Technische Universität München

Fakultät für Mathematik
Lehrstuhl für Angewandte Geometrie und Diskrete Mathematik

Umsteigegraphen im ÖPNV

Masterarbeit von Stephanie Nikola

Themensteller:	Prof. Dr. Peter Gritzmann
Betreuer:	Dr. René Brandenberg, Dr. Michael Ritter, Melanie Herzog
Abgabedatum:	01.05.2013

Hiermit erkläre ich, dass ich diese Arbeit selbstständig angefertigt und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 01.05.2013

Stephanie Nikola

Danksagung

Mein Dank gilt allen, die mich unterstützt haben:

Vor allem meinen Betreuern René Brandenberg, Michael Ritter und Melanie Herzog, die mir stets und unermüdlich mit wertvollen Ratschlägen und Anmerkungen zur Seite standen.

Meinem Kollegen Jan Philipp, der mich nicht nur bei all meinen Fragen zum Thema Datenverarbeitung unterstützte, sondern dessen Anmerkungen mir auch halfen die Dinge aus einem zweiten Blickwinkel zu betrachten.

Der Münchener Verkehrsgesellschaft (MVG), die mir die nötigen Daten für die Anwendung der Algorithmen zur Verfügung stellte.

Meinem Vater, Christian Nikola, der mich nicht nur während meines Studiums immer unterstützte, sondern mir auch mit seiner unermüdlichen Korrekturlesearbeit und seinen guten Fragen verhalf, diese Arbeit anschaulich zu verfassen.

Abstract

These days, the environmental awareness is of utmost importance. Hence, making public transportation attractive for private and corporate customers is more in demand than ever. In the past century, many shortest path algorithms were developed for finding optimal routes in static networks. However, real-life problems like route planning within a public transportation system are not that simple. In reality, there are certain restrictions, such as fixed departure schedules or non-accessibility for handicapped people.
This thesis is dealing with the problem of finding a route for a given start and destination at a certain point of time such that the arrival at the destination is as early as possible, respecting timetables, different choices of modes of transportation and switching points. Therefore, two different models of a public transportation network will be presented in the first part, one with time dependent edge weights and one using a static graph.
Before solving the problem of finding an optimal (shortest) route via these networks, the problem will be translated into an integer linear program (ILP) especially to enable further amplifications and constraints. Subsequently, the focus will lie on the algorithms that manage to solve the problem. In this thesis, we concentrate on an adjusted Dijkstra algorithm, which was also implemented in the course of this thesis (using C++). However, an adapted Bellman-Ford algorithm will be presented as an alternative.
In the scope of this thesis, an internet site was established, providing the possibility of using the results and testing the implemented algorithm.

Inhaltsverzeichnis

Abkürzungsverzeichnis

T_{start}	vorgegebener Startzeitpunkt
E	die Menge der Kanten
V	die Menge der Knoten
M	die Menge der Modi
$V_m^{\leftrightarrow}$	Menge der Knoten im Modus m, an denen ein Umstieg möglich ist
E_{trans}	Menge der Umstiegskanten
$i,\ j$	Knoten aus der Knotenmenge V
$e = (i, j)$	Kante aus der Kantenmenge E
$m,\ m'$	Modi aus der Menge M
$s,\ t$	vorgegebener Start- bzw. Endknoten
τ_e^m	Fahrplan (Ankunft und Abreise) für die Kante $e \in E$ in Modus m
T_m	Funktion der Reisezeiten für Modus $m \in M$
$t^{+,m}_{(i,j),n}$	n-te Abfahrtszeit in Knoten i, bei einer Fahrt in Modus m zum Knoten j
$t^{-,m}_{(i,j),n}$	n-te Ankunftszeit im Knoten j, von Knoten i kommend und bei Nutzung des Modus m
$\delta_m^+(i)$	Menge aller Kanten $e \in E$ im Modus m, die aus dem Knoten i herausführen
$\delta_m^-(i)$	Menge aller Kanten $e \in E$ im Modus m, die in den Knoten i hineinführen
$d_i^{mm'}$	Umsteigedauer beim Wechsel von Modus m zum Modus m' in Knoten i

Kapitel 1

Einleitung

In der heutigen Zeit, in der Staus und Luftverschmutzung zunehmen und gleichzeitig aber das Umweltbewusstsein der Menschen von Tag zu Tag wächst, rücken alternative Fortbewegungsmittel immer mehr in den Vordergrund. Dabei spielt der öffentliche Personennahverkehr eine große Rolle. Auch Faktoren wie steigende Treibstoff- und Energiepreise und Zeitmangel setzen eine optimale Beförderungsmöglichkeit mit öffentlichen Verkehrsmitteln in den Blickwinkel von Forschern, Anbietern und Anwendern.
Städte, Gemeinden und unabhängige Bürgerinitiativen wie der Münchener Arbeitskreis Attraktiver Nahverkehr (AAN), der sich mit der gesamten Thematik des Öffentlichen Nahverkehrs (ÖPNV) und damit verbundenen Verbesserungsmaßnahmen beschäftigt [Aan], sowie betroffene Wirtschafts- und Industrieunternehmen versuchen das Angebot des öffentlichen Verkehrs für alle so attraktiv wie möglich zu gestalten.
So finden auch Forschungsthemen wie die optimale Planung des öffentlichen Verkehrsnetzes (zum Beispiel die optimale Routenplanung von Verkehrslinien und optimale Taktungen) und die hier behandelte Suche nach optimalen Routen innerhalb eines fest gegebenen öffentlichen Verkehrsnetzes immer mehr Interessenten.
Diese Arbeit konzentriert sich auf letztere Problemstellung, die Suche nach einer optimalen Route in Form eines kürzesten Weges innerhalb des ÖPNV-Netzes (öffentlicher Personennahverkehr).

Routenplanung

Die Routenplanung – beispielsweise im Straßenverkehr – ist vor allem wegen ihrer zahlreichen Verwendungsmöglichkeiten im Alltag ein weit erforschtes Themengebiet. Dabei ist bei Privatpersonen mit Auto, öffentlichen Verkehrsmitteln oder zu Fuß ein schnelles Ankommen und die dafür nötige Planung der Strecke ebenso gefragt wie bei einem Transportunternehmen mit Lieferwagen oder Frachtzug.
Dabei kann das Problem der Routenfindung als Kürzeste-Pfad-Suche unter Verwendung eines gerichteten und gewichteten Graphen modelliert werden. Die ersten bekannten Algorithmen zur Lösung des Problems entstanden ab den fünfziger Jahren und finden bis heute Verwendung. Gemeint sind hier der Dijkstra Algorithmus [Dij59], der Algorithmus von Bellman und Ford [Bel58] und der A*-Algorithmus von Raphael, Nilsson und Hart [PEH68]. Im Laufe der Jahre wurde der Fokus immer mehr auf die zur Lösung benötigte Rechenzeit gelegt. Dafür wurden viele sogenannter Speed-up Techniken und heuristische Methoden entwickelt, die alle ein Ziel hatten: Die Laufzeit der Algorithmen zu Verbessern.
Dass die Routenplanung und die entsprechenden Algorithmen und Rechenzeit-Verbesser-

ungen für das öffentliche Nahverkehrsnetz nicht so einfach zu übertragen sind und einiges bei der Anwendung auf ein solches Verkehrsnetz beachtet werden muss, wird im Laufe dieser Arbeit deutlich werden.
Doch was genau ist für die Planung einer Route im öffentlichen Verkehr zu beachten?

Routenplanung im öffentlichen Personennahverkehr und Übersicht zu den Folgekapiteln

Bei der Routenplanung im öffentlichen Verkehr muss zunächst darauf geachtet werden, dass die Verkehrsmittel einem festen Abfahrts- und Ankunftsplan folgen. Somit kann eine Fahrt erst zum nächst möglichen Abfahrtszeitpunkt beginnen.
Hinzu kommt die Tatsache, dass meist mehr als nur ein Verkehrsmittel genutzt werden kann. Das Schlagwort hierfür ist die „Multimodalität“. Einfach gesprochen ist ein multimodales Netzwerk ein solches, in dem dem Benutzer mehr als ein Verkehrsmittel (beispielsweise eine Buslinie) zur Verfügung stehen. Ein sogenannter „Modus“ ist also ein bestimmtes Verkehrsmittel. In vielen Städten steht zum Beispiel nicht nur der Bus als Transportmittel zur Verfügung, sondern auch U-Bahn und Straßenbahn. All diese Nahverkehrsmittel können benutzt werden und sollten somit in der Planung einer Route berücksichtigt werden.
Multimodale Transportsysteme können – ebenso wie beispielsweise Straßen bei der gewöhnlichen Routenplanung – durch Graphen modelliert werden. Dabei werden die Stationen beziehungsweise Haltestellen der Verkehrsmittel durch Knoten repräsentiert. Der Unterschied zu Routenplanungsbezogenen Modellierungen ist hierbei jedoch, dass eine einzige Station hier durch mehrere Knoten repräsentiert werden kann (beispielsweise ein Knoten für jeden Modus, der an dieser Station verfügbar ist). Kanten werden im Allgemeinen, wie in anderen graphentheoretischen Modellierungsansätzen, für jede existierende Verbindung eingeführt. Allerdings treten auch hier besonderheiten auf. So werden in den Modellen Kanten auch dafür verwendet, die Umsteigezeiten oder in einigen Fällen auch die Wartezeiten zu modellieren. Die Kanten zur Repräsentation von Verbindungen innerhalb eines Modus werden dabei oft als Reisekante bezeichnet [LS01].
Damit wäre auch eine weitere Besonderheit bei der Routenplanung im öffentlichen Personennahverkehr genannt, das Umsteigen. Es genügt also nicht, nur die reinen Reisezeiten zu beachten. Ist beispielsweise keine direkte Verbindung zwischen dem Startpunkt und dem gewünschten Ziel vorhanden, so muss das Verkehrsmittel zu irgendeinem Zeitpunkt gewechselt werden. Ist dies der Fall, so muss die benötigte Umsteigezeit beachtet werden. Die Weiterfahrt kann erst nach Vollzug des Umsteigens angetreten werden.

Kapitel 2

Modellierung eines öffentlichen Verkehrsnetzes

Ein Netz des öffentlichen Verkehrs kann, wie eingangs erwähnt, durch einen Graphen modelliert werden.
In diesem Kapitel werden zwei verschiedene Modellierungsansätze für ein solches Verkehrsnetz betrachtet – ein zeitabhängiger Ansatz und ein sogenannter zeitexpandierter Ansatz. Dabei werden für die Modellaufstellung verschiedene Hilfsmittel und Begriffe benötigt:

(i) **Modi:** Modi sind anschaulich ausgedrückt diejenigen Verkehrsmittel, die für eine Fahrt zur Verfügung stehen. Ein Modus ist demnach zum Beispiel „Bus“, ein anderer „U-Bahn“. Alle Modi zusammen bilden eine endliche, nichtleere Menge, aus der die Beförderungsmittel gewählt werden können.
Für eine realistische Abbildung eines öffentlichen Verkehrsnetzes werden als Modi die einzelnen Verkehrslinien bezeichnet wie beispielsweise die U-Bahn Linie „U6“ oder der Bus „B100“.

(ii) **Knoten:** Für die graphentheoretische Modellierung von Stationen werden Graphenknoten verwendet.

(iii) **Kanten:** Es gibt in den nachfolgenden Modellierungsansätzen verschiedene Arten von Kanten. Diese stellen Verbindungen zwischen Stationen eines einzelnen Modus (intramodal) dar oder aber einen Weg zum Wechsel des Beförderungsmittels, also des Modus. Im zeitexpandierten Ansatz werden Kanten auch dafür verwendet, Wartezeiten an einer Station zu modellieren.

(iv) **Der multimodale Umsteigegraph:** Dies ist die Bezeichnung für einen Graphen, der mehrere Modi abbildet und gegebenenfalls einen Wechsel (Umstieg) zwischen den einzelnen Modi berücksichtigt.

Bei der ab Abschnitt 2.3 folgenden graphentheoretischen Modellierung wird zunächst der zeitabhängige Ansatz, auf den hier das Hauptaugenmerk gerichtet wird, vorgestellt. Um den Aufbau im multimodalen Fall besser nachvollziehen zu können, erfolgt er in zwei Schritten. So wird zunächst ein zeitabhängiger, unimodaler Graph modelliert, der schließlich zu einem multimodalen Umsteigegraphen mit mehreren Verkehrslinien ausgeweitet wird.

Anschließend wird der zeitexpandierte Graph, der eine etwas einfachere Struktur als der zeitabhängige Graph aufweist, skizziert und mit dem ersten Modell verglichen.

2.1 Statische und zeitabhängige Graphen

Bevor mit der eigentlichen graphentheoretischen Modellierung für den öffentlichen Personennahverkehr begonnen wird, soll in diesem Abschnitt kurz deutlich gemacht werden, was der Unterschied zwischen einem zeitabhängigen (gerichteten) Graphen und einem sogenannten statischen oder zeitunabhängigen (gerichteten) Graphen ist.
Zunächst der statisch gewichtete Graph:

Definition. Sei $G = (V, E)$ ein Graph mit einer Knotenmenge V und einer Kantenmenge E. Desweiteren seien u und v zwei Knoten aus V und $e = (u, v)$ Element in E. Ist der Graph mit einer Gewichtsfunktion c versehen, für die gilt

$$c : E \to \mathbb{R}_0^+,$$

dann heißt $G = (V, E, c)$ statisch gewichteter Graph. □

Die Gewichtsfunktion c kann dabei bei einem Modell eines öffentlichen Verkehrsnetzes beispielsweise eine konstante oder durchschnittlich benötigte Fahrtzeit von u nach v angeben.
Im zeitabhängigen Fall sei ebenfalls ein (gerichteter) Graph $G = (V, E)$ mit Knotenmenge V und Kantenmenge E als Basis gegeben. Eine Kanten $e \in E$ besitzen nun jedoch nicht mehr nur ein statisches Gewicht $c(e) \in \mathbb{R}_0^+$. Zu jeder Kante $e \in E$ gibt es nun eine Funktion $f_e : I \to \mathbb{R}_0^+$, die auf dieser Kante e jedem Zeitpunkt aus einem Zeitfenster I ein eigenes Gewicht aus $\mathbb{R}_0^+$ zuordnet. Diese Zeitpunkte aus I sollen für die Anwendung in dieser Arbeit in ganzen Minuten angegeben sein – somit ist das Zeitintervall $I \subseteq \mathbb{N}_0$. Die Größe von I, das heißt die Länge des Zeitabschnitts, die es umfasst ist dabei fallabhängig. So können beispielsweise ein paar Stunden, ein Tag oder eine Woche betrachtet werden. Für den Gesamtgraphen ergibt sich damit eine Gewichtsfunktion, die in einen Funktionenraum abbildet und mit $c : E \to \mathcal{F} \subseteq \mathrm{Abb}(I, \mathbb{R}_0^+)$ für ein festes I bezeichnet werden soll.
Die Definition eines zeitabhängig gewichteten Graphen lautet demnach wie folgt:

Definition. Sei $G = (V, E)$ ein (gerichteter) Graph mit einer Knotenmenge V und einer Kantenmenge E. Sei desweiteren eine Menge $I \subseteq \mathbb{N}_0$ gegeben. Eine Gewichtsfunktion c der Form

$$c : E \to \mathcal{F} \subseteq \mathrm{Abb}(I, \mathbb{R}_0^+)$$

heißt dynamische Kantengewichtung für das Zeitintervall I.
Der Graph $G = (V, E, c)$ heißt dynamisch gewichteter (gerichteter) Graph.
Steht I dabei für ein Zeitintervall, so wird G auch zeitabhängig gewichtet genannt. □

Die Gewichtsfunktion $c(e)$ für eine Kante $e \in E$ kann mit einer Funktion

$$f_e : I \to \mathbb{R}_0^+$$

wie oben bezeichnet werden. Die Funktionen f_e können dabei je nach Anwendung unterschiedliche Eigenschaften besitzen.

2.2 Fahrpläne und andere Besonderheiten

Bei der Modellierung eines öffentlichen Verkehrsnetzes treten einige Besonderheiten auf, die beachtet werden müssen. Um diese in späteren Abschnitten verwenden zu können, soll zunächst eine anschauliche Einführung zu den wichtigsten Eigenheiten und Modellgrößen geschehen.
Wie zu Beginn des Kapitels beschrieben ist das Auftreten verschiedener Verkehrsmittel eine besondere Eigenschaft des öffentlichen Verkehrsnetzes. Die einzelnen Verkehrsmittel können dabei meist noch weiter in Verkehrslinien unterteilt werden. Es existiert demnach eine endliche Menge M mit

$$M = M_{\text{Bus}} \cup M_{\text{Tram}} \cup M_{\text{U-Bahn}} \cup ...$$

und

$$M_{\text{Bus}} = \{\text{Menge aller Buslinien}\} \text{ usw.}$$

Ein Element $m \in M$ heißt dabei „Modus“ und bezeichnet bildlich gesprochen eine der Verkehrslinien aus M.

Definition. Sei $G = (V, E)$ ein Graph, $M \neq \emptyset$ eine endliche Menge und

$$g : E \to M$$

eine Abbildung, die jeder Kante $e \in E$ des Graphen G ein Element $m \in M$ zuordnet. Ist

$$G_m = (\{u, v \in V : g((u, v)) = m\}, \{e \in E : g(e) = m\})$$

zusammenhängend für alle $m \in M$, dann heißt $G = (V, E, g)$ modaler Graph, die Menge M „Menge von Modi“ und $m \in M$ Modus. □

In der ÖPNV Praxisanwendung ist der Graph $G_m = (V_m, E_m)$ mit Knotenmenge $V_m \{u, v \in V : g((u, v)) = m\}$ und Kantenmenge $E_m = \{e \in E : g(e) = m\}$ derjenige Graph beziehungsweise Teilgraph, der alle Stationen des betrachteten Modus $m \in M$ als Knoten und alle Fahrstrecken dieser Linie als Kanten besitzt. Hat man in einem öffentlichen Verkehrsnetz beispielsweise mehrere Linien auf einem Straßenabschnitt, so

wird dieser Straßenabschnitt durch mehrere Kanten $e_1,\ e_2,\ e_3, ... \in E$ repräsentiert, für die gilt:

$$g(e_1) \neq g(e_2) \neq g(e_3) \neq ...$$

Eine weitere wichtige Modellgröße neben den Modi, die zu berücksichtigen ist, sind die Reisezeitpläne.
Diese Arbeit konzentriert sich, wie in der Einleitung erwähnt, auf die Modellierung des Problems in kürzest möglicher Zeit von einem gewählten Startpunkt zum gewünschten Ziel zu gelangen. Da Abfahrts- und Ankunftszeiten im öffentlichen Verkehr dabei festen Fahrplänen unterliegen, müssen diese Pläne in das Modell integriert werden, um die tatsächliche Fahrzeit und gegebenenfalls die Wartezeit bis zur nächsten Abfahrt berücksichtigen zu können.

Definition. Sei $G = (V, E, g)$ ein gerichteter, zeitabhängig gewichteter, modaler Graph mit der endlichen Menge $M \neq \emptyset$ von Modi. Die dynamische Kantengewichtung bilde dabei aus dem Zeitintervall $I \subseteq \mathbb{N}_0$ ab. Ein Tupel $(\mathrm{Ab}_e^m, \mathrm{An}_e^m) \in I \times I$ mit $g(e) = m$, $m \in M$ und $e \in E$ heißt Fahrt, falls

$$\mathrm{An}_e^m - \mathrm{Ab}_e^m \geq 0$$

gilt. Ist das Tupel $(\mathrm{Ab}_e^m, \mathrm{An}_e^m)$ eine Fahrt, so wird Ab_e^m Abfahrtszeit und An_e^m Ankunftszeit genannt. □

Definition. Sei $G = (V, E, g)$ ein gerichteter, zeitabhängig gewichteter, modaler Graph mit einer endlichen Menge $M \neq \emptyset$ von Modi. Ist τ_e^m mit $e \in E$ und $m \in M$ eine endliche Menge von n Fahrten mit $n \in \mathbb{N}$, sprich

$$\tau_e^m = \{(\mathrm{Ab}_{e,i}^m, \mathrm{An}_{e,i}^m) | e \in E, m \in M, i = 1..l\}$$

so heißt τ_e^m Fahrplan. □

Ein Fahrplan ist also einer Kante in einem speziellen Modus $m \in M$ zugeordnet und listet alle Ab- und Anfahrten dieses Modus, am Start- bzw. Endknoten der (gerichteten) Kante.
Die Einbindung der Fahrpläne in das Modell erfolgt je Modus durch die Reisezeiten-Funktion

$$T_m : E_m \to \{\tau_e^m : e \in E_m\}.$$

Diese ordnet jeder gerichteten Kante $e \in E_m$ einen Fahrplan zu.
Um deutlich zu machen, dass der zu modellierende Graph von der fest vorgegebenen Reisezeiten-Funktion abhängt, wird die Bezeichnung

$$G_m = (V_m, E_m, T_m)$$

für den zeitahängigen Teilgraphen G_m von $G = (V, E, g)$ eingeführt (kurz: $G_m = (V_m, E_m)$).

2.3 Zeitabhängiger Modellierungsansatz

2.3.1 Graphentheoretische Modellierung eines einzelnen Modus

Nach der Einführung in den vorherigen Abschnitten soll nun das zeitabhängige Modell für die hier betrachtete Problemanwendung im öffentlichen Personennahverkehr dargestellt werden. Hierfür wird zunächst der Graph eines einzelnen Modus m, der beispielsweise einer einzelnen U-Bahn-Linie entspricht, modelliert (siehe Abbildung 2.1).

