.

Nachfrage

Die Nachfrage wird von unserem Algorithmus als Menge an Quadrupeln

- Startzeit
- Startzone
- Zielzone

[1] Existiert ein Fußweg nur in eine Richtung, so fügen wir die Rückrichtung hinzu. Dabei ist die Dauer des Rückwegs identisch mit dem Hinweg.

Tabelle 4.1.: Parameterwerte des Algorithmus mit Beschreibung

Parameter	Wert	Beschreibung
maximale Verspätung	30 *min*	Gibt die maximal mögliche Verspätung an, die ein Verkehrsmittel in unserem Modell haben kann.
Zeit-Offset	22 *min*	Gibt an welche Abweichung wir zwischen der besten MEAT und der schlechtesten MEAT maximal in Kauf nehmen. Das heißt wir gehen davon aus, dass die Passagiere maximal eine Offset-große spätere Ankunft akzeptieren.
Umsteigekosten	3 *min*	Strafzeit die bei jedem Umstieg, der anfällt auf die MEAT gerechnet wird. Damit fließt die Anzahl an Umstiegen in die Bewertung einer Connection mit ein.
Initiale Start-Intervalle	5 *min*	Größe eines Intervalls für die Diskretisierung der Startzeit der Passagiere.
Passagiervervielfachung	10	Gibt die Genauigkeit der Umlegung an. Wir vervielfachen Fahrgäste während der Umlegung für eine höhere Präzision. Dadurch simulieren wir für einen Fahrgast, 10 Fahrgäste. Nach der Umlegung teilen wir das Ergebnis durch diesen Faktor, womit die Umlegung auch nicht ganze Fahrgäste enthalten kann.
maximale Fußwegedauer	30 *min*	Fußwege mit einer Laufzeit von mehr als der maximalen Fußwegedauer werden nicht berücksichtigt.
Wartezeit-Faktor	8	Faktor mit dem die Umsteigezeit multipliziert wird um längere Wartezeiten stärker zu bestrafen. Dient nur dem Vergleich der Alternativen beim Umsteigen.
Transfer-Faktor	10	Faktor mit dem die Dauer von Fußwegen multipliziert wird, um längere Fußwege stärker zu bestrafen. Dient nur dem Vergleich der Alternativen beim Umsteigen.

- Anzahl an Fahrgästen

verlangt. Die uns vorliegenden Daten sind jedoch Quintupel. Dabei ist die Abfahrtszeit als 15 Minuten Intervall modelliert. Um dieses Intervall zu diskretisieren und ein Quadrupel zu erhalten unterteilen wir es in Intervalle der Dauer 5 Minuten. Anschließend spalten wir das Quintupel in drei Quadrupel auf, wobei die Abfahrtszeit jeweils den Beginn des Intervalls darstellt. Die Fahrgäste verteilen wir gleichmäßig. Diese Diskretisierung verfälscht das Ergebnis kaum, da durch die Umlegung später eine weitere Verwischung am Startstopp stattfindet (siehe 3.3.3).

4.2. Experimentaler Aufbau

Der vorgestellte Algorithmus verlangt die Wahl einiger Parameter. Diese haben einen großen Einfluss auf die Qualität der Ergebnisse. In Tabelle 4.1 stellen wir die Parameter vor und geben unsere Wahl an. Diese haben wir durch experimentelle Beobachtungen getroffen.

Die Wahl der Parameter kann zu leichten Veränderungen der Laufzeit führen. Einzig die Passagiervervielfachung führt zu einer leichten Erhöhung der Laufzeit. Dies ist dadurch begründet, dass mehr Fahrgäste simuliert werden. Bei realen Eingabedaten bleibt die Laufzeit stets im Bereich einiger Minuten.

Tabelle 4.2.: Übersicht über die Stuttgart-Instanz

Messgröße	Wert
∅ Größe zusammenhängender Teilgraphen (Cliquen)	3.675
größter zusammenhängender Teilgraph (Clique)	23
Anzahl isolierter Knoten im Fußwegegraph	9 742
# Connections	795 820
# Trips	48 412
# Stopps	14 038
# Zonen	1 175
# Nachfrage-Quintupel	935 735
Nachfrage Personen	1 161 240
# Fußwege (gegeben)	11 494
# Fußwege (transitiv abgeschlossen)	25 532

4.3. Experimente

Wir evaluieren in diesem Kapitel unseren Algorithmus mit realen Daten aus dem Großraum Stuttgart. Dazu verwenden wir den Algorithmus wie in Kapitel 3 beschrieben. Die Parameter sind wie in Kapitel 4.2 gesetzt. Um aus den ursprünglichen Daten die Eingaben für unseren Algorithmus zu gewinnen, wenden wir die Vorberechnungsschritte aus Kapitel 4.1 an.
Bedanken möchten wir uns bei Maximilian Hartl von der Universität Stuttgart, der die Daten aus Visum exportiert hat.

4.3.1. Instanz

Für die Evaluierung des Algorithmus verwenden wir die Daten des öffentlichen Personen-Verkehrs in Stuttgart. Weder Fahrplan noch Nachfrage liegen in der benötigten Form vor. Deshalb bringen wir die Daten durch Vorberechnungen in die entsprechende Form aus Kapitel 2. Eine Übersicht über wichtige Größen der Daten ist in Tabelle 4.2 zu sehen. Die Instanz hat 800 000 Connections die zu 50 000 Trips gehören. Es werden 14 000 Stopps angefahren, die in 12 000 Zonen aufgeteilt sind. 4 000 der 14 000 Stopps sind über Transfer-Fußwege zu erreichen.

