Diplomarbeit

MEHRWEG

Marcus Mack 1996

Universität Stuttgart

Institut für Eisenbahn- und Verkehrswesen

MARCUS MACK

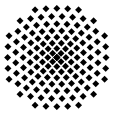
Burghaldenstr. 82/1
71065 Sindelfingen
✆ (07031) 813749

Prüfer:	Prof. Dr. rer. nat. Volker Claus	(Informatik)
	Prof. Dr.-Ing. Gerhard Heimerl	(Verkehrswesen)
Betreuung:	Dipl.-Inform. Friedhelm Buchholz	(Informatik)
	Dipl.-Inform. Jürgen Hoffmann	(Verkehrswesen)
begonnen am:	1. Dezember 1995	
beendet am:	12. Juli 1996	
CR-Klassifikation:	G.2.2	Graph Theory
	J.1	Administrative Data Processing

Diplomarbeit Nr. 1374

Untersuchung von effizienten Algorithmen zur Bestimmung der k-kürzesten Wege innerhalb von ÖPNV-Verkehrsnetzen

Marcus Mack



Universität Stuttgart

Fakultät Informatik
Institut für Informatik

Breitwiesenstraße 20-22
D-70565 Stuttgart

Fakultät Bauingenieurwesen
Institut für Eisenbahn- und Verkehrswesen

Pfaffenwaldring 7
D-70569 Stuttgart

Zusammenfassung

Die Verkehrsumlegung ist der letzte Teilschritt bei der klassischen Verkehrsprognose im öffentlichen Personennahverkehr. Die Prognoseschritte vor der Umlegung ermitteln für jeweils zwei Verkehrszellen A und B, die Anzahl an Fahrten, die täglich von A nach B erfolgen (z.B. 4000 Fahrten täglich von einem Vorort A zur Innenstadt).
Aufgabe der Umlegungsrechnung ist es, für diese Fahrten zu ermitteln, welche Wege (Routen) sie von A nach B benutzen. Die Belastung durch diese Fahrten wird dann auf die Teilabschnitte dieser Wege addiert. So erhält man am Ende der Umlegung die (prognostizierte) tägliche Belastung der einzelnen Busse, S-Bahnen, etc. und kann damit Einsatzgröße und Auslastung der Fahrzeuge bestimmen.

Bisher war es üblich, bei der Umlegung von A nach B den kürzesten Weg zu suchen und alle Fahrten von A nach B über diesen einen Weg laufen zu lassen. Dieses Modell vereinfacht die Realität jedoch in grobem Maße, da viele Menschen auch Alternativwege in Betracht ziehen, solange diese innerhalb vernünftiger Grenzen liegen. Deshalb ist es sinnvoller, nicht nur den Bestweg von A nach B zu verwenden, sondern die k-kürzesten Wege von A nach B. Damit wird dann ein Teil der Fahrten auch auf Alternativrouten verteilt.

Aufgabe dieser Diplomarbeit ist es, eine solche Umlegungsrechnung mit Alternativrouten (eine sogenannte Mehrwegumlegung) zu verwirklichen und dabei effiziente Algorithmen für die Bestimmung der k-kürzesten Wege zu untersuchen.
Auf der einen Seite geht es daher um die praktische Ausführung der Umlegungsrechnung im Verkehrswesen, unter Berücksichtigung aller verkehrlichen Aspekte und ihrer Einschränkungen, und auf der anderen Seite in der Informatik-Theorie, um die Untersuchung effizienter Algorithmen zur Bestimmung der k-kürzesten, schleifenfreien Wege in Graphen.

Inhaltsverzeichnis

1. Einleitung
- 1.1 Aufgabenstellung ... 5
- 1.2 Verkehrsgrundlagen ... 6
- 1.3 Ausgangsdaten für die ÖPNV-Umlegung ... 8
- 1.4 Verkehrsbegriffe ... 13
- 1.5 Vorgaben ... 14

2. Vorgehensweise
- 2.1 Vorgehensweise ... 15
- 2.2 Mehrwegsuche ... 16
- 2.3 Ablauf Mehrwegumlegung ... 18

3. Aufbau des Graphen
- 3.1 Definition des Graphen ... 19
- 3.2 Abbildung des Netzes auf den Graphen ... 21
- 3.3 Gewichtsfunktion ... 28

4. Algorithmen zur Wegsuche
- 4.1 Allgemeines ... 33
- 4.2 Dijkstra ... 35
- 4.3 Methoden aus der KI ... 36
- 4.4 Naiver Algorithmus ... 37
- 4.5 Floyd/Minieka ... 39
- 4.6 Hoffman & Pavley ... 42
- 4.7 Azevedo et. al. ... 47
- 4.8 Clarke et. al. ... 49
- 4.9 Yen ... 49
- 4.10 Weitere Algorithmen ... 52
- 4.11 Zusammenfassung ... 53
- 4.12 Wegecheck ... 53

5. Umlegung
- 5.1 Belastungsfaktoren der Wege ... 58
- 5.2 Mandatsverteilung ... 60
- 5.3 Umlegung auf Haltestellen und Linien ... 62

6. Implementation
- 6.1 Programmaufbau und -Ablauf ... 64
- 6.2 Programm-Module ... 66
- 6.3 Datenstrukturen und Funktionen ... 67
- 6.4 Programmparameter ... 74
- 6.5 Implementierungsgrenzen ... 75

7. Laufzeitergebnisse
7.1 Laufzeitsystem 76
7.2 Testnetze 78
7.3 Testläufe 79
7.4 Laufzeiten 79

Ausblick 83

Anhang A: Beispielgraph für Wegsuche 85

Anhang B: Aufbau der Dateistruktur 87

Anhang C: Beispielumlegung 91

Literaturverzeichnis 102

1. Einleitung

1.1 Aufgabenstellung

Im Rahmen von Verkehrsnachfragemodellen zur Prognose des zukünftigen Verkehrsaufkommens spielt der klassische Vier-Stufen-Algorithmus (siehe Abschnitt 1.2) eine große Rolle. Den letzten Schritt in diesem Algorithmus stellt die Verkehrsumlegung dar. Als Ergebnis dieser Umlegung, und somit der gesamten Verkehrsprognose, erhält man Daten über die zukünftige Belastung von Knotenpunkten und Verkehrsabschnitten im Netz.

Im vorliegenden Fall geht es speziell um die Umlegungsrechnung im öffentlichen Personennahverkehr (ÖPNV) einer Stadt oder Region. Man unterscheidet dazu zwei Arten der Umlegung:

- **Bestwegumlegung**
- **Mehrwegumlegung**

Während die Bestwegumlegung die gesamte Belastung einer Quell-Ziel-Beziehung auf den kürzesten Weg zwischen Quelle und Ziel verteilt, hat die Mehrwegumlegung die Aufgabe, die Belastung sowohl auf den kürzesten Weg, als auch auf den zweit-, drittkürzesten Weg und eine Reihe von weiteren Alternativrouten zu verteilen. Die Bestimmung dieser Alternativrouten erfolgt derart, daß alle gefundenen Wege noch innerhalb bestimmter Grenzen liegen müssen, damit sie von den Fahrgästen auch tatsächlich angenommen werden.
Auf Grund der großen Netze und der damit erheblichen Anzahl von Möglichkeiten für Wege, erhält der Algorithmus für die Wegsuche und seine Effizienz eine zentrale Rolle.

Ziel dieser Diplomarbeit ist es, verschiedene, schon vorhandene Algorithmen für die Wegsuche auf Eignung und Effizienz für die Mehrwegumlegung zu untersuchen und eine anschließende Realisierung der Mehrwegumlegung mit dem „optimalen“ Ansatz, damit auch größere Netze schnell bearbeitet werden können.
Insbesondere müssen spezielle verkehrliche Aspekte, wie parallele Verkehre und Umsteigevorgänge, die von den meisten Algorithmen nicht verarbeitet werden, korrekt beachtet werden.

Bisherige Umlegungsrechnungen verwenden für die Gewichtung der Wege meist nur „einfache“ Gewichte, wie Streckenlänge und Fahrtdauer. Daher ist zusätzlich zu prüfen, um welche Parameter des Verkehrsangebots sich die bisherige Gewichtungsfunktion der Wege erweitern läßt (z.B. Wartezeiten, Komfortkriterien, etc.).

1.2 Verkehrsgrundlagen

Dieser Abschnitt soll die verkehrlichen Grundlagen der Verkehrsprognose grob erläutern.

Die Prognose des zukünftigen Verkehrsaufkommens spielt im Verkehrswesen eine sehr wichtige Rolle. Auf Grund solcher Prognosen können neue Verkehrsnetze dimensioniert oder bestehende Netze ausgebaut und verbessert werden.
Zur Bestimmung der zukünftigen Verkehrsnachfrage wendet man üblicherweise den klassischen Vier-Stufen-Algorithmus an:

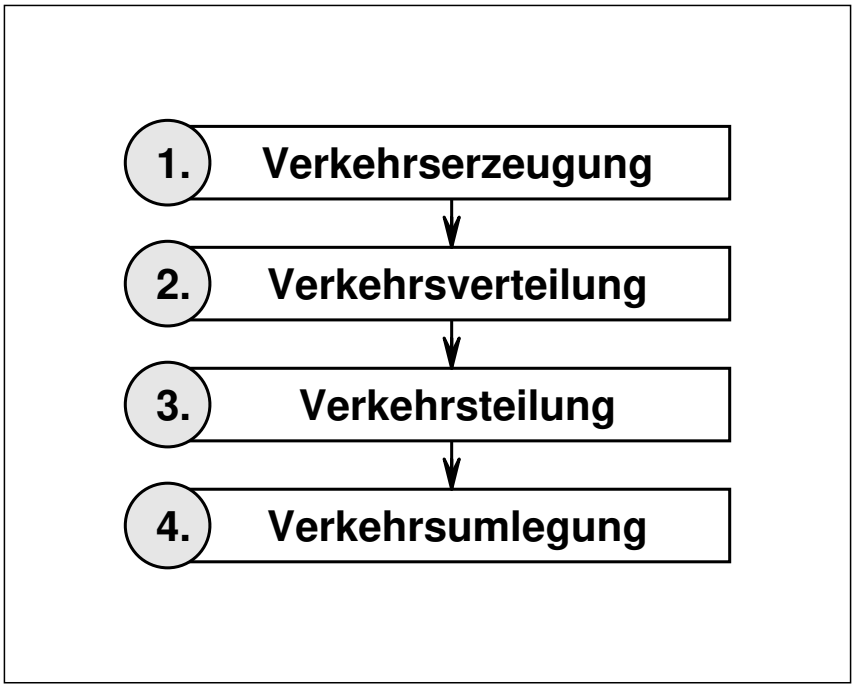
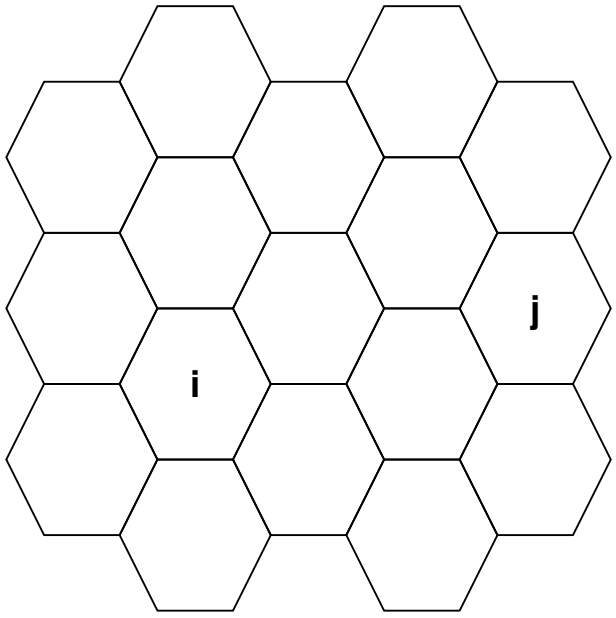
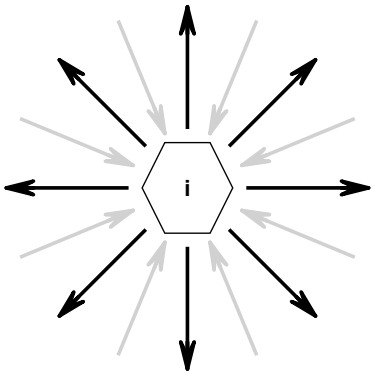


Abb. 1.1: Der klassische Vier-Stufen-Algorithmus

Zu Anfang wird das zu untersuchende Verkehrsnetz/-gebiet in Verkehrszellen, sogenannte Bezirke, aufgeteilt:

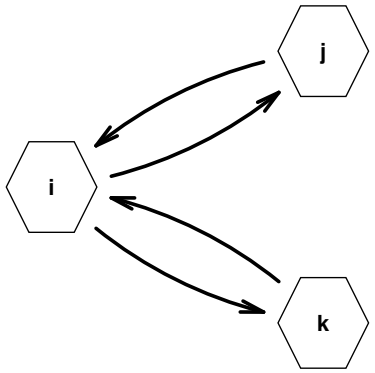


Diese Bezirke bilden die Basiseinheit für die zu prognostizierenden Verkehrsbeziehungen. Je feiner die Einteilung des Untersuchungsgebiets vorgenommen wird, desto detaillierter, aber auch aufwendiger, wird das Ergebnis. Dies bedeutet, man kann entweder aus jedem Vorort einen Bezirk machen, oder man verfeinert so weit, bis jede Haltestelle und ihr Einzugsgebiet (z.B. ein paar Häuserblocks) einen Bezirk bilden, vorausgesetzt, die Daten hierfür liegen vor. Diese Bezirksgröße wäre das Optimum, verursacht gleichzeitig aber auch den größten Berechnungsaufwand.
Üblicherweise werden im Kernbereich des Untersuchungsgebietes (z.B. Altstadt/Innenstadt) drei bis vier Haltestellen zu einem Bezirk zusammengefaßt. In Richtung des Randbereiches wird dann das Raster der Bezirkseinteilung immer gröber (z.B. jeder Stadtteil/Vorort ein Bezirk).
An Hand beliebiger Bezirke i und j wird nun der Vier-Stufen-Algorithmus erläutert.



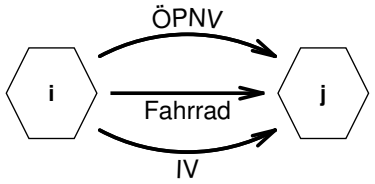
1. Stufe: Verkehrserzeugung

Die erste Stufe ermittelt den Quell- und Zielverkehr jedes Bezirkes. Darunter versteht man die Anzahl an Fahrten die täglich von diesem Bezirk ausgehen bzw. ankommen.
Diese Prognosewerte erhält man unter anderem aus den Einwohner- und Arbeitsplatzzahlen pro Bezirk (den sogenannten Strukturdaten). Damit ist es möglich Fahrten vom und zum Arbeitsplatz, Einkaufsfahrten und Freizeitfahrten abzugrenzen (z.B. von einem Wohngebiet, das sehr wenige Arbeitsplätze bietet, werden viele Fahrten in andere Bezirke mit mehr Arbeitsplätzen gehen, etwa zu einem Industriegebiet).



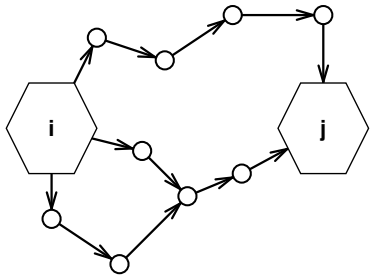
2. Stufe: Verkehrsverteilung

In dieser Stufe wird der Quell- und Zielverkehr aus der Verkehrserzeugung auf die entsprechenden umliegenden Bezirke verteilt. Das heißt, es ist nicht mehr nur bekannt, wie viele Fahrten täglich von einem Bezirk i ausgehen (Quellverkehr), sondern auch zu welchen anderen Bezirken diese führen.
Dadurch erhält man eine Fahrtenmatrix, die für beliebige Bezirke i und j die täglichen Fahrten von i nach j enthält.



3. Stufe: Verkehrsteilung

Hier wird für jede Verkehrsbeziehung von einem Bezirk i zu einem Bezirk j der Modal Split errechnet. Darunter versteht man die Aufteilung der Fahrten von i nach j auf die vorhandenen Verkehrsmittel: Individualverkehr (IV; Auto), Fahrrad, öffentliche Verkehrsmittel (ÖPNV), oder die eigenen Füße (Fußweg). Eine direkte Aufteilung auf die ÖPNV-Verkehrsmittel Bus, S-Bahn, Eisenbahn etc. erfolgt nicht, da diese normalerweise nicht überall im ganzen Bezirk verfügbar sind, sondern nur an bestimmten Haltstellen, während auf das Auto, das Fahrrad oder die eigenen Füße jederzeit zurückgegriffen werden kann.
Die Entscheidung für ein ÖPNV-Verkehrsmittel erfolgt erst im vierten Schritt: wenn die Fahrtroute bekannt ist.



4. Stufe: Verkehrsumlegung

Bis jetzt ist bekannt, wieviele Fahrten täglich von einen Bezirk i zu einem Bezirk j mit welchem Verkehrssystem (IV, ÖPNV) unternommen werden. Es bleibt noch zu klären, auf welchem Weg (Route) diese Fahrten stattfinden.
Bei der Bestwegumlegung wird wie erwähnt, diese Zahl der Fahrten nur auf den Bestweg (kürzesten Weg) von i nach j umgelegt, während bei einer Mehrwegumlegung mehrere Wege von i nach j in Betracht gezogen werden.
Die Belastung durch die Fahrten von i nach j wird dann auf diese Wege verteilt und aufaddiert.

Als Ergebnis dieses Algorithmus erhält man die einzelnen Belastungen der Haltestellen und Linien im Untersuchungsgebiet (z.B. wieviele Einsteiger, Umsteiger und Aussteiger täglich an einer bestimmten Haltestelle auftreten). Mit diesen Zahlen kann dann bestimmt werden, ob eine Haltestelle oder Linie ausgelastet ist, oder erweitert bzw. eingestellt werden sollte.

Nähere Informationen zu den Grundlagen der Verkehrsprognose findet man in [Dobeschinsky 89] und [Grote 89].

1.3 Ausgangsdaten für die ÖPNV-Umlegung

In diesem Kapitel möchte ich kurz die Ausgangdaten für die ÖPNV-Mehrwegumlegung an Hand eines Beispiels beschreiben.

Wie schon erwähnt wird das Untersuchungsgebiet in Bezirke eingeteilt:

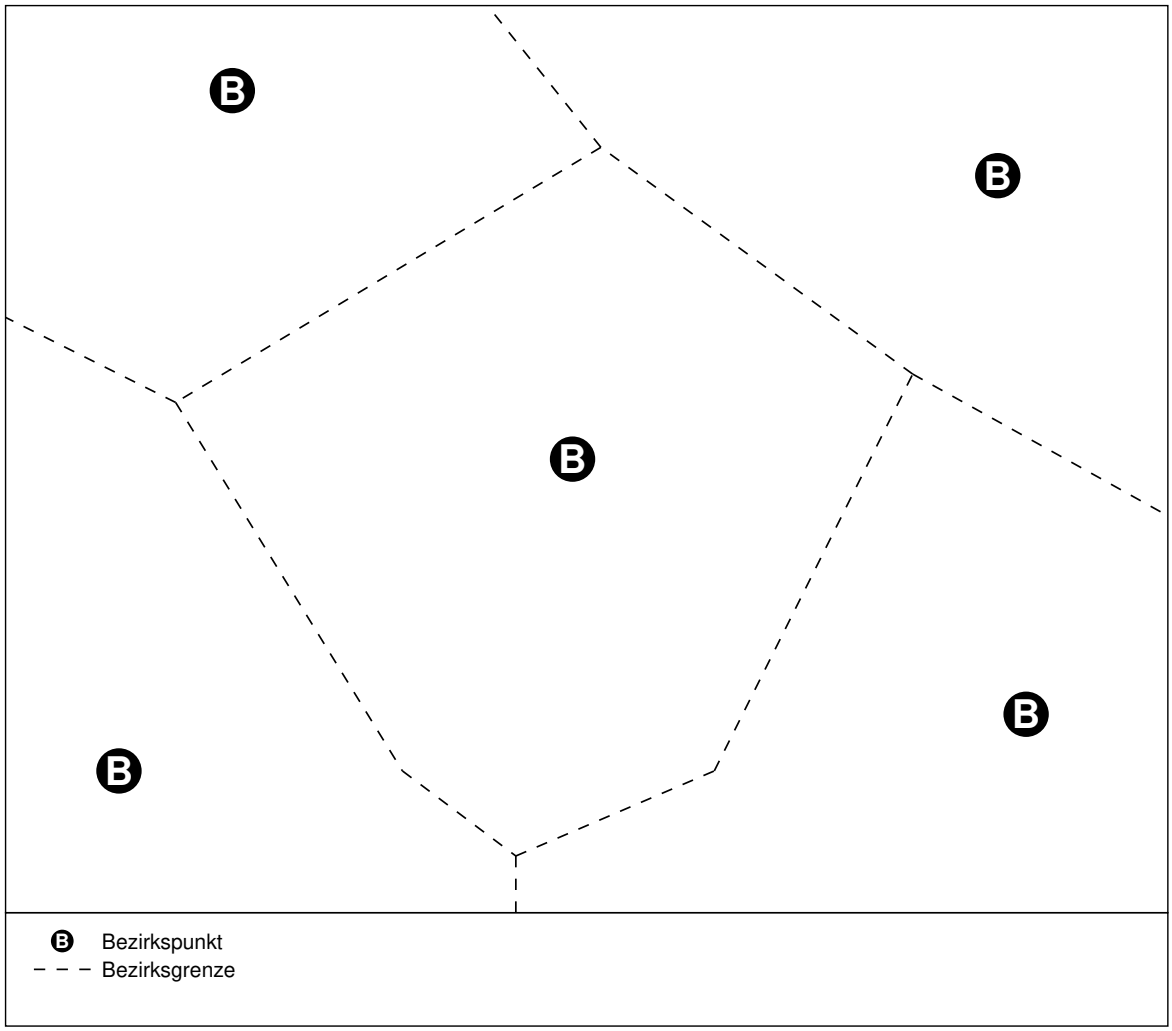


Abb. 1.7: Bezirke

Jeder Bezirk wird dabei durch einen einzelnen Punkt, den Bezirkspunkt, repräsentiert. Man muß sich die gesamte Bevölkerung eines Bezirks an diesem Punkt konzentriert denken. Daher finden alle zu verteilenden Fahrten immer von einem Bezirkspunkt zu einem anderen statt. Dies ist zwar nicht sehr realistisch, aber zweckmäßig, und je feiner die Bezirksaufteilung vorgenommen wird, desto mehr entspricht dieses Modell der Wirklichkeit.

Weiterhin enthält das Untersuchungsgebiet neben den Bezirken und Bezirkspunkten auch Haltestellen und Linien, die zwischen diesen Haltestellen verkehren:

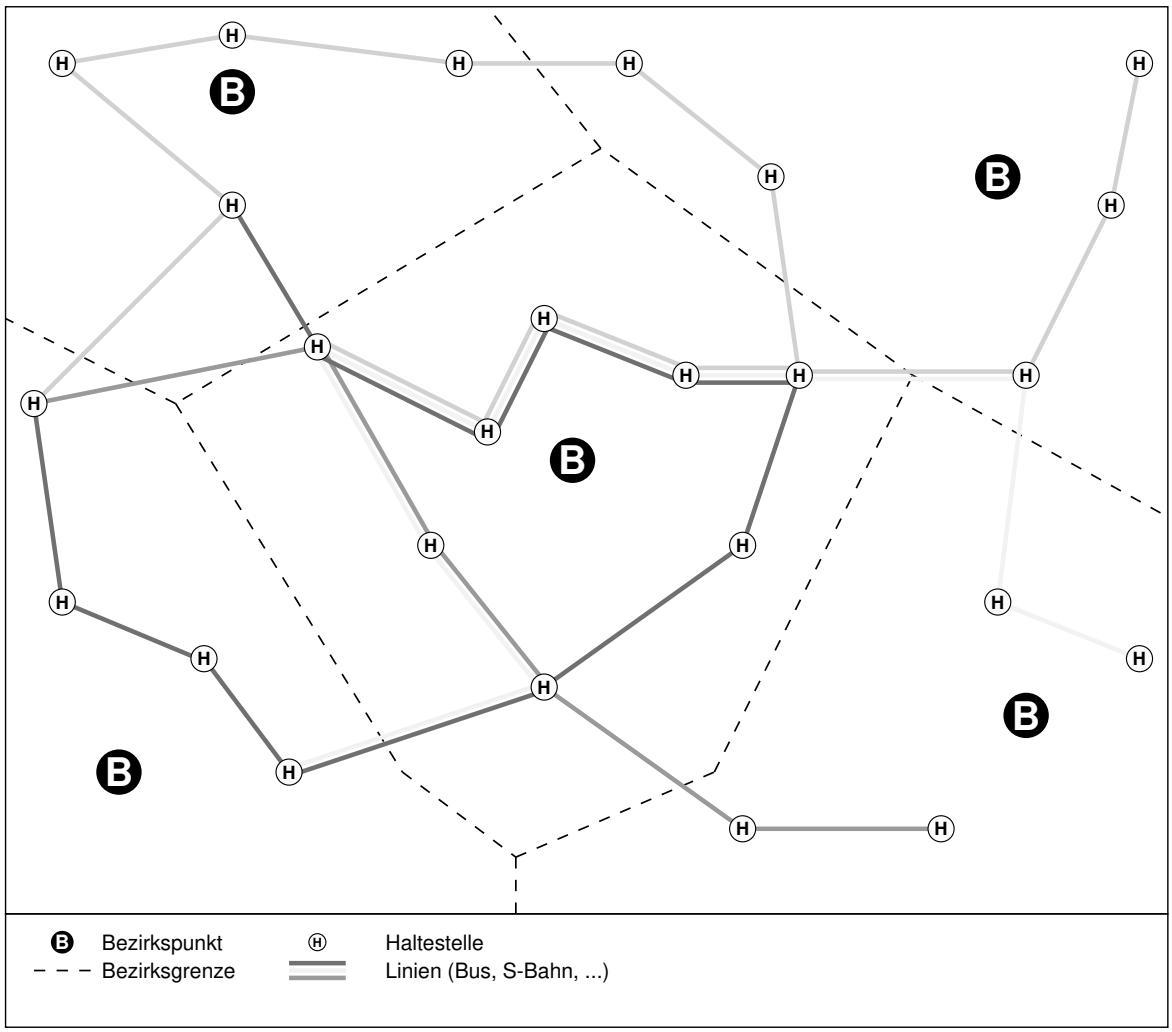


Abb. 1.8: Haltestellen und Linien

Jeder Bezirk kann beliebig viele Haltestellen enthalten, wobei der Bezirkspunkt jedoch niemals selbst eine Haltestelle ist. Zwischen den Haltestellen verlaufen Linien von öffentlichen Verkehrsmitteln:

- Bus
- Straßenbahn
- Stadtbahn
- U-Bahn
- S-Bahn
- Eisenbahn

Für jede Linie gibt es einen Streckenverlauf von einer Starthaltestelle zu einer Zielhaltestelle und von der Zielhaltestelle zurück zur Starthaltestelle. Dies wird mit Richtung und Gegenrichtung bezeichnet, wobei die Verläufe von Richtung und Gegenrichtung nicht symmetrisch sein müssen und durchaus voneinander abweichen können.
Verlaufen von einer Haltestelle zu einer anderen mehrere Linien nebeneinander, so spricht man von Parallelverkehr.

Zusätzlich sind noch Fußwege im Netz enthalten, die die Bezirkspunkte mit bestimmten Haltestellen verbinden:

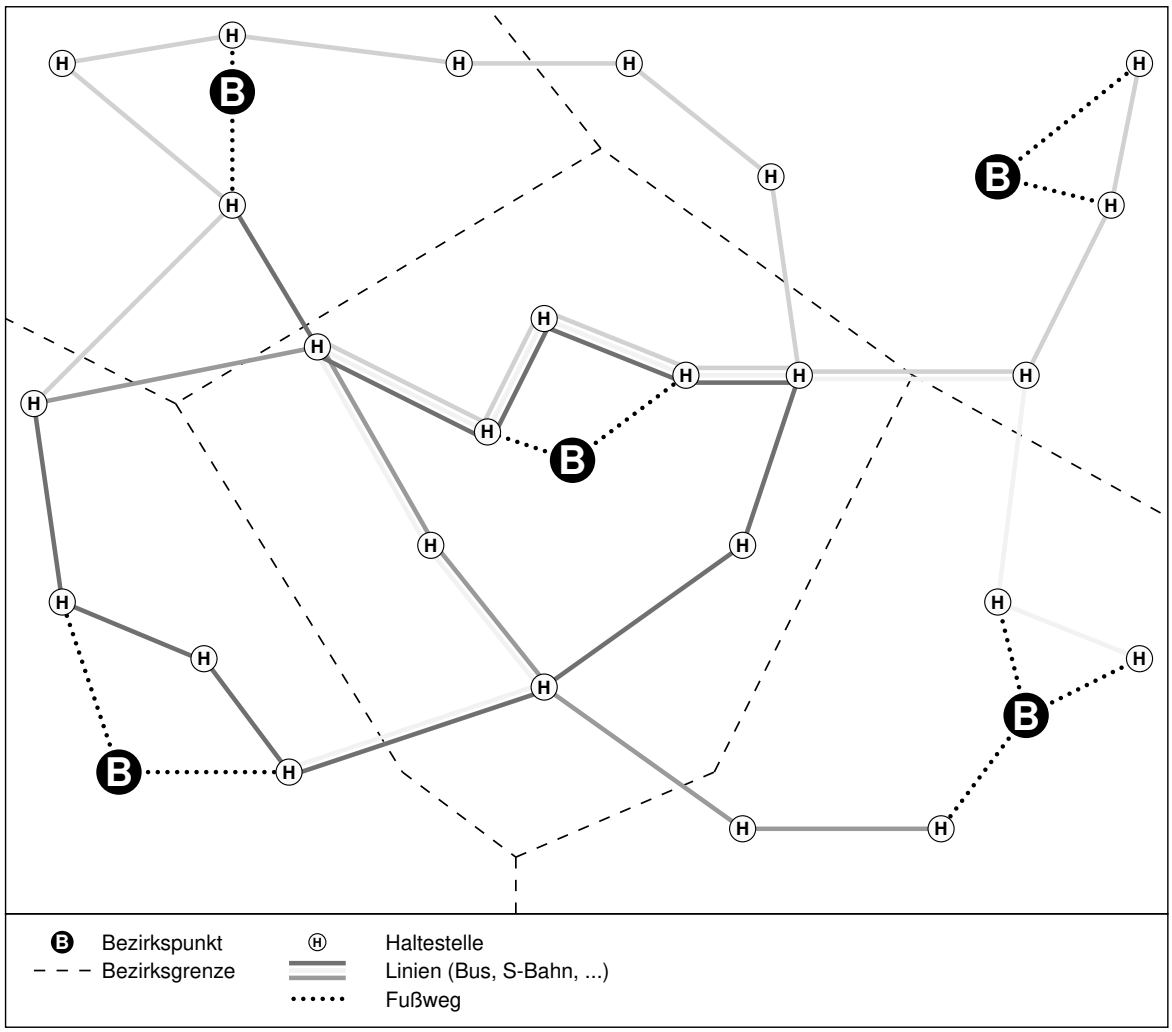
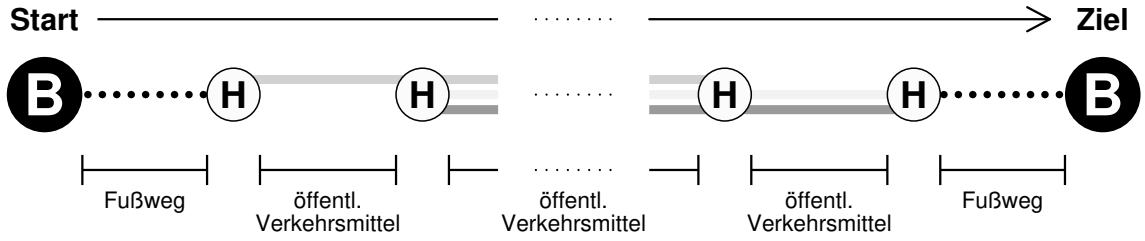


Abb. 1.9: Fußwege

Diese Fußwege werden „Anbindungen“ genannt, da sie den Bezirkspunkten erst den Zugang an das eigentliche Verkehrsnetz ermöglichen. Damit soll modelliert werden, daß normalerweise eine Person keine Haltestelle direkt vor ihrem Haus hat, sondern erst einen Fußweg zur nächsten Haltestelle zurücklegen muß. Üblicherweise ist jeder Bezirkspunkt an zwei bis drei Haltestellen in seiner nächsten Umgebung angebunden.
Da Fahrten, die von Personen eines Bezirkes unternommen werden, immer von Bezirkspunkt zu Bezirkspunkt verlaufen, besitzt jede Fahrt den gleichen Aufbau:



Es wird davon ausgegangen, daß keine zwei Bezirkspunkte an die gleiche Haltestelle angebunden sind. Somit enthält jede Fahrt von einem Bezirkspunkt zu einem anderen mindestens eine Teilfahrt mit einem öffentlichen Verkehrsmittel.

In Abbildung 1.10 sind nun einige Fahrten von Bezirkspunkten zu anderen Bezirkspunkten dargestellt. Jeder dieser schwarzen Pfeile wird im weiteren als „**Bezirkspaar**“ bezeichnet und umfaßt folgende Informationen:

den Startbezirk, den Zielbezirk und die Anzahl Fahrten pro Tag.

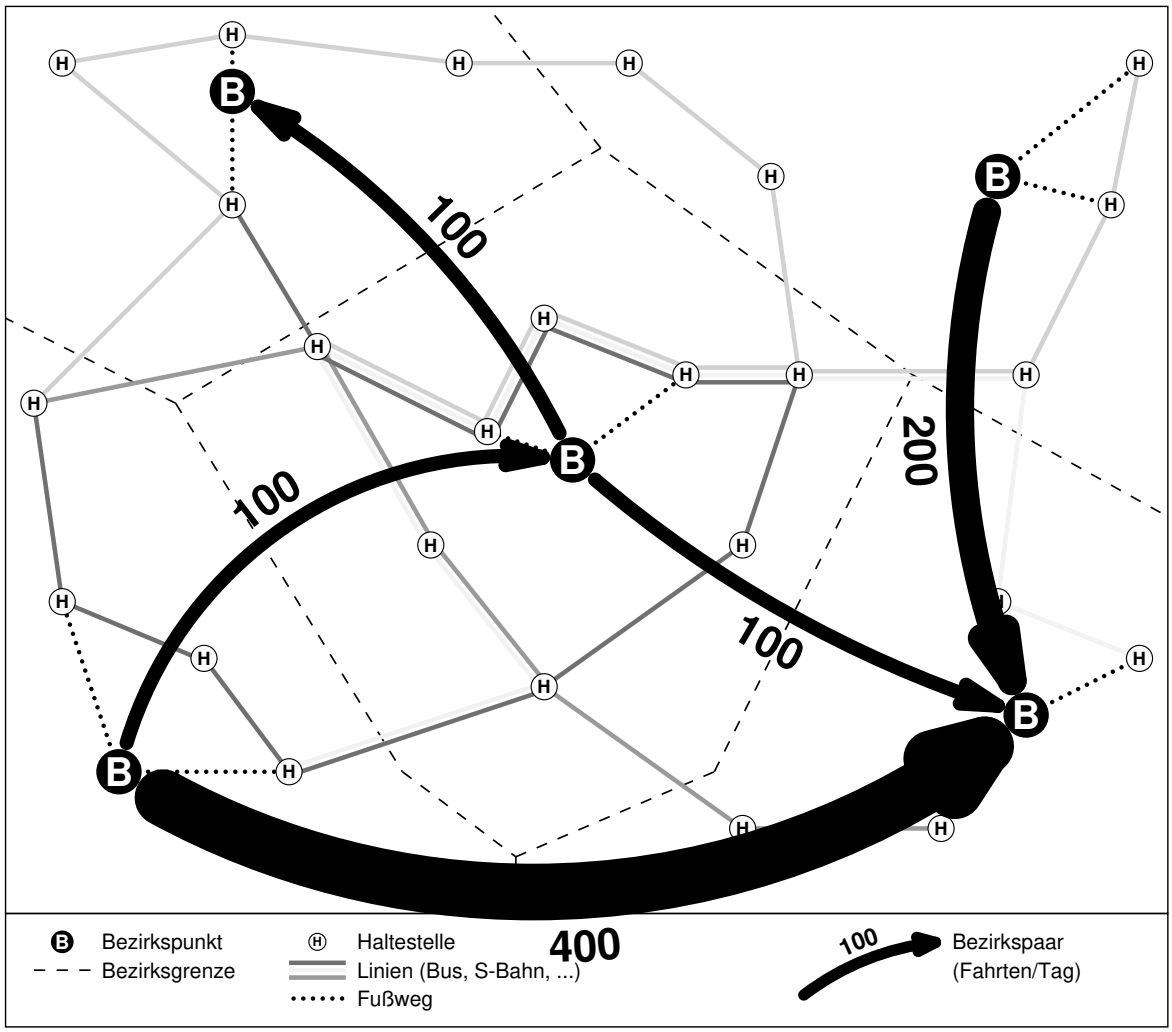


Abb. 1.10: Bezirksbeziehungen

Im obigen Beispiel sind aus Gründen der Übersicht nur wenige Bezirkspaare eingezeichnet. Normalerweise hat eine Umlegungsrechnung wesentlich mehr Paare zu berücksichtigen, da Personen, die zum Beispiel morgens eine Fahrt zum Arbeitsplatz unternehmen, mit großer Sicherheit abends zurück nach Hause fahren werden. Andererseits sind diese Bezirkspaare meist nicht vollständig, das heißt, es besteht nicht immer von jedem Bezirk zu allen anderen Bezirken ein Bezirkspaar, da für die Personen eines Bezirks nicht alle anderen Bezirks interessant erscheinen. Beispielsweise wird es von einem Wohnbezirk, der am Rande des Untersuchungsgebiets liegt, zu einem anderen Wohnbezirk, auf der anderen Seite des Untersuchungsgebiets, kaum Fahrten geben.
Durch diese Fahrten ändert sich die Verteilung der Personen über das Stadtgebiet im Verlauf des Tages immer wieder. Sie ist jedoch zu Beginn und am Ende des Tages immer gleich.

Die Umlegungsrechnung muß nun zu jedem vorgegebenem Bezirkspaar die entsprechenden Wege im Netz ermitteln und die Anzahl der Fahrten darauf verteilen.

In Abbildung 1.11 ist nun das Ergebnis einer Umlegung der Bezirkspaare aus Abbildung 1.10 zu sehen:

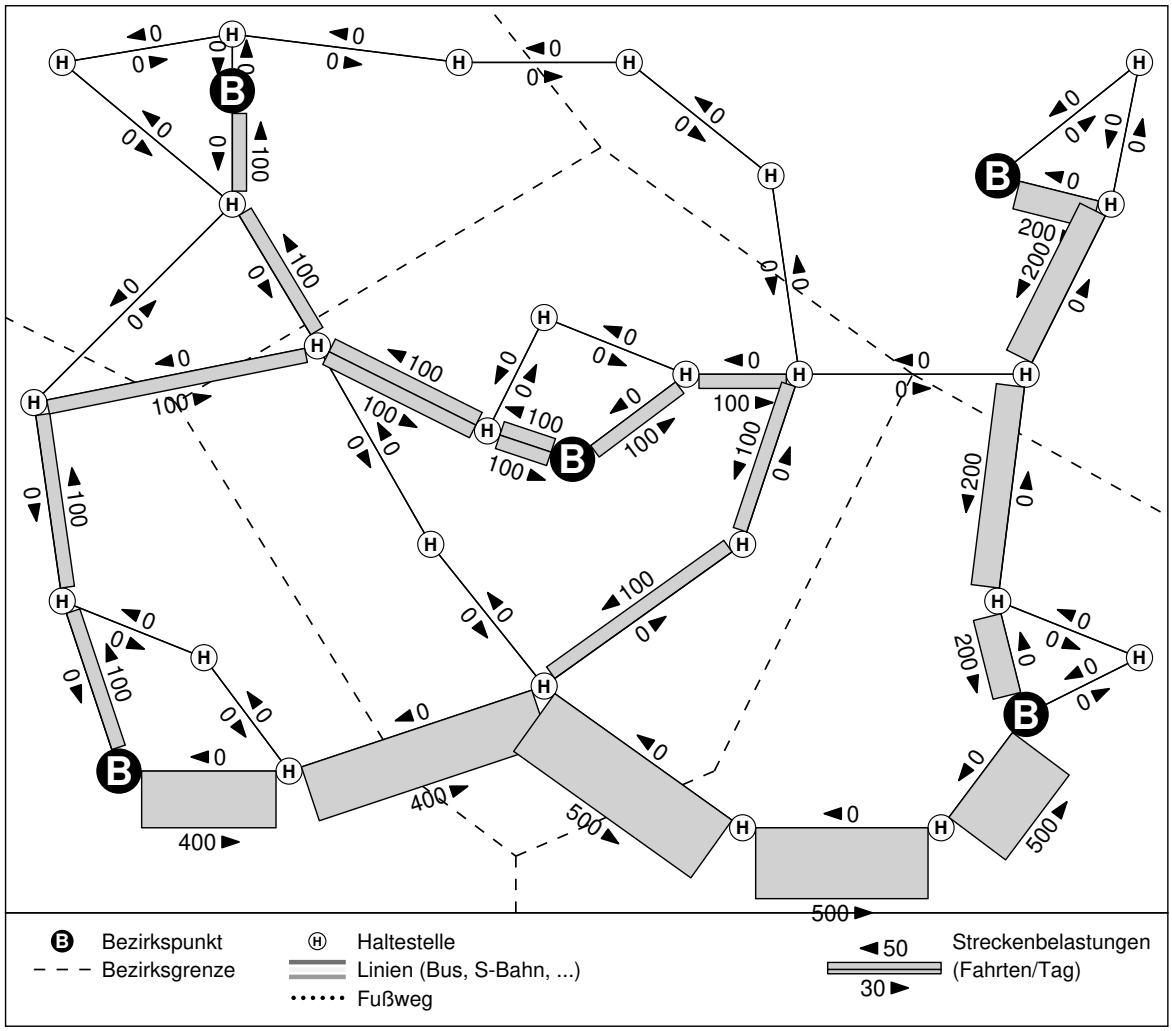


Abb. 1.11: Streckenbelastungen

Die Umlegung der Fahrten erfolgte aus Übersichtsgründen nur auf dem Bestweg.

Zu sehen ist, wie die „globale“ Belastung, die durch jedes Bezirkspaar verursacht wird, auf eine „lokale“ Belastung einzelner Streckenabschnitte in einer Richtung umgerechnet wird. Dabei summieren sich die Belastungen von verschiedenen Bezirkspaaren, die auf den gleichen Streckenabschnitt umgelegt werden (siehe Abb. 1.11, untere Mitte: die Belastungen von 100 und 400 Fahrten/Tag addieren sich zur Gesamtbelastung von 500 Fahrten/Tag).
Die Umlegungsrechnung ermittelt weniger die Streckenbelastung (wie in Abb. 1.11 zu sehen), sondern vielmehr die Linienbelastung, das heißt, wieviele Fahrten pro Tag auf jedem Teilabschnitt einer Linie erfolgen. Solte ein Streckenabschnitt zwischen zwei Haltestellen nur von einer Linie befahren werden, so entspricht auf diesem Abschnitt die Streckenbelastung der Linienbelastung.
Zusätzliche wird auch die Belastung für jede Haltestelle ermittelt: Wieviele Fahrten (Personen) an jeder Haltestelle begonnen oder beendet wurden und wieviele hier umgestiegen sind.

Die Daten für eine Umlegungsrechnung bestehen demnach aus zwei Teilen:

den	**Netzdaten**	(Alle Angaben über Bezirke, Haltestellen und Linien)
und		
den	**Belastungsdaten**	(Die Menge aller zu berechnenden Bezirkspaare).

1.4 Verkehrsbegriffe

Zum besseren Verständnis werden hier die wichtigsten Verkehrsbegriffe zu diesem Thema und ihre Bedeutung kurz definiert:

Belastung
Maßangabe, wie häufig eine Haltestelle, ein Linienabschnitt oder ein ganzer Weg benutzt wird. Diese Zahl wird im allgemeinen in Fahrten pro Tag angegeben.

Bestweg
Derjenige Weg von einem bestimmten Startpunkt A zu einem Ziel B, der von allen Wegen von A nach B das kleinste Gewicht hat. Haben mehrere Weg von A nach B das gleiche, kleinste Gewicht, so ist der Bestweg, ein beliebiger von diesen.

Bezirk
Nach bestimmten Kriterien abgegrenzter Teil des zu untersuchenden Raumes (z.B. Stadt oder Großraum). Enthält immer einen eindeutigen Bezirkspunkt, der den zentralen „Schwerpunkt“ aller Einwohner dieses Bezirkes darstellt.