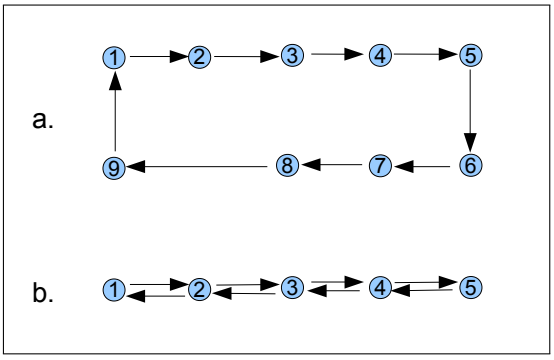


Abbildung 2.1: Zwei Modelle eines monomodalen, gerichteten Graphen G_m

Sei $G_m = (V_m, E_m, T_m)$ ein gerichteter Graph mit der Knotenmenge V_m, der Kantenmenge E_m und der Funktion der Reisezeiten T_m. Letztere ordnet, wie bereits beschrieben, jeder Kante $e \in E_m$ den zugehörigen Fahrplan τ_e^m zu. Die Anzahl von Einträgen von τ_e^m (sprich $|\tau_e^m|$) gibt die Zahl der Fahrten auf der Kante e im Modus m an.
Sei die n-te Abfahrtszeit am Knoten j auf dem Weg zum Knoten k im Modus m und $e = (j,k) \in E_m$ mit $t_{e,n}^{+,m}$ bezeichnet, die zugehörige n-te Ankunftszeit am Knoten k, bei Abfahrt in j mit $t_{e,n}^{-,m}$. Angenommen es finden in I genau s_e^m Fahrten $(t_e^{+,m}, t_e^{-,m})$ im Modus m auf der Kante e statt, so lässt sich der Fahrplan τ_e^m formell darstellen als:

$$\tau_e^m = \left\{ (t_{e,1}^{+,m}, t_{e,1}^{-,m}), ..., (t_{e,s_e^m}^{+,m}, t_{e,s_e^m}^{-,m}) \right\}, \quad e \in E_m;\ s_e^m = |\tau_e^m| \tag{2.1}$$

Dabei gilt nach Definition für die Fahrt-Tupel:

$$\forall n \in \{1, ..., s_e^m\} : \quad (t_{e,n}^{+,m},\ t_{e,n}^{-,m}) \in I \times I, \quad I \subseteq \mathbb{N}_0. \tag{2.2}$$

sowie

$$t_{e,n}^{-,m} - t_{e,n}^{+,m} \geq 0, \quad \forall n \in \{1, ..., |\tau_e^m|\}\ ;\ \forall e \in E_m \tag{2.3}$$

In den meisten Transportnetzwerken ist die Reisezeit zwischen zwei aufeinander folgenden Knoten j und k zeitabhängig und stochastisch [MHM00; Pat+03]. Es wird sich in dieser Arbeit jedoch auf eine Reisedauer beschränkt, die deterministisch ist und lediglich von der Abreise- und Ankunftszeit abhängt (zum Beispiel „Rush Hour“ etc.). So ist die

reine Reisezeit auf der Kante $e \in E_m$ bei einer Wahl des n-ten Abreisezeitpunkts $t_{e,n}^{+,m}$ ($n \in \{1, ..., |\tau_e^m|\}$) als

$$t_{e,n}^{-,m} - t_{e,n}^{+,m} = c_{e,n}^m$$

gesetzt.
Um nun eine Funktion der Form $f_e : I \to \mathbb{R}_0^+$ (siehe Abschnitt 2.1) zu erhalten, die auf der Kante $e \in E_m$ jedem Zeitpunkt aus I ein eigenes Gewicht aus $\mathbb{R}_0^+$ zuordnet und die Gewichtsfunktion für diese Kante darstellt, kann folgende Überlegung angestellt werden: Für jeden Ankunftszeitpunkt $T \in I$ an einer Station ist es sinnvoll, T das kleinstmögliche Kantengewicht zuzuordnen. Dieses ergibt sich für die Kante $e = (j, k) \in E_m$ aus $c_{e,n}^m$ mit einem zu bestimmenden n und der (positiven) Wartezeit zwischen der Ankunftszeit T an der Station j und der Abfahrtszeit in j. So ist die Kantengewichtsfunktion für alle $e \in E_m$ gleich

$$\tilde{f}_e^m(T) = \min_{n, T \leq t_{e,n}^{+,m}} \left(\underbrace{(t_{e,n}^{+,m} - T)}_{\text{Wartezeit } \geq 0} + \underbrace{c_{e,n}^m}_{\text{Reisezeit}} \right) = \min_{n, T \leq t_{e,n}^{+,m}} \; t_{e,n}^{-,m} - T. \tag{2.4}$$

Die Funktion $\tilde{f}_e^m(T)$ ist dabei stückweise linear und es gilt

$$\tilde{f}_e^m : I \to \mathbb{N}_0.$$

Dabei ist I das Zeitintervall, innerhalb dessen die Ankunftszeit an einer Station liegen muss, um die Fahrt von dort aus weiter fortsetzen zu können. Die Bedingung

$$T \leq t_{e,n}^{+,m} \tag{2.5}$$

gibt an, dass die Abfahrtszeit am Knoten j immer nach (oder gleichzeitig mit) der Ankunftszeit an diesem Knoten liegen muss. Ein Fahrgast kann eine Fahrt nicht beginnen solange er noch nicht am Startpunkt eingetroffen ist.
In dieser Arbeit wird angenommen, dass die Funktionen $\tilde{f}_e^m$ periodisch sind, mit der Periode

$$\pi = 24 \text{ Stunden},$$

das heißt

$$\tilde{f}_e^m(T) = \tilde{f}_e^m(T + 24h) \quad \forall e \in E_m, \; \forall m \in M, \; \forall T : \; T, \; (T + 24h) \in I$$

Die Erweiterung auf den nicht-periodischen Fall ist ohne Probleme machbar und würde für eine vollständige Modellierung bei gleichem I lediglich mehr Input-Daten ($|\tau_e^m|$ wäre größer) benötigen.
Betrachtet man ein Intervall I, das mehrere Tage umfasst, so werden in dem in dieser Arbeit dargestellten Fall die Abfahrts- und Ankunftszeiten am zweiten Tag für die

Berechnungen um 1440 Minuten, also 24 Stunden erhöht (siehe Tabelle 2.1). Die Berechnung für weitere Tage erfolgt analog. Ein Zeitintervall I, das mehrere Tage umfasst ist beispielsweise bei überregionalen Verbindungen oder einer späten Startzeit sinnvoll, da die Reise unter Umständen nicht am gleichen Tag beendet werden kann.

Originaldaten		
Tag	**Uhrzeit**	**mögliche Zeitumrechnung (in Minuten)**
1	8.00 Uhr	480
1	9.00 Uhr	540
2	9.00 Uhr	1980

Tabelle 2.1: Umrechnung der Abfahrts- und Ankunftszeiten

Zudem ist zu der Darstellung (2.4) anzumerken, dass in der Regel eine spätere Ankunft an einer Station zu einer späteren oder bestenfalls gleichzeitigen Ankunft an der nachfolgenden Station führen soll. Es soll demnach die FIFO (first in first out) Eigenschaft erfüllt ist. Es gilt [CH66]:

$$\forall T' > T : T' + \tilde{f}_e^m(T') \geq T + \tilde{f}_e^m(T) \quad \forall e \in E_m \tag{2.6}$$

2.3.2 Der multimodale Umsteigegraph

Um ein multimodales Netzwerk zu erhalten, führt man nun in einem zweiten Schritt die einzelnen unimodalen Graphen zusammen und verbindet die Modi an ihren Kreuzungspunkten durch gerichtete Kanten, sogenannte Umstiegskanten (siehe Abb. 2.2). Dabei sei die Menge $V_m^{\leftrightarrow} \subseteq V_m$ die Menge aller Knoten im Modus m, an denen ein Moduswechsel vollzogen werden kann. Zum Beispiel könnte $V_m^{\leftrightarrow}$ alle Knoten aus V_m enthalten oder aber nur Knoten (Stationen) aus V_m, die außer von m noch von einem anderen Modus (Transportmittel) m' angefahren werden. Letzteres soll von nun an in dieser Arbeit gelten. Angenommen wir haben in unserem System l verschiedene Modi, die mit 1 bis l bezeichnet sind. Es ergibt sich ein gerichteter, zeitabhängiger, (multi-)modaler Umsteigegraph $\overleftrightarrow{G} = (G, E_{\text{trans}}, M)$, wobei $M = \{1, ..., l\}$ die Menge der Modi darstellt und $G = \{G_1, ..., G_l\}$ die Menge der zugehörigen zeitabhängigen, gerichteten, unimodalen Graphen ist. Anders ausgedrückt gilt $G = (V, E, g)$.
Der Unterschied zwischen $\overleftrightarrow{G}$ und dem modalen Graphen $G = (V, E, g)$ ist, dass nun die Möglichkeit des Umsteigens besteht. Den Umstiegskanten E_{trans} ist dabei kein Modus zugeordnet.
Mit E_{trans} wird die Menge der Kanten (zum Beispiel Fußwege) bezeichnet, welche die einzelnen Modi an ihren Kreuzungspunkten verbinden. Jede gerichtete Kante $(i, i)_{m,m'} \in E_{\text{trans}}$ mit $i \in V_m^{\leftrightarrow} \cap V_{m'}^{\leftrightarrow}$ hat ein Gewicht $d_i^{mm'} = d(i, i)_{m,m'}$ mit $d : E_{\text{trans}} \to \mathbb{R}_0^+$, das die

benötigte Umsteigezeit für den Moduswechsel in Knoten i von m nach m' angibt.
Es gibt die Möglichkeit auch die Umsteigedauer zeitabhängig zu modellieren oder die Option von einer Station zu laufen mit aufzunehmen. Dies soll hier jedoch außer Acht gelassen werden.
Die Umsteigezeiten sollen in diesem Modell symmetrisch sein, es gilt:

$$d_i^{mm'} = d_i^{m'm} \quad \forall (i,i)_{m,m'} \in E_{\text{trans}},\ i \in V_m^{\leftrightarrow} \cap V_{m'}^{\leftrightarrow}$$

Sei $i \in V_m^{\leftrightarrow} \cap V_{m'}^{\leftrightarrow}$ und $e \in E_m$ die Verbindungsstrecke (Kante), auf der man zur Station i gelangt ist. Desweiteren sei die Ankunft in i zum Zeitpunkt $t_{e,n}^{-,m}$ gegeben. Dann ist bei einem Wechsel zum Modus $m' \in M$ mit $m' \neq m$, analog zur Gleichung 2.5, die Weiterfahrt auf einer von i abgehenden Kante $e' \in E_{m'}$ nur möglich, wenn für $t_{e',n'}^{+,m'}$

$$t_{e',n'}^{+,m'} - t_{e,n}^{-,m} \geq d_i^{mm'} \quad \text{mit } e \in \delta_m^-(i),\ e' \in \delta_{m'}^+(i),\ n \in \{1,...,|\tau_e^m|\},\ n' \in \{1,...,|\tau_{e'}^{m'}|\}$$

mit den zugehörigen Fahrplänen τ_e^m und $\tau_{e'}^{m'}$ gilt.
Die Anschlussfahrt auf einer Kante e' in einem neuen Modus m' darf demnach nur zu einem Zeitpunkt $t_{e',n'}^{+,m'}$ gestartet werden, wenn der Fahrgast angekommen ($t_{e,n}^{-,m}$) und umgestiegen ($d_i^{mm'}$) ist.
Zusätzlich sollen die im Abschnitt 2.3.1 aufgeführte Voraussetzung (2.2) an die Inputdaten sowie die Charakterisierung (2.4) der übrigen Kantengewichte $\tilde{f}_e^m(T)$ gelten. Hierbei ist T bei Betrachtung der Kante $e = (j,k) \in E_m$ nun die Ankunftszeit im Knoten $j \in V_m$ im Modus $m \in M$.
Es soll auch in diesem Modell die Ungleichung (2.6) gelten. Genauer gesagt:

$$\forall T' > T : T' + \tilde{f}_e^m(T') \geq T + \tilde{f}_e^m(T) \quad \forall e \in E_m,\ \forall m \in M \tag{2.7}$$

$$\forall T' > T : T' + d_i^{mm'} \geq T + d_i^{mm'} \quad \forall (i,i)_{m,m'} \in E_{\text{trans}},\ i \in V_m^{\leftrightarrow} \cap V_{m'}^{\leftrightarrow}. \tag{2.8}$$

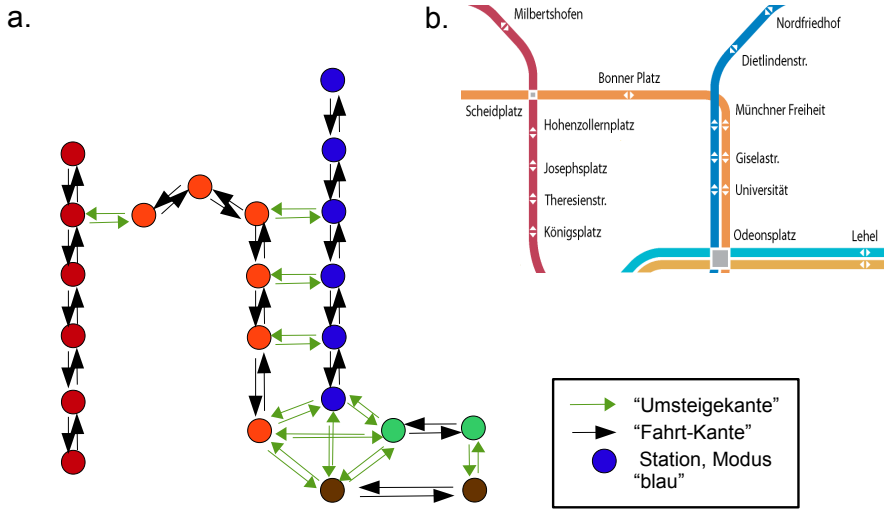


Abbildung 2.2: a) multimodales Modell b) zugehöriger Ausschnitt des Münchner U-Bahnnetzes, Quelle: Wikimedia Commons (Urheber: Maximilian Dörrbecker)

2.3.3 Konstante und variable Umsteigedauer

Im Folgenden werden zwei Möglichkeiten zur Modellierung der Umstiegskanten und deren Gewichte vorgestellt, die je nach Intention angewendet werden können.
Während mit der ersten Möglichkeit unter Umständen einiges an Rechenaufwand gespart werden kann, kann die Zweite dazu genutzt werden auch große Stationen, an denen viele Modi zusammentreffen, realitätsnah zu modellieren.

Die konstante Umsteigedauer

In vielen vorhergehenden Arbeiten (vergleiche beispielsweise [Gei11; BBM06]) wird bei der Modellierung eines zeitabhängigen Umsteigegraphen ein zusätzlicher „Umstiegsknoten“ S zu jeder Station hinzugefügt. Gleiche Stationen in unterschiedlichen Modi teilen sich dabei einen solchen Umstiegsknoten (siehe Abb 2.4a). Eine Kante $(i, S)_m$ von einem Knoten $i \in V_m$ im Modus m zu einem Umstiegsknoten S besitzt dabei das konstante Gewicht „0“. Das Gewicht der Kante $(S, i)_{m'}$ von S zu dem Knoten $i \in V_{m'}$ des Modus m' wird auf den positiven Wert des zum Umsteigens benötigten Zeitaufwands $d_i^{mm'}$ gesetzt [Paj09]. Benötigt man an Station B also fünf Minuten zum Umsteigen von Modus „Rot“ zu Modus „Gelb“ (und umgekehrt), so sähe der Umsteigegraph an dieser Stelle wie in Abbildung 2.3 dargestellt aus. Unter Betrachtung von Abbildung 2.4a und der Tatsache, dass alle in S hineingehenden Kanten das Gewicht „0“ haben ist zu erkennen, warum hier von einer „konstanten Umsteigedauer“ gesprochen wird. Egal von welchem der mit S verbundenen Knoten der Modi „Blau“, „Orange“ oder „Grün“ aus der Umstieg nach

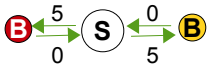


Abbildung 2.3: Beispiel für die Modellierung mit konstanter Umsteigedauer

„Braun“ gestartet wird, die Umsteigedauer ist immer gleich.
Der Vorteil dieses Vorgehens wird vor allem deutlich, wenn die Anzahl der Modi an einer Station drei übersteigt, da dann durch eine solche Maßnahme die Zahl der Kanten reduziert werden kann (vergleiche Abb 2.4a und 2.4c). Allerdings hat es in der Praxis auch starke Einschränkungen zur Folge. Selbst wenn $(i, S)_m$ als beliebige positive, natürliche Zahl gesetzt werden dürfte, können vor allem in größeren Stationen mit unterschiedlichen Plattformen nicht alle Fälle abgedeckt werden. So gibt es keine Möglichkeit Abbildung 2.4b in ein System mit konstanter Umsteigedauer umzuwandeln. Dieser Fall kann zum Beispiel bei U-Bahnen auftreten, wenn die Schienen der Modi „Blau“ und „Grün“ beziehungsweise „Orange“ und „Braun“ direkt nebeneinander liegen (1 Minute), diese Schienen-Paare allerdings weiter voneinander entfernt sind (4 Minuten).

Die variable Umsteigedauer

Variable Umsteigezeiten, wie in Abbildung 2.4c zu sehen, sind gegenüber der konstanten Modellierung flexibler und schaffen die Möglichkeit auch komplexe Systeme realistisch darzustellen. Dies geht jedoch meist auf Kosten des Rechenaufwands.

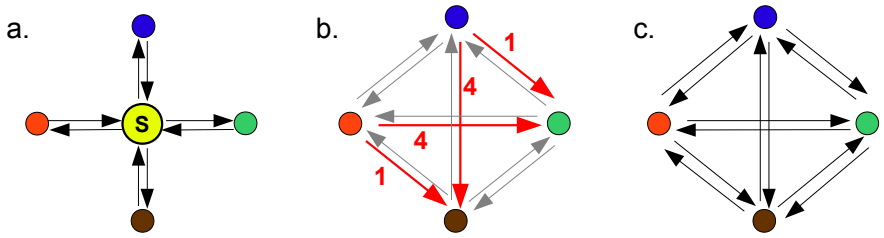


Abbildung 2.4: a) konstante Umsteigedauer - b) Beschränkung der Möglichkeiten bei konstanter Umsteigedauer: dieser Fall wäre mit konstanter Umsteigedauer-Modellierung nicht möglich - c) variable Umsteigedauer

2.4 Zeitexpandierter Modellierungsansatz

Es gibt, wie erwähnt, andere Möglichkeiten einen Umsteigegraphen zu modellieren. Die wohl bis zum heutigen Zeitpunkt mit am häufigsten gewählte Darstellung eines multimodalen Umsteigegraphen ist die Modellierung eines sogenannten „zeitexpandierten“ Graphen. Der Aufbau dieses Ansatzes soll hier nur informell skizziert werden, um später die Unterschiede zum zeitabhängigen Modell herausarbeiten zu können.
Die Idee in diesem Modell ist es, jedes spezifische zeitliche Ereignis (Umstieg, Abreise, Ankunft) durch einen Knoten darzustellen. So wird in Abbildung 2.5 beispielsweise für das Ereignis „Abfahrt 10:06 der Linie 1 an der Station B“ ebenso wie für „Ankunft 10:13 der Linie 2 an der Station D“ ein Knoten benötigt. Zu jedem Knoten gibt es demnach einen festen Zeitwert. Die Kantengewichte ergeben sich als Differenz zwischen den Zeiten auf End- und Startknoten der gerichteten Kante und sind positiv. Die benötigte Umsteigezeit wurde in dieser Abbildung auf zwei Minuten gesetzt, wodurch sich der Pfeil von „An 10:05“ nach „Um 10:10“ ergibt (zwischen diesem Ankunftszeitpunkt und den anderen Umsteigezeitpunkten liegen weniger als zwei Minuten). Der modellierte Graph ist statisch.

Linie 1 (Zeitpunkt$_1$)	
Station	Zeit
A (Abfahrt)	10:00
B (Ankunft)	10:05
B (Abfahrt)	10:06
C (Ankunft)	10:10

Linie 2 (Zeitpunkt$_1$)	
Station	Zeit
B (Abfahrt)	10:06
D (Ankunft)	10:09

Linie 2 (Zeitpunkt$_2$)	
Station	Zeit
B (Abfahrt)	10:10
D (Ankunft)	10:13

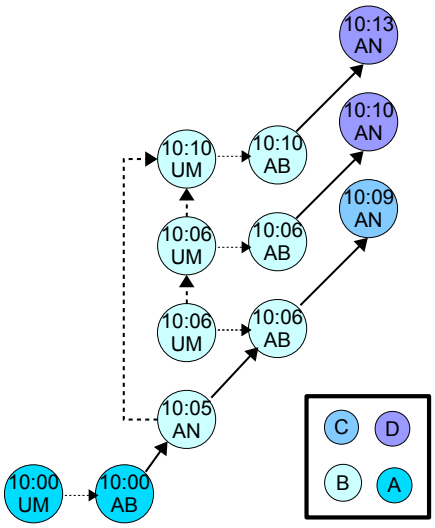


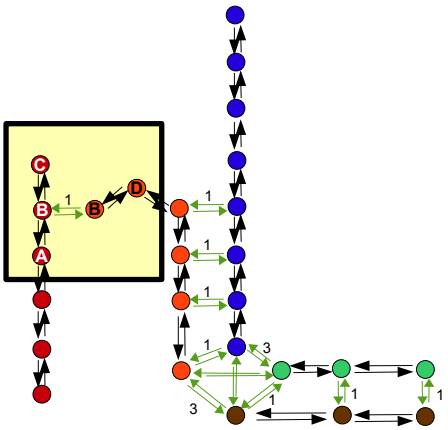
Abbildung 2.5: Der zeitexpandierte Graph für den Zeitraum von 10:00 Uhr bis 10:15 Uhr (siehe Tabelle)

2.5 Vergleich der beiden vorgestellten Modelle

Im Folgenden werden die Hauptunterschiede des Aufbaus der beiden Modelle diskutiert. Für eine ausführliche Gegenüberstellung unter verschiedenen Problemstellungen sei an

dieser Stelle auf [Pyr+04] verwiesen.
Zunächst der augenscheinliche Vorteil des zweiten Modells: Bei Betrachtung des zeitexpandierten Modells ist leicht zu erkennen, dass die Kantengewichte eine einfachere Struktur haben als die Gewichte des zeitabhängigen Ansatzes. Die Zeitabhängigkeit ist hier nämlich nicht mithilfe von Kantengewichtsfunktionen modelliert, sondern durch die Anzahl der Knoten. Dadurch sind die Kantengewichte, wie im vorherigen Abschnitt erwähnt, keine zeitabhängigen Funktionen sondern konstante Gewichte – es liegt also ein statischer Graph vor.
Da die Kantengewichte konstant und nicht-negativ sind, kann beispielsweise ein Kürzester-Pfad-Problem einfach mit dem Dijkstra Algorithmus gelöst werden [Dij59]. Dass dies im zeitabhängigen Fall weiterer Bedingungen – genauer gesagt der Eigenschaft 2.6 – und Anpassungen des Algorithmus bedarf, wird am Ende des nachfolgenden Kapitels genauer erläutert.
Das zeitexpandierte Modell hat jedoch einen großen Nachteil: Die Größe des Graphen selbst. Schon bei einem sehr kleinen Modell wie in Abbildung 2.6 werden, im Vergleich zum zeitabhängigen Ansatz, mehr als doppelt so viele Knoten und eine größere Anzahl an Kanten benötigt. Dies wirkt sich wiederum, zum Beispiel bei der Suche nach einem kürzesten Pfad, schlecht auf die maximal benötigte Laufzeit von Problem-Lösungsalgorithmen aus, da diese von der Größe des Netzwerkes abhängt.
Das Fehlen von Genauigkeitsaspekten wie die Angabe einer von der Ankunftszeit abweichenden Abfahrtszeit an einer Station, ohne dass der Modus gewechselt wird, sind kein wirklicher Nachteil des zeitabhängigen Modells, da diese in einer entsprechenden Erweiterung mit aufgenommen werden können.

a.



b.

Linie 1 (Zeitpunkt$_1$)	
Station	Zeit
A (Abfahrt)	10:00
B (Ankunft)	10:05
B (Abfahrt)	10:06
C (Ankunft)	10:10

Linie 2 (Zeitpunkt$_1$)	
Station	Zeit
B (Abfahrt)	10:06
D (Ankunft)	10:09

Linie 2 (Zeitpunkt$_2$)	
Station	Zeit
B (Abfahrt)	10:10
D (Ankunft)	10:13

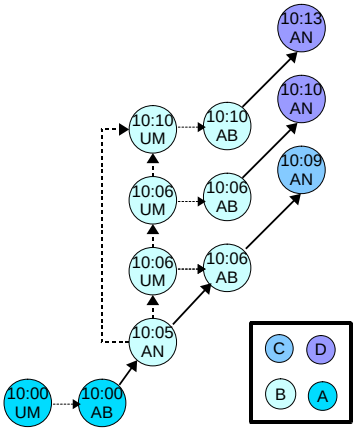


Abbildung 2.6: Der im Kasten(Abb.2.6a) befindliche Ausschnitt soll zum Vergleich in einen zeitexpandierten Graphen(Abb.2.6b) für den Zeitraum von 10:00 Uhr bis 10:15 Uhr(siehe Tabelle) ummodelliert werden

Kapitel 3

Problemstellungen und mathematische Formulierungen

Bei der Planung einer Route von Punkt a zu Punkt b kann sich der „optimale“ Weg nach verschiedenen Kriterien richten. Eines der am meisten behandelten Probleme ist dabei die Minimierung der Reisedauer, also bei gegebener Abfahrtszeit möglichst schnell am Ziel anzukommen (Earliest Arrival Problem – EAP) [Paj09; Gei11; BJ04; Pyr+07]. Aber auch andere Kriterien sind denkbar. Hierzu sei auf Kapitel „Ausblick auf mögliche Modellerweiterungen“ verwiesen, in dem weitere Anwendungen vorgestellt werden.

3.1 Earliest Arrival Problem (EAP) und Zeitabhängiges-Kürzester-Pfad-Problem (TDSPP)

Eines der wichtigsten Probleme im Bereich der Routenfindung ist das Earliest Arrival Problem (frühstmögliche Ankunftszeit). Dieses soll im Folgenden zunächst informell definiert und später als Ganzzahliges Lineares Programm (Integer Linear Programm – ILP) dargestellt werden.