4.3.2. Laufzeitmessungen

Die Laufzeitmessungen wurden auf vier 64-bit Systemen durchgeführt:

Laptop Intel Core i7 mit 4 Threads auf einem Die mit Hyperthreading und 2.9GHz Taktung, 8GB DDR3 Hauptspeicher und Microsoft Windows 10
Kompilierung mit Visual C++ 2015

Compute11 Intel Xeon mit 16 Threads auf zwei Dies ohne Hyperthreading und 2.6GHz Taktung, 64GB DDR3 Hauptspeicher und Linux openSUSE 13.2
Kompilierung mit gcc Version 5.4.0

Compute12 Intel Xeon mit 4 Threads auf einem Die ohne Hyperthreading und 3.7GHz Taktung, 128GB DDR4 Hauptspeicher und Linux openSUSE 13.2
Kompilierung mit gcc Version 5.4.0

Compute3 Baugleich zu Compute12.

Für die Parallelisierung wurde die OpenMP Schnittstelle verwendet. Wir haben dabei dynamisches Scheduling eingesetzt. Bei der Kompilierung wurde das Flag *-O3* für eine maximale Optimierung und schnellen Code gesetzt. Es wurde C++ in Version 11 verwendet.

Tabelle 4.3.: Laufzeiten des Algorithmus

Maschine	Threadzahl	Laufzeit		Speedup
		seriell	parallel	
Laptop	4(HT)	1 093.6s	459.5s	2.380
Compute11	16	593.3s	111.1s	5.340
Compute12	4	495.8s	232.7s	2.131
Compute3	4	485.4s	229.7s	2.113

Tabelle 4.4.: Laufzeit-Aufteilung auf die Phasen des Algorithmus bei serieller Ausführung

Maschine	Laufzeit der Phasen				
	Gesamt	I	II	III	IV
Laptop	1 093.6s	344.8s	8.3s	220.3s	520.2s
Compute11	593.3s	322.3s	6.0s	108.7s	156.3s
Compute12	495.8s	255.5s	4.9s	90.5s	144.9s
Compute3	485.4s	247.0s	4.8s	89.0s	144.6s

Phase I: MEAT-Berechnung, Phase II: Zonen-Stopp-Fußwege, Phase III: Umlegung, Phase IV: Zykleneliminierung

Die Laufzeit für die Instanz aus Stuttgart liegt für den parallelisierten Algorithmus bei zwei bis acht Minuten, je nach verwendeter Maschine. Bei mehr Connections oder mehr Fahrgästen ist eine Erhöhung der Laufzeit zu erwarten. Wir erhalten für die parallele Ausführung einen Speedup von 2.113 bis hin zu 5.340. Durch eine Verdoppelung der Threads erreichen wir keinen Speedup von zwei. Ein Grund dafür ist, dass wir zum einen kritische Bereiche zur Akkumulation haben und zum anderen der Algorithmus memory-bounded ist. Um den Algorithmus zu beschleunigen, reicht es in der parallelen Version aus, die Anzahl an verfügbaren Threads zu erhöhen. Führt man den Algorithmus sequentiell aus, so hängt die Laufzeit maßgeblich von der Taktung des Prozessors und der verfügbaren Bandbreite zum Hauptspeicher ab. Hier schneiden die Maschinen mit DDR4 RAM und schnellerer Taktung des Prozessors deutlich besser ab. Die Laufzeit erhöht sich auf vier bis 18 Minuten. In Tabelle 4.3 sind die Laufzeitmessungen einsehbar.
Wie aus Tabelle 4.4 hervorgeht, benötigt die MEAT-Berechnung und die Zyklenelimination die meiste Laufzeit. Auffällig ist, dass beim Laptop die Zyklenelimination die Hälfte der Zeit benötigt. Vermutlich liegt dies daran, dass durch das Hyperthreading in dieser Phase keine signifikante Beschleunigung mehr erreicht wird. Die Berechnung der initialen Zonen-Stopp-Fußwege benötigt vergleichsweise kaum Rechenzeit. Für die Umlegung selbst wird ungefähr $\frac{1}{5}$ der Laufzeit benötigt. Daraus lässt sich schließen, dass Laufzeitverbesserungen zuerst bei der MEAT-Berechnung und der Zyklenelimination versucht werden sollen.
In Tabelle 4.5 haben wir die Werte eingetragen, die wichtig für die Laufzeitbegründung sind. Wie schon in 3.5 erläutert, sind diese Werte in der Realität durch Konstanten begrenzt. Ein Beispiel ist die Anzahl der Einträge im Profil. Im Maximum beträgt sie 794, was bei einer binären Suche zu maximal $\log_2 794 \approx 10$ Schritten führt.

4.3.3. Ergebnisse

Um die Ergebnisse des Algorithmus qualitativ zu bewerten, schauen wir uns im ersten Abschnitt einige Kennzahlen an. Im zweiten Abschnitt führen wir noch einen Vergleich mit einer Umlegung von Visum [vis] durch.
Unsere Umlegung enthält für jede Connection eine Liste an Fahrgästen. Die Umlegung aus Visum aggregiert die Fahrgäste pro Connection, sodass wir hier nur die Anzahl kennen.