Bezirkspaar
Teil der Ausgangsdaten für die Umlegungsrechnung. Enthält Startbezirkspunkt, Zielbezirkspunkt und Belastung von Start nach Ziel. Für jedes Bezirkspaar der Ausgangsdaten muß die Umlegungsrechnung die entsprechenden Wege berechnen.

Fahrt
Ortsveränderung einer Person von einem Startpunkt zu einem Ziel von mindestens einem öffentlichen Verkehrsmittel. Da Fahrten immer Fahrgastbezogen sind, entspricht ein Bus, der mit 50 Fahrgästen von A nach B fährt genau 50 Fahrten von A nach B.

Haltestelle
Start- oder Zielpunkt einer Fahrt. Gleichzeitig definierter Punkt des Verkehrsnetzes, der es Personen ermöglicht, an diesem Punkt in eine Verkehrslinie, die diese Haltestelle benutzt, einzusteigen oder diese zu verlassen.

Linie
Damit ist eine Verkehrslinie gemeint (z.B. Buslinie, Eisenbahnlinie). Jede Linie hat für Richtung und Gegenrichtung einen Streckenverlauf, der aus einzelnen Linienabschnitten besteht.

Linienabschnitt
Kleinster Teil des Linienverlaufes. Das heißt, die Verbindung zwischen einer Haltestelle und der darauf folgenden Haltestelle des Linienverlaufs.

Parallelverkehr
Wenn mehrere Verkehrslinien die gleiche Strecke in der gleichen Richtung befahren. Parallelverkehr im engeren Sinne ist dann gegeben, wenn diese Verkehrslinien alle genau der gleichen Verkehrsmittelart angehören, da sie dann meistens die gleiche Straße/Schiene benutzen (z.B: mehrere S-Bahnlinien die auf dem gleichen Gleis von A nach B fahren).

Strecke
Kleinste Verbindung zwischen zwei Haltestellen, die auch tatsächlich von einer Linie befahren wird, aber unabhängig von einer Linie zu sehen ist. Sie besteht daher aus mindestens einem Linienabschnitt einer Linie, kann aber bei Parallelverkehr mehrere Linienabschnitte von verschiedenen Linien enthalten. Enthält keine weiteren Haltestellen zwischen Anfangs- und Endpunkt.

Verkehrsmittel(art)

Art des Beförderungsmittels. In dieser Arbeit sind folgende öffentliche Verkehrsmittel vorgesehen:

- 0 : Fußgänger
- 1 : Bus
- 2 : Straßenbahn
- 3 : Stadtbahn (Straßenbahn die auf eigenem Gleiskörper fährt, also nicht in der Straße zwischen dem normalen Autoverkehr ; wird z.B. in Stuttgart eingesetzt).
- 4 : U-Bahn
- 5 : S-Bahn
- 6 : Eisenbahn

Fußgänger sind keine Verkehrsmittel im engeren Sinne, werden hier jedoch aufgeführt, damit zusätzliche Fußwege im Netz (nicht die Fußwege zur Anbindung von Bezirkspunkten an die Haltestellen) modelliert werden können.

Die Zahl vor dem jeweiligen Namen in der obigen Liste bezeichnet die Kennziffer für dieses Verkehrsmittel. Diese Liste und die Kennziffern sind Vorgaben des Verkehrswesens, da sie in dieser Art auch in anderen Programmen Verwendung finden.

Zusätzlich gibt es noch den Begriff der Schnellbahnverkehrsmittel. Damit sind die etwas komfortableren Verkehrsmittel Stadtbahn und S-Bahn gemeint.

1.5 Vorgaben

Diese Vorgaben habe ich vom Institut für Eisenbahn- und Verkehrswesen der Universität Stuttgart für meine Arbeit erhalten:

- Die Art und die Zusammensetzung der Gewichtskomponenten.
- Die Vorgehensweise bei der Verteilung (Umlegung) der Belastungen auf die einzelnen Wege.
- Verschiedene Ausgangsnetze und -Belastungen für Test und Laufzeitversuche.
- Die Dateiformate für Eingangsdaten und Ergebnisdaten.

weiterhin:

- Das Programm ist auf einem PC in Borland Pascal für Windows 7.0 zu erstellen. Ein modularer Aufbau soll eine eventuelle, spätere Ergänzung oder Wartung erleichtern.

2. Lösungsansatz

2.1 Vorgehensweise

Die Wegsuche in Netzen ist ein klassischer Teil der Graphentheorie.
Dazu wird das Verkehrsnetz mit einem Graphen nachgebildet und kann dann mit verschiedenen Algorithmen auf Wege untersucht werden. Die Graphentheorie bietet hierzu eine Menge von Standardalgorithmen für die Wegsuche. Allerdings setzen fast alle dieser Algorithmen einen gerichteten Digraphen voraus (gerichtete Digraphen sind Graphen mit relativ einfachem Aufbau; werden in Kapitel 3 näher erläutert).
Das Verkehrsnetz enthält aber einige Konstrukte, die eine einfache Übertragung in einen gerichteten Digraphen nicht zulassen. Zu diesen Konstrukten gehört zum Beispiel der Parallelverkehr, die Umsteigegewichte und anderes. Auch dazu mehr in Kapitel 3.
Deshalb ergeben sich zwei Möglichkeiten des Vorgehens:

A Verwendung eines komplexeren Graphentyps (z.B. Multigraph), der den Verkehrsnetzen vom Aufbau her ähnlicher ist. In diesem Fall läßt sich das Verkehrsnetz sehr einfach in den Graphen überführen. Andererseits müssen nun neue Algorithmen für diesen Graphentyp gefunden werden, oder die bisherigen Standardalgorithmen für Digraphen an diesen Graphentyp angepaßt werden.

B Die andere Möglichkeit wäre, das Verkehrsnetz durch eine komplexe und aufwendige Abbildung doch in einen gerichteten Digraphen zu überführen, so daß trotzdem die üblichen Algorithmen für die Wegsuche verwendet werden können.

Im ersten Fall müßte jeder Standardalgorithmus an den neuen Graphentyp angepaßt werden und da es auch Aufgabe dieser Arbeit ist, verschiedene Algorithmen zu untersuchen und zu vergleichen, würde dieser Anpassungsaufwand mehrmals entstehen.
Im zweiten Fall entsteht nur einmal Aufwand. Nämlich bei der Überführung des Verkehrsnetzes in den gerichteten Digraphen.
Diese beiden Möglichkeiten sollen durch Abbildungen 2.1 und 2.2 noch einmal verdeutlicht werden:

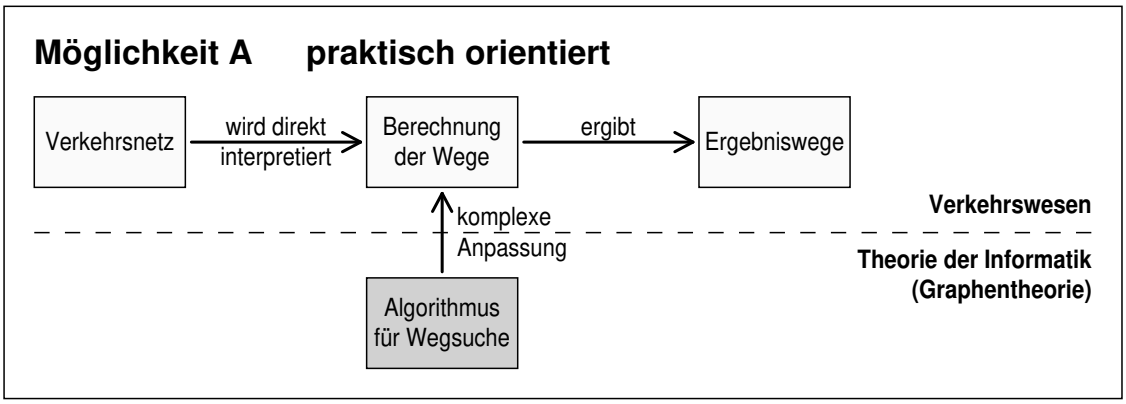


Abb. 2.1: mögliche Vorgehensweise A

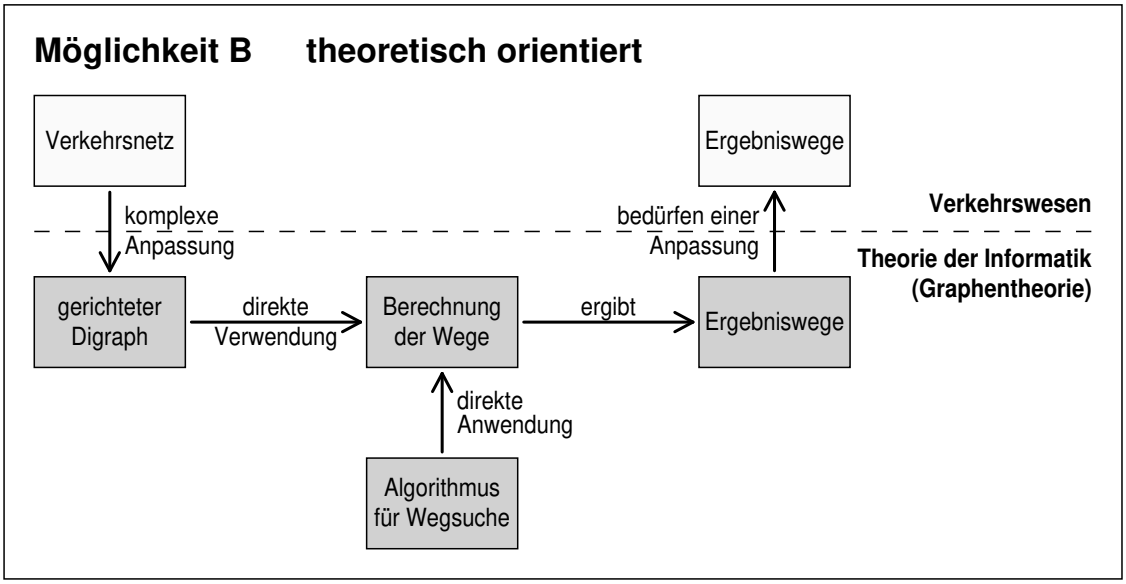


Abb. 2.2: mögliche Vorgehensweise B

Da im zweiten Fall weniger Anpassungsaufwand anfällt und sich die Algorithmen zur Wegsuche hier sehr schön auf der Theorieseite (unabhängig vom Verkehrswesen) untersuchen und vergleichen lassen, habe ich mich entschieden, auf die zweite Art vorzugehen.
Trotz dieser Vorgehensweise wird es bei der Implementierung dann doch notwendig, kleinere Anpassungen an den Algorithmen vorzunehmen, da das Ergebnis wieder auf die Seite des Verkehrswesens abgebildet werden muß.
Auf Grund der großen Datenmengen von üblichen Verkehrsnetzen, ist es abzusehen, daß die Mehrwegumlegung mit großem Rechenaufwand verbunden sein wird. Daher spielt die Effizienz des nachher zum Einsatz kommenden Algorithmus für die Laufzeit eine zentrale Rolle. Trotzdem war es von vorne herein mein Ziel, zunächst auf jeden Fall eine **korrekte** Lösung für die Umlegungsrechnung zu finden. Die Möglichkeit, der Verwendung von Heuristiken sowohl bei der Überführung des Netzes in den Graphen, als auch bei dem Algorithmus zur Wegsuche, um den Berechnungsaufwand zu reduzieren und damit die Laufzeit herabzusetzen, sollte erst danach in einem zweiten Schritt Verwendung finden.
Erst wenn eine korrekte Berechnungsmethode zur Verfügung steht und diese einen zu hohen Berechnungsaufwand verursacht, kann auf der praktischen Seite (Verkehrswesen) über den Einsatz von Heuristiken nachgedacht werden. Eine sinnvolle Abschätzung der Vor- und Nachteile einzelner Heuristiken, nämlich Laufzeitverbesserung auf der einen Seite und mögliche Fehler im Berechnungsergebnis auf der anderen, ist erst dann sinnvoll, wenn Laufzeit und Ergebnis einer korrekten Lösung vorhanden sind.

2.2 Mehrwegsuche

Dieses Kapitel soll erläutern, welche Wege für die Mehrwegumlegung in Frage kommen.
Jede Fahrt von einem Bezirkspunkt zu einem anderen wird auf einem bestimmten Weg durchgeführt. Diese Wege erhalten ein Gewicht zugewiesen, das aus mehreren Komponenten besteht (siehe Kapitel 3). Je höher das Gewicht eines Weges ist, desto länger ist seine Ausdehnung und desto mehr Zeit wird für ihn benötigt. Dies bedeutet, die Fahrgäste versuchen Wege mit möglichst geringem Gewicht zu verwenden.
Die Bestwegumlegung berechnet für jedes Bezirkspaar den besten Weg vom Startbezirk zum Zielbezirk und verteilt dann die gesamte Belastung dieses Bezirkspaares auf diesen einen Weg (unter Bestweg wird derjenige Weg, mit dem geringsten Gewicht verstanden).
Ziel der Mehrwegumlegung ist es, zusätzlich zum Bestweg auch Alternativwege zu berücksichtigen, die ein Gewicht haben, das innerhalb eines bestimmten Grenzabstandes zum Gewicht des Bestweges liegt. Hierzu werden zwei Grenzfaktoren eingeführt:

der **relative** und der **absolute** Grenzfaktor.

Der relative Grenzfaktor ist eine Prozentzahl, die angibt, um wieviel das Gewicht eines Alternativweges vom Gewicht des Bestweges maximal abweichen darf, damit der Alternativweg für die Umlegung noch berücksichtigt wird.

Beispiel: Der Bestweg hat ein Gewicht von 140 und der relative Grenzfaktor liegt bei 0,20 (20%). Dann werden bei der Mehrwegumlegung alle Wege berücksichtigt, die ein Gewicht zwischen 140 und 168 haben.

Der absolute Grenzfaktor ist eine absolute Gewichtszahl, die ebenfalls angibt, um wieviel das Gewicht eines Alternativweges maximal abweichen darf.

Beispiel: Der Bestweg hat wieder ein Gewicht von 140 und der absolute Grenzfaktor liegt bei 40. Dann werden alle Wege berücksichtigt, die ein Gewicht zwischen 140 und 180 haben.

Während der Umlegung findet aber immer nur einer der beiden Grenzfaktoren Anwendung. Dazu wird bei jedem Bezirkspaar (von A nach B) folgendermaßen verfahren:

1. Ermittlung des Bestweges von A nach B
2. Ermittlung des relativen Grenzgewichts
   relatives Grenzgewicht = Gewicht des Bestwegs
   x (1,00 + relativer Grenzfaktor)
3. Ermittlung des absoluten Grenzgewichts
   absolutes Grenzgewicht = Gewicht des Bestwegs
   + absoluter Grenzfaktor
4. Ermittlung des Grenzgewichts
   Grenzgewicht = Minimum (relatives Grenzgewicht,
   absolutes Grenzgewicht)
5. Ermittlung aller Wege von A nach B deren Gewicht zwischen dem Gewicht des Bestweges und dem Grenzgewicht liegt. Diese Wege werden zu einer Menge zusammengefaßt und als „gültig“ bezeichnet (siehe Abb. 2.3).
6. Verteilung der Belastung dieses Bezirkspaares auf die Wege in Schritt fünf gefunden wurden (einschließlich des Bestweges).

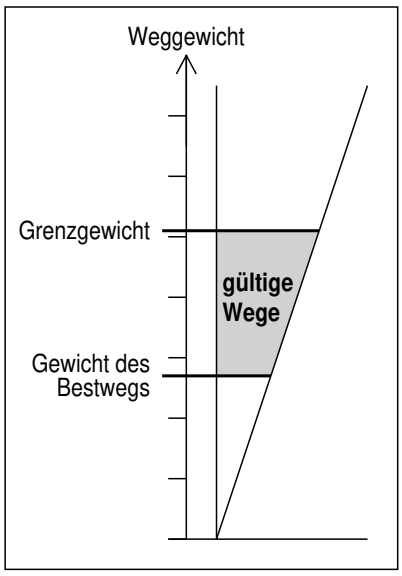


Abb. 2.3: Menge der gültigen Wege

Es wird also immer nur das Minimum der beiden Grenzgewichte verwendet. Je nachdem, ob das Gewicht des Bestweges sehr klein oder sehr groß ist, kommt entsprechend der relative oder der absolute Faktor zum Tragen (siehe Abb. 2.4):

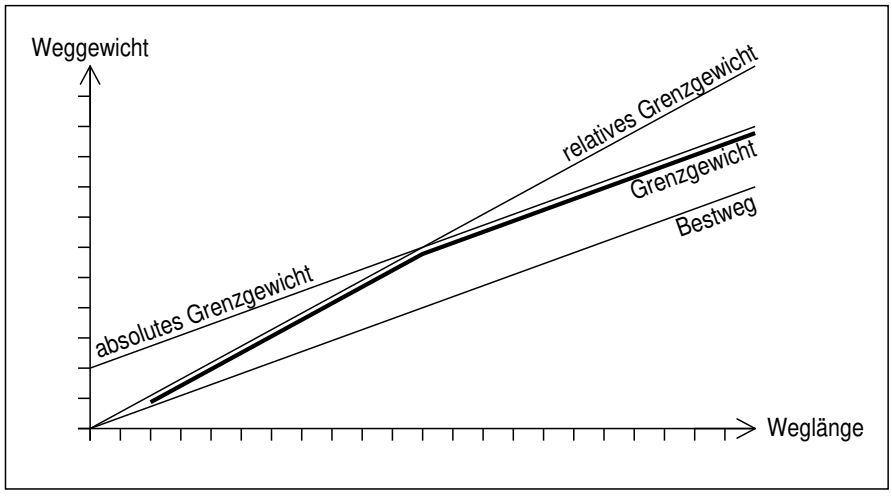


Abb. 2.4: absolute und relative Grenzen

Das absolute Grenzgewicht verhindert, daß bei immer größerem Gewicht des Bestweges, es immer mehr gültige Wege gibt, wie das bei einer reinen Berechnung mit dem relativem Grenzgewicht der Fall wäre.

Dieses Verfahren mit den beiden Grenzfaktoren war eine Vorgabe für meine Arbeit. Es findet auf diese Art seit längerer Zeit im Verkehrswesen Verwendung.

2.3 Ablauf Mehrwegumlegung

Auch wenn damit schon ein kleines Stück vorgegriffen wird, zeigt Abbildung 2.5 aus Gründen der Übersicht, den groben Ablauf einer Mehrwegumlegung. Die Details der einzelnen Schritte werden in den weiteren Kapiteln ausführlich erläutert.

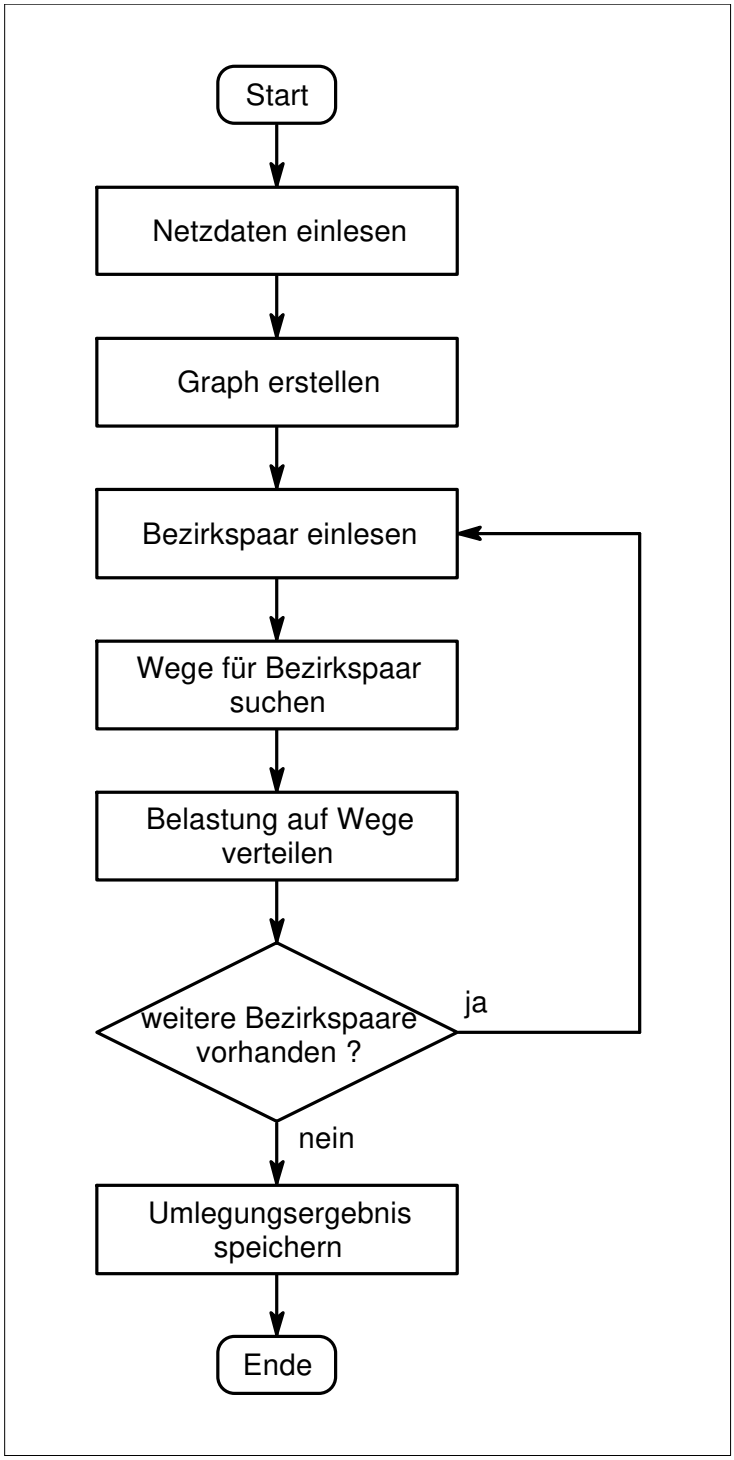


Abb. 2.5: Ablauf einer Mehrwegumlegung

3. Aufbau des Graphen

In diesem Kapitel wird beschrieben, wie die Abbildung eines Verkehrsnetzes auf einen gerichteten Digraphen vorgenommen wird und wie die Ergebniswege im Graphen dann wieder als Verkehrswege zu interpretieren sind.

3.1 Definition des Graphen

Es folgt die formale Definition des Graphen, auf den das Verkehrsnetz abgebildet werden soll.

DEF 1 Ein **gerichteter, endlicher Graph** $\mathbf{G = (V, E)}$ (*englisch: digraph*) besteht aus

(i) einer endlichen Menge $\mathbf{V} = \{1, 2, ..., |V|\}$ von **Knoten** und

(ii) einer endlichen Menge $\mathbf{E} \subseteq V \times V$ von **gerichteten Kanten**.

Ein Paar **(v, u)** $\in E$ heißt **Kante** von v nach u. Man nennt v den **Anfangs**- und u den **Endknoten** der Kante (v, u); v und u heißen benachbart oder adjazent. Da E eine Menge ist, kann von einem Knoten v zu einem anderen Knoten u höchstens eine Kante existieren (es sind keine parallelen Kanten erlaubt).
Knoten werden üblicherweise als Punkte und Kanten als Verbindungslinien zwischen zwei Punkten mit einer Pfeilspitze am Endknoten dargestellt.

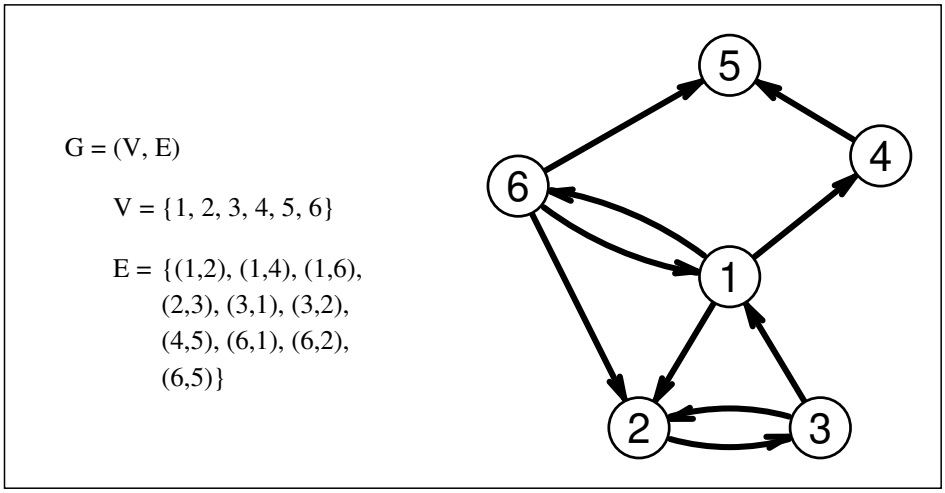


Abb. 3.1: Beispiel eines gerichteten Digraphen und seiner graphischen Darstellung

Der **Eingangsgrad indeg(v)** eines Knotens v ist die Anzahl der in v einmündenden Kanten:

$$\text{indeg}(v) = |\{ u \mid (u, v) \in E \}|$$

Der **Ausgangsgrad outdeg(v)** ist demnach die Anzahl, der von v ausgehenden Kanten:

$$\text{outdeg}(v) = |\{ u \mid (v, u) \in E \}|$$

Im Beispielgraphen aus Abbildung 3.1 gilt z.B.: indeg(1) = 2 und outdeg(1) = 3.

DEF 2 Ein **Weg p von u nach v**, wobei u, v $\in V$, ist eine Folge $(v_0, v_1, ..., v_k)$ von Knoten mit $v_0 = u$, $v_k = v$ und $(v_i, v_{i+1}) \in E$ für $0 \le i < k$.
Für jedes v $\in V$ gibt es einen trivialen Weg von v nach v mit Länge 0.
Ein Weg heißt **schleifenfrei**, wenn kein Knoten mehrfach besucht wird, d.h., wenn gilt:

$$\forall_{i \neq j} \; v_i \neq v_j$$

In Abbildung 3.1 ist der Weg (1, 6, 5) einfach und der Weg (1, 2, 3, 1, 6, 5) nicht einfach.

Ein **Zyklus** ist ein Weg, der am Ausgangsknoten wieder endet, also ein Weg von v nach v.
Im weiteren werden triviale Wege und triviale Zyklen, die nur aus einem Knoten und keiner Kante bestehen, nicht weiter betrachtet.

Ein Digraph heißt **zyklenfrei** oder **azyklisch**, wenn er keinen Zyklus enthält. Der Beispielgraph aus Abbildung 3.1 ist demnach nicht zyklenfrei: Er enthält die Zyklen (1, 2, 3) und (1, 6, 2, 3).
Für das Auffinden von Alternativwegen ist noch wichtig:
Zwei Wege $p=(v_0, v_1, ..., v_k)$ und $q=(u_0, u_1, ..., u_m)$ sind voneinander **verschieden**, wenn sie eine unterschiedliche Anzahl von Knoten haben oder (bei gleicher Knotenanzahl), wenn beide Knotenfolgen in mindestens einem Knoten nicht übereinstimmen:

$$p \neq q \;\Leftrightarrow\; \left((k \neq m) \;\vee \left(k = m \;\wedge \; \exists_{1 \le i \le k} \; v_i \neq u_i \right) \right)$$

DEF 3 Ein Digraph $G = (V, E)$ mit einer Bewertungsfunktion $\gamma : E \rightarrow |R$ heißt **bewerteter Digraph** (V, E, γ).
Für eine Kante e $\in E$ heißt γ(e) **Bewertung (Länge, Gewicht, Kosten)** der Kante e.
Das Gewicht γ(p) **eines Weges** p = $(v_0, v_1, ..., v_k)$ ist die Summe aller Kantengewichte dieses Weges:

$$\gamma(p) \;=\; \sum_{i=0}^{k-1} \gamma\left((v_i,\, v_{i+1})\right)$$

Die **Entfernung d(u, v)** von einem Knoten u zu einem Knoten v ist definiert als:

$$d(u,v) \;=\; \begin{cases} \min\{\gamma(p) \mid p \text{ ist Weg von } u \text{ nach } v\} & \text{falls ein } p \text{ existiert} \\ \infty & \text{sonst} \end{cases}$$

Gilt für einen Weg p von u nach v: γ(p) = d(u, v), so heißt p **kürzester Weg** von u nach v.
Wenn im weiteren vom Bestweg gesprochen wird, so ist damit immer der kürzeste Weg gemeint.

Ist keine Länge einer Kante negativ, also $\gamma : E \rightarrow |R_0^+$, so heißt der bewertete Digraph $G = (V, E, \gamma)$ **Distanzgraph**.

Diese formalen Definitionen orientieren sich an [Ottmann 93].
Wenn künftig von einem Graphen die Rede ist, so ist damit immer ein gerichteter, endlicher und positiv bewerteter Graph, also ein Distanzgraph, gemeint.

Wie in Kapitel 2 erwähnt, wurde dieser recht einfache Graph verwendet, um eine möglichst große Zahl von Algorithmen für die Wegsuche in diesem Graphen untersuchen zu können. Intuitiv würde man jede Haltestelle des Netzes auf einen Knoten und jeden Linienabschnitt auf eine Kante des Graphen abbilden. Warum dies nicht möglich ist und welche Probleme dabei auftreten, wird in Abschnitt 3.2 geschildert.

3.2 Abbildung des Netzes auf den Graphen

3.2.1 Problematische Eigenheiten von Verkehrsnetzen

Es gibt in den normalen Verkehrsnetzen einige Konstrukte, die eine direkte Abbildung des Netzes auf einen Distanzgraphen erschweren. Dies sind im einzelnen:

Mehrere parallelverlaufende Linien
Es ist möglich, daß von einer Haltestelle A zu einer anderen Haltestelle B mehrere Linien verlaufen. Zum Beispiel zwei Buslinien und eine S-Bahnlinie wie in Abbildung 3.2:

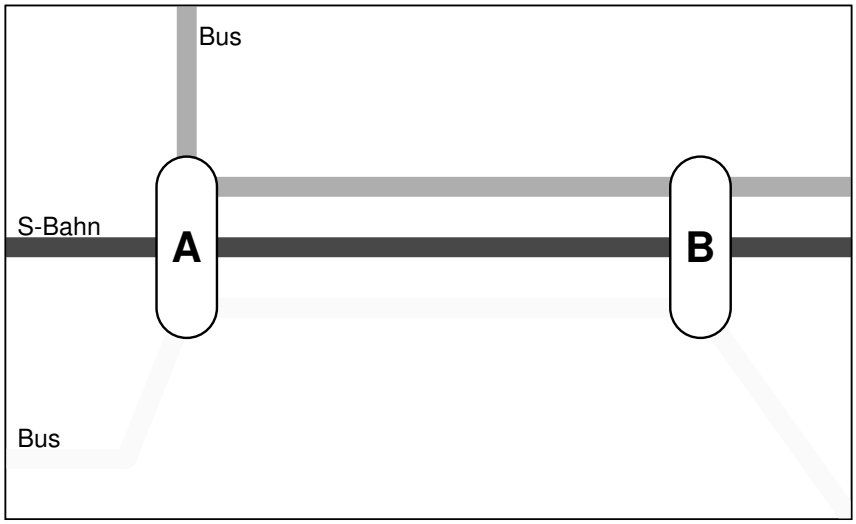


Abb. 3.2: Beispiel für parallelverlaufende Linien

Dies ist kein Parallelverkehr im engeren Sinne (siehe Abschnitt 1.4), aber in den dichten Kernzonen der Netze sehr häufig anzutreffen.
Würde nun jede Haltestelle auf einen Knoten und jeder Linienabschnitt auf eine Kante des Graphen abgebildet, so wäre die Gesamtheit aller Kanten keine Menge, da es von einem Knoten zu einem anderen Knoten mehrere Kanten geben könnte. Dies würde jedoch der Definition des Distanzgraphen nicht entsprechen, da dieser eine echte Menge für alle Kanten voraussetzt.
Auf der anderen Seite ist es nicht sinnvoll, alle Linien die von A nach B führen auf eine Kante abzubilden, da jede Linie andere Fahr- und Bedienzeiten hat und somit jeweils ein anderes Gewicht für die Kante ergeben würde.

Einsteige- und Umsteigewartezeiten
Jeder Fahrgast muß normalerweise an einer Haltestelle eine kleine Zeit auf die Linie warten, in die er einsteigen möchte. Dies wird als Einsteigewartezeit bezeichnet. Auch wenn er an einer Haltestelle in eine andere Linie umsteigen möchte, muß er, obwohl die Linien meist aufeinander abgestimmt verkehren, üblicherweise etwas warten (die Umsteigewartezeit).
Diese Wartezeiten sind zwei Komponenten, die bei der Bestimmung des Gewichts eines Weges eine Rolle spielen (dazu mehr in Abschnitt 3.3). Der Entstehungsort dieser Gewichte sind die Haltestellen. Im Distanzgraphen ist es aber nach Definition nur möglich, den Kanten, nicht aber den Knoten, Gewichte zuzuweisen.
Eine sinnvolle Verteilung der Wartezeitgewichte auf die Kanten (Linien) existiert aus Sicht des Verkehrswesens nicht.

Die oben genannten Gründe machen eine direkte Abbildung der Haltestellen auf die Knoten und der Linienabschnitte auf die Kanten des Graphen unmögliche. Daher wird eine andere Abbildung verwendet, die als „Knotensplittung“ bezeichnet wird und im nächsten Abschnitt beschrieben wird.

3.2.2 Knotensplittung

Aus den im vorherigen Abschnitt genannten Gründen, ist es nicht möglich, jede Haltestelle auf einen Knoten und jeden Linienabschnitt auf eine Kante im Graphen abzubilden.
Daher wird jede Haltestelle nicht auf einen, sondern auf mehrere Knoten des Graphen abgebildet. Genauer: Für jede Linie, die die entsprechende Haltestelle benutzt, wird im Graphen ein Knoten angelegt. Das heißt, sollte eine Haltestelle im Verkehrsnetz von fünf Linien bedient werden, so wird sie im Graphen von fünf Knoten repräsentiert:

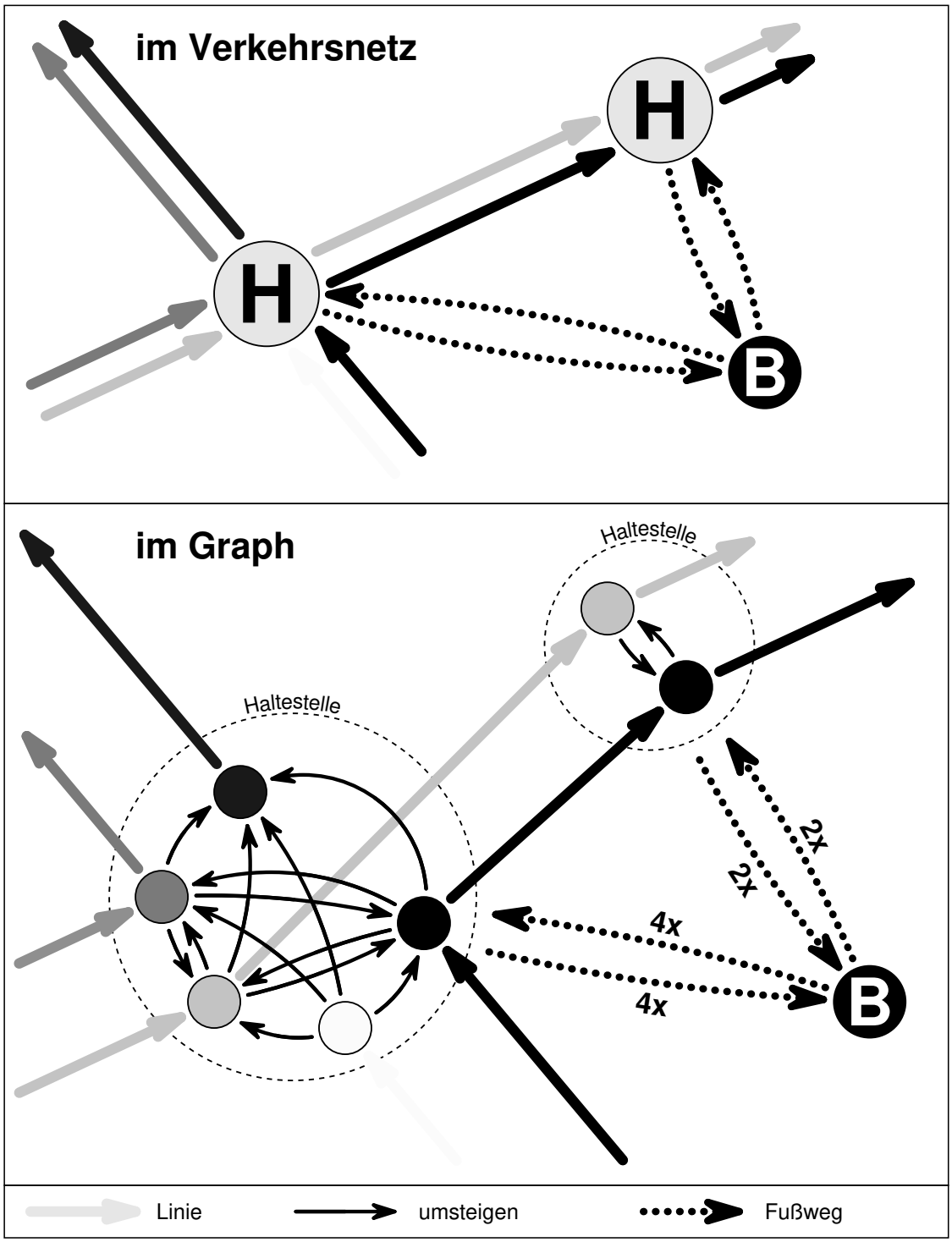


Abb. 3.3: Repräsentation von Haltestellen im Graphen

Diese Aufsplittung der Haltestelle in mehrere Graphenknoten hat den Vorteil, daß zum einen die Gesamtheit aller Kanten eine Menge ist, wie es der Distanzgraph verlangt. Zum anderen können, zwischen den Knoten, die zusammen eine Haltestelle repräsentieren, zusätzliche Umsteigekanten eingefügt werden. Durch eine solche Kante wird ein entsprechendes Umsteigen von einer auf eine andere Linie an dieser Haltestelle modelliert. Auf diese Kanten könnte dann auch das Gewicht der Einsteige- und Umsteigewartezeit gelegt werden.
Abbildung 3.3 läßt erkennen, daß sich die Anzahl der Kanten mit zunehmender Zahl von Linien, die eine Haltestelle benutzen, erhöht. Es ist nun nicht notwendig, von jedem Knoten einer Haltestelle zu allen anderen Knoten derselben Haltestelle Umsteigekanten zu bilden. Es reicht aus, wenn nur von Knoten, bei denen eine Linien ankommt, zu Knoten bei denen eine Linie abfährt, Umsteigekanten eingerichtet werden. Es ist aber auch ersichtlich, daß sich die Anzahl der Fußwege mit jeder weiteren Linie an einer Haltestelle erhöht. Auch hier ist es ausreichend, wenn Fußwege nur von einem Bezirksknoten zu Haltestellenknoten, an denen eine Linie abfährt, und umgekehrt, von Haltestellenknoten, an denen eine Linie ankommt, zurück zum Bezirksknoten, gebildet werden.
Die Unterscheidung der Kanten in Linienkanten, Umsteigekanten und Fußwegkanten soll nicht darüber hinwegtäuschen, daß im Graphen keine Unterscheidung mehr stattfindet. Hier gibt es nur noch Knoten und Kanten einer Sorte. Die Bezeichnungen Linienkanten etc. soll nur darlegen, „woher“ dieses Kante kommt und was sie repräsentiert. Für den Algorithmus zur Wegsuche später spielt dies keine Rolle.
Nachteilig ist, daß diese Umsteigekanten auch ein mehrmaliges Umsteigen innerhalb einer Haltestelle zulassen. Da dies aber nicht der menschlichen Logik entspricht und deshalb Fahrgäste nur Wege verwenden, die an einer Haltestelle maximal einmal umsteigen, muß entweder der Algorithmus zur Wegsuche oder aber ein anschließender Check die Wege mit Mehrfachumsteigen herausfiltern. Mehr dazu im Abschnitt 4.12: Wegecheck.

Eine genaue, schrittweise Beschreibung, wie das Verkehrsnetz auf einen Distanzgraphen zu übertragen ist, wird erst in Abschnitt 3.2.4 beschrieben.

Mit dieser Knotensplittung sind die in Abschnitt 3.2.1 beschriebenen Eigenheiten des Verkehrsnetzes abbildbar. Es gibt jedoch noch einen Punkt, der die Knotensplittung noch einmal um einige Knoten erhöht: der Parallelverkehr.

3.2.3 Parallelverkehr

Fahren von einer Haltestelle zu einer anderen Haltestelle mehrere Linien des gleichen Verkehrsmittels (der Parallelverkehr im eigentlichen Sinn), dann gibt es an diesen Haltestellen mit der bisherigen Knotensplittung ein Problem bei der Einsteige- und Umsteigewartezeit:

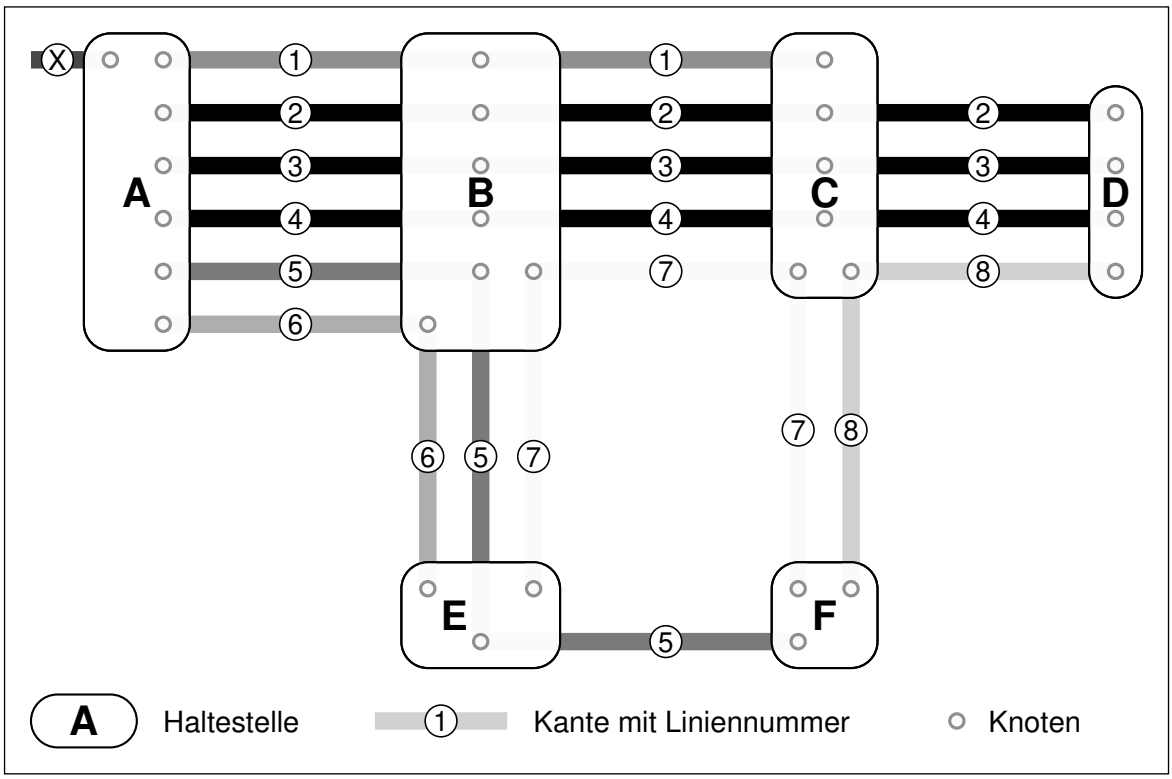


Abb. 3.4: Parallelverkehr

Abbildung 3.4 stellt einen Ausschnitt aus einem Verkehrsnetz bzw. Graphen dar, wobei auf die Darstellung von Umsteigekanten, Bezirkspunkten und Fußwegen aus Gründen der Übersicht verzichtet wurde. Alle Kanten haben eine Ziffer, die die Liniennummer bezeichnet, die sie im Graphen repräsentieren. Alle Linien sollen der gleichen Verkehrsmittelart angehören (z.B. alles S-Bahnlinien).