Definition (*Earliest Arrival Problem – EAP*). Sei ein zeitabhängiges Netzwerk mit einem gerichteten Umsteigegraphen $\overleftrightarrow{G} = (G, E_{trans}, M)$ mit $G = (V, E, g)$ gegeben. Zusätzlich sei eine Quelle (Startknoten) $s \in V$ und eine Senke (Endknoten) $t \in V$ sowie eine Startzeit $T_{\text{start}} \in \mathbb{N}_0$ gegeben.
Das EAP sucht einen Weg in $\overleftrightarrow{G}$, der folgende Eigenschaften erfüllt:

(i) Der Weg muss in s starten,

(ii) Der Weg muss in t enden,

(iii) Die Startzeit (frühstmögliche Abfahrtszeit) des Weges in s ist T_{start},

(iv) Die Länge des Weges muss kleiner oder maximal gleich der Länge aller anderen Wege in $\overleftrightarrow{G}$ sein, bei denen Voraussetzungen $(i) - (iii)$ gelten,

und die Differenz aus der Ankunftszeit in t und der Startzeit T_{start} minimiert. □

Gesucht wird also ein s-t-Weg, von Startknoten s zum Endknoten t, der bei gegebener Startzeit so früh wie möglich das Ziel erreicht und somit die Reisezeit ab T_{start} minimiert. Der Weg kann dabei beispielsweise aus Tramfahrten, U-Bahnfahrten oder Fußwegen bestehen.
Verwandt zu dem eben beschriebenen EAP ist das Zeitabhängige-Kürzester-Pfad-Problem (Time Dependent Shortest Path Problem – TDSPP) beziehungsweise das (zeitunabhängige) Kürzester-Pfad-Problem (Shortest Path Problem – SPP). Ersteres soll an dieser Stelle nach [DYQ08] definiert werden.

Definition (*SPP und TDSPP*). Gegeben sei ein zeitunabhängiger (SPP) oder ein zeitabhängiger (TDSPP) Graph $G = (V, E)$ beziehungsweise $\overleftrightarrow{G} = (G, E_{trans}, M)$ mit gegebenen Start- und Endknoten s und t. Im zeitabhängigen Fall sei zudem ein Startzeitintervall $I_{\text{start}} \subseteq \mathbb{R}_0$ gegeben. Das SPP beziehungsweise das TDSPP sucht nun einen optimalen s-t-Pfad $P^*(T^*)$ mit optimalem Startzeitpunkt T^*, der folgende Eigenschaften erfüllt:

(i) $T^* \in I_{\text{start}}$

(ii) Der Pfad startet in Knoten s.

(iii) Der Pfad endet in Knoten t.

(iv) Die Länge des Pfades $P^*(T^*)$, $L(P^*, T^*)$, ist kleiner oder gleich der Länge aller anderen Pfade mit Startzeit $T_{\text{start}} \in I_{\text{start}}$, die die Eigenschaften (i)-(iii) erfüllen.

Minimiert wird in diesem Fall die Summe der Fahrzeiten zwischen den einzelnen Stationen. □

Zu bemerken ist, dass die obigen Definitionen auch für gemischte Graphen gelten, bei denen zeitabhängige wie auch zeitunabhängige Kanten existieren. Dies ist zum Beispiel bei dem in Kapitel 2 vorgestellten Modellierungsansatz der Fall. Hier gibt es sowohl zeitabhängige, intramodale Kanten als auch zeitunabhängige Umstiegskanten. Doch nicht nur bei Umstiegskanten, auch bei Fußwegen kann durch zeitunabhängige Kanten die Realität oft gut approximiert werden.

Es ist offensichtlich, dass bei dem in dieser Arbeit verwendeten Modell das EAP-Ergebnis mit dem des TDSPP gleichgesetzt werden kann. Für eine ausführlichere, anschauliche Darstellung warum dies bei verschiedenen Modi möglich ist, siehe [Paj09].

Zuletzt soll ein erweitertes Kürzester-Pfad-Problem skizziert werden, das „Many-To-Many“-Kürzester-Pfad-Problem.
Anstelle eines Start- und eines Endknotens ist eine Menge von Startknoten X⊆ V und eine Menge von Endknoten Y⊆ V gegeben. Gesucht werden hier die kürzesten Pfade zwischen allen Start- und Endknoten-Paaren $(s, t) \in X \times Y$.

Sowohl das Kürzester-Pfad-Problem als auch das „Many-To-Many“-Kürzester-Pfad-Problem können mit einem, im zeitabhängigen Fall modifizierten, mehrfach angewendeten Dijkstra Algorithmus gelöst werden [Dij59].
Anwendbar ist das „Many-To-Many“-Kürzester-Pfad-Problem beispielsweise dann, wenn Start- und Zielstation in einem gewissen Umkreis eines gegebenen Start- bzw. Zielorts liegen können.

3.2 Mathematische Formulierung des EAP

Das Ziel dieses Abschnittes ist es, das vorgestellte Earliest-Arrival-Problem (EAP) bei gegebener Abreisezeit als Ganzzahliges Lineares Programm zu formulieren (Integer Linear Program - ILP). Neben der Einführung der Formulierung 3.2.1 ist in Tabelle 3.1 eine Kurzübersicht der wichtigsten verwendeten Bezeichnungen abgebildet (siehe auch Kapitel 2).

T_{start}	vorgegebener Startzeitpunkt
E	die Menge der Kanten
V	die Menge der Knoten
M	die Menge der Modi
$V_m^{\leftrightarrow}$	Menge der Knoten im Modus m, an denen ein Umstieg möglich ist
E_{trans}	Menge der Umstiegskanten
$i,\ j$	Knoten aus der Knotenmenge V
$e = (i, j)$	Kante aus der Kantenmenge E
$m,\ m'$	Modi aus der Menge M
$s,\ t$	vorgegebener Start- bzw. Endknoten
τ_e^m	Fahrplan(Ankunft und Abreise) für die Kante $e \in E$ in Modus m
$t^{+,m}_{(i,j),n}$	n-te Abfahrtszeit in Knoten i, bei einer Fahrt in Modus m zum Knoten j
$t^{-,m}_{(i,j),n}$	n-te Ankunftszeit im Knoten j, von Knoten i kommend und bei Nutzung des Modus m
$\delta_m^+(i)$	Menge aller Kanten $e \in E$ im Modus m, die aus dem Knoten i herausführen
$\delta_m^-(i)$	Menge aller Kanten $e \in E$ im Modus m, die in den Knoten i hineinführen
$d_i^{mm'}$	Umsteigedauer beim Wechsel von Modus m zum Modus m' in Knoten i
$x_{e,n}^m$	Entscheidungsvariable (siehe Abschnitt 3.2.1)
$x_i^{mm'}$	Entscheidungsvariable (siehe Abschnitt 3.2.1)
y_i^m	Entscheidungsvariable (siehe Abschnitt 3.2.1)

Tabelle 3.1: Kurzübersicht zur mathematischen Formulierung des EAP

3.2.1 Einführung der mathematischen Formulierung

Nach der Einführung des EAPs in Abschnitt 3.1 soll nun die mathematische Formulierung in Form eines Ganzzahligen Linearen Problems folgen.
Hierfür werden einige Bezeichnungen aus der graphentheoretischen Modellierung in Kapitel 2 benötigt. So sei M die Menge der Modi, V_m die Menge der Knoten und E_m die Menge der Kanten in Modus $m \in M$ sowie $V = \bigcup_{m \in M} V_m$ die Menge aller Knoten und $E = \bigcup_{m \in M} E_m$ die Menge aller Kanten eines Umsteigegraphen, ausgenommen der Umstiegskanten E_{trans}. In der Menge $V_m^{\leftrightarrow} \subseteq V_m$ sind, ebenfalls analog zu Kapitel 2, alle Knoten aus V_m enthalten, an denen ein Umstieg in einen anderen Modus möglich ist. Dabei bezeichnet $d_i^{mm'}$ die Zeit, die für den Moduswechsel in Knoten $i \in V_m^{\leftrightarrow} \cap V_{m'}^{\leftrightarrow}$ von Modus m zu Modus m' nötig ist. Zudem sei für alle Kanten $e \in E_m$ mit $m \in M$ ein Fahrplan τ_e^m gegeben. Wie im vorherigen Kapitel bezeichnet $t_{(i,j),n}^{-,m}$ die n-te Ankunftszeit am Knoten j, bei Abfahrt in i und „Fahrtmodus“ m beziehungsweise $t_{(i,j),n}^{+,m}$ die n-te Abfahrtszeit im Modus m am Knoten i in Richtung j.
Um nun ein Ganzzahliges Lineares Programm zu formulieren werden zunächst die ganzzahligen Entscheidungsvariablen

$$
\begin{array}{ll}
x_{e,n}^m \in \{0,1\} & \forall m \in M;\ \forall e \in E;\ \forall n = 1, \dots |\tau_e^m| \\
x_i^{mm'} \in \{0,1\} & \forall m, m' \in M;\ \forall i : (i_m, i_{m'}) \in E_{\text{trans}} \\
y_i^m \in \{0,1\} & \forall i = 1, \dots, |V|\ ;\ \forall m \in M
\end{array}
$$

eingeführt. Der Lösungssubgraph besteht dann aus genau den Knoten und Kanten, die in der bestimmten, zulässigen Lösung enthalten sind. Hierbei sollen folgende Eigenschaften gelten:

$x_{e,n}^m = 1$	genau dann, wenn die n-te Fahrt aus der Kante $e \in E_m$ in der Lösung enthalten ist.
$x_i^{mm'} = 1$	genau dann, wenn die Umstiegskante $(i,i)_{m,m'} \in E_{\text{trans}}$ im Lösungssubgraphen enthalten ist.
$y_i^m = 1$	genau dann, wenn Knoten $i \in V_m$ im Modus m im Lösungssubgraphen enthalten ist.

Die Problemstellung ist nun, einen optimalen Pfad zu finden, sodass man eine minimale, also frühstmögliche Ankunftszeit am Ziel erreicht. Dies soll in Form eines Ganzzahligen Linearen Problems aufgestellt werden.
Da in dem hier betrachteten Fall die möglichen Ankunftszeiten als Input gegeben sind, bietet das EAP für die Formulierung der Zielfunktion, wie in Definition 3.1 bereits

angedeutet, eine adäquate Möglichkeit, die das obige Problem umgeht:

$$\min \quad \left(\sum_{m \in M} \sum_{e \in \delta_m^-(t)} \sum_{n=1}^{|\tau_e^m|} t_{e,n}^{-,m} x_{e,n}^m \right) - T_{\text{start}} \tag{3.1}$$

Dabei sei $\delta_m^-(t)$ die Menge aller Kanten in Modus m, die in den Zielknoten t hineinführen und T_{start} abermals der Zeitpunkt, an dem die Reise begonnen wird (siehe Tabelle 3.1). Anzumerken ist, dass das Subtrahieren von T_{start} dabei auch vernachlässigt werden könnte und nur aufgrund der Problemdefinition geschieht.
Würde man versuchen, die Zielfunktion ähnlich wie bei einem Traveling Salesman Problem als eine Summe von Gewichten multipliziert mit der Entscheidungsvariable x darzustellen und so einen kürzesten Pfad zu suchen, sähe der Ansatz wie folgt aus:
Zunächst würden für die Formulierung des Problems die zeitabhängigen Kantengewichte

$$\tilde{f}_e^m(T) = \min_{n, T \leq t_{e,n}^{+,m}} \quad t_{e,n}^{-,m} - T$$

aus Kapitel 2 zu einem Vektor von Funktionen

$$f_e^m(T) = (f_{e,1}^m(T), ..., f_{e,|\tau_e^m|}^m(T))$$

mit

$$f_{e,n}^m(T) = \begin{cases} t_{e,n}^{-,m} - T & \text{für } n\text{: } T \leq t_{e,n}^{+,m} \\ +\infty & \text{sonst} \end{cases}$$

vereinfacht werden.
Anschließend kann die Zielfunktion bei gegebenem Startzeitpunkt T_{start}, Startknoten s und Endknoten t durch

$$\begin{aligned} \min \quad & \sum_{i \in V} \sum_{m \in M} \sum_{e \in \delta_m^+(i)} \sum_{n=1}^{|\tau_e^m|} f_{e,n}^m \left(\sum_{e' \in \delta_m^-(i)} \sum_{a=1}^{|\tau_{e'}^m|} t_{e',a}^{-,m} x_{e',a}^m + \sum_{\substack{m' \in M \\ m' \neq m}} \sum_{e' \in \delta_{m'}^-(i)} \sum_{l=1}^{\left|\tau_{e'}^{m'}\right|} (t_{e',l}^{-,m'} + d_i^{m'm}) x_{e',l}^{m'} \right) x_{e,n}^m + \\ & + \sum_{m \in M} \sum_{e \in \delta_m^+(s)} \sum_{n=1}^{|\tau_e^m|} f_{e,n}^m(T_{\text{start}}) x_{e,n}^m + \sum_{\substack{m,m' \in M \\ m' \neq m}} \sum_{i \in V_m^{\leftrightarrow} \cap V_{m'}^{\leftrightarrow}} d_i^{m'm} x_i^{m'm} \end{aligned} \tag{3.2}$$

dargestellt werden. Dabei bezeichne $\delta_m^-(i)$ die Menge aller Kanten in Modus m, die in den Knoten i hineinführen und $\delta_m^+(i)$ die Menge aller Kanten in Modus m, die aus dem Knoten i herausführen.
Damit würde die Verwendung des Vektors $f_e^m(T)$ in dieser Formulierung dasselbe wie

eine Anwendung der Funktion $\tilde{f}_e^m(T)$ bewirken.
Die Formel (3.2) hat jedoch ein entscheidendes Manko: Sie ist nicht linear. Da hier jedoch ein Ganzzahliges Lineares Problem formuliert werden soll, entfällt dieser Ansatz.
Die Zielfunktion zur Lösung des Problems ist demnach:

$$\min \quad \left(\sum_{m \in M} \sum_{e \in \delta_m^-(t)} \sum_{n=1}^{|\tau_e^m|} t_{e,n}^{-,m} x_{e,n}^m \right) - T_{\text{start}} \tag{3.1}$$

Da die Abfahrt an einer Station in einem Modus m nur so gewählt werden darf, dass die Abfahrt später als die Ankunft im gleichen Modus an der selbigen geschieht, benötigt man eine zusätzliche Bedingung:

$$\underbrace{\sum_{e \in \delta_m^+(i)} \sum_{n=1}^{|\tau_e^m|} t_{e,n}^{+,m} x_{e,n}^m}_{\substack{\text{Dies wird betrachtet, falls} \\ \text{die Weiterfahrt im gleichen} \\ \text{Modus } m \text{ erfolgen soll}}} + \underbrace{\sum_{\substack{m' \in M \\ m' \neq m}} \sum_{e' \in \delta_{m'}^+(i)} \sum_{l=1}^{\left|\tau_{e'}^{m'}\right|} (t_{e',l}^{+,m'} - d_i^{mm'}) x_{e',n}^{m'}}_{\substack{\text{Dies wird betrachtet, falls die Weiterfahrt in} \\ \text{einem anderen Modus } m' \neq m \text{ erfolgen soll und} \\ \text{somit die Umsteigezeit berücksichtigt werden muss}}} \geq \sum_{e \in \delta_m^-(i)} \sum_{n=1}^{|\tau_e^m|} t_{e,n}^{-,m} x_{e,n}^m$$

$$\forall i \in V \setminus \{s,t\};\ \forall m \in M \tag{3.3}$$

Ungleichung (3.3) garantiert, dass der Abfahrtszeitpunkt an einem Knoten $i \in V$ stets später als die Ankunftszeit an diesem sein muss. Dies ist die erste Nebenbedingung des Problems.
Setzt man zusätzlich die frühstmögliche Abfahrtszeit im Startknoten s auf den gegebenen Wert T_{start} mit der Bedingung

$$\sum_{m \in M} \sum_{e \in \delta_m^+(s)} \sum_{n=1}^{|\tau_e^m|} t_{e,n}^{+,m} x_{e,n}^m \geq T_{\text{start}} \tag{3.4}$$

so ist der erste Schritt in Richtung einer Ganzzahligen Linearen Problemformulierung getan.
Es sollte jedoch beachtet werden, dass bei Gesamtfahrzeiten, die durch teilweise lange Wartezeiten unabhängig davon sind ob an einer bestimmten Station der erste ankommende Zug oder der darauf Folgende gewählt wird, auch mehrere optimale Lösungen auftreten können.
Als Nächstes soll garantiert werden, dass an Startknoten s und an Endknoten t nicht umgestiegen wird. Dies geschieht beispielsweise mit den Nebenbedingungen

$$\sum_{m \in M} y_s^m = 1 \tag{3.5}$$

$$\sum_{m \in M} y_t^m = 1 \tag{3.6}$$

da hier lediglich der Fall betrachtet wird, dass nur in einem Knoten $i \in V$ ein Umsteigen möglich ist, in dem für mindestens zwei bestimmte Modi $m, m' \in M$ $i \in V_m^{\leftrightarrow} \cap V_{m'}^{\leftrightarrow}$ gilt. Damit der Lösungssubgraph einen Pfad ergibt, benötigt man des Weiteren die Bedingungen

$$\sum_{e \in \delta_m^-(i)} \sum_{n=1}^{|\tau_e^m|} x_{e,n}^m + \sum_{\substack{m' \in M \\ m' \neq m}} x_i^{m'm} = y_i^m \quad \forall i \in V \setminus \{s\},\ \forall m \in M \tag{3.7}$$

$$\sum_{e \in \delta_m^+(i)} \sum_{n=1}^{|\tau_e^m|} x_{e,n}^m + \sum_{\substack{m' \in M \\ m' \neq m}} x_i^{m'm} = y_i^m \quad \forall i \in V \setminus \{t\},\ \forall m \in M \tag{3.8}$$

Die Nebenbedingungen (3.7) und (3.8) garantieren, dass – ausgenommen von Start- und Endknoten – jeweils zwei Kanten an einem im Lösungssubgraphen enthaltenen Knoten anliegen: Eine ausgehende sowie eine eingehende Kante.
Eine Subtourenelimination wird hier nicht benötigt, da die Gleichungen (3.7) und (3.8) zusammen mit Ungleichung (3.3) das Auftreten von Subtouren bereits ausschließen.
Was nun noch ausgeschlossen werden muss ist, dass eine Station mehrfach, aber in verschiedenen Modi angefahren oder verlassen wird (abgesehen von Umstiegen innerhalb einer Station). Dies ist durch die Gleichungen (3.7) und (3.8) nicht abgedeckt, da dort die Modi einzeln betrachtet werden. So werden folgende Ungleichungen zum Vermeiden einer mehrfachen An- beziehungsweise Abreisen an derselben Station eingeführt:

$$\sum_{m \in M} \sum_{e \in \delta_m^+(i)} \sum_{n=1}^{|\tau_e^m|} x_{e,n}^m \leq 1 \quad \forall i \in V \tag{3.9}$$

$$\sum_{m \in M} \sum_{e \in \delta_m^-(i)} \sum_{n=1}^{|\tau_e^m|} x_{e,n}^m \leq 1 \quad \forall i \in V \tag{3.10}$$

Um sicherzustellen, dass jeweils lediglich eine einzige gerichtete Kante an Start- und Endknoten anliegt, eine abgehende für den Startknoten s und eine eingehende für Endknoten t, benötigt man zuletzt nur noch die folgende Gleichung (3.11):

$$\sum_{m \in M} \sum_{e \in \delta_m^-(s)} \sum_{n=1}^{|\tau_e^m|} x_{e,n}^m = 0 \tag{3.11}$$

Das zusätzliche Verbieten von ausgehenden Kanten im Endknoten t ist in diesem Fall überflüssig, da es unter den gegebenen Nebenbedingungen keinen zulässigen Pfad gibt, der über t hinaus geht.

3.2.2 Die ILP-Formulierung des EAP

Insgesamt ergibt sich nach Abschnitt 3.2.1 also folgende Formulierung:

$$\min \quad \left(\sum_{m \in M} \sum_{e \in \delta_m^-(t)} \sum_{n=1}^{|\tau_e^m|} t_{e,n}^{-,m} x_{e,n}^m \right) - T_{\text{start}} \tag{3.1}$$

u.d.N.

$$\sum_{e \in \delta_m^+(i)} \sum_{n=1}^{|\tau_e^m|} t_{e,n}^{+,m} x_{e,n}^m + \sum_{\substack{m' \in M \\ m' \neq m}} \sum_{e' \in \delta_{m'}^+(i)} \sum_{l=1}^{\left|\tau_{e'}^{m'}\right|} (t_{e',l}^{+,m'} - d_i^{mm'}) x_{e',n}^{m'} \geq \sum_{e \in \delta_m^-(i)} \sum_{n=1}^{|\tau_e^m|} t_{e,n}^{-,m} x_{e,n}^m$$

$$\forall i \in V \setminus \{s,t\};\ \forall m \in M \tag{3.3}$$

$$\sum_{m \in M} \sum_{e \in \delta_m^+(s)} \sum_{n=1}^{|\tau_e^m|} t_{e,n}^{+,m} x_{e,n}^m \geq T_{\text{start}} \tag{3.4}$$

$$\sum_{m \in M} y_s^m = 1 \tag{3.5}$$

$$\sum_{m \in M} y_t^m = 1 \tag{3.6}$$

$$\sum_{e \in \delta_m^-(i)} \sum_{n=1}^{|\tau_e^m|} x_{e,n}^m + \sum_{\substack{m' \in M \\ m' \neq m}} x_i^{m'm} = y_i^m \quad \forall i \in V \setminus \{s\},\ \forall m \in M \tag{3.7}$$

$$\sum_{e \in \delta_m^+(i)} \sum_{n=1}^{|\tau_e^m|} x_{e,n}^m + \sum_{\substack{m' \in M \\ m \neq m'}} x_i^{mm'} = y_i^m \quad \forall i \in V \setminus \{t\},\ \forall m \in M \tag{3.8}$$

$$\sum_{m \in M} \sum_{e \in \delta_m^+(i)} \sum_{n=1}^{|\tau_e^m|} x_{e,n}^m \leq 1 \quad \forall i \in V \tag{3.9}$$

$$\sum_{m \in M} \sum_{e \in \delta_m^-(i)} \sum_{n=1}^{|\tau_e^m|} x_{e,n}^m \leq 1 \quad \forall i \in V \tag{3.10}$$

$$\sum_{m \in M} \sum_{e \in \delta_m^-(s)} \sum_{n=1}^{|\tau_e^m|} x_{e,n}^m = 0 \tag{3.11}$$

$$x_{e,n}^{m} \in \{0,1\} \qquad \forall m \in M;\ \forall e \in E;\ \forall n = 1,...|\tau_e^m| \tag{3.12}$$

$$x_i^{mm'} \in \{0,1\} \qquad \forall m, m' \in M;\ \forall i : (i,i)_{m,m'} \in E_{\text{trans}} \tag{3.13}$$

$$y_i^m \in \{0,1\} \qquad \forall i = 1,...,|V|\,;\ \forall m \in M \tag{3.14}$$

Ist ein Pfad also eine zulässige Lösung des ILPs, so hat dieser folgende Eigenschaften:

- Start- und Zielknoten sind Teil des Lösungssubgraphen (Gleichungen (3.5) und (3.6)).
- Die Abfahrtszeit im Startknoten s ist größer oder gleich der vorgegebenen Startzeit.
- In jedem Knoten ist die Abfahrtszeit größer oder gleich der Ankunftszeit im selbigen.
- In Start- und Zielknoten wird nicht umgestiegen (Gleichungen (3.5) und (3.6)).
- Der Startknoten wird verlassen (Ungleichung (3.4)) und zwar genau ein Mal (Gleichung (3.8)).
- Jeder Knoten, der betreten wird, mit Ausnahme des Endknotens t, wird auch wieder verlassen (Ungleichung (3.3)) und zwar genau ein Mal (Gleichungen (3.8) und (3.5)).
- Jede Menge von Knoten, die eine gleiche Station in unterschiedlichen Modi symbolisieren wird bei einem gültigen Lösungspfad genau einmal betreten (3.10) beziehnugsweise verlassen (3.9). Das heißt, an einer Station (nicht Knoten!) liegt genau eine eingehende und eine ausgehende Kante $e \in E$, die keine Umstiegskante ist. Szenarien wie in Abbildung 3.1 – Station 1 wird hier zweimal angefahren (in unterschiedlichen Modi) und zweimal verlassen – sind nicht möglich.