Tabelle 4.5.: Wichtige Werte der Stuttgart-Instanz für die Laufzeitbetrachtung

Variable	Wert
maximale Anzahl Einträge pro Profil (α)	794
∅ Anzahl betrachteter Profileinträge zur Evaluation der Arrivalstop-MEAT	3.426
maximale Anzahl betrachteter Profileinträge zur Evaluation der Arrivalstop-MEAT (β)	60
∅ Größe einer Clique / Anzahl adjazenter Stopps	3.675
maximale Cliquengröße (γ)	23
∅ Anzahl an Einträgen im Profil mit einer früheren Abfahrtszeit	0.017
∅ Anzahl an Connections pro Journey (δ)	10.804
maximale Anzahl an Connections pro Journey (ζ)	82
∅ Anzahl Stopps pro Zone	5.314
maximale Anzahl Stopps pro Zone (η)	36
∅ Anzahl Einträge pro Profil	35.651

Von den Fahrgästen aus unserer Umlegung kennen wir die Start- und Zielzonen, sowie ihre Journey. Damit lässt sich Rückverfolgen von wo und nach wo die Passagiere in einem Verkehrsmittel kommen bzw. gehen. Ausgehend von diesen Listen erfolgt die Analyse.

Qualitative Bewertung

In folgendem Abschnitt muss bei sämtlichen Abbildungen beachtet werden, dass 10-fach mehr Fahrgäste abgebildet sind als in der Nachfrage angegeben. Dies liegt an der Passagiervervielfachung, die uns eine höhere Präzision liefert. Dadurch wird jeder Fahrgast als 10 einzelne Fahrgäste betrachtet.

Für eine erste Bewertung betrachten wir die durchschnittliche Anzahl an gefahrenen Connections pro Fahrgast. Diese liegt bei circa 11. Für den Nahverkehr ist dieser Wert plausibel. Betrachten wir die Anzahl an Fahrgästen pro Connection - ohne die nicht benutzten Connections - so zeigt sich, dass meistens zwischen 5 und 50 Fahrgäste mitfahren. Es gibt noch einige Connections mit 50 bis 100 Fahrgästen. Connections mit mehr als 100 Fahrgästen bilden dagegen die Ausnahme. Ein Histogramm über die Aufteilung findet sich in Abbildung 4.1.

In Abbildung 4.2 haben wir die Reisezeiten der Fahrgäste abgebildet. Die durchschnittliche Reisezeit liegt bei circa 70 Minuten. Allerdings ist ersichtlich, dass die meisten Journeys kürzer sind. Es sind wenige lange Journeys, die die mittlere Reisezeit erhöhen. Wie wir durch Stichproben beobachtet haben, sind dies hauptsächlich Fernverkehrs-Journeys oder Journeys mit einem Start zu Uhrzeiten zu denen kaum Züge fahren, wie zum Beispiel um 1 Uhr nachts.

Eine Journey hat im Mittel zwei Umstiege. Ein Großteil der Fahrgäste erreicht ohne Umstiege ihr Ziel. Nur wenige Fahrgäste müssen vier Mal oder öfter umsteigen. Wie manuelle Stichproben vermuten lassen, enthalten Journeys mit vielen Umstiegen oft eine Busverbindung zum nächsten Bahnhof. Die Anzahl an Umstiegen sind in Abbildung 4.3 als Histogramm zu finden.

Im Durchschnitt aller am Ziel ankommender Fahrgäste benutzt ein Fahrgast 11 Connections. Der Großteil an Journeys hat weniger als 15 Connections. Nur ein kleiner Teil der Journeys hat mehr als 20 Connections. Diese Ergebnisse passen zu unserem Verkehrsnetzwerk aus