Beispiel:
In diesem Netz fährt ein Fahrgast mit einer beliebigen Linie X zur Haltestelle A. Diese Linie endet in A. Er möchte zur Haltestelle C weiterfahren. Dafür kann er unter anderem auch die direkt nach C führenden Linien 1, 2, 3 oder 4 verwenden. Das heißt, in A existieren 4 Umsteigekanten: jeweils von Linie X auf jede der Linien 1, 2, 3 oder 4 (Die Umsteigekanten sind nicht eingezeichnet). Diese Umsteigekanten erhalten als Gewicht die Umsteigewartezeit, das heißt, die mittlere Zeit, die der Fahrgast an dieser Haltestelle auf die neue Linie warten muß. Die Gewichte von Kanten und die Wartezeiten werden in Abschnitt 3.3 näher erläutert. Es soll hier nur zum besseren Verständnis gesagt sein: Die Umsteigewartezeit ist ungefähr die halbe Taktzeit der Linie. Das heißt, eine Linie, die einen Takt von 30 Minuten hat, kommt alle 30 Minuten an der Haltestelle vorbei und ein Fahrgast muß im Durchschnitt etwa 15 Minuten auf diese Linie warten.
Wenn alle Linien in Abbildung 3.4 eine Taktzeit von 30 Minuten haben, erhält also jeder der vier genannten Umsteigekanten ein Gewicht von „15 Minuten“ für die Umsteigewartezeit.
Die bisherige Modellierung des Netzes kann damit nur folgendes Verhalten nachbilden: Der Fahrgast kommt mit Linie X an Haltestelle A an und muß sich dort für eine der Linien 1, 2, 3 oder 4 entscheiden (Wahl der Umsteigekante). Nach seiner Entscheidung muß er die Umsteigewartezeit für seine gewählte Linie abwarten und er fährt dann mit dieser Linie nach B und C weiter.
Da der Fahrgast aber in A auf jeden Fall umsteigen muß und nach C möchte, und aller vier Linien von der gleichen (**!**) Verkehrsmittelart gebildet werden und somit gleich schnell sind, könnte es ihm in A egal sein, welche Linie er nach C verwendet. Das heißt, er wird die Linie verwenden, die als nächstes in A abfährt.

In diesem Fall verringert sich die Umsteigewartezeit erheblich: Da jede Linie zwei mal in der Stunde (Taktzeit von 30 Minuten) die Haltestelle bedient, und es vier mögliche Linien gibt, wird die Haltestelle demnach acht mal in der Stunde von einer günstigen Linie bedient. Dies bedeutet, daß alle 7,5 Minuten eine für diesen Fall günstige Linie in A abfährt und der Fahrgast in A in Wirklichkeit nur noch eine Umsteigewartezeit von 3,25 Minuten abwarten muß.

Um diesen Fall bei der Wegsuche zu berücksichtigen, gibt es zwei Möglichkeiten. Zum einen könnte der Wegsuchealgorithmus bei jedem Schritt jeweils prüfen, ob mehrere Linien parallel genutzt werden können und dann das Umsteigegewicht (= Umsteigewartezeit) entsprechend verkleinern.
Damit der Wegsuchealgorithmus nicht mit diesen aufwendigen Untersuchungen zusätzlich belastet wird, habe ich mich für die andere Lösung entschieden: Alle weiteren Möglichkeiten durch parallel genutzte Linien müssen im Graphen durch zusätzliche Kanten modelliert werden. Das heißt im Falle des obigen Beispiels: Für die Linien 1, 2, 3 und 4 wird jeweils von A nach B und von B nach C eine neue Kante eingefügt. Diese Kanten repräsentieren eine neue Linie mit der Bezeichnung (1234). Sie hat die gleichen Streckenlängen und Fahrzeiten wie die anderen Linien, mit einer Ausnahme: Die Umsteigekante innerhalb der Haltestelle A von Linie X auf diese Linie 1234 hat das entsprechend kleinere Umsteigegewicht.
Eine solche neue Linie, die mehrere, parallel genutzte Linien repräsentiert, wird im weiteren eine Linienkombination (kurz **LINK**) genannt. Die Taktrate dieser neuen Linie läßt sich wie folgt bestimmen:

$$\text{Neuer Takt} = \frac{1}{\sum_i \frac{1}{\text{Takt}_i}}$$ über alle Linien i, die parallel genutzt werden.

Die Art und Anzahl dieser neuen Kanten von Linienkombinationen hängt davon ab, von wo nach wo der Fahrgast fahren möchte. Daher müssen in den Graphen alle Linienkombinationen eingefügt werden, die auf irgend einem Weg möglich sind. Hierzu wird wie folgt vorgegangen:
Für jede Strecke (Abschnitt von einer Haltestelle zu einer benachbarten), und für jedes Verkehrsmittel, wird zunächst die größtmögliche Linienkombination aller Parallelverkehrslinien ermittelt. Diese wird als Hauptlinienkombination bezeichnet. In Abbildung 3.4 besteht die Hauptlinienkombination für die Strecke A nach B aus den Linien 1, 2, 3, 4, 5 und 6.
Besteht diese Hauptlinienkombination aus mehr als einer Linie, dann wird sie als neue Kante in den Graphen eingefügt und zusätzlich in einer globale Liste, der sogenannten LINK-Liste, vermerkt. Nachdem dies für alle Strecken und alle Verkehrsmittel durchgeführt wurde, wird in einem zweiten Durchgang erneut für alle Strecken und alle Verkehrsmittel geprüft, ob es eine echte Teilmenge der Hauptlinienkombination dieser Strecke und jeder Hauptlinienkombination der LINK-Liste gibt. Wenn es solche Teilkombinationen zwischen zwei Hauptlinienkombinationen gibt, wird diejenige Teilmenge mit der größten Linienzahl ebenfalls als neue Kante in den Graphen eingefügt.
Dieses Verfahren sichert ab, daß auf eine Strecke auch Linienkombinationen verwendet werden können, die sich auf benachbarten Stecken fortsetzten.

Wendet man diese Vorgehensweise auf das Beispiel in Abbildung 3.4 an, so erhält man die Abbildung 3.5 auf der nächsten Seite:

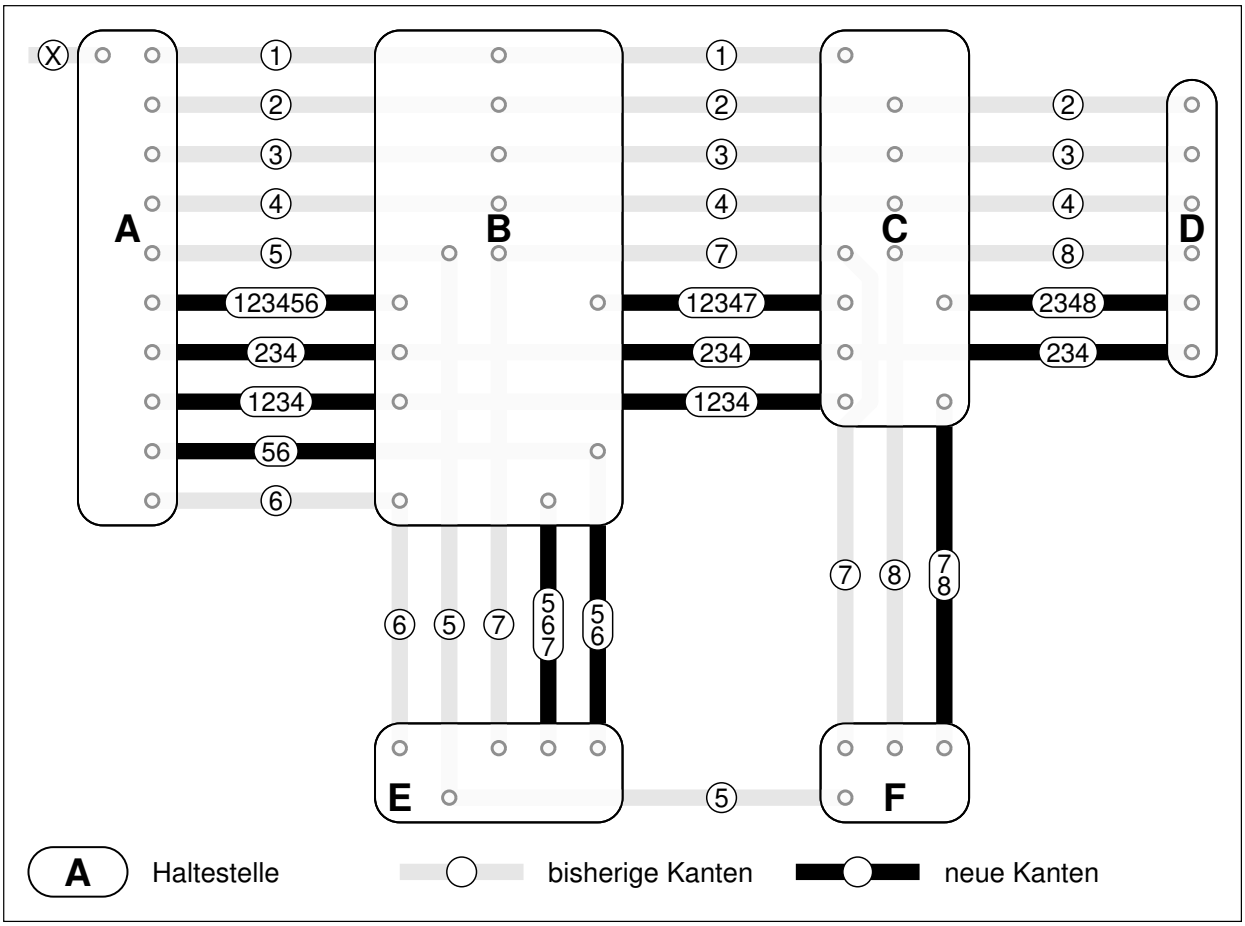


Abb. 3.5: zusätzliche Parallelkanten

In diesem Fall ergeben sich folgende Hauptlinienkombinationen:

Von	Nach	Hauptlinienkombination
A	B	123456
B	C	12347
C	D	2348
B	E	567
C	F	78
E	F	5*

* Diese Linienkombination besteht nur aus einer Linie und ist damit keine eigentliche Hauptlinienkombination. Sie wird auch nicht in der globalen Liste gespeichert.

In Abbildung 3.5 ist erkennbar, daß durch den zweiten Durchgang aller Strecken zum Beispiel auch die Linienkombination 234 in die Strecken A-B, B-C und C-D eingefügt wird.

Diese neuen Kanten erhalten ebenfalls, nach der in Abschnitt 3.2.2 beschriebenen Weise, Umsteigekanten, jedoch mit einer Ausnahme: Es wird niemals von einer Linienkombination auf eine andere Linienkombination umgestiegen, wenn diese eine echte Teilmenge der vorherigen ist. Denn in diesem Fall könnte der Fahrgast ohne Zeitverlust ebenfalls mit der bisherigen Linienkombination zum gleichen Ziel weiterfahren.

3.2.4 Vorgehensweise der Abbildung

In diesem Abschnitt wird die gesamte Vorgehensweise bei der Abbildung des Verkehrsnetzes auf den Graphen noch einmal zusammengefaßt und beschreiben. Doch vorher soll noch definiert werden, was in dieser Arbeit unter einem LINK verstanden wird.

DEF 4 Ein **LINK** ist eine Menge aus Liniennummern, z.B. {1,2,5,7}
Jedem Knoten und jeder Kante im Graphen ist ein LINK zugeordnet.
Er soll im Graphen das repräsentieren, wofür im Verkehrsnetz die Liniennummer verwendet wird: Die Identifikation, welche Linien etwas befahren oder benutzen. Da im Graphen eine Kante nur eine einfache Kante ist und nichts mehr mit dem Verkehrsnetz zu tun hat, ermöglicht es zum Beispiel der LINK, von dieser Kante wieder zurück in das Verkehrsnetz zu kommen. Die LINKs werden nicht für die Wegsuche benötigt.
In den meisten Fällen bestehen die im Graphen vorhandenen LINKs aus nur einer Linie. Wenn eine Kante im Graphen den LINK {5} hat, so bedeutet das, daß diese Kante einen Linienabschnitt der Linie 5 im Verkehrsnetz verkörpert.
Erst wenn zusätzliche Kanten für parallel genutzte Linien in den Graphen eingefügt werden, gibt es im Graphen auch Kanten mit LINKs, die aus mehreren Liniennummern bestehen. Dies ist auch der Grund für die Einführung der LINKs, denn in diesem Fall würde eine einfache Liniennummer nicht ausreichen, da diese immer nur eine einzelne Linie bezeichnen kann.
Die globale LINK-Liste ist wiederum nur eine Menge von LINKs. Allerdings darf sie nur solche LINKs enthalten, die aus mindestens zwei Liniennummern bestehen.

Zusätzlich wird für jeden Knoten des Graphen eine weitere Information gespeichert: Ob dieser Knoten einen Bezirkspunkt oder einen Teil einer Haltestelle des Verkehrsnetzes repräsentiert. Diese Information wird später für die „Rückkehr“ vom Graphen zum Verkehrsnetz benötigt.

Die Ausgangsdaten für die Abbildung sind:

- Ein Verkehrsnetz mit Haltestellen, Bezirken und Linien
- Ein leerer Distanzgraph $G=(V, E, \gamma)$ mit $|V| = 0$ und $|E| = 0$
- Eine leere LINK-Liste

Vorgehensweise der Abbildung:

1. Schritt Für jeden Bezirkspunkt wird im Graph ein Knotenpunkt gebildet. Diese Knoten erhalten einen leeren LINK { } und die Information, daß sie diesen Bezirkspunkt repräsentieren.

2. Schritt Für jede Haltestelle wird eine Anzahl von Knoten gebildet, die diese Haltestelle zusammen im Graphen repräsentieren. Diese Anzahl von Knoten besteht aus je einem Knoten für jede Linie, die an dieser Haltestelle hält. Jeder dieser Knoten erhält zusätzlich die Information, zu welcher Haltestelle des Netzes er gehört und einen LINK {i} für die entsprechende Linie i, für die er gebildet wurde. Der Knoten ist damit die „Anlaufstelle“ für Kanten der Linie i an dieser Haltestelle. Dies bedeutet, daß an diesem Knoten nur Kanten der Linie i, Fußwegkanten oder Umsteigekanten beginnen oder enden dürfen, jedoch niemals Kanten einer Linie $j \neq i$.

3. Schritt Für jeden Teilabschnitt einer Linie und Richtung wird zwischen den entsprechenden Knoten der beteiligten Haltestellen eine Kante eingefügt. Wenn zum Beispiel ein Linienabschnitt der Linie k von Haltestelle A zu Haltestelle B führt, so wird von einem der Knoten, die die Haltestelle A im Graphen repräsentieren, zu einem der Knoten von B eine Kante eingefügt, und zwar zwischen den beiden Knoten, die den LINK {k} haben.

4. Schritt In diesem Schritt wird für jede Verkehrsmittelart und für jede Strecke (von Haltestelle A nach B, beliebig) des Netzes die Hauptlinienkombination bestimmt. Enthält diese Kombination mindestens zwei Linien, dann wird für die beiden Haltestellen der Strecke jeweils ein neuer Graphenknoten wie in Schritt 2 gebildet. Diese Knoten erhalten als LINK die ermittelte Hauptlinienkombination. Außerdem wird vom neuen Knoten von A zum neuen Knoten von B eine Linienkante mit dieser Hauptlinienkombination eingefügt.
Zusätzlich wird diese Lininenkombination auch in die globale LINK-Liste aufgenommen.

5. Schritt Wiederum wird für jede Verkehrsmittelart und für jede Strecke des Netzes die Hauptlinienkombination bestimmt. Enthält diese mindestens zwei Linien, wird sie mit allen anderen LINKs der LINK-Liste verglichen. Gibt es eine Linienkombination mit mindestens zwei Linien, die sowohl von der momentanen Hauptlinienkombination, als auch von einem LINK der LINK-Liste eine **echte** Teilmenge darstellt, wird für diese Linienkombination ebenfalls wie in Schritt 4 verfahren (zwei neue Knoten und eine Kante mit diesem LINK einfügen).

6. Schritt Wenn für zwei beliebige Knoten u, $v \in V$ ($u \neq v$) gilt:
- u repräsentiert einen Bezirkspunkt B
- und v repräsentiert eine Haltestelle H
- und der Bezirkspunkt B ist an die Haltestelle H angebunden
- und outdeg(v) > 0,

oder es gilt für u und v:
- u repräsentiert eine Haltestelle H
- und v repräsentiert einen Bezirkspunkt B
- und der Bezirkspunkt B ist an die Haltestelle H angebunden
- und indeg(u) > 0,

dann wird von u nach v eine Fußwegkante mit leerem LINK { } eingefügt.

7. Schritt Wenn für zwei beliebige Knoten $u,v \in V$ ($u \neq v$) gilt:
- u und v repräsentieren dieselbe Haltestelle
- und indeg(u) > 0 und outdeg(v) > 0
- und es existiert kein LINK x mit ($x \subset$ LINK(u) und $x \subset$ LINK(v)),

dann wird von u nach v eine Umsteigekante mit leerem LINK { } eingefügt.

Nun ist das Verkehrsnetz vollständig auf den Distanzgraphen abgebildet. Es bleibt noch zu klären welchen Kante welches Gewicht erhält:

3.3 Gewichtsfunktion

In diesem Abschnitt wird beschrieben, welche Gewichte im Verkehrsnetz auftreten und wie die Gewichte der einzelnen Kanten des Graphen berechnet werden.
Ein Teil der Aufgabe dieser Arbeit war es auch zu untersuchen, welche Gewichte des Verkehrsnetzes für die Wegsuche geeignet sind und berücksichtigt werden können. Es hat sich gezeigt, daß nicht alle vorgegebenen Gewichte hierfür geeignet waren.
Es sei an dieser Stelle ein kleiner Vorgriff auf Kapitel 4: „Algorithmen für die Wegsuche“ gestattet. Alle Algorithmen, die untersucht wurden, haben für die Wegsuche eines vorausgesetzt:

Die Greedy-Bedingung: **lokales Optimum = globales Optimum**

Dies bedeutet, daß der Algorithmus, wenn er auf der Suche nach dem Bestweg mit zwei Alternativen an den gleichen Punkt kommt, hier eine (lokale) Entscheidung treffen muß, welche Alternative im Moment die günstigere ist und diese Alternative wird dann weiterverfolgt. Die Greedy-Bedingung fordert, daß diese Entscheidung auch dann noch Gültigkeit besitzt, wenn der Weg zu Ende berechnet wurde. Zeigt sich dann, daß diese lokale Entscheidung global betrachtet falsch gewesen ist, ist die Greedy-Bedingung verletzt und der Algorithmus kann auf einem solchen Graphen nicht eingesetzt werden.

Aus der Greedy-Bedingung folgt nun, daß alle Kantengewichte statisch sein müssen, das heißt vor der Berechnung eines Weges bereits feststehen sollen. Ein dynamisches Gewicht, das seine Größe nach einer Berücksichtigung in einer Teilberechnung des Weges seinen Wert ändert, könnte die Greedy-Bedingung verletzen und ist damit nicht zulässig.

Fahrtkosten
Ein Gewicht, das sich in Verkehrsnetzen findet und die eben beschriebene, dynamische Wertänderung besitzt, sind die Fahrtkosten. Nun ist diese nicht immer der Fall, denn in den heutigen Verkehrsnetzen kommen verschiedene Tarifsysteme zum Einsatz:

Einheitstarif
Jede Fahrt kostet den gleichen Betrag. Dieses Gewicht ist statisch.

Streckentarif
Hier wird für jeden zurückgelegten Kilometer ein bestimmter Preis erhoben. Auch dieses Gewicht ist statisch.

Zonentarif
Dieses Tarifsystem ist etwas komplexer und findet zum Beispiel im Großraum Stuttgart Verwendung. Hierbei wird das Netz in Zonen eingeteilt und der Fahrpreis richtet sich nach der Anzahl der Zonen, die der Fahrgast auf seiner Fahrt benutzt. Diese gestaffelten Zonentarife sind leider meist nicht linear, das heißt, wenn die Person zuerst eine Fahrt über drei Zonen unternimmt und dann nochmals eine Fahrt mit zwei Zonen tätigt, ist ein höheren Fahrpreis zu entrichten, als wenn die Person eine Fahrt über fünf Zonen unternommen hätte.
Dieses Zonentarifsystem hat aber genau die Eigenheit, daß der Fahrpreis eines einzelnen Linienabschnitts von der Länge des Gesamtweges abhängt, der benutzt wird und somit dynamisch ist. Es war mir in Zusammenarbeit mit meinen Betreuern nicht möglich, eine Modellierung dieses Zonentarifes zu finden, die diese auf statische Gewichte abgebildet hätte.

Es kam daher der Streckentarif, bei der Gewichtsbildung im Programm, zum Einsatz.

Maximale Taktzeit
Darunter ist die größte Taktzeit, die eine auf dem Weg verwendete Linie besitzt, zu verstehen. Hier wird sehr schnell klar, daß diese Information erst dann mit Gewißheit richtig berechnet und als Gewicht eingesetzt werden kann, wenn der ganze Weg bekannt ist. Es ist nicht möglich nach einer Teilberechnung des Weges schon mit absoluter Korrektheit zu sagen, welches die maximale Taktzeit des gesamten Weges sein wird, da der noch zu berechnende Teil des Weges ja noch unbekannt ist. Trifft man daher eine Entscheidung auf Grund der bisher bekannten Informationen, kann sich diese am Ende der Berechnung als falsch herausstellen.
Daher kam diese Art von Gewicht ebenfalls nicht zum Einsatz bei der Implementation.

Die folgenden Gewichte sind aber alle statisch und wurden zur Wegsuche eingesetzt:

Fußwege
Dies sind die Wege die zu Fuß von den Bezirkspunkten zu den nächsten (angebundenen) Haltestellen, an denen die Fahrt mit den öffentlichen Verkehrsmitteln begonnen wird, zurückgelegt werden. Dieser Weg wird in Metern gemessen und kann mit einer durchschnittlichen Fußgängergeschwindigkeit in eine Zeitdauer umgerechnet werden.

Einsteigewartezeit
Die Einsteigewartezeit ist die Zeit, die der Fahrgast warten muß, wenn er zu Fuß an die erste Haltestelle seiner Fahrt kommt und dort auf sein erstes Verkehrsmittel warten muß. Hierfür wird die halbe Taktzeit angesetzt, das heißt, wenn von dieser Haltestelle alle zehn Minuten ein Bus abfährt, so muß der Fahrgast im Durchschnitt fünf Minuten auf einen Bus warten.
Aus Untersuchungen heraus hat sich gezeigt, daß ein Fahrgast bei einer Taktzeit von mehr als 13 Minuten nicht mehr unüberlegt zu dieser Haltestelle läuft, sondern, daß er zu einem bestimmten Zeitpunkt gezielt auf die Haltestelle losläuft, damit er nicht zu lange warten muß. Beträgt die Taktzeit weniger als 13 Minuten, macht er sich im allgemeinen keine Gedanken über seine Wartezeit an der Haltestelle und läuft einfach zu der Haltestelle, so wie er dafür Zeit findet. Diesen Grenzwert von 13 Minuten nennt man maximale Einsteigewartezeit.
Damit ergibt sich : Einsteigewartezeit = Minimum{ Taktzeit, maximale Einsteigewartezeit } x 0,5

Fahrzeit
Dies ist die Zeit, die ein Verkehrsmittel braucht um eine bestimmte Strecke ohne Halt zurückzulegen. Der Wert ist bereits für jeden Linienabschnitt in den Netzdaten vorhanden.

Fahrtkosten
Wie schon beschrieben, kommt in diesem Fall nur der Streckentarif zum Einsatz. Er berechnet sich aus den zurückgelegten Kilometern (diese sind ebenfalls für alle Linienabschnitte in den Netzdaten bereits enthalten), und einem Kilometerpreis.

Umsteigewartezeit
Immer dann, wenn innerhalb einer Haltestelle von einem Verkehrsmittel auf ein anderes Verkehrsmittel umgestiegen wird, muß, wie bei der Einsteigewartezeit, auf dieses Verkehrsmittel gewartet werden. Nur gibt es in diesem Fall nicht die Möglichkeit, bei einer größeren Taktzeit von 13 Minuten, erst zu einem gezielten Zeitpunkt die Haltestelle zu betreten, da das vorherige Verkehrsmittel sich von einem einzelnen Fahrgast nicht in dieser Weise beeinflussen läßt. Das Verkehrsmittel liefert den Fahrgast eben dann an der Haltestelle ab, wenn die Linie dort ankommt. Der Fahrgast hat nicht die Möglichkeit dies zu verzögern.
Trotzdem gibt es auch hier eine maximale Umsteigewartezeit, da davon ausgegangen wird, daß diese Netze im allgemeinen von Verkehrsexperten so konzipiert werden, daß sich keine größeren Umsteigezeiten ergeben.
Damit gilt auch hier: Umsteigewartezeit = Minimum { halbe Taktzeit des nächsten Verkehrsmittels,
maximale Umsteigewartezeit } x 0,5
Der Multiplikator von 0,5 wird als Fahrplanabstimmungsfaktor bezeichnet. Dieser Wert gibt an, wie gut die Verkehrsplaner des Netzes die einzelnen Linien aufeinander abgestimmt haben, so daß der Fahrgast beim Umsteigen nicht zu lange warten muß.

Umsteigewert
Dieses Gewicht soll, zusätzlich zur Umsteigewartezeit, dafür sorgen, daß bei jedem Umsteigen, der Weg grundsätzlich um einen gewissen Betrag erschwert wird.

Schnellbahnanteil
Der Schnellbahnanteil soll ein Komfortkriterium für den Weg sein, da Schnellbahnen (S-Bahnen und Stadtbahnen) im allgemeinen eine komfortablere Ausstattung und mehr Platz bieten, als Busse oder Eisenbahnen.
Wird der Schnellbahnanteil prozentual vom gesamten Weg berücksichtigt, wird die Greedy-Bedingung nicht erfüllt, da auch hier erst der ganze Weg bekannt sein muß, um einen korrekten Wert zu errechnen. Wird der Schnellbahnanteil jedoch absolut berücksichtigt, das heißt, jeder Meter des Weges, der mit einem Schnellbahn-Verkehrsmittel zurückgelegt wird, bekommt einen Komfortbonus auf das Weggewicht zugeschlagen, können diese Werte vor Beginn der Wegsuche ermittelt und festgelegt werden. Damit wäre diese Gewichtskomponente ebenfalls statisch.
Da dieses Gewicht jedoch negativ ist, weil das Weggewicht um den Bonus verringert wird, und der Distanzgraph aber nur positive Kantengewichte zuläßt, wird der Schnellbahnanteil in einen Nichtschnellbahnanteil umgewandelt. In diesem Fall erhalten alle Linienkanten, die nicht von einer Schnellbahn befahren werden, ein zusätzliches Gewicht für jeden Meter Fahrstrecke.

Aus diesen Komponenten wird nun die Gewichtsfunktion gebildet. Wichtig in diesem Zusammenhang ist, daß man bei der Verkehrsprognose vor allem an den häufig wiederkehrenden Fahrten interessiert ist (z.B. der täglichen Fahrt mit dem Bus zum Arbeitsplatz). In diesem Fall hat sehr wohl jeder einzelne der beschriebenen Gewichtskomponenten einen Einfluß auf die Wahl des Weges.
Im Gegensatz dazu, wird bei einer Spontanfahrt, also bei einer nicht regelmäßigen Fahrt (z.B. am Wochenende in den Zoo), nicht jede Komponente in Erwägung gezogen. Der Fahrgast nimmt sich dazu nicht die Zeit. In einem solchen Fall zählen dann meist ausschließlich Fahrzeit und Fahrtkosten. Es sei aber noch einmal betont, daß die Verkehrsprognose an Spontanfahrten nicht interessiert ist. Sie machen auch nur einen geringen Prozentsatz aller Fahrten im Netz aus.
Einige der einzelnen Komponenten erhalten noch einen zusätzlichen Faktor, der den Zweck hat, daß jede Komponente die einheitliche, physikalische Einheit „Sekunde“ erhält. Damit kann das Gesamtgewicht G einer Kante (ebenfalls in Sekunden) problemlos aus den einzelnen Komponenten summiert werden.Die Angaben in eckigen Klammern [ ] auf der nächsten Seite geben die physikalische Einheit des jeweiligen Wertes an.

Gesamtgewicht G:

Das Gesamtgewicht G einer Kante besteht aus einzelnen Gewichtskomponenten $g_1,...,g_8$.
Jede Gewichtskomponente g_i wird mit einem eigenen Gewichtungsfaktor gf_i multipliziert. Über diese Gewichtungsfaktoren kann der Einfluß der einzelnen Gewichtskomponenten g_i auf das Gesamtgewicht G geregelt werden.

Verwendete Konstanten:

Bezeichn.	phys. Einheit	Bedeutung	Wert
$gf_1,\ldots,gf_8$	$[\ ]$	Gewichtungsfaktoren	unterschiedlich
FGv	$[m/s]$	Fußgängergeschwindigkeit	$1{,}2\ m/s$
FPA	$[\ ]$	Fahrplanabstimmungsfaktor	0,5
KMP	$[Pf/km]$	Kilometerpreis	$30\ Pf/km$, (je nach Netz)
NSB	$[s/km]$	Nicht-Schnellbahn-Faktor	$5\ s/km$
UMS	$[s]$	Umsteigefaktor	$30\ s$
$MaxEWZ$	$[\min]$	Maximale Einsteigewartezeit	$13\ \min$
$MaxUWZ$	$[\min]$	Maximale Umsteigewartezeit	$30\ \min$

Berechnung der Teilkomponenten g_i

Fußwege $\quad g_1 = gf_1 \cdot \frac{Weg[m]}{FGv[m/s]}$ bzw. $g_8 = gf_8 \cdot \frac{Weg[m]}{FGv[m/s]}$

Einsteigewartezeit $\quad g_2 = gf_2 \cdot \min\{Takt[\min],\ MaxEWZ\} \cdot 0{,}5 \cdot 60\,s/\min$

Fahrzeit $\quad g_3 = gf_3 \cdot Fahrzeit[s]$

Fahrtkosten $\quad g_4 = gf_4 \cdot \frac{Weg[m]}{1000\,m/km} \cdot KMP \cdot 1\,s/Pf$

Umsteigewert $\quad g_5 = gf_5 \cdot UMS$

Umsteigewartezeit $\quad g_6 = gf_6 \cdot \min\{Takt[\min],\ MaxUWZ\} \cdot FPA \cdot 60\,s/\min$

Nichtschnellbahngewicht $\quad g_7 = gf_7 \cdot \frac{Weg[m]}{1000\,m/km} \cdot NSB$

Für das Gesamtgewicht $G\ [s]$ einer Kante werden nicht immer alle Komponenten g_i addiert. Dies hängt von der Art der Kante ab:

Komponente / Kantenart	Fußweg Zugang	Einsteige-wartezeit	Fahrzeit	Fahrt-kosten	Umsteige-wert	Umsteige-wartezeit	Nicht-schnell-bahngew.	Fußweg Abgang
Fußwegkante G = (Startbezirk → 1. HS.)	g1	+ g2						
Linienkante G = (Fahrstrecke)			g3	+ g4			+ g7	
Umsteigekante G = (Innerhalb einer HS)					g5	+ g6		
Fußwegkante G = (letzte HS → Zielbezirk)								g8

Die einzelnen Kantengewichte können demnach alle, unabhängig von der Wegsuche, vorher berechnet und zusammen mit den Kanten im Graphen gespeichert werden.

Abbildung 3.6 zeigt an einem Beispiel die Berechnung der Kantengewichte:

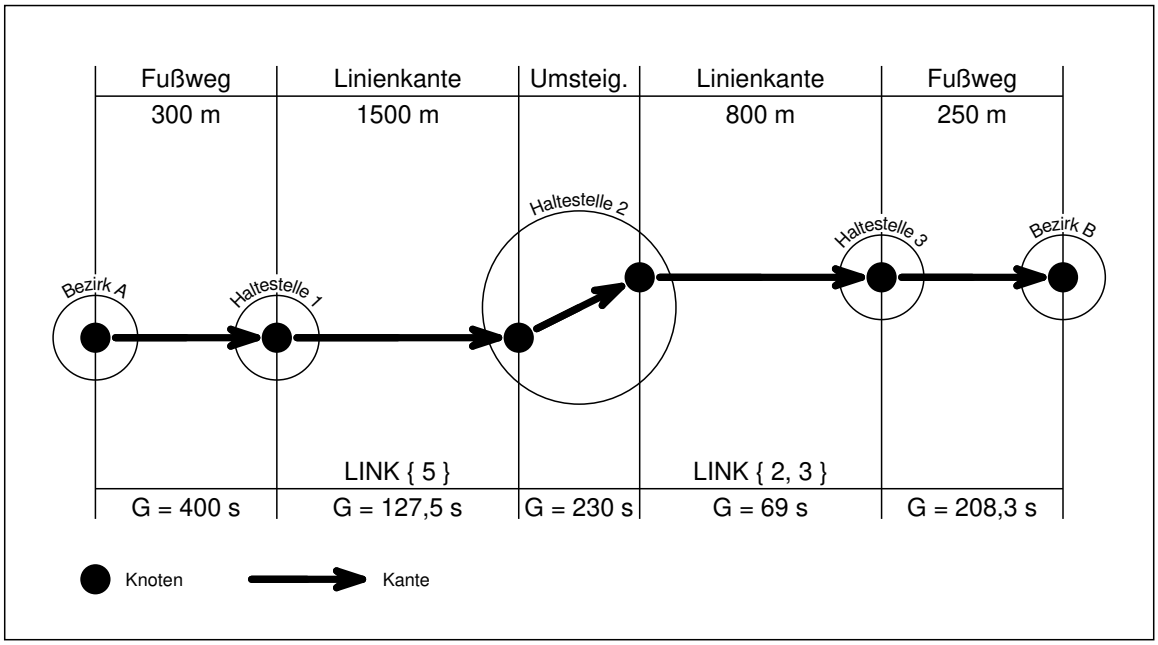


Abb. 3.6: Beispielberechnung von Kantengewichten

Zusätzliche Angaben zu Abbildung 3.6:

Linie 2	S-Bahn	Taktzeit 10 Minuten Fahrzeit von Haltestelle 2 zu Haltestelle 3: 45 Sekunden
Linie 3	S-Bahn	Taktzeit 20 Minuten Fahrzeit von Haltestelle 2 zu Haltestelle 3: 45 Sekunden
Linie 5	Buslinie	Taktzeit: 5 Minuten Fahrzeit von Haltestelle 1 zu Haltestelle 2: 75 Sekunden

Alle $\mathrm{gf_i}$ sind in auf 1,0 gesetzt.

Berechnung der einzelnen Kantengewichte:

Von	Nach	Kantengewicht	Bemerkungen
Bezirk A	HS 1	G = 300 [m] / 1,2 [m/s] + 150 [s]	g1 / Fußweg Zugang g2 / Einsteigewartezeit in Linie 5
HS 1	HS 2	G = 75 [s] + 1500 / 1000 * 30 + 1500 / 1000 * 5	g3 / Fahrzeit von Linie 5 auf dieser Strecke g4 / Fahrtkosten g7 / Nichtschnellbahngewicht, da Buslinie
HS 2	HS 2	G = 30 [s] + 200 [s]	g5 / allgemeiner Umsteigezuschlag g6 / Umsteigewartezeit in Linie {2,3} Taktzeit von Linie {2,3} = 6,67 Minuten
HS 2	HS 3	G = 45 [s] + 800 / 1000 * 30	g3 / Fahrzeit von Linie {2,3} g4 / Fahrtkosten
HS 3	Bezirk B	G = 250 [m] / 1,2 [m/s]	g8 / Fußweg Abgang

Das Gesamtgewicht des Weges beträgt: G_{Weg} = 1034,8 [s]

4. Algorithmen zur Wegsuche

4.1 Allgemeines

Dieses Kapitel konzentriert sich, völlig getrennt von der verkehrlichen Seite, auf die reine Wegsuche im Distanzgraphen. Hierfür werden verschiedene Algorithmen vorgestellt und untersucht.
Im Anhang A ist ein Beispielgraph zu finden, ein sogenannter „Negativ-Graph“. Er enthält alle Probleme und Spezialfälle die bei der Wegsuche in dieser Arbeit aufgetreten sind. An ihm können alle weiteren Algorithmen, die in Zukunft noch für dieses Problem untersucht werden, getestet werden.

Da in der Literatur eine große Anzahl von Algorithmen für die verschiedensten Arten von Wegsuchen zu finden ist, wird an dieser Stelle noch einmal die Anforderung an die benötigte Wegsuche beschrieben:

> **Vorgegeben ist ein gerichteter, gewichteter und zusammenhängender Digraph $G = (V, E, \gamma)$ mit $|V| = N$ und positiver Gewichtsfunktion $\gamma : E \rightarrow |R_0^+$ (Distanzgraph). Dieser Graph muß nicht schleifenfrei sein. Er kann aber keine negativen Schleifen enthalten (γ ist so definiert).**
> **Es sind alle schleifenfreien Wege von einem bestimmten Knoten u zu einem anderen Knoten v ($u,v \in V$) gesucht, deren Gewicht zwischen dem Gewicht des/eines Bestweges von u nach v und einem bestimmten Grenzgewicht liegt (jeweils einschließlich). Das Grenzgewicht berechnet sich wie folgt:**
>
> **GB = Gewicht des/eines Bestweges von u nach v**
> **Grenzgewicht = Minimum { GB x (1,00 + relFaktor), GB + absFaktor }**
>
> **Die Werte relFaktor und absFaktor sind ebenfalls vorgegeben (siehe Kapitel 2.2).**

Es wurden in der Literatur keine Algorithmen gefunden, die genau dieses Problem lösen würden, aber es gibt eine ganze Reihe, die ähnliche Wegprobleme lösen. Diese lassen sich in verschiedene Gruppen einteilen:

shortest path	der kürzeste Weg	
k-shortest paths	die k kürzesten Wege	
one-to-one oder *single-pair*	von einem Knoten zu einem anderen Knoten	(1:1)
one-to-all oder *single-source*	von einem Knoten zu allen anderen Knoten	(1:N)
all-to-all oder *all-pairs*	von allen Knoten zu allen anderen Knoten	(N:N)

Für die Wegsuche der Mehrwegumlegung kommen die Algorithmen der Gruppe

single-pair k-shortest loopless paths

am ehesten in Frage.
Ob *single-pair* oder *all-pairs* kann auf Anhieb nicht entschieden werden. Algorithmen, die speziell *all-pairs* in einem Durchgang berechnen, benötigen im Allgemeinen weniger Aufwand, wie ein *single-pair*-Algorithmus, der entsprechend oft aufgerufen wird, da er Informationen von bereits durchgeführten Teilberechnungen in weiteren Berechnungen weiterverwenden kann. Auf der anderen Seite sind in den Belastungsdateien der Verkehrsnetze meist nicht alle möglichen Bezirkspaarkombinationen enthalten, sondern oftmals nur zehn oder zwanzig Prozent aller möglichen Paare. Mit einem *all-pairs*-Algorithmus würden in diesem Fall viele unnötige Wege berechnet.
Es hat sich bei der Untersuchung der Algorithmen gezeigt, daß alle Algorithmen, mit Ausnahme des Algorithmus von Floyd/Minieka, vom Typ *single-pair* sind. Weiterhin hat sich gezeigt, daß es zwar bei der Suche nach dem Bestweg, nicht jedoch bei der Mehrwegsuche möglich ist, Informationen von einem Berechnungsdurchgang auf ein anderes Bezirkspaar zu übertragen und dort weiterzuverwenden.

Die Algorithmen vom Typ *single-pair k-shortest paths* berechnen zuerst den Bestweg. Ausgehend von diesem, wird dann durch Einbau von Umwegen der zweitbeste, drittbeste usw. Weg ermittelt. Damit lassen sich sehr leicht alle Wege innerhalb der vorgegebenen Grenzen ermitteln. Dazu wird so lange iterativ der nächstbeste Weg ermittelt, bis er außerhalb der Grenzen liegt:

```
Berechne Bestweg p_1
Berechne aus Gewicht(p_1) das Grenzgewicht GG
i := 1
repeat
   i := i + 1
   Berechne i.-besten Weg p_i
until Gewicht(p_i) > GG
Ergebniswege sind dann p_1,...,p_i-1
```

Ein Großteil der untersuchten Algorithmen liefert nicht nur schleifenfreie Wege, sondern auch Wege, die Schleifen enthalten. Für die Verkehrsumlegung sind Schleifenwege jedoch nicht sinnvoll, da ein Fahrgast kaum freiwillig Schleifen im Netz fahren wird, es sei denn, er befindet sich auf einer Stadtrundfahrt, was dann wiederum nicht relevant für die Umlegungsrechnung wäre.
Die nachfolgend angegebene, einfache Funktion „Schleifencheck“ testet einen Weg darauf, ob er schleifenfrei ist oder nicht. Bei Algorithmen die auch Schleifenwege in ihrem Berechnungsergebnis enthalten, müßten zum Beispiel nach der Berechnung noch alle Schleifenwege mit diesem Schleifencheck herausgefiltert werden. Dadurch würde jedoch ein höherer Aufwand verursacht, als wenn der Algorithmus gleich bei seiner Ausführung die Schleifenwege unterdrücken und gar nicht berechnen würde. Aber dies ist bei einigen Algorithmen nicht möglich (siehe z.B. Hoffman & Pavley, Abschnitt 4.6).

Funktion Schleifencheck

Mit dieser einfachen Funktion kann ein gegebener Weg auf Schleifenfreiheit hin überprüft werden.

Der zu prüfende Weg sei $p = (u_1, u_2, ..., u_j)$ mit $u_i \in V$ für $1 \le i \le j$ und es sei $|V| = N$

Bei der Prüfung des Weges wird ermittelt, ob es einen Knoten in p gibt, der mindestens zweimal in p enthalten ist. Dies würde eine Schleife bedeuten. Hierzu wird für jeden Knoten $u \in V$ eine Feldvariable $B(v)$ angelegt, wobei gelten soll:

$$B(v) = \begin{cases} 1 & \text{, falls } v \text{ mindestens einmal in } p \text{ enthalten ist} \\ 0 & \text{, sonst} \end{cases}$$

Die Prüfroutine läuft dann wie folgt ab:

```
Result := false                  * Initialisierung von Result *
for all v ∈ V do
    B(v) := 0                    * Initialisierung aller B(v) *
end for
for all u_i ∈ p do               * Prüfung aller Knoten von p *
    if B(u_i) = 1 then
       Result := true
       exit for
    else
       B(u_i) := 1
    end if
end for
```

Wenn die Routine als Ergebnis `Result = true` liefert, so enthält p mindestens einen Knoten, der mindestens doppelt in p enthalten ist, und damit eine Schleife. Ist hingegen `Result = false`, so ist p schleifenfrei.

Aufwand: Die erste `for`-Schleife wird genau N-mal durchlaufen (|V| = N). Da ein Weg theoretisch unendlich viele Knoten enthalten kann, liefert die Schleifenbedingung der zweiten Schleife keine Abschätzung für die Laufzeit. Aber nach spätestens N Durchläufen der Schleife, muß der Weg p einen Knoten benutzen, den er bereits vorher einmal besucht hat, und der `then`-Teil der `if`-Anweisung beendet die Schleife.
Damit kann die Schleifenfreiheit von Wegen mit dieser Routine in $\mathrm{O}(2N)$ Schritten bestimmt werden.

In den weiteren Abschnitten dieses Kapitels werden nun die einzelnen Algorithmen vorgestellt, die ich für die Wegsuche der Mehrwegumlegung in der Literatur gefunden habe.

4.2 Dijkstra (Bestwegsuche)

Der Algorithmus von E. W. Dijkstra aus dem Jahre 1959 [Dijkstra 59] ist der klassische und bekannteste Greedy-Algorithmus für die Bestwegsuche in Graphen. Er gehört zur Gruppe der *single-source shortest-path* Algorithmen und berechnet somit den Bestweg von einem Startknoten zu allen anderen Zielknoten.

Dieser Algorithmus wird hier deswegen erwähnt, weil viele Algorithmen vom Bestweg ausgehen und aus diesem durch Einbau eines Umwegs den zweitbesten, drittbesten usw. Weg berechnen. Zur Berechnung des Bestweges verweisen sie jedoch auf einen geeigneten Algorithmus, und das ist in diesem Falle der von Dijkstra.

Ausgangspunkt ist ein Graph $G = (V, E, \gamma)$ mit positiven Kantengewichten (Distanzgraph). Für den genauen Ablauf des Algorithmus soll hier auf die Literatur verwiesen werden (z.B. [Ottmann 93] oder [Atluja 93]), wo dieser Algorithmus ausführlich beschrieben und diskutiert wird.
Der normale Aufwand für Dijkstra beträgt $\mathrm{O}(N^2)$ mit $N = |V|$.
Ein wichtiger Punkt in diesem Zusammenhang bildet die sogenannte Randverwaltung. Sie muß folgende Operationen zur Verfügung stellen:

I. Rand als leer initialisieren
II. Prüfung, ob Rand leer ist
III. Entfernung des Knotens mit dem geringsten Gewicht
IV. Einen Knoten neu in den Rand aufnehmen
oder das Gewicht eines bestehenden Knotens ändern.

Für ihre Implementation dieser Randverwaltung gibt es mehrere Ansätze [siehe auch Hoffmann 93], die zu unterschiedlichen Laufzeiten führen:

Unsortierter Rand/Selection Sort
Hierbei wird der Rand in keinem eigenen Speicherbereich verwaltet, so wie es Dijkstra in seinem Originalalgorithmus vorgestellt hat. Jeder Knoten erhält einen Zustand (besucht, im Rand, unbesucht) und für Operation III werden alle Knoten durchgegangen und derjenige mit dem kleinsten Gewicht entfernt. Diese Methode eignet sich vor allem für sehr dichte Graphen.

Insert Sort
In diesem Fall wird der Rand als sortierte Liste verwaltet, das heißt, bei Operation IV wird der Knoten gleich sortiert in die Liste aufgenommen bzw. sein momentaner Platz geprüft und eventuell verändert. Die Operation III wird daher sehr schnell ausgeführt, da nur auf das erste Element der Liste zugegriffen werden muß.

