Technische Universität Berlin

FACHBEREICH 3
MATHEMATIK

VERTEILTE VERBINDUNGSSUCHE IM ÖFFENTLICHEN PERSONENVERKEHR: GRAPHENTHEORETISCHE MODELLE UND ALGORITHMEN

by

ROLF H. MÖHRING

No. 624/99

Verteilte Verbindungssuche im öffentlichen Personenverkehr: Graphentheoretische Modelle und Algorithmen

Rolf H. Möhring
Technische Universität Berlin *

5. März 1999

Zusammenfassung

Inzwischen verfügen viele Verkehrsunternehmen über lokale, computerbasierte Auskunftsysteme, bei denen Informationen über Verbindungen im jeweiligen System abrufbar sind. Derzeit entstehen Kopplungen solcher Systeme zu übergreifenden Auskunftsystemen, die die unterschiedlichen Teilsysteme und Datenbestände in einer *verteilten Verbindungssuche* geeignet kombinieren.

Diese Arbeit behandelt die verteilte Verbindungssuche aus graphentheoretischer Sicht. Im ersten Teil wird untersucht, wie weit sich unterschiedliche Abfragen auf die Berechnung kürzester Wege in einem geeignet definierten Graphen zurückführen lassen. Die Rückführung auf ein Graphenmodell eröffnet zum einen den Zugang zu Standardalgorithmen für eine Fülle ähnlich gelagerter, auch mehrkriterieller Kürzeste-Wege Berechnungen, und bietet aus struktureller Sicht den geeigneten Rahmen für die verteilte Verbindungsabfrage, die dann eng mit Dekompositionsmöglichkeiten des Graphen zusammenhängt.

Im zweiten Teil wird dann untersucht, welche Anfragen sich in einem verteilten System berechnen lassen und wie groß der dafür benötigte Aufwand ist. Insbesondere wird der Frage nachgegangen, welche Daten von den Teilsystemen bereit gestellt werden müssen, ob sich Pre-Processing dieser Daten lohnt und über welche Techniken ein Master-Algorithmus verfügen muß, der mit den Teilsystemen kommuniziert. Dabei wird insbesondere auf (approximative) Methoden für mehrkriterielle Zielfunktionen eingegangen.

Es zeigt sich, daß sich nahezu alle Standardabfragekriterien wie Reisezeit, Anzahl der Umstiege, Wahl des Verkehrsmittels sowie Kombinationen hiervon bei einer verteilten Abfrage berücksichtigen lassen, wobei der Rechenaufwand vom Grad der Vernetzung der Teilsysteme abhängt. Auch mehrkriterielle Abfragen sind möglich.

1 Verbindungsabfragen als Kürzeste-Wege Problem in Graphen

1.1 Ein Graphenmodell für Verbindungsabfragen

Die Basisaufgabe einer Verbindungsabfrage besteht darin, zwischen verschiedenen Orten a und b eine Verbindung zu suchen, wobei verschiedene Verkehrssysteme (Nahverkehr, Fernverkehr, Bahn, Flugzeug) involviert sind und verschiedene Kriterien an die Art der Verbindung angelegt werden können (Zeitfenster für die Abfahrt bzw. Ankunft, Kriterien an die Art der Verbindung wie Dauer, Anzahl der Umstiege, Ausschluß bestimmter Verkehrsmittel usw.).

In diesem Abschnitt wird der Frage nachgegangen, ob bzw. für welche Abfragekriterien sich das zugehörige Abfrageproblem durch ein Kürzeste-Wege-Problem in einem Graphen modellieren läßt.

* Technische Universität Berlin, Fachbereich Mathematik, Straße des 17. Juni 136, 10623 Berlin, email moehring@math.tu-berlin.de, WWW http://www.math.tu-berlin.de/˜moehring/. Diese Arbeit entstand in Kooperation mit dem Forschungsinstitut für anwendungsorientierte Wissensverarbeitung (FAW), Ulm, im Rahmen des Vorhabens DELFI (*D*eutschlandweite *EL*ektronische *F*ahrplan-*I*nformation).

Ist dies der Fall, so stehen effiziente Standardalgorithmen zur Lösung der Verbindungsabfrage zur Verfügung.

Dabei ist natürlich zu beachten, daß der zur Durchführung der Abfrage letztlich implementierte Algorithmus im allgemeinen nicht den Graphen konstruieren muß, sondern direkt auf den vorliegenden Daten operiert, wobei dann seine Suchstrategie implizit der Suche nach einem kürzesten Weg in einem Graphen entspricht.

Die Modellierung mit Graphen hat jedoch den methodischen Vorteil, daß die Vielzahl neuer Forschungsergebnisse und Methoden über Graphenalgorithmen und Dekomposition von Graphen direkt auf die verteilte Verbindungsabfrage übertragbar und anwendbar sind.

1.1.1 Der Liniengraph

Die Ausgangsdaten für die Verbindungssuche sind Fahrplandaten. Diese können in unterschiedlicher Form vorliegen. Wir gehen davon aus, daß auf jeden Fall *Information über die Linien* vorhanden ist. Zur Information über eine Linie gehört für jede mögliche Abfahrtzeit am Start eine Liste der angefahrenen Orte in der Reihenfolge der Route, Ankunft und Abfahrtzeiten, Tarifinformation, Umsteigemöglichkeiten auf andere Linien usw.

Aus einer Menge von Linien ergibt sich der zugehörige *Liniengraph*. Dies ist ein (im allgemeinen gerichteter) Graph[1], der als *Knoten* die Orte aller Linien enthält, und in dem die *Kanten* genau je zwei aufeinander folgende Orte einer Linie verbinden. Der Liniengraph enthält also Informationen über prinzipiell mögliche Verbindungen (Erreichbarkeitsinformation), aber noch keine Zeit- und Tarifinformation wie Abfahrtzeiten, Fahrtdauer, und Preise. Abbildung 1 gibt als Beispiel eines Liniengraphen das S- und U-Bahnnetz von Berlin an.

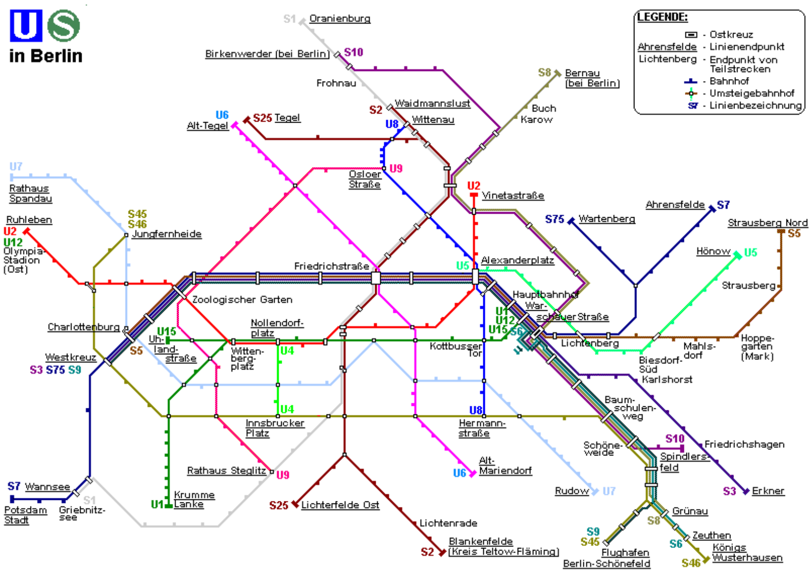

Abbildung 1: Das U- und S-Bahnnetz von Berlin als Beispiel eines Liniengraphen

1.1.2 Grundlagen einer Verbindungsabfrage

Für die Verbindungsabfragen werden für jede Linie (d. h. insbesondere *für jede mögliche Abfahrzeit der Linie*) pro Kante (i, j) des Verbindungsgraphen Fahrplandaten der Form

$$(\text{Ort } i, \text{Ort } j, \text{Abfahrtzeit } t_i, \text{Ankunftszeit } t_j, \text{Art des Verkehrsmittels}, \ldots)$$

[1] Für Grundlagen über Graphen sei auf [5] verwiesen.

benötigt. Hier können weitere Informationen wie Anzahl der Umstiege, Tarifinformation usw. abgelegt werden.

Ein solcher Datensatz soll im folgenden *Elementarverbindung* genannt werden. Alle weiteren Überlegungen zur verteilten Verbindungssuche basieren darauf, daß *die Teilsysteme mindestens diese Daten liefern können*.

Eine Verbindung von a nach b ist nun ein Folge solcher Elementarverbindungen, wobei zwischen Elementarverbindungen beim Wechsel der Linie noch geeignet Übergänge zu berücksichtigen sind. Diese Übergänge modellieren die Zeit, die erforderlich ist, falls ein Umstieg erfolgt und enthalten die Wartezeit auf den nächsten Anschluß.

Stellt man diese Abfolge von Elementarverbindungen und Umstiegen im Ort-Zeit-Diagramm dar, so ergibt sich die Situation aus Abbildung 2.

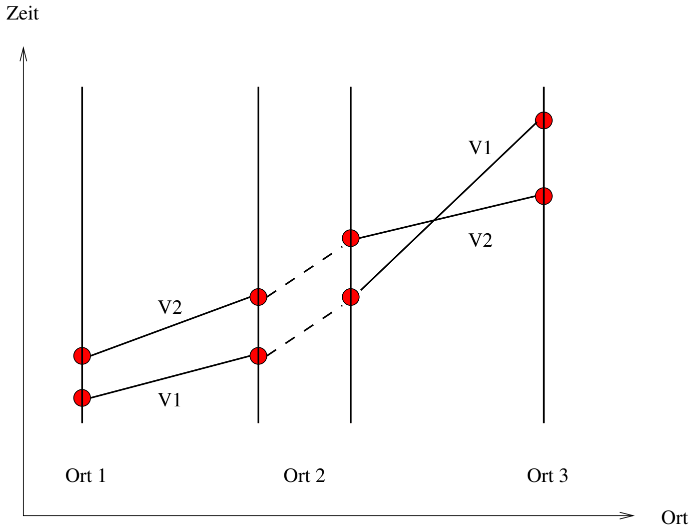

Abbildung 2: Ort-Zeit Diagramm

Die Umstiege sind dabei gestrichelt gekennzeichnet. Man erkennt das Phänomen, daß eine spätere Verbindung früher am Ziel sein kann, wenn bei den Umstiegen jeweils die nächst mögliche Verbindung genommen wird. Es müssen daher bei Umstiegen auch Wartezeiten auf spätere Verbindungen in Betracht gezogen werden. Hat man auf diese Weise die früheste Ankunftszeit am Zielort ermittelt, so erfolgt üblicherweise eine Rückwärtsrechnung, um zu der frühesten Ankunftszeit am Ziel die spätest mögliche Abfahrtzeit am Start zu ermitteln. Auf diese Weise würde man in Abbildung 1, ausgehend von Verbindung $V1$ zunächst im Ort 2 den Umstieg auf die schnellere Verbindung $V2$ ermitteln (gestrichelte Linie) und dann bei der Rückrechnung den Start am Ort 1 mit der Verbindung $V2$. Dies schließt jedoch nicht aus, daß am Ort 1 eine etwas frühere Verbindung existiert, die noch schneller zum Ziel führen könnte. Aus diesen Gründen muß bei der Verbindungsabfrage immer um eine angegebene Abfahrtzeit (Ankunftszeit) herum ein vernünftiges Zeitintervall gewählt werden, innerhalb dessen Verbindungen gesucht werden.

1.1.3 Der Ort-Zeit Graph

Diese Standardtechniken lassen sich direkt in ein graphentheoretisches Modell zur Verbindungssuche umsetzen. Der zugrunde liegende Graph $G = (V, E)$ wird im weiteren als *Ort-Zeit Graph* bezeichnet. *Knoten* in diesem Graphen sind die Orte mit den jeweiligen Abfahrts- bzw. Ankunftzeiten. Jeder Ort tritt also so oft als Knoten auf, wie Abfahrten von ihm bzw. Ankünfte in ihm möglich sind. *Kanten* sind einerseits die Elementarverbindungen sowie andererseits die Umstiege. Eine Weiterfahrt auf derselben Linie ist formal gesehen dann ebenfalls ein Umstieg. Die zugehörige Kante verbindet den entsprechenden Ankunftsknoten (Ort, Ankunftszeit) mit dem entsprechenden Abfahrtsknoten (Ort, Abfahrtzeit), vgl. Abbildung 3.