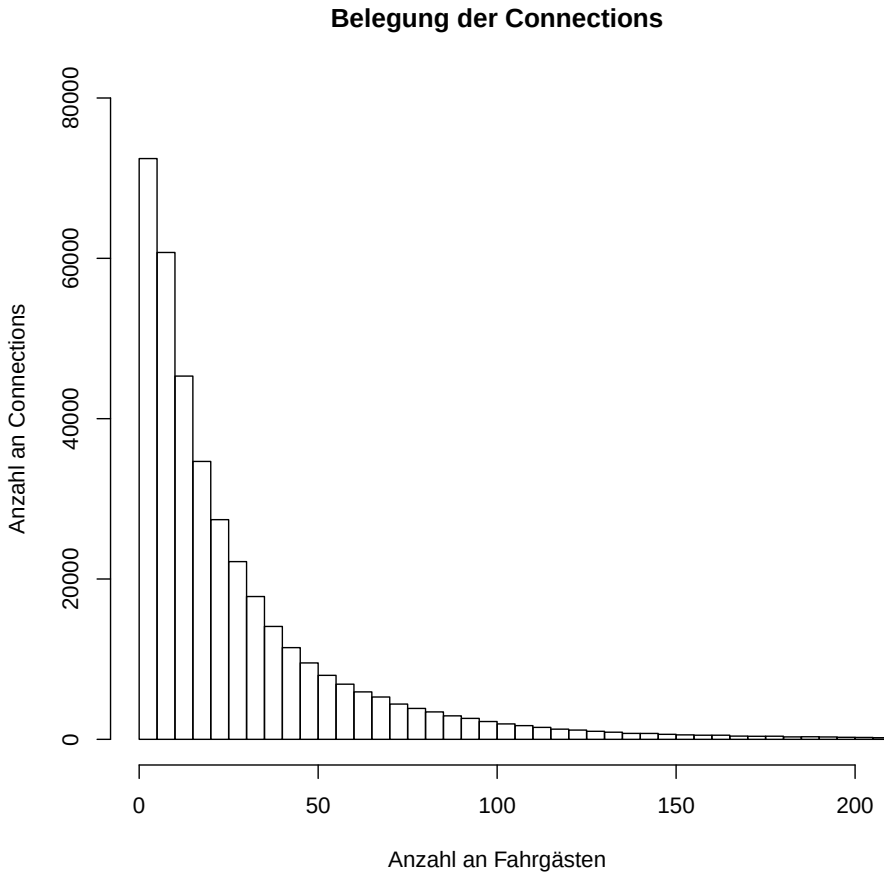


Abbildung 4.1.: Histogramm über die Anzahl der Fahrgäste pro Connection

Stuttgart. Dieses beinhaltet vor allem den Personennahverkehr mit vielen S- und U-Bahnen, Bussen und Regionalzügen. Diese Verkehrsmittel bedienen in der Regel viele Stopps, wodurch eine Fahrt damit aus vielen Connections besteht. Durchquert man Stuttgart mit einem solchen Verkehrsmittel, so können in einigen Fällen sehr viele Connections benutzt werden. Dies tritt jedoch nur in sehr wenigen Journeys auf, wie aus Abbildung 4.4 hervorgeht.

Der Nachfrage nach sollen 1 161 240 Fahrgäste umgelegt werden. Von unserem Algorithmus werden jedoch nur 1 058 619 Fahrgäste verteilt. Dies liegt daran, dass nicht immer ein Verkehrsmittel vom Startpunkt abfährt. Teilweise fährt die nächste Connection auch erst am Folgetag. Diesen Fall betrachten wir nicht. Ist es einem Fahrgast nicht möglich am selben Tag sein Ziel zu erreichen, so streichen wir ihn aus der Nachfrage.
Zur Visualisierung der Umlegung haben wir alle Connections geplottet. Dieser Plot findet sich im Anhang A.1. Dabei entspricht grün wenigen Fahrgästen, gelb mehr Fahrgästen und rot sehr vielen Fahrgästen. Wie zu sehen ist, erhöht sich die Fahrgastdichte je weiter wir ins Zentrum von Stuttgart vordringen. Durch die Pendler entspricht dies der Nachfrage und der Erwartung.

Vergleich mit Visum

Mit dem Verkehrsplanungstool Visum [vis] ist basierend auf der selben Instanz von der Universität Stuttgart eine Umlegung vorgenommen worden. Die Rechenzeit war deutlich höher als bei unserem Algorithmus. Diese Umlegung nehmen wir zum Vergleich.