Fibonacci-Heap

Diese Art der Randverwaltung ist im vorliegenden Fall unbrauchbar, da die Schlüsselmenge (die Gewichte) disjunkt sein muß und dies ist nicht der Fall.

Heapsort

Bei dieser Randverwaltung werden die Knoten in einem sortierten Heap ähnlich der Methode Insert Sort verwaltet. Hier ergibt sich jedoch der Vorteil, daß das Einfügen eines Knotens in einen Heap weniger Aufwand verursacht, als beim Insert Sort (sehr effizient bei dünnen Graphen).

Für das vorliegende Problem eignen sich demnach nur die Methoden Selection Sort, Insert Sort und Heapsort, wobei Selection Sort die längste Laufzeit haben dürfte, da aus Verkehrsnetzen im allgemeinen nicht sehr dichte Graphen gewonnen werden. Die Verwaltung mittels eines sortierten Heaps hat kleinere Vorteile gegenüber dem Insert Sort und dürfte damit am schnellsten Ablaufen. Diese theoretischen Überlegungen können das Laufzeitverhalten der Randverwaltung aber nur bis zu einem gewissen Grad abschätzen, denn auch der Aufbau der Verkehrsnetze und damit der Graphen, hat Einfluß auf die Laufzeit der Randverwaltung.
Da über den Aufbau der Netze/Graphen keine relevanten Aussagen getroffen werden konnten, wurden alle drei Randverwaltungen implementiert. Bei den Laufzeitmessungen eines kleineren (LUMMERLAND) und eines mittleren Netzes (ULM) zeigte es sich, daß die Randverwaltung mittels eines sortierten Heaps am geeignetsten erscheint:

	Laufzeit von Dijkstra	
Randverwaltung	Lummerland	ULM
Selection Sort	1,02 Sek.	5:18 Min.
Insert Sort	1,04 Sek.	4:05 Min.
Heapsort	0,57 Sek.	2:17 Min.

Es wird daher im weiteren immer Dijkstras Algorithmus mit sortiertem Heap als Randverwaltung verwendet.
Diese Variante hat einen Gesamtaufwand von $O(M \log N)$ mit $N = |V|$, $M = |E|$. Für sehr dichte Graphen $\left(M \approx N^2\right)$ ist dieser Aufwand schlechter als der ursprüngliche Dijkstra mit $O(N^2)$, aber bei Verkehrsnetzen handelt es sich immer um sehr lichte Graphen.

4.3 Methoden aus der KI

Im Bereich der „Künstlichen Intelligenz“ wird ebenfalls nach Wegen innerhalb von Graphen gesucht, wobei die Graphen hier meist einen Baum aller Lösungsmöglichkeiten und Strategien zur Lösung eines bestimmten Problems darstellen. In diesem Bereich sind zwei Algorithmen erwähnenswert, die in [Winston 89] sehr schön beschrieben werden:

Breitensuche
Die Breitensuche oder auch *breadth-first-search* liefert nicht den besten Weg von einem Start zu einem Ziel, sondern irgend einen Weg. Sie beginnt mit ihrer Suche am Startknoten und breitet sich in konzentrischen Kreisen um diesen Punkt aus, wobei alle möglichen Kanten in Betracht gezogen werden. Erreicht diese Ausdehnung irgendwann den Zielpunkt, so ist die Suche beendet und ein beliebiger Weg vom Start zum Ziel gefunden. Wenn dieses Abbruchkriterium jedoch außer Kraft gesetzt und die Suche fortgeführt wird, bis alle Kanten des Netzes besucht worden sind, so ist es damit möglich alle Wege vom Start bis zum Ziel zu erhalten. Für die Mehrwegsuche im vorliegenden Fall müßten aus diesen Wegen nur noch die gültigen Wege herausgefiltert werden.
Der große Nachteil der Breitensuche ist ihr erheblicher Aufwand, da alle möglichen Wege berechnet werden. Wegen ihres großen Aufwands wird die Breitensuche daher nur dann eingesetzt, wenn keine Heuristiken verwendet werden können und tatsächlich alle Wege berechnet werden müssen.
Dijkstras Algorithmus ist vom Grundaufbau ebenfalls eine Breitensuche, nur nutzt er den Vorteil aus, daß in einem Distanzgraphen ein Weg durch Hinzunahme einer weiteren Kante nicht kürzer werden kann und beschleunigt die Suche damit erheblich.

Die Breitensuche ist eigentlich kein Algorithmus speziell aus der KI sondern viel mehr aus der Graphentheorie. Da sie aber in der KI sehr häufig zur Anwendung kommt, weil dort in vielen Fällen keine andere Wahl besteht, als alle Möglichkeiten durchzutesten, habe ich sie in diesen Abschnitt mit aufgenommen.

Der Algorithmus A*
Ein in der KI oft verwendeter Algorithmus zur Berechnung des Bestweges trägt den Namen A*. Er vereinigt die Techniken *branch-and-bound* und dynamisches Programmieren, sowie eine Heuristik. Und genau diese Heuritik unterscheidet A* von Dijkstras Algorithmus. Während Dijkstra, ähnlich der Breitensuche, in konzentrischen Kreisen vom Startpunkt aus seine Umgebung erkundet, sucht A* mit Hilfe seiner Heuristik nur in Richtung auf das Ziel weiter. Als Heuristik eignet sich zum Beispiel die Luftlinienentfernung zum Zielpunkt sehr gut. Damit kann die Wegsuche sehr gut auf die Richtung zum Ziel beschränkt werden und es wird verhindert, daß der Bestweg vom Startpunkt aus in entgegengesetzter Richtung gesucht wird (wo er in diesem Fall garantiert nicht zu finden ist).
Dies setzt aber wiederum voraus, daß die Luftlinienentfernung ein geeignetes Maß für die Abschätzung des Weggewichts sein muß. Das heißt, die Weggewichte müssen den Entfernungen zwischen den Knoten entsprechen.
Da im vorliegenden Fall die Kantengewichte nur in geringem Maße der Knotenentfernung entsprechen und diese ebenfalls auch keine weitere Heuristik ermöglichen, die die Wegsuche auf ein Teilgebiet beschränken könnte, ist der A*-Algorithmus für die Mehrwegsuche ebenfalls nicht brauchbar.

Alle folgenden Algorithmen haben die Aufgabe die k besten Wege durch einen Graphen zu ermitteln.

4.4 Naiver Algorithmus

Dieser Algorithmus ist einer simplen Überlegung von Prof. Claus und mir entsprungen. Dazu ein einfacher Beispielgraph:

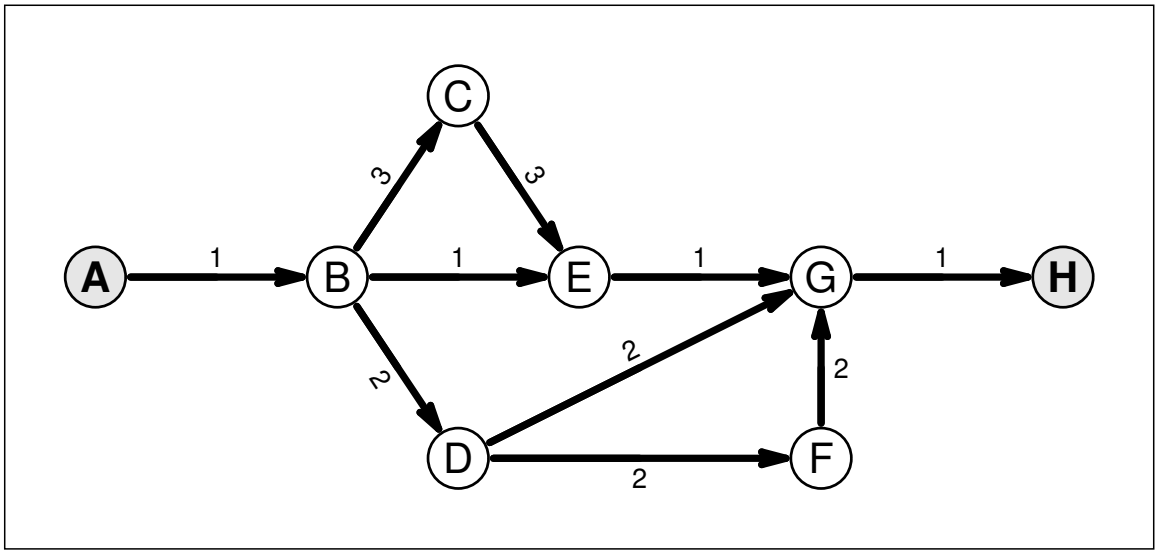


Abb. 4.1: Beispielgraph

An Hand dieses Beispielgraphen soll die Berechnung der drei besten Wege von Knoten A zu Knoten H demonstriert werden:

1. Schritt
Berechnung des Bestweges mit Hilfe von Dijkstra
Bestweg: A-B-E-G-H (Gewicht = 4)

2. Schritt
Die Berechnung des zweitbesten Weges erfolgt dadurch, daß sukzessive jede Kante des Bestweges einzeln aus dem Graphen vorübergehend entfernt wird, danach wiederum der Bestweg nach Dijkstra ermittelt wird und dann die Kante wieder in den Graphen eingefügt wird. Aus dieser Menge von Wegen wird derjenige mit dem geringsten Gewicht ermittelt. Dies ist der zweitbeste Weg.

vorübergehend gelöschte Kante: A-B	Bestweg: keine Weg vorhanden
vorübergehend gelöschte Kante: B-E	Bestweg: A-B-D-G-H (Gewicht = 6)
vorübergehend gelöschte Kante: E-G	Bestweg: A-B-D-G-H (Gewicht = 6)
vorübergehend gelöschte Kante: G-H	Bestweg: kein Weg vorhanden
	⇒ zweitbester Weg: A-B-D-G-H (Gewicht = 6)

Die Grundüberlegung hinter dieser Methode ist die, daß der zweitbeste Weg sich in mindestens einer Kante vom Bestweg unterscheiden muß. Daher zwingt die vorübergehende Löschung einer Kante des Bestweges den zweitbesten Weg auf eine andere Route. Weil aber unbekannt ist, welche Kante dies ist, müssen schrittweise alle Kanten des Bestweges durchprobiert werden.

3. Schritt
Berechnung des drittbesten Weges. Es wird wie im 2. Schritt vorgegangen, nur daß in diesem Falle zusätzlich für jede Kante des Bestweges auch sukzessive alle Kanten des zweitbesten Weges einzeln gelöscht werden müssen und wiederum der Bestweg bestimmt wird.

Bestweg:	A-B-E-G-H
zweitbester Weg:	A-B-D-G-H

vorüberg. gelöschte Kante: A-B und A-B	Bestweg: kein Weg vorhanden
vorüberg. gelöschte Kante: -“- und B-D	Bestweg: kein Weg vorhanden
vorüberg. gelöschte Kante: -“- und D-G	Bestweg: kein Weg vorhanden
vorüberg. gelöschte Kante: -“- und G-H	Bestweg: kein Weg vorhanden
vorüberg. gelöschte Kante: B-E und A-B	Bestweg: kein Weg vorhanden
vorüberg. gelöschte Kante: -“- und B-D	Bestweg: A-B-C-E-G-H (Gewicht = 9)
vorüberg. gelöschte Kante: -“- und D-G	Bestweg: A-B-D-F-G-H (Gewicht = 8)
vorüberg. gelöschte Kante: -“- und G-H	Bestweg: kein Weg vorhanden
vorüberg. gelöschte Kante: E-G und A-B	Bestweg: kein Weg vorhanden
vorüberg. gelöschte Kante: -“- und B-D	Bestweg: kein Weg vorhanden
vorüberg. gelöschte Kante: -“- und D-G	Bestweg: A-B-D-F-G-H (Gewicht = 8)
vorüberg. gelöschte Kante: -“- und G-H	Bestweg: kein Weg vorhanden
vorüberg. gelöschte Kante: G-H und A-B	Bestweg: kein Weg vorhanden
vorüberg. gelöschte Kante: -“- und B-D	Bestweg: kein Weg vorhanden
vorüberg. gelöschte Kante: -“- und D-G	Bestweg: kein Weg vorhanden
vorüberg. gelöschte Kante: -“- und G-H	Bestweg: kein Weg vorhanden
	⇒ drittbester Weg: A-B-D-F-G-H (Gewicht = 8)

Man sieht, daß der Aufwand für jeden weiteren Weg sehr stark ansteigt.
Angenommen der Bestweg besteht aus m Kanten und der zweitbeste Weg aus n Kanten, so läßt sich der allgemeine Aufwand wie folgt beschreiben:

Für den Bestweg:	$O(\text{Dijkstra})$
Für den zweitbesten Weg:	$O(m \times O(\text{Dijkstra}))$
Für den drittbesten Weg:	$O(n \times m \times O(\text{Dijkstra}))$
...	...

Es ist ersichtlich, daß dieser Algorithmus nur für sehr kleine Graphen und Wege, und für kleine k geeignet ist. Auf der anderen Seite hat dieser Algorithmus den Vorteil nur schleifenfreie Wege zu liefern, da alle Wege als Bestwege in irgend einem verkleinerten Graphen berechnet werden und Bestwege enthalten keine Schleifen, denn sonst wären sie keine Bestwege.

Dieser Algorithmus ist auch in der Literatur bei [Pollack 61a] zu finden.

4.5 Floyd / Minieka

1962 hat R. W. Floyd [Floyd 62] einen Algorithmus zur Bestimmung aller kürzesten Wege eines gerichteten Digraphen veröffentlicht, der heute genauso wie Dijkstras Algorithmus als Standardalgorithmus für die Wegsuche betrachtet wird. Er sollte daher in jedem Grundlagenbuch, das auch Graphen behandelt, zu finden sein.
Der Unterschied ist, daß Floyds Algorithmus (*all-pairs shortest paths*) wesentlich effizienter ist, als die Wiederholung von Dijkstras Algorithmus (*single-source shortest path*) für jeden Knoten.
1974 veröffentlichte E. Minieka [Minieka 74] einen Algorithmus, der durch Generalisierung des Floyd-Algorithmus die k-kürzesten Wege in einem Graphen finden soll (*all-pairs k-shortest paths*).

Voraussetzungen dafür sind:
- ein gerichteter Graph ohne Schleifen mit negativer Länge
- keine zwei verschiedenen Wege in diesem Graphen dürfen die gleiche Länge haben

Minieka definiert:

DEF 5 Ein **k-stelliger Vektor** A, sei ein Vektor mit k (der Größe nach) aufsteigenden Komponenten a_i, der Form:

$$A = \begin{pmatrix} a_1 \\ a_2 \\ \dots \\ a_k \end{pmatrix}, \text{ wobei } (a_i < a_{i+1}) \vee (a_i = a_{i+1} = \infty) \text{ für } 1 \le i \le k-1$$

DEF 6 Das **i-te Minimum** einer endlichen Menge $\{e_1, e_2, \dots, e_h\}$ ist das Element e_r, wobei höchstens i-1 Elemente der Menge kleiner als e_r sind:

$$\min_i\{e_1, e_2, \dots, e_h\} = \begin{cases} e_r, \text{ wobei } \left|\{e_j \mid e_j < e_r \text{ für } 1 \le j \le h\}\right| = i-1 & \text{für } 1 \le i \le h \\ \infty & \text{für } i > h \end{cases}$$

DEF 7 A, B, C seien k-Vektoren der Form $\begin{pmatrix} a_1 \\ a_2 \\ \dots \\ a_k \end{pmatrix}$ bzw. $\begin{pmatrix} b_1 \\ b_2 \\ \dots \\ b_k \end{pmatrix}$ bzw. $\begin{pmatrix} c_1 \\ c_2 \\ \dots \\ c_k \end{pmatrix}$.

Die **Addition A+B** der Vektoren A und B ist derjenige k-Vektor C, dessen Komponenten $c_1, c_2, ..., c_k$ den kleinsten Komponenten aus $\{(a_i + b_j) \mid \text{für } 1 \le i \le k \text{ und } 1 \le j \le k\}$ entspricht:

$$C = A + B = \begin{pmatrix} \min_1\{(a_i + b_j) \mid \text{für } 1 \le i \le k \text{ und } 1 \le j \le k\} \\ \min_2\{(a_i + b_j) \mid \text{für } 1 \le i \le k \text{ und } 1 \le j \le k\} \\ \dots \\ \min_k\{(a_i + b_j) \mid \text{für } 1 \le i \le k \text{ und } 1 \le j \le k\} \end{pmatrix}$$

Die Operation A^* ist definiert als: $A^* = A + A + ... + A$ (k-mal)

DEF 8 Das **Minimum von zwei Vektoren** A, B ist definiert als:

$$\min(A, B) = \begin{pmatrix} \min_1\{a_1, a_2, \ldots, a_k, b_1, b_2, \ldots, b_k\} \\ \min_2\{a_1, a_2, \ldots, a_k, b_1, b_2, \ldots, b_k\} \\ \vdots \\ \min_k\{a_1, a_2, \ldots, a_k, b_1, b_2, \ldots, b_k\} \end{pmatrix}$$

Mit diesen Definitionen läßt sich nun die **Vorgehensweise** beschreiben:

Gegeben ist ein Digraph $G = (V, E, \gamma)$ mit $N = |V|$. Die Kantengewichte im Graph dürfen auch negativ sein, es muß nur sichergestellt sein, daß keine Schleifen mit negativer Länge enthalten sind.
Gesucht werden die k-kürzesten Wege zwischen allen Knoten.

Ähnlich wie der zu Grunde gelegte Algorithmus von Floyd, berechnet Minieka eine Folge von Matrizen der Form $D^1, D^2, \ldots, D^N$, mit D^M = N × N Matrix für $1 \le M \le N$. Jedoch besteht hier jede Matrix D^M aus Elementen $\delta_{i,j}^{M}$ und diese Elemente sind k-Vektoren wie in **DEF 5** definiert. Die einzelnen Komponenten eines solchen k-Vektors werden mit

$$\delta_{i,j}^{M} = \begin{pmatrix} \delta_{i,j}^{M,1} \\ \delta_{i,j}^{M,2} \\ \vdots \\ \delta_{i,j}^{M,k} \end{pmatrix}$$

bezeichnet. Eine Komponente $\delta_{i,j}^{M,r}$ entspricht damit der Länge des r-kürzesten Weges von i nach j, wobei nur die Knoten $1, 2, \ldots, M$ besucht werden dürfen.

Die Ausgangsmatrix D^0 wird wie folgt initialisiert:

$$\delta_{i,j}^{0} = \begin{pmatrix} \gamma((i,j)) \\ \infty \\ \vdots \\ \infty \end{pmatrix} \text{ und } \delta_{i,i}^{0} = \begin{pmatrix} 0 \\ \infty \\ \vdots \\ \infty \end{pmatrix} \text{ für alle } 1 \le i \le N \text{ und } 1 \le j \le N,\ i \ne j$$

Aus dieser Matrix D^0 werden mit den nächsten zwei Formeln sukzessive $D^1, D^2, \ldots, D^N$ berechnet:

(1) $$\delta_{M+1,M+1}^{M+1} = \left(\delta_{M+1,M+1}^{M}\right)^*$$

(2) $$\delta_{i,j}^{M+1} = \min\left\{\delta_{i,j}^{M}, \left(\delta_{i,M+1}^{M} + \delta_{M+1,M+1}^{M+1} + \delta_{M+1,j}^{M}\right)\right\}$$

Der genaue Ablauf:

```
for M = 1 to N
    δ^{M+1}_{M+1,M+1} = (δ^{M}_{M+1,M+1})*                                  (1)
    for i = 1 to N
        for j = 1 to N
            δ^{M+1}_{i,j} = min{δ^{M}_{i,j}, (δ^{M}_{i,M+1} + δ^{M+1}_{M+1,M+1} + δ^{M}_{M+1,j})}   (2)
        end for
    end for
end for
```

Nach Beendigung des Algorithmus, enthält D^N alle gesuchten Ergebnisse, allerdings nur die Längenangaben der entsprechenden Wege. So enthält zum Beispiel $\delta_{i,j}^{N,3}$ die Länge des drittkürzesten Weges von i nach j.
Die zugehörigen Wegverläufe lassen sich nach Minieka während der Berechnung leicht mitprotokollieren.

Aufwandsanalyse

Allgemein:	
Für eine Addition von zwei k-Vektoren:	k^2 Additionen und k^2 Minimumberechnungen
Für die Operation A^*:	k^3 Additionen und k^3 Minimumberechnungen
Für $\min(A, B)$:	$2k$ Minimumberechnungen
Aufwand für Formeln:	
Formel (1)	$O(k^3)$
Formel (2)	$O(2k + 2k^2) = O(k^2)$

Formel (1) wird innerhalb der M-Schleife N-mal berechnet und Formel (2) innerhalb der Schleifen M, i, j N^3-mal.

Gesamtaufwand:		$O(Nk^3 + 2N^3(k^2+k)$
	=	$\mathbf{O(Nk^3 + N^3k^2)}$

[Lawler 77] hat Verbesserungen bei der Berechnung der Vektor-Operationen vorgestellt und erhält einen besseren Aufwand:

Formel (1)	$O(k\ (\log k)^2)$
Formel (2)	$O(k \log k)$
Gesamtaufwand (nach Lawler):	$\mathbf{O(Nk\ (\log k)^2 + N^3k \log k)}$

Dieser Algorithmus von Minieka hat einen geringeren Aufwand als alle weiteren Algorithmen, die noch vorgestellt werden, da er der einzige *all-pairs*-Algorithmus ist. Alle anderen Algorithmen sind vom Typ *single-pair*. Daß dieser Algorithmus dennoch nicht für die Mehrwegumlegung geeignet ist, liegt vor allem an drei wesentlichen Nachteilen:

1. Die Vorbedingung
Die Bedingung, daß alle Wege im Graphen unterschiedliche Gewichte haben müssen, ist weder in Verkehrsnetzen gegeben noch mit vertretbarem Aufwand herzustellen. Minieka erwähnt, daß dies durch eine leichte Veränderung der einzelnen Kantengewichte realisierbar sei. Eine Methode oder den Aufwand hierfür gibt er jedoch nicht an.

Wird der Algorithmus dennoch auf einen Graphen angewendet, der zum Beispiel zwei verschiedene Wege von i nach j mit dem gleichen Gewicht enthält, so wird von diesem Algorithmus nur einer der beiden Wege gefunden. Der andere Weg fällt bei der Minimumberechnung in Formel (2) heraus, da in einer Menge nicht zwei Elemente der gleichen Größe (d.h. zwei gleiche Elemente) enthalten sein können.

2. Konstantes k für alle Knotenpaare
Das k muß in diesem Fall vor Beginn des Algorithmus festliegen und ist für alle Knotenpaare gleich. Dies mag bei anderen Aufgabenstellungen kein Problem sein, aber hier sind ja eigentlich nicht die k-kürzesten Wege, sondern alle Wege innerhalb der Grenzen Gewicht(Bestweg) und Grenzgewicht gewünscht. Das heißt, die Anzahl der Wege, die zwischen zwei Knoten berechnet werden, differiert von Knotenpaar zu Knotenpaar. Es ist möglich, daß bei einem Knotenpaar innerhalb der Grenzen vier Wege liegen und bei einem anderen Paar fünfzig Wege.
Einzige Möglichkeit wäre, das maximale k für alle Knotenpaare zu bestimmen und dann den Algorithmus mit diesem k zu starten, wobei in Kauf genommen wird, daß bei vielen Knotenpaaren zu viele Wege berechnet werden. Der Aufwand zur Bestimmung dieses maximalen k jedoch, läßt eine Realisation dieser Idee nicht zu.

3. Unvollständigkeit der Belastungsdateien
Der Vorteil von *all-pairs*-Algorithmen, daß sie Teilergebnisse bei der Berechnung von Wegen zwischen zwei Knoten eventuell bei der Berechnung von zwei anderen Knoten wiederverwenden können, wird hier durch den Nachteil gedämpft, daß die Belastungsdateien meist nicht vollständig sind. Das heißt, es werden gar nicht alle Wege zwischen allen Knoten benötigt und der Algorithmus berechnet „zu viel“. Zum Teil enthalten die Belastungsdateien nur 10% aller möglichen Knotenpaare.
Eine genaue Abschätzung des Vorteils von *all-pairs* gegen die „Zu-Viel-Berechnung“ ist sehr schwierig.

4.6 Hoffman & Pavley

1959 stellten Hoffman und Pavley in [Hoffman 59] eine Methode vor, den k-besten Weg von i nach j zu Berechnen, in dem Umwege in den kürzesten Weg von i nach j eingebaut werden.
Der Algorithmus ist vom Typ *single-pair k-shortest paths*.

Auch hierzu werden einige Definitionen gemacht:

DEF 9 Ein **Weg** p, ist eine Folge von Knoten aus V:

$$p = (v_1, v_2, \ldots, v_j) \quad \text{mit } v_i \in V \text{ für } 1 \le i \le j$$

Die **Knotenanzahl** l eines Weges $p = (v_1, v_2, \ldots, v_j)$ ist definiert als:

$$l(p) = j$$

Das **Gesamtgewicht** g eines Weges $p = (v_1, v_2, \ldots, v_j)$ ist definiert als:

$$g(p) = \sum_{i=1}^{l(p)-1} \gamma((v_i, v_{i+1}))$$

Die **Konkatenation** $\oplus$ von zwei Wegen $p = (v_1, v_2, \ldots, v_j)$ und $q = (u_1, u_2, \ldots, u_m)$, wobei gelten muß, daß $v_j = u_1$ ist, ist definiert als:

$$p \oplus q = (v_1, v_2, \ldots, v_j, u_2, u_3, \ldots, u_m)$$
$$l(p \oplus q) = l(p) + l(q) - 1$$

Für die Ermittlung der k kürzesten Wege von einem bestimmten Knoten a zu einem bestimmten Knoten b werden Mengen benötigt:

DEF 10 Φ_i ist eine **Menge** von i-kürzesten Wegen von a nach b mit gleichem Gewicht. Das heißt, in Φ_3 sind alle drittkürzesten Wege von a nach b enthalten. Wenn keine zwei Wege von a nach b das gleiche Gewicht haben, so bestehen alle Φ_i immer nur aus einem Weg.

Das **Gewicht** W aller Wege in Φ_i ist definiert als: $W(\Phi_i) = g(p)$ für alle $p \in \Phi_i$
In [Hoffman 93] wird für das Gewicht $V(\Phi_i)$ verwendet, da man dies aber leicht mit der Knotenmenge V verwechseln kann, wird in dieser Arbeit W verwendet.

DEF 11 ERG ist eine Menge von Wegen. Sie enthält im Laufe der Berechnung ein paar Wege als Zwischenergebnis.

DEF 12 Für alle Knoten $v \in V$ wird ein bestimmter Weg definiert: Die Spur.
Unter der Spur $sp(v)$ eines Knotens v, versteht man den kürzesten Weg von v zum Zielknoten b. Diese Spurwege müssen vor Beginn der Berechnung durch einen geeigneten Algorithmus (z.B. Dijkstra) berechnet werden.

DEF 13 **Abweichung**: Der Wert $Dev(c, j, s)$ ist das Gewicht des Weges, der entsteht, wenn vom Weg $s = (a, u_1, u_2, \ldots, u_z, b)$ am Knoten $c \in (a, u_1, u_2, \ldots, u_z)$ zum Knoten j abgewichen und vom Knoten j zum Ziel b der kürzeste Weg verwendet wird:

$$Dev(c,j,s) = g\big((a, u_1, u_2, \ldots, u_i) \oplus (u_i, j) \oplus sp(j)\big) \text{ mit } u_i = c \text{ und } j \neq u_{i+1}$$

Gegeben: Ein zusammenhängender Digraph $G = (V, E, \gamma)$, der keine negativen Schleifen enthält. Weiterhin einen Startknoten a, einen Zielknoten b und eine Zahl k, $(a,b \in V)$.

Gesucht: Die k kürzesten Wege von a nach b.

Vorgehensweise des Algorithmus:

Die grundlegende Idee dieses Algorithmus ist:

Der k-kürzeste Weg ist eine Abweichung vom r-kürzesten Weg, $r < k$

Ausgehend vom kürzesten Weg, der zum Beispiel mit Dijkstra berechnet wurde, wird an jedem Knoten dieses Weges zu einem anderen Knoten als dem Nachfolgeknoten abgewichen (sofern eine Kante vorhanden ist) und von diesem neuen Knoten der kürzeste Weg zum Ziel verwendet.
Alle diese Wege werden in einer Menge gesammelt und derjenige, mit dem geringsten Gewicht innerhalb der Menge ist der zweitkürzeste Weg.

Allgemein: $W(\Phi_1)$ = Gewicht des kürzesten Weges von a nach b

$$W(\Phi_i) = \min\{Dev(c,j,s) \mid Dev(c,j,s) > W(\Phi_{i-1})\}$$

Dieses Minimum wird berechnet über: alle $s \in \Phi_h,\ h < i$
alle c, die Anfangsknoten von Kanten in s sind,
alle j, die Endknoten von Kanten sind, die bei c beginnen

Formaler Ablauf

Konstruktion der $\Phi_2, \Phi_3, \ldots, \Phi_r$:

(1) r = 1
(2) while $\left(\sum_{x=1}^{r} |\Phi_x|\right) < k$ do
(3) r = r + 1
(4) h = r −1
(5) for all $q \in \Phi_h$
(6) o.B.d.A. sei $q = (v_1, v_2, \ldots, v_j)$ mit $v_1 = a$ und $v_j = b$
(7) for all $v_i \in q \quad 1 \le i \le j-1$
(8) for all $u \in V$ mit $(v_i, u) \in E \wedge (u \neq v_i) \wedge (u \neq v_{i+1})$
(9) $w = (v_1, v_2, \ldots, v_i) \oplus (v_i, u) \oplus sp(u)$
(10) if $g(w) > W(\Phi_{r-1})$ then
(11) $Erg = Erg \cup w$
(12) end if
(13) end for u
(14) end for v_i
(15) end for q
(16) $m = \min\{g(q) \mid q \in Erg\}$
(17) $W(\Phi_r) = m$
(18) $\Phi_r = \{q \mid g(q) = m \wedge q \in Erg\}$
(19) $Erg = Erg \setminus \Phi_r$
(20) end while

Aufwand

Es wird hier der größere Aufwand $O(N^2)$ des einfachen Dijkstra verwendet um zu zeigen, daß der Aufwand für Dijkstra in jedem Fall immer unter dem Aufwand für den eigentlichen Algorithmus bleibt.

Aufwand für *single-pair k-shortest paths*: (1:1)
$= O(\text{Dijkstra } 1{:}N) + O(N^2k)$
$= O(N^2 + N^2k)$
$= O(N^2k)$

Aufwand für *all-pairs k-shortest paths*: (N:N)
$= O(\text{Dijkstra } N{:}N) + O(N^4k)$
$= O(N^3 + N^4k)$
$\mathbf{= O(N^4k)}$

Das Problem ist, daß der Algorithmus auch Wege mit Schleifen findet. Dazu ein Beispiel:

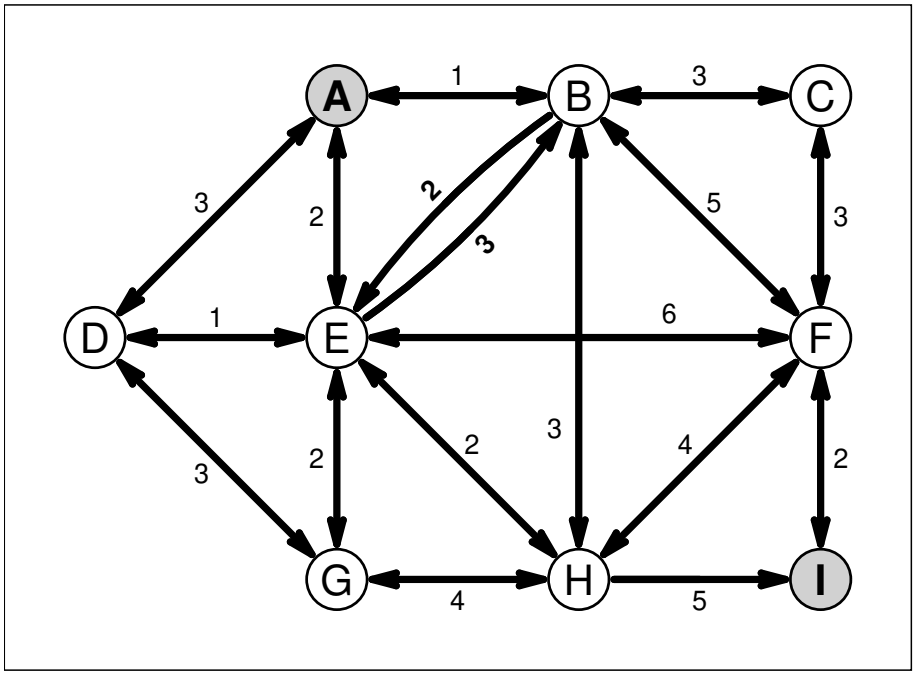


Abb. 4.2: Beispielgraph für Hoffman & Pavley

Beispielberechnung der acht besten Wege von Knoten A nach Knoten I im Graphen der Abbildung 4.2:

Bestweg: Φ_1 = {(A-B-F-I)} $W(\Phi_1) = 8$ Von Dijkstra ermittelt.

Berechnung von Φ_2 (Abweichungskanten sind **fett**)

Erg = { (**A-D**-E-H-I) Gewicht = 11
(**A-E**-H-I) Gewicht = 9
(A-**B-A**-B-F-I) Gewicht = 10
(A-**B-C**-F-I) Gewicht = 9
(A-**B-E**-H-I) Gewicht = 10
(A-**B-H**-I) Gewicht = 9
(A-B-**F-B**-F-I) Gewicht = 18
(A-B-**F-C**-F-I) Gewicht = 14
(A-B-**F-E**-H-I) Gewicht = 19
(A-B-**F-H**-I) } Gewicht = 14

Φ_2 = { (A-E-H-I), (A-B-C-F-I), (A-B-H-I) } $W(\Phi_2) = 9$

....

Φ_3 = { (A-B-A-B-F-I), (A-B-E-H-I), (A-E-F-I), (A-E-H-F-I) $W(\Phi_3) = 10$

....

Voraussetzung für diesen Algorithmus ist allerdings, daß in Φ_1 alle Bestwege enthalten sind und nicht nur einer, denn weitere Bestwege w würden zwar bei der Berechnung von Φ_2 gefunden, aber durch die Bedingung $g(w) > W(\Phi_1)$ nicht weiter berücksichtigt werden.

Ein schon erwähnter Nachteil dieses Algorithmus ist die Möglichkeit, daß auch Schleifenwege berechnet werden, wie in Φ_3 des Beispiels zu sehen ist.

Um das Problem der Schleifenwege zu lösen, könnte man innerhalb der inneren Schleife einen neu gefundenen Weg w erst einem Schleifencheck unterziehen, bevor man ihn zur Menge Erg vereinigt:

```
...
(10)          if g(w) > W(Φ_r-1) then
(10a)            if Schleifencheck(w) = false then      (neue Zeile)
(11)                Erg = Erg ∪ w
(11a)            end if                                 (neue Zeile)
(12)          end if
...
```

Eine Aufwandsabschätzung für die Berechnung der schleifenfreien Wege ist nicht möglich, da nicht bekannt ist, wieviele Wege insgesamt berechnet werden müssen, damit k schleifenfreie Wege darunter sind.
Außerdem scheitert die oben beschriebene Methode daran, daß es erstaunlicherweise bestimmte Graphen gibt, in denen alle korrekten und schleifenfreie Wege nur dann mit dem Algorithmus von Hoffman und Pavley gefunden werden, wenn die Wege mit Schleifen auch berechnet und berücksichtigt werden. Ein Beispiel:

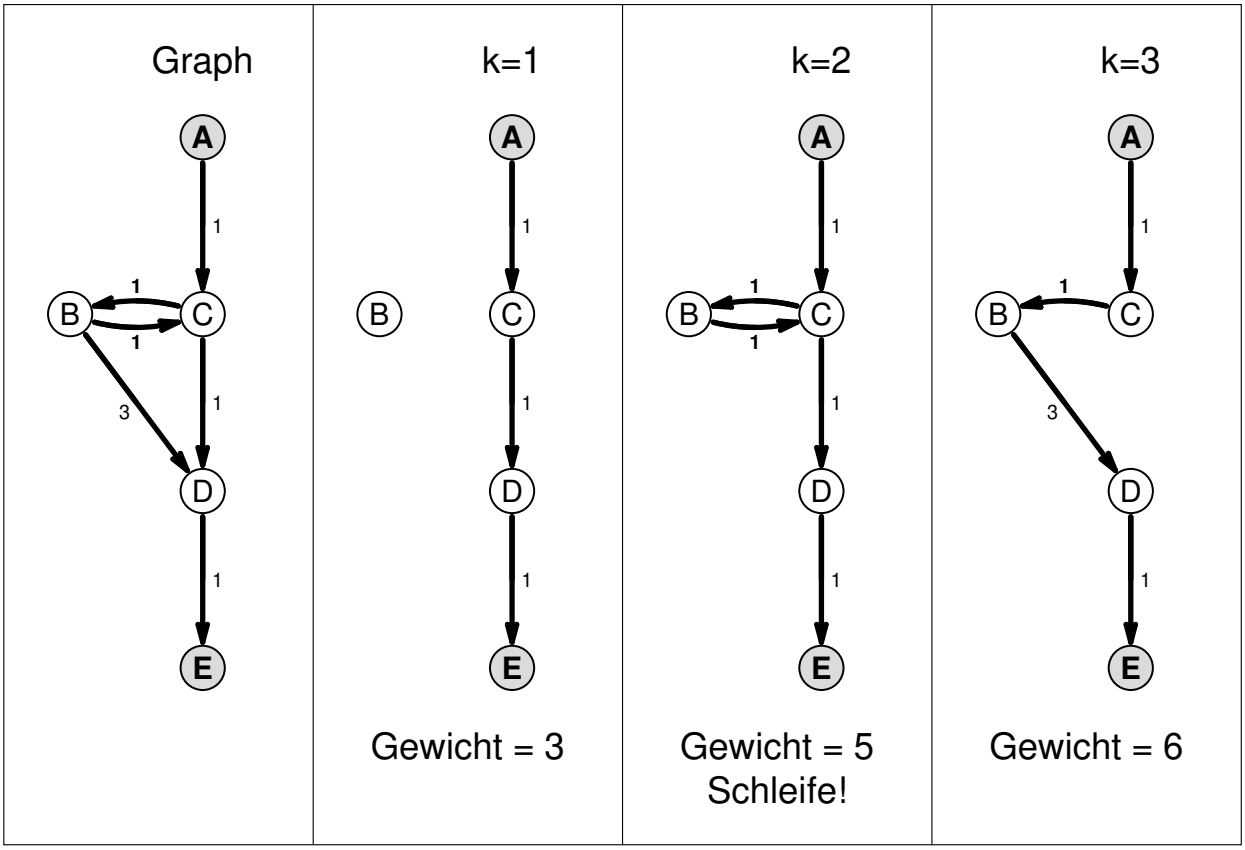


Abb. 4.3: Problem der Schleifenwege bei Hoffman & Pavley

In Abbildung 4.3 ist ein kleiner Graph zu sehen. Für $k = 1$ ist der Bestweg von A nach E zu sehen. Für $k = 2$ würde ein Weg mit einer Schleife berechnet werden. Wenn nun, wie oben vorgeschlagen, dieser Schleifenweg gleich wieder verworfen und nicht in Erg aufgenommen wird, so kann der zweite schleifenfreie Weg im Schritt $k = 3$ nicht gefunden werden, da es dann keinen vorherigen Weg gibt, der über den Knoten B verläuft, so daß man von B nach D abweichen könnte.
Es ist also notwendig den Schleifenweg aus $k = 2$ in Erg und die Φ-Mengen aufzunehmen. Um die Schleifenwege dennoch zu eliminieren, bleibt nur der Weg, erst nach dem Algorithmus alle Ergebniswege einem Schleifencheck zu unterziehen.

Diese Variante hätte dann einen Aufwand von $O(N^5k)$.

4.7 Azevedo et. al.

In [Azevedo 90] wird ein relativ junger Algorithmus für das Problem *single-pair k-shortest paths* beschrieben. Auch dieser Algorithmus basiert, wie der von Hoffman & Pavley, auf dem Theorem:

Der k-kürzeste Weg wird aus Teilen des j-kürzesten Weges gebildet, $j \leq k$

Dazu wird folgende Methode verwendet: Ausgehend von einem Basisgraphen G_1 und dem Bestweg von A nach B, wird dieser Graph in einen Graphen G_2 derart umgebaut, daß der kürzeste Weg im Graphen G_2 dem zweitkürzesten Weg im Graphen G_1 entspricht. Danach werden weitere Graphen G_3, G_4, ... so gebildet, daß allgemein gilt:

Der kürzeste Weg von A nach B im Graph G_i,
ist der zweitkürzeste Weg von A nach B im Graph G_{i-1}
und der i-kürzeste Weg von A nach B im Graph G_1.

DEF 14 Die Menge $A(v)$ eines Knotens $v \in V$, sei die Menge der in i eingehenden Kanten:
$A(v) = \{(u, v) \in E \mid u \in V\}$

Gegeben
- Digraph Graph $G = (V, E, \gamma)$, der keine Schleifen mit einer Länge ≤ 0 enthalten darf
- Ein Startknoten A und einen Endknoten B mit A,B $\in V$
- Eine Zahl k ≥ 1

Gesucht Die k kürzesten Wege von A nach B.

Initialisierung
Damit die Start- und Endknoten A und B nicht mehrmals in Wegen auftauchen, wird aus dem gegebenen Graphen $G = (V, E, \gamma)$ ein Ausgangsgraph $G_1 = (V_1, E_1, \gamma)$ gebildet, indem A und B je einem „Superknoten“ A’ und B’ erhalten:

$$G_1 = (V_1, E_1, \gamma) = (V \cup \{\mathrm{A'}\} \cup \{\mathrm{B'}\}, E \cup \{(\mathrm{A'}, \mathrm{A}), (\mathrm{B}, \mathrm{B'})\}, \gamma)$$

Der neue Startknoten ist demnach A’ und der neue Zielknoten B’.
Der kürzeste Weg von A’ nach B’ muß mit einem anderen Algorithmus (z.B. Dijkstra) gefunden werden.

Berechnung von G_{i+1} ausgehend von G_i:

Es sei $p_i = (\mathrm{A'} = s_0, s_1, \ldots, s_h = \mathrm{B'})$ der kürzeste Weg von A nach B in G_i.

Für s_1 bilde Superknoten s_1’:

$$A(s_1') = \{(x, s_1') \mid (x, s_1) \in E_i \wedge x \neq s_0\}$$

Für $k = 2, 3, \ldots, h\text{-}1$ bilde von s_k den Superknoten s_k’:

$$A(s_k') = \{(x, s_k') \mid (x, s_k) \in E_i \wedge x \neq s_{k-1}\} \cup \{(s_{k-1}', s_k')\}$$

Für s_h = B’ setze:

$$A(\mathrm{B'}) = \{(x, \mathrm{B'}) \mid (x, \mathrm{B'}) \in E_i \wedge x = s_{h-1}\} \cup \{(s_{h-1}', \mathrm{B'})\}$$

Dies sei der Graph G_{i+1}.