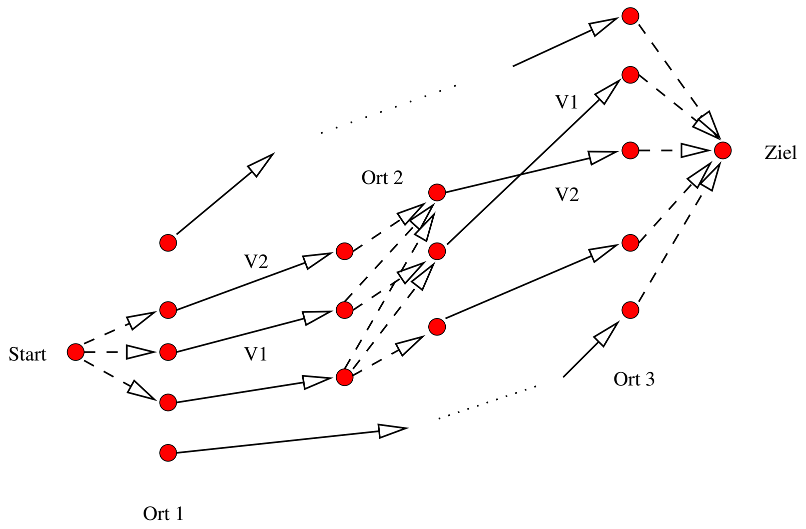

Abbildung 3: Ein Ausschnitt aus dem Ort-Zeit Graphen für das Ort-Zeit Diagramm aus Abb. 2

Für eine konkrete Verbindungsabfrage wird zusätzlich ein Startknoten *Start* eingeführt und mit Kanten zu allen Knoten verbunden, die zu in Frage kommenden Abfahrtzeiten am Startort korrespondieren. Entsprechend wird ein zusätzlicher Zielknoten *Ziel* eingeführt, zu dem Kanten von allen Knoten führen, die Ankünfte am Ziel repräsentieren (wobei eine Beschränkung auf ein Zeitintervall mit vorab berechneten oberen und unteren Schranken möglich, aber für die weiteren Überlegungen ohne Bedeutung ist).

In dem so definierten Ort-Zeit Graph G entsprechen offenbar alle gerichteten Wege vom Knoten *Start* zum Knoten *Ziel* genau den möglichen Verbindungen innerhalb des bei *Start* gewählten Zeitfensters, das durch die von *Start* ausgehenden Kanten repräsentiert wird. Daher lassen sich *alle Verbindungsabfragen als Suche nach einem* (geeignet bewerteten) *Weg* in G durchführen. Insbesondere ist *keine Rückwärtsrechnung* mehr erforderlich, da die durch sie gefundenen Verbindungen bereits als Wege im Ort-Zeit Graphen G enthalten sind[2].

Ein anderer Ansatz wird in [9] verfolgt. Anstelle der hier vorgenommenen Expansion der Orte werden zeitabhängige Kantenbewertungen eingeführt. Der Ansatz beschränkt sich jedoch auf einkriterielle Zielfunktionen (Reisezeit).

1.1.4 Erweiterungen des Ort-Zeit Graphen

Bisher wurden aus den Linien nur die Elementarverbindungen zwischen aufeinanderfolgenden Orten im Ort-Zeit Graph berücksichtigt. Natürlich ist es auch möglich, längere Streckenabschnitte einer Linie als Verbindung anzusehen und sie als entsprechende Kante im Ort-Zeit Graphen einzufügen. Dies ist natürlich nicht auf Teilstrecken einer Linie beschränkt, sondern auf zusammengesetzte Verbindungen über verschiedene Linien hinweg ausdehnbar, so daß im Extremfall zwischen je zwei Orten Kanten (zu den entsprechenden Zeiten) verlaufen könnten. In Abbildung 4 geben die bogenförmigen Kanten solche zusammengesetzten Verbindungen an.

Die Bewertung solcher nicht-elementarer Verbindungen muß natürlich dann im Ort-Zeit Graphen verfügbar sein, also berechnet oder aus bereits erstellten Rechnungen abgelesen werden. Solche

[2] Der Graph muß natürlich bei der Verbindungsabfrage nicht explizit konstruiert werden. Wie man sieht, ist er sehr stark zu dem Ort-Zeit-Diagramm verwandt, da lediglich Kanten für Umstiege, Wartezeiten und Weiterfahrten auf derselben Linie hinzu kommen. Diese können natürlich bei der Verbindungsabfrage implizit berücksichtigt werden, so daß es nicht erforderlich ist, den gesamten Graphen zu konstruieren.

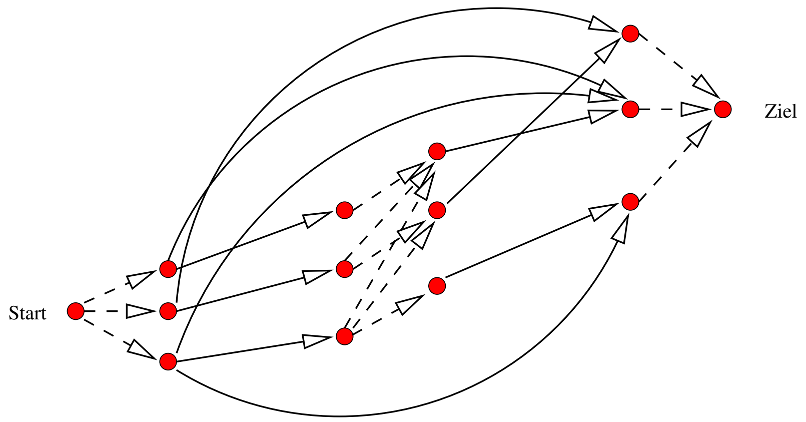

Abbildung 4: Eine Erweiterung des Ort-Zeit Graphen aus Abbildung 3

vorab durchgeführten Rechnungen bezeichnen wir als *Preprocessing*.

Durch ein solches Preprocessing kann vorab für jeden möglichen Start- und Zielort und bestimmte Zeitintervalle innerhalb eines Verkehrssystems die bezüglich der angelegten Bewertung beste Verbindung berechnet werden und entsprechend abgelegt werden, so daß man bei Angabe des Startortes, des Zielortes und der frühest möglichen Abfahrtzeit *in konstantem Aufwand die entsprechende Information erhalten kann*. In diesem Fall braucht kein Graph mehr aufgebaut zu werden, sondern die Information steht als eine Art *Table Look-up* sofort zur Verfügung. Solche Information ist, wie weiter unten in Abschnitt 4.1.2 erläutert wird, gerade bei der verteilten Verbindungsabfrage von großer Bedeutung.

1.2 Bewertungskriterien für Verbindungsabfragen

Im vorhergehenden Abschnitt wurde gezeigt, daß die Verbindungsabfrage der Suche nach einem Weg zwischen vorgegebenen Knoten (*Start* und *Ziel* in Abschnitt 1.1.3) entspricht. Um zwischen Verbindungen/Wegen zu differenzieren, werden Bewertungskriterien abgelegt. Gängige Bewertungskriterien bei der Verbindungsabfrage sind: schnellste Verbindung, kürzeste Verbindung (Bahnkilometer), Umsteigehäufigkeit, Zeitreserven bei Umstiegen, Art des Verkehrsmittels (Zuschläge, Ausstattung), Gültigkeit von Sondertarifen (Wochenendticket, Fahrt innerhalb eines Tarifgebietes), billigste Verbindung, Mindestfahrzeiten (bei Nachtzügen), touristische Attraktivität sowie mehrkriterielle Kombinationen dieser Kriterien (schnellste Verbindung mit den wenigsten Umstiegen).

1.3 Berechnung bester Verbindungen

Nachfolgend soll nun untersucht werden, ob und ggf. wie schnell die Berechnung einer besten Verbindung bezüglich des jeweiligen Kriteriums möglich ist und welche Informationen dazu benötigt werden. Ausgangspunkt dafür ist der Ort-Zeit Graph.

1.3.1 Der Dijkstra Algorithmus

Der Algorithmus von Dijkstra ist der klassische Algorithmus zur Berechnung kürzester Wege in gerichteten Graphen mit nicht-negativen Kantenbewertungen, wenn sich die Bewertung (Länge) eines Weges additiv aus den Bewertungen (Längen) seiner Kanten ergibt.

Er berechnet von einem Knoten aus kürzeste Wege zu allen anderen Knoten des Graphen, wobei die Suche abgebrochen werden kann, wenn der (die) gewünschten Zielknoten einen bestimmten Zustand ("markiert", vgl. unten) im Verlauf des Algorithmus bekommen hat.

Sei s der Startknoten und $\lambda(u, v)$ die Bewertung (Länge) der Kante (u, v). Der Dijkstra Algorithmus berechnet für jeden Knoten v eine Distanz $d[v]$, die die Bewertung (Länge) eines kürzesten Weges von s nach u angibt, sowie einen Knoten *Vorgänger*$[v]$, der der Vorgänger von v auf einem kürzesten Weg von s zu v ist. Über diese Information kann der kürzeste Weg von s nach v rekonstruiert werden.

Die Werte $d[v]$ und *Vorgänger*$[v]$ sind zunächst vorläufig, bleiben aber für einmal *markierte* Knoten endgültig und haben die zuvor beschriebene Interpretation.

Anfangs ist nur s markiert und wird

$$d[v] := \begin{cases} 0 & \text{falls } v = s \\ \lambda(s, v) & \text{falls die Kante } (s, v) \text{ existiert} \\ \infty & \text{sonst} \end{cases}$$

und

$$\textit{Vorgänger}[v] := \begin{cases} s & \text{falls die Kante } (s, v) \text{ existiert} \\ NIL & \text{sonst} \end{cases}$$

gesetzt.

Die Hauptschleife des Algorithmus ist dann gegeben durch

while es gibt einen unmarkierten Knoten **do**
    wähle einen unmarkierten Knoten v mit kleinstem $d[v]$;
    markiere v;
    for alle Kanten (v, w) mit unmarkiertem w **do**
        if $d[v] + \lambda(v, w) < d[w]$ **then**
            $d[w] := d[v] + \lambda(v, w)$;
            Vorgänger$[w] := v$;
        end if
    end for
end while

Der Dijkstra Algorithmus hat durch die Verwendung geeigneter Datenstrukturen (Fibonacci-Heaps) eine Laufzeit von $O(m + n \log n)$ Zeit, wobei m die Anzahl von Kanten und n die Anzahl der Knoten in dem Graphen ist. Hier sind jedoch viele alternative Implementationen möglich. Zu näheren Informationen sei auf [1, S. 108 ff] und [2] verwiesen.

Man erkennt an der Formulierung, daß für den Algorithmus zwei Operationen von Bedeutung sind, die *Addition* im Ausdruck

$$d[v] + \lambda(v, w)$$

und die Minimumsbildung im Vergleich

$$d[v] + \lambda(v, w) < d[w],$$

der auch als

$$\min\{d[u] + \lambda(v, w),\ d(w)\} \neq d[w]$$

geschrieben werden kann.

Daher ist der Dijkstra Algorithmus auch auf viele andere Operationen übertragbar, die ein ähnliches algebraisches Verhalten wie die Operationen min und + haben. Die Arbeiten [5, 16, 18] entwickeln (über sogenannte *Semiringe*) einen allgemeinen algebraischen Rahmen, in dem "beste" Wege noch bei veränderten Operationen mit dem Dijkstra Algorithmus berechnet werden können. Vor dem Hintergrund dieses Rahmens lassen sich die in Abschnitt 1.2 aufgelisteten Zielfunktionen (und mögliche andere) dann sehr einfach in Dijkstra-berechenbare und nicht-Dijkstra-berechenbare klassifizieren. Wir geben nachfolgend einige Beispiele.

1.3.2 Schnellste Verbindungen

Zur Ermittlung schnellster Verbindungen benötigt man pro Kante des Ort-Zeit Graphen die Fahrtdauer. Bei (Elementar-) Verbindungen ist dies die aus dem Fahrplan oder durch Preprocessing ermittelte Dauer, bei Umstiegen die Summe aus Umsteigezeit und Wartezeit auf den nächsten Anschluß.

Sind diese Dauern ermittelt, so lassen sich kürzeste Verbindungen/Wege innerhalb bestimmter Zeitintervalle für die Abfahrt (analog für die Ankunft) direkt auf dem Ort-Zeit Graphen mit dem Dijkstra Algorithmus berechnen.

Der gefundene kürzeste Weg ergibt dann die Abreise- und Ankunftzeiten und die entsprechenden Umsteigezeiten aus den Zeiten der einzelnen Knoten.

1.3.3 Kürzeste Verbindungen

Hierfür werden pro (Elementar)-Verbindung die Entfernungskilometer (bzw. Tarifkilometer) benötigt. Umstiege haben die Länge Null. Da sich die Länge eines Weges additiv ergibt, ist wieder der Dijkstra Algorithmus direkt anwendbar.

1.3.4 Umsteigehäufigkeit

Die Umsteigehäufigkeit auf einer Verbindung läßt sich direkt additiv durch die Umsteigehäufigkeit der Kanten entlang des zugehörigen Weges im Ort-Zeit-Graph ermitteln. Daher läßt sich eine Verbindung mit der geringsten Anzahl der Umstiege ebenfalls mit dem Dijkstra-Algorithmus ermitteln.

Voraussetzung ist natürlich wieder, daß die Anzahl der Umstiege für jede (Elementar-) Verbindung bekannt ist. Umstiegskanten erhalten natürlich die Bewertung 1, falls das Verkehrsmittel gewechselt wird.

Oft möchte man jedoch Zeit und Anzahl der Umstiege zueinander in Relation setzen. Hier entsteht eine *mehrkriterielle Zielfunktion*, und der Benutzer ist dann daran interessiert, die *Pareto-optimalen Verbindungen* für diese Kriterien zu berechnen, also solche Kombinationen von Fahrzeit und Anzahl der Umstiege, die von keiner anderen Kombination komponentenweise unterboten werden können. Wir behandeln solche Pareto-optimal Wege gesondert in den Abschnitten 2 und 4.2.

1.3.5 Zeitreserven bei Umstiegen

Dieses Kriterium läßt sich berücksichtigen, wenn pro Kante des Ort-Zeit Graphen die minimal erforderliche Umsteigezeit als *Attribut* mitgeführt wird. Durch Verwendung dieser Attribute können solche Kanten des Graphen "ausgeblendet" werden, die eine zu kurze Umsteigezeit haben. Auf diese Weise wird ein geeigneter Teilgraph erzeugt, in dem dann eine geeignete Verbindung gesucht wird. Hier sind auch im Prinzip verschiedene Zeitreserven bei verschiedenen Umstiegen möglich.