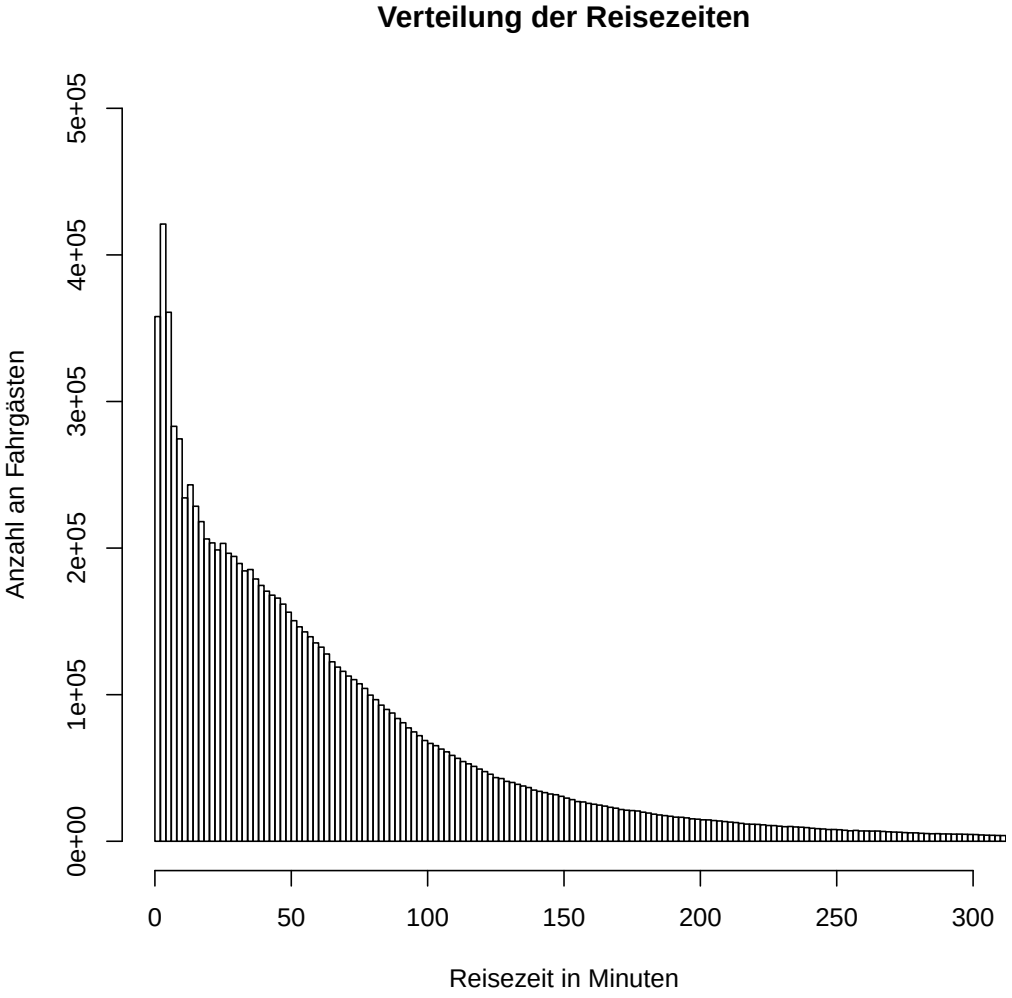


Abbildung 4.2.: Histogramm der Reisezeiten pro Fahrgast

Auffällig ist, dass wir deutlich weniger Connections benötigen, um unsere Fahrgäste zum Ziel zu routen. Da wir von Anfang an 102 621 weniger Fahrgäste betrachten, ist eine Abweichung zu erwarten. Wir ignorieren Fahrgäste, die nicht ans Ziel gelangen können. Dies erklärt nicht die große Differenz von 5 826 820 Connections. Die Connections ohne Fahrgäste sind in beiden Umlegungen ungefähr die Gleichen. Da wir allgemein weniger Connections benutzen, ist die Anzahl an Fahrgästen pro Connection bei unserem Algorithmus im Mittel geringer.
Der Vergleich mit Werten ist in Tabelle 4.6 zu sehen.

In Tabelle 4.7 ist der Vergleich der beiden Umlegungen mittels Perzentilen zu sehen. Die Perzentile wurden über den Betrag der Differenzen der Anzahl an Fahrgästen pro Connection durchgeführt.

Aufgrund der leeren Connections ist das 25%-Quantil gleich 0. Nach oben hin wird die Abweichung größer. Dies liegt vor allem an der unterschiedlichen Anzahl benutzter Connections.

Schlussfolgerung

Es lässt sich sagen, dass die Umlegungen im Groben ähnlich sind. Es werden die selben Connections benutzt und die Anzahl an Fahrgästen in den Connections ist ähnlich. Connec-

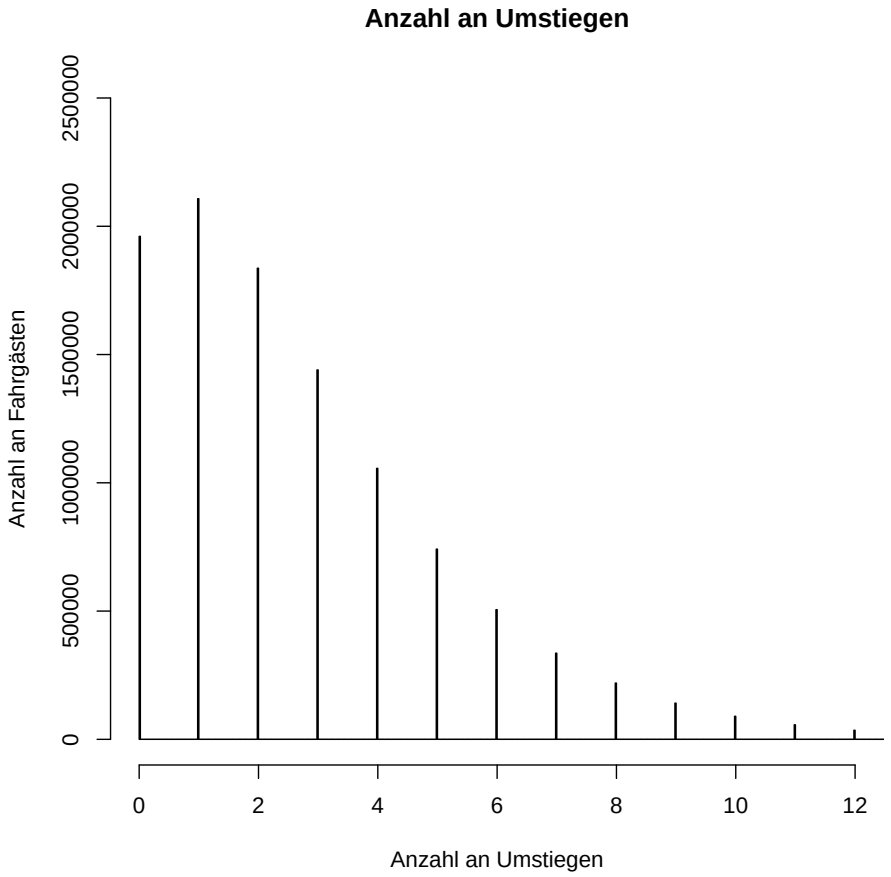


Abbildung 4.3.: Histogramm über die Anzahl an Transfers pro Journey

tions, denen eine Umlegung viele Fahrgäste zuweist, weist die andere Umlegung ebenfalls viele Fahrgäste zu. Wir schließen daraus, dass die Differenz durch unterschiedliche Journeys bedingt ist, nicht jedoch durch komplett verschiedene Umlegungen.