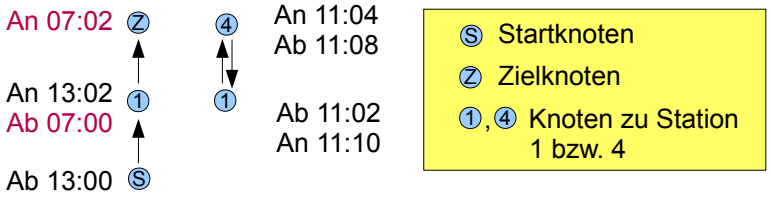


Abbildung 3.1: Zwei separate Ein-/Ausgänge bei Station 1

- Zusätzlich wird jeder Knoten eines gültigen Pfades bis auf den Startknoten s nur genau einmal betreten (3.7).

- Damit kann ein Weg, der in s beginnt nicht in einem Knoten $i \in V \setminus \{s, t\}$ enden.

- Ein Weg, der in s startet, kann wegen Bedingung (3.11) nicht in s enden.

- Es gibt, wie in Abbildung 3.2 zu sehen, zwei Möglichkeiten wie eine in den Lösungssubgraphen eingebundene Subtour aussehen kann. Da s nicht betreten werden darf (3.11), kann die Subtour um s so nicht auftreten, s muss also mit der Subtour um t verbunden sein. Nach (3.7) und (3.8) kann an jedem Knoten eines zulässigen Lösungssubgraphen nur eine abgehende und eine eingehende Kante anliegen. Der in Abbildung 3.2b abgebildete Fall ist also nicht möglich.

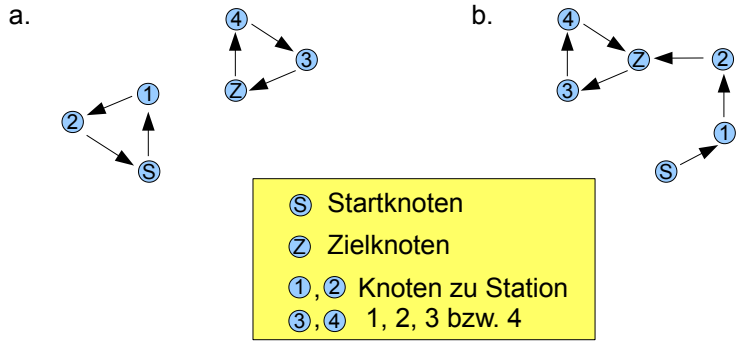


Abbildung 3.2: Subtour an Start- oder Zielknoten

- Eine Subtour wie in Abbildung 3.3 zu sehen ist ebenfalls ausgeschlossen, wenn die Gleichungen (3.7) und (3.8) sowie Ungleichung (3.3) gelten.

- Ein Pfad der in s beginnt muss in t enden. Dafür gibt es zwei Gründe: Zum Einen muss aufgrund von Ungleichung 3.3 (bzw. (3.8)) aus jedem Knoten, der nicht t ist, eine Kante herausführen. Zum Anderen kann keine Tour beziehungsweise Subtour auftreten, sodass keine zulässige Lösung ausgegeben werden kann, in der an jedem Knoten (außer dem Startknoten) eine eingehende und eine ausgehende Kante anliegen.

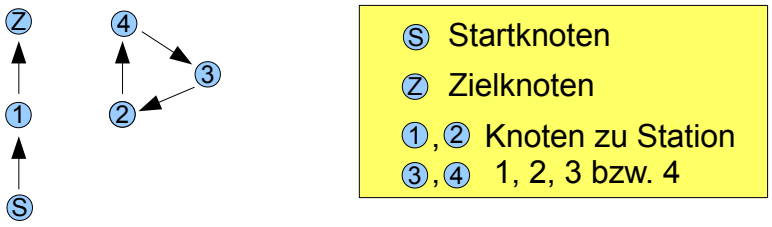


Abbildung 3.3: Abgetrennte Subtouren

3.3 Komplexität des Earliest Arrival Problems

Im Folgenden wird die Komplexität des EAPs in Abhängigkeit der Gültigkeit von Ungleichung (2.6) beziehungsweise von (2.7) und (2.8) aus Kapitel 2 betrachtet und der zugehörige Beweis nach [KS93] skizziert. Zur Wiederholung seien die Ungleichungen (2.7) und (2.8) für ein gegebenes Kantengewicht $\tilde{f}_e^m$ beziehungsweise ein gegebenes Umstiegsgewicht erneut aufgeführt:

$$\forall T' > T : T' + \tilde{f}_e^m(T') \geq T + \tilde{f}_e^m(T) \quad \forall e \in E_m, \ \forall m \in M \tag{2.7}$$

$$\forall T' > T : T' + d_i^{mm'} \geq T + d_i^{mm'} \quad \forall (i,i)_{m,m'} \in E_{\text{trans}}, \ i \in V_m^{\leftrightarrow} \cap V_{m'}^{\leftrightarrow}. \tag{2.8}$$

Im Beweis selbst wird der Begriff „Label-Setting-Verfahren“ (LS-Verfahren) verwendet. Ein LS-Verfahren ist eine Gattung der sogenannten „Markierenden Algorithmen “ (*engl*: labeling algorithms).
Markierende Algorithmen sind eine Klasse von Algorithmen für Pfadfindungsprobleme in Graphen und Netzwerken. Es gibt dabei zwei Arten von Markierenden Verfahren, das Label-Setting-Verfahren und das Label-Correcting-Verfahren [KS93].
Der Begriff des LS-Verfahrens beschreibt dabei ein Lösungsverfahren, das in jeder Iteration einen Knoten mit dem tatsächlichen Wert eines vom Startknoten aus kürzesten Weges markiert. Eine getroffene Entscheidung zur Markierung wird nicht revidiert und auch der Wert der Markierung wird nicht nachträglich noch einmal verbessert. Der „abgearbeitete“ Knoten wird kein zweites Mal überprüft. Ein Beispiel für ein solches LS-Verfahren ist der Dijkstra Algorithmus.
Der Begriff des Label-Correcting-Verfahrens wird in Kapitel 4 wieder aufgegriffen und soll dort genauer erklärt werden.

Definition (Anpassendes LS-Verfahren). Angenommen es existiert ein Label-Setting-Verfahren für die Berechnung eines kürzesten Pfades auf einem statischen, zeitexpandierten

Graphen $G = (V, E)$, welches für einen Knoten $i \in V$ das Label v_i ermittelt. Dann ist ein anpassendes LS-Verfahren ein Verfahren, das für den zugehörigen zeitabhängigen Graphen $\overleftrightarrow{G}$ mit der Kantengewichtsfunktion $\tilde{f}_{(i,j)}(T)$ das Label von $j \in V$ bei Betrachtung der Kante $(i, j) \in E$ mit Hilfe des bestimmten Labels von v_i von $i \in V$ wie folgt setzt:

$$v_j = v_i + \tilde{f}_{(i,j)}(v_i)$$

anstatt den Knoten zu expandieren. □

Zu zeigen bleibt, dass ein solches „anpassendes LS-Verfahren“ im dynamischen Fall eine Lösungsmethode für das EAP darstellt, sofern die Ungleichungen 2.7 und 2.8 erfüllt sind.

Satz. Das EAP in Netzwerken mit Kantengewichtsfunktionen $\tilde{f}_e^m$, welche die Ungleichungen 2.7 und 2.8 erfüllen, ist in polynomieller Zeit lösbar.

Beweis (Beweisskizze).

(i) Gilt die Eigenschaft 2.7, so gilt, dass ein anpassender LS-Algorithmus $\tilde{L}$ ein Label-Setting-Verfahren für dynamische, also zeitabhängige Netzwerke darstellt, falls der unterliegende Algorithmus L für das statische Netzwerk ein Label-Setting-Verfahren ist.

- Der Beweis führt über das statische, zeitexpandierte Modell: Angenommen es gibt ein LS-Verfahren L für das zeitexpandierte Modell. Dieses betrachtet dabei alle möglichen Reisezeiten von einem Knoten i zu einem anderen Knoten j.
- Da L ein LS-Verfahren ist, ist das Label v_i von i, das in dem hier behandelten Fall die Ankunftszeit in i ist, optimal (also minimal) sobald der nächste Schritt von i aus betrachtet wird.
- Sei L nun das zugrunde liegende Verfahren für das anpassende LS-Verfahren $\tilde{L}$. Da $\tilde{L}$ das Label als
$$v_j = v_i + \tilde{f}_{(i,j)}(v_i)$$
setzt und v_i minimal ist, gilt mit Ungleichung 2.7, dass v_j das minimale, von i aus erzielbare Label ist. Somit ist v_j auch bei Anwendung des Verfahrens $\tilde{L}$ optimal, sobald die Betrachtung von j abgeschlossen ist.

(ii) Gilt Ungleichung 2.7, so löst ein anpassendes LS-Verfahren $\tilde{L}$ zur Berechnung von kürzesten Wegen ein EAP für zwei gegebene Punkte optimal.
Eine obere Schranke für den Berechnungsaufwand ist dabei identisch mit der Schranke eines entsprechenden statischen Systems.

- Da für die Kürzester-Pfad-Berechnung für jede Kante nur die minimal mögliche Reisezeit, also ein einziges Gewicht benutzt wird, existiert ein statisches Netzwerk mit identischer Topologie, aber festen Kantengewichten.
- Auf diesem statischen Netzwerk operiert das LS-Verfahren L wie das anpassende LS-Verfahren $\tilde{L}$ auf dem dynamischen Netzwerk
  (siehe auch (i)).
- Da die Laufzeitschranke lediglich von der Topologie des Netzwerkes abhängt, ist diese im statischen und dynamischen Fall gleich.

(iii) Da der Dijkstra-Algorithmus das entsprechende statische System in polynomieller Zeit lösen kann, ist das Problem auch im dynamischen Fall in polynomieller Zeit lösbar. □

Kapitel 4

Lösungsmethoden

Der nachfolgende Abschnitt beschäftigt sich zunächst mit Algorithmen zur Lösung des präsentierten Problems der frühestmöglichen Ankunft (EAP). Wie bereits am Ende des vorhergehenden Kapitels erwähnt, kann unter gewissen Veraussetzungen das Zeitabhängiger-Kürzester-Pfad-Problem und damit in diesem Fall auch das EAP durch verallgemeinerte LS-Verfahren gelöst werden. Zu diesen Verfahren gehört unter anderem der Dijkstra Algorithmus, der hier zunächst für den einfacheren, monomodalen Fall eingeführt und anschließend für die Anwendung auf ein multimodales Netzwerk erweitert wird. Anschließend wird eine weitere Möglichkeit für die Lösung des EAPs im öffentlichen Personennahverkehr vorgestellt.
Die in diesem Kapitel gezeigten Algorithmen sind dabei für den statischen Fall konzipiert, das heißt die Kantengewichte werden als zeitunabhängig betrachtet. Auf die realistischere Implementierung mit zeitabhängigen Kantengewichten und einer gegebenen Startzeit soll erst in Kapitel 5 eingegangen werden.
Neben den aufgezählten Lösungsverfahren gibt es noch weitere Methoden wie etwa genetische Algorithmen [HF], die in dieser Arbeit aber nicht näher betrachtet werden.

4.1 Der Dijkstra Algorithmus

Der Dijkstra Algorithmus ist ein LS-Verfahren, das – wie im vorherigen Kapitel gezeigt – auch bei einer Erweiterung zu einem Multimodalen Dijkstra Algorithmus (MMD) das Frühestmögliche-Ankunft-Problem (EAP) exakt löst. Es soll nun zuerst der ursprüngliche Dijkstra Algorithmus skizziert werden, um anschließend die für den Multimodalen Dijkstra Algorithmus nötigen Erweiterungen deutlich machen zu können.
Das grundlegende Vorgehen des Verfahrens nach Dijkstra ist im nachfolgenden Algorithmus 1 zu sehen. Dabei seien die Nachbarn eines jeden Knoten $u \in V$ bekannt und deren Anzahl mit $deg(u)$ bezeichnet. Desweiteren wird eine Prioritätenliste Q verwendet, die Paare der Form $(u, Abstand[u])$ enthält und diese nach aufsteigendem $Abstand[u]$ vom Startknoten ordnet. Die Bezeichnung $c(u, v)$ gibt in den folgenden Algorithmen den direkten Abstand zwischen einem Knoten $u \in V$ und einem Knoten $v \in V$ aus dessen Nachbarschaft $N(u)$ an.

Der monomodale Dijkstra Algorithmus

Algorithmus 1 : Dijkstra Algorithmus

Eingabe: Digraph $G = (V, E)$, zugehörige Gewichte $c : E \to \mathbb{R}_0^+$, Startknoten $s \in V$
Ausgabe: Abstände $D : V \to \mathbb{R}_0^+ \cup \{\infty\}$ vom Startknoten s (auf kürzestem Weg)

```
for alle Knoten v ∈ V do
    D[v] := ∞;
end
Abstand Startknoten D[s] := 0;
Q := {(v, D[v]), v ∈ V} ;
while Q ≠ ∅ do
    u := Knoten aus vorderstem Paar in Q;
    Q := Q\(u, D[u]);
    for v ∈ N(u) ∩ Q do
        if D[u] + c(u, v) < D[v] then
            D[v] := D[u] + c(u, v);
        end
    end
end
```

Algorithmus 1 setzt dabei zunächst für alle Knoten den jeweiligen Abstand zum Startknoten auf unendlich und den Abstand des Startknoten zu sich selbst auf „0“. Die Liste Q enthält zunächst alle Knoten aus V und die zugehörigen Gewichte, die anfangs einheitlich den Wert „∞“ besitzen. Innerhalb eines while-Schleifendurchlaufs wird dann der Knoten mit dem geringsten Abstand zum Anfangsknoten aus Q gewählt und der Abstand D von dessen Nachbarn (die sich noch in Q befinden) aktualisiert. Der neue Abstand ist dann $\min\{D[u] + c(u, v), D[v]\}$. Sind alle Nachbar, die sich noch in Q befinden, durchlaufen, so wird der Knoten aus der Liste Q gelöscht.
Die Laufzeit des Algorithmus 1 ist dabei abhängig von der Datenstruktur der Prioritätenliste Q.

Satz. Sei $G = (V, E)$ mit $|V| = n$ und $|E| = m$. Algorithmus 1 hat die Laufzeit $O(n^2)$, falls die Prioritätenliste durch ein gewöhnliches Feld dargestellt wird. Ist die Struktur dieser Liste ein Fibonacci-Heap so beträgt die Laufzeit $O(m + n\log(n))$

Beweis. Die while-Schleife wird n-mal durchlaufen. Innerhalb der while-Schleife:
Ist die Prioritätenliste wie ein gewöhnliches Feld aufgebaut, so benötigt man eine Laufzeit der Größe $O(n)$ für „finde Knoten u mit minimalen Abstand“ und für jeden Knoten u eine Aufwand von $O(deg(u))$ für das Update von den zu den Nachbarn gehörigen Gewichten. Insgesamt ergibt sich somit ein Aufwand von

$$O(n^2 + \sum_{v \in V} deg(v)) = O(n^2 + m) = O(n^2)$$

für die Berechnung des kürzesten Weges von einem Startknoten zu allen anderen Knoten. Ist die Prioritätenliste als Fibonacci-Heap implementiert, so beträgt die Laufzeit des oben aufgeführten Algorithmus

$$O(m + n\log(n))$$

(siehe Beweisforsetzung, Seite 34) □

Für die Definition eines „Fibonacci-Heap“ soll an dieser Stelle ein kleiner Exkurs in die Welt der Datenstrukturen gemacht werden. Für weitere Details zu Fibonacci-Heap Strukturen sei auf die Literatur [Way; AS05; FT84] verwiesen.
Ein Fibonacci-Heap ist eine Ansammlung von sogenannten „heapgeordneten Bäumen“ (siehe Abbildung 4.1).

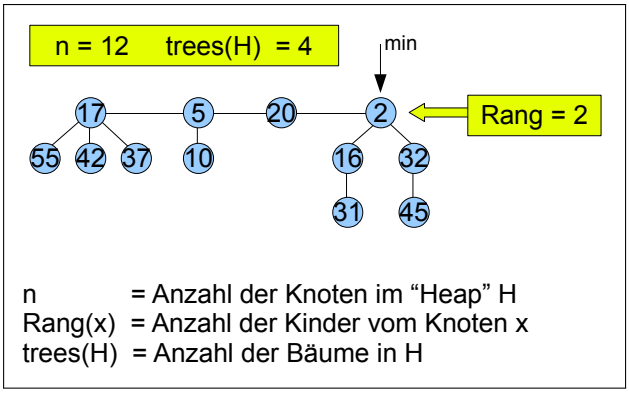


Abbildung 4.1: Vier verbundene, heapgeordnete Bäume

Heapgeordnete Bäume sind so implementiert, dass Nachfolgeknoten stets ein größeres Gewicht haben als ihr Vaterknoten. Dabei soll keine der Wurzeln gleich viele Nachfolger besitzen.
Bei der Zeigerstruktur eines Fibonacci-Heaps geht nur ein Zeiger vom Vaterknoten aus, der auf genau eines der Kinder (Nachfolger) zeigt. Die Geschwisterknoten sind unterdessen untereinander in einer zyklisch geschlossenen, doppelt verketteten Liste miteinander verknüpft (siehe Abbildung 4.2). Außerdem hat jeder Knoten einen Zeiger auf seinen Vater [AS05]. Die Wurzeln der einzelnen heapgeordneten Bäume sind ebenfalls untereinander in einer zyklisch geschlossenen, doppelt verketteten Liste verknüpft.
Darüber hinaus werden in jedem Knoten Gewicht und Rang, also Priorität und Anzahl der Kinder gespeichert. In diesem Fall ist die Priorität nach aufsteigendem Gewicht gesetzt, je niedriger desto wichtiger.
Man benötigt nun noch einen Hauptzeiger, der auf das kleinste Element zeigt, das aufgrund der Heap-Eigenschaft in einer der Wurzeln zu finden ist.
Zuletzt wird für die Anwendung jeder Knoten mit einer Boolesche Variable versehen, die später angibt, ob der Knoten bereits vom LS-Verfahren markiert ist oder nicht.

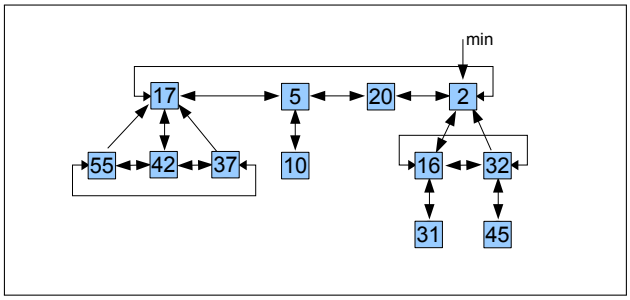


Abbildung 4.2: Einen Fibonacci-Heap repräsentierende Zeiger

Beweis (Fortsetzung). Zu zeigen bleibt die folgende Aussage: Ist die Prioritätenliste implementiert als Fibonacci-Heap, so beträgt die Laufzeit des oben aufgeführten Algorithmus

$$O(m + n \log(n))$$

Die Verwendung von Fibonacci-Heaps greift an der Stelle „finde Knoten u mit minimalen Abstand und lösche diesen Knoten aus der Prioritätenliste“ ein. Aufgrund des Hauptzeigers benötigt man nun nur noch $O(1)$ Operationen für „finde Knoten u“. Dafür wird etwas mehr Rechenaufwand für das Löschen aus der Prioritätenliste benötigt. Da der Vaterknoten herausgelöscht wird, fungieren die direkten Nachfolger (Kinder) des gefundenen Knoten u nun als Wurzeln heapgeordneter Bäume. Zusätzlich muss der Hauptzeiger wieder auf das aktuell kleinste Element gesetzt werden (eines der Wurzeln). Da nun unter Umständen zwei „Bäume“ die gleiche Anzahl an Nachfolgern haben, müssen diese gegebenfalls noch zusammengeführt werden. Dieser Prozess benötigt einen amortisierten Aufwand von $O(log(n))$ (für eine anschauliche Darstellung hierzu siehe [FT84]) □

Der Multimodale Dijkstra Algorithmus

Für den multimodalen Fall müssen nun Änderungen an Algorithmus 1 vorgenommen werden. Der Pseudocode für den Multimodalen (zeitunahängigen) Dijkstra Algorithmus ist im nachfolgenden Algorithmus 2 dargestellt. Die Prioritätenliste Q soll dabei dieselbe Gestalt haben wie im zuletzt beschrieben, monomodalen Fall. Zur Erinnerung sei zudem erwähnt, dass

$$G = \bigcup_{m \in M} G_m(V_m, E_m) \quad \text{sowie} \quad V = \bigcup_{m \in M} V_m$$

gelten, E_{trans} die Menge der Umstiegskanten und M die Menge der Modi bezeichnet. Die Funktion der Reisezeiten T_m sei hier aufgrund der zeitunabhängigen Betrachtung außer Acht gelassen.
Die Bezeichnung $c_m(u, v)$ gibt ähnlich wie im monomodalen Fall den direkten Abstand zwischen einem Knoten $u \in V_m$ und einem Knoten $v \in V_m$ aus dessen Nachbarschaft (im Modus m) an.

Algorithmus 2 : Multimodaler Dijkstra Algorithmus

Eingabe: Graph $\overleftrightarrow{G}(G, E_{\text{trans}}, M)$, Gewichte $c_m : E_m \to \mathbb{R}_0^+ \ \forall m \in M$,
Umstiegsgewichte $c_{\text{trans}} : E_{\text{trans}} \to \mathbb{R}^+$, Startknoten $s \in V$
Ausgabe: Abstände $D : V \to \mathbb{R}_0^+ \cup \{\infty\}$ vom Startknoten s (auf kürzestem Weg)

$Q := \emptyset$;
for *alle Knoten* $v \in V$ **do**
  $D[v] := \infty$;
  $\text{Mode}_{\text{alt}}[v] := NULL$;
end
Hilfsgewicht $w := NULL$;
Umstiegsgewicht $w_{\text{um}} := NULL$;
$D[s] := 0$;
Füge $(s, D[s])$ zu Q hinzu;
while $Q \neq \emptyset$ **do**
  $u :=$ Knoten aus vorderstem Paar in Q;
  $Q := Q \backslash (u, D[u])$;
  if *u ist „betrachtet“* **then**
    continue;
  end
  for $i = 1...deg(u)$ **do**
    $v :=$ Nachbar i;
    $m :=$ Modus für Fahrt von u nach v;
    if $u = s$ **then**
      $Mode_{alt}[u] := m$;
    end
    if $Mode_{alt}[u] = m$ **then**
      $w := c_m(u, v)$;
    else
      $w_{\text{um}} := c_{\text{trans}}(u \ in \ \text{Mode}_{\text{alt}}[u], u \ in \ m)$;
      $w := c_m(u, v) + w_{\text{um}}$;
    end
    if *v noch nicht „betrachtet“* ***and*** $D[u] + w < D[v]$ **then**
      $D[v] := D[u] + w$;
      $\text{Mode}_{\text{alt}}[v] := m$;
      Füge $(v, D[v])$ zu Q hinzu;
    end
  end
  Setze u als „betrachtet“;
end

Für den multimodalen Dijkstra Algorithmus werden, wie in Algorithmus 2 zu sehen, weitere Variablen für Abfrage und Zuordnung der einzelnen Modi benötigt. Zunächst wird also in der Initialisierungsphase die Variable $\text{Mode}_{\text{alt}}[u]$ eingeführt. In dieser wird später der Modus gespeichert, in dem man, bei dem bis dahin berechneten s-u-Pfad, in Knoten $u \in V$ angekommen ist. Da gemäß den Annahmen dieser Arbeit am Startknoten nicht umgestiegen werden darf, wird für den Startknoten s die Variable $\text{Mode}_{\text{alt}}[s]$ gleich dem aktuellen Modus gesetzt.
Eine zusätzliche Erweiterung ist die neu eingeführte Fallunterscheidung. Hier wird je nachdem, ob für die aktuell betrachtete, von u abführende Wegstrecke zu einem Knoten $v \in V$ umgestiegen werden muss oder nicht, die Umstiegsdauer bei der Aktualisierung des Abstands $D[v]$ berücksichtigt bzw. außer Acht gelassen. Dass der Algorithmus die richtige Lösung findet, selbst wenn sich das Umsteigen erst auf lange Sicht lohnt, ist in der nachfolgenden Abbildung illustriert. Gesucht wird hier ein kürzester s-t-Pfad. Die in der Abbildung 4.3 dargestellten Schritte (1. - 6.) entsprechen den einzelnen Schritten des Algorithmus bis zur Lösungsfindung.