Diese Technik, durch Attribute an den Kanten zu geeigneten Teilgraphen überzugehen, wird auch bei anderen Kriterien noch eine Rolle spielen. Wir sprechen in diesem Zusammenhang von *attributierten Graphen*.

1.3.6 Billigste Verbindungen

Bei diesem Zielkriterium können Schwierigkeiten auftreten, da die Kosten einer Fahrt im allgemeinen nicht additiv aus den Kosten pro Kante zu ermitteln sind.

Gerade im Flugverkehr sind Situationen konstruierbar, wo durch Buchung zusätzlicher (nicht genutzter) Flüge eine fiktive Wochenendsituation herbeigeführt wird, die bei Verlängerung der Strecke eine Senkung des Gesamtpreises bewirkt.

Für solche Fälle kann kein allgemein gültiges Verfahren aufgestellt werden, hier muß (wie jetzt schon) auf die Findigkeit einzelner Reisebüros vertraut werden.

2 Kürzeste Wege bei mehreren Zielkriterien

Wir stellen zunächst das Basismodell für Pareto-optimale Wege in Graphen vor. Für jede Kante e des Graphen sei $\lambda(e) = (\lambda_1(e), \ldots, \lambda_r(e))$ ein r-dimensionaler Bewertungsvektor (z. B. Fahrzeit, Anzahl Umstiege, Kosten, ...). Der Einfachheit halber nehmen wir an, daß sich die Bewertung eines Weges W bezüglich der k-ten Komponente ($k = 1, \ldots, r$) *additiv* ergibt, d. h. $\lambda_k = \sum_{e \in W} \lambda_k(e)$. Im allgemeinen sind hier beliebige Semiringe möglich, vgl. [18].

Ein Weg W von s nach t mit einer Bewertung $\lambda = (\lambda_1, \ldots, \lambda_r)$ heißt *Pareto-optimal* oder *effizient*, falls es keinen anderen Weg W' von s nach t mit einer Bewertung $\lambda' = (\lambda'_1, \ldots, \lambda'_r)$ gibt mit $\lambda'_k \leq \lambda_k$ für $k = 1, \ldots, r$ und $\lambda'_\ell < \lambda_\ell$ für ein $\ell \in \{1, \ldots, k\}$.

Beim mehrkriteriellen s, t-Kürzeste-Wege Problem möchte man alle Pareto-optimalen s, t-Wege berechnen. Dies ist mit einer einfachen Erweiterung des Dijkstra-Algorithmus möglich.

2.1 Der Dijkstra-Algorithmus für Pareto-optimale Wege

Anstelle der beim einfachen Dijkstra-Algorithmus in einem Knoten v abgespeicherten Distanz $d[v]$ betrachtet man jetzt r-dimensionale Vektoren $d[v] = (d_1[v], \ldots, d_r[v])$, wobei $d_k[v]$ die Distanz von s nach v bezüglich der k-ten Kantenbewertung $\lambda_k(e)$ angibt. Zu einem Knoten werden im allgemeinen mehrere solche Vektoren abgespeichert. Sie repräsentieren die Werte der bisher im Algorithmus ermittelten Pareto-optimalen s, v-Wege. Zu Beginn des Algorithmus ist [3]

$$d[v] := \begin{cases} (0, 0, \ldots, 0) & \text{falls } v = s, \\ \lambda(s, v) & \text{falls die Kante } (s, v) \text{ existiert,} \\ (\infty, \infty, \ldots, \infty) & \text{sonst.} \end{cases}$$

Anstelle von Knoten werden nun Bewertungsvektoren markiert. Die Wahl des unmarkierten Knotens v mit kleinstem $d[v]$ wird ersetzt durch die Wahl eines *lexikographisch kleinsten unmarkierten Vektors* $d[v]$, der dann markiert wird. Vom zugehörigen Knoten v aus aktualisiert man für alle Kanten e von v nach w die bei w gespeicherten Pareto-Optima, indem man den Vektor $d[v] + \lambda(e)$ zu den bei w gespeicherten Vektoren hinzunimmt und nicht Pareto-optimale Vektoren streicht. Die Hauptschleife des erweiterten Dijkstra-Algorithmus ist also gegeben durch

while es gibt einen unmarkierten Vektor **do**
  wähle einen lexikographisch kleinsten unmarkierten Vektor;
  markiere diesen Vektor;
  sei v der zugehörige Knoten;
  for alle Kanten (v, w) **do**
    nehme $d[v] + \lambda(v, w)$ zu den Vektoren von w hinzu;
    streiche nicht Pareto-optimale bei w;
  end for
end while

Um die Wege rekonstruieren zu können, muß für jeden Vektor (bisher jeder Knoten) der Vorgängervektor und Vorgängerknoten abgespeichert werden, aus dem der Vektor entstanden ist.

Die Korrektheit dieses erweiterten Dijkstra-Algorithmus kann völlig analog zur einfachen Version bewiesen werden, vgl. etwa [18].

Man sieht in der Hauptschleife, daß für jeden gespeicherten Vektor in einem Knoten v alle ausgehenden Kanten betrachtet werden. Der Aufwand hierfür dominiert die gesamte Laufzeit und berechnet sich zu

$$\sum_{v \in V} \text{pareto}(v) \cdot \text{outdegree}(v).$$

Dabei ist pareto(v) die Anzahl der in v bei Abbruch des Algorithmus gespeicherten Pareto-Optima und outdegree(v) die Anzahl der aus v ausgehenden Kanten.

[3] Bei mehreren parallelen Kanten (s, v) ist die Vereinigung der $\lambda(s, v)$ zu nehmen, und hieraus sind nicht Pareto-optimale Vektoren zu streichen.

Tabelle 1: Alle Pareto-Optima zum Graphen aus Abbildung 5

a	b	c	d	e
(0,0)	(0,1)	(0,3)	(0,7)	(0,15)
	(1,0)	(2,1)	(4,3)	(8,7)
		(1,2)	(1,6)	(1,14)
		(3,0)	(5,2)	(9,6)
			(2,5)	(2,13)
			(6,1)	(10,5)
			(3,4)	(3,12)
			(7,0)	(11,4)
				(4,11)
				(12,3)
				(5,10)
				(13,2)
				(6,9)
				(14,1)
				(7,8)
				(15,0)

Der Aufwand hängt also im wesentlichen von der Anzahl der Pareto-optimalen (s, v)-Wege, $v \in V$, ab. Der Algorithmus ist zwar pro Pareto-Optimum so schnell wie der normale Dijkstra-Algorithmus, aber die Anzahl der Pareto-Optima kann exponentiell groß in Anzahl der Knoten werden, so daß im allgemeinen eine exponentielle Laufzeit in Kauf genommen werden muß, um *alle* Pareto-Optima zu bestimmen. Tatsächlich ist die Bestimmung aller Pareto-optimaler s, t-Wege im allgemeinen schwach $\mathcal{NP}$-schwer, da sich das als schwach $\mathcal{NP}$-schwer bekannte KNAPSACK Problem hierauf reduzieren läßt [19].

Als Beispiel für eine exponentielle Anzahl von Pareto-optimalen Wegen betrachte man den Graphen aus Abbildung 5.

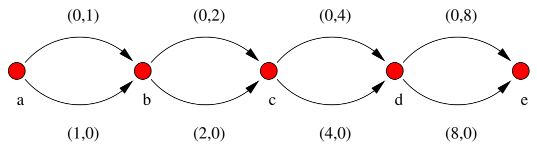

Abbildung 5: Ein Beispiel für die Berechnung Pareto-optimaler Wege

Der auf Seite 8 dargestellte Algorithmus erzeugt bei Abbruch die in Tabelle 1 angegebenen Pareto-Optima (in der angegebenen Reihenfolge pro Spalte) in den Knoten a, b, c, d, e. Bei n Knoten werden also im letzten Knoten 2^{n-1} Pareto-optimale Lösungen erzeugt.

Eine einfache obere Schranke für die maximale Anzahl Pareto-optimaler Lösungen in einem Knoten v läßt sich aus der maximalen Weglänge $L_k(v)$ von s nach v für das k-te Kriterium herleiten, falls die Werte $\lambda_k(e)$ ganzzahlig und nichtnegativ sind. Dann gilt

$$\text{pareto}(v) \leq (L_k(v))^r.$$

Daher bleibt die Anzahl pareto(v) *polynomial* in der Knotenzahl n, wenn die Kantenbewertungen $\lambda_k(e)$ polynomial in n sind und r fest ist. Gilt z. B. $\lambda_k(e) \leq n$, so ist $L_k(v) \leq (n-1)n \leq n^2$ und daher pareto$(v) \leq n^{2r}$. Bei der Verbindungssuche werden in der Regel sogar wesentlich kleinere Kantenbewertungen auftreten, die unabhängig von n durch eine Konstante L beschränkt sind. In diesem Fall ergibt sich

$$\text{pareto}(v) \leq (L \cdot (n-1))^r.$$

Darüberhinaus kann in vielen Fällen auf Wege mit zu hohen Bewertungen bei Nebenkriterien verzichtet werden, wie z. B. bei der Anzahl der Umstiege (sicher nicht mehr als 3 oder 4) oder der maximalen Wartezeit zwischen Verbindungen. Dann kann das zugehörige $L_k(v)$ für alle Knoten v klein gehalten werden, und die Anzahl der zu betrachtenden Pareto-optimalen Lösungen bleibt klein.

2.2 Approximation Pareto-optimaler Wege

Verzichtet man auf die exakte Ermittlung aller Pareto-optimalen Wege, so gibt es verschiedene Methoden um "Näherungslösungen" zu berechnen. Generell betrachtet man anstelle des Ausgangsproblems P ein modifiziertes Problem P', löst dieses exakt mit geringem Aufwand und konstruiert aus der Lösung von P' eine approximative Lösung von P. In der Literatur lassen sich hierzu drei Ansätze unterscheiden.

2.2.1 Gewichtete Summe der Kriterien

Dieser Ansatz wird in [8] betrachtet. Dort wird ein Zulässigkeitsproblem behandelt, d. h. man möchte zu gegebenem Vektor $(a_1, \ldots, a_r)$ einen Weg W finden, dessen Länge $\lambda(W)$ in jedem Kriterium $\lambda_k(W)$ den Wert a_k nicht überschreitet. Man transformiert dieses Problem auf die Berechnung eines kürzesten Weges bezüglich der einkriteriellen Bewertung

$$\lambda'(e) := \alpha_1 \cdot \lambda_1(e) + \ldots + \alpha_r \cdot \lambda_r(e),$$

wobei $\alpha_1, \ldots, \alpha_r$ geeignete, von den a_k abhängige Gewichtungsfaktoren sind. Der resultierende kürzeste Weg W' wird als Näherungslösung für den gesuchten Weg W genommen.

Im allgemeinen werden hierdurch nur schlechte Approximationen erzielt. Bevorzugt man etwa im Beispiel aus Abbildung 5 für Schranken $a_1 < a_2$ das erste Kriterium ($\alpha_1 > \alpha_2$), so wird der Weg W' mit Pareto-Optimum $(0, 15)$ berechnet, was bezüglich des zweiten Kriteriums um fast den Faktor 2 von der geforderten Länge a_2 abweichen kann. Allgemein ergibt sich [18, S. 135]:

Ist W ein Pareto-optimaler Weg mit $\lambda(W) \leq (a_1, \ldots, a_r)$, und gilt $\lambda(W') = (b_1, \ldots, b_r)$, so gilt für $\alpha_1 = \alpha_2 = \ldots = \alpha_r = 1$ die Schranke

$$\max\{b_1, \ldots, b_r\} \leq r \cdot \max\{a_1, \ldots, a_r\}.$$

Diese ist im allgemeinen scharf.

Bei vielen Kriterien können sich die Lösungen also um den Faktor r unterscheiden. Dies ist sicherlich für praktische Anwendungen zu groß.

2.2.2 Einfache Skalierung der Längen

Dieser Ansatz wird in [18, S. 171 ff] betrachtet. Hierbei *skaliert* man die (als ganzzahlig vorausgesetzten) Kantengewichte $\lambda(e) = (\lambda_1(e), \ldots, \lambda_r(e))$ auf den weniger wichtigen Nebenkriterien $k = 2, \ldots, r$ um den Faktor $\alpha > 1$ und erhält die neuen ganzzahligen Kantengewichte

$$\lambda'(e) := (\lambda_1(e), \lfloor \frac{\lambda_2(e)}{\alpha} \rfloor, \ldots, \lfloor \frac{\lambda_r(e)}{\alpha} \rfloor).$$

Da diese um den Faktor α kleiner sind, ist die Anzahl pareto(v) der Pareto-optimalen Lösungen pro Knoten v kleiner, da die Weglängen $L_k(v)$ kürzer sind und pareto$(v) \leq (L_k(v))^r$ gilt (vgl. Seite 9). Falls $L_k(v)$ polynomial in n ist, so läßt sich dadurch eine polynomiale Laufzeit zur Ermittlung aller Pareto-Optima für die Kantenlängen $\lambda'(e)$ erreichen.

