UNIVERSITÄT DORTMUND

Ulf Schneider

PG 431: Metaheuristiken

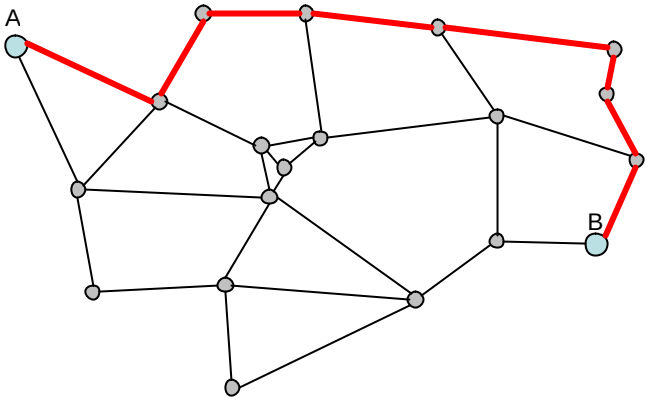
Das Routing Problem

Dortmund, im März 2003

1 Einleitung 3
2 Grundlagen 4
2.1 Graphentheorie 4
2.2 Darstellung von Graphen in Rechnern 6
2.2.1 Matrixdarstellung 6
2.2.2 Listendarstellung 7
2.2.3 Sortierverfahren 8
3 Single Source Shortest Path 8
3.1 Der Algorithmus von Dijkstra 8
3.2 Der Algorithmus von Fulkerson zur Bestimmung eines minimal spannenden Wurzelbaumes 9
3.3 Baumalgorithmen 10
3.3.1 FIFO-Kürzeste Wege 10
3.3.2 Weitere Algorithmen 11
4 All Pairs Shortest Path 11
4.1 Der Tripel-Algorithmus 11
5 Entfernung zwischen zwei Punkten 12
6 Routing auf EDV-Netzwerken 12
6.1 Der Netchange-Algorithmus 13
7 Literatur 15

1 Einleitung

In vielen Gebieten ist es wichtig, eine Strecke zwischen zwei Orten zu ermitteln, die nicht direkt miteinander verbunden, sondern lediglich indirekt über andere Orte zu erreichen sind. Beispiele hierfür sind z.B. die Suche nach einer Straßenverbindung von Dortmund nach München, die Suche nach einer Busverbindung von der Universität in die Innenstadt, die Suche nach einer Möglichkeit z. B. über das Internet ein Datenpaket von einem Webserver aus an einen Clientrechner zu schicken.



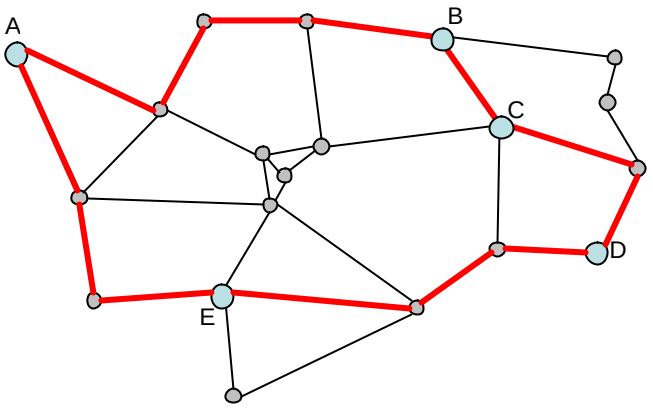
Während die ersten beiden Wege für den Nutzer transparent sein müssen – die Person muss wissen, wo sie umsteigen bzw. abbiegen muss – so ist es beim Routing im Internet nicht wichtig, ob der Nutzer den Weg seiner Daten nachvollziehen kann. Bei den ersten beiden Beispielen ist es auch weiterhin recht üblich, die Suche nach einer guten Lösung manuell, also ohne die Unterstützung einer wie auch immer gearteten elektronischen Einrichtung zu lösen. In den letzten 10 Jahren wird allerdings auch hier vermehrt auch von Endkunden eine elektronische Fahrplanauskunft oder ein Routenplaner genutzt.