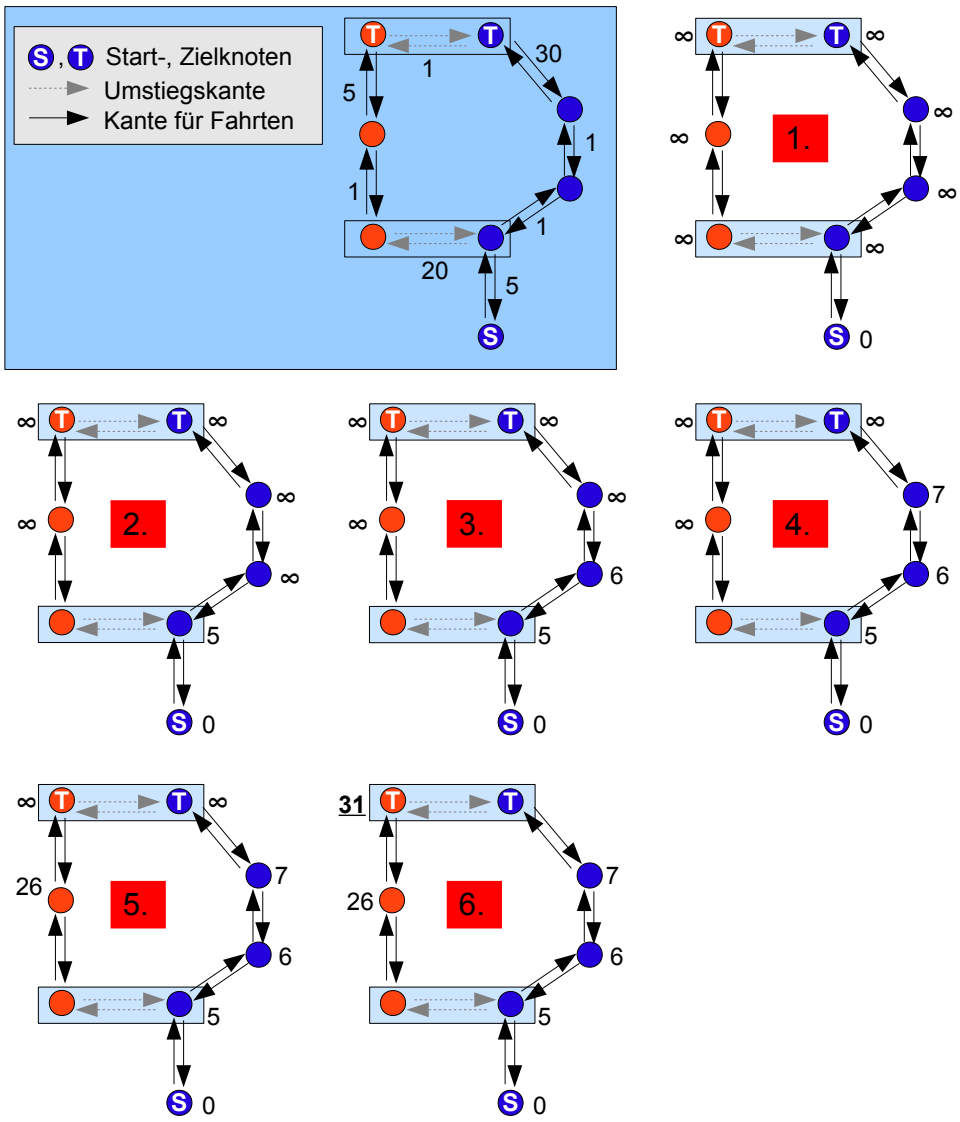


Abbildung 4.3: Berücksichtigung langfristig rentabler Umsteigemöglichkeiten

Eine vollständige Implementierung des Multimodalen Dijkstra Algorithmus in C++ wird in Kapitel 5 genauer erläutert.

Satz. Sei $\overleftrightarrow{G} = (G, E_{\text{trans}}, M)$ ein gegebener Umsteigegraph mit $G = \{G_m = (V_m, E_m) : m \in M\}$ und $|M| = k$. Dann benötigt der Multimodale Dijkstra Algorithmus eine Laufzeit von $O(\sum_{m=1}^{k} |V_m|^2)$ falls die Prioritätenliste durch ein gewöhnliches Feld dargestellt wird.

Beweis. Die while-Schleife wird $(\sum_{m=1}^{k} |V_m|)$-mal durchlaufen.
Ist die Prioritätenliste wie ein gewöhnliches Feld aufgebaut, so benötigt man in der while-Schleife eine maximale Laufzeit von $O(\sum_{m=1}^{k} |V_m|)$ für „finde den Knoten v mit minimalen Abstand“. Für den gefundenen Knoten v werden anschließend die Nachbarn, die sich in allen Modi $m \in M$ befinden können, in denen v vertreten ist und deren Anzahl gleich $\deg(v)$ ist, in $O(deg(v))$ durchlaufen. Insgesamt ergibt sich somit eine Laufzeit von

$$O(\sum_{m=1}^{k} |V_m|^2 + \sum_{v \in \bigcup_{m \in M} V_m} deg(v)) = O(\sum_{m=1}^{k} |V_m|^2).$$

□

Analog zum Algorithmus 1 kann auch die Laufzeit des Multimodalen Dijkstra Algorithmus durch einen Fibonacci-Heap verbessert werden.

4.2 Weitere Lösungsverfahren

Neben dem Dijkstra Algorithmus gibt es weitere Verfahren, die erfolgreich zur Lösung des EAP in multimodalen Systemen eingesetzt werden können.
So ist etwa ein verallgemeinerter A*-Algorithmus unter speziellen Voraussetzungen [Pea84] ebenfalls ein LS-Verfahren und kann zur Lösung des Problems, wie in Kapitel 3 gezeigt, angewendet werden.
Was in Kapitel 3 noch nicht erwähnt wurde ist, dass auch eine andere Art der markierenden (labeling) Algorithmen, das sogenannte „Label-Correcting“-Verfahren, das EAP im multimodalen Fall lösen kann [KS93]. Der Beweis, das ein solches Verfahren das EAP löst, ist dabei ähnlich aufgebaut wie der Beweis bezüglich des Label-Setting-Verfahrens in Kapitel 3. Ein Label-Correcting-Verfahren ist eine Methode, bei der in jeder Iteration eine Markierung (Label) mit der Schätzung eines vom Startknoten aus kürzesten Weges gesetzt wird. Bei Termination haben schließlich alle Markierungen den exakten Wert des kürzesten Weges.
Der Algorithmus von Bellman und Ford ist ein solcher Algorithmus und findet einen kürzesten Pfad zwischen zwei Punkten in einem Graphen ohne negative Zyklen. Zu bemerken ist, dass hier – im Gegensatz zum Dijkstra Algorithmus – zwar keine negativen Zyklen, jedoch negative Kantengewichte erlaubt sind.

Der monomodale Algorithmus von Bellman und Ford

Der Algorithmus von Bellman und Ford geht im Allgemeinen bei der Lösungssuche wie in Algorithmus 3 beschrieben vor:

Algorithmus 3 : Algorithmus von Bellman und Ford

Eingabe: Digraph $G = (V, E)$, zugehörige Gewichte $c : E \to \mathbb{R}$, Startknoten $s \in V$
Ausgabe: Abstand $D : V \to \mathbb{R} \cup \{\infty\}$ vom Startknoten s (auf kürzestem Weg)

```
for alle Knoten v ∈ V do
    D[v] := ∞;
end
Abstand Startknoten D[s] := 0;
for i = 1...|V| − 1 do
    for alle Kanten (u, v) ∈ E mit u, v ∈ V do
        if D[u] + c(u, v) < D[v] then
            D[v] := D[u] + c(u, v);
        end
    end
end
for alle Kanten (u, v) ∈ E mit u, v ∈ V do
    if D[u] + c(u, v) < D[v] then
        STOPP „Es gibt negativen Zyklus“;
    end
end
```

Algorithmus 3 setzt, wie der Dijkstra Algorithmus zuvor, zunächst die Abstände $D[V]$ für alle $v \in V$ auf unendlich und den Abstand des Startknotens zu sich selbst auf 0. Anschließend wird so lange iteriert, bis keine Verbesserung der Abstände mehr möglich ist. Da jeder kürzeste Weg (falls existent) aus höchstens $|V| - 1$ Kanten besteht und in jedem Iterationsschritt i Teillösungen für kürzeste Wege unter Verwendung von maximal i Kanten entstehen, sind $|V| - 1$ Iterationsphasen ausreichend. In jedem Iterationsschritt wird nun für jede Kante $(u, v) \in E$ geprüft, ob der Weg über u den Abstand von v zum Startknoten verbessert und $D[v]$ gegebenenfalls aktualisiert.

Satz. Sei $G = (V, E)$ mit $|V| = n$ und $|E| = m$. Dann ist die Laufzeit für den Algorithmus 3 von Bellman und Ford nach oben beschränkt durch $O(n \cdot m)$.

Beweis. Die innere for-Schleife besteht aus der Relaxationsoperation ($O(1)$) und wird jeweils m mal ausgeführt. Es ergibt sich somit ein Aufwand von $O(m)$. Die äußere for-Schleife wird $(n - 1)$-mal ausgeführt, jeweils mit dem Aufwand von $O(m)$, für die geschachtelte for-Schleife sind somit $O(n \cdot m)$ zu kalkulieren. Die Überprüfung auf negative

Zyklen kostet $O(m)$. Insgesamt ergibt sich somit eine Laufzeit von

$$O(n + n \cdot m + m) = O(n \cdot m).$$ □

Der multimodale Algorithmus von Bellman und Ford

Soll der Algorithmus von Bellman und Ford nun auf den multimodalen Fall angewendet werden, benötigt man – ähnlich wie beim Dijkstra Algorithmus – einige Anpassungen. Da bei dem Anwendungsbereich dieser Arbeit keine negativen Kantengewichte und somit auch keine negativen Zyklen auftreten können, sei im Folgenden die Abfrage bezüglich der Existenz negativer Zyklen vernachlässigt. Soll das Anwendungsgebiet jedoch beispielsweise auf den Güterverkehr ausgeweitet werden, könnten durchaus negative Gewichte auftreten. So könnten die Kantengewichte den Gewinn pro Teilstrecke darstellen, der unter Umständen auch negativ sein kann, wenn die Einnahmen an der Station an dem die Kante endet die Transportkosten auf dieser Kante unterschreiten.
In [LL09] wird ein vereinfachter multimodaler Bellman und Ford Algorithmus vorgestellt, der gewissen Einschränkungen unterliegt (siehe unten). Der nachfolgende Algorithmus soll, angelehnt an die in [LL09] vorgestellte Methode, ein jedoch allgemeiner gültiges Lösungsverfahren nach Bellman und Ford darstellen.

Algorithmus 4 : Multimodaler Algorithmus von Bellman und Ford

Eingabe: Graph $\overleftrightarrow{G}(G, E_{\text{trans}}, M)$, Gewichte $c_m : E_m \to \mathbb{R}$, Umstiegsgewichte $c_{\text{trans}} : E_{\text{trans}} \to \mathbb{R}$, Startknoten $s \in V$
Ausgabe: Abstand $D : V \to \mathbb{R} \cup \{\infty\}$ vom Startknoten s (auf kürzestem Weg)

```
for alle Knoten v ∈ V do
    D[v] := ∞;
    Mode_alt[v] := NULL;
end
Abstand Startknoten D[s] := 0;
for m' = 1... |M| do
    for i = 1... |V_m'| − 1 do
        for m = 1... |M| do
            for alle Kanten (u, v) ∈ E_m mit u, v ∈ V_m do
                if u = s then
                    Mode_alt[u] := m;
                end
                if Mode_alt[u] = m then
                    if D[u] + c_m(u, v) < D[v] then
                        D[v] := D[u] + c_m(u, v);
                        Mode_alt[v] := m;
                    end
                else
                    if Mode_alt(u) ≠ NULL then
                        w_um := c_trans(u in Mode_alt(u), u in m);
                        if D[u] + c_m(u, v) + w_um < D[v] then
                            D[v] := D[u] + c_m(u, v) + w_um;
                            Mode_alt[v] := m;
                        end
                    end
                end
            end
        end
    end
end
```

Ist es möglich die Liste der Modi M so zu ordnen, dass ein Modus abgeschlossen ist sobald der nächste betrachtet wird (dafür dürfte die Fahrt in dem neu betrachteten Modus zu keiner Verbesserung bei den bereits betrachteten Modi führen), so kann die nochmalige Summierung über die Modi vor Beginn der for-Schleife über die Kanten weggelassen werden und man erhält einen vereinfachten multimodalen Bellman-Ford-Algorithmus.

Fälle wie in Abbildung 4.4 sind dann aber nicht mehr optimierbar, da M so sortiert werden müsste, dass zunächst Modus Grün, anschließend Modus Blau und zuletzt Modus Orange durchlaufen wird. Dadurch würde der optimale s-t-Weg über Orange zurück zu Blau nicht in Betracht gezogen werden.

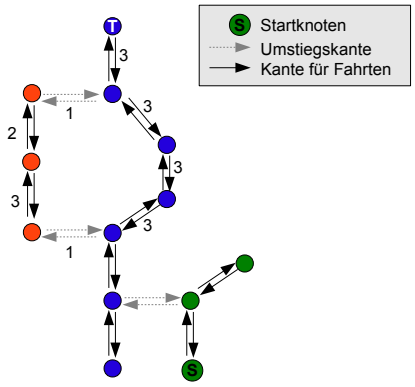


Abbildung 4.4: Beispiel für Beschränkung des vereinfachten multimodalen Bellman-Ford-Algorithmus

In [LL09] wird experimentell gezeigt, dass sich der Multimodale Dijkstra Algorithmus aufgrund der Laufzeit wesentlich besser für die reale Implementierung eignet als der (vereinfachte) Multimodale Algorithmus von Bellman und Ford. Theoretisch lässt sich über die Laufzeitschranke des Algorithmus 4 folgende Aussage treffen:

Satz. Sei $\overleftrightarrow{G} = (G, E_{trans}, M)$ mit $G = \{G_m = (V_m, E_m) : \ m \in M\}$ und $|M| = k$. Dann ist die Laufzeit für den Multimodalen Algorithmus 4 von Bellman und Ford nach oben beschränkt durch $O\left(\sum_{m=1}^{k} |V_m| \left(\sum_{j=1}^{k} |E_j|\right)\right)$.
Ist eine Sortierung der Menge M derart möglich, dass die nochmalige Summierung über die Modi vor Beginn der for-Schleife über die Kanten außer Acht gelassen werden kann, so ist die obere Schranke gegeben durch $O\left(\sum_{m=1}^{k} |V_m| \, |E_m|\right)$.

Beweis. Für jeden Modus m wird eine for-Schleife $(|V_m| - 1)$-mal durchlaufen. Diese beinhaltet eine weitere, innere for-Schleife, die $O(\sum_{j=1}^{k} |E_j|)$-mal (beziehungsweise $O(|E_m|)$-mal für den vereinfachten Fall) Operationen im Laufzeitbereich $O(1)$ ausführt. Damit ist insgesamt für die geschachtelte for-Schleife im Modus m ein Aufwand von $O(|V_m| \left(\sum_{j=1}^{k} |E_m|\right))$ beziehungsweise $O(|V_m| \, |E_m|)$ zu kalkulieren.
Damit ergibt sich eine Laufzeitschranke von

$$O(\sum_{m=1}^{k} |V_m| \left(\sum_{j=1}^{k} |E_j|\right))$$

beziehungsweise

$$O(\sum_{m=1}^{k} |V_m|\,|E_m|).$$

□

Kapitel 5

Implementierung

5.1 Aufbereitung der Inputdaten

Die praktische Anwendung dieser Arbeit wurde durch die Münchner Verkehrsgesellschaft mbH, welche die kompletten Fahrplandaten mit dem Stand von 2012 in Form von PDF-Seiten zur Verfügung stellte, unterstützt.
Für die Weiterverwendung der Daten wurden die Fahrten-Tabellen dieser PDF-Dateien in MS Excel® importiert und aufgrund fehlender Formaterkennung re-formatiert.
Alle Taktungseinträge (zum Beispiel „alle 10 Minuten“, siehe Abbildung 5.1) wurden mit Hilfe einer Funktion in MS Excel® zu den tatsächlichen Fahrtzeiten umgewandelt und sonstige Besonderheiten herausgefiltert. In Abbildung 5.1 und 5.2 sind zwei Ausschnitte einer solchen Tabelle in PDF-Format zu sehen. Für die komplette Bereitstellung der Fahrplandaten siehe [Mün].

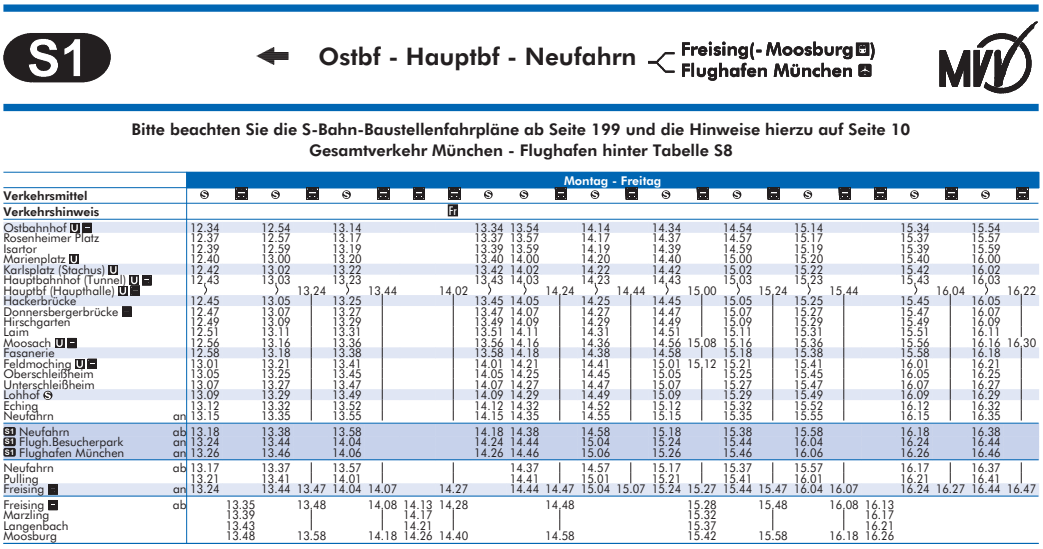
BUS 133 Rotkreuzplatz U - Aidenbachstraße U - Forstenrieder Allee U MVV

Verkehrshinweis		Montag - Freitag																								
										S			S		S				S			S				
Rotkreuzplatz U	ab		5.18	5.38		5.58		6.08	6.13		14.23	14.33		14.43		14.53		15.03				18.13				
Landshuter Allee			5.19	5.39		5.59		6.09	6.15		14.25	14.35		14.45		14.55		15.05				18.15				
Schlörstraße			5.20	5.40		6.00		6.10	6.16		14.26	14.36		14.46		14.56		15.06				18.16				
Donnersbergerstraße			5.21	5.41		6.01		6.12	6.17		14.27	14.37		14.47		14.57		15.07				18.17				
Donnersbergerbrücke S			5.22	5.42		6.02		6.13	6.19		14.29	14.39		14.49		14.59		15.09				18.19				
Trappentreustraße			5.23	5.43		6.03		6.15	6.20		14.30	14.40		14.50		15.00		15.10				18.20				
Gollierplatz			5.24	5.44		6.04		6.16	6.21		14.31	14.41		14.51		15.01		15.11				18.21				
Kazmairstraße			5.25	5.45		6.05		6.17	6.22		14.32	14.42		14.52		15.02		15.12				18.22				
Heimeranplatz U S	an		5.26	5.46		6.06		6.18	6.24		14.34	14.44		14.54		15.04		15.14				18.24				
Heimeranplatz U S	ab		5.26	5.46		6.06		6.18	6.25		14.35	14.45		14.55		15.05		15.15	alle			18.25				
Garmischer Straße			5.27	5.47		6.07		6.19	6.26		14.36	14.46		14.56		15.06		15.16	10			18.26				
Siegenburger Straße			5.28	5.48		6.08		6.20	6.27		14.37	14.47		14.57		15.07		15.17	Min			18.27				
Hinterbärenbadstraße			5.30	5.50		6.10		6.22	6.30		14.40	14.50		15.00		15.10		15.20				18.30				
Westpark U			5.31	5.51		6.11		6.23	6.32		14.42	14.52		15.02		15.12		15.22				18.32				
Luise-Kiesselbach-Platz			5.33	5.53		6.13		6.25	6.34		14.44	14.54		15.04		15.14		15.24				18.34				
Pilsenseestraße			5.34	5.54		6.14		6.27	6.36		14.46	14.56		15.06		15.16		15.26				18.36				
Höglwörther Straße			5.35	5.55		6.15		6.28	6.37	alle	14.47	14.57		15.07		15.17		15.27				18.37				
Murnauer Straße			5.36	5.56		6.16		6.29	6.39	10	14.49	14.59		15.09		15.19		15.29				18.39				
Ratzingerplatz			5.37	5.57		6.17		6.30	6.40	Min	14.50	15.00		15.10		15.20		15.30				18.40				
Aidenbachstraße U	an		5.38	5.58		6.18		6.31	6.41		14.51	15.01		15.11		15.21		15.31	———			18.41				
Aidenbachstraße U	ab	5.17	5.38	5.58		6.18		6.32	6.42		14.52	15.02	15.07	15.12	15.17	15.22	15.27	15.32		18.37		18.42	18.47			
Kistlerhofstraße		5.18	5.39	5.59		6.19		6.33	6.43		14.53	15.03	15.08	15.13	15.18	15.23	15.28	15.33		18.38		18.43	18.48			
Leutstettener Straße		5.19	5.40	6.00		6.20		6.34	6.44		14.54	15.04	15.09	15.14	15.19	15.24	15.29	15.34		18.39		18.44	18.49			
Siemensallee		5.20	5.41	6.01		6.21		6.35	6.45		14.55	15.05	15.10	15.15	15.20	15.25	15.30	15.35	alle*	18.40		18.45	18.50			
Marienstern		5.21	5.42	6.02		6.22		6.36	6.46		14.56	15.06	15.11	15.16	15.21	15.26	15.31	15.36	5	18.41		18.46	18.51			
Hofbrunnstraße		5.22	5.43	6.03		6.23		6.38	6.48		14.58	15.08	15.13	15.18	15.23	15.28	15.33	15.38	Min	18.43		18.48	18.53			
Plattlinger Straße		5.23	5.44	6.04		6.24		6.39	6.49		14.59	15.09	15.14	15.19	15.24	15.29	15.34	15.39		18.44		18.49	18.54			
Wilhelm-Busch-Straße		5.24	5.45	6.05		6.25		6.41	6.51		15.01	15.11	15.16	15.21	15.26	15.31	15.36	15.41		18.46		18.51	18.56			
Gulbranssonstraße		5.25	5.46	6.06	6.16	6.26	6.36	6.42	6.52		15.02	15.12	15.17	15.22	15.27	15.32	15.37	15.42	———	18.47		18.52	18.57			
Stäblistraße		5.26	5.47	6.07	6.17	6.27	6.37	6.43	6.53		15.03	15.13		15.23		15.33		15.43				18.53				
Munckerstraße		5.27	5.48	6.08	6.18	6.28	6.38	6.44	6.54		15.04	15.14		15.24		15.34		15.44				18.54				
Staffelseestraße		5.28	5.49	6.09	6.19	6.29	6.39	6.46	6.56		15.06	15.16		15.26		15.36		15.46	alle			18.56				
Züricher Straße		5.29	5.50	6.10	6.20	6.30	6.40	6.48	6.58		15.08	15.18		15.28		15.38		15.48	10			18.58				
Limmatstraße U		5.30	5.51	6.11	6.21	6.31	6.41	6.49	6.59		15.09	15.19		15.29		15.39		15.49	Min			18.59				
Forstenrieder Allee U	an	5.31	5.52	6.12	6.22	6.32	6.42	6.50	7.00		15.10	15.20		15.30		15.40		15.50				19.00				

Abbildung 5.1: Ausschnitt 1 aus der Fahrtentabelle der Münchner Verkehrsgesellschaft

Da die Linien während eines Tages zum Teil unterschiedliche oder unterschiedlich viele Stationen anfahren, wurden die Tabellenspalten mit Hilfe eines Algorithmus in Visual Ba-

sic umsortiert, um die Daten geschickt einlesen und zuordnen zu können. Neben weiteren Umformatierungsmaßnahmen in MS Excel® wurden auch die An- und Abfahrtszeiten in Minuten umgerechnet, sodass der Algorithmus direkt mit den aufbereiteten Daten arbeiten kann.