Zu den bezüglich λ' berechneten Pareto-optimalen Wegen ermittelt man dann ihre Bewertung bezüglich der ursprünglichen Kantenbewertung λ. Diese sind wegen der ordnungserhaltenden Skalierung auch Pareto-Optima bezüglich λ.

Tabelle 2: Pareto-Optima zum Graphen aus Abbildung 6 bzgl. λ und λ'

Pareto-Optima bzgl. λ'					zugehörige Wegbewertungen bzgl. λ				
a	b	c	d	e	a	b	c	d	e
(0,0)	(0,0)	(0,0)	(0,1) (4,0)	(0,3) (8,1) (4,2) (12,0)	(0,0)	(0,1)	(0,3)	(0,7) (4,3)	(0,15) (8,7) (4,11) (12,3)

Wir erläutern das Verfahren am Beispiel aus Abbildung 5. Setzt man $\alpha = 4$, so ergeben sich die in Abbildung 6 dargestellten Kantenlängen mit den in Tabelle 2 angegebenen Pareto-Optima bezüglich λ' und den zu den Längen der zugehörigen Wege bezüglich λ.

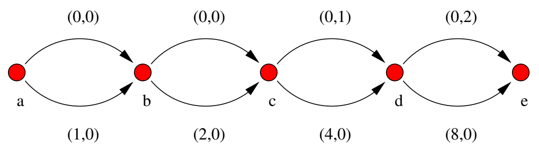

Abbildung 6: Einfache Skalierung des Beispiels aus Abbildung 5 mit $\alpha = 4$

Sei X die Menge der Pareto Optima im Knoten v bezüglich λ und sei $Y \subseteq X$ die durch die Approximation berechnete Teilmenge dieser Pareto-Optima. Diese Menge Y erfüllt folgende *Approximationseigenschaft*:

> Zu einem Pareto-optimalen Weg W mit Bewertungsvektor $x \in X$ gibt es einen Pareto-optimalen Weg W' mit Bewertungsvektor $y \in Y$ mit
>
> $$|x_k - y_k| \leq \ell(\alpha - 1) \text{ für } k = 1, 2, \ldots, r,$$
>
> wobei ℓ die Anzahl der Kanten von W' ist.

Im Beispiel gibt es zu $x^1 = (9,6)$ den Vektor $y^1 = (8,7)$, zu $x^2 = (13,2)$ den Vektor $y^2 = (12,3)$ usw. Die maximale Abweichung pro Komponente beträgt 3, was wesentlich kleiner als die allgemein ableitbare (pessimistische) Schranke von $\ell(\alpha - 1) = 4 \cdot 3 = 12$ ist.

Eine genaue Fehleranalyse wird in [18, S. 172 ff und S. 203 ff] durchgeführt. Sie zeigt, daß die angegebene Schranke von $\ell(\alpha - 1)$ im allgemeinen zwar scharf ist, in praktischen Rechnungen jedoch wesentlich geringere Abweichungen auftreten.

2.2.3 Ein voll polynomiales Approximationsschema

Warburton [19] entwickelt ein polynomiales Approximationsschema zur Berechnung Pareto-optimaler Wege. Es bestimmt zu vorgegebener "Genauigkeitsschranke" ε eine Menge von Wegen, die jeden Pareto-optimalen Weg bis auf den Faktor $1 + \varepsilon$ genau approximiert, wobei der Aufwand polynomial in der Größe des Graphen und $\frac{1}{\varepsilon}$ ist.

Die berechnete Menge $Y(\varepsilon)$ der Längenvektoren der ermittelten Wege hat folgende Eigenschaften:

(1) Jeder Pareto-optimale Vektor x wird von einem Vektor $y \in Y(\varepsilon)$ *ε-dominiert*, d. h. $y_k \leq (1 + \varepsilon)x_k$ für $k = 1, \ldots, r$.

(2) Jeder Vektor $y \in Y(\varepsilon)$ ist Pareto-optimal bezüglich $Y(\varepsilon)$.

Die Grundidee des Algorithmus von Warburton ist mehrfache Skalierung und, für jede festgehaltene Skalierung, die Lösung eines Zulässigkeitsproblems im Sinne von Abschnitt 2.2.1.

Die Anzahl der Skalierungen ergibt sich zu $m = \lceil \log B \rceil$ mit $B := \max_{k=1,\ldots,r} L_k(t)$. Dabei muß für jeden Vektor $(a_2, \ldots, a_r)$ mit $a_i \in \{2^0, 2^1, \ldots, 2^m\}$ ein Zulässigkeitsproblem mit freier Komponente $k = 1$ gelöst werden. Bereits für $B = 32$ und $r = 5$ ergeben sich $5^4 = 625$ Berechnungen eines Zulässigkeitsproblems, von der jede genauso so zeitaufwendig ist wie eine einzelne Dijkstra-Berechnung.

Obwohl theoretisch von großer Bedeutung, muß daher der Ansatz von Warburton für die Praxis eher als ungeeignet eingeschätzt werden.

2.2.4 Lexikographische Ordnung

Einfacher werden mehrkriterielle Probleme, wenn eine lineare (etwa die lexikographische) Ordnung auf den Kriterien gegeben ist. Beispielsweise bevorzugt man eine möglichst kleine Anzahl von Umstiegen und nimmt als zweites Kriterium die Reisezeit, die zwischen Verbindungen mit gleicher Anzahl von Umstiegen differenziert. In diesem Fall läßt sich der Dijkstra-Algorithmus direkt übertragen, wobei die Addition durch die komponentenweise Addition der beiden Kriterien ersetzt wird und die Minimumsbildung durch das lexikographische Minimum. In diesem Fall ist die Berechnung also genauso effizient wie der einfache Dijkstra-Algorithmus.

Da diese Berechnung sehr schnell geht, können mehrere Anordnungen der Kriterien mit der entsprechenden lexikographischen Ordnung durchgerechnet werden. Man erhält in jedem Fall ein Pareto-Optimum, aber natürlich in der Regel nicht alle.

Im Beispiel aus Abbildung 5 gibt es nur 2 mögliche Anordnungen der Kriterien, die zu den Pareto-Optima $(0, 15)$ und $(15, 0)$ führen. Wie man sieht, werden sehr viele Pareto-Optima nicht erfaßt, und es ist auch keine Gütegarantie beweisbar.

2.3 Vergleich der Methoden

In theoretischer Hinsicht ist das Verfahren von Warburton (Abschnitt 2.2.3) am befriedigsten. Es bietet die Möglichkeit, die Pareto-Optima durch eine kleine Menge von Wegen mit beliebiger Genauigkeit ε zu approximieren. Anzahl der Wege und Laufzeit hängen dabei polynomial von der Größe des Graphen und dem Inversen $\frac{1}{\varepsilon}$ der Genauigkeit ab. Leider muß dieses Verfahren trotz der beweisbaren Güte und polynomialen Laufzeit wohl als für die Praxis zu aufwendig eingestuft werden.

Einen Kompromiß zwischen Laufzeit und Güte stellen die Verfahren der *gewichteten Summe der Kriterien* (Abschnitt 2.2.1) und der *einfachen Skalierung* (Abschnitt 2.2.2) dar. Auch die Beschränkung auf das Zulässigkeitsproblem bei Haupt- und Nebenkriterien (Beschränkung der Reisezeit, der Anzahl der Umstiege, des Preises) scheint möglich und sinnvoll, so daß die in den Abschnitten 2.2.1 und 2.2.2 gegebenen Gütegarantien gültig sind. Beide Verfahren sind schnell und können auch öfter mit verschiedenen Wahlen der Gewichte durchgerechnet werden.

Das Verfahren der lexikographischen Ordnung kann als zusätzliche Heuristik herangezogen werden.

3 Beschleunigung von Kürzeste-Wege Berechnungen

Die bisherigen Überlegungen reduzierten Verbindungsabfragen auf Kürzeste-Wege Algorithmen in geeignet definierten Graphen, wobei die Berechnung selbst durch den Dijkstra Algorithmus (bzw. seine mehrkriteriellen Varianten) erfolgte.

Der Dijkstra Algorithmus ist bekanntlich ein *single-source multiple-target* Verfahren, d. h. er berechnet kürzeste Wege vom Startknoten s zu prinzipiell allen Knoten im Graphen (wobei natürlich abgebrochen werden kann, sobald der Zielknoten t markiert wird (vgl. Seite 5).

Hier ergeben sich durch die Betrachtung *zielgerichteter* oder *bidirektionaler* Suchverfahren Möglichkeiten, den Berechnungsaufwand zu reduzieren. Eine weitere Reduktionsmöglichkeit besteht darin, im Ort-Zeit-Graphen das Zeitfenster für die vom Knoten *Start* ausgehenden Kanten (vgl. Abschnitt 1.1.3) klein zu halten. Die nächsten Unterabschnitte behandeln diese Reduktionsmöglichkeiten im Detail.

3.1 Zielgerichtete Suche

Die zielgerichtete Suche basiert auf der Betrachtung von *Schranken* oder *Schätzungen* $b(v,t)$ (b wie *b*ound) für die Distanz $d(v,t)$ eines beliebigen Knotens v zum Zielknoten t und geht auf [6] zurück. Die grundsätzliche Idee besteht darin, beim Dijkstra Algorithmus statt der Distanzwerte $d[v]$ eines Knotens v von Startknoten s mit den Werten

$$d'[v] := d[v] + b(v,t)$$

zu rechnen und die *Auswahl des als nächsten zu markierenden Knotens* v anhand des minimalen $d'[v]$ Wertes vorzunehmen. Hierbei bezeichnet $d[v]$ die momentane Distanz im Dijkstra Algorithmus (vgl. Abschnitt 1.3.1). Da Knoten v in Richtung Ziel dann tendenziell kleinere $d'[v]$ Werte haben als solche, die nicht in Zielrichtung liegen, wird der Zielknoten schneller markiert und somit schneller ein Weg von s nach t gefunden (der jedoch i. a. nicht optimal sein muß).

Diese Suche wird oft als *Best-First Search* bezeichnet. Auf Datenstrukturebene entspricht sie einem Übergang von einer Breitensuche (breadth-first search) zu einer Mischung aus Breiten- und Tiefensuche (depth-first search), bei der man tendenziell eher Knoten in Richtung Ziel t markiert. Auf Programmebene bleibt es natürlich eine Breitensuche bzgl. der neuen $d'[v]$ Werte.

Durch zusätzliche Forderungen an die Schranke $b(v,t)$ lassen sich Aussagen an die Qualität (Optimalität) und Laufzeit der zielgerichteten Suche machen.

3.1.1 Konsistente untere Schranken

Eine scharfe Bedingung, die sog. *Konsistenzbedingung* wird in [6] eingeführt. Die Schranken $b(v,t)$ müssen *untere Schranken* für die Distanz sein, also

$$b(v,t) \leq d(v,t)$$

und zusätzlich die *Konsistenzbedingung*

$$\lambda(v,w) + b(w,t) \geq b(v,t)$$

erfüllen. Diese Bedingung besagt, daß die untere Schranke bei Annäherung an das Ziel t über die Kante (v,w) eher besser wird, was für viele Bewertungsfunktionen (z. B. euklidischer Abstand in der Ebene) zutrifft. Auch für Fahrtzeitabschätzungen könnte diese Konsistenzbedingung erfüllbar sein.

Der Vorteil der Konsistenzbedingung liegt darin, daß man nun modifizierte Kantenbewertungen

$$\lambda'(v,w) := \lambda(v,w) - b(v,t) + b(w,t)$$

einführen kann. Rechnet man mit dem gewöhnlichen Dijkstra Algorithmus bezüglich dieser Bewertungen $\lambda'(v,w)$, so ändern sich Weglängen von Wegen zwischen s und t nur um den konstanten additiven Term $b(s,t) - b(t,t)$. Kürzeste Wege bleiben also erhalten.

Im Lauf dieses Dijkstra Algorithmus sind jedoch die Label $d'[v]$ der Knoten in Richtung Zielknoten t zum Zeitpunkt der Markierung deutlich kleiner, da sie sich als

$$d'[v] = d[v] + b(v,t) - b(s,t)$$

ergeben, wobei $d[v]$ die kürzeste Distanz von s nach v im Graphen ist.

Je "näher" v bei t liegt, desto kleiner wird $d'[v]$. Dies bewirkt, daß der Dijkstra Algorithmus sich zielgerichtet nach t bewegt. Das Ziel t wird nie später erreicht als bei den ursprünglichen Bewertungen $\lambda(v,w)$ (vgl. [10]), in der Regel jedoch früher, so daß man im Mittel lineare Laufzeit für viele Verteilungen der Kantenlängen erreicht [17].

3.1.2 Untere Schranken

Verzichtet man auf die Konsistenzbedingung, verlangt aber weiterhin, daß die untere Schrankeneigenschaft $b(v,t) \leq d(v,t)$ gewährleistet ist, so gibt es keine Längentransformation, die einerseits eine zielgerichtete Suche ermöglicht, und andererseits kürzeste Wege erhält.

Jedoch bleibt folgende wichtige Eigenschaft gültig (*Optimalitätseigenschaft*):

> Wird der Zielknoten t markiert, so ist $d[t]$ gleich der kürzesten Distanz von s nach t bzgl. der gegebenen Kantenbewertungen $\lambda(v,w)$, und der gefundene Weg ist ein kürzester Weg von s nach t.