Tabelle 4.6.: Übersicht über die Umlegungen und Vergleich mit der Umlegung von Visum

Eigenschaft	Visum-Umlegung	eigene Umlegung
Anzahl benutzter Connections	17 264 168	11 437 348
Anzahl leerer Connections	451 629	415 384
Anzahl gemeinsamer leerer Connections	377 635	377 635
vollste Connection	2 193.59	1 014.3
∅ Anzahl an Fahrgästen pro Connection	50.120	30.034

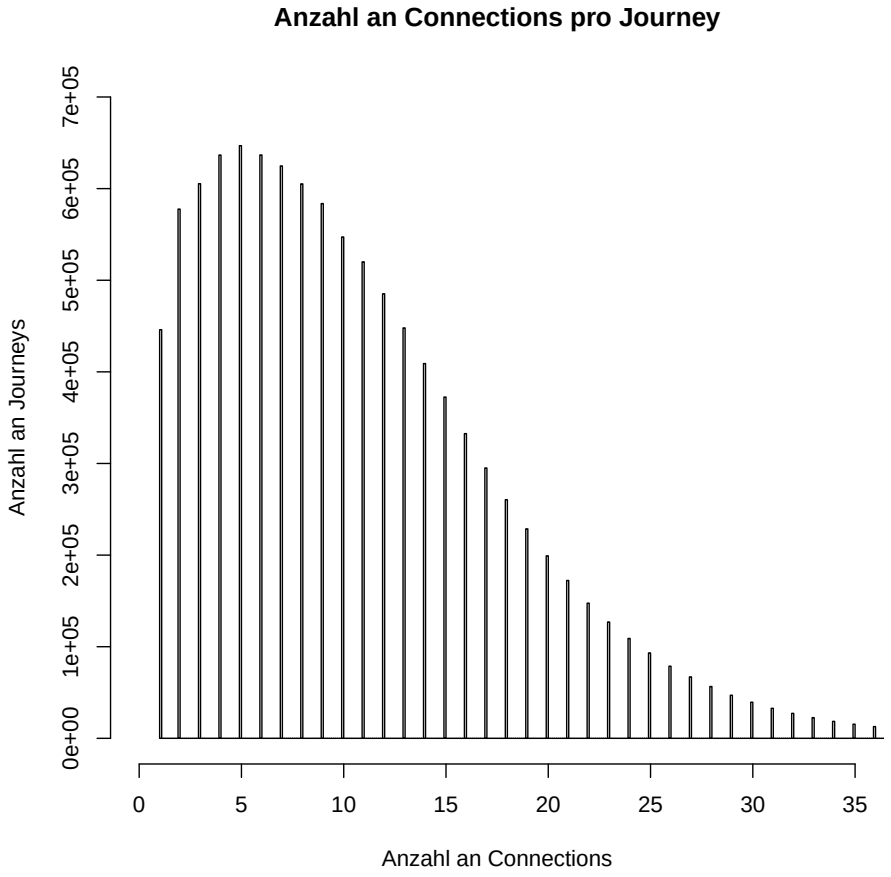


Abbildung 4.4.: Histogramm über die Anzahl an Connections pro Journey

Tabelle 4.7.: Analyse der Umlegung: Perzentile über den Betrag der Abweichung der Umlegungen

Perzentil	Wert
25%	0.000
50%	1.076
75%	13.761
80%	19.565
90%	43.428
95%	76.535
98%	135.887

5. Zusammenfassung

5.1. Ausblick

Wir haben in Kapitel 3 einen effizienten Algorithmus kennengelernt um das Problem einer Umlegung in öffentlichen Netzen zu lösen. Dieser Algorithmus stellt ein grundlegendes Verfahren für eine solche Berechnung dar. Es sind noch einige Erweiterungen und Verbesserungen denkbar:

Kapazitäten Bisher berücksichtigt unser Algorithmus nicht die Kapazität eines Verkehrsmittels. Jedoch hat in der Realität ein Bus eine andere Kapazität als ein Zug. Eine Kapazitätsbeschränkung könnte somit zu einer Umlegung führen, die die Realität präziser abbildet.

Pareto-Optimierung Fahrgäste sind meist an einer schnellen Journey mit möglichst wenig Umstiegen interessiert. Der vorgestellte Algorithmus optimiert primär nach der frühesten Ankunftszeit und sekundär fließen die Umstiege als Strafen mit ein. Eine andere Möglichkeit wäre, die Anzahl an erwarteten Umstiegen zu berechnen um damit eine Pareto-Optimierung durchzuführen. Man könnte somit sowohl nach frühester Ankunftszeit als auch nach minimaler Anzahl an Umstiegen optimieren.

Zyklenvermeidung Beim Auftreten von Zyklen ist auffällig, dass wenige Connections oft in Zyklen vorkommen. Dies ist eventuell vermeidbar, wenn der MEAT-Wert einer solchen Connection genügend groß ist. Dazu wäre es möglich im MEAT-Berechnungsschritt solche Connections heraus zu filtern und deren MEAT entsprechend zu vergrößern.

Entscheidungsmodelle Außer dem in Unterkapitel 3.3.5 vorgestellten Entscheidungsmodell gibt es in der Literatur [Fri] noch weitere:

- Logit
- Kirchhoff
- EVA
- Box-Cox

Interessant könnte sein, geeignete Parametrisierungen für diese Modelle zu finden um sie anschließend in den Algorithmus einzubetten und zu evaluieren.

5.2. Fazit

Wir haben einen Algorithmus vorgestellt, der das Problem einer Umlegung für den öffentlichen Verkehr adressiert. In einem ersten Schritt modellieren wir das Verkehrsnetz und die Nachfrage. Auf dieses Modell wenden wir unseren Algorithmus für eine Umlegung an. Dieser simuliert die einzelnen Fahrgäste im Netz. Als Ergebnis bekommen wir pro Connection eine Liste der Fahrgäste.