Da in [Azevedo 90] der Algorithmus sehr ausführlich und übersichtlich in halbformalem Programmcode wiedergegeben ist, soll hier darauf verzichtet werden. Statt dessen soll die Vorgehensweise noch einmal anschaulich dargestellt werden:

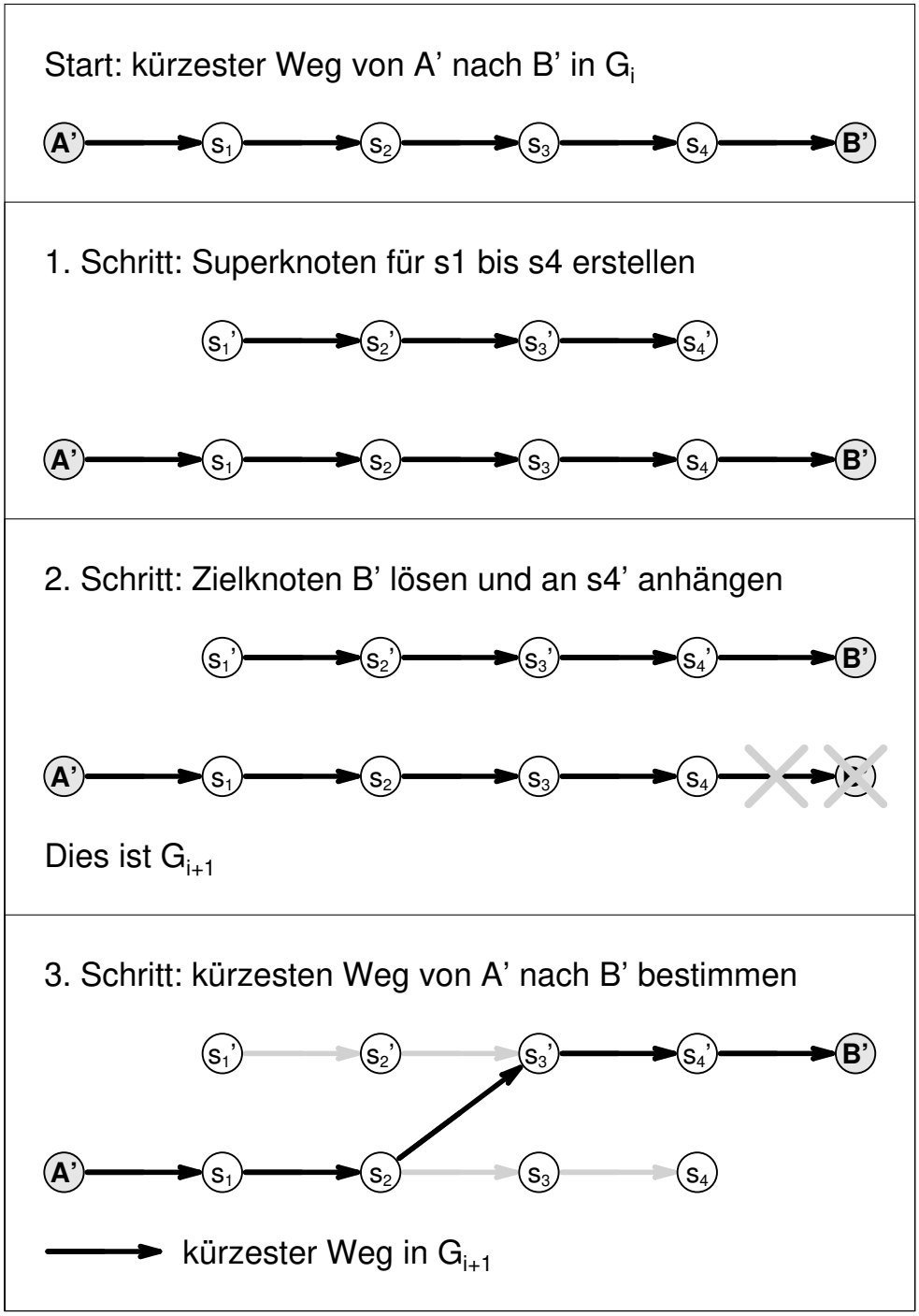


Abb. 4.4: Anschauliche Vorgehensweise

Ausgehend vom kürzesten Weg in G_i, wird für jeden Zwischenknoten des Weges ein Superknoten erstellt. Danach wird der Zielknoten B' an diese Superknoten angehängt.
Es liegen nun zwei „Ebenen" vor, wobei der Startpunkt A' in der unteren Ebene und der Zielpunkt B' in der oberen Ebene liegt. Wenn nun erneut der *shortest path* Algorithmus auf diesen neuen Graphen angewendet wird, so muß der neue kürzeste Weg irgendwann von der unteren Ebene auf die obere Ebene „springen". Dies ist aber nur mit einer Kante möglich, die bisher in keinem kürzesten Weg enthalten war. Der Weg kann natürlich auch ganz aus bisher unbenutzten Kanten bestehen. Wichtig ist nur, daß er mindestens eine neue Kante enthält, die ihn vom alten kürzesten Weg unterscheidet, und somit länger macht.

Da innerhalb der Graphen immer nur der kürzeste Weg gesucht wird, und ein kürzester Weg keine Schleife enthalten kann, denn sonst wäre er nicht der Kürzeste, könnte vermutet werden, daß der Algorithmus von Azevedo nur schleifenfreie Wege berechnet. Das wird leider durch die zusätzlichen Superknoten verhindert. Wenn zum Beispiel im Graphen G_5 ein kürzester Weg berechnet wird und dieser Weg die Knoten s_x und s_x' enthält, ist dieser Weg im Graphen G_5 zweifelsfrei schleifenfrei. Dennoch muß der Weg natürlich wieder zurück auf den Ausgangsgraphen G interpretiert werden und dort werden die Knoten s_x und s_x' auf denselben Knoten abgebildet: Der Weg enthält eine Schleife.
Damit bleibt auch hier nur die Möglichkeit eines anschließenden Schleifenchecks nach dem Algorithmus.

Aufwand

Der Zeitaufwand für diesen Algorithmus ist niedriger als der bei Hoffman & Pavley. Es ist aber zu beachten, daß durch die Superknoten einiges an Platz benötigt wird und auch das Anlegen und Löschen der neuen Kanten Aufwand verursacht.
Azevedo et. al beschreiben den Aufwand für ihren Algorithmus mit:

Aufwand für *single-pair k-shortest paths*	$(1{:}1)$	$O(N^2 + Mk)$	$M = \|E\|$ *und* $N = \|V\|$
Aufwand für *all-pairs k-shortest paths*	$(N{:}N)$	$O(N^3 + N^2Mk)$	

Für die *all-pairs* Berechnung bleibt nichts anderes übrig, als den Algorithmus N^2-mal auszuführen.

Werden nur die schleifenfreien Wege gesucht, besteht auch hier nur die Möglichkeit, nach dem Algorithmus alle Ergebniswege mit dem Schleifencheck zu untersuchen und die Schleifenwege entsprechend zu löschen.
Einen Aufwand für schleifenfreie Wege kann ebenfalls nicht angegeben werden, da nicht bekannt ist, wie viele Wege ermittelt werden müssen, damit k schleifenfreie Wege darunter sind.

4.8 Clarke et. al.

Der in [Clarke 63] beschriebene Algorithmus ist einer der wenigen, der die k-kürzesten schleifenfreien Wege in einem Graphen berechnet. Leider wird dabei aber keine interessante und neuartige Methode verwendet, sondern, ähnlich dem Algorithmus von Hoffman & Pavley, ebenfalls Mengen von Wegen berechnet die auch Schleifenwege enthalten. Mit einem Zähler wird nebenbei festgehalten, wieviele schleifenfreie Wege sich unter den bisher berechneten Wegen befinden.
Die genaue Vorgehensweise ist jedoch sehr umständlich und bringt im Vergleich zu Hoffman & Pavley keine wesentlichen Neuerungen für die Wegberechnung zum Vorschein. Da es auf Grund dieser umständlichen Berechnung nicht möglich ist, den Aufwand für den Algorithmus vernünftig abzuschätzen, und dieser Aufwand sicherlich eine ähnliche Größenordnung wie der von Hoffman & Pavley hat, wird der Algorithmus hier nicht weiter ausgeführt und auf die Literatur [Clarke 63] verwiesen.
Daß andere Literaturstellen kaum auf diesen, doch schon älteren, Algorithmus verweisen, und auch dann keine Aufwandsabschätzung angeben können, zeigt, daß es sich hierbei nicht um einen besonders herausragenden oder effizienten Algorithmus handeln kann.

4.9 Yen

Die Veröffentlichung dieses Algorithmus in [Yen 93] ist sehr schwierig zu lesen, da offenbar eine ältere und sehr gewöhnungsbedürftige formale Notation verwendet wurde. Erschwerend kommt hinzu, daß sich an verschiedenen Stellen Zweideutigkeiten befinden, die es ermöglichen, den Algorithmus auf unterschiedliche Weise auszulegen.
Hat man den Algorithmus jedoch, am Besten an Hand eines Beispiels, einmal durchschaut und durchgespielt, ist er sehr gut zu verstehen und auch die Zweideutigkeiten lassen sich dann nur noch auf eine Art auslegen, damit der Algorithmus korrekt abläuft.
Yens Algorithmus gehört zur Gruppe der *single-pair k-shortest* ***loopless*** *paths* und ist der einzige Algorithmus, der eine genaue Aufwandsabschätzung für schleifenfreie Wege ermöglicht. Auf Grund seiner Effizienz ist er am geeignetsten für die Wegsuche einer Mehrwegumlegung.

Gegeben:
- Graph $G = (V, E, \gamma)$ mit positiven Kantengewichten
- Startknoten A und Zielknoten B

Gesucht: Die K kürzesten, schleifenfreien Wege von A nach B
(Achtung! Das kleine k wird von Yen als Laufvariable verwendet.)

DEF 15 Yen bezeichnet diese k kürzesten schleifenfreien Wege mit A^1, A^2, ..., A^K.
Jeder dieser Wege A^j besteht aus den Knoten a_1^j, a_2^j, ...

Die Anzahl der Knoten von A^j wird durch $L(A^j)$ beschrieben.

Die Matrix $\gamma_2(i, j)$ sei eine $N \times N$ - Matrix, die alle Kantengewichte der Funktion γ aufnehmen kann.

$\oplus$ ist die Konkatenation von zwei Wegen wie in **DEF 9** aus Abschnitt 4.5 beschrieben.

Idee des Algorithmus:

Auch bei diesem Algorithmus wird zur Berechnung des i-kürzesten Weges A^i, vom $(i\text{-}1)$-kürzesten Weg $A^{i\text{-}1}$ ausgegangen. Dazu wird für jeden Knoten a_j^{i-1} von $A^{i\text{-}1}$ eine Schleife ausgeführt:
Für alle Wege A^m mit $1 \le m \le j$, wird geprüft, ob A^m den selben Weganfang bis zum Punkt a_j^{i-1} hat, wie der Weg $A^{i\text{-}1}$:

$$\left(a_1^m, a_2^m, \ldots, a_j^m\right) = \left(a_1^{i-1}, a_2^{i-1}, \ldots, a_j^{i-1}\right)$$

Ist dies der Fall, wird die Kante $\left(a_j^m, a_{j+1}^m\right)$ vorübergehend aus dem Graphen gelöscht (Das Kantengewicht wird auf ∞ gesetzt).
Sind alle Wege A^m verglichen, wird der kürzeste Weg vom Knoten a_j^{i-1} zum Ziel B mit einem speziellen Dijkstra-Algorithmus ermittelt. Dieser gesuchte Weg wird Spur genannt. Der spezielle Dijkstra ist ein normaler Dijkstra mit der zusätzlichen Bedingung, daß für die Spur keine Knoten verwendet werden, die auf dem Weg $A^{i\text{-}1}$ vom Start bis zum Knoten a_j^{i-1} besucht werden:

$$\forall_{x \in Spur} \; x \notin \left(a_1^{i-1}, a_2^{i-1}, \ldots, a_j^{i-1}\right)$$

Ist die Spur gefunden, wird aus dem bisherigen Weganfang von $A^{i\text{-}1}$ und der Spur ein neuer Weg gebildet:

$$W_j = \left(a_1^{i-1}, a_2^{i-1}, \ldots, a_j^{i-1}\right) \oplus spur$$

Da alle vom Knoten a_j^{i-1} ausgehenden Kanten, die von einem der Wege A^m mit $1 \le m \le j$ benutzt wurden, gelöscht worden sind, wird der neue Weg W_j, wenn er an diesem Knoten ankommt, gezwungen, eine neue und bisher nicht benutzte Kante zu verwenden (Dies ist die erste Kante der Spur). Das heißt, W_j ist garantiert länger als $A^{i\text{-}1}$. Weil W_j aber auch gleichzeitig der kürzeste Weg in diesem Graphen ist, ist er der nächstkürzeste Weg nach $A^{i\text{-}1}$.

Da diese Wegsuche in der Schleife für jeden Knoten a_j^{i-1} von $A^{i\text{-}1}$ ausgeführt wird, wird jedes Mal ein kürzester Weg W_j ermittelt, der an diesem Knoten a_j^{i-1} in eine neue Kante verzweigt.
Alle diese Wege werden zur Menge B addiert und der endgültige Weg A^i ist dann das Minimum dieser Menge B.

Algorithmus

A^1 = Dijkstra(A,B)
$B = \emptyset$
for k = 2 to K
    $\gamma_2(i,j) = \gamma((i,j))$ für alle i und j
    ----- (I) -----
    for i = 1 to k-1
        ----- (Ia) -----
        for j = 1 to k-1
            if $(a_1^{k-1}, a_2^{k-1}, \ldots, a_i^{k-1}) = (a_1^j, a_2^j, \ldots, a_i^j)$ then
                $\gamma_2(a_i^j, a_{i+1}^j) = \infty$
            end if
        end for j
        ----- (Ib) -----
        $Spur$ = Spezieller Dijkstra von a_i^{k-1} zum Ziel
        ----- (Ic) -----
        $B = B \cup ((a_1^{k-1}, a_2^{k-1}, \ldots, a_i^{k-1}) \oplus Spur)$
    end for i
    ----- (II) -----
    $A^k = \min\{B\}$
    $B = B \setminus \min\{B\}$
end for k

Aufwand

Programmteil Ia		$O(Nk)$
Programmteil Ib		$O(N^2)$
Programmteil Ic		$O(1)$
Programmteil I		$O(N^3)$
Programmteil II		weniger als $O(N^3)$
Algorithmus Gesamt	(1:1)	$O(N^3k)$
Algorithmus Gesamt	(N:N)	$\boldsymbol{O(N^5k)}$

Damit ist es bei diesem Algorithmus als einzigem möglich, den Aufwand für *single-pair k-shortest loopless paths* genau zu ermitteln.

4.10 Weitere Algorithmen

In diesem Kapitel soll auf weitere Algorithmen zur Wegsuche hingewiesen werden, die aus verschiedenen Gründen in dieser Arbeit nicht dargelegt werden konnten.

Dreyfus
Der in [Dreyfus 69] geschilderte Algorithmus wird in mehreren Literaturstellen als Standardalgorithmus für die Suche der k-kürzesten Wege beschrieben. Er hat die Besonderheit, daß er die k-kürzesten Wege von allen Knoten zu einem bestimmten Zielknoten ermittelt und hat mit einem Aufwand von $O(N^3 + N^2k)$ für eine $N{:}1$-Berechnung klare Vorteile gegenüber dem Algorithmus von Hoffman & Pavley, obwohl seine Berechnungsmethode ähnliche Ansätze aufweist.
Leider erfolgt seine Beschreibung auf einem sehr hohen mathematischen und formalem Niveau, so daß man gezwungen ist, wesentlich mehr Zeit für das Verständnis der Funktionsweise aufzubringen. Da der Algorithmus aber ebenfalls Wege mit Schleifen berechnet, habe ich meine Untersuchungen nach einiger Zeit zugunsten von anderen Algorithmen eingestellt.

Shier
Auch auf den Algorithmus von [Shier 76] wird in mehreren Literaturstellen verwiesen. In [Azevedo 90] wird beschrieben, daß es drei große Klassen von Algorithmen zur Suche der k-kürzesten Wege gibt. Das ist zum einen die Klasse, die auf dem „Prinzip der Optimalität“ basiert. Hierzu gehören die Algorithmen von Dreyfus und Hoffman & Pavley. Eine zweite Klasse beschäftigt sich mit dem Konzept der „path deletion“, wie Azevedo selbst. Die dritte Klasse würde durch Algorithmen, wie den von Shier, gebildet werden.
Leider war es während meiner Arbeit trotz Fernleihe nicht möglich, die Literaturstelle von [Shier 76] zu besorgen. Ich kann daher nur darauf verweisen.

Perko
Der Algorithmus von [Perko 86] soll zumindest dem Titel nach, einen Algorithmus für die k kürzesten **schleifenfreien** Wege enthalten. Unglücklicherweise befindet sich auch diese Literaturstelle (wie die von Shier), in der Zeitschrift NETWORKS. Diese Zeitschrift ist jedoch an der Universitätsbibliothek nicht vorrätig und der Artikel war über die Fernleihe während meine Arbeit nicht erhältlich.

4.11 Zusammenfassung

In diesem Abschnitt sollen noch einmal die erwähnten Algorithmen nach Art und Aufwand übersichtlich gegenübergestellt werden.

Verwendete Größen: k ist die Anzahl der k-kürzesten Wege
$N = |V|$ und $M = |E|$

Algorithmus	Art	Ausführung	Aufwand		Bemerkungen
			mit Schleifen	ohne Schleifen	
Dijkstra	single-source short. path	N:1	-	N^2	Original-Dijkstra
Dijkstra	single-source short. path	N:1	-	MlogN	mit Heap als Rand
Naiver Alg.	single-pair k-short. paths	1:1	Nk	-	exp. Aufwand
Floyd/Minieka	all-pairs k-short. paths	N:N	$Nk^3 + N^3k^2$	-	festes k für alle Knoten
Floyd/Minieka	all-pairs k-short. paths	N:N	$Nklog^2k + N^3klogk$	-	nach Lawler
Hoffman & Pavley	single-pair k-short. paths	1:1	N^2k	?	
Hoffman & Pavley	single-pair k-short. paths	N:N	N^4k	?	
Azevedo et. al.	single-pair k-short. paths	1:1	$N^2 + Mk$	?	
Azevedo et. al.	single-pair k-short. paths	N:N	$N^3 + N^2Mk$	?	
Clarke	single-pair k-short. loopless p.	1:1	-	?	Aufwand nicht berech.
Yen	single-pair k-short. loopless p.	1:1	-	N^3k	
Yen	single-pair k-short. loopless p.	N:N	-	N^5k	

Der Algorithmus von Yen ist der einzige, dessen Aufwand für die k-kürzesten schleifenfreien Wege abschätzbar ist. Aus diesem Grund ist er am besten für die Implementierung geeignet.
Zusätzlich wäre es interessant, an Hand einer Implementierung von Hoffman & Pavley oder Azevedo, zu sehen, wie viele Wege insgesamt berechnet werden müssen, um darunter k schleifenfreie Wege zu erhalten.
Die Anzahl der Wege mit Schleifen ist bei dem derzeitigen Graphenmodell mit Knotensplittung sicher nicht gering, da in fast jeder Haltestelle umgestiegen und sofort wieder zurück auf die alte Linie umgestiegen werden kann.

4.12 Wegecheck

Im letzten Abschnitt dieses Kapitels soll geschildert werden, wie die berechneten Wege nach Beendigung der Wegsuche wieder vom Graphen zurück auf das Verkehrsnetz abgebildet werden.

Die einzelnen Wege liegen als Folge von Knoten vor: $p = (v_1, v_2, \ldots, v_j)$

Zu jedem einzelnen Knoten wurde bei der Erstellung des Graphen gespeichert, ob er einen Bezirkspunkt oder einen Teil einer Haltestelle im Verkehrsnetz repräsentiert. Mit diesen Informationen ist es möglich, aus zwei aufeinanderfolgenden Knoten v_i, v_{i+1} des Weges, Rückschlüsse auf die entsprechende Kante im Verkehrsnetz zu erhalten:

	v_i	v_{i+1}	
Fall 1	BEZ	HS	= Fußweg vom Bezirk zur Haltestelle
Fall 2	HS	BEZ	= Fußweg von der Haltestelle zum Bezirk
Fall 3	HS_a	HS_a	= Umsteigekante innerhalb der Haltestelle a
Fall 4	HS_a	HS_b	= Linienkante von Haltestelle a zur Haltestelle b

Die Abbildung des Graphenweges auf den Verkehrsweg ist einfach und unkompliziert, wie eben geschildert.
Das größere Problem jedoch ist die Tatsache, daß nicht alle gefundenen Wege korrekte Verkehrswege sind. Daher die Überschrift dieses Abschnitts: Wegecheck. Das heißt, nach Beendigung des k-Wege-Algorithmus erfolgt eine Kontrolle (der Wegecheck) aller berechneten Wege und erst danach werden die übrig gebliebenen Wege an die Umlegung (siehe nächstes Kapitel) weitergereicht.
Was sind das für Wege, die vom k-Wege-Algorithmus korrekt ermittelt werden, aber trotzdem keine sinnvollen Wege im Verkehrsnetz sind? Es lassen sich zwei Typen unterscheiden:

Typ A: Durch die Knotensplittung bei der Bildung des Distanzgraphen entstehen wesentlich mehr Knoten, als das zugrundeliegende Verkehrsnetz eigentlich hat. Dadurch kann es im Graphen korrekte Wege geben, die im Verkehrsnetz gar nicht möglich sind.

Typ B: Dies sind Wege, die sowohl im Graphen, als auch im Verkehrsnetz korrekt und möglich sind, aber aus Sicht des Verkehrswesens sind sie nicht sinnvoll. Daß heißt, kein Fahrgast würde einen solchen Weg benutzen.

Die Wege von Typ A müssen unbedingt durch den Wegecheck eliminiert werden, da sie nicht vom Verkehrswesen, sondern durch die Abbildung auf den Distanzgraphen verursacht werden. Weil dieses Problem erst durch diese Arbeit entsteht, muß daher auch für die vollständige Lösung des Problems gesorgt werden.
Etwas differenzierter verhält sich die Sache bei den Wegen vom Typ B. Hier liegt das Problem auf der Seite des Verkehrswesens. Darüber, welche Wege im Verkehrsnetz sinnvoll bzw. nicht sinnvoll sind - welche Weg also von den Fahrgästen verwendet werden und welche nicht - läßt sich mit Verkehrswissenschaftler ausgiebig diskutieren. Dieses Problem wird etwas später noch einmal aufgegriffen.

Zunächst einige Beispiele für Wege vom Typ A:

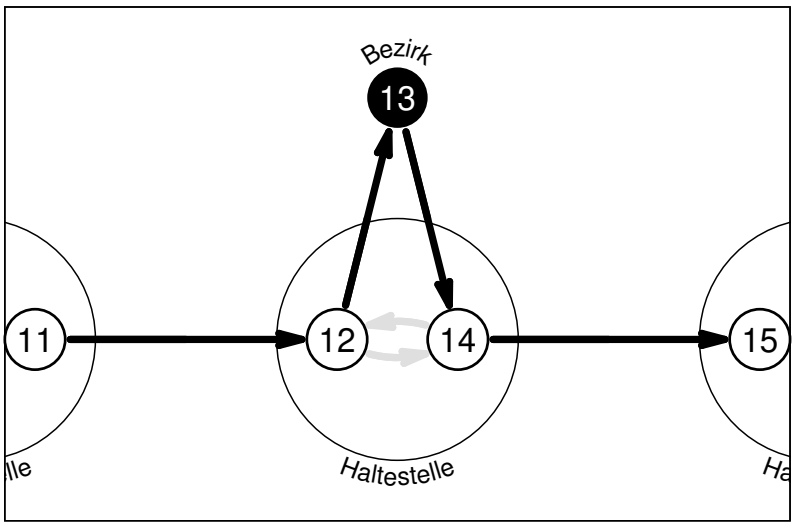


Abb. 4.5: Nicht sinnvoller Weg (Ausschnitt)

Die Abbildung 4.5 zeigt einen Ausschnitt aus einem korrekten Weg im Graphen. Der Weg kommt über die Knoten 11 und 12 zu einer neuen Haltestelle, geht von dort über die Fußwegkanten (12, 13) zu einem Bezirk, der an die Haltestelle angebunden ist. Dieser Bezirk ist nicht der Zielbezirk.
Anschließend führt der Weg über die Fußwegkante (13, 14) zurück zur Haltestelle, von wo der Weg mit einer anderen Linie in Richtung Knoten 15 führt.
Dieser Weg ist zwar im Graphen korrekt, jedoch nicht aus Sicht des Verkehrswesens. Fahrgäste steigen auf ihrem Weg nicht in eine andere Linie um, in dem sie zu Fuß zu einem Bezirkspunkt laufen und wieder zurück zur Haltestelle. Da Fahrtunterbrechungen, bei denen eine andere Person „besucht“ wird, nicht vorgesehen sind, sind diese Fußwege nicht sinnvoll. Es sollte daher nur die Umsteigekante 12, 14) verwendet werden um vom Knoten 12 zum Knoten 14 zu gelangen.
Fußwege sind demnach nur am Anfang und am Ende eines Weges zugelassen. Damit diese Forderung auch vom k-Wege-Algorithmus eingehalten wird, werden alle Fußwegkanten bei der Erstellung des Distanzgraphen zunächst nicht eingebunden. Erst wenn eine konkrete Berechnung der Wege von einem Bezirk A zu einem Bezirk B ansteht, werden vor der Berechnung alle Fußwege der Bezirke A und B in den Graphen eingebunden. Anschließend erfolgt die Wegberechnung und danach werden diese Fußwege wieder aus dem Graphen gelöscht. Eine Prüfung durch den Wegecheck ist hier daher nicht erforderlich.

Einen weiteren Fall vom Typ A zeigt die Abbildung 4.6:

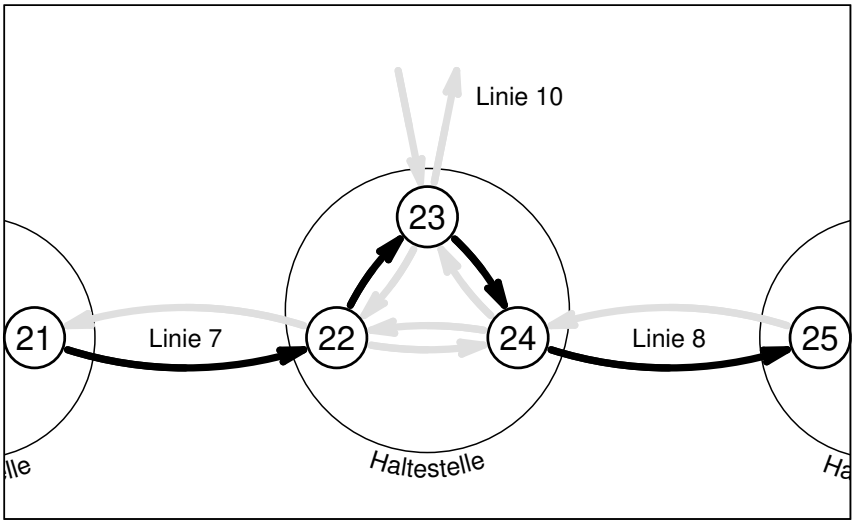


Abb. 4.6: Zweifaches Umsteigen hintereinander

Der Weg verläuft von Knoten 21 mit der Linie 7 zu Knoten 22. Dort wird innerhalb der Haltestelle erst zum Knoten 23 und dann zum Knoten 24 umgestiegen. Anschließend wird der Weg mit Linie 8 in Richtung Knoten 25 fortgesetzt.
Dieses zweimalige Umsteigen innerhalb der Haltestelle ist unsinnig, da es eine direkte Umsteigekante vom Knoten 22 zum Knoten 24 gibt und außerdem länger (über Knoten 23), da zusätzlich die in der Kante (22, 23) enthaltene Einsteigewartezeit für Linie 10 beachtet werden muß.

Da es Haltestellen gibt, an denen mehr als 3 Linien ankommen bzw. abfahren, ist es auch möglich, Wege mit mehr als zwei Umsteigekanten hintereinander zu bilden. Daher müssen Vorkehrungen getroffen werden, damit nur Wege gefunden werden, in denen auf jeder Umsteigekante eine Linienkante folgt. Diese Kontrolle braucht ebenfalls nicht durch den Wegecheck zu erfolgen, sondern kann direkt in den k-Wege-Algorithmus eingebaut werden. Dort erhält jede Umsteigekante ein besonderes Flag und wenn während der Berechnung der aktuelle Weg mit einer Umsteigekante endet, darf er mit einer weiteren Umsteigekante fortgesetzt werden (Diese Methode ist in den implementierten Yen-Algorithmus eingebaut worden).

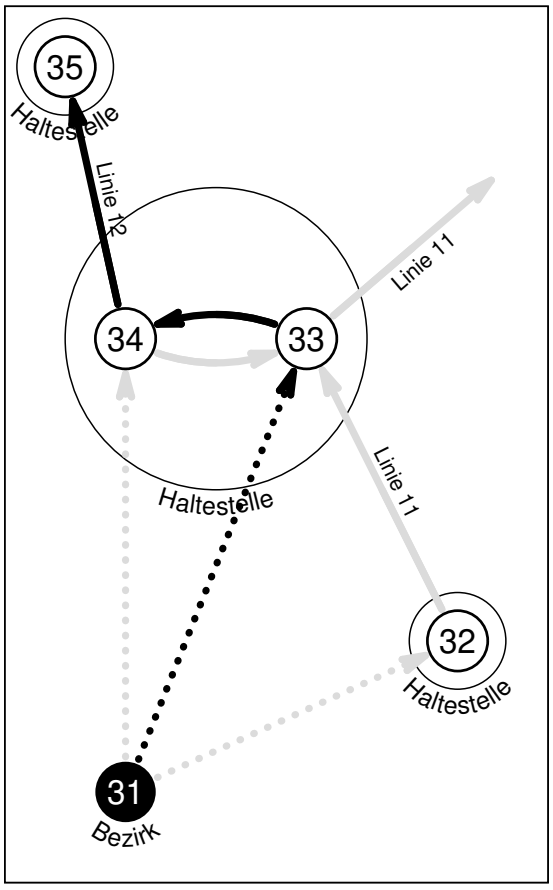


Abb. 4.7: Umsteigen nach Fußweg

Die letzte Möglichkeit eines Weges vom Typ A zeigt Abbildung 4.7:
Ein Bezirk (Knoten 31) ist mit Fußwegen an zwei Haltestellen angebunden.
Hier ist es möglich, ähnlich wie im vorherigen Fall, daß ein Weg vom Knoten 31 aus zuerst den Fußweg zum Knoten 33 nimmt, dort zum Knoten 34 umsteigt und dann die Linie 12 zum Knoten 35 verwendet. Auch dieser Weg ist unsinnig, da es einen direkten Fußweg zum Knoten 34 gibt.
In diesem Fall jedoch können Wege dieses Typs nicht direkt im k-Wege-Algorithmus abgefangen werden. Es ist zwar sehr einfach diese Wege zu erkennen, eine Löschung würde jedoch dazu führen, daß andere Wege nicht mehr gefunden würden. Der Weg 31-32-33-34-35 zum Beispiel, ist ein korrekter und sinnvoller Weg. Er würde aber vom Algorithmus nicht mehr gefunden werden, wenn zuvor nicht der Weg 31-33-34-35 berechnet wird. Denn nur von diesem Weg aus kann durch Einbau von Umwegen oder neuen Kanten der Weg 31-32-33-34-35 gefunden werden. Daher können Wege dieser Art erst nach der Berechnung gelöscht werden.
Das gleiche trifft ebenso auf Wege zu, die umgekehrt kurz vor der letzen Fußwegkante zum Zielbezirk noch eine Umsteigekante enthalten.
Deshalb wird nach der Wegberechnung ein Wegecheck notwendig, der alle Wege löscht, die als zweite oder vorletzte Wegkante eine Umsteigekante besitzen.

Viel schwieriger ist die Bestimmung der nicht sinnvollen Verkehrswege vom Typ B. Darunter sind Wege zu verstehen, die im Verkehrsnetz möglich sind, aber von keinem Fahrgast benutzt werden, weil sie für nicht sinnvoll erachtet werden. Dazu ein Beispiel:

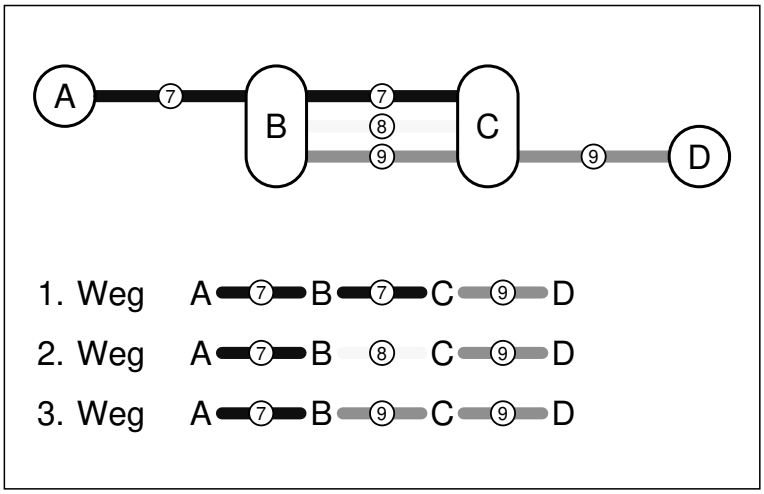


Abb. 4.8: Ein kleines Netz und mögliche Wege

In Abbildung 4.8 ist ein kleines Netz mit den vier Haltestellen A, B, C, D und den drei Linie 7, 8 und 9 zu sehen (alle drei Linien sind unterschiedliche Verkehrsmittel). Darunter sind die drei möglichen Wege von A nach D abgebildet.
Da keine Linie durchgehend von A nach D fährt, muß der Fahrgast demzufolge mindestens einmal in eine andere Linie umsteigen.
Die Frage ist, ob Weg Nr. 2 einen sinnvollen Weg in diesem Netz beschreibt. Auf den ersten Blick erscheint der zweite Weg nicht besonders attraktiv, da er ein zweimaliges umsteigen erfordert.

Wenn aber andererseits zum Beispiel die Linie 7 eine langsame Buslinie, Linie 9 eine langsame Straßenbahn und Linie 8 eine schnelle S-Bahn wäre, so könnte sich Weg 2 als der schnellste der drei Wege herausstellen und würde dann mit Sicherheit von den Fahrgästen angenommen werden. Es ist daher nicht so klar ersichtlich, welche Wege sinnvoll sind und welche nicht.

Ein zweites Beispiel aus dem realen Verkehrsnetz von Stuttgart:

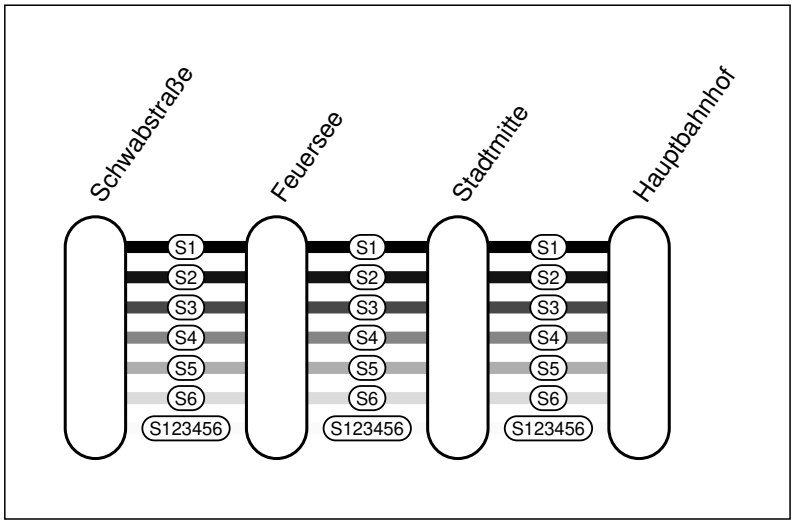


Abb. 4.9: Ausschnitt aus dem Stuttgarter Netz

Eine Person möchte von der Schwabstraße zum Hauptbahnhof fahren. Dazu stehen ihr die sechs S-Bahnlinien S1 bis S6 zur Verfügung. Da alle S-Bahnen durchgehend bis zum Hauptbahnhof fahren, ist es egal mit welcher Linie die Person fährt. Deshalb wurde die Parallelverkehrslinie S123456 zusätzlich aufgeführt (Sie symbolisiert die Verwendung der nächsten S-Bahn, die an der Schwabstraße abfährt).
Der schnellste Weg von der Schwabstraße zum Hauptbahnhof ist demnach die Linie S123456.
Es stellt sich nun die Frage, ob die restlichen, möglichen Wege überhaupt sinnvoll sind. Zum Beispiel der Weg:

Fahrt von Schwabstraße nach Feuersee mit S4,
Fahrt von Feuersee zur Stadtmitte mit S1 und
Fahrt von Stadtmitte zum Hauptbahnhof mit S6.

Meiner Meinung nach, findet sich kein Fahrgast, der diesen Weg benutzen würde. Das heißt, dieser Weg und alle anderen Kombinationsmöglichkeiten der S-Bahnlinien sind unter keinen Umständen Alternativwege zur durchgehenden Benutzung der Linie S123456. Der einzig sinnvolle Weg von der Schwabstraße zum Hauptbahnhof ist in diesem Fall die Linie S123456 und sonst nichts. Der k-Wege-Algorithmus sollte in daher von allen $7^3 = 343$ möglichen Wegen nur diesen einen Weg liefern und keinen weiteren.

Die beiden bisher gezeigten Beispiele für Wege vom Typ B sollen vor allem zwei Tatsachen darlegen:

I. Besonders in den Kernbereichen von Netzen, wo viele Linien parallel verkehren, wächst die Anzahl der möglichen Wege rapide an und kann sehr große Ausmaße annehmen (Während der Laufzeitmessungen ergaben sich bis zu 30000 Wege zwischen zwei Bezirksknoten im Netz von Ulm!).

II. Die Entscheidung, welcher Weg in einem Verkehrsnetz sinnvoll ist und welcher nicht, ist nicht trivial. In diesem Bereich sollte es zusätzliche Untersuchungen des Fahrverhaltens von Personen geben, um festzustellen, welche Wege die Fahrgäste als Alternativwege definieren und annehmen.

In dieser Arbeit habe ich nur die im zweiten Beispiel mit dem Stuttgarter Netz angesprochenen Wege durch einen Wegecheck eliminiert. Alle weitergehenden Entscheidungen über sinnvolle und unsinnige Wege bedürfen einer genaueren Untersuchung und Abwägung.

Die genaue Anforderung an den implementierten Wegecheck nach der Wegberechnung lautet demnach:

Eingangsdaten: Eine Menge von Wegen, die von einem Bezirk X zu einem Bezirk Y verlaufen. Diese wurden vom k-Wege-Algorithmus berechnet.

Ausgangsdaten: Die Eingangsdaten, wobei folgende Wege aus der Menge eliminiert wurden:

- Jeder Weg p, der als zweite oder vorletzte Wegkante eine Umsteigekante beinhaltet (p ist vom Typ A).
- Jeder Wege p, dessen Gewicht größer ist als das eines anderen Weges q, wobei die Menge aller LINKs der Kanten von p eine echte Teilmenge der Menge aller LINKs der Kanten von q sein muß (q ist vom Typ B).

5. Umlegung

5.1 Belastungsfaktoren der Wege

Wenn der „k-Wege-Algorithmus“ die entsprechende Menge an Wegen für ein Bezirkspaar „A nach B“ ermittelt hat, kann jetzt die eigentliche Umlegung beginnen. Dazu wird die Anzahl von Fahrten, die täglich von A nach B erfolgen, auf die errechneten Wege verteilt.
Bei dieser Verteilung spielen die Gewichte des Bestweges (-wege) und des schlechtesten Weges (Wege) innerhalb dieser Menge eine besondere Rolle:

Bestweg = Weg mit dem niedrigsten Gewicht in der Menge

schlechtester Weg = Weg mit dem größten Gewicht in der Menge

Es kann durchaus mehrere Bestwege geben, die dann alle das gleiche Gewicht haben. Dieser Fall hat aber keine Auswirkungen auf die Umlegungsberechnung, da jeweils nur die Gewichte der Wege relevant sind und nicht die einzelnen Wege. Das gleiche gilt für den schlechtesten Weg/ Wege.

Die Verteilung soll nun gerade so erfolgen, daß der Bestweg genau vier mal so viele Fahrten erhält, wie der schlechteste Weg. Alle übrigen Wege (Gewichte der Wege) dazwischen erhalten Fahrten entsprechend linear verteilt.

Belastung(Bestweg) = 4 x Belastung(schlechtester Weg)

Diese „4 zu 1“-Verteilung habe ich als Vorgabe erhalten, da sie die Akzeptanz der Fahrgäste für den Bestweg und die kaum größeren Alternativen gegenüber den anderen Alternativen, die zwar ebenfalls noch innerhalb der Grenze liegen, aber doch ein weit größeres Gewicht haben, ausdrücken soll. Sie wird seit mehreren Jahren so eingesetzt und verwendet.
Denkbar wäre auch eine einfache, lineare Verteilung entsprechend der einzelnen Weggewichte.

Um diese „4 zu 1“-Verteilung zu realisieren, wird für jeden Weg der Menge ein Belastungsfaktor berechnet, wobei alle Wege mit dem geringsten Gewicht innerhalb der Menge (Bestwege) den Faktor 4.0 und alle Wege mit dem größten Gewicht (schlechteste Wege) den Faktor 1.0 erhalten. Alle übrigen Wege erhalten entsprechend ihren Gewichten einen Faktor linear dazwischen:

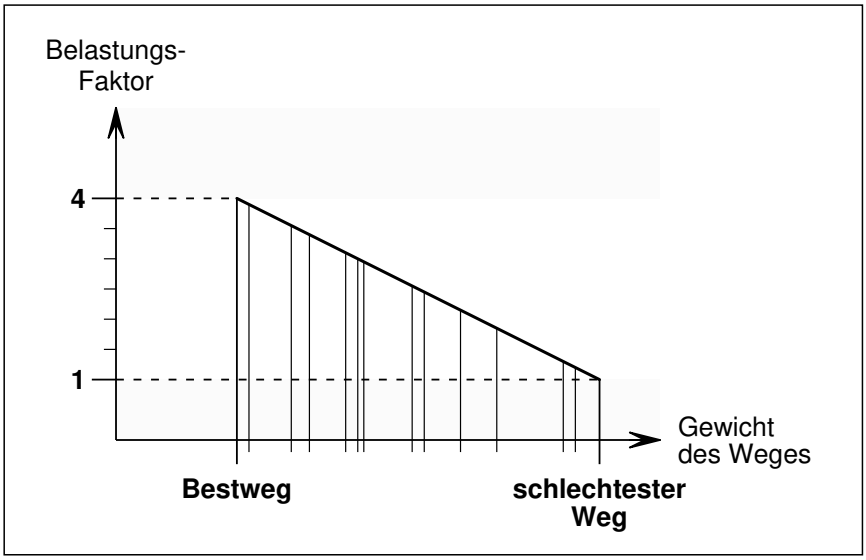


Abb. 5.1: ’4 zu 1’-Berechnung der Belastungsfaktoren

Die Gesamtbelastung b_{ges} des öffentlichen Verkehrs von Bezirk A nach Bezirk B wird dann über diese Belastungsfaktoren prozentual verteilt und ergibt die einzelne Belastung b_i jedes Weges.

Berechnung der Einzelbelastungen:

Gegeben sei: k Wege von Bezirk A nach Bezirk B

$g_1, g_2, \ldots, g_k$	Gewicht der Wege
$g_{\min} = \min\{g_1, g_2, \ldots, g_k\}$	kleinstes Weggewicht
$g_{\max} = \max\{g_1, g_2, \ldots, g_k\}$	größtes Weggewicht
b_{ges}	Gesamtbelastung von A nach B (Fahrten)

Gesucht: Einzelbelastungen $b_1, b_2, \ldots, b_k$ der Wege

Berechnung: Zunächst werden aus den Weggewichten die „4 zu 1"-verteilten Belastungsfaktoren $z_1, z_2, \ldots, z_k$ ermittelt:

$$z_i = \frac{g_i - g_{\max}}{g_{\min} - g_{\max}} \cdot 3 + 1 \qquad \text{für } i = 1 \ldots k$$

Aus diesen Belastungsfaktoren z_i wird für jeden Weg ein Prozentfaktor p_i berechnet (prozentuale Verteilung der z_i):

$$p_i = \frac{z_i}{\sum_{j=1}^{k} z_j} \qquad \text{für } i = 1 \ldots k$$

Damit läßt sich aus der Gesamtbelastung b_{ges} von Bezirk A nach B pro Tag, die einzelnen Belastungen b_i für jeden Weg i berechnen:

$$b_i = p_i \cdot b_{ges} \qquad \text{für } i = 1 \ldots k$$

Beispiel

Die Gesamtbelastung b_{ges} beträgt 2500 Fahrten/Tag.

Der „k-Wege-Algorithmus" hat 5 Wege mit den unten angegebenen Gewichten ermittelt.
Damit ergeben sich folgende Werte:

Weg i	Gewicht g_i	Belastungsfaktor z_i	Prozentanteil p_i	Einzelbelastung b_i
1	150,0	4,0	0,36	900
2	478,9	2,667	0,24	600
3	588,5	2,222	0,20	500
4	835,2	1,222	0,11	275
5	890,0	1,0	0,09	225
				Σ 2500

Man sieht, daß der Weg 1 (Bestweg) genau vier mal so viele Fahrten wie Weg 5 (schlechtester Weg) erhält.

5.2 Mandatsverteilung

Anders als in dem eben gezeigten Beispiel, erhält man üblicherweise für die Einzelbelastungen b_i der Wege keine natürlichen Zahlen, sondern Kommazahlen.
Da aber auf einem Weg anschaulich nicht (z.B.) 54,7 Fahrten stattfinden können, sonder entweder 54 oder 55 Fahrten, bieten sich hier zwei Lösungsmöglichkeiten an:

- **runden**
  Die Kommazahlen werden auf- bzw. abgerundet und man nimmt in Kauf, bei jedem Weg einen Fehler von plus/minus einer Fahrt zu produzieren.

oder

- **Mandatsverteilung**
  Es zeigen sich folgende Parallelen zu Wahlen:

  Prozentanteil p_i $\Leftrightarrow$ Stimmenanteil für Partei X
  Einzelbelastung b_i $\Leftrightarrow$ Abgeordnetenmandate für Partei X

  Das heißt, man greift auf das bekannte Verteilungsproblem bei Wahlen zurück.
  Hierzu gibt es mehrere Ansätze, die in [Schedel 90] beschrieben sind.
  In dieser Arbeit wurde das Divisorverfahren von Webster/St. Laguë angewendet, da es sich nach [Schedel 90] für diese Aufgabe am besten eignet.