Dies ergibt sich aus folgender Überlegung. Wenn t markiert, so wird als $d[t]$ die Länge des Weges (über Rückverfolgung der *Vorgänger*$[v]$ Zeiger, vgl. Abschnitt 1.3.1) berechnet, der zu t führt. Ist $d[t]$ größer als die kürzeste Weglänge ℓ von s nach t, so erfüllt jeder nicht markierte Knoten v auf einem kürzesten Weg die Ungleichung

$$d[v] + b(v,t) \leq d < \ell$$

und wäre daher vor t zur Markierung an der Reihe, im Widerspruch zur Wahl von t. Also gilt die Optimalitätsbedingung.

Untere Schranken sind in Anwendungen meist leicht zu generieren. Darauf basierend ist also eine zielgerichtete Suche möglich, die bereits bei der (ersten) Markierung des Zielknotens t zu einem kürzesten Weg führt.

3.1.3 Näherungswerte

Sind die $b(v,t)$ nur *Schätzwerte* für die Distanz $d(v,t)$ von v nach t, die *nicht* die untere Schrankeneigenschaft erfüllen, so kann man die Optimalitätseigenschaft nicht mehr garantieren. Bei der Markierung des Zielknoten t wird dann zwar ein Weg gefunden, aber dieser ist i. a. kein kürzester Weg.

In dieser Situation sind mehrere Ansätze möglich. Die erste besteht darin, weiter zu rechnen und *mehrfache Markierung* eines Knotens v zuzulassen. Jede zusätzliche Markierung führt dann zu einem kürzeren Weg von s nach v. Dies ist genau die Vorgehensweise des A^* Algorithmus aus der KI (vgl. [13]), der i. a. eine exponentielle Laufzeit erfordert, um einen kürzesten Weg zu ermitteln.

Durch zusätzliche Forderungen an die Schätzung $b(v,t)$ lassen sich bessere Aussagen über Laufzeiten und zum Teil auch Fehlerschranken angeben, vgl. etwa [3, 7, 18].

Wir schlagen hier statt des schwer kontrollierbaren A^* Algorithmus einen anderen Ansatz vor, der auf den Methoden für *Pareto-optimale Wege* beruht (vgl. Abschnitt 2).

Die aus der Schätzung $b(v,t)$ resultierenden Werte

$$d'[v] := d[v] + b(v,t)$$

werden bei den Knotenbewertungen als zweites Kriterium mitgeführt. Jeder Knoten v hat dann das mehrwertige Label $(d[v],\ d'[v])$.

Verwendet man nun die lexikographische Ordnung (vgl. Abschnitt 2.2.4), so wird bei der Auswahl des als nächsten zu markierenden Knoten wird der mit *lexikographisch kleinstem Label* ausgewählt. Dies garantiert, daß einerseits die *Optimalitätsbedingung* gilt (bei Markierung ist ein kürzester Weg gefunden), andererseits das zweite Kriterium für eine (gewisse) Zielgerichtetheit sorgt.

Dieser Effekt kann (unter Verzicht auf die Optimalitätsbedingung) ganz wesentlich verstärkt werden, wenn Pareto-optimale Lösungen bzgl. einer *gewichteten Summe der Kriterien* (vgl. Abschnitt 2.2.1) ermittelt werden. Die Wahl der Gewichte ermöglicht dann die Steuerung eines *Tradeoff zwischen Optimalitätsbedingung und Zielgerichtetheit*.

3.1.4 Schätzungen in Verkehrsnetzen

Die Einsetzbarkeit der zielgerichteten Suche bei der Verbindungsabfrage hängt von den Möglichkeiten ab, gute untere Schranken bzw. Schätzungen $b(u,v)$ für die "Distanz" zwischen zwei Knoten

u und v anzugeben, wobei unter "Distanz" das jeweilige Zielkriterium (Fahrzeit, Länge usw.) zu verstehen ist.

Unproblematisch ist eine solche Schätzung bzgl. der Entfernung. Hier kann der geographische Abstand (Luftlinie) genommen werden, der sich etwa aus abgespeicherten geographischen Koordinaten einfach berechnen läßt. Hieraus lassen sich auch angenähert Fahrzeit und Tarif ableiten, jedoch kaum Kriterien wie Umsteigehäufigkeit, Zeitreserven bei Umstiegen usw.

Erwägenswert ist hier ein *Preprocessing*, das solche Schätzungen unabhängig von der tatsächlichen Fahrzeit ermittelt und das durch Table look-up sehr schnell zur zielgerichteten Suche genutzt werden kann.

Dieses Preprocessing kann auch in *reduzierter Form* erfolgen, bei dem nur "Distanzen" $d(G_i, G_j)$ zwischen Gebieten G_i, G_j abgespeichert werden (z. B. zwischen den Hauptverkehrsknoten zwischen den Gebieten), und bei jedem Knoten v sein Gebiet G_v und die "Distanz" $d_G(v)$ zum Hauptverkehrsknoten in seinem Gebiet abgelegt wird. Auf diese Weise hält man den quadratischen Teil der "Distanzen" $d(G_i, G_j)$ zwischen je zwei Gebieten klein und kann doch mit 3 Look-ups eine gute Schätzung $b(u, v)$ durch

$$b(u, v) = d_G(u) + d(G_u, G_v) + d_G(v)$$

bekommen.

3.2 Bidirektionale Suche

Bei der bidirektionalen Suche startet man den Dijkstra Algorithmus gleichzeitig im Startknoten s und im Zielknoten t (von t aus bei umgedrehten Kantenorientierungen). Dabei werden für jeden Knoten v Entfernungen $d_s[v]$ und $d_t[v]$ von s bzw. t nach v berechnet. Wird dabei zum ersten mal ein Knoten v_0 in beiden Algorithmen markiert, so kann die Wegesuche abgebrochen werden. Unter den bisher markierten Knoten befindet sich dann ein kürzester Weg der Länge $d_s[u] + \lambda(u, v) + d_t[v]$, der von s zu einem von s aus markierten Knoten u verläuft, dann eine Kante (u, v) zu einem von t aus markierten Knoten v benutzt, und von dort nach t läuft, vgl. [1, p. 112].

Dieser Weg kann wie folgt durch eine geschickte Minimumssuche ermittelt werden. Betrachte den Schritt vor der Markierung des ersten gemeinsamen Knotens v_0 und ermittle $\min_u \{d_s[u] + d_t[u]\}$ über alle Knoten u mit $d_s[u] + d_t[u] < \infty$. Die sind gerade die Knoten u, die sich beim Abbruch der Suche in den Datenstrukturen befinden, mit denen die als nächste von s bzw. t aus zu markierenden Knoten gewählt werden (z. B. Heaps). Die Verwaltung dieses Minimums kann daher ebenfalls während der bidirektionalen Suche in einem Heap geschehen, so daß die Laufzeit gegenüber dem gewöhnlichen Dijkstra Algorithmus kaum wächst.

Durch diese Vorgehensweise werden die Mengen der markierten Knoten i. a. deutlich kleiner gehalten. Geht man (Beispiel Nahverkehr) von einem dicht vermaschten Netz aus, so untersucht der Dijkstra Algorithmus ein etwa kreisförmiges Gebiet bis der Zielknoten erreicht ist, wobei die Entfernung $d = d(s, t)$ dem Radius des Kreises entspricht. Bei der bidirektionalen Suche entstehen idealerweise zwei Kreise mit halb so großem Radius, so daß die Anzahl der betrachteten Knoten, die in etwa der Fläche der Kreise entspricht, gemäß dem Verhältnis

$$\frac{2 \cdot \pi (\frac{d}{2})^2}{\pi d^2} = \frac{1}{2}$$

halbiert wird.

Die bidirektionale Suche läßt sich natürlich mit der zielgerichteten Suche verbinden, wodurch weitere Beschleunigungseffekte erzielt werden.

Theoretisch sind zwar kaum beweisbare Vorteile nachweisbar [10], aber empirische Untersuchungen zeigen eine Überlegenheit der bidirektionalen Suche über die unidirektionale Suche [14].

3.3 Zeitfenster im Ort-Zeit-Graphen

Eine andere Art, die Berechnung kürzester Wege zu beschleunigen, besteht in der Einschränkung der Anzahl der Knoten im Ort-Zeit-Graphen, die mit dem Startknoten verbunden sind (vgl. Abschnitt 1.1.3). Da jeder dieser Knoten einem Zeitpunkt im Startort entspricht, geht es also um die *Auswahl eines Zeitfensters* für die zu untersuchenden Abfahrtzeiten.

Die Grundproblematik bei der Festlegung des Zeitfensters besteht darin, daß eine spätere Verbindung früher am Ziel sein kann (vgl. Abschnitt 1.1.2), man diese Verbindung aber durch ein zu klein gewähltes Zeitfenster nicht ermittelt.

Eine Auffangmöglichkeit hierfür bietet die *Rückwärtsrechnung vom Zielknoten*: man wählt ein zweites Zeitfenster, das mit der zunächst ermittelten Ankunftszeit endet, und berechnet kürzeste Verbindungen zum Startknoten in der Hoffnung, die "schnelle" Verbindung nun dabei zu haben. Dies gelingt natürlich genau dann nicht, wenn diese schnelle Verbindung außerhalb beider Zeitfenster liegt, also *später* als im betrachteten Zeitfenster am Startknoten beginnt und *früher* als im betrachteten Zeitfenster am Zielknoten ankommt.

Solche Fälle sind sicher selten, aber nicht auszuschließen, gerade bei Verknüpfung vieler Teilsysteme.

Hier bieten sich nun verschiedene Möglichkeiten für die Einschränkung der Zeitfenster an.

In *getakteten Systemen* reicht es natürlich, nur das Zeitfenster am Start zu betrachten und als Länge die Taktzeit zu nehmen.

In ungetakteten oder "fast" getakteten (Taktung mit Ausnahmen) lassen sich *untere Schranken* (vgl. Abschnitt 3.1) zur Abschätzung heranziehen. Man ermittelt in einer Vorwärtsrechnung bei Abfahrzeit t_{start} eine Ankunftszeit t_{ziel}. Ist b eine untere Schranke für die kürzeste Fahrtdauer, so braucht man für die Rückwärtsrechnung nur das Zeitintervall von $t_{start} + b$ bis t_{ziel}, also $[t_{start} + b,\ t_{ziel}]$ betrachten. Entsprechend reicht am Start das Intervall $[t_{start}, t_{ziel} - b]$, falls die Abfahrtzeit auf jeden Fall nach t_{start} erfolgen soll.

Sind nur Schätzungen für die Fahrtdauer angebbar, so lassen sich die Zeitfenster ebenfalls nur schätzen, und generelle Garantien für das Finden einer kürzesten Verbindung können nicht gegeben werden.

Die hier gemachten Aussagen gelten prinzipiell auch für andere Zielkriterien, wobei es wohl im Einzelfall (Anzahl der Umstiege, billigste Verbindung) schwer sein kann, Schranken anzugeben. Hier bleibt jedoch stets der Ausweg, im Zeitfenster des wichtigsten Kriteriums Reisezeit, für das gute Schranken möglich sind, auch die anderen Kriterien zu betrachten und nach einem der mehrkriteriellen Verfahren aus Abschnitt 2 zu bewerten.

4 Verteilte Verbindungssuche

Bei der verteilten Verbindungssuche dürfte der Regelfall darin bestehen, daß verschiedene Teilnetze nur an wenigen, besonders ausgezeichneten Übergangspunkten miteinander verbunden sind. Dieser Fall läßt sich graphentheoretisch einfach modellieren, was zunächst am Beispiel von zwei Teilnetzen geschehen soll.

Dazu betrachten wir den Liniengraphen, der aus den Linien beider Teilnetze besteht. Die Orte, bei denen ein Übergang zwischen beiden Teilnetzen möglich ist, bilden dann im Liniengraphen einen sogenannten *Separator*, d. h. eine Menge von Knoten, nach deren Wegnahme der Graph nicht mehr zusammenhängend ist. Die Anzahl der Übergangspunkte wird im allgemeinen klein im Verhältnis zur Anzahl der Knoten des Graphen insgesamt sein. In diesem Fall sprechen wir von *schwach verbundenen Systemen oder Netzen*. Durch die Separatoreigenschaft ist insbesondere gesichert, daß keine Verbindung zwischen den verschiedenen Teilnetzen existiert, die keinen der Übergangspunkte benutzt. Jede Verbindung muß einen Knoten des Separators benutzen. Abbildung 7 illustriert diese Situation.

Ein Separator in einem gerichteten Graphen wird dann über den zugrunde liegenden ungerichteten Graphen definiert.

Ist $G = (V, E)$ ein (gerichteter oder ungerichteter) Graph und $S \subseteq V$ ein Separator von G, so entstehen durch Wegnahme der Knoten aus S $k \geq 2$ zusammenhängende Teilgraphen mit paarweise disjunkten Knotenmengen $V_1, V_2, \ldots, V_k$, wobei $V = V_1 \cup \ldots \cup V_k \cup S$ ist. Jede Knotenmenge $V_i \cup S$ repräsentiert dann alle Knoten eines Teilnetzes mit den Verbindungsknoten zu den anderen

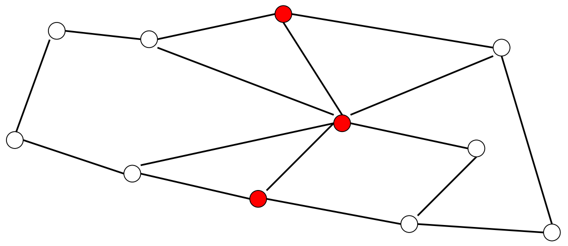

Abbildung 7: Ein Separator (dunkle Knoten) in einem ungerichteten Graphen

Teilnetzen. Das zu $V_i \cup S$ *gehörige Teilnetz* besteht dann aus den Knoten aus $V_i \cup S$ und allen Kanten, die zu Linien in diesem Teilnetz gehören. Dies können auch Kanten zwischen Knoten aus S sein.