Hauptaugenmerk bei der Suche nach einer Verbindung ist natürlich das zuverlässige Erreichen des Zielortes, wichtig aber sind natürlich auch unter anderem folgende Kriterien:

- Kostenminimal – der Zielort sollte auf dem
  - schnellsten (man nimmt den Weg über die Autobahn und nicht über die Landstraße, auch wenn dieser Weg kürzer sein kann)
  - kürzesten (Vermeidung von unnötigen Umwegen, eine gute Busverbindung aus obigem Beispiel geht sicher nicht über Unna)
  - einfachsten Weg erreicht werden (z.B. bei Busfahrten die Anzahl der Umstiege minimieren).

  Obige 3 Kriterien können sich aber durchaus widersprechen, so kann eine Busfahrt unter Umständen länger dauern, auch über eine größere Strecke führen, als eine andere, aber den Vorteil haben, dass man nur halb so oft umsteigen muss und deshalb diese Lösung der anderen vorzuziehen ist.
- Wahl von Zwischenorten – z.B. bei der Autoreise ein Zwischenstopp in einer Stadt, um eine Sehenswürdigkeit zu besichtigen oder ein paar Bekannte einzusammeln.
- Zeitabhängig – man möchte zu einer bestimmten Zeit einen Ort erreichen, bzw. verlassen, bei Busfahrten kann dies heißen, das Busse unter Umständen zu komplett anderen Zeiten fahren / gar nicht mehr fahren.

In einem weiteren Schritt kann es dann auch interessant werden, nicht nur eine Verbindung von A nach B zu suchen, sondern ausgehend von einem Punkt A eine Reihe von Orten in einer beliebigen, aber kostenminimalen Reihenfolge zu besuchen. Dies gehört allerdings streng genommen nicht mehr in das Gebiet des Routing-Problems, sondern in das Gebiet der Tourenplanung, bzw. des so genannten „Traveling-Salesman-Problem“ (kurz TSP). Zur Errechnung einer so genannten Rundreise braucht man aber die Angabe über die Entfernung zweier Orte, diese Angabe wird dann wiederum über eine Lösung des Routing-Problems.



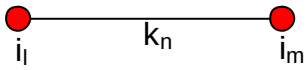
2 Grundlagen

2.1 Graphentheorie

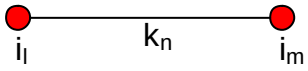
Zur formalen Darstellung von Orten und deren Verbindung bedient man sich der Graphentheorie. Orte haben hierbei ihre Entsprechung als Knoten und Strecken als Kanten, die Kosten zur Zurücklegung einer Strecke wird als Gewichtung einer Kante realisiert.

Ein Graph G besteht aus einer nichtleeren Menge von Knoten V und einer Menge von Kanten E, wobei E und V keine gemeinsamen Elemente haben dürfen (also $V \cap E = 0$). Hinzu kommt eine Abbildung w (Inzidenzabbildung oder Verbindungsabbildung), die jeder Kante einen Start- und einen Endknoten zuordnet. Hierbei gibt es folgende Sorten von Graphen:

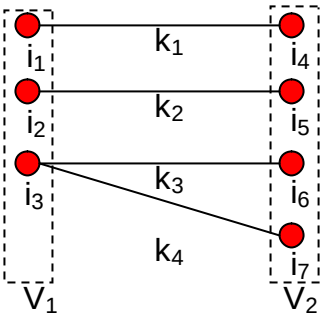
- ungerichteten Graphen: hier sind alle Verbindung zwischen zwei Knoten beidseitig benutzbar (Notation $w : k_n \rightarrow [i_l, i_m]$ oder kurz $k_n = [i_l, i_m]$)



- gerichteten Graphen: hier gibt es zwar eine Verbindung vom jeweiligen Start- zum Endknoten einer Kante, aber nicht automatisch wieder zurück. Die Kanten werden in diesem Fall als Pfeile bezeichnet (Notation $w : k_n \rightarrow (i_l, i_m)$ oder kurz $k_n = (i_l, i_m)$)



  Man sagt auch k_n ist positiv inzident zu i_l, der Pfeil k_n geht vom Knoten i_l aus, oder k_n ist negativ inzident zu i_m, der Pfeil k_n mündet in den Knoten i_m.
- Schlichte Graphen: Graphen, die keine Schlingen und oder parallele Kanten/Pfeile besitzen.
- endliche Graphen: V und E sind beide endlich
- Digraphen: Ein endlicher, schlichter, gerichteter Graph.