S1 ← Ostbf - Hauptbf - Neufahrn ⟨ Freising(- Moosburg) / Flughafen München

Bitte beachten Sie die S-Bahn-Baustellenfahrpläne ab Seite 199 und die Hinweise hierzu auf Seite 10
Gesamtverkehr München - Flughafen hinter Tabelle S8

Montag - Freitag

Verkehrsmittel		S		S		S				S	S		S		S		S			S			S		S	
Verkehrshinweis									f																		
Ostbahnhof U		12.34		12.54		13.14				13.34	13.54		14.14		14.34		14.54		15.14			15.34		15.54		
Rosenheimer Platz		12.37		12.57		13.17				13.37	13.57		14.17		14.37		14.57		15.17			15.37		15.57		
Isartor		12.39		12.59		13.19				13.39	13.59		14.19		14.39		14.59		15.19			15.39		15.59		
Marienplatz U		12.40		13.00		13.20				13.40	14.00		14.20		14.40		15.00		15.20			15.40		16.00		
Karlsplatz (Stachus) U		12.42		13.02		13.22				13.42	14.02		14.22		14.42		15.02		15.22			15.42		16.02		
Hauptbahnhof (Tunnel) U		12.43		13.03		13.23				13.43	14.03		14.23		14.43		15.03		15.23			15.43		16.03		
Hauptbf (Haupthalle) U		›		›	13.24	›	13.44		14.02	›	›	14.24	›	14.44	›	15.00	›	15.24	›	15.44		16.04	›	16.22		
Hackerbrücke		12.45		13.05		13.25				13.45	14.05		14.25		14.45		15.05		15.25			15.45		16.05		
Donnersbergerbrücke		12.47		13.07		13.27				13.47	14.07		14.27		14.47		15.07		15.27			15.47		16.07		
Hirschgarten		12.49		13.09		13.29				13.49	14.09		14.29		14.49		15.09		15.29			15.49		16.09		
Laim		12.51		13.11		13.31				13.51	14.11		14.31		14.51		15.11		15.31			15.51		16.11		
Moosach U		12.56		13.16		13.36				13.56	14.16		14.36		14.56	15.08	15.16		15.36			15.56		16.16	16.30	
Fasanerie		12.58		13.18		13.38				13.58	14.18		14.38		14.58		15.18		15.38			15.58		16.18		
Feldmoching U		13.01		13.21		13.41				14.01	14.21		14.41		15.01	15.12	15.21		15.41			16.01		16.21		
Oberschleißheim		13.05		13.25		13.45				14.05	14.25		14.45		15.05		15.25		15.45			16.05		16.25		
Unterschleißheim		13.07		13.27		13.47				14.07	14.27		14.47		15.07		15.27		15.47			16.07		16.27		
Lohhof S		13.09		13.29		13.49				14.09	14.29		14.49		15.09		15.29		15.49			16.09		16.29		
Eching		13.12		13.32		13.52				14.12	14.32		14.52		15.12		15.32		15.52			16.12		16.32		
Neufahrn	an	13.15		13.35		13.55				14.15	14.35		14.55		15.15		15.35		15.55			16.15		16.35		
S1 Neufahrn	ab	13.18		13.38		13.58				14.18	14.38		14.58		15.18		15.38		15.58			16.18		16.38		
S1 Flugh.Besucherpark	an	13.24		13.44		14.04				14.24	14.44		15.04		15.24		15.44		16.04			16.24		16.44		
S1 Flughafen München	an	13.26		13.46		14.06				14.26	14.46		15.06		15.26		15.46		16.06			16.26		16.46		
Neufahrn	ab	13.17		13.37		13.57					14.37		14.57		15.17		15.37		15.57			16.17		16.37		
Pulling		13.21		13.41		14.01					14.41		15.01		15.21		15.41		16.01			16.21		16.41		
Freising	an	13.24		13.44	13.47	14.04	14.07		14.27		14.44	14.47	15.04	15.07	15.24	15.27	15.44	15.47	16.04	16.07		16.24	16.27	16.44	16.47	
Freising	ab		13.35		13.48		14.08	14.13	14.28			14.48				15.28		15.48		16.08	16.13						
Marzling			13.39					14.17								15.32					16.17						
Langenbach			13.43					14.21								15.37					16.21						
Moosburg			13.48		13.58		14.18	14.26	14.40			14.58				15.42		15.58		16.18	16.26						

Abbildung 5.2: Ausschnitt 2 aus Fahrtentabelle der Münchner Verkehrsgesellschaft

Um nun noch die Umstiege und die benötigte Umsteigedauer - die in dieser Arbeit konstant und für jeden Umstiegspunkt einzeln gewählt wurde - anzulegen, wurde eine weitere MS Excel®-Datei mit den Einträgen $Umstiegsknoten\text{-}Modus(Ankunft)\text{-}Modus(Abfahrt)\text{-}Umsteigedauer$ angelegt.
Da einige Stationen unter unterschiedlichen Namen aufgeführt wurden, wurden

(i) die Namen angepasst, falls es sich um eine identische Station handelte (zum Beispiel „Heimeranplatz an“ und „Heimeranplatz ab“) und

(ii) eine weitere MS Excel® Datei angelegt für Stationen, die eng zusammenhängen (zum Beispiel „Hauptbahnhof Nord“ und „Hauptbahnhof (Tunnel)“). Diese Datei erweitert die Umsteigemöglichkeiten, sodass auch zwischen diesen Stationen gewechselt werden kann.

5.2 Der Algorithmus

Nachdem in Kapitel 4 bereits der statische Fall des Multimodalen Dijkstra Algorithmus eingeführt wurde, soll in diesem Abschnitt die etwas aufwendigere realistische Implemen-

tierung mit gegebener Startzeit und zeitabhängigen Kantengewichten erörtert werden. Sei der Umsteigegraph $\overleftrightarrow{G}(G, E_{trans}, M)$ mit der Funktion der Reisezeiten T_m für alle Modi $m \in M$ gegeben. Zusätzlich gebe man einen Startknoten $s \in V$, einen Endknoten $e \in V$ sowie eine Startzeit T_{start} vor. Die Umstiegsmöglichkeiten mit der jeweils benötigten Zeit sind in einer – wie im oberen Abschnitt beschriebenen – Exceltabelle gespeichert. Die Daten dieser Datei sind als Liste anzusehen und werden im Folgenden mit U bezeichnet. Gesucht wird nun der minimale Abstand $D[e]$ mit $D : V \to \mathbb{N}_0 \cup \{\infty\}$ des Endknotens e zum Startknoten s. Der Kopf des implementierten Algorithmus sieht demnach wie folgt aus:

Eingabe: Graph $\overleftrightarrow{G}(G, E_{trans}, M)$, Reisezeitfunktion T_m für $m \in M$, Startknoten $s \in V$, Endknoten $e \in V$, Umsteigeliste U, Startzeit T_{start}
Ausgabe: Abstand $D[e]$

Bevor die eigentliche Berechnung beginnen kann, werden in einem nächsten Schritt alle benötigten Variablen initialisiert. Auch hier wird die Prioritätenliste Q verwendet (siehe Kapitel 4), welche die Paare $(v, D[v])$ mit $v \in V$ nach aufsteigendem Abstand D zum Startknoten s sortiert. Neu ist in diesem Schritt die Variable $t_{alt}(v)$ mit $v \in V$, die den Wert der Ankunftszeit in dem Knoten v trägt sowie die Variable t_{ab}, die später die aktuell gewählte Abfahrtszeit angibt.

$Q := NIL$;
for *alle Knoten* $v \in V$ **do**
  $D[v] := \infty$;
end
$D[s] := 0$;
$t_{alt}(s) := T_{start}$;
Umstiegsgewicht $w_{um} := 0$;
Abfahrtszeit $t_{ab} := 0$;
Füge $(s, D[s])$ zu Q hinzu;

Nachdem die Variablen eingeführt sind, beginnt analog zu Algorithmus 2 aus Kapitel 4 die while-Schleife mit den eigentlichen Berechnungsschritten. Innerhalb dieser Schleife wird dabei zunächst wieder der Knoten $u \in V$ mit dem kleinsten bislang berechneten Abstand zum Startknoten s aus Q gewählt und anschließend aus der Prioritätenliste gelöscht. Ist das ausgewählte u der Endpunkt, so folgt der Abbruch, da der kürzeste s-e-Pfad schon gefunden ist. Sollte u schon betrachtet worden sein, so gehe man direkt in den nächsten Schleifen-Durchlauf und wähle ein neues u mit $(u, D[u]) \in Q$.
Nun beginnt die innere for-Schleife, bei der die Nachbarn von u betrachtet werden. Dabei berücksichtigt man u in allen Modi m für die gilt: $u \in V_m$.
Für die Wahl des Gewichts wird unterschieden, ob die betrachtete Verbindung zwischen u

und dem Nachbarn im gleichen Modus liegt, in dem man in u angekommen ist oder nicht. Je nach Fall wird dann eine kleinstmögliche Abfahrtszeit in u bestimmt, die entweder größer als die Ankunftszeit in u oder größer dieser Ankunftszeit plus der benötigten Umsteigezeit ist. Das Kantengewicht w für die Strecke zu dem Nachbarknoten wird daraufhin als die zu dieser Abfahrtszeit gehörige Ankunftszeit in diesem Nachbarknoten abzüglich der Ankunftszeit in u festgesetzt. Dadurch entspricht das Gewicht w entweder der reinen Fahrzeit oder fallbedingt der Fahrzeit plus Wartezeit und beziehungsweise oder der benötigten Umsteigezeit.
Im letzten Schritt wird ein Update ausgeführt und die Variablen t_{alt} und $Mode_{alt}$ für einen späteren Schleifendurchlauf upgedated. Dabei entspricht $Mode_{alt}[u]$ wie gewohnt dem Ankunftsmodus in $u \in V$.

```
while Q nicht leer do
    u = Knoten aus vorderstem Paar in Q;
    if u = e then
        break;
    end
    Lösche vorderstes Paar aus Q;
    if u ist „betrachtet“ then
        continue;
    end
    for i = 1...deg(u) do
        v := Nachbar i;
        Mode_neu ist Mode von u wenn Nachbar ist v;
        if u ist s then
            Mode_alt(u) := Mode_neu;
        end
        if Mode_alt(u) = Mode_neu then
            while Abfahrtszeit von u nach v < t_alt(u) do
                nimm nächste Abfahrtszeit als t_ab;
            end
            w := zu t_ab gehörige Ankunftszeit minus t_alt;
        else
            suche die benötigte Umsteigedauer von u in Mode_alt(u) nach u in
            Mode_neu aus U;
            w_um := dieser Umsteigedauer;
            while Abfahrtszeit von u nach v minus w_um < t_alt(u) do
                nimm nächste Abfahrtszeit als t_ab;
            end
            w := zu t_ab gehörige Ankunftszeit minus t_alt(u);
        end
        if v noch nicht „betrachtet“ and D[u] + w < D[v] then
            D[v] = D[u] + w;
            t_alt(v) = Ankunftszeit in v;
            Mode_alt(v) = Mode_neu;
            Füge (v, D[v]) zu Q hinzu;
        end
    end
    Setze u als „betrachtet“;
end
```

Kapitel 6

Speed-up Techniken

Die Effizienz des Multimodalen Dijkstra Algorithmus ist im Allgemeinen stark von der Anzahl an Knoten und Kanten abhängig. Da der Graph schon für das in dieser Arbeit auf den innerstädtischen Bereich von München eingeschränkte Gebiet 1421 Knoten sowie 10669 Kanten besitzt, würde sich die Laufzeit signifikant verbessern, sofern eine Reduzierung des Graphen gelingt.
Um die Größe an Inputdaten zu verringern kann sowohl die Anzahl der Kanten als auch die der Knoten im Graphen reduziert werden.
Neben der Möglichkeit den Graphen zu verkleinern gibt es weitere Techniken, um die Laufzeit des Algorithmus zu verbessern (siehe beispielsweise [Paj09]). Eine davon ist die sogenannte bidirektionale Suche, mit der das Gebiet, in dem ein optimaler Pfad gesucht wird, eingeschränkt werden soll.

6.1 Bidirektionale Suche

Allgemeines Vorgehen

Die Idee der bidirektionalen Suche ist es sowohl die Suche nach einem optimalen Start(s)-Ziel(t)-Pfad im Graphen $G = (V, E)$ zu starten (Vorwärtssuche) als auch gleichzeitig eine t-s-Pfad-Suche im rückwärts gerichteten Graphen $\overleftarrow{G}$ anzustoßen (Rückwärtssuche).
Wird ein LS-Verfahren (siehe Kapitel 3) verwendet, so terminiert der Algorithmus zur bidirektionalen Suche genau dann, wenn die Vorwärts- oder Rückwärtssuche einen Knoten markiert, der bereits mit einem Label des jeweils anderen Suchverfahrens versehen ist (siehe Abbildung 6.1). Geschieht das an einem Knoten $u \in V$, so ergibt der s-u-Pfad, der durch die Vorwärtssuche gefunden wurde, zusammen mit dem u-t-Pfad der Rückwärtssuche den optimalen s-t-Pfad. Dabei ist der u-t-Pfad nichts anderes als der t-u-Pfad in $\overleftarrow{G}$ mit umgedrehten Kanten, die somit den ursprünglichen Kanten aus G wieder entsprechen.

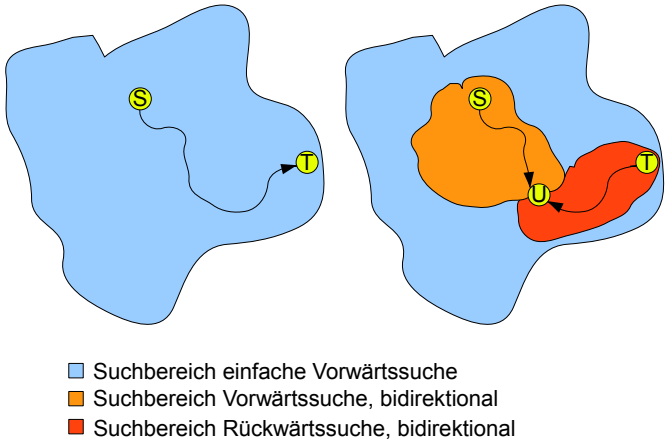


Abbildung 6.1: Einfache Vorwärtssuche und bidirektionale Suche

Wie in Abbildung 6.1 zu sehen, ist der Suchbereich bei einem solchen Vorgehen meist kleiner als der Suchbereich einer einfachen Vorwärtssuche, da im Allgemeinen nicht alle Knoten abgefragt werden bevor ein s-u-Pfad und ein t-u-Pfad (beziehungsweise ein u-t-Pfad) gefunden sind [BDW11].

Die Methode im multimodalen, zeitabhängigen Fall

Die Erweiterung auf den multimodalen Fall ist bei dieser Methode möglich. Die Betrachtung des zeitabhängigen Falls ist jedoch komplizierter.
Da der eingeführte, erweiterte Dijkstra Algorithmus im zeitabhängigen Fall bei gegebener Startzeit zur gewünschten Lösung führt, ist dieser auch bei der Vorwärtssuche uneingeschränkt anwendbar. Die allgemeine Methode scheitert demnach lediglich an der Rückwärtssuche.
Mit Hilfe der Rückwärtssuche einen kürzesten Pfad in einem zeitabhängigen Graphen zu finden gestaltet sich als schwieriges Unterfangen, da die Abfahrtszeit am Knoten $u \in V$ und damit im Normalfall auch die Wartezeit und Fahrtdauer von der Ankunftszeit im Knoten u abhängt. Diese ist bei der Rückwärtssuche an sich jedoch nicht bekannt.
Um dennoch einen sinnvollen Algorithmus zur Laufzeitverbesserung zu erhalten, soll die bidirektionale Suche so angepasst werden, dass trotz fehlender Informationen für die Rückwärtssuche das Gebiet, auf dem der optimale s-t-Pfad gefunden werden soll, eingeschränkt wird [DN12; Nan10]. Dafür bedarf es einiger Anpassungen. Im Folgenden sollen zunächst die einzelnen Schritte für eine zeitabhängige, bidirektionale Suche erläutert werden und anschließend die Korrektheit dieses Vorgehens gezeigt werden.

Sei G(V,E) ein zeitabhängiger Graph. Im ersten Schritt wird – ähnlich wie im zeitunabhängigen Fall – die Vorwärtssuche auf dem Graphen $G = (V, E)$ vom Startpunkt

$s \in V$ aus mit gegebener Startzeit T_{start} angestoßen. Gleichzeitig soll analog zum vorherigen Vorgehen die Rückwärtssuche gestartet werden, diesmal jedoch nicht auf dem Graphen $\overleftarrow{G}$, sondern auf einem Graphen $\overleftarrow{G}_g$. Dabei sei G_g äquivalent zu G, jedoch mit zeitunabhängigen Kantengewichten $abst_g$, für die gilt:

$$abst_g(u, v) \leq abst(u, v, t_{u_{\text{start}}})$$

Dabei seien $u,\ v \in V$ zwei benachbarte Knoten und $abst(u, v, t_{u_{\text{start}}})$ deren Abstand im Graphen G bei Startzeit $t_{u_{\text{start}}}$ in u. $\overleftarrow{G}_g$ ist analog zu $\overleftarrow{G}$ der Graph zu G_g mit rückwärtig gerichteten Kanten.
Beim Lauf der Rückwärtssuche auf $\overleftarrow{G}_g$ werden nun alle Knoten, die dabei von dem Dijkstra Algorithmus markiert werden, in einer Menge B gespeichert.
Dieser Schritt terminiert, wie im ersten Fall auch, sobald ein Knoten $w \in V$ von einer der beiden Suchen markiert wird, der bereits von der jeweils anderen Suchart gelabelt wurde. $P(w, g, t_{\text{start}})$ soll nun den Pfad bezeichnen, der aus dem zeitabhängigen s-w-Pfad der Vorwärtssuche und einem zeitabhängigen w-t-Pfad besteht. Letzterer enthält genau die Knoten und zugehörigen Kanten (jetzt mit den zeitabhängigen Gewichten aus G) des durch die Rückwärtssuche gefundenen, zeitunabhängigen t-w-Pfades in $\overleftarrow{G}_g$.
Da an dieser Stelle nicht sicher ist, ob der Pfad $P(w, g, t_{\text{start}})$ optimal ist, stellen dessen Kosten c eine obere Schranke für die Kosten c^* des optimalen s-t-Pfades mit Startzeit t_{start} dar.

Nun beginnt der zweite Schritt. In diesem lässt man sowohl die Vorwärtssuche als auch die Rückwärtssuche einfach weiterlaufen bis die Prioritätenliste der Rückwärtssuche nur noch Knoten enthält, deren Abstand auf $\overleftarrow{G}_g$ vom Endknoten t größer ist als die Schranke c oder bis die Vorwärtssuche t erreicht hat. Bricht Schritt zwei aufgrund der Rückwärtssuche ab, so werden die zusätzlich markierten Knoten der Rückwärtssuche zur Menge B hinzugefügt und Schritt drei gestartet.

Im letzten Schritt wird die Vorwärtssuche erneut fortgeführt, jedoch beschränkt sich der Suchbereich nun auf die Knotenmenge B.

Theorem. Die vorgestellte zeitabhängige, multimodale, bidirektionale Suche berechnet den kürzesten Pfad von einem Startknoten s zu einem Endknoten t in einem zeitabhängigen Graphen.

Beweis. Die Vorwärtssuche ist exakt dieselbe Methode wie im nicht-bidirektio-nalen Fall und findet damit einen optimalen Pfad.
Was nun gezeigt werden muss ist, dass die Einschränkung des Suchbereichs keinen Einfluss auf die Optimalität des gefundenen Pfades hat. Sei der in den ersten beiden Schritten gefundene Pfad der Vorwärtssuche ein s-v-Pfad mit dem Startknoten $s \in V$ und $v \in V$. Es ist nun zu zeigen, dass der im optimalen s-t-Pfad enthaltene Teilpfad von dem Knoten v zum Endknoten t keinen Knoten u enthält, der nicht in B liegt, also keinen Knoten u

mit $u \in V \backslash B$.
Angenommen es existiert ein solcher Knoten u mit $u \in V \backslash B$. Da in diesem Fall der Knoten u von dem in der Rückwärtssuche verwendeten Dijkstra Algorithmus nicht markiert wurde, muss gelten:

$$c \leq abst_g(u,t)$$

Dabei bezeichnet c die Kosten oder anders ausgedrückt die Länge des Pfades $P(w,g,t_{\text{start}})$ und $abst(u,t,t_{u_{\text{start}}})$ den (kleinsten) Abstand vom Knoten u zum Endknoten t in G mit Startzeit $t_{u_{\text{start}}}$ in u. $abst_g(u,t)$ sei der kleinste Abstand von u zu t in G_g. Sei nun $P^*(t_{\text{start}})$ der optimale s-t-Pfad und c^* dessen Länge. Damit muss

$$c^* \leq c \overset{Abbruch}{<} abst_g(u,t)$$

$$\leq abst(u,t,t_{u_{\text{start}}}) \leq abst(s,u,t_{\text{start}}) + abst(u,t,t_{\text{start}} + abst(u,t,t_{\text{start}})) = c^*$$

erfüllt sein muss. Wiederspruch $\lightning$ □

Die parallele Implementierung der beiden Vorgänge „Vorwärtssuche“ und „Rückwärtssuche“ geschieht aus Gründen der Zeitersparnis. Dennoch ist anzumerken, dass diese Methode zwar in schnellerer Zeit einen kürzesten s-t-Pfad finden kann, allerdings auch die Möglichkeit besteht, dass dieses Vorgehen zu einer Verschlechterung der Laufzeit führt.

6.2 Kontraktion

Wie zu Beginn dieses Kapitels erwähnt ist die Reduzierung von Knoten und Kanten (Kontraktion) ein effektives Mittel, um die Laufzeit des Multimodalen Dijkstra Algorithmus zu verkürzen. Dies kann dabei nicht nur durch eine Einschränkung des Suchgebiets über die bidirektionale Suche geschehen, sondern auch über das direkte Herausnehmen von Knoten und Kanten.
Die nachfolgend beschriebenen Verfahren sind angelehnt an [Paj09].

6.2.1 Entfernen von Knoten

Allgemeines Vorgehen

Die Grundidee bei der Reduzierung des Graphen G durch geeignetes Entfernen von Knoten ist es, den Graphen in zwei Subgraphen zu teilen, den Hauptgraphen (Core) G_{core} und die Komponenten (Component) G_{comp}, sodass $G = G_{core} \cup G_{\text{comp}}$ gilt. Dabei befinden sich zunächst alle Knoten im Hauptgraphen. Iterativ werden nun (meist vernachlässigbare) Knoten entfernt und in die Komponenten „gelegt“. Die „entfernten“ Knoten werden nicht komplett gelöscht, um somit nach Lösung des Kürzesten-Pfad-Problem den ursprünglichen Pfad schneller angeben zu können. Dauert die spätere Suche des

ursprünglichen Pfades nämlich zu lange, so könnte dieses Verfahren unter Umständen sogar zu einer Verschlechterung der Laufzeit führen.
Die an dem entfernten Knoten anliegenden Kanten werden durch neue Kanten ersetzt, die die Abstände der übrig gebliebenen Knoten erhalten. Seien also (u, v) und (v, w) zwei Kanten im Graphen G. Wird der Knoten v nun vom Hauptgraphen in die Komponenten versetzt, so werden diese Kanten ersetzt durch eine Kante $(u, w)'$, die den Wert des u-w-Pfades über den Knoten v trägt. Existiert bereits eine Kante (u, w), so wird keine neue Kante hinzugefügt und lediglich das Gewicht der vorhandenen Kante (u, w) auf den Wert $\min\{\mathit{Gewicht}\ (u, w),\ \mathit{Gewicht}\ (u, w)'\}$ gesetzt.
Die Regel, nach der die Knoten aus dem Hauptgraphen herausgenommen werden ist dabei anwendungsabhängig.