Divisorverfahren nach Webster/St. Laguë

Gegeben:	M_{ges}	Gesamtanzahl der zu vergebenden Mandate
	k	Anzahl Parteien
	$s_1, ..., s_k$	Anzahl Stimmen, die die jeweilige Partei i erhalten hat
Gesucht:	$m_1, ..., m_k$	Anzahl der Mandate der jeweiligen Partei i
Berechnung:	$f_1, ..., f_k$	Quotientenfaktoren
	$w_1, ..., w_k$	aktuelle Stimmenanzahl
	g	Anzahl der bereits vergebenen Mandate

Zunächst werden alle Quotientenfaktoren f_i auf 1 gesetzt und alle $w_i = s_i$ gesetzt:

```
for i = 1 to k do
   w(i) := s(i)
   f(i) := 1
end i
g := 0
```

Nun wird für jedes einzelne Mandat eine Schleife ausgeführt:

```
while g < Mges do
   nr = 0
   m = 0
   for j = 1 to k do                    Ermittlung des momentan größten w_nr
      if w(j) > m then  m := w(j); nr := j
   end j
   f(nr) := f(nr) + 1                   Diese Partei nr erhält ein Mandat.
   w(nr) := s(nr) / (f(nr) * 2 - 1)     Ihre aktuelle Stimmenzahl
   g := g + 1                           wird verkleinert.
end while
```

Zum Schluß muß jedes f_i um 1 vermindert werden:

```
for i := 1 to k do
   f(i) := f(i) - 1
end i
```

Jetzt sind alle Mandate verteilt und die f_i geben an, wieviele Mandate die Partei i erhält.
Der Aufwand für dieses Verfahren beträgt O($M_{ges} \cdot k$).

Divisorverfahren zur Mandatsverteilung beruhen darauf, daß immer die Partei mit der aktuell größten Stimmenanzahl (w_i) ein Mandat erhält und dann ihre Stimmenanzahl durch einen Quotienten verkleinert wird. Erst wenn diese neue Stimmenanzahl wiederum die größte von allen anderen ist, erhält diese Partei wieder ein Mandat.
Bei dem Verfahren von Webster/St. Laguë werden die Quotienten, durch die geteilt wird, durch die ungeraden Zahlen (1, 3, 5, ...) gebildet.
Der aktuelle Quotient einer Partei wird in der obigen Berechnung indirekt in f_i gespeichert:

aktueller Quotient = $f_i \cdot 2 - 1$

Dadurch läßt sich in f_i auch gleichzeitig die Anzahl der Mandate für die Partei i speichern.

Beispiel

5 Parteien, M_{ges} = 50

Partei i	1	2	3	4	5
Stimmen s_i	13987	14500	8634	1460	9875

Berechnungsablauf:

		Partei 1		Partei 2		Partei 3		Partei 4		Partei 5	
Schritt	g	w_1	f_1	w_2	f_2	w_3	f_3	w_4	f_4	w_5	f_5
	0	13987	1	14500	1	8634	1	1460	1	9875	1
1	0			erhält 1 Mandat							
	1	13987	1	4833,33	2	8634	1	1460	1	9875	1
2	1	erhält 1 Mandat									
	2	4662,33	2	4833,33	2	8634	1	1460	1	9875	1
3	2									erhält 1 Mandat	
	3	4662,33	2	4833,33	2	8634	1	1460	1	3291,67	2
	48	482,31	15	500,00	15	454,42	10	486,67	2	470,24	11
49	48			erhält 1 Mandat							
	49	482,31	15	467,74	16	454,42	10	486,67	2	470,24	11
50	49							erhält 1 Mandat			
	50	482,31	15	467,74	16	454,42	10	292,00	3	470,24	11

Mandate	14	15	9	2	10

Dieses Ergebnis entspricht dem einer prozentualen Verteilung mit Rundung:

Divisorverfahren:

Mandate	14	15	9	2	10

Prozentuale Verteilung mit Rundung:

Stimmen	13987	14500	8634	1460	9875
Stimmenanteil	28,87 %	29,92 %	17,82 %	3,01 %	20,38 %
Mandate	14	15	9	2	10

Dieses Divisorverfahren kann direkt für die Berechnung der Einzelbelastungen der Wege übernommen werden, wenn man folgende Variablen ersetzt:

$$M_{ges} \quad \text{durch} \quad b_{ges}$$
$$s_i \quad \text{durch} \quad p_i$$
$$f_i \quad \text{durch} \quad b_i$$

5.3 Umlegung auf Haltestellen und Linien

Sind die Einzelbelastungen für jeden Weg ermittelt, müssen diese jetzt auf die jeweils benutzten Haltestellen und Linien verteilt werden.
Dazu sind an jeder Haltestelle bzw. Linie folgende Größen von Interesse:

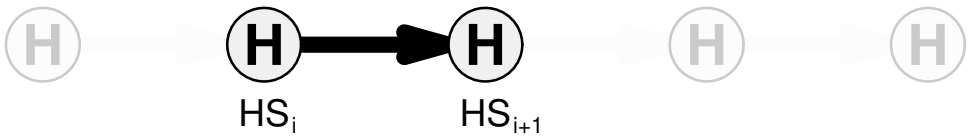
Für jede Haltestelle

Einsteiger Anzahl Fahrten, die zu Fuß an diese Haltestelle geführt haben und ab hier mit einer Verkehrslinie weitergefahren sind

Aussteiger Anzahl Fahrten, die mit einer Verkehrslinie zu dieser Haltestelle gefahren sind und von hier zu Fuß weitergeführt haben

Umsteiger Anzahl Fahrten, die in dieser Haltestelle von einer in eine **andere** Verkehrslinie gewechselt haben

Für jeden Abschnitt einer Linie



Mitfahrer Anzahl Fahrten, die diesen Linienabschnitt (von HS_i nach HS_{i+1}) dieser Linie benutzt haben

Aussteiger diejenigen Fahrten unter den „Mitfahrern“, die nach diesem Linienabschnitt an der nächsten Haltestelle (HS_{i+1}) auf eine andere Linie gewechselt bzw. einen Fußweg benutzt haben.

Für jeden Weg und seine Einzelbelastung wird bestimmt, welche Haltestellen und Linien er verwendet und seine Belastung auf diese Elemente aufaddiert.

Vorgehensweise

Gegeben: Weg als Folge von Graphenknoten $(v_1, v_2, ..., v_m)$
Einzelbelastung b dieses Weges

Betrachtung der einzelnen Knotenpaare: (v_i, v_{i+1}) für alle $i = 1,...,m-1$
Für beide Graphenknoten wird bestimmt, welches Objekt im Verkehrsnetz sie repräsentieren:

Haltestelle (HS) oder Bezirkspunkt (BEZ)

Hierfür werden vier Fälle unterschieden:

	v_i	v_{i+1}		
Fall 1	BEZ	HS	=	Fußweg zur Haltestelle; die Fahrt wird dort mit einer Linie fortgesetzt: → Einsteiger-Größe der Haltestelle HS um b erhöhen
Fall 2	HS	BEZ	=	Fußweg zum Bezirkspunkt; die Fahrt mit einer Linie wird an der Haltestelle beendet und zu Fuß fortgesetzt: → Aussteiger-Größe der Haltestelle HS um b erhöhen
Fall 3	HS_i	HS_i	=	Fahrt wechselt innerhalb der Haltestelle auf eine andere Linie um: → Umsteiger-Größe der Haltestelle HS_i um b erhöhen
Fall 4	HS_i	HS_j	=	Fahrt von Haltestelle HS_i zur Haltestelle HS_j ($i \neq j$) mit einer Linie: → Mitfahrer-Größe dieses Linienabschnitts um b erhöhen Wenn in diesem Fall ein Knoten v_{i+2} im Weg vorhanden ist und dieser Knoten v_{i+2} eine andere Linienkombination besitzt als v_{i+1}, wird die Fahrt an der Haltestelle HS_j mit der aktuellen Linie beendet und eine Linie verwendet: → Aussteiger-Größe dieses Linienabschnitts um b erhöhen

Wenn im Fall 4 der befahrene Linienabschnitt (Kante im Graph) keine einzelne Linie, sondern eine Linienkombination ist, muß die Belastung b des aktuellen Weges auf alle Linien dieser Linienkombination aufgeteilt werden.

Beispiel: Der befahrene Linienabschnitt besteht aus einer Linienkombination von 3 S-Bahnen, so muß die Belastung b von 600 Fahrten/Tag auf alle 3 Linien aufgeteilt werden: jede S-Bahn erhält eine Belastung von 200 Fahrten/Tag.

6. Implementierung

6.1 Programmaufbau und - ablauf

Das Programm ist in Borland Pascal 7.0 für Windows geschrieben. außer dem Speichermanagement wurden aber keine besonderen Eigenschaften des Betriebssystems Windows verwendet. Deshalb kann das Programm relativ einfach auf andere Pascal-Systeme übertragen werden. Windows wurde nur verwendet, da es hier leichter ist, größere Speicherblöcke für Daten zu reservieren.

Für die wenigen Ausgaben von Informationen auf dem Bildschirm, wird ein spezielles Fenster verwendet, das eine zeichenorientierte Ausgabe von Text ermöglicht, denn eine zeichenorientierte Ausgabe wird von allen Pascal-Systemen unterstützt und benötigt keine Funktionen von Windows.
Eine spezielle Kommunikationsschnittstelle zwischen Benutzer und Programm wurde nicht ausgearbeitet, da dies nicht Teil der Arbeit ist und mit dem Institut für Eisenbahn- und Verkehrswesen so vereinbart wurde. Veränderungen der Programmparameter müssen daher direkt im Programmcode erfolgen (mehr dazu im Abschnitt 6.4).

Das programm ist übersichtlich in mehrere Module aufgeteilt, welche jeweils die Daten und Funktionen eines entsprechenden Teilgebietes des Themas umfassen. Die genaue Aufteilung und Beschreibung der Module erfolgt in Abschnitt 6.2.

Der Programmumfang in Zahlen:

4973	Lines of Code
9	Module (Hauptprogramm + 8 Units)
79	Funktionen und Prozeduren

Als k-Wege-Algorithmus wurde der Algorithmus von Yen implementiert. Dies erfolgte ebenfalls in einem separaten Modul, sodaß der Algorithmus mit wenig Aufwand gegen einen anderen Algorithmus ausgetauscht werden kann. Mit dem Yen-Algorithmus wurden auch alle Laufzeitmessungen (Kapitel 7) durchgeführt. Eine Implementation und Laufzeitmessung von anderen Algorithmen zum Vergleich wäre wünschenswert und sinnvoll, ließ sich aber aus Zeitgründen leider nicht durchführen.

Auf Grund der großen Datenmengen ist die Laufzeit des Programms von großer Bedeutung. Daher wurde bei vielen Datenstrukturen die Prinzipien von Kapselung und Zugriffsschutz zu Gunsten der Effizienz nicht in vollem Umfang verwirklicht.

Zusätzlich zur Mehrwegumlegung wurden zahlreiche Funktionen implementiert, mit denen Zwischenergebnisse der Berechnungen, sowie Daten des Netzes und des Graphen, ausgegeben oder protokolliert werden können. Weiterhin wurden Funktionen zur Messung von Ausführungszeiten vorgesehen. Diese zusätzlichen Programmteile werden von der eigentlichen Mehrwegumlegung nicht benutzt. Sie sind aber nötig, um Laufzeitmessungen oder Korrektheitsprüfungen vornehmen zu können.

Die Implementierung der Mehrwegumlegung folgt den inhaltlichen Schritten, wie sie in den vorangegangenen Kapiteln beschrieben wurden. Daraus ergibt sich der Programmablauf in Abbildung 6.1 auf der nächsten Seite:

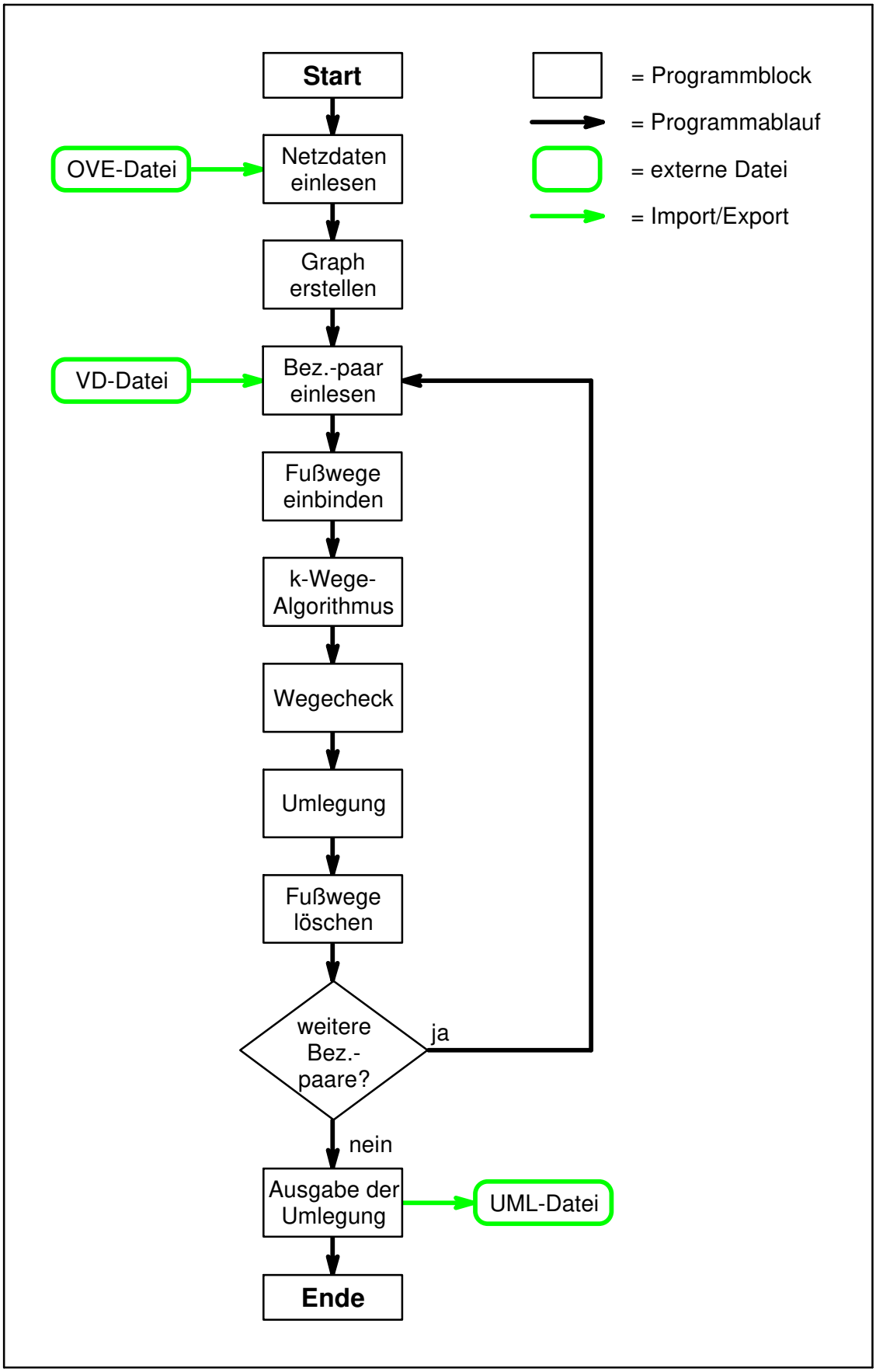


Abb. 6.1: Programmablauf

6.2 Programm-Module

Abbildung 6.2 zeigt die vorhandenen Module des Programms und ihre Verknüpfungen:

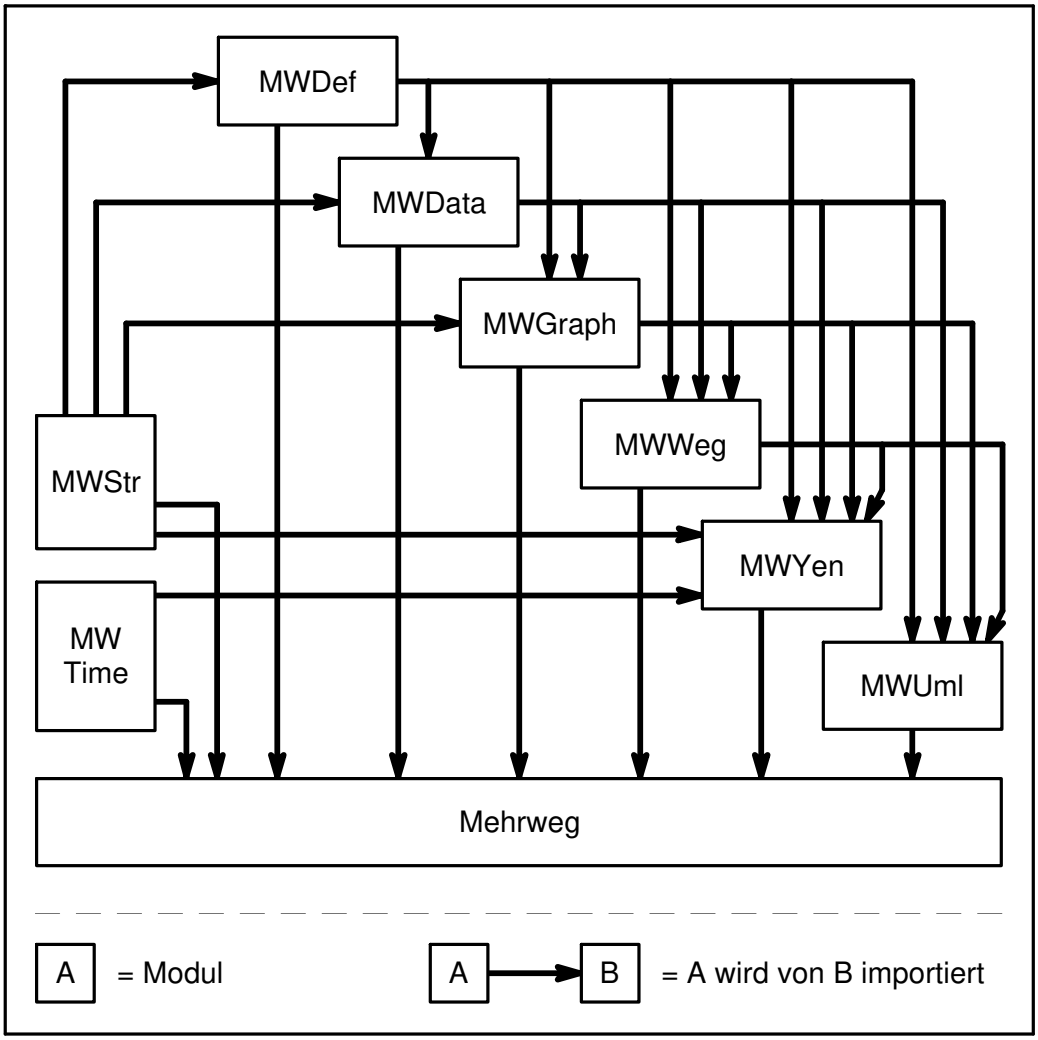


Abb. 6.2: Übersicht über die Module

Das Hauptprogramm wird aus der Datei MEHRWEG.PAS gebildet. Alle anderen Module sind Pascal-Units mit Datenstrukturen und Funktionen für verschiedene Bereiche:

Modul	Inhalt
MWTIME.PAS	Enthält Funktionen mit denen die Laufzeit einzelner Programmblöcke gemessen und ausgegeben werden kann.
MWSTR.PAS	Dies ist eine kleine Unit mit fünf Prozeduren zur Stringverarbeitung.
MWDEF.PAS	Enthält nur Konstanten- und Typendeklarationen. Hier werden die einzelnen Gewichtskonstanten, die Grenzfaktoren und einige Grundtypen definiert.
MWDATA.PAS	Beinhaltet alles, was mit dem Verkehrsnetz zu tun hat (Datenstrukturen & Funktionen für Haltestellen, Linien und Bezirke).
MWGRAPH.PAS	Beinhaltet alles, was mit dem Distanzgraphen zu tun hat (Knoten und Kanten).
MWWEG.PAS	Implementation der Datenstrukturen „Weg“ und „Menge von Wegen“.
MWYEN.PAS	Enthält die Funktionen zur Wegsuche: Dijkstra und Yen.
MWUML.PAS	Enthält die Funktionen für die Verteilung der Belastung auf die Wege.
MEHRWEG.PAS	Das Hauptprogramm.

6.3 Datenstrukturen und Funktionen

In diesem Abschnitt sollen die wichtigsten Datenstrukturen und ihren Funktionen beschrieben werden.

Allgemein ist festzustellen, daß es sechs Datenobjekte gibt, die in einer größeren Anzahl auftreten:

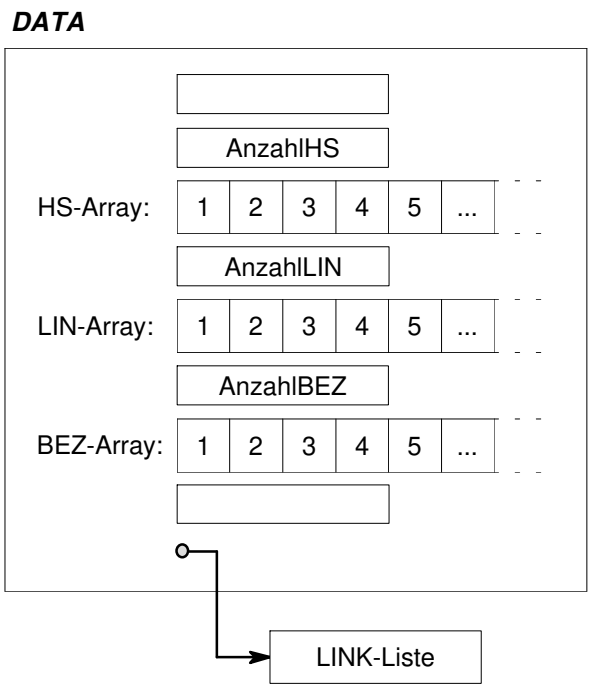
Haltestellen, Bezirke, Linien, Knoten, Kanten und Wege.

Größere Anhäufungen von Objekten, bei denen zudem ihre maximale Anzahl nicht bekannt ist, werden üblicherweise als verkettete Listen gespeichert. Nachteilig hierbei ist jedoch, daß nicht sofort auf die einzelnen Objekte zugegriffen werden kann, sondern erst die Liste ab einem definierten Anfangspunkt durchlaufen werden muß, bis das entsprechende Objekt in Erscheinung tritt.
Deshalb werden die oben genannten Objekte jeweils auch dynamisch gespeichert, aber die Zeiger auf diese Objekte werden in einem Array zusammengefaßt, damit über einen Index direkt und schnell auf die einzelnen Objekte zugegriffen werden kann. Der Nachteil hierbei ist die maximale Anzahl von 16000 Zeigern pro Array.

Im weiteren werden die Datenstrukturen für das Verkehrsnetz, den Graphen und die Wege näher erläutert.

Das Verkehrsnetz

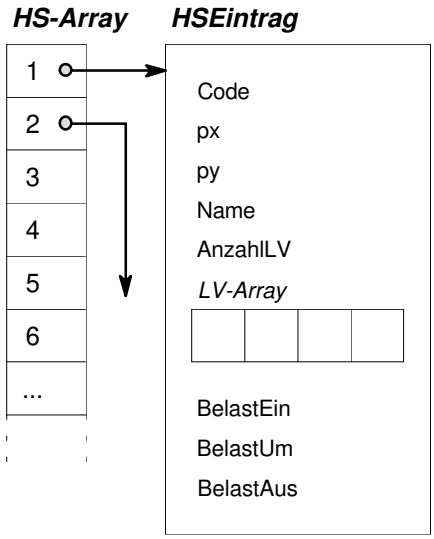
Die Datenstrukturen für das Verkehrsnetz werden durch die Unit MWDATA bereitgestellt:

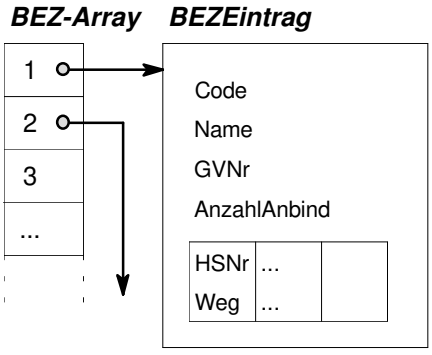


Für das zu untersuchende Verkehrsnetz wird ein Datenblock DATA angelegt. Er enthält neben einigen kleineren Werten drei Arrays mit Zeigern für die Haltestellen, die Linien und die Bezirke des Netzes. Außerdem ist jeweils ein Wert vorhanden, in dem die Anzahl der Objekte gespeichert wird.
Die drei Arrays enthalten nur Zeiger auf dynamische Objekte im Speicher, die dann die eigentlichen Informationen zu jeder Haltestelle bzw. Linie oder Bezirk enthalten.
Zusätzlich ist noch ein Zeiger auf die globale LINK-Liste vorhanden.

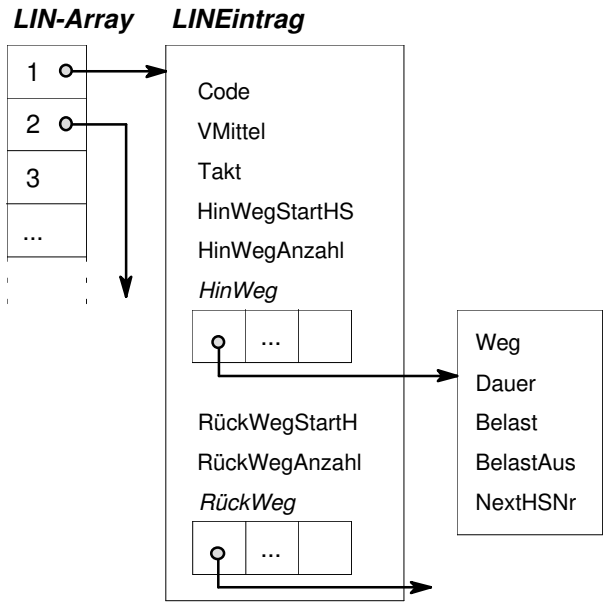
Bei Haltestellen, Linien und Bezirken ist zu beachten, daß jedes Element dieser drei Gruppen eine eindeutige Nummer besitzt. Diese wird im folgenden als Code bezeichnet, da diese Nummern zwar eindeutig, aber nicht fortlaufend sind. Deshalb ist es notwendig den Elementen jeweils noch eine interne Nummer zu geben. Unter dieser Nummer ist der Zeiger auf das Element im entsprechenden Array zu finden.

Das Haltestellenarray enthält wie erwähnt nur Zeiger auf die eigentlichen Haltestellendaten (*HSEintrag*). Zu diesen Daten gehören der Haltestellencode, die geographischen Koordinaten px/py und der Name.
Weiterhin enthält der Datenblock ein Array, in dem später die Knotennummern aller Knoten gespeichert werden, die diese Haltestelle im Graphen repräsentieren, da für jeden LINK der Haltestelle ein Knoten gebildet wird. Die Anzahl der Knoten ist in der Variablen AnzahlLV enthalten.
Ebenfalls enthalten sind die drei Werte für die Einsteiger-, Umsteiger- und Aussteigerbelastung dieser Haltestelle. Diese Werte sind das Ergebnis der Mehrwegumlegung und werden am Ende in der UML-Datei gespeichert.





Ähnlich wie bei den Haltestellen, enthält auch der Datenblock eines Bezirks den Code und den Namen. Da ein Bezirk nur auf einen Knoten im Graphen abgebildet wird, genügt die Variable GVNr für diese Knotennummer.
Auch dieser Datenblock enthält ein Array. Es enthält jeweils die (interne) Nummer und die Weglänge für jede Haltestelle, die an diesen Bezirk angebunden ist. In der Variablen AnzahlAnbind wird die Anzahl der Anbindungen gespeichert.

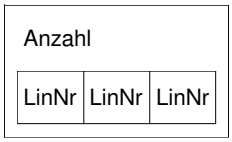


Der Datenblock für eine Linie enthält folgende Informationen: Den Liniencode, den Verkehrsmitteltyp, die Taktzeit sowie die Daten des Hinwegs bzw. des Rückwegs.
Der Hinweg beginnt an derjenigen Haltestelle, deren interne Nummer in HinWegStartHS steht. Für jeden weiteren Linienabschnitt auf dem Hinweg enthält das Array HinWeg einen Zeiger. Mit diesem Zeiger werden die Daten dieses Linienabschnitts erreicht: Der Weg, die Fahrtdauer und die Zielhaltestelle dieses Abschnitts.
Weiterhin enthält jeder Abschnittsblock die beiden Werte Belast und BelastAus für die Belastung des Abschnitts und die an der Zielhaltestelle aussteigenden Personen. Diese Werte werden durch die Mehrwegumlegung erst berechnet.

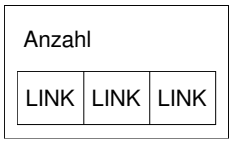
Für den Rückweg der Linie ist die gleiche Struktur wie für den Hinweg vorhanden.

Wie in Abschnitt 3.2.3 definiert, ist ein LINK eine Kombination von Linien. Ein LINK wird benötigt, wenn auf einem Wegabschnitt eine beliebige Linie aus einer Gruppe von Linien benutzt wird. In diesem Fall kann die Gruppe nicht mehr durch eine einzige Liniennummer eindeutig bezeichnet werden. Deshalb wird in der Implementierung an allen Stellen, an denen eine eindeutige Identifikation einer Linie oder einer Liniengruppe benötigt wird, keine Liniennummer, sondern ein LINK verwendet. Das heißt, auch für eine einzelne Linie wird ein LINK verwendet, er besteht dann nur aus einer Liniennummer.
Die Abbildung auf der rechten Seite zeigt den Aufbau eines LINKs: Er enthält eine Variable für die Anzahl der Linien und ein Array, in der die entsprechende Zahl von (internen) Liniennummern eingetragen ist.
Die globale LINK-Liste, die beim Aufbau der Parallelkanten benötigt wird, besteht ebenfalls aus einer Variablen für die Anzahl und einem Array, der alle entsprechenden LINKs enthält.

LINK



LINK-Liste



Die wichtigsten Prozeduren und Funktionen der Unit MWDATA:

```
procedure LoadData(OVEName: string);
```
Diese Prozedur lädt das Verkehrsnetz, dessen Dateiname übergeben wurde, in den Speicher und füllt die entsprechenden Datenstrukturen.

```
function HSNr(HSCode : HSCodeTyp): HSNrTyp;
```
Diese Prozedur ermittelt die interne Nummer der Haltestelle mit dem übergebenen Haltestellencode.

```
function BEZNr(BEZCode : BEZCodeTyp): BEZNrTyp;
```
Diese Prozedur ermittelt die interne Nummer des Bezirks mit dem übergebenen Bezirkscode.

```
function LINKVergleich (LINK1, LINK2, LINKTyp): boolean;
```
Diese Funktion liefert true, wenn die beiden übergebenen LINKs genau die gleichen Liniennummern enthalten, sonst false.

```
procedure LINKSort (var LINK: LINKTyp);
```
Diese Funktion sortiert die Liniennummern des LINK aufsteigend, da sich sortierte LINK wesentlich effizienter Vergleichen lassen.

```
function CheckSubLINK(LINK1, LINK2: LINKTyp): boolean;
```
Diese Funktion liefert true, wenn die Menge der Liniennummern von LINK1 eine Teilmenge der Liniennummern von LINK2 ist, sonst false.

```
prozedure LINKListeAdd(LINK: LINKTyp);
```
Diese Prozedur fügt den übergebenen LINK in die globale LINK-Liste ein.

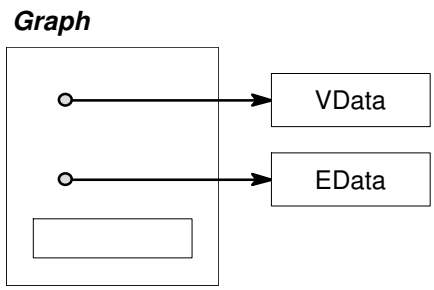
```
procedure DataStatus;
```
Diese Prozedur gibt den aktuellen Status des geladenen Verkehrsnetzes auf dem Bildschirm aus: Anzahl Haltestellen, Linien, Bezirke, benötigter Speicher und anderes. Sie dient nur der Kontrolle und wird von der Mehrwegumlegung nicht benötigt.

```
procedure SaveBelast(UMLName: string);
```
Die Prozedur speichert alle Ergebnisse der Mehrwegberechnung in einer Datei unter dem übergebenen Namen.

```
procedure DisposeData;
```
Diese Prozedur löscht alle Daten des Verkehrsnetzes im Speicher.

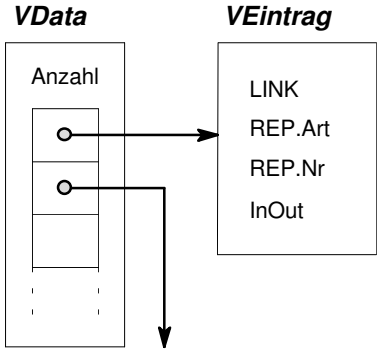
Der Distanzgraph

Alle Datenstrukturen und Funktionen für den Distanzgraphen sind in der Unit MWGRAPH enthalten:

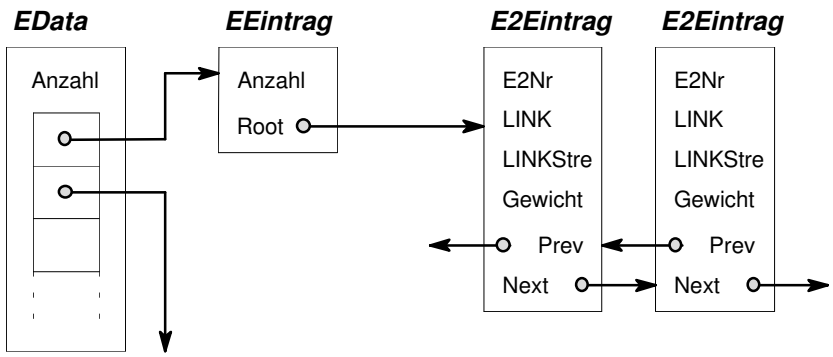


Die Basisstruktur für den Graphen heißt Graph und beinhaltet neben einigen kleineren Werten je einen Zeiger auf die Knoten- und die Kantendaten.

Die Knotendaten haben folgende Struktur:



Der Datenblock VData enthält eine Variable für die Anzahl der Knoten im Graphen und ein Array mit je einem Zeiger für jeden Knoten.
Diese Zeiger führen zu Datenblöcken vom Typ VEintrag. In diesen Datenblöcken stehen die Informationen für jeden Knoten: Der LINK bezeichnet die Linienkombination, die diesen Knoten benutzt. In der mit REP bezeichneten Struktur wird gespeichert, welches Objekt dieser Knoten im Verkehrsnetz darstellt (Haltestelle oder Bezirk) und die entsprechende interne Nummer. Zuletzt besitzt der VEintrag noch ein InOut-Byte: Ist das Bit 1 dieses Bytes gesetzt, so gibt es Linien, die an diesem Knoten ankommen. Ist das Bit 2 gesetzt, so gibt es Linien, die von diesem Knoten abfahren.



Die Kantendaten bestehen aus einer Variablen Anzahl, die die Gesamtanzahl aller Kanten im Graphen enthält und einem Array mit je einem Zeiger für jeden Knoten im Graphen. Diese Knoten bilden jeweils die Startknoten der Kanten.
Der von diesem Array ausgehende Zeiger führt zu einem EEintrag. Dieser enthält die Anzahl der von dem Knoten ausgehenden Kanten und einen Basiszeiger für eine doppelt verkettete Liste. Die Datenblöcke dieser Adjazenzliste beinhalten jeweils den Endknoten E2Nr der Kante, den LINK dieser Kante, eine Struktur LINKStre, das Gewicht der Kante und die zwei nötigen Zeiger der Liste.
Die LINKStre-Struktur ist ähnlich aufgebaut, wie der LINK, nur das an Stelle der Liniennummern, Zeiger auf den entsprechenden Linienabschnitt der Linienwege enthalten sind. Dadurch kann später bei der Verteilung der Belastung dieser Kante sehr schnell auf die zugehörigen Linien zugegriffen werden.

Die wichtigsten Funktionen von MWGRAPH.PAS:

`procedure MakeGraph;`
Diese Prozedur erzeugt aus dem zuvor geladenen Verkehrsnetz den entsprechenden Graphen.

`procedure AddBEZ(BEZNr: BEZNrTyp);`
Diese Prozedur bindet die Fußwegkanten des übergebenen Bezirks in den Graphen ein.

`prozedur RemoveBEZ(BEZNr: BEZNrTyp);`
Diese Prozedur entfernt die Fußwegkanten des übergebenen Bezirks aus dem Graphen.

`function ExistKante(V1, V2: VNrTyp): boolean;`
Diese Funktion liefert true, wenn eine Kante vom Knoten V1 zum Knoten V2 existiert, sonst false.

`funktion Gamma(V1, V2: VNrTyp): GewichtTyp;`
Diese Funktion liefert das Gewicht der Kante vom Knoten V1 zum Knoten V2. Wenn diese Kante nicht existiert, so wird die Knostante MaxGewicht zurückgeliefert.

`function CheckUMSKante(V1, V2: VNrTyp): boolean;`
Diese Funktion liefert true, wenn die Kante vom Knoten V1 zum Knoten V2 eine Umsteigekante ist, sonst false.

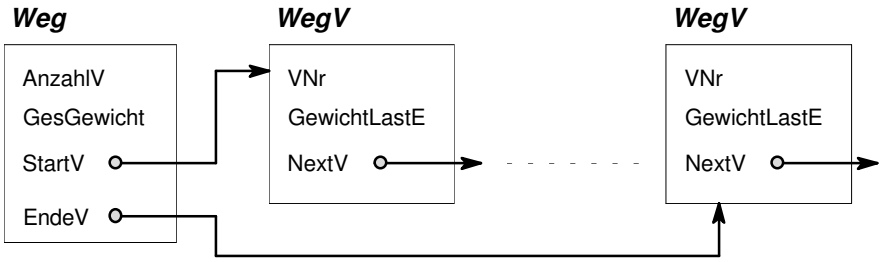
`procedure GraphStatus;`
Diese Prozedur gibt auf dem Bildschirm den aktuellen Status des Graphen aus: Anzahl Knoten, Anzahl Kanten und benötigter Speicherplatz.

`procedure GraphDumpDatei;`
Diese Funktion erstellt sechs Dateien mit den aktuellen Listen der folgenden Elemente:

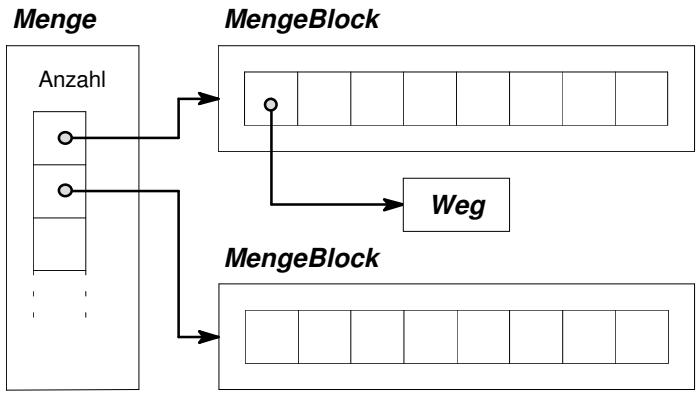
HS	Alle Haltestellen des Netzes
LIN	Alle Linien des Netzes
BEZ	Alle Bezirke des Netzes
V	Alle Knoten des Graphen
E	Alle Kanten des Graphen
LINK	Die globale LINK-Liste

`procedure DisposeGraph;`
Diese Prozedur löscht alle Daten des Graphen im Speicher.

Wege



Jeder Weg hat einen Datenblock Weg. Er beinhaltet die Anzahl der Knoten des Weges (AnzahlV) und das Gesamtgewicht des Weges (GesGewicht). Der eigentliche Wegverlauf wird durch eine einfach verkettete Liste beschrieben, wobei StartV auf den Beginn und EndeV auf das Ende dieser Liste zeigt.
Jeder Datenblock der Liste enthält die Knotennummer Vnr, das Gewicht der Kante vom vorherigen Knoten in der Liste bis zu diesem Knoten und einen Zeiger auf den nächsten Knoten der Liste.
Der Wert GewichtLastE des ersten Listenelements ist Null, da dieses Element den Startknoten der ersten Wegkante darstellt.



Der k-Wege-Algorithmus benötigt zur Berechnung und zur Rückgabe des Ergebnisses eine Menge von Wegen:
Der Menge-Datenblock enthält eine Angabe über die Gesamtanzahl an Wegen in der Menge und ein Zeigerarray. Normalerweise könnten diese Zeiger direkt auf die einzelnen Wege zeigen. Da jedoch auf Grund von Speicherrestriktionen innerhalb eines Array nur 16000 Zeiger gespeichert werden können und der k-Wege-Algorithmus stellenweise bis zu 40000 Wege berechnet, wird ein erneutes Array benötigt: ein sogenannter Block. Damit lassen sich jetzt maximal 16000 x 16000 = 256 Mio. Wege in einer Menge speichern.

Prozeduren und Funktionen für Wege und Mengen:

```
procedure WegNew(var Weg: PWegTyp);
```
Erzeugt einen neuen (leeren) Weg und gibt einen Zeiger darauf zurück.

```
procedure WegAddKnotenVorne(Weg: PWegTyp; KnotenNr: VNrTyp;
                            Gewicht: GewichtTyp);
```
Der übergebene Knoten wird an den übergebenen Weg vorne angehängt (mit Gewicht).

```
function WegGetKnotenNr(Weg: PWegTyp; KnotenNr: longint):VNrTyp;
```
Liefert die KnotenNr der entsprechenden Knotens des Weges.

```
function WegVergleich(Weg1, Weg2: PWegTyp):boolean;
```
Liefert nur dann true, wenn beide Wege exakt die gleiche Knotenfolge haben.

```
function WegVergleichAnfang(Weg1, Weg2: PWegTyp;
                            Anzahl: longint):boolean;
```
Vergleicht die ersten „Anzahl“ Knoten der beiden Wege.

```
function WegGetRoot(Weg: PWegTyp; AnzahlRoot: longint):PWegTyp;
```
Liefert den Anfang eines Weges (AnzahlRoot Knoten) als neuen Weg zurück.

```
function WegConcat(Weg1, Weg2: PWegTyp) : PWegTyp;
```
Hängt Weg2 and Weg1 an,

```
procedure WegPrint(Weg: PWegTyp);
procedure WegPrintkurz(Weg: PWegTyp);
```
Diese beiden Prozeduren geben die Knotenfolge des Weges auf dem Bildschirm aus.

```
procedure WegClear(var Weg: PWegTyp);
```
Löscht die Knotenfolge des Weges. Er ist dann leer.

```
procedure WegDispose(var Weg: PWegTyp);
```
Löscht einen Weg im Speicher.

```
procedure MengeNew(var Menge: PMengeTyp);
```
Erzeugt eine neue, leere Menge.

```
procedure MengeAddWeg(Menge: PMengeTyp; Weg: PWegTyp);
```
Fügt den übergebenen Weg in die Menge ein.

```
procedure MengeDeleteWeg(Menge: PMengeTyp; WegNr: longint);
```
Löscht den Weg mit der Nummer „WegNr“ in der Menge.

```
function MengeGetAnzahlWege(Menge: PMengeTyp):longint;
```
Gibt die Anzahl der Wege in der Menge zurück.

```
function MengeGetWegZeiger(Menge: PMengeTyp;
                           WegNr: longint):PWegTyp;
```
Liefert einen Zeiger auf den Weg mit der Nummer „WegNr“ in der Menge.

```
function MengeGetMinWeg(Menge: PMengeTyp):PWegTyp;
```
Liefert den Weg mit dem kleinsten Gesamtgewicht in der Menge.

```
procedure MengeClear(Menge: PMengeTyp);
```
Alle Zeiger auf die Wege der Menge werden gelöscht. Die Wege selber werden nicht gelöscht. Die Menge ist anschließend leer.

```
procedure MengeDispose(var Menge: PMengeTyp);
```
Die Menge wird aus dem Speicher gelöscht.