Der Algorithmus führt für jede mögliche Zielzone die folgenden Berechnungen durch:

- Im ersten Schritt wird jede Connection bewertet und für jeden Stopp ein Profil der abfahrenden Connections erstellt.
- Im zweiten Schritt werden die Fahrgäste von ihren Startzonen auf die Stopps in der Zone verteilt.
- Anschließend legen wir die Fahrgäste auf das Netz um, indem wir für jede Connection entscheiden, welche Fahrgäste mitfahren.
- Da wir nun Zyklen in den Journeys der Fahrgäste haben können, entfernen wir diese im letzten Schritt.

Nach jeder Iteration akkumulieren wir die einzelnen Ergebnisse und erhalten am Ende die komplette Umlegung.

Der Algorithmus erlaubt es gewisse Parameter zu setzen, um die Umlegung entsprechend zu beeinflussen. Wir können zum Beispiel verschiedene Entscheidungsmodelle verwenden. Aufgrund dessen, dass der Grad der Parallelisierung mit der Anzahl an Zielzonen korreliert, erreichen wir durch eine Erhöhung der Threads eine deutliche Verbesserung der Performance.

Wie die Verkehrsdaten aus dem Großraum Stuttgart gezeigt haben, hat unser Algorithmus eine vergleichsweise geringe Laufzeit. Die Qualität der Ergebnisse ist nach einer Analyse und dem Vergleich mit einer zweiten Umlegung des Tools Visum als gut zu bewerten.

Für Weiterentwicklungen und Performance-Verbesserungen lässt der vorgestellte Algorithmus noch Platz, sodass mit einer weiteren Verbesserung der Laufzeit und einer erhöhten Präzision der Ergebnisse zu rechnen ist.

Literaturverzeichnis

[Bri] Lars Briem. Abbildung von Öffentlichem verkehr in mobitopp.

[DPSW13] Julian Dibbelt, Thomas Pajor, Ben Strasser, and Dorothea Wagner. Intriguingly simple and fast transit routing. In International Symposium on Experimental Algorithms, pages 43–54. Springer, 2013.

[DSW14] Julian Dibbelt, Ben Strasser, and Dorothea Wagner. Delay-robust journeys in timetable networks with minimum expected arrival time. In OASIcs-OpenAccess Series in Informatics, volume 42. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2014.

[FF02] Markus Friedrich and M Friedrich. Analyse und optimierung von verkehrsnetzen im iv und öv. Schriftenreihe des Lehrstuhls für Verkehrs-und Stadtplanung, 14, 2002.

[Fri] Markus Friedrich. Modellierung von verkehrsangebot und verkehrsnachfrage.

[FV05] Markus Friedrich and Peter Vortisch. Verfahren zur dynamischen verkehrsumlegung-ein methodischer überblick. Straßenverkehrstechnik, 49(3):128–135, 2005.

[goo] google earth, https://www.google.de/intl/de/earth/.

[LS11] Dieter Lohse and Werner Schnabel. Grundlagen der Straßenverkehrstechnik und der Verkehrsplanung: Band 2-Verkehrsplanung. Beuth Verlag, 2011.

[RB] Axel Rauschenberger and Matthias Bode. Lösung der statischen verkehrsumlegung mit hilfe der partikelschwarmoptimierung.

[vis] Ptv-visum, http://vision-traffic.ptvgroup.com/de/produkte/ptv-visum/.

[Wei] David Weiß. Efficient enumeration of all reasonable journeys in public transport networks.

Anhang

A. Experimente

Die folgenden Bilder wurden mit *google earth* [goo] erstellt.