Die Methode im multimodalen, zeitabhängigen Fall

Das Vorgehen ist hier grundsätzlich dasselbe wie im oben beschriebenen allgemeinen Fall. Will man eine Regel für das Entfernen eines Knoten v aus dem Hauptgraphen festsetzen, so ist es sinnvoll, die Struktur des Graphen (Modi, Umsteigepunkte, etc.) nicht zu verändern. Eine denkbare und gut zu praktizierende Regel für den multimodalen Fall (zeitunabhängig oder zeitabhängig) ist, dass ein Knoten v nur dann aus dem Hauptgraphen herausgenommen wird, wenn gilt:

(i) Alle eingehenden und ausgehenden Kanten des Knotens v sind im selben Modus (diesen Modus bekommt auch die nach dem Löschen neu eingefügte Kante).

(ii) $\#\text{Kanten}_{\text{neu}} \leq K(\deg_{\text{in}}(v) + \deg_{\text{out}}(v))$, dabei ist K eine Konstante, die je nach gewünschtem Reduzierungsgrad gewählt werden kann (je höher, desto öfter wird die Reduzierungsmethode eingesetzt). Hier bezeichnet $\deg_{\text{in}}(v)$ die Anzahl der Nachbarn von v, von denen aus v über eine Kante erreichbar ist (eingehende Kante in v) und $\deg_{\text{out}}(v)$ steht für die Anzahl der Nachbarn von v, zu denen eine Kante von v aus führt. Bei der Anzahl von neuen Kanten $\#\text{Kanten}_{\text{neu}}$ ist zu beachten, dass nur Kanten gezählt werden, die tatsächlich neu eingefügt werden. Gewichtupdates werden nicht berücksichtigt, da hier an dieser Stelle lediglich verhindert werden soll, dass die Zahl der Kanten anwächst.

(iii) Sei die maximale Anzahl an, von einer in v eingehenden oder aus v herausgehenden Kante, „übersprungenen“ Knoten mit $\text{hop}(v)$ bezeichnet (siehe Abbildung 6.2). Dann soll $\text{hop}(v) \leq H$ gelten, wobei H je nach Absicht und Anwendung gewählt wird.

Die Knotenreduzierung terminiert, sobald kein Knoten mehr diese Bedingungen erfüllt.

Auch wenn das Vorgehen im zeitabhängigen Fall grundsätzlich dasselbe ist wie im zeitunabhängigen, muss doch beim Entfernen des Knoten v beachtet werden, dass

beispielsweise die Gewichte der Kanten (u, v) und (v, w) nun Funktionen der Form $f_{(u,v)}$ beziehungsweise $f_{(v,w)}$ sind. Entsprechend muss das Gewicht der Kante $(u, w)'$ beim Herausnehmen des Knotens v aus G_{core} definiert sein als

$$f_{(u,w)'} := f_{(u,v)} \oplus f_{(v,w)}.$$

Existiert bereits eine Kante (u, w) im Hauptgraphen, so wird analog zum allgemein beschriebenen Fall das Kantengewicht als

$$f_{(u,w)} := \min \left\{ f_{(u,w)'}, f_{(u,w)} \right\}$$

gesetzt und keine zusätzliche Kante von u nach w eingefügt. Die Methode lässt sich somit gut auf den multimodalen, zeitabhängigen Fall übertragen.
Die Kehrseite ist jedoch, dass sich die Berechnungen der neuen Kantengewichte ungünstig auf die Laufzeit auswirken können. Eine zusätzliche Auswirkung auf die Laufzeit kann die Reihenfolge, in der die Knoten aus dem Graphen gelöscht werden haben, da diese die Anzahl der löschbaren Kanten beeinflussen kann [Gei+08].

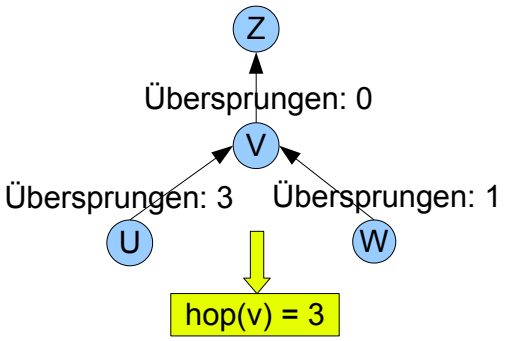


Abbildung 6.2: Berechnung von hop(v)

6.2.2 Entfernen von Kanten

Allgemeines Vorgehen

Da bei der vorhergehenden Methode oft unnötige Kanten eingeführt werden, werden diese mit der in diesem kurzen Abschnitt beschriebenen Methode unter Umständen wieder gelöscht. Es werden also erneut die Subgraphen G_{comp} und G_{core} zur Betrachtung herangezogen. Um zu entscheiden, ob eine Kante (u, w) aus dem Hauptgraphen G_{core} entfernt wird, wird für den Knoten u ein kürzester Pfad zu allen seinen Nachbarn in G_{core} berechnet. Die Kante (u, w) wird genau dann aus G_{core} entfernt, wenn die Länge

des kürzesten u-w-Pfades kleiner ist als das Gewicht dieser Kante. Für die Berechnung eines kürzesten Pfades von einem Startpunkt s zu einem Endpunkt t bleiben somit die Abstände unverändert.

Die Methode im multimodalen, zeitabhängigen Fall

Hier stellt die Erweiterung auf den multimodalen Fall, anders als beim Reduzieren von Knoten, keine Einschränkung dar und bedarf keiner weiteren Beachtung.
Die Zeitabhängigkeit bedeutet dafür eine umso größere Einschränkung auf diese Art der Graphenverkleinerung. Im zeitabhängigen Fall darf eine Kante nur dann gelöscht werden, wenn für jede Ankunftszeit am Knoten u ein kürzester u-w-Pfad gefunden werden kann, dessen Länge kleiner ist als das Gewicht der Kante (u, w).
Eine weitere Besonderheit ist, dass gleiche Knoten in unterschiedlichen Modi durch Umstiegskanten zwar verbunden sind, diese aber von der hier vorgestellten Methode nicht beachtet werden und somit auch nicht heraus gelöscht werden.

6.2.3 Lösungssuche mit Hilfe von Kontraktion im zeitabhängigen Fall

Die nun vorgestellte Lösungsmethode ist angelehnt an die in [DN12] aufgeführte Kontraktionsmethode und zeigt, wie mit Hilfe der Kontraktion ein kürzester s-t-Pfad gefunden werden kann.
Im Rahmen eines Preprocessings werden zunächst die Knoten und Kanten wie oben beschrieben aus dem Graphen herausgenommen und der Graph in Hauptgraph G_{core} und Komponenten G_{comp} aufgeteilt. Dabei können der Startknoten s und der Zielknoten t entweder in G_{core} oder G_{comp} liegen. Die Kontraktion ist somit vollzogen und die Suche nach einem kürzesten s-t-Pfades kann auf dem so aufgeteilten Graphen begonnen werden. Dafür wird in folgenden zwei Schritten vorgegangen:

(i) Im ersten Schritt wird eine bidirektionale Suche, wie in Abschnitt 5.3.1 beschrieben, mit Hilfe des Dijkstra Algorithmus auf dem zeitabhängigen Graphen G_{comp} gestartet. Startpunkt für die Vorwärtssuche ist hierbei wieder der Startknoten s, für die Rückwärtssuche der Zielknoten t.
Alle Knoten, die während der Vorwärtssuche markiert werden, werden in einer Menge F gespeichert, alle durch die Rückwärtssuche markierten in der Menge B. Dabei werden die Knoten aus F und B sowohl aus der Prioritätenliste der Vorwärtssuche als auch aus der der Rückwärtssuche herausgelöscht. Falls ein Knoten v aus G_{core} von einer der beiden Suchen markiert wird, wird die von v weiterführende Suche abgebrochen.
Befindet sich der Knoten s oder der Knoten t im Hauptgraphen G_{core} wird die jeweilige Suche (Vorwärts- oder Rückwärtssuche) sofort gestoppt. Andernfalls wird die bidirektionale Suche solange fortgeführt, bis entweder
   a) $F \cap B \cap G_{\text{comp}} \neq \emptyset$ gilt, oder

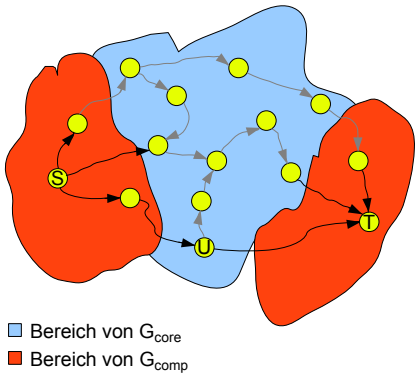


Abbildung 6.3: Abbruchkriterium 1

b) die Prioritätenlisten beider Suchen leer sind.

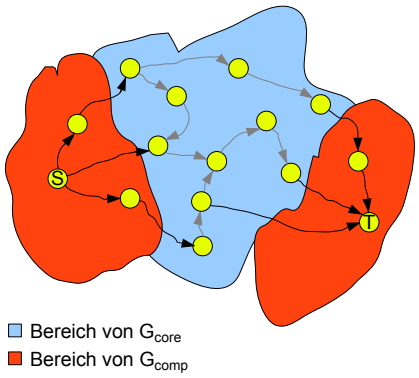


Abbildung 6.4: Abbruchkriterium 2

Falls der Abbruch der bidirektionalen Suche aufgrund von Punkt (i) geschieht, wird eine einfache Vorwärtssuche auf ganz G gestartet bis ein kürzester s-t-Pfad errechnet ist.
Falls weiterhin $F \cap B \cap G_{\text{comp}} = \emptyset$ gilt und der Abbruch aufgrund von Punkt (ii) geschieht, wird der zweite Schritt eingeleitet.

(ii) Im zweiten Schritt wird nun die bidirektionale Suche auf dem Hauptgraphen $G_{\text{core}} = (V_{\text{core}}, E_{\text{core}})$ gestartet (siehe Abbildung 6.5). Dabei wird die Vorwärtssuche in jedem Punkt aus $F \cap V_{\text{core}}$ angestoßen, die Rückwärtssuche in jedem Punkt aus $B \cap V_{core}$, es handelt sich demnach um eine „Many-to-Many“-Kürzester-Pfad-Suche.

Wie in Abschnitt 5.3.1 ist der letzte Schritt einer bidirektionalen Suche das nochmalige Anstoßen der Vorwärtssuche. Dieser Schritt wird auch an dieser Stelle wieder ausgeführt. Das Suchgebiet ist dabei ähnlich zu dem in Abschnitt 5.3.1. In diesem Fall operiert die Vorwärtssuche auf dem Suchgebiet $G_{\text{core}} \cup B$. Dabei soll B nur über einen Knoten aus $B \cap V_{\text{core}}$ betreten werden können, was das Suchgebiet weiter verfeinert.
Die Gewichte der Knoten aus $F \cap V_{\text{core}}$ und B werden dabei aus Schritt Eins übernommen. Der zweite Schritt endet mit der Ausgabe eines s-t-Pfades.

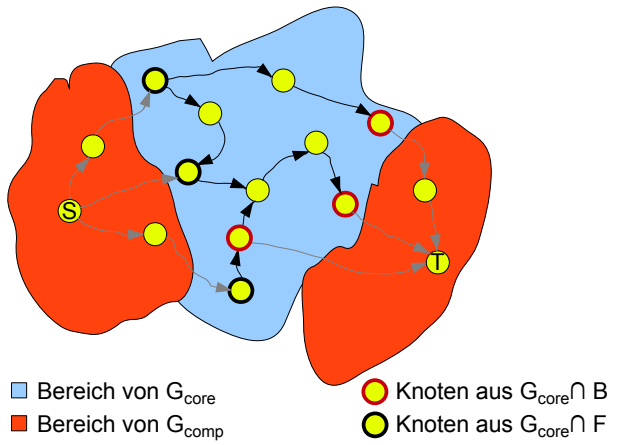


Abbildung 6.5: Der zweite Schritt der Lösungssuche

Die Effizienz dieses Verfahrens ist stark von der Aufteilung des Graphen in G_{core} und G_{comp} abhängig.

Theorem. Die vorgestellte Methode berechnet den kürzesten Pfad von einem Startknoten s zu einem Endknoten t in einem zeitabhängigen Graphen. □

Beweis. Angenommen Schritt Eins bricht ab mit dem Ergebnis $F \cap B \cap G_{\text{comp}} \neq \emptyset$. Da daraus folgt, dass der s-t-Pfad mit dem Multimodalen (zeitabhängigen) Dijkstra Algorithmus gesucht wird, ist die Methode in diesem Fall korrekt. Dass in diesem Fall das Vorgehen aus Schritt Zwei nicht angewendet werden kann, zeigt sich zum Beispiel in Abbildung 6.6.

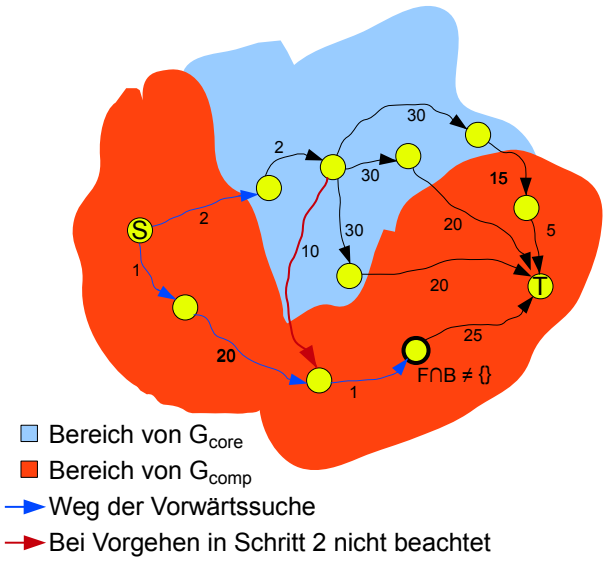


Abbildung 6.6: Der zweite Schritt der Lösungssuche

Bricht Schritt Eins ab nachdem die Prioritätenliste der Vorwärts- sowie der Rückwärtssuche leer sind und ist zu diesem Zeitpunkt $F \cap B \cap G_{\text{comp}}$ immer noch eine leere Menge, so ist der kürzeste s-t-Pfad von der Form $P = (s \to ... \to u \to ... \to w \to ... \to t)$. u sei dabei der Eintrittsknoten von G_{comp} in G_{core} und w der Austrittsknoten aus G_{core}.

1.Fall: $u \neq w$

Der s-u-Teilpfad von P liegt vollständig in G_{comp}. Dessen Länge wird durch die Vorwärtssuche bereits im ersten Schritt korrekt berechnet. Innerhalb des Hauptgraphen G_{core} werden die kürzesten Pfade inklusive des kürzesten u-w-Pfades mit Hilfe der bidirektionalen Methode aus Abschnitt 5.3.1 berechnet. Dieses Verfahren ist wie gezeigt wurde ebenfalls korrekt. Da der w-t-Teilpfad von P komplett in G_{comp} liegt und auch keine Verbindung zwischen Knoten aus der Menge F und dem Knoten t gibt, wird jeder der Knoten dieses Pfades von der Rückwärtssuche im ersten Schritt markiert und in B

gespeichert. Da die Vorwärtssuche mit Start in u die Knoten aus $V_{core} \cup T$ abläuft ist aufgrund der Korrektheit des Multimodalen Dijkstra Algorithmus auch der berechnete w-t-Pfad und damit der gesamte Pfad P optimal.

2.Fall: $u = w$

analog. □

Kapitel 7

Ergebnisse und Auswertung

In diesem Kapitel werden sowohl der multimodale Dijkstra Algorithmus – implementiert in C++ – als auch ein in Xpress (Mosel) implementierter Lösungsalgorithmus auf unterschiedliche Probleminstanzen bezüglich Knoten- und Kantenmenge sowie Anzahl der Modi getestet. Dabei soll für die Testläufe der innerstädtische öffentliche Personennahverkehr Münchens betrachtet werden. Die Zeitpläne werden im Folgenden als periodisch angenommen. Als Referenzfahrpläne wurden die Daten für die Fahrten von Montag bis Donnerstag beziehungsweise Montag bis Freitag (je nach Verkehrslinie) herangezogen. Zudem werden gesonderte Fahrzeiten während der Schulferien außer Acht gelassen.

7.1 Beispielläufe des implementierten Programms

In diesem Abschnitt soll das Zeitintervall I (siehe Kapitel 2), das für die Lösung und die einzelnen Fahrzeiten durchsucht wird, zwei Tage umfassen, den Tag der Startzeit und den darauf folgenden. Dies ist ausreichend, da sich die Anwendung hier lediglich auf den innerstädtischen Verkehr der Stadt München beschränkt.

Erstes Szenario

Im ersten Szenario soll der gesamte innerstädtische Verkehr von U-Bahn, S-Bahn, Bus und Tram sowie Nachtlinien betrachtet werden. Damit ergibt sich eine Größenordnung der Daten von:

- 1421 Knoten
- 10669 Kanten
- 109 Modi und
- 3298 Umsteigemöglichkeiten (einfach gezählt, insgesamt: 6596)

In nachfolgender Tabelle 7.1 sind die Ergebnisse der Testläufe abgebildet. Hierbei ist die Rechenzeit die Laufzeit des Algorithmus, die nach Einlesen der Daten für das finden einer optimalen Lösung benötigt wird. Die Dauer ist zur Verdeutlichung in Millisekunden angegeben. Da die Laufzeiten gewissen Schwankungen unterliegen, wurden die Laufzeiten von je 50 Durchläufen gemittelt.

Start	Ziel	Startzeit	#Modi	Fahrzeit	Rechenzeit
Chin. Turm	Odeonsplatz	16:20 Uhr	2	15 min	4
Theresienstraße	Nawiaskystraße	09:05 Uhr	4	31 min	168
Odeonsplatz	Garching	10:00 Uhr	1	23 min	195
Garching	Fraunhoferstrasse	12:00 Uhr	3	43 min	57
Theresienstraße	Chin. Turm	01:00 Uhr	4	63 min	170
Theresienstraße	Nawiaskystraße	01:00 Uhr	3	71 min	189
Chin. Turm	Theresienstraße	01:50 Uhr	2	156 min	196

Tabelle 7.1: Ausgabe der Testläufe bei Szenario 1

Möchte man also beispielsweise von der Haltestelle „Chinesischer Turm“ zur Haltestelle „Odeonsplatz“ fahren, so würde man dem Programm zufolge bei Startzeit 16:20 Uhr 15 Minuten benötigen. Dies entspricht im Übrigen exakt der Zeit, die vom Programm des Münchner Verkehrs- und Tarifverbundes (MVV) ausgegeben wird (siehe [Mün]). Lediglich die Wartezeit zu Beginn wird bei der Ausgabe auf der genannten Internetseite vernachlässigt.
Betrachtet man die Ausgaben aus Tabelle 7.1, so lässt diese vermuten, dass die Berechnungszeit hauptsächlich von der jeweiligen Fahrtzeit abhängt. Dies liegt vor allem daran, dass – umso länger die Fahrt zum Ziel dauert – umso mehr Knoten vor dem eigentlichen Ziel von dem Dijkstra Algorithmus durchlaufen werden und die zugehörigen Abstände zum Startknoten aktualisiert werden. Dabei spielt auch die Anzahl der Nachbarn vom Startknoten und den übrigen vor dem Ziel durchlaufenen Knoten eine Rolle. Die Aussage ist also nicht allgemein gültig, aber für gleichbleibenden Startpunkt und Startzeitpunkt wahr.

Zweites Szenario

Um die Abhängigkeit der Rechenzeit des Algorithmus von der Menge der Inputdaten und der Beschaffenheit des zugrunde liegenden Graphen näher zu betrachten, wurde ein zweites Szenario erstellt. Dieses betrachtet abermals den innerstädtischen Verkehr von U-Bahn, S-Bahn, Bus und Tram. Nachtlinien wurden nur für das Straßenbahnnetz mit einbezogen, Nachtbuslinien sollen in diesem Szenario allerdings nicht existieren. Die Buslinien 50, 51 sowie 53 seien zusätzlich ausgenommen, da sich dadurch zusätzlich die Anzahl der Knoten erheblich verringert. Damit werden:

- 1372 Knoten
- 9276 Kanten
- 93 Modi und
- 1957 Umsteigemöglichkeiten (einfach gezählt, insgesamt: 3914)

für das zweite Szenario betrachtet.
Um die Szenarien besser vergleichen zu können, werden in der nachfolgenden Tabelle dieselben Eingabedaten (Startpunkt, Ziel und Startzeit) für die Testläufe gewählt wie in Szenario 1 (siehe Tabelle 7.1).

Start	Ziel	Startzeit	#Modi	Fahrzeit	Rechenzeit
Chin. Turm	Odeonsplatz	16:20 Uhr	2	15 min	4
Theresienstraße	Nawiaskystraße	09:05 Uhr	4	31 min	138
Odeonsplatz	Garching	10:00 Uhr	1	23 min	131
Garching	Fraunhoferstrasse	12:00 Uhr	2	43 min	35
Theresienstraße	Chin. Turm	01:00 Uhr	3	248 min	165
Theresienstraße	Nawiaskystraße	01:00 Uhr	4	252 min	187
Chin. Turm	Theresienstraße	01:50 Uhr	4	227 min	95

Tabelle 7.2: Ausgabe der Testläufe bei Szenario 2

Vergleicht man die oben aufgeführte Tabelle 7.2 mit der des ersten Szenarios, so ist der größte Unterschied in der Berechnungszeit der letzten Fahrt zu erkennen. Für die Optimierung des Fahrtweges von der Haltestelle „Chinesischer Turm“ zur Haltestelle „Theresienstrasse“ werden nun lediglich etwa 48% der Rechenzeit benötigt. Die Zahl der Kantengewichts-Updates reduziert sich dabei um etwa 63% im Vergleich zu Szenario 1. Bei den unteren drei Fahrten ändert sich allerdings auch die Fahrzeit beträchtlich. Während die Fahrt von der Haltestelle „Theresienstraße“ zur Station „Chinesichen Turm“ in Szenario 1 noch eine Dauer von 63 Minuten hatte, wäre das Ergebnis in Szenario 2 nun eine Fahrzeit von 248 Minuten, was einem Unterschied von über 3 Stunden entspricht.