- Bipartite Graphen: Die Knotenmenge V ist in zwei Knotenmengen V_1, V_2 teilbar, mit $V_1 \cup V_2 = V$, $V_1 \cap V_2 = 0$ und sämtliche Knoten können zwar mit Knoten aus der anderen Menge verbunden sein, aber nicht untereinander, es darf also keine Kanten oder Pfeile der Art $k_n = [i_l, i_m]$ mit $i_l, i_m \in V_n$, $n \in [1,2]$ geben.
- Vollständige Graphen: ein Graph heißt vollständig, wenn jeder Konten direkt mit jedem andern Knoten verbunden ist und bei gerichteten Graphen dies in jede Richtung.
- Zusammenhängende Graphen: ein Graph heißt zusammenhängend, wenn jeder Konten mit jedem andern Knoten über eine Folge von Kanten verbunden ist, sind diese beiden Knoten bei gerichteten Graphen nicht nur verbunden, sogar erreichbar, so spricht man von einem stark zusammenhängenden Graphen.
- Symmetrische Graphen: gerichtete Graphen, bei denen gilt, $(i,j) \in E \Rightarrow (j,i) \in E$, ein Graph hießt antisymmetrisch, wenn gilt, $(i,j) \in E \Rightarrow (j,i) \notin E$.
- Bewertete Graphen: Ein Graph, bei dem den Kanten eine Zahl zugeordnet wurde. Meistens sind dies Informationen über die Entfernung zwischen den beiden Knoten, zwischen denen die Kante verläuft. Auch wenn eine Kante im Allgemeinen negativ bewertet werden kann, so wird dies im folgenden ausgeschlossen, die noch vorzustellenden Algorithmen können zwar teilweise mit negativen Kantenbewertungen umgehen, aber der zusätzliche Aufwand, der mit negativen Kantenbewertungen verbunden ist, erschwert ein Verständnis der Algorithmen.
- Teilgraphen: Eine Auswahl von Knoten und Kanten eines Graphen.

Man unterscheidet noch die Begriffe (unmittelbarer) Nachfolger und (unmittelbarer) Vorgänger eines Knotens. Ein Nachfolger eines Knotens i ist ein Knoten j, für den es eine Kante zwischen i und j oder ein Pfeil von j nach i gibt. Die Menge der Nachfolger wird mit $N(i)$ bezeichnet, die Vorgänger sind entsprechend definiert und die Menge der Vorgänger wird mit $V(i)$ bezeichnet, die Vorgänger und Nachfolger bilden zusammen die Nachbarn des Knoten i und die Menge der Nachbarn wird als $NB(i)$ bezeichnet.

Der Grad g_i eines Knoten gibt an, wie viele Kanten inzidient zum Knoten sind, hierbei sind Schlingen doppelt zu zählen. Bei gerichteten Graphen unterscheidet man zwischen positivem Grad g_i^+, also der Menge, der von i ausgehenden Pfeilen und dem negativem Grad g_i^-, also der Menge, der in i mündenden Pfeile.

Weiterhin wird ein Knoten eines gerichteten Graphen als Quelle bezeichnet, wenn gilt $g_q^+ > 0$ und $g_q^- = 0$, das Gegenstück mit $g_q^+ = 0$ und $g_q^- > 0$ wird als Senke bezeichnet. Sind $g_s^+ = 0$ und $g_s^- = 0$, bzw. ist im ungerichteten Graphen $g_i = 0$, so spricht man von einem isolierten Knoten.

Eine Zusammenhangskomponente ist ein maximaler zusammenhängender Teilgraph, in dem Teilgraphen müssen also alle mögliche Kanten und Knoten bzw. Pfeile vorhanden sein, die zusammenhängend sind, bei starken Zusammenhangskomponenten müssen diese stark zusammenhängend sein.

Ein zusammenhängender, kreisfreier Graph heißt Baum, besteht der Graph aus mehreren kreisfreien Zusammenhangskomponenten wird der Graph als Wald und jede Zusammenhangskomponente als Baum bezeichnet. Ein Baum hat einen Knoten mehr, als Kanten.
Gibt es eine Quelle r in einem Baum, von dem aus alle anderen Knoten erreicht werden können (also die Knoten nicht nur verbunden sind), so heißt dieser Baum Wurzelbaum mit der Wurzel r. In einem Wurzelbaum, kann es keine 2 Quellen geben, da die Quellen zwar miteinander verbunden sind, aber gegenseitig nicht erreichbar.
Ist der Baum Teilgraph eines Graphen G und hat die selbe Knotenmenge, so nennt man diesen Teilgraphen spannenden Baum oder Gerüst, dieser heißt in einem bewerteten Graphen sogar minimal, wenn es keinen anderen spannenden Baum gibt, dessen Summe von Kantenbewertungen kleiner ist.