Abbildung 8 gibt die *Teilnetze* zum Separator aus Abbildung 7 an. Man beachte, daß die Kanten, die zwischen Separatorknoten verlaufen, dem richtigen Teilnetz zugeordnet werden müssen.

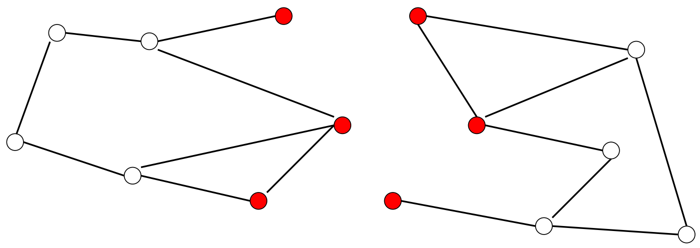

Abbildung 8: Die Teilnetze zum Separator aus Abbildung 7

Neben dem Fall schwach verbundener Netze gibt es die schwieriger zu behandelnde Situation, in der verschiedene Teilsysteme auf der gleichen Menge von Knoten operieren (etwa gemeinsames S-Bahn- und U-Bahnnetz in einer Stadt). In diesem Fall sprechen wir von *stark verbundenen Systemen oder Netzen*. Wir werden nun beide Fälle nacheinander analysieren.

4.1 Schwach verbundene Systeme

Wir behandeln hier zunächst den Fall von drei schwach miteinander verbundenen Netzen. Er repräsentiert die häufige Standardabfrage, von einem Ort aus dem Netz 1 (Nahverkehr in einer Stadt) zu einem größeren Verkehrsknoten zu fahren (Flughafen, Hauptbahnhof), von dort einen Weg in einem überregionalen Netz (Bahn, Fluglinien) zu einem Verkehrsknoten in der Nähe des Zielortes zu finden und dann im lokalen System des Zielortes zum eigentlichen Ziel zu fahren, vgl. Abbildung 9.

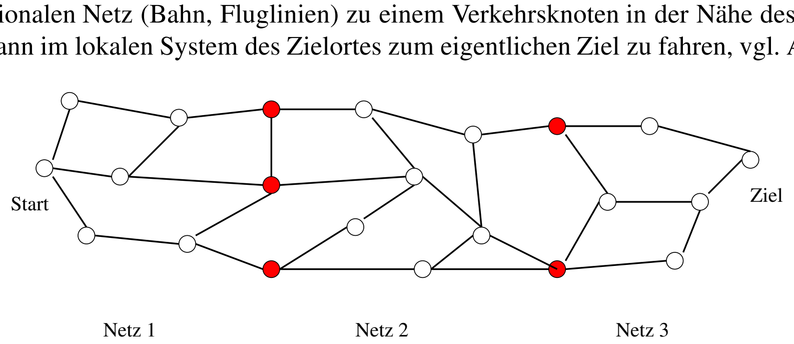

Abbildung 9: Netz zur verteilten Standardabfrage

Eine solche Verbindungsabfrage soll von einem übergeordneten Algorithmus (im weiteren *Ma-*

ster genannt) durchgeführt werden, wobei dieser geeignet mit Algorithmen (z. B. über API-Schnittstellen) kommunizieren muß, die Verbindungsabfragen auf den Teilsystemen durchführen können.

4.1.1 Identifikation der Teilsysteme

Bei der Anfrage sind dem Master zunächst nur Start und Ziel bekannt. Der Master muß dann die Teilnetze lokalisieren, in denen Start und Ziel liegen, und das dazwischen liegende Netz (bzw. mehrere dazwischen liegende Netze). Auf die Ermittlung dieser Netze soll in dieser Studie nicht weiter eingegangen werden, da hierzu andere Methoden als Kürzeste-Wege Berechnungen erforderlich sind.

Sind dem Master nun Startnetz, Zielnetz und Zwischennetz(e) bekannt, so braucht er als weitere Information die Separatoren zwischen den einzelnen Netzen. Auch diese müssen ihm durch andere Methoden bereitgestellt werden.

Um nun eine Kürzeste-Verbindungs-Suche durchzuführen, kann der Master Anfragen nach kürzesten Verbindungen an die einzelnen Teilnetze stellen. Jedoch kennt er von Ihnen zunächst nur Zielknoten und Übergangsknoten. Hier sind nun, abhängig von der Information, die der Master von den Teilsystemen bekommt, verschiedene Berechnungsmethoden denkbar. Wir betrachten zunächst den Fall, daß bei Anfragen an ein Teilsystem nur Information über (Elementar-) Verbindungen im Sinne von Abschnitt 1.1.2 geliefert wird.

4.1.2 Ein Basismodell für den Master bei minimaler Information aus den Teilsystemen

Der Master kennt nur Start, Ziel, die zugehörigen Teilsysteme und die Zwischensysteme mit den Verbindungsknoten zwischen ihnen. Diese Systeme liefern ihm Antworten auf einzelne Verbindungsabfragen, jedoch bekommt der Master keine Information über Fahrpläne, Liniengraphen, Ort-Zeit Graphen oder sonstige interne Informationen der Teilsysteme.

Der Master muß daher in gewisser Weise eine Kürzeste-Wege Suche auf einem Graphen durchführen, der nur aus diesen Knoten und den Umstiegen auf zeitlich in Frage kommende Verbindungen besteht. Für die drei Netze aus Abbildung 9 ist dieser Graph in Abbildung 10 wiedergegeben.

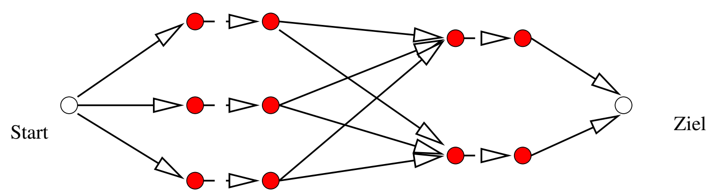

Abbildung 10: Der Graph für die Suche des Master Algorithmus für die Netze aus Abbildung 9 bei jeweils einem Umstieg zwischen den Netzen (gestrichelte Kanten)

Vom Startknoten aus gibt es zunächst Übergangskanten auf zeitlich in Frage kommende Abfahrten (liefert das Teilsystem für Netz 1). Von diesen aus gibt es Kanten zu jedem der in Frage kommenden Übergangsknoten zwischen Netz 1 und Netz 2. Von diesen Übergangsknoten aus sind wieder die entsprechenden zeitlichen Übergänge als gestrichelte Kanten dargestellt (natürlich zu Abfahrten aus denselben Übergangsknoten), von denen aus dann Kanten durch Netz 2 zu den Übergangsknoten in Netz 3 verlaufen, die dort entsprechend wieder mit zeitlichen Übergangskanten verbunden sind, von denen schließlich dann Kanten zum Zielort führen. Wir nennen diesen Graphen im weiteren den *Master-Graphen*.

Der Master führt also eine Kürzeste-Wege Berechnung auf diesem Graphen durch, wobei er für die nicht-gestrichelten Kanten Information über Verbindungen aus den einzelnen Teilsystemen benötigt, während für die gestrichelten Kanten Anschlußinformation benötigt wird.

Da im Prinzip jede dieser Kanten auf einem kürzesten Weg liegen kann, muß bei dieser Vorgehensweise die Dauer für jede Kante aus dem Teilsystem abgefragt werden. Dies führt also bei jeweils maximal r zeitlichen Alternativen, a Übergangsknoten zwischen Netz 1 und Netz 2 und b

Übergangsknoten zwischen Netz 2 und Netz 3 zu

$$r \cdot (a + b + a \cdot b)$$

Verbindungsabfragen und

$$r \cdot (a + b)$$

eher unproblematischen Anschlußfragen.

Der Aufwand entspricht also im wesentlichen dem Produkt der Größe der beiden Separatoren mal der maximalen Anzahl der in Frage kommenden zeitlichen Alternativen am Startort bzw. den Zwischenorten

Es ist daher wichtig, daß solche Verbindungsabfragen an Teilsysteme möglichst schnell durchgeführt werden können. Dies hängt natürlich davon ab, in welcher Information Wissen über existierende Verbindungen bereits in den Teilsystemen vorhanden ist. Im Idealfall liegen über *Preprocessing* (vgl. Abschnitt 1.1.4) bereits Informationen in Form von berechneten Verbindungen zu bestimmten Zeitpunkten vor, über die in $O(1)$ Zeit Information gewonnen werden kann (z. B. die Fahrzeit, die Anzahl der Umstiege usw.).

Kann diese Information nicht in $O(1)$ Zeit ermittelt werden, so ist im Extremfall eine Abfrage im Teilnetz, also eine Dijkstra-Berechnung durchzuführen. Dadurch steigt der Gesamtaufwand bei Teilnetzen mit n Knoten und m Kanten auf maximal

$$r \cdot (a + b + a \cdot b) \cdot O(m + n \log n).$$

Das hier diskutierte Verhalten von Kürzesten-Wege Algorithmen bei der Dekomposition von Graphen mittels Separatoren wurde ausführlich in [4] untersucht. Zwar handelt die Arbeit von planaren Graphen, aber die Planarität wird nur benötigt, um kleine Separatoren zu garantieren, die dort gesucht werden müssen, aber hier bei den Verkehrsanwendungen natürlich vorgegeben sind. Frederickson zeigt in dieser Arbeit, daß die wiederholte Dekomposition mittels Separatoren und die Verwendung von Preprocessing, um Information über kürzeste Verbindungen in $O(1)$ Zeit zur Verfügung zu haben, insgesamt zu einer Beschleunigung des Dijkstra-Algorithmus führen. Dies ist ein weiteres Indiz dafür, daß, falls möglich, bei der Verbindungssuche Preprocessing in den einzelnen Teilsystemen äußerst wünschenswert ist.

Weitere Hinweise zur Berechnung kürzester Wege mittels Dekomposition durch Separatoren insbesondere für Straßengraphen sind ausführlich in der Arbeit [15] dokumentiert.

Natürlich lassen sich bei der Berechnung im Master auch alle Beschleunigungsmöglichkeiten aus Abschnitt 3 nutzen.

4.1.3 Abfragekriterien im Basismodell

Da die Verbindungssuche durch den Master im Prinzip wieder auf einen Graphen zurückführbar ist, gelten für alle diskutierten Zielfunktionen die Überlegungen aus Abschnitt 1.3. Voraussetzung ist, daß die Teilsysteme die erforderliche Information bei den Abfragen zur Verfügung stellen und bei der Ankunft in einem Übergangsknoten die in Frage kommenden Abfahrten liefern.

Für eindimensionale Kriterien wie z. B. die Umsteigehäufigkeit bedeutet dies, daß nur eine Zahl, z. B. die Anzahl der Umstiege pro abgefragter Verbindung, zurückgegeben werden muß. Bei der mehrkriteriellen Bewertung müssen allerdings bei der Suche nach Pareto-optimalen Lösungen alle Pareto-optimalen Lösungen der abgefragten Teilstrecke zurückgegeben werden. Die hieraus resultierenden Anforderungen und Schwierigkeiten werden in Abschnitt 4.2 diskutiert.

Auch für Anfragen nach Fahrten innerhalb eines Tarifgebietes oder mit bestimmten Verkehrsmitteln ist eine verteilte Berechnung möglich, sofern die Teilsysteme solche Anfragen beantworten können. Diese Antworten werden als Attribute auf den Kanten des Graphen, auf dem der Master rechnet, aufgefaßt, und für diesen attributierten Graphen gilt dann das gleiche wie in Abschnitt 1.3. Die dort verwendeten Techniken gestatten es, über diese Attribute auf geeignete Teilgraphen überzugehen und die gewünschten Abfragen vorzunehmen.

Zusammenfassend zeigt sich, daß *bei schwach verbundenen Systemen alle Anfragen verteilt durchführbar sind, die auch in einem globalen System durchführbar sind*, falls die erforderlichen

Informationen aus den Teilsystemen zur Verfügung gestellt werden können. Die dafür erforderliche Anzahl von Abfragen bei Teilsystemen ist proportional zum Produkt aus der maximalen Anzahl der betrachteten Abfahrtzeiten sowie den Anzahlen der Verbindungsknoten zwischen den einzelnen Netzen.

4.2 Pareto-optimale Wege bei verteilter Verbindungssuche

Zur Durchführung einer mehrkriteriellen Verbindungssuche im Master-Graphen ist es für die Bestimmung *aller* Pareto-optimalen s, t-Wege erforderlich, daß die Teilsysteme *alle* Pareto-optimalen Lösungen der Verbindungen der Teilnetze, die Kanten des Master-Graphen entsprechen, zur Verfügung stellen.

Dabei entspricht jetzt eine Kante im Master-Graphen einer Menge von Verbindungen, nämlich einer für jedes Pareto-Optimum. Im Modell des Master-Graphen kann man dies dadurch berücksichtigen, daß entsprechend viele parallele Kanten eingeführt werden, so daß jede dieser Kanten einen Pareto-optimalen Weg repräsentiert und mit dem zugehörigen Vektor bewertet wird.