7.2 Lösung in Xpress-IVE

Wie zu Anfang des Kapitels erwähnt wurde im Rahmen dieser Arbeit eine weitere Implementierung zur Lösung des EAP durchgeführt. Um eine weitere Herangehensweise zu testen, wurde die in Kapitel 3 aufgeführte Problemformulierung in Mosel implementiert und eine Lösung mithilfe des „Xpress-Optimizer“ gesucht. Dabei war nicht nur die Programmierung von Nöten, sondern auch eine Apassung der Daten. Während der in C++ programmierte Algorithmus noch damit umgehen konnte, dass dem gleichen Modus mehrere Fahrpläne zugeordnet sind, war das in diesem Fall nicht möglich. Daher wurden gegebenfalls die Modi noch einmal unterteilt und in Folge dessen auch die Daten der Umsteigemöglichkeiten angepasst.
Wie bei der Implementierung des multimodalen Dijkstra Algorithmus, wurden auch hier unterschiedliche Szenarien getestet. Für die Abdeckung desselben Netzes wie im obigen Szenario 1 gelten die folgenden Inputgrößen:

- 1421 Knoten

- 10669 Kanten
- 651 Modi und
- 184985 Umsteigemöglichkeiten (einfach gezählt, insgesamt: 369970)

Damit erhöht sich die Anzahl der Modi auf rund das sechsfache, die Anzahl der Inputdaten bezüglich der Umsteigemöglichkeiten sogar auf das 56-fache. Die unten aufgeführte Tabelle zeigt die Auswertungsergebnisse der Implementierung hinsichtlich der für die Lösungsfindung benötigten Berechnungszeit. Es sei allerdings vorab erwähnt, dass die Lösungssuche – trotz dynamischer Arrays – in der hier behandelten Anwendung auf den innerstädtischen, öffentlichen Verkehr in München um ein vielfaches langsamer war als die Lösungssuche des C++ Programmes.
Für die folgenden Aussagen wurde jeweils eine Startzeit von 12:00 Uhr gewählt.
Nimmt man aus der Datenmenge lediglich 100 Modi heraus, liefert das Programm bei einfachen Fahrten relativ gute Laufzeitergebnisse. So benötigt das Programm bei einer Fahrt von der Haltestelle „Giselastrasse“ zur Station „Chinesischer Turm“ eine Rechenzeit von 2,6 Sekunden zur Bestimmung der Lösung. Dabei wird für die Fahrt lediglich ein Modus verwendet und der Pfad (inklusive Start- und Endstation) umfasst drei Haltestellen.
Für die etwas komplexere Strecke von der Haltestelle „Prinzregentenplatz“ nach „Kustermannpark“, für die ein Moduswechsel vollzogen werden muss, benötigt das Programm mit der oben angegeben Datenmenge von 100 Modi etwa 4,6 Sekunden.
Erweitert man nun das Suchgebiet auf 120 Modi, so erfolgt die Berechnung der ersten Strecke, von der Station „Giselastrasse“ zum Stop „Chinesischer Turm“, in 4,4 Sekunden. Jedoch schon für die Berechnung der Fahrt vom „Prinzregentenplatz“ zur Haltestelle „Kustermannpark“ benötigt das Programm mehrere Stunden. Lässt man die erste Fahrt unter Berücksichtigung aller 651 Modi bestimmen, so dauert auch dies mehrere Stunden.
Das Problem bei der Lösungsbestimmung liegt in den Symmetrie des Problems. In Abbildung 7.1 ist eine Teilausgabe der LP-Lösung des relaxierten Problems mit Startpunkt „Potsdamer Straße“ und Ziel „Arabellapark“ dargestellt (bei Betrachtung der wesentlichen Kanten). Als Startzeit wurde dabei 12:00 Uhr (720) gewählt. Die angegebenen Zeiten in Abbildung 7.1 beziehen sich auf die zur jeweiligen Kante gehörigen Startzeiten. Modi sind durch die Verwendung unterschiedlicher Farben gekennzeichnet. Zusätzlich ist anzumerken, dass mit einer blauen Box gekennzeichnete Stationen jeweils zusammengehören (d.h. diese stellen die gleiche Station in unterschiedlichen Modi dar). Desweiteren wurde eine zusätzliche Nebenbedingung eingefügt, nämlich

$$\text{Startzeit} \leq \text{Ankunftszeit}.$$

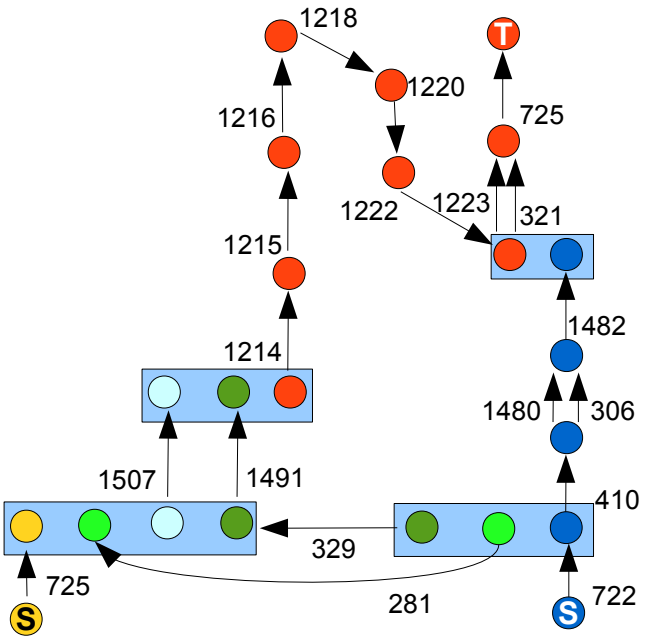


Abbildung 7.1: LP-Lösung des relaxierten Problems

In Abbildung 7.1 ist gut zu sehen, dass durch die Kombination von unterschiedlichen Kanten dem Algorithmus die Möglichkeit gegeben ist bei dem relaxierten Problem sehr späte mit sehr frühen Abfahrtszeiten zu kombinieren und so die Nebenbedingung zu erfüllen, die besagt, dass die (Summe der) Abfahrtszeiten an einer Station immer höher sein muss als Ankunftszeit an der Station, und gleichzeitig die Ankunftszeit unrealistisch niedrig zu halten. Je mehr Modi beziehungsweise Fahrten zur Auswahl stehen, desto mehr solche Kombinationen gibt es, die zu einem besseren Ergebnis als die Lösung des nicht-relaxierten Problems führen.

Kapitel 8

Die benutzerfreundliche Oberfläche

Im Rahmen dieser Arbeit wurde auch eine Web-Anwendung programmiert, die es dem Benutzer ermöglicht, das Programm auf einfache Art und Weise zu testen und sich einen optimalen Weg von seinem gewünschten Startpunkt zu einem gewählten Ziel für jede beliebige Startzeit ausgeben zu lassen.
Das Programm wurde für die Stadt München und das Verkehrsnetz der Münchner Verkehrsgesellschaft zuzüglich der S-Bahn Strecken implementiert.
Im Folgenden wird nun zuerst die Hauptoberfläche, das für den Benutzer sichtbare „Frontend“ beschrieben und illustriert. Anschließend soll die Verknüpfung mit dem in C++ implementierten Algorithmus genauer erläutert werden.

8.1 Das Frontend

Die Hauptoberfläche (Frontend) ist das Werkzeug, welches dem Benutzer später zur Verwendung zur Verfügung stehen wird. Daher zielt das Layout in diesem Fall darauf ab, möglichst übersichtlich und einfach in der Handhabung zu sein.
Dem Anwender steht hierbei ein Eingabefeld (siehe Abbildung 8.1) zur Verfügung, bei dem er die gewünschte Startzeit, die Startstation und das Ziel auswählen kann.

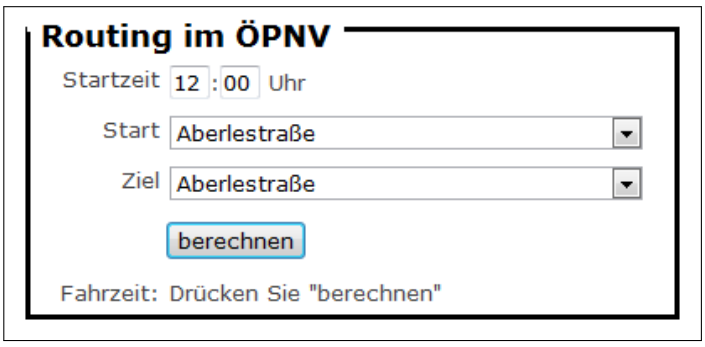


Abbildung 8.1: Eingabefeld

Nach Eingabe der Fahrdetails muss der Schalter „berechnen“ angeklickt werden. Dieser Vorgang stößt den Algorithmus im Backend an und soll im nachfolgenden Abschnitt genauer betrachtet werden.
Nachdem der Algorithmus im Hintergrund den optimalen Pfad errechnet hat, wird dieser Pfad – inklusive der benötigten Fahrzeit – zurückgegeben. Die Fahrzeit wird für den

Nutzer in dem dafür vorgesehenen Bereich des Eingabefeldes sichtbar (siehe Abbildung 8.2).

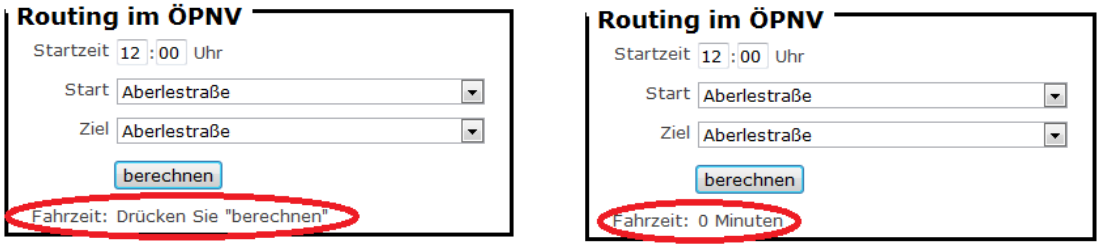


Abbildung 8.2: Angabe der benötigten Fahrzeit

Die Fahrzeit beträgt in diesem Beispiel „0“, da für die Fahrt die gleiche Station („Aberlestraße“) für An- und Abfahrt gewählt wurde.
Da der Nutzer nun auch wissen muss, was die bestmögliche Route ist und an welchen Stationen er umsteigen muss, wird ihm der Weg auf zwei verschiedene Arten ausgegeben. Zunächst wird der Weg durch die Rückgabe der jeweiligen Koordinaten in der nebenstehenden Karte eingezeichnet. In Abbildung 8.3 ist hierfür ein Beispiel zu sehen, bei dem als Startpunkt die Haltestelle „Luisenstraße“ eingegeben und die Station „Münchner Freiheit“ als Ziel gewählt wurde. Die Startzeit in diesem Beispiel ist 12.00 Uhr am Mittag.

Abbildung 8.3: Karte mit eingezeichnetem Weg von der Haltestelle „Luisenstraße“ zur Haltestelle „Münchner Freiheit“; Quelle: OSM (Open Street Maps)

Die Karte und die Route werden dabei mit Hilfe einer JavaScript-Datei gestaltet. Diese wird in den HTML-Code eingebunden und durch einen PHP-Aufruf eingeblendet.
Da aus der Karte nicht genau ersichtlich ist, welche Linien zu wählen sind und an welchen Stationen der Passagier umsteigen muss, ist der Weg noch auf eine zweite Weise illustriert: Mit Hilfe einer genauen Pfadbeschreibung inklusive kenntlich gemachter Umsteigepunkte. Für die folgende Beispielabbildung 8.4 wurden dabei die gleichen Daten wie für Abbildung 8.3 verwendet.

Abbildung 8.4: Ausgabe Pfadbeschreibung

8.2 Das Backend

Nachdem der Benutzer die Daten in das Frontend eingegeben hat, muss der Optimierungsalgorithmus angestoßen werden, um einen optimalen Pfad auf dem Frontend ausgeben zu können. Dieser Optimierungsalgorithmus bildet das Kernstück des Backend.
Der für diese Arbeit implementierte Algorithmus kann die Namen der Stationen nicht zuordnen, da er auf der Basis natürlicher Zahlen operiert. Somit müssen diese Namen zunächst umgewandelt werden. Dies geschieht durch das Aufrufen einer Datei, in der jedem Stationsnamen eine für das Programm benötigte ganzzahlige (natürliche) Identifizierungsnummer zugeordnet ist. Wird dann die Drop-down-Liste für das Frontend erzeugt, so ist bei der Auswahl von Start- und Zielknoten für den Benutzer zwar der Name der Station sichtbar, an das Programm im Backend wird allerdings nur die zugehörige

Identifizierungsnummer weitergegeben.

```
//Liste von Bahnhoefen/IDs aus Datei einlesen
if($file = fopen("inputdatei.csv", 'r')
or die("can't open file")){
  $stations = array();
  $i = 0;
  while($csv_line = fgetcsv($file,1164,';')) {
    $stations[$i] = $csv_line;
    $i++;
  }
  // Dropdown-Optionen
  $station_opts = "";
  foreach($stations as $n => $data){
    $n = count($data);
    if($n < 2) {
      die("data error");
    }
    $station_name = $data[1];
    $station_ID = $data[0];
    $station_options .= '<option value="'
    .$station_ID.'">'.$station_name.'</option>';
  }
}
```

Dabei ist „inputdatei.csv“ die Referenzdatei aus der die genaue Zuordnung von Identifizierungsnummer und Namen der Station ausgelesen werden kann. Die zugewiesene natürliche Zahl wird dann an den Algorithmus übergeben. Hierfür müssen die eingegebenen Daten in einer Datei gespeichert werden. Dies geschieht mit dem PHP-Aufruf

```
$myFile = "start.txt";
$fileHandle = fopen($myFile, 'w');
//eingegebene Startzeit in Minuten umrechnen
$myTime = $_POST['timeHours']*60 + $_POST['timeMinutes'];
//Eingaben des Benutzers in start.txt schreiben
$data = $_POST['Startpunkt']." ".$_POST['Zielstation']." "
.$myTime;
fwrite($fileHandle, $data);
fclose($fileHandle);
```

Darauf folgend wird der in C++ implementierte Algorithmus angestoßen. Dieser berechnet unter Verwendung der Datei „start.txt“ einen kürzesten Pfad und übergibt ihn an das HTML-Programm zur Weiterverwendung auf dem Frontend.
Damit ist die Webapplikation und ihre Funktionalität vollständig dargestellt.

8.3 Anpassungen und Zusatzoptionen

Das für diese Arbeit implementierte Programm ist ein Entwurf, der viele Erweiterungen und Verfeinerungen zulässt. Derzeit werden dabei zum Beispiel Verbindungswege zwischen zwei Stationen lediglich durch das Einzeichnen einer einfachen Strecke zwischen den jeweiligen Punkten angezeigt. Eine Erweiterungsmöglichkeit besteht darin, stattdessen den tatsächlichen Verlauf der Ubahntunnel, Tramgleise und Buslinien einzuzeichnen. Zudem könnte man für An- und Abfahrtsstationen separate Knoten (Point of interest – POI) anlegen, falls diese voneinander abweichen.
Ein weiterer Zusatz wäre eine Erweiterung der Wahlmöglichkeiten für den Benutzer, wie beispielsweise die Wahl der Gehgeschwindigkeit beim Umstieg oder die Möglichkeit die maximale Zahl der Umstiege festzusetzen.
Eine einfach zu kodierende Ergänzung wäre die Auswahl eines festen Zwischenstopps. Hierfür muss das Backendprogramm lediglich zweimal angesprochen werden, dabei könnte zum Beispiel auch der Zeitpunkt der Weiterfahrt gewählt werden.
Als letzter Punkt sei die Möglichkeit erwähnt, einzelne Verkehrsmittel auszuschließen.
Die Implementierung dieser Punkte ist ohne Einschränkung möglich, wobei die meisten dieser Änderungen sowohl Frontend als auch Backend betreffen würden.

Kapitel 9

Ausblick auf mögliche Modellerweiterungen

Der Fokus dieser Arbeit lag darauf, die Fahrzeit von einem Ort zu einem anderen, bei gegebener Startzeit, in einem multimodalen Graphen zu minimieren. Zur Lösung des Problems wurde ein multimodaler Dijkstra Algorithmus implementiert und ein Vergleichsprogramm – basierend auf der in Kapitel 3 neu aufgestellten ILP-Formulierung – in Mosel geschrieben.
Aufgrund der enormen Datenmenge wurden jedoch einige Einschränkungen vorgenommen. So blieben Rush-Hours etc., die nicht in den Fahrplänen berücksichtigt sind unbeachtet. Um eine noch realistischere Implementierung zu gewährleisten, könnten Stoßzeiten beispielsweise bei den Umsteigezeiten berücksichtigt werden.
Sonderfahrpläne während der Schulferien und Wochenenden wurden für diese Arbeit außer Acht gelassen und müssten für eine reale Abbildung ergänzt werden. Auch eine Berücksichtigung längerer Umsteigezeiten für Menschen mit Gehbehinderung und die eingeschränkte Nutzbarkeit von Verkehrsmitteln für Rollstuhlfahrer sollte in einem vollständigen Modell berücksichtigt werden. So sind beispielsweise nicht alle Busse in München barrierefrei.
Doch nicht nur Erweiterungen in Bezug auf eine realistischere Abbildung sind denkbar, auch weitere Optionen für den Nutzer des Programms beziehungsweise der Web-Applikation sind möglich. So kann zum Beispiel die Wahl mehrerer Zwischenstopps mit aufgenommen werden.
Eine weitreichendere Anpassung wäre es, dem Anwender die Möglichkeit zu bieten, einen beliebigen Start- und Zielort angeben zu können, der nicht zwangsläufig eine Haltestelle ist. Damit verbunden könnte eine Umkreissuche angestoßen werden, die den kürzesten Pfad sucht, der von einer beliebigen, in einem gewissen Umkreis vom Startpunkt befindlichen Haltestelle ausgehen kann.
Zusätzlich kann man die Fußwege zwischen den Stationen optimieren und die erhaltenen Wege als zusätzlichen Modus mit aufnehmen.
Rein implementierungstechnisch wäre auch die Berücksichtigung von Taktungen interessant. Dies wurde jedoch aufgrund der unregelmäßigen Taktungen im Münchner Personennahverkehr in dieser Arbeit vernachlässigt, würde aber unter anderen Gegebenheiten Sinn ergeben.

Neben den oben genannten Erweiterungen der hier behandelten Problemstellung, der Suche nach einem kürzesten Pfad, gibt es weitere Möglichkeiten im Anwendungsgebiet „öffentlicher Verkehr“.
So kann es beispielsweise von Interesse sein die Kosten für die benötigten Fahrkarten

so gering wie möglich zu halten, sprich die Kosten zu minimieren. Da dies bei Vernachlässigung der Fahrzeit in den meisten Fällen zu sehr vielen Lösungen führt, erscheint es sinnvoller entweder unter den pareto-optimalen, günstigsten Verbindungen diejenige mit der geringsten Fahrzeit zu wählen (zum Beispiel durch Einführen eines sehr kleinen fahrzeitabhängigen Straf-Gewichts) oder aber weiterhin nach einem kürzesten Pfad zu suchen, jedoch eine Kostenschranke für die benötigte Fahrkarte einzuführen.
Eine weitere Möglichkeit ist es die Anzahl der Umstiege zu minimieren. Auch hier empfiehlt es sich, die Fahrzeit nicht außer Acht zu lassen.
Steht im multimodalen Modell ein Auto zu Verfügung, so ist ein denkbares Ziel die Minimierung der Länge des Fahrtweges, um Benzinkosten zu sparen.
Einen weiteren Bereich bilden die multikriteriellen Ansätze (Multikriterielles Multimodales Kürzeste Wege Problem - MMSPP), bei denen beispielsweise nach Anzahl der Umstiege, Dauer der Fahrt und Reisegesamtkosten optimiert wird. Für einen tieferen Einstieg in die Verwendung von multikriteriellen Ansätzen auf den öffentlichen Personennahverkehr wird auf die Literatur [KK09; MH+04; HF] verwiesen.

Abbildungsverzeichnis

Alle Abbildungen in diesem Dokument stammen vom Autor selbst, sofern nicht weiter angegeben.

Tabellenverzeichnis

Liste der Algorithmen

Literatur

[Aan] *http://muenchner-forum.squarespace.com/attraktiver-nahverkehr-aan/.*

[AS05] S. H. Alexander Schiffel. *Fibonacci-Heaps und deren Anwendung.* Techn. Ber. RWTH Aachen University, 2005.

[BBM06] M. Bielli, A. Boulmakoul und H. Mouncif. „Object modeling and path computation for multimodal travel systems". In: *European Journal of Operational Research* 175.3 (2006), S. 1705–1730.

[BDW11] R. Bauer, D. Delling und D. Wagner. „Experimental study of speed up techniques for timetable information systems". In: *Networks* 57.1 (2011), S. 38–52.

[Bel58] R. Bellman. „On a Routing Problem". In: *Quarterly of Applied Mathematics* 16.1 (1958), S. 87–90.

[BJ04] G. S. Brodal und R. Jacob. „Time-dependent Networks as Models to Achieve Fast Exact Time-table Queries". In: *Electronic Notes in Theoretical Computer Science* 92 (2004), S. 3–15.

[CH66] K. Cooke und E. Halse. „The Shortest Route Through a Network with Time-Dependent Intermodal Transit Times". In: *Journal of Mathematical Analysis and Applications* 14(3) (1966), S. 493–498.

[Dij59] E. W. Dijkstra. „A note on two problems in connexion with graphs". In: *Numerische Mathematik* 1 (1959), S. 269–271.

[DN12] D. Delling und G. Nannicini. „Core Routing on Dynamic Time-Dependent Road Networks". In: *INFORMS Journal on Computing* 24.2 (2012), S. 187–201.

[DYQ08] B. Ding, J. X. Yu und L. Qin. „Finding time-dependent shortest paths over large graphs". In: *EDBT*. Hrsg. von A. Kemper, P. Valduriez, N. Mouaddib, J. Teubner, M. Bouzeghoub, V. Markl, L. Amsaleg und I. Manolescu. Bd. 261. ACM International Conference Proceeding Series. ACM, 2008, S. 205–216. ISBN: 978-1-59593-926-5.

[FT84] M. L. Fredman und R. E. Tarjan. *Fibonacci Heaps And Their Uses In Improved Network Optimization Algorithms.* 1984.

[Gei+08] R. Geisberger, P. Sanders, D. Schultes und D. Delling. „Contraction hierarchies: faster and simpler hierarchical routing in road networks". In: *Proceedings of the 7th international conference on Experimental algorithms.* WEA'08. Springer-Verlag, 2008, S. 319–333. ISBN: 3-540-68548-0, 978-3-540-68548-7.

[Gei11] R. Geisberger. „Advanced Route Planning in Transportation Networks“. Diss. Karlsruher Instituts für Technologie, 2011.

[HF] Y. Haicong und L. Feng. „A Multi-Modal Route Planning Approach with an Improved Genetic Algorithm“. In: *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* Vol. 38, Part II ().

[KK09] N. A. Konstantinos und G. Z. Konstantinos. „Solving the multi-criteria time-dependent routing and scheduling problem in a multimodal fixed scheduled network“. In: *European Journal of Operational Research* 192.1 (2009), S. 18–28.

[KS93] D. E. Kaufman und R. L. Smith. „Fastest Paths in Time-Dependent Networks for Intelligent Vehicle-Highway Systems Application“. In: *I V H S Journal: Technology, Planning, and Operations* 1:1 (1993), S. 1–11.

[LL09] L. Lu und M. Liqiu. „Algorithms of Multi-Modal Route Planning Based on the Concept of Switch Point“. In: *Photogrammetrie - Fernerkundung - Geoinformation (PFG)* 5.5/2009 (2009), S. 431–444.

[LS01] A. Lozano und G. Storchi. „Shortest viable path algorithm in multimodal networks“. In: *Transportation Research Part A: Policy and Practice* 35.3 (2001), S. 225–241.

[MH+04] M. Müller-Hannemann, F. Schulz, D. Wagner und C. D. Zaroliagis. „Timetable Information: Models and Algorithms“. In: *ATMOS*. Hrsg. von F. Geraets, L. G. Kroon, A. Schöbel, D. Wagner und C. D. Zaroliagis. Bd. 4359. Lecture Notes in Computer Science. Springer, 2004, S. 67–90. ISBN: 978-3-540-74245-6.

[MHM00] E. Miller-Hooks und H. Mahmassani. „Least expected time paths in stochastic, time-varying transportation networks“. In: *Transportation Science* 34(2) (2000), S. 198–215.

[Mün] M. München. *http://www.mvv-muenchen.de/de/fahrplanauskunft/fahrplan-buch/index.html*.

[Nan10] G. Nannicini. „Point-to-point shortest paths on dynamic time-dependent road networks.“ In: *4OR* 8.3 (2010), S. 327–330.

[Paj09] T. Pajor. „Multi-Modal Route Planning“. Magisterarb. Universität Karlsruhe (TH), 2009.

[Pat+03] P. Pattanamekar, D. Park, L. Rilett, J. Lee und C. Lee. „Dynamic and stochastic shortest path in transportation networks with two components of travel time uncertainty“. In: *Transportation Research* C 11(5) (2003), S. 331–354.

[Pea84] J. Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison & Wesley, 1984.

[PEH68] B. R. Peter E. Hart Nils J. Nilsson. „A Formal Basis for the Heuristic Determination of Minimum Cost Paths“. In: *IEEE Trans. Systems Science and Cybernetics* 4(2) (1968), S. 100–107.

[Pyr+04] E. Pyrga, F. Schulz, D. Wagner und C. D. Zaroliagis. „Towards Realistic Modeling of Time-Table Information through the Time-Dependent Approach“. In: *Electr. Notes Theor. Comput. Sci.* 92 (2004), S. 85–103.

[Pyr+07] E. Pyrga, F. Schulz, D. Wagner und C. D. Zaroliagis. „Efficient models for timetable information in public transportation systems“. In: *ACM Journal of Experimental Algorithmics* 12 (2007).

[Way] K. Wayne. *http://www.cs.princeton.edu/wayne/teaching/fibonacci-heap.pdf*.