2.2 Darstellung von Graphen in Rechnern

Da Datenstrukturen zur Speicherung von Verbindungen eigentlich bekannt sein sollten, folgt hier nur eine kurze, ausgewählte Zusammenfassung von möglichen Formen der Speicherung. Es gibt keine für alle Algorithmen optimale Darstellungsform von Graphen, bei den im späteren Teil vorgestellten Algorithmen wird die jeweils optimale Form der Speicherung genannt.

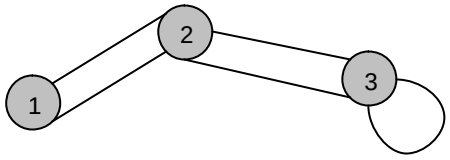
2.2.1 Matrixdarstellung

Es gibt mehrere Möglichkeiten einen Graphen als Matrix darzustellen, je nachdem, ob der Graph bewertet oder unbewertet, gerichtet oder ungerichtet ist. Im Folgenden werden nur gerichtete Graphen dargestellt, ungerichtete Graphen können leicht entsprechend dargestellt werden, da eine Verbindung von einem Knoten i zu einem Knoten j in einem ungerichteten Graphen auch automatisch bedeutet, dass es eine Verbindung von j nach i gibt, kann hier auf eine vollständige Matrix verzichtet werden und es ist Speicher-effizienter nur die Elemente über oder unter der Hauptdiagonale zu speichern (im Allgemeinen beschränkt man sich auf die Speicherung der Elemente unter der Hauptdiagonale). Allerdings ist es hierbei erforderlich, eine zusätzliche Logik zu entwerfen, die die Kanten aus der nicht gespeicherten einen Dreiecksmatrix automatisch auf die andere Dreiecksmatrix abbildet. Bei großen Projekten mit vielen Kanten kann sich der minimale zusätzliche Rechenaufwand aber durchaus lohnen, da die Speichereinsparung fast 50% beträgt und Änderungen in der Matrix nur an einer Stelle und nicht an zwei durchgeführt werden müssen, was wiederum Daten-Inkonsistenz vermeidet.

Elementar bei den folgenden Vorfahren ist, dass die Knoten keine frei wählbare Bezeichnung haben, sondern lediglich durchnummeriert sind. Eine Abbildung der Nummern auf eine frei wählbare Bezeichnung kann aber leicht mittels eines entsprechenden Arrays oder ähnlichem realisiert werden. Die Anzahl der Knoten im Graphen G betrage n, die Anzahl der Kanten m.

Adjazenzmatrix (bei unbewerteten Graphen)**:** Eine Matrix A (auch AZ genannt), für die gilt a_{ij} = Anzahl der von i nach j führenden Pfeile.

$$A(G) = \begin{pmatrix} 0 & 2 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$



Kosten- und Kapazitätsmatrix (bei bewerteten Digraphen)**:** Kommt zu den eigentlichen Kanten noch eine zugehörige Kostenbewertung c, eine minimale Kapazitätsbewertung l, sowie eine maximale Kapazitätsbewertung k, so hat man drei n×n-Matrizen, der Form

1. Eine Kostenmatrix C(G)=(c_{ij}) mit

$$c_{ij} = \begin{cases} c(i,j) & \text{für alle } (i,j) \in E \\ 0 & \text{für } i = j \\ \infty & \text{sonst} \end{cases}$$

2. Eine Minimalkapazitätsmatrix LA(G)=(l_{ij}) mit

$$l_{ij} = \begin{cases} l(i,j) & \text{für } (i,j) \in E \\ 0 & \text{sonst} \end{cases}$$

3. Einen Kapazitätsmatrix K(G)=(k_{ij}) mit

$$k_{ij} = \begin{cases} k(i,j) & \text{für } (i,j) \in E \\ 0 & \text{sonst} \end{cases}$$

Knotenbewertung: Es kann zusätzlich noch interessant sein, zu speichern, wie viel ein Knoten anbieten, bzw. nachfragen kann (z.B. interessant in der Logistik, um zu speichern wie viel ein Produzent von einer Menge anbieten, bzw. ein Kunde verbrauchen kann), hierzu speichert einfach zwei zusätzliche Liste mit n Elementen, wobei die Liste $a = (a_1, \ldots, a_n)$ die Angebote enthält und die Liste $b = (b_1, \ldots, b_n)$ die Nachfrage.