Durch diese Modellierung ist der Master-Graph wieder ein Graph mit mehrkriteriellen Kantenbewertungen, so daß alle in Abschnitt 2.2 vorgestellten Methoden auf ihn anwendbar sind. Insbesondere steht die Erweiterung des Dijkstra-Algorithmus zur *Berechnung aller Pareto-Optima im Master-Graphen* zur Verfügung. Wir illustrieren dies am Graphen aus Abbildung 6, der an Knoten c in 2 Teilnetze verlegt wird. Die Berechnung in den beiden Teilnetzen ergibt dann die Pareto-Optima $(0, 3), (1, 2), (2, 1), (3, 0)$ im Knoten c und $(0, 12), (4, 8), (8, 4), (12, 0)$ im Knoten e. Hieraus resultiert der in Abbildung 11 dargestellte Master-Graph, der dann auf dieselben Pareto-Optima im Knoten d wie der Ausgangsgraph führt.

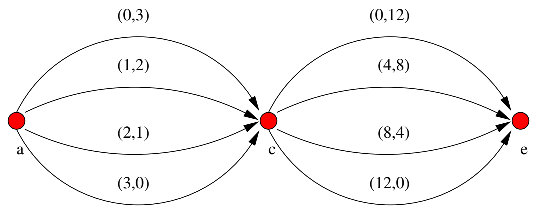

Abbildung 11: Der Master-Graph zum Beispiel aus Abbildung 5 mit $\{c\}$ als Separator

Das Beispiel zeigt auch, daß man auf *keine* Pareto-optimale Lösung eines Teilnetzes verzichten kann, wenn man *alle* Pareto-optimalen Lösungen des Gesamtnetzes im Master-Graphen berechnen will.

4.2.1 Approximierbarkeit Pareto-optimaler Wege bei verteilter Verbindungssuche

Wir stellen nun die Frage nach der Approximierbarkeit der Pareto-Optima bei der verteilten Verbindungssuche.

Die generelle Idee besteht darin, auf den Teilnetzen nur eine Auswahl der Pareto-optimalen Wege zu berechnen, die zugehörigen Bewertungsvektoren auf die Kanten des Master-Graphen zu übertragen, und dann im Master-Graphen ggf. eine weitere Auswahl Pareto-optimaler Wege zu ermitteln, die dann als Approximation der Pareto-Optima im Gesamtnetz genommen wird. Es kann also auf beiden Stufen, an den Teilnetzen und im Master, approximiert werden, wobei allerdings der Approximation in den Teilnetzen wegen der kleinen Kantenzahl der Wege im Master eine größere Bedeutung zukommt.

Die Verfahren der Gewichtung und der einfachen Skalierung aus Abschnitt 2.2.1 und 2.2.2 machen eine Aussage über die maximale Differenz pro Komponente des Bewertungsvektors. Da sich diese *additiv* entlang eines Weges ergibt, folgt, daß sich der in den Teilsystemen maximal mögliche Fehler auch additiv im Master fortsetzt. Allerdings sind die Kantenzahlen pro Weg in den Teilsystemen kürzer, so daß sich dies mit der Fortpflanzung im Master aufhebt.

Wir analysieren dies genauer am Verfahren der *einfachen Skalierung*.

Ist ℓ die Kantenzahl des approximierenden Weges W im Gesamtnetz, und zerfällt er bezüglich der Teilnetze einer verteilten Standardabfrage in die Wegstücke W_1, W_2, W_3 mit Kantenzahlen ℓ_1, ℓ_2, ℓ_3, so entsteht pro Teilnetz bei der einfachen Skalierung um den Faktor α ein Fehler von $\ell_i \cdot (\alpha - 1)$, der sich dann im Master zu $\ell \cdot (\alpha - 1)$ addiert. Im Prinzip entsteht also durch die verteilte Berechnung kein Verlust, denn auch im Gesamtnetz hätte sich der Fehler von $\ell \cdot (\alpha - 1)$ ergeben.

Der Grund hierfür ist die additive Fortpflanzung des Fehlers entlang der Kanten eines Weges. Da die Wege in Teilwege bezüglich der Komponenten zerfallen, ist der Fehler im Gesamtnetz gleich der Summe der Fehler in den Teilnetzen, sofern man nur in den Teilnetzen skaliert.

Eine zusätzliche Skalierung im Master-Graphen um den Faktor β erhöht diesen Fehler um $3(\beta - 1)$. Ist nämlich x ein Pareto-Optimum im Gesamtnetz, y die durch einfache Skalierung um α in den Teilnetzen berechnete Approximation von x und z die durch einfache Skalierung um β im Master-Graphen berechnete Approximation von y, so gilt

$$|x_k - z_k| \leq |x_k - y_k| + |y_k - z_k| \leq \ell(\alpha - 1) + 3(\beta - 1).$$

Für das *volle Approximationsschema von Warburton* läßt sich eine ähnliche Überlegung anstellen. Wir setzen zunächst eine Genauigkeit von δ in den Teilnetzen an und erhalten approximierende Mengen $Y_j(\delta)$ für die Teilnetze N_j. Ein Pareto-optimaler Weg W zerfällt bezüglich der Teilnetze in Pareto-optimale Wege W_j. Seien x bzw. x^j die zu W bzw. W_j gehörigen Pareto-Optima. Dann gilt $x_k = \sum_j x_k^j$ für $k = 1, \ldots, r$. Die Approximation auf den Teilnetzen sichert uns die Existenz von Pareto-optimalen Wegen W_j' mit zugehörigen Vektoren y^j für die $y_k^j \leq (1 + \delta)x_k^j$ gilt. Für den zusammengesetzten Weg $W' = \cup_j W_j^j$ mit Vektor y gilt dann

$$y_k = \sum_j y_k^j \leq \sum_j (1 + \delta)x_k^j \leq (1 + \delta)x_k.$$

Also definiert die Menge $Y(\delta)$ aller zusammengesetzten Wege W' eine δ-approximierende Menge für das Gesamtnetz, die *vollständig* und *ohne Verlust an Genauigkeit* durch eine *verteilte Verbindungsabfrage* berechnet werden kann.

Es ist sogar möglich, auf dem Master-Graphen eine zusätzliche Approximation vorzunehmen. Setzt man hier den Faktor ε an, so erhält man eine ε-approximierende Menge $Z(\varepsilon)$ für die Wege im Master-Graphen. Sei nun x ein Pareto-Optimum im Gesamtnetz, und y die oben aus den Teilnetzen berechnete δ-Approximation zu x. Für diese existiert wegen der ε-Approximation auf dem Master-Graphen ein Vektor z mit $z_k \leq (1 + \varepsilon)y_k$ für $k = 1, \ldots, r$. Also ist

$$z_k \leq (1 + \varepsilon)y_k \leq (1 + \varepsilon)(1 + \delta)x_k.$$

Die Gesamtgenauigkeit ergibt sich also *multiplikativ* aus den Genauigkeiten beider Approximationen.

Zusammenfassend läßt sich feststellen, daß die Approximationstechniken mit der verteilten Abfrage verträglich sind, und daß sich die Genauigkeit bei den Verfahren mit einer Kontrollmöglichkeit auch kontrollieren läßt. Generell gilt dabei, daß eine zusätzliche Approximation auf dem Master-Graphen mit einem weiteren Verlust an Genauigkeit einhergeht bzw. das Einhalten einer vorgegebenen Gesamt-Genauigkeit eine größere Genauigkeit in den Teilnetzen erfordert. Da der Master-Graph im allgemeinen relativ klein sein dürfte, sollte daher auf eine zweite Approximation auf dem Master verzichtet werden.

Pareto-optimale Wege lassen sich grundsätzlich approximativ mit einer verteilten Abfrage berechnen. Das volle Approximationsschema von Warburton sichert dabei prinzipiell eine beliebig einstellbare Genauigkeit, so daß bei polynomialer Rechenzeit kein untragbarer Verlust an herausragenden Gesamtlösungen befürchtet werden muß.

In der Praxis wird man allerdings wohl auf größere Geschwindigkeit setzen müssen, so daß das Verfahren von Warburton nicht in Frage kommt. Dann stehen die in Abschnitt 2.2 diskutierten Verfahren ebenfalls für eine verteilte Berechnung zur Verfügung. Sofern hier eine Fehlerabschätzung möglich ist, läßt sie sich auch auf die verteilte Verbindungsabfrage übertragen.

4.2.2 Komfortablere Information aus den Teilsystemen

Müssen bei den bisher betrachteten Anfragen in den Teilsystemen tatsächlich noch kürzeste-Wege Berechnungen vorgenommen werden, so steigt, wie oben erläutert, der Aufwand auf

$$r \cdot (a + b + a \cdot b) \cdot O(m + n \log n).$$

Hierbei entsteht sehr viel unnötiger Rechenaufwand, da die einzelnen Abfrage-Berechnungen unkoordiniert erfolgen. Man kann hier wesentlich effizienter werden, indem man spezielle Kenntnisse über den angewendeten Algorithmus in den Teilnetzen ausnutzt. Weiß man z. B. daß in Teilnetz 1 mit dem Dijkstra Algorithmus gerechnet wird, so reicht im Prinzip für jede zeitliche Alternative am Startort eine Dijkstra-Berechnung, die ja dann wegen der *Single Source Multiple Target*-Eigenschaft des Dijkstra Algorithmus die kürzesten Reisezeiten zu allen Übergangsknoten zwischen Netz 1 und Netz 2 berechnet. Diese Information könnte im Prinzip dem Master zur Verfügung gestellt werden, so daß er mit weit weniger Dijkstra-Berechnungen auskommt.

Die konsequente Übertragung dieser Idee auf alle beteiligten Teilnetze liefert dann bei maximal r zeitlichen Alternativen in einem Ort

- r Dijkstra Berechnungen am Ziel
- $r \cdot a$ Dijkstra Berechnungen an den Übergangsknoten von Netz 1 zu Netz 2
- $r \cdot b$ Dijkstra Berechnungen an den Übergangsknoten von Netz 2 zu Netz 3,

also insgesamt

$$r \cdot (a + b) \text{ Dijkstra-Berechnungen.}$$

Im Vergleich zum Basismodell werden also

$$r \cdot (a \cdot b) \text{ Dijkstra-Berechnungen}$$

eingespart.

Diese wesentliche Beschleunigung setzt voraus, daß die Teilsysteme bei einem übergebenen Paar vom Startort und Zeit, die jeweils benötigte Information (Fahrzeit, Anzahl Umstiege usw.) für jeden Übergangsort zum nächsten Teilnetz zurückgeben. Wir bezeichnen dies als *Single Source, Multiple Target* Information.

Für die mit diesem Ansatz behandelbaren Zielkriterien gilt das gleiche wie im Basismodell: Alle, die auf kürzeste Wege zurückführbar sind, können auch in diesem Modell optimal berechnet werden.

4.3 Stark verbundene Systeme

Stark verbundene Systeme zeichnen sich dadurch aus, daß die beteiligten Teilsysteme im Liniengraphen sehr viele Übergangspunkte haben. Sie operieren auf einer gemeinsamen Knotenmenge und unterscheiden sich nur um die jeweiligen Kantenmengen, die die Linien der einzelnen Netze ausmachen. In diesem Fall gibt es also keine kleinen Separatoren, durch die sich die Teilsysteme trennen lassen. Ein Beispiel hierfür ist das U- und S-Bahnnetz in Abbildung 1.

In diesem Fall versagen Mechanismen zur verteilten Berechnung gemäß des vorigen Abschnittes 4.1. Es bleibt dem Master nichts anderes übrig, als sich durch entsprechende Abfragen in den Teilsystemen eine vollständige Übersicht über das Gesamtsystem zu beschaffen.

Tatsächlich ist zu beobachten, daß für solche stark verbundenen Systeme ein einziges Abfragesystem zuständig ist. So wird z. B. im Berliner Raum ein einheitliches Abfragesystem im gesamten Verkehrsverbund angeboten, das Fahrten zwischen Bussen, U-Bahnen und S-Bahnen beliebig koppelt. Gleiches gilt für Verkehrsverbunde anderer Regionen. Dieser Fall wird daher bei der verteilten Verbindungssuche eine eher untergeordnete Rolle spielen und soll hier – trotz vorhandener theoretischer Modelle [4] – nicht weiter behandelt werden.

[4]Diese Modelle basieren auf einer hierarchischen Zerlegung des Graphen, bei dem Kanten und Knoten verschiedenen Hierarchiestufen (=Verkehrsmitteln) zugeordnet sind, vgl. [12]. Für die Verbindungsabfrage scheint jedoch ein integriertes Gesamtsystem geeigneter zu sein, was auch durch die Praxis bestätigt wird.

5 Resumée

In dieser Arbeit wurde nachgewiesen, daß sich die gängigen Abfragekriterien im öffentlichen Personenverkehr auf Kürzeste-Wege-Probleme in Graphen zurückführen lassen. Ferner wurde gezeigt, daß Kürzeste-Wege-Berechnungen grundsätzlich in verteilten Systemen möglich sind, wobei schwach zusammenhängende Systeme im Hinblick auf den Aufwand für eine verteilte Berechnung günstiger sind.

Hieraus resultiert, daß für alle Abfragekriterien, die noch mit Kürzesten-Wege Algorithmen behandelbar sind (vgl. Abschnitt 1.3), eine verteilte Verbindungsabfrage möglich ist. Dies gilt auch für die (approximative) Berechnung mehrkriterieller Zielfunktionen.