6.4 Programmparameter

Im Hauptprogramm können folgene Parameter der Mehrwegumlegung verändert werden:

Parameter	Bedeutung	mögliche Werte
Verkehrszeit	Verkehrstageszeit, in der die Mehrwegumlegung berechnet werden soll	1 = HVZ 2 = NVZ 3 = SVZ
ABSFaktor	Absoluter Grenzfaktor (in Sekunden)	z.B. 900.00
RELFaktor	Relativer Grenzfaktor (in Prozent)	z.B. 0.20
Verteilung	Verteilungsverfahren bei der Umlegung der Belastungen	1 = prozentual 2 = Mandatsverfahren
IOTrace	Unterschiedliche Stufen der Ausgaben auf dem Bildschirm	0 bis 4
DataDump	Ausgabe aller Elemente in Dateien (Haltestellen, Linien, etc.)	0 = nein 1 = ja
PROMode	Art der Protokollierung der Berechnung 1 = Nur die Datenübersicht und die Zeitmessung 2 = zusätzlich die Anzahl von Wegen bei jedem Bezirkspaar 3 = zusätzlich die einzelnen Wege	0 bis 3
ParLIN	Berücksichtigung paralleler Linien	0 = nein 1 = ja
Netz	Auswahl des Verkehrsnetzes 1 = Lummerland, 2 = MesseB, 3 = Ulm, 4 = Ostfildern, 5 = Messe	1 bis 5

Zusätzlich zu diesen Parametern gibt es in einigen Modulen noch einige Compilerschalter:

Schalter	Modul	Bedeutung	Zustände
TIME	MEHRWEG	Zeitmessung eingeschaltet	TIME oder NOTIME
MEHRWEG	MEHRWEG	Es wird eine Mehrwegumlegung durchgeführt	
BESTWEG	MEHRWEG	Es wird eine Bestwegumlegung durchgeführt, das heißt an Stelle des k-Wege-Algorithmus, wird Dijkstras Algorithmus aufgerufen und nur der Bestweg bestimmt	
CHECKFLAG	MWDATA MWWEG MWYEN	Ist der Schalter NOCHECKFLAG aktiviert, so werden in vielen Prozeduren und Funktionen bestimmte Fehler nicht mehr geprüft, z.B. wird ein Zeiger vor seiner Verarbeitung nicht mehr geprüft, ob er nil ist. Dies spart Zeit, erzeugt aber auch keine Fehlermeldungen mehr.	CHECKFLAG oder NOCHECKFLAG
HEAP	MWYEN	Art der Randverwaltung für den Dijkstra-Algorithmus	ISORT, SSORT oder HEAP
TIME	MWYEN	Zeitmessung während des Yen-Algorithmus	TIME oder NOTIME

6.5 Implementierungsgrenzen

Da bei sehr vielen Datenstrukturen aus Effizienzgründen Arrays verwendet wurden, ergeben sich für diese Strukturen fogende Maximalgrenzen

Maximale Anzahl von Haltestellen	5000
Maximale Anzahl von Knoten, die zusammen eine Haltestelle repräsentieren	13000
Maximale Anzahl Linien	5000
Maximale Anzahl von Haltestellen eines Linienverlaufs	750
Maximale Anzahl von parallelen Linien	255
Maximale Anzahl von Bezirken	5000
Maximale Anzahl von Anbindungen eines Bezirks	8000
Maximale Anzahl Knoten	15900

7. Laufzeitergebnisse

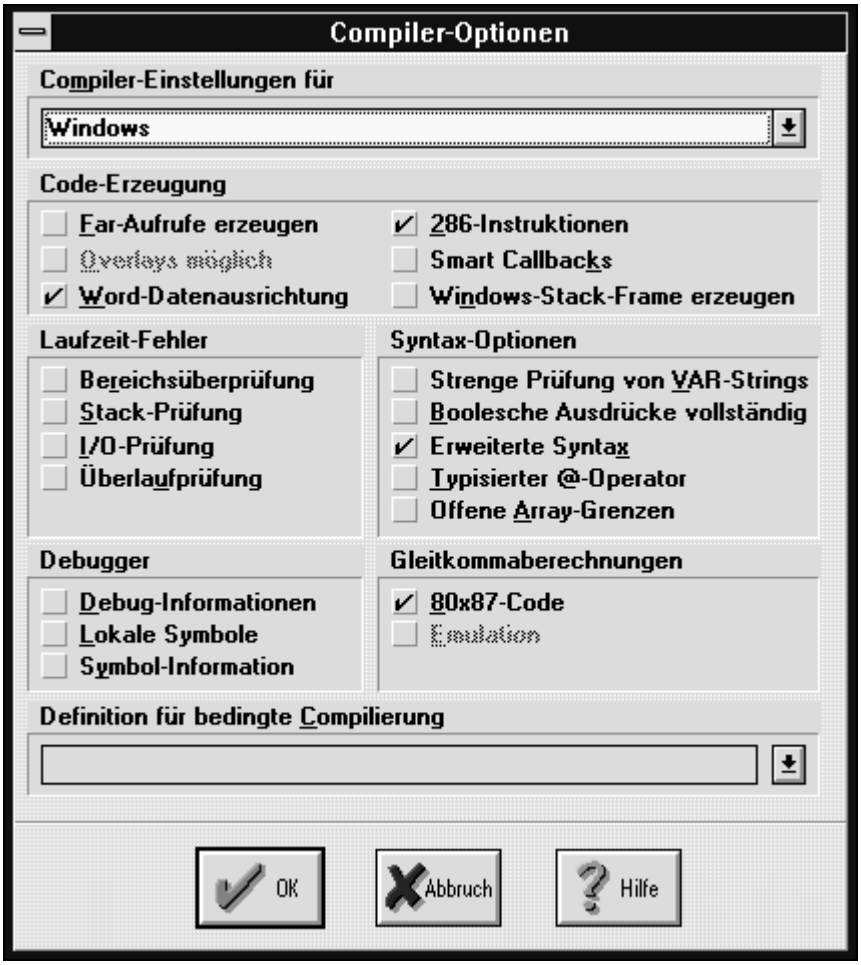
7.1 Laufzeitsystem

Für die Laufzeitmessung der Mehrwegumlegung am Institut für Eisenbahn- und Verkehrswesen der Universität Stuttgart, wurde folgendes System benutzt:

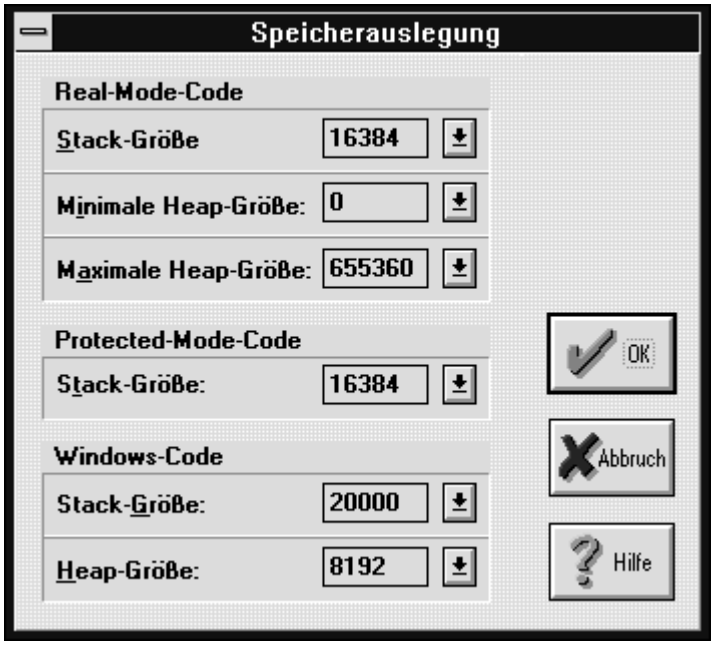
- Intel Pentium PC, 100 MHz
  256 KB Cache
  128 MB Hauptspeicher
- Betriebsystem Windows 3.1

Auf Software/Compiler-Seite galten folgende Bedingungen:

- Programmcode in Borland Pascal 7.0 für Windows
- Compileroptionen:



- Speicheroptionen:



- Programmausführung ohne Fehlerprüfung (Schalter NOCHECKFLAG, siehe Abschnitt 6.4)

Sonstige Testbedingungen:

- keine Ausführung von Hintergrundprogrammen während dem Test (Bildschirmschoner, etc.)
- Abkopplung von vorhandenen Netzwerken

7.2 Testnetze

Für die Testphase der Mehrwegumlegung wurden fünf verschiedene Verkehrsnetze verwendet:

LUMMERLAND Dieses relativ kleine und fiktive Netz wurde von mir entwickelt, um vor allem die implementierten Algorithmen auf ihre Korrektheit zu prüfen. Es enthält alle Spezial- und Problemfälle, die in Verkehrsnetzen auftreten können (Parallelverkehr, hohe Konzentrationen im Kernbereich, etc.). Dennoch ist es sehr klein dimensioniert, damit die Ergebnisse noch „von Hand nachrechenbar" und überprüfbar sind.

Die vier weiteren Netze sind reale Netze öffentlicher Verkehrsmittel aus verschiedenen Städten. Sie alle werden vom Institut für Eisenbahn- und Verkehrswesen der Universität Stuttgart für Untersuchungen und Prognosen eingesetzt:

ULM Dies ist ein Netz der Stadt Ulm von mittlerer Größe.

OSTFILDERN Dieses Netz ist ein Teilbereich des Stuttgarter Verbundnetzes. Es deckt vor allem den Bereich Ostfildern und seine Verbindungen zum übrigen Verbundnetz ab. In der Lehre wurde es für Projektstudien der Studenten angewendet.

MESSE Hierbei handelt es sich um das öffentliche Verkehrsnetz des Großraums Stuttgart. Darin enthalten ist auch schon der Planungsfall eines zukünftigen Messestandorts am Flughafen Stuttgart. Auf Grund seiner Ausdehnung und der feinen Aufteilung der Verkehrszellen, ist es das Umfangreichste der fünf Testnetze.

MESSE-B Als direkte Teilmenge des Netzes MESSE enthält dieses Netz nur die Eisenbahnlinien und die zugehörigen Haltestellen und Bezirke.

Hier die genauen Netzdaten (OVE-Dateien):

NETZDATEN	Lummerland	MesseB	Ulm	Messe
Anzahl Haltestellen	**17**	**53**	**320**	**784**
Anzahl Linien	**9**	**12**	**40**	**404**
Maximale Linienlänge	8	14	26	28
Anzahl Bezirke	**8**	**68**	**96**	**787**
Max. Anzahl Anbindungen	3	2	10	4

Die Belastungszahlen der VD-Dateien:

BELASTUNGSDATEN	Lummerland	MesseB	Ulm	Messe
Bezirkspaare[1]	**56**	**841**	**5885**	**40267**
Vollst. Anzahl Bezirkspaare[2]	**56**	**4556**	**9120**	**618582**
Vollständigkeit in %	100 %[4]	19 %	65 %	7 %
Gesamtanzahl Fahrten[3]	**56000**	**54387**	**118552**	**1045242**

[1] Die VD-Dateien enthalten oft mehr Bezirkspaare, als hier angegeben, da aber in der VD-Datei auch die Fahrten des Individualverkehrs enthalten sind, kann es sein, daß bei einem Bezirkspaar nur Fahrten im IV zu verteilen sind und keine Fahrten im ÖPNV. Daher wurden hier nur die Paare aufgelistet, die für die ÖPNV-Mehrwegumlegung relevant sind.

[2] Diese Zahl ist die maximal mögliche Anzahl von Bezirkspaaren. Das heißt, wenn von jedem Bezirk zu jedem anderen Bezirk gerechnet würde. Sie errechnet sich aus der Bezirksanzahl der Netzdaten:
Vollständige Anzahl Bezirkspaare = (Anzahl Bezirke) · (Anzahl Bezirke - 1)

[3] Die Summe aller Belastungszahlen

[4] Diese Bezirkspaare sind vollständig vorhanden, da Lummerland ein fiktives Netz ist.

7.3 Testläufe

Zunächst wurden einige Testläufe mit dem kleinen Netz „Lummerland“ durchgeführt, wobei alle Ergebniswege und Belastungen in Dateien protokolliert wurden. Diese Wege sind dann mit Ergebniswegen, die von Hand berechnet wurden, verglichen worden. Auf diese Weise konnte die Korrektheit der Implementierung nachgewiesen werden.
Danach wurden auch Testläufe mit den anderen Netzen durchgeführt und in Stichproben Plausibilitätsprüfungen vorgenommen. Hier hat sich gezeigt, daß die Resultate im Rahmen der erwarteten Ergebnisse lagen.
Damit steht fest, daß die Mehrwegumlegung korrekt abläuft - aber leider auch sehr lange, wie die Laufzeiten im nächsten Abschnitt belegen.

7.4 Laufzeiten

Die Datenstruktur für eine Menge von Wegen läßt eine große Anzahl von Wegen zu: ca. 250 Mio. Wege pro Menge. Dieses Ausmaß wird jedoch nie erreicht und verlangsamt die Berechnung doch erheblich. Da aber auf der anderen Seite von vorne herein nicht bekannt ist, wieviele gültige Wege sich für ein Bezirkspaar ergeben werden, waren mehrere Testläufe nötig um eine brauchbare Größe zu ermitteln: Für alle Laufzeitmessungen wurde mit einer Größenordnung von 10000 Wegen pro Block und vier Blöcken pro Menge gerechnet. Dies entspricht 40000 Wegen pro Menge.

Des weiteren wurden folgende Parameter verwendet:

Umsteigefaktor	30	s pro Umsteigen
Absoluter Grenzfaktor	900	s
Relativer Grenzfaktor	20	%

Zu Beginn wurde zunächst eine Bestwegumlegung durchgeführt, das heißt, es wurde nicht der Yen-Algorithmus zur Wegsuche verwendet, sondern der normale Dijkstra-Algorithmus. Dabei ergaben sich folgende Ergebnisse:

Bestweg

	Lummerland	**MesseB**	**Ulm**	**Messe**
Bezirkspaare	56	841	5885	40267
Zeitdauer	**1,5 sek**	**21,3 sek**	**7,3 min**	**1,3 h**
Anzahl berechneter Wege	56	841	5885	40267
Durchschnittl. Weglänge[1]	5,55	7,76	16,34	11,77

[1] Durchschnittliche Anzahl von Graphenknoten pro Weg

Der Nachteil einer Bestwegumlegung mit Dijkstra ist, daß es durchaus mehrere Bestwege mit dem gleichen Gewicht von einen Bezirk zu einem anderen Bezirk geben kann. Der Algorithmus von Dijkstra liefert jedoch nur einen dieser Wege und die ganze Belastung wird auf diesen einen Weg umgelegt.
Eine Verbesserung wird erreicht, wenn hier die Mehrwegumlegung mit dem Yen-Algorithmus und den Grenzfaktoren 0 und 0% verwendet wird. In diesem Fall berechnet Dijkstras Algorithmus einen der Bestwege, der dann dem Yen-Algorithmus als Ausgangsweg dient und dieser berechnet alle weiteren Bestwege.
Eine derartige Mehrwegumlegung mit den Grenzen 0 und 0% benötigt folgende Laufzeiten:

Mehrweg 000 / 0,00 / 30*

	Lummerland	MesseB	Ulm	Messe
Bezirkspaare	56	841	5885	40267
Zeitdauer	**2,8 sek**	**34,8 sek**	**21,2 min**	**6,2 h**
Σ G-Wege[1]	56	849	6395	46416
Σ V-Wege	56	849	6102	44223
Max. Anzahl G-Wege[2]	1	5	7	16
davon V-Wege	1	5	1	14
Durchschnittl. Weglänge	5,55	7,71	16,51	11,84

* Diese Angaben charakterisieren die Laufzeitmessung und sollen einen schnellen Überblick ermöglichen. Die drei Werte entsprechen dabei dem absoluten Grenzfaktor, dem relativen Grenzfaktor und dem Umsteigefaktor, da diese Werte bei den Laufzeitmessungen variiert wurden.

[1] Das sind die Summen der berechneten Wege. Die G-Wege sind die Wege, die vom Yen-Algorithmus zurückgeliefert werden, also die Wege im Graphen. Die V-Wege sind die Wege, die nach dem Wegecheck noch übrig geblieben sind. Also die sinnvollen Verkehrswege.

[2] Dies ist die maximale Anzahl von G-Wegen, die bei einem Bezirkspaar gefunden wurde und die Anzahl an V-Wegen, die davon übrig geblieben sind.

Wenn bei dieser Mehrwegumlegung die Anzahl der V-Wege mit der Anzahl der Bezirkswege verglichen wird, zeigt sich deutlich, daß doch bei einigen Bezirkspaaren mehrere Bestwege vorhanden sind.

Als nächstes wurde eine echte Mehrwegumlegung mit den Grenzfaktoren 900 und 20% durchgeführt:

Mehrweg 900 / 0,20 / 30

	Lummerland	MesseB	Ulm	Messe
Bezirkspaare	56	841	5885	40267
Zeitdauer	**4,5 sek**	**35,3 sek**	**142,4 h**	**181 Tage (!)**
Σ G-Wege[1]	56	849	-	-
Σ V-Wege	56	849	-	-
Max. Anzahl G-Wege[2]	1	5	-	-
davon V-Wege	1	5	-	-
Durchschnittl. Weglänge	5,55	7,71	-	-
			696	153
			16,8 h	**16,7 h**
			163210	64602
			18737	4318
			10635	8221
			209	276
			24,91	18,12

Auf Grund der enormen Laufzeiten der Netze Ulm und Messe konnte die Mehrwegumlegung nicht bis zum Ende berechnet werden. Daher wurde nach einiger Zeit abgebrochen und die Werte der bisherigen Berechnung sind in den Zeilen unter den grauen Bereichen dargestellt.
Aus der Anzahl der bisher berechneten Bezirkspaare und der Gesamtanzahl von Paaren, läßt sich die wahrscheinliche Gesamtlaufzeit prozentual hochrechnen. Diese Hochrechnungswerte sind in den grauen Bereichen abgebildet, wobei diese Hochrechnungen nur sehr vage Annäherungen an die echten Werte sind. Dies hängt von der Verteilung der aufwendigen und einfachen Bezirkspaare in der VD-Datei ab.

Vermutlich wird die hohe Laufzeit durch die Berechnung der vielen Wege verursacht. Um diese Behauptung zu prüfen, wurde die prozentuale Verteilung der Laufzeit innerhalb des Programms untersucht:

Die prozentuale Verteilung der Laufzeit bei der Mehrwegumlegung von Ulm:

	Programmabschnitt	Laufzeitanteil
	• Daten einlesen	< 0,01 %
	• Graph erstellen	< 0,01 %
Hauptschleife	• Bezirkspaar einlesen und Fußwegkanten einbinden	< 0,01 %
	• k-Wege-Algorithmus	98,9 %
	• Wegecheck	0,7 %
	• Umlegung	0,2 %
	• Fußwegkanten und Wege löschen	0,1 %
	• Umlegung speichern und Daten und Graph löschen	< 0,01 %

Es ist deutlich sichtbar, daß die hohe Laufzeit der Berechnung allein durch den k-Wege-Algorithmus verursacht wird. Eine Untersuchung der Laufzeitverteilung innerhalb des Yen-Algorithmus ergibt:

	Programmabschnitt innerhalb von Yen	Laufzeitanteil
	Dijkstra (Bestweg)	2,0 %
	(Ia) Weganfänge vergleichen und Kanten löschen	0,5 %
k-Wege-Schleife	(Ib) spezieller Dijkstra (Initialisierung)	0,5 %
	(Ib) spezieller Dijkstra (Hauptschleife)	96,5 %
	(Ib) spezieller Dijkstra (Spur ermitteln)	0,2 %
	(Ic) Neuen Weg zusammensetzen	0,1 %
	(II) Minimumweg in B finden	0,2 %

Auch hier wird deutlich, daß der Berechnungsaufwand in der Wegsuche selber, der Hauptschleife des speziellen Dijkstras, liegt. Es liegt also nicht an irgendwelchen zusätzlichen Prüfroutinen oder Initialisierungssequenzen, sondern der Aufwand wird allein durch die große Anzahl an Wegen bedingt.
Sehr auffällig ist auch, daß der k-Wege-Algorithmus sehr viele Wege berechnet und daß anschließend im Wegecheck der größte Teil der Wege wieder gelöscht wird, da sie nicht sinnvoll sind. Das hängt vor allem mit dem Parallelverkehr zusammen. Wie in Abschnitt 4.12 beim Wegecheck beschrieben, werden sehr viele Wege gefunden, wenn eine größere Anzahl von Linien parallel verläuft. Wird in einem Weg die Hauptlinienkombination dieser Linien verwendet, dann haben alle weiteren Kombinationsmöglichkeiten dieser Linien auf jeden Fall ein höheres Gewicht und sind damit nicht mehr sinnvoll. Berechnet werden sie aber gleichwohl, da es bis jetzt keine Möglichkeit gibt, diese Wege direkt im Algorithmus abzufangen.
Weitere Untersuchungen haben auch ergeben, daß der geographische Abstand der Bezirke eine Rolle spielt: Je größer der Abstand zwischen den beiden Bezirkspunkten ist, desto längere Weg ergeben sich und um so höher ist das Gewicht des Bestweges. Liegen die beiden Bezirke an gegensätzlichen Rändern des Untersuchungsraumes und laufen die meisten Wege durch die Kernzone des Raumes, der üblicherweise viele parallele Linien enthält, so ergibt sich eine sehr große Anzahl an Wegen. In einem solchen Fall gibt es in der Kernzone sehr viele alternative Möglichkeiten und diese werden auch zahlreich vom Wegsuchealgorithmus verwendet, weil hier der Umsteigefaktor im Gegensatz zum großen Gesamtgewicht des Weges relativ gering ist und deshalb nicht schon nach ein- oder zweimaligem Umsteigen zur Eliminierung des Weges führt (Außerhalb der Grenze). In solchen Fällen werden oftmals über 20000 Wege gefunden und vom Wegecheck aber nur knapp 100 für sinnvoll befunden.

Nach Absprache mit dem Institut für Eisenbahn- und Verkehrswesen, wurde beschlossen den Umsteigefaktor von 30 auf 150 Sekunden heraufzusetzen, da 30 Sekunden ein relativ kleiner Wert sei und 150 Sekunden realistischer wäre.
Durch den höheren Umsteigefaktor müßten Wege, die ein häufiges Umsteigen beinhalten, früher die erlaubte Grenze verlassen und damit ausscheiden. Dies hat sich auch in der erneuten Mehrwegumlegung mit dem höheren Umsteigefaktor gezeigt:

Mehrweg 900 / 0,20 / 150

	Lummerland	MesseB	Ulm	Messe
Bezirkspaare	56	841	5885	40267
Zeitdauer	**4,2 sek**	**27,7 sek**	**195,4 h**	**97 Tage**
Σ G-Wege[1]	56	849	967945	-
Σ V-Wege	56	849	113023	-
Max. Anzahl G-Wege[2]	1	5	9399	-
davon V-Wege	1	5	106	-
Durchschnittl. Weglänge	5,55	7,71	25,53	-
				295
				17,0 h
				75079
				6336
				8197
				313
				18,16

Durch die Erhöhung des Umsteigefaktors fallen jetzt Wege, die zum Beispiel ein fünffaches Umsteigen enthalten und damit nicht mehr sehr sinnvoll sind, aus der Menge der gültigen Wege heraus. Damit verkürzt sich die Laufzeit um einiges.
Daß die Laufzeit von Ulm mit 195 Stunden, im Gegensatz zu 142 Stunden der vorherigen Laufzeitmessung, höher ist, täuscht. Denn die 142 Stunden waren nur eine vage Hochrechnung, während die 195 Stunden tatsächlich gerechnet wurden.

Diese Laufzeiten sind immer noch nicht praktikabel und deshalb wurden nach erneuter Beratung mit dem Institut für Eisenbahn- und Verkehrswesen, die Grenzfaktoren erniedrigt. Mit den Faktoren 900 und 0,20 ist bei den Verkehrswissenschaftlern noch nie gerechnet worden. Sie haben bisher immer nur 300 und 0,10 verwendet. Die 900 und 0,20 sind sozusagen Wunschfaktoren.
Aus diesem Grund wurde eine erneute Meßreihe mit den Faktoren

absoluter Grenzfaktor = 300
relativer Grenzfaktor = 0,10

durchgeführt:

Mehrweg 300 / 0,10 / 150

	Lummerland	MesseB	Ulm	Messe
Bezirkspaare	56	841	5885	40267
Zeitdauer	**3,63 sek**	**22,9 sek**	**3,0 h**	**57,1 h**
Σ G-Wege[1]	112	934	61924	-
Σ V-Wege	72	916	17317	-
Max. Anzahl G-Wege[2]	6	5	228	-
davon V-Wege	3	5	11	-
Durchschnittl. Weglänge	6,15	7,89	20,92	-
				16320
				23,1 h
				127488
				43916
				367
				54
				14,58

Natürlich können noch bessere Laufzeiten gewonnen werden, wenn die Verkehrsparameter noch weiter gesenkt werden, aber dies würde dem Aufgabenzweck widersprechen. Es müssen daher andere Lösungen zur Verbesserung der Laufzeit gefunden werden.

Ausblick

Ergebnis dieser Diplomarbeit ist zunächst einmal eine **korrekte** Ausarbeitung und Implementierung einer Mehrwegumlegung. Dies war das primäre Ziel welches auch erreicht worden ist. Die Untersuchung der mir zugänglichen Algorithmen hat Yens Algorithmus als klaren Faforiten für die Mehrwegsuche geliefert. Um ein endgültiges Urteil zu fällen, sollten aber die in der Literatur erwähnten Artikel von Shier und Dreyfus noch beachtet werden. Mir war es leider nicht möglich diese Artikel einzusehen.

Es wäre wünschenswert wenn die Algorithmen von Hoffman & Pavley oder Azevedo implementiert und gemessen würden, damit eine Abschätzung möglich ist, wieviele Wege berechnet werden müssen, damit k schleifenfreie darunter sind. Aus Zeitgründen habe ich diese Untersuchungen nicht selbst durchführen können.

Das Laufzeitverhalten der jetzigen Implementierung ist nicht zufriedenstellend und für das Verkehrswesen kaum praktikabel. Um dies zu ändern gibt es verschiedene Ansätze die einer weiteren Untersuchung bedürfen:

Den größten Einfluß auf die Laufzeit dieser Implementierung hat neben dem Algorithmus zur Wegsuche auch der Aufbau des Graphen. In diesem Fall hat sich gezeigt, daß der verwendete Distanzgraph mit der Knotensplittung vielleicht nicht das Optimum ist. Er wurde vor allem deshalb ausgewählt, um eine Untersuchung von möglichst vielen Algorithmen zu ermöglichen. Nachdem jetzt der Yen-Algorithmus für geeignet befunden wurde, sollte darüber nachgedacht werden, ob es nicht eine bessere Repräsentation des Verkehrsnetzes gibt, für die der Yen-Algorithmus dann angepaßt werden könnte.
Ebenso ist es möglich, in der Literatur doch noch einen besseren Algorithmus zu finden.

Eine weitere Möglichkeit wäre die Verwendung von Heuritiken, was allerding zu Lasten eines korrekten Ergebnisses gehen würden. Daher muß eine vorgesehene Heuristik sorgfältig geprüft werden, ob ihr Nutzen, sprich die Laufzeitverbesserung, den Nachteil einer Abweichung vom richtigen Ergebnis überwiegt.

In diesem Zusammenhang wäre auch eine psychologische Untersuchung des menschlichen Fahrverhaltens von Nutzen. Insbesondere könnte die Analyse der menschlichen Entscheidungsindung - welche Wege in Betracht gezogen werden und welche abgelehnt werden - heuritische Regeln aufzeigen, die in der Implementierung verwendet werden könnten.

Eine Erweiterung oder Veränderung des Programms müßte auf Grund des modularen Aufbaus leicht möglich sein.

Das Thema Wegsuche wird in der Informatik wohl immer ein Thema bleiben. Ich hoffe, ich habe mit dieser Arbeit auch ein kleines Stück des Weges dazu beigetragen.

Ein herzlicher Dank gebührt meinen Betreuern, die mich auf beiden Seiten (Theorie und Verkehrswesen) ausgiebig unterstützt haben.

Anhang A: Beispielgraph für Wegsuche

Dieser Anhang zeigt einen Beispielgraphen, der alle Problemfälle enthält, die mir im Zusammenhang mit der Suche nach „k-kürzesten, schleifenfreien Wegen“ während meiner Diplomarbeit begegnet sind. Es geht hierbei rein um die theoretische Wegsuche in einem gerichteten Graphen, ohne den verkehrlichen Aspekt und seine Einschränkungen.

An diesem Graphen (Abb. A.1) können alle weitergehenden Untersuchungen und Algorithmen zu diesem Thema getestet und geprüft werden. Dazu sind im Anschluß (nächste Seite) alle schleifenfreien Wege vom Startknoten A zum Zielknoten R in der Reihe ihrer Gewichte aufgelistet. Zukünftige Algorithmen sollten sicherstellen, daß sie auch tatsächlich alle diese, und nur diese Wege von A nach R finden.

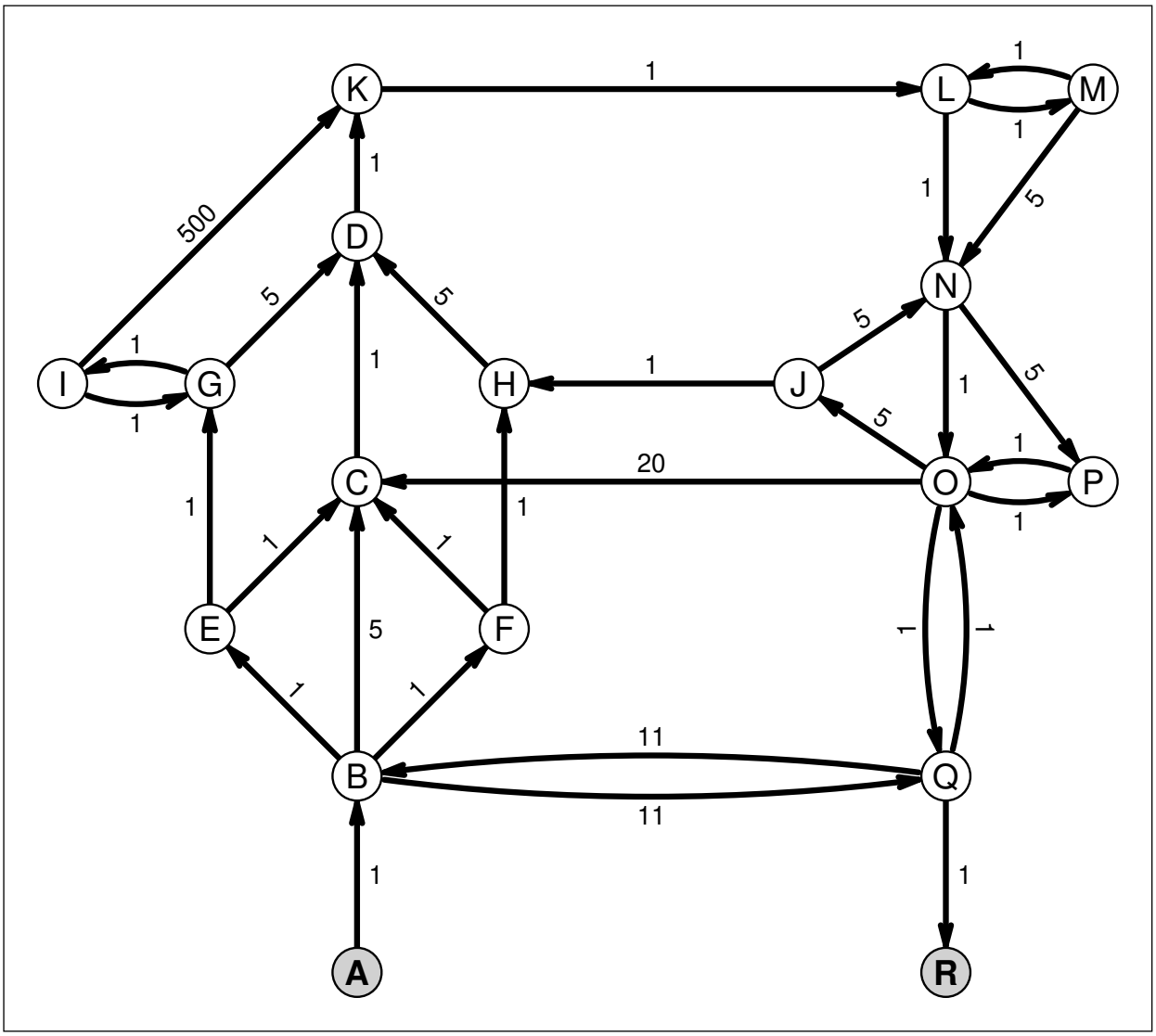


Abb. A.1: Beispielgraph für Wegsuche

Dieser Graph (Abb. A.1) umfaßt folgende Werte:

- gerichteter Digraph
- 18 Knoten (von A bis R)
- 33 Kanten mit positiven Kantengewichten im Bereich 1 bis 500
- enthält nur positive Schleifen
- nicht planar
- speziell ausgerichtet auf die Wegsuche von A nach R

Er enthält unter anderem folgende Problemfälle:

- Schleifen
- ungünstige Weggewichte
- mehrere Wege mit gleichem Gewicht
- Konstrukte, die es nur dann erlauben, schleifenfreie Wege nach der Methode der „Wegerweiterung durch Umwegeinbau“ zu finden, wenn auch Wege, die Schleifen enthalten, berücksichtigt und berechnet werden

Hier die komplete Liste aller schleifenfreien Wege vom Knoten A zum Knoten R:

Weg Nr.	Rang*	Gewicht	Wegverlauf (Knotenfolge)
1	1	10	A - B - E - C - D - K - L - N - O - Q - R
2			A - B - F - C - D - K - L - N - O - Q - R
3	2	13	A - B - Q - R
4			A - B - C - D - K - L - N - O - Q - R
5	3	14	A - B - E - G - D - K - L - N - O - Q - R
6			A - B - F - H - D - K - L - N - O - Q - R
7	4	15	A - B - E - C - D - K - L - N - P - O - Q - R
8			A - B - E - C - D - K - L - M - N - O - Q - R
9			A - B - F - C - D - K - L - N - P - O - Q - R
10			A - B - F - C - D - K - L - M - N - O - Q - R
11	5	18	A - B - C - D - K - L - N - P - O - Q - R
12			A - B - C - D - K - L - M - N - O - Q - R
13	6	19	A - B - F - H - D - K - L - N - P - O - Q - R
14			A - B - F - H - D - K - L - M - N - O - Q - R
15			A - B - E - G - D - K - L - N - P - O - Q - R
16			A - B - E - G - D - K - L - M - N - O - Q - R
17	7	20	A - B - F - C - D - K - L - M - N - P - O - Q - R
18			A - B - E - C - D - K - L - M - N - P - O - Q - R
19	8	23	A - B - C - D - K - L - M - N - P - O - Q - R
20	9	24	A - B - F - H - D - K - L - M - N - P - O - Q - R
21			A - B - E - G - D - K - L - M - N - P - O - Q - R
22	10	509	A - B - E - G - I - K - L - N - O - Q - R
23	11	514	A - B - E - G - I - K - L - M - N - O - Q - R
24			A - B - E - G - I - K - L - N - P - O - Q - R
25	12	519	A - B - E - G - I - K - L - M - N - P - O - Q - R

* bezogen auf das Gewicht

Anhang B: Aufbau der Dateiformate

Für eine Umlegungsrechnung werden zwei Dateien benötigt:

XXX.OVE	(enthält Netzdaten)
XXX.VD	(enthält Belastungszahlen)

und eine Datei erzeugt:

XXX.UML	(enthält die Umlegungsergebnisse)

Da diese Dateien wiederum als Ausgangs- bzw. Eingangsdaten für andere Programme der Verkehrsprognose dienen, ist ihr Dateiformat durch diese Programme festgelegt.
Im Folgenden werden die Strukturen dieser drei Dateiarten beschrieben, damit spätere Änderungen oder Erweiterungen der Umlegungsrechnung sich ebenfalls an diesem Aufbau orientieren können. Die Zeilen in

```
nicht-proportionaler Schrift
```

stellen Auszüge aus den entsprechenden Dateien der Beispielumlegung für Lummerland dar.

Alle drei Dateiarten sind reine ASCII-Files und können daher auch mit einfachen Editoren erstellt oder bearbeitet werden.

Aufbau von XXX.OVE-Dateien (ASCII)

OVE-Dateien enthalten alle Angaben über ein Verkehrsnetz (Haltestellen, Linien, Bezirke, etc).

Die ersten drei Zeilen sind Kommentarzeilen und enthalten meist Kommentare über Art, Ausstattung und Netzvarianten. Die dritte Zeile enthält meist einen Kommentar über die Bedeutung der Zahlen in der vierten Zeile:

```
Beispielnetz LUMMERLAND für die Mehrweg-Umlegung (Diplomarbeit M. Mack 1996)

Anzahl der Haltestellen, der Linien, der Bezirke und der Tarifzonen:
```

In Zeile vier stehen vier Zahlenwerte. Die ersten drei sind die Größenordnungen (Anzahl) für die Haltestellen, Linien und Bezirke des Netzes. Die vierte Zahl gibt die Anzahl der Zonen an, in die das Netz aufgeteilt ist (falls eine Zoneneinteilung existiert):

```
17      9      8      1
```

Nun folgen fünf Zeilen, die nicht alle Kommentarzeilen sind, aber von meinem Programm so behandelt werden, da sie außer echten Kommentaren nur Daten enthalten, die die Umlegungsberechnung nicht benötigt oder nicht verwenden kann:

```
Tarifsystem: Kilometerpreis und Preis für einfache Fahrt durch n Zonen in Pf.
   110

Dauer der Haupt-, Neben- und Spätverkehrszeit in Minuten:
```

Die nächste Zeile enthält die Dauer der Haupt- , Neben- und Spätverkehrszeit in Minuten:

```
240    540    360
```

Es folgen wieder zwei Kommentarzeilen:

```
Haltestellen: Nummer / x-Koordinate / y-Koordinate / Tarifzone(n) / Name
```

Jetzt folgen die Haltestellendaten.
Für jede Haltestelle gibt es eine Zeile. Sie enthält die Nummer, die x- und y-Koordinate, vier Nummern der Tarifzonen, in der die Haltestelle liegen kann (bei weniger als vier Zonen, werden die restlichen Positionen mit -1 besetzt), und den Namen der Haltestelle:

```
1001       135        50  -1  -1  -1  -1  HBF
1002       120        85  -1  -1  -1  -1  Post
1003       140       105  -1  -1  -1  -1  Bank
1004       165        90  -1  -1  -1  -1  Kirche
...
1015       255       130  -1  -1  -1  -1  Grotte
1016       260       165  -1  -1  -1  -1  Vulkan
1017       210       165  -1  -1  -1  -1  Drachenhöhle
```

Es folgen vier Kommentarzeilen:

```
Linien: Nummer / Art / 3 x Bedienungshäufigkeit bzw. Takt in Minuten
        Haltestellenfolge für Hin- und Gegenrichtung:
        jeweils Haltestellennummer / Fahrweg in Metern / Fahrzeit in Sekunden
```

Dann die Liniendaten:
Jede Linie ist durch drei Zeilen beschrieben
Die erste Zeile enthält Nummer, Art (Verkehrsmittel) und drei mal Bedienungshäufigkeit bzw. Takt in Minuten für die Haupt-, Neben- und Spätverkehrszeit. Ist eine dieser letzen drei Zahlen negativ, so handelt es sich hier um eine Bedienungshäufigkeit (mit -1 multiplizieren): -5 bedeutet, daß diese Linie fünf mal in der entsprechenden Verkehrszeit verkehrt. Ist die Ziffer positiv, so entspricht sie dem Takt in Minuten: 12 bedeutet, daß die Linie alle 12 Minuten verkehrt.
Die zweite und dritte Zeile jeder Linie enthält den Linienverlauf für Richtung und Gegenrichtung der Linie: Starthaltestelle, Fahrweg in m, Fahrzeit in sek., nächste Haltestelle, Fahrweg, ..., Endhaltestelle.
Die Haltestellen werden durch ihre Nummern bezeichnet.:

```
2001  6   35.00   50.00      -3
1001  925  333 1007  350  126 1008 1000  360 1009
1009 1000  360 1008  350  126 1007  925  333 1001
2002  5   20.00   45.00      -2
1001 1450  348 1015  350   84 1016  500  120 1017  775  186 1010
1010  775  186 1017  575  138 1015 1450  348 1001
2003  5   15.00   30.00   60.00
1009 1200  288 1010  650  156 1011  775  186 1001  650  156 1012  350   ...
1016  350   84 1015  800  192 1014  450  108 1013  350   84 1012  650   ...
...
2008  3   12.00   20.00   60.00
1002  250   45 1005  325   59 1004  300   54 1003  450   81 1010  650   ...
1008  750  135 1011  650  117 1010  450   81 1003  300   54 1004  325   ...
2009  0    0.50    0.50    0.50
1001    200   167   1005
1005    200   167   1001
```

Es folgen wieder drei Kommentarzeilen:

```
Bezirke: Nummer / Name / Einbindungen:
         jeweils Haltestellennummer / mittlere Fußweglänge in Metern
```

Zum Schluß noch die Bezirksdaten:
Jeder Bezirk hat zwei Zeilen: in der ersten Zeile stehen Nummer und Name des Bezirks. Die zweite Zeile entält die Anbindungen des Bezirks an Haltestellen: für jede Anbindung ein Zahlenpaar bestehend aus: Nummer der Haltestelle und den mittleren Fußweg in Metern.

```
3001  Südstadt
       1001 200
3002  Zentrum
       1003 175  1004 150  1005 225
...
3007  Warmhausen
       1013 275  1014 225
3008  Moor
       1015 200  1016 175  1017 450
```

Dateiende.

Aufbau von XXX.VD-Dateien (ASCII)

VD-Dateien enthalten Angaben über die täglichen Belastungen (Fahrten) zwischen den Bezirken untereinander.

Die ersten zwei Zeilen sind Kommentarzeilen und enthalten meist Kommentare über Art, Ausstattung und das dazugehörige Netz.

```
Belastungsdatei für Beispielnetz LUMMERLAND
Kommentarzeile
```

Dann folgen die Belastungsdaten. In jeder Zeile stehen vier Werte: der Startbezirk, der Zielbezirk, die Belastung des Individualverkehrs (IV) und die Belastung des Öffentlichen Personennahverkehrs (ÖV).
Die IV-Werte werden überlesen, da sie für die Mehrwegumlegung im ÖPNV nicht relevant sind.

```
3001   3002   1000   1000
3001   3003   1000   1000
3001   3004   1000   1000
3001   3005   1000   1000
3001   3006   1000   1000
3001   3007   1000   1000
3001   3008   1000   1000
3002   3001   1000   1000
3002   3003   1000   1000
...
3007   3008   1000   1000
3008   3001   1000   1000
3008   3002   1000   1000
3008   3003   1000   1000
3008   3004   1000   1000
3008   3005   1000   1000
3008   3006   1000   1000
3008   3007   1000   1000
```

Dateiende.

Die oben abgebildeten IV- und ÖV-Werte sind nur Testwerte und haben keinen realen Bezug. Normalerweise variieren die Werte von Zeile zu Zeile.
Es ist ebenfalls möglich, daß nicht alle möglichen Bezirkspaare überhaupt erscheinen, da nicht von jedem Bezirk in jeden anderen Verkehrsbelastungen auftreten. Die Zeilenanzahl der VD-Datei ist daher üblicherweise kleiner als das Quadrat der Bezirkszahl: (Anzahl VD-Zeilen) ≤ (Anzahl Bezirke)^2.

Aufbau von XXX.UML-Dateien (ASCII)

UML-Dateien enthalten neben den Netzdaten (Haltestellen und Linien) auch die Ergebnisse der Umlegungsrechnung: die täglichen Belastungen der Haltestellen und der Linienabschnitte. Aufgrund dieser Angaben können zum Beispiel Auslastungsgrade oder der Einsatzumfang eines Verkehrsmittels bestimmt werden.

Die ersten drei Zeilen sind wie üblich Kommentarzeilen (Netzart, Version und Ausstattung):

```
Ergebnisse der Mehrweg-Umlegung
Berechnung erfolgte am 19.6.1996
Maximalgrößen für BEZ HS LIN; Dauer der Verkehrstageszeit
```

Darunter stehen vier Zahlenwerte. Die ersten drei sind die Speichergrenzen für Bezirke, Haltestellen und Linien. Die letzte Zahl dieser Zeile entspricht der Dauer der Verkehrstageszeit, in der die Umlegungsrechnung durchgeführt wurde (in Minuten):

```
800   800   450   240
```

Dann folgt eine Zeile mit zehn Zahlenwerten: den aktuellen Größenordnungen (Anzahl) für Bezirke, Haltestellen und Linien. Es folgen die Parameter, die während der Wegsuche oder Umlegung verwendet wurden: Kennziffer der Verkehrstageszeit, Gewichtungsfaktoren für Ein- und Umsteigewartezeiten, Gewichtungsfaktor für Fahrplanabstimmung, die Fußgängergeschwindigkeit und die relative und absolute Grenze für zulässige Alternativrouten (relative Grenze in Prozent 0,xx / absolute Grenze in Sekunden):

```
8   17   9   1   1.0   1.0   0.5   1.2   0.20   900
```

Es folgt wieder eine Kommentarzeile

```
Haltestellenverzeichnis: ext.Nr  xKoor yKoor  Ein Um Aus  Name
```

Nun folgen die Haltestellendaten:

Für jede Haltestelle gibt es zwei Zeilen: Die erste Zeile enthält die Nummer, die x- und y-Koordinate, und die Belastungszahlen für Ein-, Um- und Aussteiger. In der zweiten Zeile steht der Name der Haltestelle:

```
1001 135 50 7000 7250 7000
HBF
1002 120 85 0 0 0
Post
1003 140 105 2059 155 2154
Bank
...
Oase
1015 255 130 3816 0 6154
Grotte
1016 260 165 2855 0 713
Vulkan
1017 210 165 329 0 133
Drachenhöhle
```

Zwei weitere Kommentarzeilen:

```
Es folgt das
Linienverzeichnis
```

Nun die Liniendaten:

Für jede Linie drei Zeilen: Die erste Zeile enthält die Nummer, die Fahrtenfolgezeit, die Anzahl der Fahrten in der ausgewählten Verkehrstageszeit und die Kennziffer der Verkehrsmittelart (siehe Kapitel 1). Die zweite Zeile enthält den Linienverlauf der Richtung und die dritte Zeilen den der Gegenrichtung.