2.2.2 Listendarstellung

Adjazenz- oder Knotenorientierte Liste: Hier wird für jeden Knoten eine Liste von Nachfolgern gespeichert. Dieses Verfahren lohnt sich meist allerdings nur für recht kleine Mengen von Kanten im Verhältnis zu den Knoten. Für einige Algorithmen ist es allerdings von Vorteil, wenn alle von einem Knoten ausgehenden Pfeile direkt nachvollziehbar sind. Parallel dazu kann man, sofern benötigt entsprechende Listen mit den Kosten- und Kapazitätsbewertungen führen.

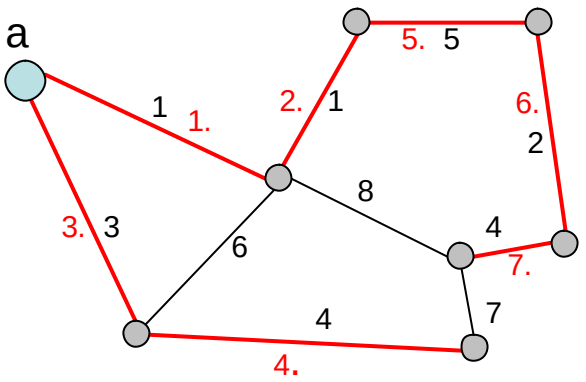
2.2.3 Sortierverfahren

Sortierverfahren sollten noch aus den Grundvorlesungen Programmierung und Datenstrukturen bekannt sein. Außerdem sind die Vorgehensweisen, wie genau das Sortieren bewerkstellig wird, nicht wirklich entscheidend für das Verständnis einer Lösung des Routing-Problems. Einzig und alleine wichtig ist, dass sichergestellt werden muss, dass nach Sortieren der Kantenbewertung sichergestellt ist, dass die zu dieser Bewertung gehörige Kante auch effizient wieder gefunden werden kann.

3 Single Source Shortest Path

Ausgehend von einem festen Startpunkt aus, kann es interessant sein, zu erfahren, über welchen Weg alle anderen Punkte im System optimal zu erreichen sind, bzw. welche Strecke zwischen ihnen liegt. Diese Art von Problem nennt man „single source shortest path“. Anwendungsgebiet hierfür ist z.B. die Verlegung von Rohrleitungen zur Versorgung eines Neubaugebietes abzweigend von einem Punkt an der Hauptwasserleitung, die in der der Nähe verläuft. Diese Routing-Probleme können direkt auf die Graphentheorie übertragen werden, da bei Übertragung der Realität auf den entsprechenden Graphen nichts anderes gesucht wird, als ein minimaler Spannbaum.

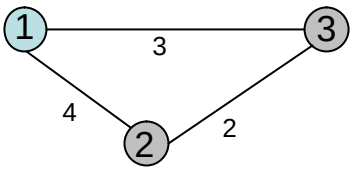
3.1 Der Algorithmus von Dijkstra



Der Algorithmus von Dijkstra beginnt mit dem Baum T_a, T_a besteht zu Beginn des Durchlaufes des Algorithmus nur aus dem Startknoten a. Nun werden alle Kanten ihrer Bewertung entsprechend aufsteigend sortiert. Es wird die Kante k mit der niedrigsten (nicht negativen) Bewertung dem Baum hinzugefügt, deren Startknoten in T_a liegt und deren Endknoten $p \in IK$ ist. p wird ebenfalls de Baum hinzugefügt. IK ist hierbei die Menge der Knoten, die noch nicht in den Baum aufgenommen worden sind, am Anfang ist $IK = V \setminus \{a\}$, also die Menge aller Knoten, außer dem Startknoten. Dieser Algorithmus terminiert nach n-1 Iterationen. Die Laufzeit hängt stark von der zu Grunde gelegten Datenstruktur für den Graphen ab und bei effizienter Realisierung kann die Laufzeit in Größenordnungen von $O(m \cdot \log_2 m)$ oder $O(n + m \cdot \log_2 n)$ liegen. Diese Vorgehensweise funktioniert allerdings nur bei zusammenhängenden Graphen, ist der Graph nicht zusammenhängend, muss vorher entweder der Graph auf die Zusammenhangskomponente, die den Startpunkt enthält, reduziert werden, oder abgefragt werden, ob das nächste hinzuzufügende Elemente nur noch die minimale Entfernung ∞ hat.