Bei der verteilten Verbindungsabfrage wurden zwei Modelle unterschieden. Ein Basismodell, bei dem die Teilsysteme nur die minimal erforderliche Information liefern, und ein komfortableres Modell, bei dem Kenntnisse über die Arbeitsweise des Algorithmus in den Teilsystemen vorausgesetzt wird. Die beiden Modelle unterscheiden sich in ihrem Berechnungsaufwand um eine Größenordnung (additiv gegenüber multiplikativ in der Anzahl der Übergangspunkte zwischen den Netzen).

Für die verteilte Verbindungsabfrage resultieren hieraus folgende Anforderungen. Die Subsysteme müssen im Basismodell in der Lage sein, jeweils beste Verbindungen bezüglich des jeweiligen Kriteriums berechnen zu können und den Ergebniswert an den Masteralgorithmus weiter zu geben. Es wird also nur die Information aus den Teilsystemen benötigt, die direkt in die Zielfunktion bzw. die benötigten Attribute eingeht. Damit ist jedes verteilte System genauso gut wie das schwächste seiner Teilsysteme.

Eine Beschleunigung über das Basismodell hinaus wird dadurch erreicht, daß statt einer Ort-zu-Ort-Abfrage eine Abfrage von einem Ort zu einer Menge anderer ausgezeichneter Orte durchgeführt wird. Da diese Vorgehensweise inhärent für Kürzeste-Wege Algorithmen ist, kann diese komplexitätsmindernd verwendet werden, sofern diese Information in den Teilsystemen explizit gemacht und an den Master weitergegeben werden kann.

Die in dieser Studie angestellten Überlegungen orientierten sich jeweils an der Ermittlung *optimaler Verbindungen* innerhalb eines bestimmten Zeitraumes am Ausgangsort. Ist man nur an approximativen Lösungen interessiert, so sind Beschleunigungen gemäß Abschnitt 3 möglich. Die Grundlage bei der Beschleunigung basiert wieder auf Kürzeste-Wege Algorithmen, und sie kann sowohl in einem Gesamtsystem, wie auch bei der verteilten Verbindungsabfrage angewendet werden. Solche Beschleunigungsmethoden in Verbindung mit verteilter Berechnung sind auch aus anderen Anwendungen (Schaltkreislayout, vgl. etwa [11]) bekannt.

Literatur

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows. Theory, Algorithms, and Applications*. Prentice Hall, Englewood Cliffs, NJ, 1993.

[2] B. V. Cherkassy, A. V. Goldberg, and T. Radzik. Shortest path algorithms: Theory and experimental evaluation. *Math. Programming*, 73:129–174, 1996.

[3] R. Dechter and J. Pearl. Generalized best-first search strategies and the optimality of A^*. *J. Assoc. Comp. Mach.*, 32(3):505–536, 1985.

[4] G. N. Frederickson. Fast algorithms for shortest paths in planar graphs with applications. *SIAM J. Comput.*, 16:1004–1022, 1987.

[5] M. Gondran and M. Minoux. *Graphs and Algorithms*. John Wiley & Sons, New York, 1984.

[6] P. Hart, N. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Systems, Science and Cybernetics*, SCC-4(2):100–107, 1968.

[7] U. Huckenbeck and D. Ruland. A generalized best-first search method in graphs. In R. H. Möhring, editor, *Proceedings 16th International Workshop on Graph-Theoretic Concepts in*

Computer Science WG'90, pages 41–60. Springer-Verlag, Lecture Notes in Computer Science, vol. 484, 1990.

[8] J. M. Jaffe. Algorithms for finding paths with multiple constraints. *Networks*, 14:95–116, 1984.

[9] T. Kämpke and F. J. Radermacher. Deutschlandweite Verbindungsauskunft im öffentlichen Personennahverkehr: verteilte Wegesuche in Graphen. Manuskript, 1999.

[10] E. L. Lawler, M. G. Luby, and B. Parker. Finding shortest paths in very large networks. In M. Nagl and J. Perl, editors, *Proceedings 9th International Workshop on Graph-Theoretic Concepts in Computer Science WG'83*, pages 184–199. Trauner Verlag, Linz, 1983.

[11] T. Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout*. Wiley-Teubner, 1990.

[12] T. Lengauer and E. Wanke. Efficient solutions of connectivity problems on hierarchically defined graphs. *SIAM J. Comput.*, 17:1063–1080, 1988.

[13] N. J. Nilsson. *Principles of Artificial Intelligence*. Morgan Kaufmann Publishers, 1980.

[14] I. Pohl. Bi-directional search. In B. Meltzer and D. Mitchie, editors, *Machine Intelligence*, pages 127–140. Edinburgh University Press, Edinburgh, 1971.

[15] B. Riedhofer. Hierarchische Straßengraphen. Master's thesis, Universität Stuttgart, Fakultät Informatik, 1997.

[16] G. Rote. Path problems in graphs. In G. Tinhofer, E. Mayr, H. Noltemeier, and M. Sysło, editors, *Computational Graph Theory*, Computing Supplementum 7, pages 155–190. Springer-Verlag Wien, 1990.

[17] R. Sedgewick and J. S. Vitter. Shortest paths in Euclidean graphs. *Algorithmica*, 1:31–48, 1986.

[18] D. Theune. *Robuste und effiziente Methoden zur Lösung von Wegproblemen*. Teubner Verlag, Stuttgart, 1995.

[19] A. Warburton. Approximation of Pareto optima in multiple-objective, shortest-path problems. *Oper. Res.*, 35(1):70–79, 1987.

Reports from the group

“Combinatorial Optimization and Graph Algorithms”

of the Department of Mathematics, TU Berlin

698/2000 *Sándor P. Fekete, Ekkehard Köhler, and Jürgen Teich:* More-dimensional packing with order constraints

697/2000 *Sándor P. Fekete, Ekkehard Köhler, and Jürgen Teich:* Extending partial suborders and implication classes

696/2000 *Sándor P. Fekete, Ekkehard Köhler, and Jürgen Teich:* Optimal FPGA module placement with temporal precedence constraints

695/2000 *Sándor P. Fekete, Henk Meijer, André Rohe, and Walter Tietze:* Solving a “hard” problem to approximate an “easy” one: heuristics for maximum matchings and maximum Traveling Salesman Problems

694/2000 *Esther M. Arkin, Sándor P. Fekete, Ferran Hurtado, Joseph S. B. Mitchell, Marc Noy, Vera Sacristánm and Saurabh Sethia:* On the reflexivity of point sets

693/2000 *Frederik Stork and Marc Uetz:* On the representation of resource constraints in project scheduling

691/2000 *Martin Skutella and Marc Uetz:* Scheduling precedence constrained jobs with stochastic processing times on parallel machines

689/2000 *Rolf H. Möhring, Martin Skutella, and Frederik Stork:* Scheduling with AND/OR precedence constraints

685/2000 *Martin Skutella:* Approximating the single source unsplittable min-cost flow problem

684/2000 *Han Hoogeveen, Martin Skutella, and Gerhard J. Woeginger:* Preemptive scheduling with rejection

683/2000 *Martin Skutella:* Convex quadratic and semidefinite programming relaxations in Scheduling

682/2000 *Rolf H. Möhring and Marc Uetz:* Scheduling scarce resources in chemical engineering

681/2000 *Rolf H. Möhring:* Scheduling under uncertainty: optimizing against a randomizing adversary

680/2000 *Rolf H. Möhring, Andreas S. Schulz, Frederik Stork, and Marc Uetz:* Solving project scheduling problems by minimum cut computations (Journal version for the previous Reports 620 and 661)

674/2000 *Esther M. Arkin, Michael A. Bender, Erik D. Demaine, Sándor P. Fekete, Joseph S. B. Mitchell, and Saurabh Sethia:* Optimal covering tours with turn costs

669/2000 *Michael Naatz:* A note on a question of C. D. Savage

667/2000 *Sándor P. Fekete and Henk Meijer:* On geometric maximum weight cliques

666/2000 *Sándor P. Fekete, Joseph S. B. Mitchell, and Karin Weinbrecht:* On the continuous Weber and k-median problems

664/2000 *Rolf H. Möhring, Andreas S. Schulz, Frederik Stork, and Marc Uetz:* A note on scheduling problems with irregular starting time costs

661/2000 *Frederik Stork and Marc Uetz:* Resource-constrained project scheduling: from a Lagrangian relaxation to competitive solutions

658/1999 *Olaf Jahn, Rolf H. Möhring, and Andreas S. Schulz:* Optimal routing of traffic flows with length restrictions in networks with congestion

655/1999 *Michel X. Goemans and Martin Skutella:* Cooperative facility location games

654/1999 *Michel X. Goemans, Maurice Queyranne, Andreas S. Schulz, Martin Skutella, and Yaoguang Wang:* Single machine scheduling with release dates

653/1999 *Andreas S. Schulz and Martin Skutella:* Scheduling unrelated machines by randomized rounding

646/1999 *Rolf H. Möhring, Martin Skutella, and Frederik Stork:* Forcing relations for AND/OR precedence constraints

640/1999 *Foto Afrati, Evripidis Bampis, Chandra Chekuri, David Karger, Claire Kenyon, Sanjeev Khanna, Ioannis Milis, Maurice Queyranne, Martin Skutella, Cliff Stein, and Maxim Sviridenko:* Approximation schemes for minimizing average weighted Completion time with release dates

639/1999 *Andreas S. Schulz and Martin Skutella:* The power of α-points in preemptive single machine scheduling

634/1999 *Karsten Weihe, Ulrik Brandes, Annegret Liebers, Matthias Müller–Hannemann, Dorothea Wagner and Thomas Willhalm:* Empirical design of geometric algorithms

633/1999 *Matthias Müller–Hannemann and Karsten Weihe:* On the discrete core of quadrilateral mesh refinement

632/1999 *Matthias Müller–Hannemann:* Shelling hexahedral complexes for mesh generation in CAD

631/1999 *Matthias Müller–Hannemann and Alexander Schwartz:* Implementing weighted b-matching algorithms: insights from a computational study

629/1999 *Martin Skutella:* Convex quadratic programming relaxations for network scheduling problems

628/1999 *Martin Skutella and Gerhard J. Woeginger:* A PTAS for minimizing the total weighted completion time on identical parallel machines

624/99 *Rolf H. Möhring:* Verteilte Verbindungssuche im öffentlichen Personenverkehr: Graphentheoretische Modelle und Algorithmen

627/1998 *Jens Gustedt:* Specifying characteristics of digital filters with FilterPro

620/1998 *Rolf H. Möhring, Andreas S. Schulz, Frederik Stork, and Marc Uetz:* Resource constrained project scheduling: computing lower bounds by solving minimum cut problems

619/1998 *Rolf H. Möhring, Martin Oellrich, and Andreas S. Schulz:* Efficient algorithms for the minimum-cost embedding of reliable virtual private networks into telecommunication networks

618/1998 *Friedrich Eisenbrand and Andreas S. Schulz:* Bounds on the Chvátal rank of polytopes in the 0/1-Cube

617/1998 *Andreas S. Schulz and Robert Weismantel:* An oracle-polynomial time augmentation algorithm for integer proramming

616/1998 *Alexander Bockmayr, Friedrich Eisenbrand, Mark Hartmann, and Andreas S. Schulz:* On the Chvátal rank of polytopes in the 0/1 cube

615/1998 *Ekkehard Köhler and Matthias Kriesell:* Edge-dominating trails in AT-free graphs

613/1998 *Frederik Stork:* A branch and bound algorithm for minimizing expected makespan in stochastic project networks with resource constraints

612/1998 *Rolf H. Möhring and Frederik Stork:* Linear preselective policies for stochastic project scheduling

611/1998 *Rolf H. Möhring and Markus W. Schäffter:* Scheduling series-parallel orders subject to 0/1-communication delays

609/1998 *Arfst Ludwig, Rolf H. Möhring, and Frederik Stork:* A computational study on bounding the makespan distribution in stochastic project networks

605/1998 *Friedrich Eisenbrand:* A note on the membership problem for the elementary closure of a polyhedron

596/1998 *Andreas Fest, Rolf H. Möhring, Frederik Stork, and Marc Uetz:* Resource constrained project scheduling with time windows: A branching scheme based on dynamic release dates

595/1998 *Rolf H. Möhring Andreas S. Schulz, and Marc Uetz:* Approximation in stochastic scheduling: The power of LP-based priority policies

591/1998 *Matthias Müller–Hannemann and Alexander Schwartz:* Implementing weighted b-matching algorithms: Towards a flexible software design

590/1998 *Stefan Felsner and Jens Gustedt and Michel Morvan:* Interval reductions and extensions of orders: bijections to chains in lattices

584/1998 *Alix Munier, Maurice Queyranne, and Andreas S. Schulz:* Approximation bounds for a general class of precedence constrained parallel machine scheduling problems

577/1998 *Martin Skutella:* Semidefinite relaxations for parallel machine scheduling

Reports may be requested from: Hannelore Vogt-Möller
Fachbereich Mathematik, MA 6–1
TU Berlin
Straße des 17. Juni 136
D-10623 Berlin – Germany

e-mail: moeller@math.TU-Berlin.DE

Reports are also available in various formats from

`http://www.math.tu-berlin.de/coga/publications/techreports/`

and via anonymous ftp as

`ftp://ftp.math.tu-berlin.de/pub/Preprints/combi/Report-`*number*`-`*year*`.ps`