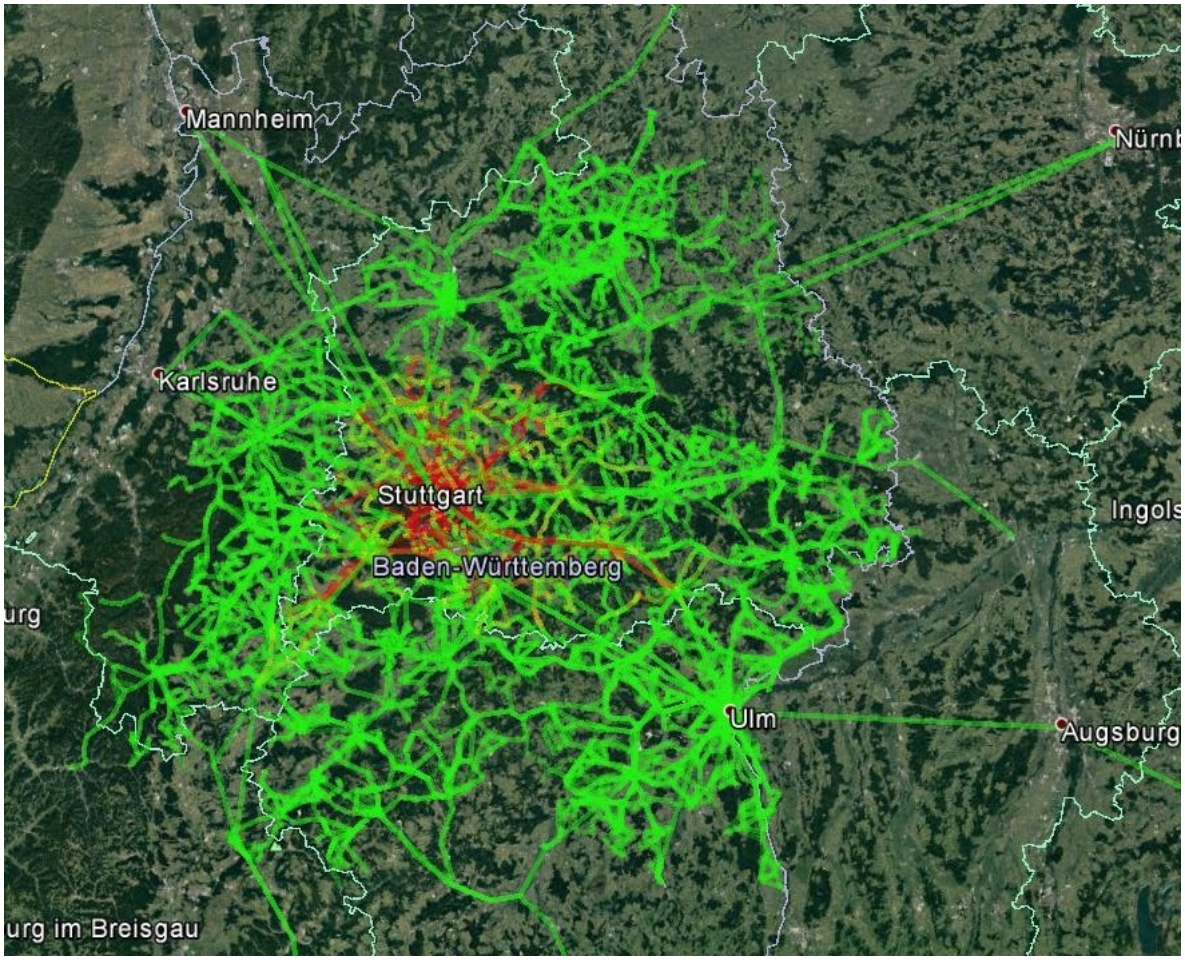


Abbildung A.1.: Visualisierung der Umlegung. Das Fahrgastaufkommen ist auf Routen zusammengefasst.

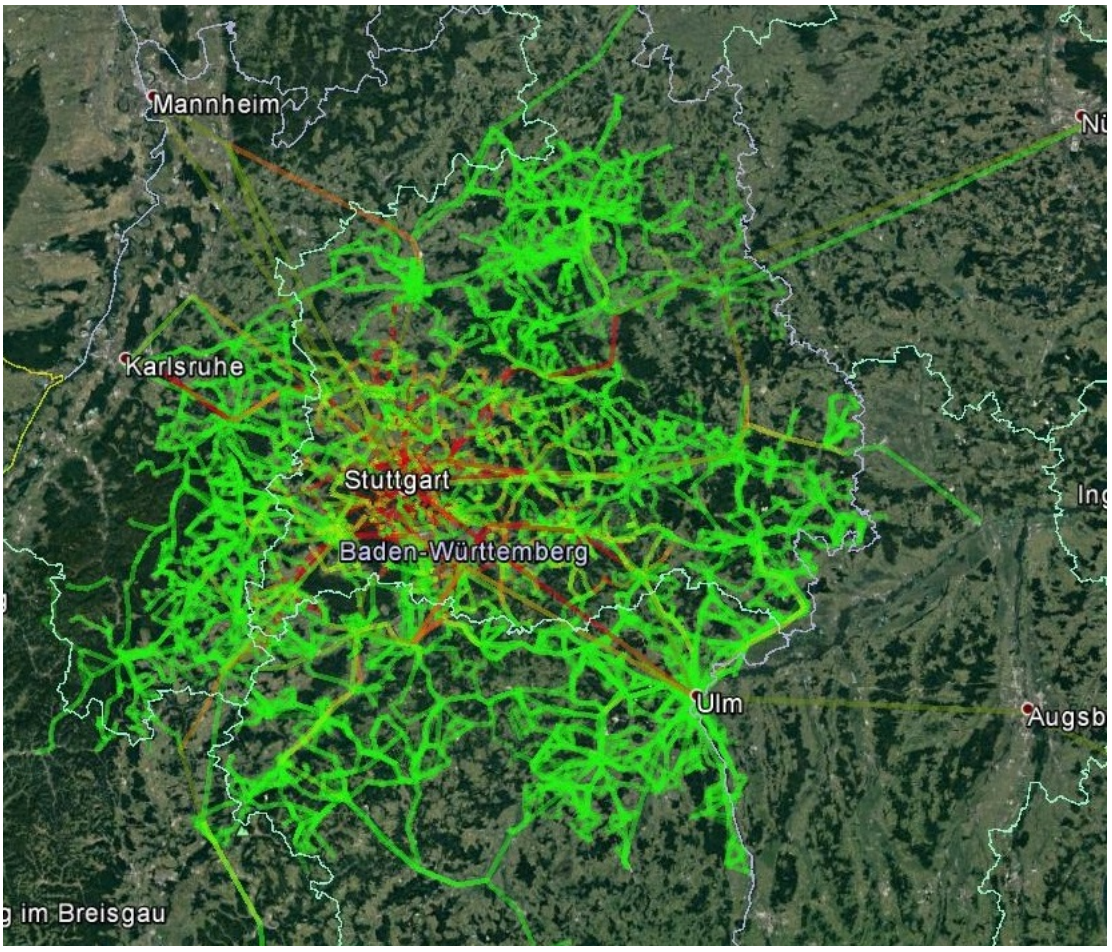


Abbildung A.2.: Visualisierung der Visum-Umlegung. Das Fahrgastaufkommen ist auf Routen zusammengefasst.