3.2 Der Algorithmus von Fulkerson zur Bestimmung eines minimal spannenden Wurzelbaumes

Betrachtet man nebenstehenden Wurzelbaum, so ist erkenntlich, dass ein minimal spannender Wurzelbaum über die mit 4 und 2 bewerteten Pfeile (1,2) und (2,3) geht. Der Algorithmus von Dijkstra wählt aber dank seines Greedy-Ansatzes erst den mit 3 niedriger bewerteten Pfeil (1,3), (2,3) kann nicht als nächster Pfeil gewählt werden, da dies entgegen der Pfeilrichtung wäre, also wählt der Algorithmus als nächstes den Pfeil (1,2). Der minimale spannende Wurzelbaum aus dem Beispiel oben hat zusammen eine Kostenbewertung von 6, der Baum aus dem Algorithmus von Dijkstra aber von 7.



Fulkerson nutzt nun folgende Eigenschaften von Wurzelbäumen aus:

- In jeden Knoten des Wurzelbaumes mit Ausnahme der Wurzel mündet genau ein Pfeil.
- In jede zusammenhängende Teilmenge der Knotenmenge, die nicht die Wurzel enthält, mündet genau ein Pfeil, dessen Anfangsknoten zum Komplement der Teilmenge gehört.

Um nun für den Graphen G=(V,E,c) mit n Knoten den zugehörigen Wurzelbaum zu ermitteln, benutzt der Algorithmus zwei Phasen, den Vorwärts- und den Rückwärtsprozess. Weiterhin benutzt der Algorithmus zwei zusammenhängende Stapel PE und KOM, der Stapel PE beinhaltet die Menge der für den Baum in Frage kommenden Pfeile und in KOM sind die von dem jeweiligen Pfeil erreichten bisher gebildeten Zusammenhangskomponenten gespeichert.

Der Vorwärtsprozess: Zunächst wird der kleinste Kantenwert c_{min} ermittelt, danach werden alle Knoten j=1 ... n, außer der Wurzel durchgegangen und geguckt, ob dieser Knoten j eine in sich eingehende Kante mit der Bewertung c_{min} hat, ist dies der Fall, so wird der Pfeil auf den Stapel PE gelegt und der Endknoten als einelementige Menge auf den Stapel KOM gelegt. Von allen weiteren in j eingehenden Pfeilen wird die Bewertung c_{ij} um c_{min} reduziert und c_{min} wird zum Wert Z, der den Wert des Baumes darstellt, addiert. Dieses Verfahren wird wiederholt, bis n-1 Kanten nach diesem Verfahren ausgewählt wurden. Ist der Baum nach hinzufügen von n-1 Kanten zusammenhängend und somit kreisfrei, so ist der gebildete Graph der gesuchte minimale spannende Wurzelbaum und der Rückwärtsprozess wird begonnen. Andernfalls suche für jede gefundene Zusammenhangskomponente nach allen Kanten, deren Endknoten sich schon in der Zusammenhangskomponente befinden, deren Anfangsknoten aber noch nicht. Unter alle diesen Kanten wird diejenige mit der minimalen Bewertung c_{min} ausgewählt und auf PE abgelegt, auf KOM die Zusammenhangskomponente, von allen anderen Kanten, die in dieser Zusammenhangskomponente enden, wird c_{min} abgezogen und zu Z wird c_{min} addiert.

Der Rückwärtsprozess: Gearbeitet wird hier mit dem Baum $T = (V, \overline{E})$, $\overline{E}$ ist zu Beginn des Verfahrens die leere Menge. Nun wird von PE der oberste Pfeil (i,j) entnommen und von KOM die zugehörige Komponente Q. Befindet sich in Q

kein Knoten, in dem ein Pfeil aus $\overline{E}$ endet, so wird dieser Pfeil $\overline{E}$ hinzugefügt, ansonsten werden Q und (i,j) verworfen. Das Verfahren bricht ab, sobald $\left|\overline{E}\right| = n - 1$ ist, dies ist spätestens nach $\left|PE\right|$ Schritten der Fall.