Der Linienverlauf beginnt mit der Nummer der Starthaltestelle. Dann folgen Fahrweg, Fahrzeit, Anzahl Fahrten, Anzahl „Aussteiger“ und die Nummer der nächsten Haltestelle, bis die Endhaltestelle einer Richtung erreicht ist. Jeder Linienverlauf endet mit fünf Nullen.

Verkehrt eine Linie nicht in der Verkehrstageszeit, in der gerechnet wurde, so bestehen die beiden Zeilen der Linienverläufe für Richtung und Gegenrichtung aus jeweils sieben Nullen:

```
2001  35.0  7  6
1 925 333 200 0 7 350 126 1453 253 8 1000 360 1200 1200 9 0 0 0 0 0
9 1000 360 1200 0 8 350 126 1200 1000 7 925 333 292 292 1 0 0 0 0 0
2002  20.0  12  5
1 1450 348 1725 1340 15 350 84 714 457 16 500 120 3011 0 17 775 186 3164 ...
10 775 186 1583 133 17 575 138 1626 1378 15 1450 348 1956 1956 1 0 0 0 0 ...
2003  15.0  16  5
9 1200 288 5800 2025 10 650 156 5026 1303 11 775 186 7319 2719 1 650 156 ...
16 350 84 101 0 15 800 192 1880 1000 14 450 108 880 0 13 350 84 4752 0   ...
...
2008  12.0  20  3
2 250 45 0 0 5 325 59 118 0 4 300 54 691 0 3 450 81 2073 1464 10 650 117 ...
8 750 135 253 253 11 650 117 1248 610 10 450 81 2194 1536 3 300 54 658   ...
2009  50.0  480  0
1 200 167 2611 2611 5 0 0 0 0 0
5 200 167 2719 2719 1 0 0 0 0 0
```

Dateiende.

Anhang C: Beispielumlegung

Damit sowohl der Algorithmus als auch die Implementation auf Korrektheit überprüft werden konnten, wurde ein Beispielnetz benötigt, das auf der einen Seite klein genung ist, damit es noch überschaubar ist und die Ergebniswege von Hand überprüft werden können, das aber auf der anderen Seite auch alle schwierigen Problemfälle eines echten, großen Verkehrsnetzes enthält. Deshalb wurde das fiktive Verkehrsnetz der Insel Lummerland kreiert.
In diesem Anhang werden die vollständigen Daten von Lummerland, sowie einer darauf durchgeführten Mehrwegumlegung dargestellt, damit die gesamte Mehrwegumlegung an einem Beispiel nachvollzogen werden kann.
Dieses Netz kann auch für zukünftige Tests von Umlegungsrechnungen herangezogen werden. Alle Daten dieses Netzes sind beim Institut für Eisenbahn- und Verkehrswesen der Universität Stuttgart erhältlich.

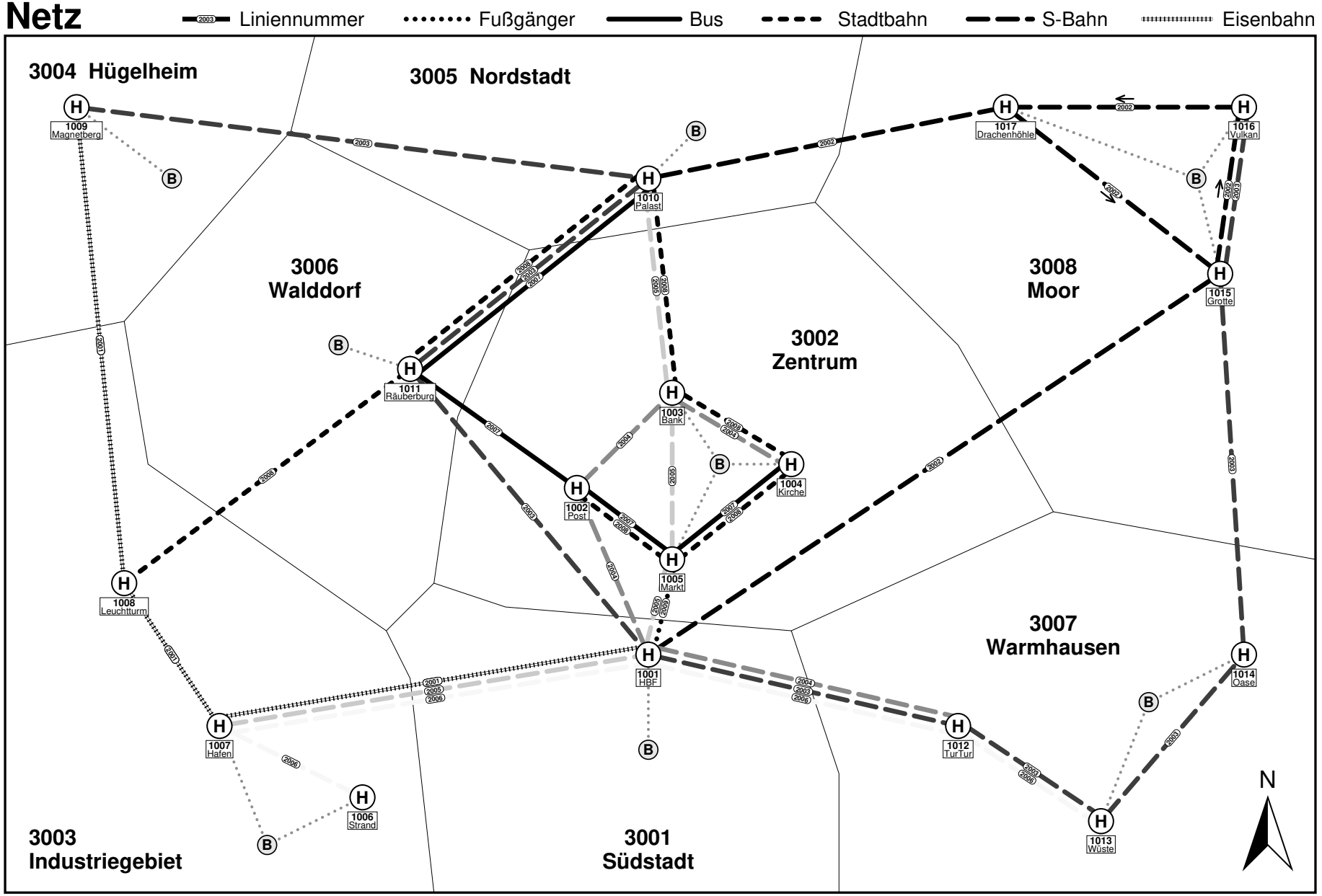
Auf Lummerland gibt es:	8	Bezirke	Nummern 3001 - 3008
	17	Haltestellen	Nummern 1001 - 1017
und	9	Linien	Nummern 2001 - 2009

Diese Geschwindigkeiten wurden zugrunde gelegt:	Fußgänger	1,2 m/s
	Bus	3,85 m/s
	Straßenbahn	5,56 m/s
	S-Bahn	4,17 m/s
	Eisenbahn	2,78 m/s

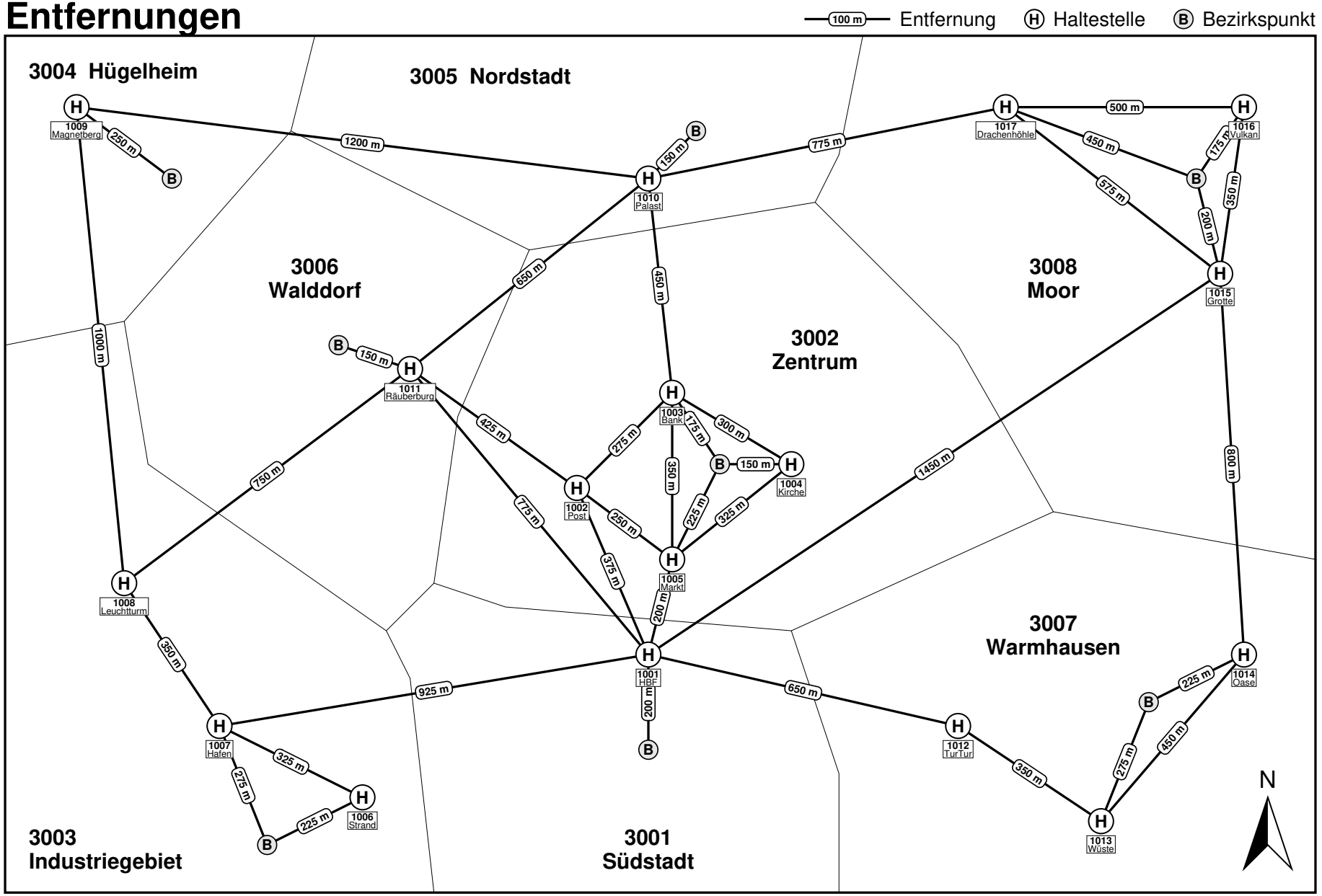
Auf den folgenden Seiten wird dargestellt:	Seite 92	Das Verkehrsnetz von Lummerland
	Seite 93	Die Entfernungen auf der Insel
	Seite 94-95	Auflistung aller Bezirke, Haltestellen und Linien des Verkehrsnetzes
	Seite 96	Der daraus resultierende Distanzgraph
	Seite 97-99	Auflistung aller Knoten und Kanten

Im Anschluß daran werden einige Ergebniswege von ausgewählten Bezirkspaaren dargestellt.

Netz


2003 Liniennummer
Fußgänger
Bus
Stadtbahn
S-Bahn
Eisenbahn
3004 Hügelheim
3005 Nordstadt
3006 Walddorf
3008 Moor
3002 Zentrum
3007 Warmhausen
3003 Industriegebiet
3001 Südstadt
1009 Magnetberg
1017 Drachenhöhle
1016 Vulkan
1010 Palast
1015 Grotte
1011 Räuberburg
1003 Bank
1004 Kirche
1002 Post
1005 Markt
1008 Leuchtturm
1001 HBF
1014 Oase
1007 Hafen
1012 TurTur
1006 Strand
1013 Wüste
N

Entfernungen



Bezirksliste von Lummerland

Bezirk			Anbindungen an Haltestellen		
int. Nr.	Bez.-Nr.	Name	int. Nr.	HS.-Nr.	Weglänge
1	3001	Südstadt	1	1001	200 Meter = 166,7 s
2	3002	Zentrum	3	1003	175 Meter = 145,8s
			4	1004	150 Meter = 125,0 s
			5	1005	225 Meter = 187,5 s
3	3003	Industriegebiet	6	1006	225 Meter = 187,5 s
			7	1007	275 Meter = 229,2 s
4	3004	Hügelheim	9	1009	250 Meter = 208,3 s
5	3005	Nordstadt	10	1010	150 Meter = 125,0 s
6	3006	Walddorf	11	1011	150 Meter = 125,0 s
7	3007	Warmhausen	13	1013	275 Meter = 229,2 s
			14	1014	200 Meter = 166,7 s
8	3008	Moor	16	1016	175 Meter = 145,8 s
			17	1017	450 Meter = 375,0 s

Haltestellenliste von Lummerland

Haltestellen			
int. Nr.	HS-Nr.	Name	LINKs (Liniennr.), die diese Haltestelle anfahren
1	1001	HBF	{2001}, [2002}, {2003}, {2004}, {2005}, {2006}, {2009}, {2003,2004,2006}, {2003,2006}, {2005,2006}
2	1002	Post	{2004}, {2007}, {2008}
3	1003	Bank	{2004}, {2005}, {2008}
4	1004	Kirche	{2004}, {2007}, {2008}
5	1005	Markt	{2005}, {2007}, {2008}, {2009}
6	1006	Strand	{2006}
7	1007	Hafen	{2001}, {2005}, {2006}, {2005,2006}
8	1008	Leuchtturm	{2001}, {2008}
9	1009	Magnetberg	{2001}, {2003}
10	1010	Palast	{2002}, {2003}, {2005}, {2007}, {2008}
11	1011	Räuberburg	{2003}, {2007}, {2008}
12	1012	TurTur	{2003}, {2004}, {2006}, {2003,2004,2006}, {2003,2006}
13	1013	Wüste	{2003}, {2006}, {2003,2006}
14	1014	Oase	{2003}
15	1015	Grotte	{2002}, {2003}, {2002,2003}
16	1016	Vulkan	{2002}, {2003}, {2002,2003}
17	1017	Drachenhöhle	{2002}

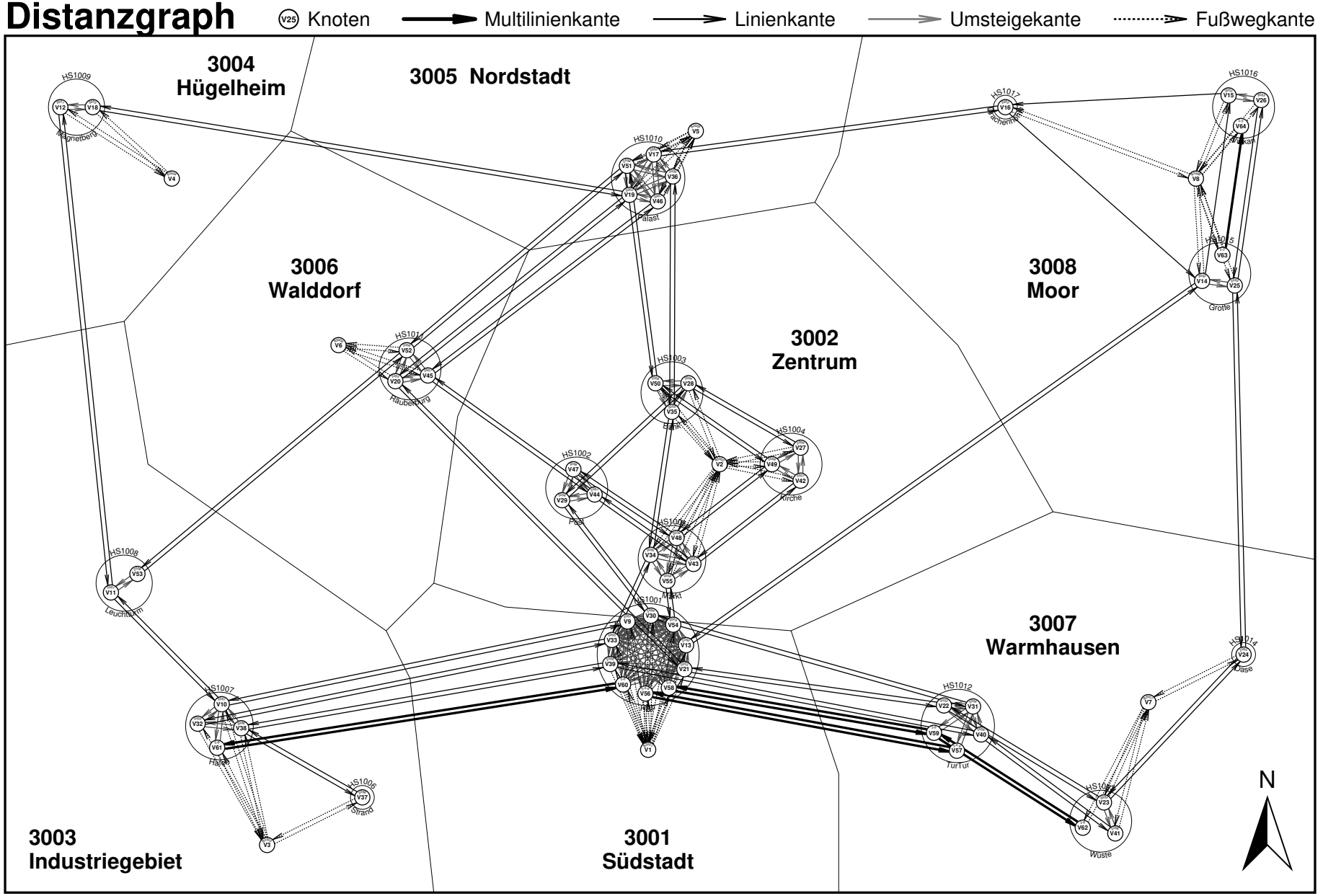
Linienliste von Lummerland

Linien				
int. Nr.	Liniennr.	Verkehrs-mittel	Takt (Min.)	Linienverlauf (Haltestellennummern)
1	2001	Eisenb.	35,00	Hinweg 1001, 1007, 1008, 1009 Rückweg 1009, 1008, 1007, 1001
2	2002	S-Bahn	20,00	Hinweg 1001, 1015, 1016, 1017, 1010 Rückweg 1010, 1017, 1015, 1001
3	2003	S-Bahn	15,00	Hinweg 1009, 1010, 1011, 1001, 1012, 1013, 1014, 1015, 1016 Rückweg 1016, 1015, 1014, 1013, 1012, 1001, 1011, 1010
4	2004	S-Bahn	10,00	Hinweg 1004, 1003, 1002, 1001, 1012 Rückweg 1012, 1001, 1002, 1003, 1004
5	2005	S-Bahn	12,00	Hinweg 1007, 1001, 1005, 1003, 1010 Rückweg 1010, 1003, 1005, 1001, 1007
6	2006	S-Bahn	15,00	Hinweg 1006, 1007, 1001, 1012, 1013 Rückweg 1013, 1012, 1001, 1007, 1006
7	2007	Bus	15,00	Hinweg 1004, 1005, 1002, 1011, 1010 Rückweg 1010, 1011, 1002, 1005, 1004
8	2008	Stadtbahn	12,00	Hinweg 1002, 1005, 1004, 1003, 1010, 1011, 1008 Rückweg 1008, 1011, 1010, 1003, 1004, 1005, 1002
9	2009	Fußg.	0,50	Hinweg 1001, 1005 Rückweg 1005, 1001

Globale Linkliste

Verkehrsmittel	Liniennummern
S-Bahn	{2003,2004,2005}
S-Bahn	{2005,2006}
S-Bahn	{2003,2006}
S-Bahn	{2002,2003}

Distanzgraph



Knotenliste

V-Nr.	LINK	Repräsentant im Netz
1	{}	Bezirk 3001
2	{}	Bezirk 3002
3	{}	Bezirk 3003
4	{}	Bezirk 3004
5	{}	Bezirk 3005
6	{}	Bezirk 3006
7	{}	Bezirk 3007
8	{}	Bezirk 3008
9	{2001}	Haltestelle 1001
10	{2001}	Haltestelle 1007
11	{2001}	Haltestelle 1008
12	{2001}	Haltestelle 1009
13	{2002}	Haltestelle 1001
14	{2002}	Haltestelle 1015
15	{2002}	Haltestelle 1016
16	{2002}	Haltestelle 1017
17	{2002}	Haltestelle 1010
18	{2003}	Haltestelle 1009
19	{2003}	Haltestelle 1010
20	{2003}	Haltestelle 1011
21	{2003}	Haltestelle 1001
22	{2003}	Haltestelle 1012
23	{2003}	Haltestelle 1013
24	{2003}	Haltestelle 1014
25	{2003}	Haltestelle 1015
26	{2003}	Haltestelle 1016
27	{2004}	Haltestelle 1004
28	{2004}	Haltestelle 1003
29	{2004}	Haltestelle 1002
30	{2004}	Haltestelle 1001
31	{2004}	Haltestelle 1012
32	{2005}	Haltestelle 1007
33	{2005}	Haltestelle 1001
34	{2006}	Haltestelle 1005
35	{2006}	Haltestelle 1003
36	{2006}	Haltestelle 1010
37	{2006}	Haltestelle 1006
38	{2006}	Haltestelle 1007
39	{2006}	Haltestelle 1001
40	{2006}	Haltestelle 1012
41	{2006}	Haltestelle 1013
42	{2007}	Haltestelle 1004
43	{2007}	Haltestelle 1005
44	{2007}	Haltestelle 1002
45	{2007}	Haltestelle 1011
46	{2007}	Haltestelle 1010
47	{2008}	Haltestelle 1002
48	{2008}	Haltestelle 1005
49	{2008}	Haltestelle 1004
50	{2008}	Haltestelle 1003
51	{2008}	Haltestelle 1010
52	{2008}	Haltestelle 1011
53	{2008}	Haltestelle 1008
54	{2009}	Haltestelle 1001
55	{2009}	Haltestelle 1005
56	{2003,2004,2006}	Haltestelle 1001
57	{2002,2004,2006}	Haltestelle 1012
58	{2003,2006}	Haltestelle 1001
59	{2003,2006}	Haltestelle 1012
60	{2005,2006}	Haltestelle 1001
61	{2005,2006}	Haltestelle 1007
62	{2003,2006}	Haltestelle 1013
63	{2002,2003}	Haltestelle 1015
64	{2002,2003}	Haltestelle 1016

Kantenliste ohne Fußwege

Von	Nach	Gewicht	LINK
9	10	353.35	{2001}
9	13	750.00	{}
9	21	600.00	{}
9	30	450.00	{}
9	33	510.00	{}
9	39	600.00	{}
9	54	165.00	{}
9	56	278.57	{}
9	58	375.00	{}
9	60	350.00	{}
10	9	353.35	{2001}
10	11	133.70	{2001}
10	32	510.00	{}
10	38	600.00	{}
10	61	350.00	{}
11	10	133.70	{2001}
11	12	382.00	{2001}
11	53	510.00	{}
12	11	382.00	{2001}
12	18	600.00	{}
13	9	1050.00	{}
13	14	372.65	{2002}
13	21	600.00	{}
13	30	450.00	{}
13	33	510.00	{}
13	39	600.00	{}
13	54	165.00	{}
13	56	278.57	{}
13	58	375.00	{}
13	60	350.00	{}
14	13	372.65	{2002}
14	15	89.95	{2002}
14	25	600.00	{}
15	16	128.50	{2002}
15	26	600.00	{}
16	14	147.77	{2002}
16	17	199.18	{2002}
17	16	199.18	{2002}
17	19	600.00	{}
17	36	510.00	{}
17	46	510.00	{}
17	51	510.00	{}
18	12	1050.00	{}
18	19	308.40	{2003}
19	17	750.00	{}
19	18	308.40	{2003}
19	20	167.05	{2003}
19	36	510.00	{}
19	46	510.00	{}
19	51	510.00	{}
20	19	167.05	{2003}
20	21	199.18	{2003}
20	45	510.00	{}
20	52	510.00	{}
21	9	1050.00	{}
21	13	750.00	{}
21	20	199.18	{2003}
21	22	167.05	{2003}
21	30	450.00	{}
21	33	510.00	{}
21	39	600.00	{}
21	54	165.00	{}
21	60	350.00	{}

Von	Nach	Gewicht	LINK
22	21	167.05	{2003}
22	23	89.95	{2003}
22	31	450.00	{}
22	40	600.00	{}
23	22	89.95	{2003}
23	24	115.65	{2003}
23	41	600.00	{}
24	23	115.65	{2003}
24	25	205.60	{2003}
25	14	750.00	{}
25	24	205.60	{2003}
25	26	89.95	{2003}
26	15	750.00	{}
26	25	89.95	{2003}
27	28	77.10	{2004}
27	42	510.00	{}
27	49	510.00	{}
28	27	77.10	{2004}
28	29	70.68	{2004}
28	35	510.00	{}
28	50	510.00	{}
29	28	70.68	{2004}
29	30	96.38	{2004}
29	44	510.00	{}
29	47	510.00	{}
30	9	1050.00	{}
30	13	750.00	{}
30	21	600.00	{}
30	29	96.38	{2004}
30	31	167.05	{2004}
30	33	510.00	{}
30	39	600.00	{}
30	54	165.00	{}
30	58	375.00	{}
30	60	350.00	{}
31	22	600.00	{}
31	30	167.05	{2004}
31	40	600.00	{}
31	59	375.00	{}
32	10	1050.00	{}
32	33	237.73	{2005}
32	38	600.00	{}
33	9	1050.00	{}
33	13	750.00	{}
33	21	600.00	{}
33	30	450.00	{}
33	32	237.73	{2005}
33	34	51.40	{2005}
33	39	600.00	{}
33	54	165.00	{}
33	56	278.57	{}
33	58	375.00	{}
34	33	51.40	{2005}
34	35	89.95	{2005}
34	43	510.00	{}
34	48	510.00	{}
34	55	165.00	{}
35	28	450.00	{}
35	34	89.95	{2005}
35	36	115.65	{2005}
35	50	510.00	{}
36	17	750.00	{}
36	19	600.00	{}

Von	Nach	Gewicht	LINK
36	35	115.65	{2005}
36	46	510.00	{}
36	51	510.00	{}
37	38	83.53	{2006}
38	10	1050.00	{}
38	32	510.00	{}
38	37	83.53	{2006}
38	39	237.73	{2006}
39	9	1050.00	{}
39	13	750.00	{}
39	21	600.00	{}
39	30	450.00	{}
39	33	510.00	{}
39	38	237.73	{2006}
39	40	167.05	{2006}
39	54	165.00	{}
40	22	600.00	{}
40	31	450.00	{}
40	39	167.05	{2006}
40	41	89.95	{2006}
41	23	600.00	{}
41	40	89.95	{2006}
42	27	450.00	{}
42	43	92.15	{2007}
42	49	510.00	{}
43	34	510.00	{}
43	42	92.15	{2007}
43	44	70.50	{2007}
43	48	510.00	{}
43	55	165.00	{}
44	29	450.00	{}
44	43	70.50	{2007}
44	45	120.35	{2007}
44	47	510.00	{}
45	20	600.00	{}
45	44	120.35	{2007}
45	46	183.30	{2007}
45	52	510.00	{}
46	17	750.00	{}
46	19	600.00	{}
46	36	510.00	{}
46	45	183.30	{2007}
46	51	510.00	{}
47	29	450.00	{}
47	44	510.00	{}
47	48	49.25	{2008}
48	34	510.00	{}
48	43	510.00	{}
48	47	49.25	{2008}
48	49	64.53	{2008}
48	55	165.00	{}
49	27	450.00	{}
49	42	510.00	{}
49	48	64.53	{2008}
49	50	59.10	{2008}

Von	Nach	Gewicht	LINK
50	28	450.00	{}
50	35	510.00	{}
50	49	59.10	{2008}
50	51	88.65	{2008}
51	17	750.00	{}
51	19	600.00	{}
51	36	510.00	{}
51	46	510.00	{}
51	50	88.65	{2008}
51	52	128.05	{2008}
52	20	600.00	{}
52	45	510.00	{}
52	51	128.05	{2008}
52	53	147.75	{2008}
53	11	1050.00	{}
53	52	147.75	{2008}
54	9	1050.00	{}
54	13	750.00	{}
54	21	600.00	{}
54	30	450.00	{}
54	33	510.00	{}
54	39	600.00	{}
54	55	171.40	{2009}
54	56	278.57	{}
54	58	375.00	{}
54	60	350.00	{}
55	34	510.00	{}
55	43	510.00	{}
55	48	510.00	{}
55	54	171.40	{2009}
56	9	1050.00	{}
56	13	750.00	{}
56	33	510.00	{}
56	54	165.00	{}
56	57	167.05	{2003,2004,2006}
57	56	167.05	{2003,2004,2006}
58	9	1050.00	{}
58	13	750.00	{}
58	30	450.00	{}
58	33	510.00	{}
58	54	165.00	{}
58	59	167.05	{2003,2006}
59	31	450.00	{}
59	58	167.05	{2003,2006}
59	62	89.95	{2003,2006}
60	9	1050.00	{}
60	13	750.00	{}
60	21	600.00	{}
60	30	450.00	{}
60	54	165.00	{}
60	61	237.73	{2005,2006}
61	10	1050.00	{}
61	60	237.73	{2005,2006}
62	59	89.95	{2003,2006}
63	64	89.95	{2002,2003}

Die Netzdaten der vorherigen Seiten zusammen mit der Belastungsdatei LUMMER.VD, die alle möglichen Bezirkspaare mit einer Belastung von je 1000 Fahrten pro Tag enthält, haben bei der Mehrwegumlegung folgende Ergebnisse geliefert (Die ermittelten Weg sind nur auszugsweise abgedruckt):

```
BERECHNUNG    Mehrweg-Umlegung
NETZ          Lummerland
PARAMETER     UMS-Faktor       :    150.00  Sekunden pro Umsteigen
GRENZEN       relativ/absolut  :      0.20 %  /  900.00 Sekunden
DATA-STATUS        17 HS          9 LIN         8 BEZ
              HVZ 240 min   NVZ 540 min   SVZ 360 min
GRAPH-STATUS        Anzahl Knoten (V) :      64
                    Anzahl Kanten (E) :     236

Von Bezirk 3001 nach Bezirk 3002
    Gesamtbelastung  :   1000 Fahrten/Tag
    Von Yen gefunden :      1 Weg
    Nach Wegecheck   :      1 Weg
    1. Weg : 1-54-55-2                    Gewicht =  540.57  Belast. = 1000

Von Bezirk 3001 nach Bezirk 3004
    Gesamtbelastung  :   1000 Fahrten/Tag
    Von Yen gefunden :      3 Wege
    Nach Wegecheck   :      2 Wege
    1. Weg : 1-21-20-19-18-4              Gewicht = 1439.63  Belast. =  800
    2. Weg : 1-9-10-11-12-4               Gewicht = 1634.05  Belast. =  200

Von Bezirk 3002 nach Bezirk 3004
    Gesamtbelastung  :   1000 Fahrten/Tag
    Von Yen gefunden :     13 Wege
    Nach Wegecheck   :      4 Wege
    1. Weg : 2-50-51-19-18-4              Gewicht = 1711.22  Belast. =  353
    2. Weg : 2-35-36-19-18-4              Gewicht = 1738.22  Belast. =  330
    3. Weg : 2-55-54-21-20-19-18-4        Gewicht = 1856.86  Belast. =  229
    4. Weg : 2-43-44-45-20-19-18-4        Gewicht = 2022.13  Belast. =   88

Von Bezirk 3002 nach Bezirk 3008
    Gesamtbelastung  :   1000 Fahrten/Tag
    Von Yen gefunden :     23 Wege
    Nach Wegecheck   :      7 Wege
    1. Weg : 2-55-54-13-14-8              Gewicht = 1663.22  Belast. =  269
    2. Weg : 2-55-54-21-22-23-24-25-8     Gewicht = 1718.82  Belast. =  230
    3. Weg : 2-50-51-17-16-14-8           Gewicht = 1858.10  Belast. =  135
    4. Weg : 2-35-36-17-16-14-8           Gewicht = 1885.10  Belast. =  117
    5. Weg : 2-28-29-30-13-14-8           Gewicht = 1902.20  Belast. =  105
    6. Weg : 2-34-33-21-22-23-24-25-8     Gewicht = 1943.82  Belast. =   76
    7. Weg : 2-28-29-30-31-22-23-24-25-8  Gewicht = 1957.80  Belast. =   67

Von Bezirk 3003 nach Bezirk 3008
    Gesamtbelastung  :   1000 Fahrten/Tag
    Von Yen gefunden :     27 Wege
    Nach Wegecheck   :      4 Wege
    1. Weg : 3-61-60-13-14-8              Gewicht = 1956.21  Belast. =  400
    2. Weg : 3-61-60-21-22-23-24-25-8     Gewicht = 2011.81  Belast. =  354
    3. Weg : 3-10-9-13-14-8               Gewicht = 2261.83  Belast. =  146
    4. Weg : 3-10-9-21-22-23-24-25-8      Gewicht = 2317.43  Belast. =  100

Insgesamt gefundene Wege von Yen        :  243 Wege
Insgesamt nach Wegecheck belastete Wege :  108 Wege
Durchschnittliche Knotenzahl pro Weg      6.84 Knoten
```

Die Dateien mit den Ausgangsdaten LUMMER.OVE und LUMMER.VD, sowie die Ergebnisdatei LUMMER.UML können am Institut für Eisenbahn- und Verkehrswesen der Universität Stuttgart eingesehen werden.

Als Ergebnis der Umlegung sind auf der nächsten Seite die Streckenbelastungen und die Haltestellenbelastungen im Netzes dargestellt.

Haltestellen-belastungen

Lage in Netz

Modell Lummerland

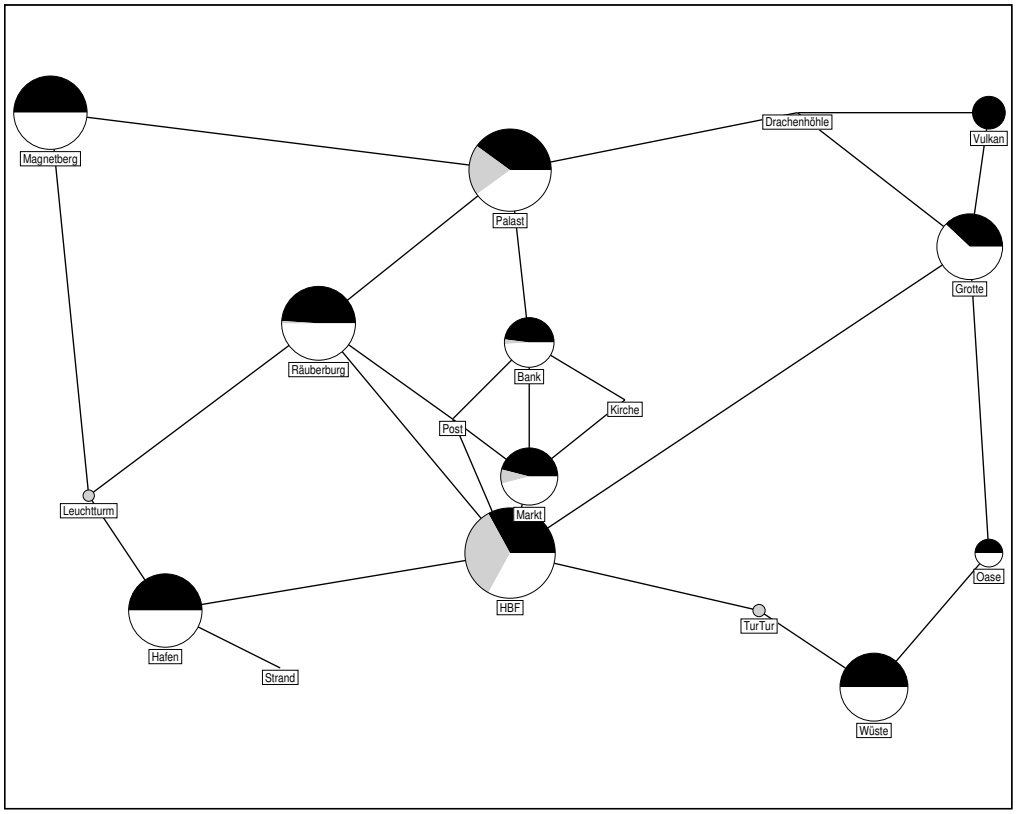
Alle Haltestellen

Belastung
[Personen/Tag]

15000
10000
5000
500
0

1 cm Durchmesser
= 5305 Personen

Einsteiger
Umsteiger
Aussteiger



4. Juli 1996

Universität Stuttgart
Institut für Eisenbahn- und Verkehrswesen

Strecken-belastungen

Lage in Netz

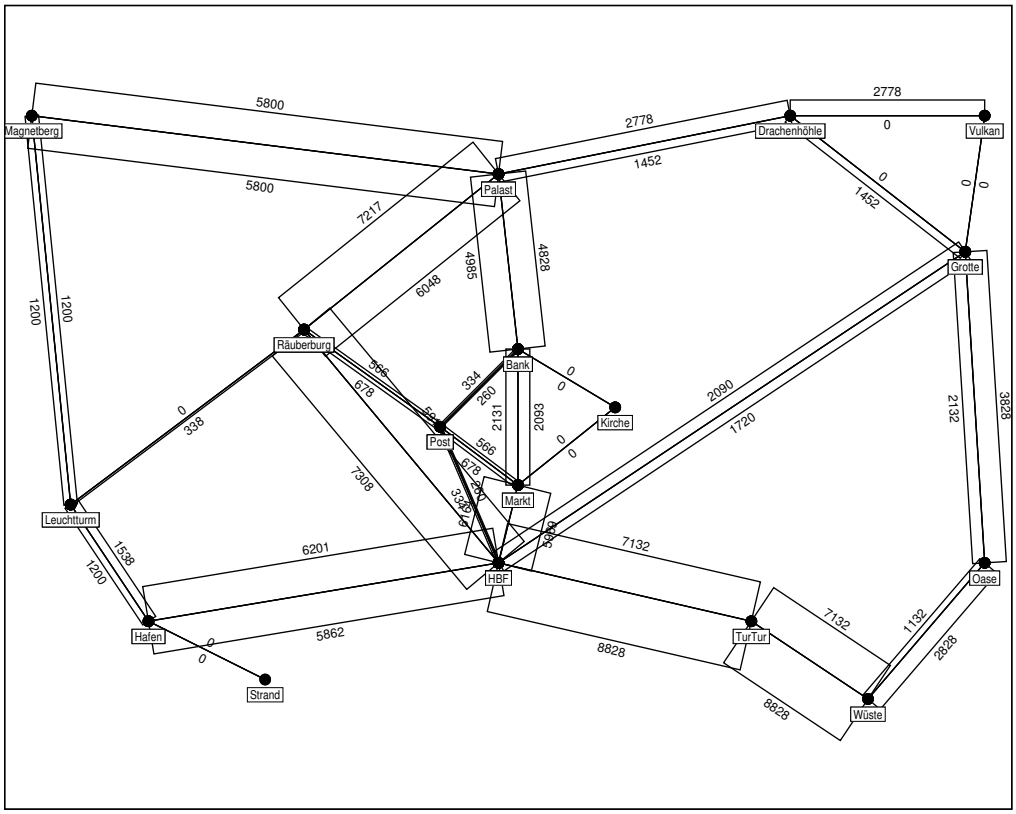
Modell Lummerland

Alle Strecken

Belastungen für Richtung und Gegenrichtung getrennt.

2500
5000
7500

Belastung
[Personen / Tag + Richtung]
1 cm = 7980 Personen



4. Juli 1996

Universität Stuttgart
Institut für Eisenbahn- und Verkehrswesen

Literaturverzeichnis

[Aho 83] Aho, A.V., Hopcroft, J.E., Ullman, J.D. (1983): Data Structures and Algorithms, Addison-Wesley

[Atluja 93] Atluja, R.K., Magnanti, T.L., Orlin, J.B. (1993): **Network Flows - Theory, Algorithms and Applications**, Prentice Hall, New Jersey

[Azevedo 90] de Azevedo, J.A., Madeira, J.J., Martins, E.Q., Pires, F.M. (1990): **A shortest path ranking algorithm**, in:
AIRO '90 - Models and Methods for Decision Support, Sorrento (Italy), Proceedings of the Annual Conference, Associazione Italiana di Ricerca Operativa, October 1990, 1001-1011

[Bellman 60] Bellman, R., Kalaba, R. (1960): **On kth best policies**, Journal of SIAM 8, Nr. 4, 582-588

[Brandstädt 94] Brandstädt, A. (1994): **Graphen und Algorithmen**, B.G. Teubner Verlag

[Clarke 63] Clarke, S., Krikorian, A., Rausen, J. (1963): **Computing the N best loopless paths in a network**, Journal of SIAM 11, 1096-1102

[Diekert 93] Diekert, V. (1993): **Algorithmen und Datenstrukturen**, Skript zur Vorlesung, Fakultät Informatik, Universität Stuttgart

[Dijkstra 59] Dijkstra, E.W. (1959): **A Note on Two Problems in Connexion with Graphs**, Numerische Mathematik 1, 269-271

[Dobeschinsky 89] Dobeschinsky, H. (1989): **Verkehrsprognose**, Skript zur Vorlesung „Projektstudie Ostfildern", Fakultät Bauingenieurwesen, Universität Stuttgart

[Dreyfus 69] Dreyfus, S.E. (1969): **An appraisal of some shortest-path algorithms**, Operations Research 17, Band 1969-1, 395-412

[Floyd 62] Floyd, R.W. (1962): **Algorithm 97 - shortest path**, Communictions of the ACM 5, Nr. 6, 345

[Fox 73] Fox, B.L. (1973): **Calculating kth shortest paths**, INFOR 11, 66-70

[Fox 75] Fox, B.L. (1975): **More on kth Shortest Paths**, Communications of the ACM 18, Nr. 5, 279

[Grote 89] Grote, U. (1989): **Verkehrserhebungen im öffentlichen Personennahverkehr**, Skript zur Vorlesung „Projektstudie Ostfildern", Fakultät Bauingenieurwesen, Universität Stuttgart

[Hoffman 59] Hoffman, W., Pavley, R. (1959): **A Method for the Solution of the Nth Best Path Problem**, Journal of the ACM 6, Nr. 4, 506-514

[Hoffmann 93] Hoffmann, J. (1993): **Entwicklung eines Programmsystems zur Wegsuche in Verkehrsnetzen des Öffentlichen und Individualverkehrs unter besonderer Berücksichtigung des P+R-Verkehrs**,
Diplomarbeit Nr. 1094, Fakultät Informatik, Universität Stuttgart

[[Lawler 77] Lawler, E.L. (1977): **Comments on Computing the k Shortest Paths in a Graph**, Communications of the ACM 20, Nr. 8, 603-604

Minieka 74] Minieka, E. (1974): **On Computing Sets of Shortest Paths in a Graph**, Communications of the ACM 17, Nr. 6, 351-353

[Ottmann 93] Ottmann, T., Widmayer, P. (1993): **Algorithmen und Datenstrukturen**, BI Wissenschaftsverlag, 2. Aufl.

[Perko 86] Perko, A. (1986): **Implementation of Algorithms for K Shortest Loopless Paths**, Networks 16, 149-160

[Pollack 61a] Pollack, M. (1961): **The kth best route through a network**, Operations Research 9, 578-580

[Pollack 61b] Pollack, M. (1961): **Solutions of the kth Best Route Through a Network - A Review**, Journal of Mathematical Analysis and Applications 3, 547-559

[Schedel 90] Schedel, F. (1990): **Programmsystem zur Durchführung von Umlegungsrechnungen in Netzen des öffentlichen Personennahverkehrs und deren Darstellung auf postscriptfähigen Laserdruckern**, Diplomarbeit Nr. 760, Fakultät Informatik, Universität Stuttgart

[Sedgewick 92] Sedgewick, R. (1992): **Algorithmen**, Addison-Wesley

[Shier 76] Shier, D.R. (1976): **Iterative Methods for Determining the k Shortest Paths in a Network**, Networks 6, 205-229

[Theune 95] Theune, D. (1995): **Robuste und effiziente Methoden zur Lösung von Wegproblemen**, B.G. Teubner Verlag

[Winston 89] Winston, P.H. (1989): **Künstliche Intelligenz**, Addison-Wesley

[Yen 71] Yen, J.Y. (1971): **Finding the k shortest loopless paths in a network**, Management Science 17, Nr. 11, 712-716

und

WWW Universität Karlsruhe http://liinwww.ira.uka.de/bibliography/Theory/k-path.html

Erklärung

Ich versichere, daß ich diese Arbeit selbständig verfaßt
und nur die angegebenen Quellen und Hilfsmittel
verwendet habe.

Sindelfingen, den 11. Juli 1996

MARCUS MACK
Burghaldenstraße 82/1
71065 Sindelfingen
Tel: (07031) 877973