3.3 Baumalgorithmen

Es gibt mehrere Algorithmen, die es ermöglichen einen minimalen Wurzelbaum zu bilden und diesen dann auch zu speichern.
Ausgehend von einem bewerteten Digraphen G der Form G=(V,E,c) mit n Knoten und Startknoten a werden in einem Array D der Länge n die Entfernungen vom Startknoten für jeden einzelnen Knoten gespeichert. In einem zusätzlichen Routenfeld R der Länge n wird an der Stelle R[j] der unmittelbare Vorgänger des aktuellen Knoten j gespeichert, über den diese kürzeste Verbindung der Länge D[j] erreicht worden ist. Durch dieses Verfahren lässt sich anschließend einfach rekursiv ermitteln, wie ein beliebiger Punkt des Graphen vom Startpunkt aus am kürzesten erreicht werden kann.
Alle wichtigen Baumalgorithmen haben folgende gemeinsame Eigenschaften:

- Iterativ
- In jedem Schritt des Algorithmus sind Knoten markiert. Ein Knoten wird dann markiert, wenn sich der Weg zwischen ihm und dem Wurzelknoten reduzieren lässt (Es also für diesen Knoten ein kleines D[i] gibt).
- In jeder Iteration des Algorithmus werden 1 oder mehrere markierte Knoten überprüft, hierbei wird für alle Nachfolger j von i überprüft, ob es einen kürzeren Weg zwischen a und i gibt, der über einen der Nachfolger geht, also, ob es ein $D[j]+c_{ij} < D[i]$ gibt. Wurde der Knoten überprüft, so wird die Markierung gelöscht.
- Am Anfang ist D[a]=0, alle anderen $D[i] = \infty$, was dadurch zu begründen ist, das anfangs nur bekannt ist, wie groß die Entfernung von a zu sich selber ist, alle anderen Entfernungen müssen noch berechnet werden.

Unterschiede liegen hauptsächlich in der Auswahl der Markierung von Knoten
Es gibt zwei unterschiedliche Verfahren zur Verfahren der Markierung:
Label-Setting-Methoden: Bei einem einmal zur Überprüfung markierten Knoten ist im Allgemeinen die kürzeste Entfernung zum Punkt a bereites bekannt und diese wird nicht mehr verändert.
Grundvorgehen ist bei dieser Methode des Graphendurchlaufes, dass am Anfang nur a markiert ist, im 1. Durchlauf dann alle N(a) markiert werden und in jedem weiteren Schritt alle weiteren Nachbarn, die noch nie markiert wurden, markiert werden, bis alle Knoten einmal markiert worden sind.
Label-Correcting-Methoden: ein einmal markierter und überprüfter Knoten kann wiederholt markiert werden, die bekannte bisher geringste Entfernung des markierten Knoten zum Knoten a spielt hierbei keine Rolle.

3.3.1 FIFO-Kürzeste Wege

Bei dieser Realisierung zur Berechnung von Routing-Tabellen werden alle markierten Knoten in einer Schlange gespeichert, neu markiert Knoten werden

hinten eingefügt und das jeweils zu überprüfende Element wird vorne herausgeholt. Anfänglich besteht die Schlange nur aus dem Startknoten a.
Das erste Element i wird aus der Schlange geholt und für alle Nachbarn j von i wird bestimmt, ob $D[i]+c_{ij}$ kleiner ist als D[j], wenn dies der Fall ist, wird die Entfernungstabelle und die Routing-Tabelle entsprechend aktualisiert, also die neue, kürzere Entfernung in D und der neue Knoten zur kürzeren Erreichung in R eingetragen. Ist der Nachbar nicht markiert in der Schlange, wird er hinten in die Schlangen eingefügt, war er schon mal in der Schlange ist es effizienter ihn als zweites Element einzufügen[1].
Übertragen wir diesen Algorithmus auf das Routing im Internet, so kann leicht von jedem Knoten i ein Datenpaket an a geschickt werden. Ein Knoten schaut in einfach in der Routing-Tabelle an der Stelle R[i] nach, an wen er das Paket schicken muss, und tut das, der Knoten j, der das Paket erhält, schaut einfach in der Routing-Tabelle an der Stelle R[j] nach, an wen er das Paket weiterleiten muss und tut dies dann auch.

3.3.2 Weitere Algorithmen

Es gibt die Möglichkeit das Ziel des obigen Verfahrens auch durch andere Algorithmen z.B. basierend auf dem Algorithmus von Dijkstra zu ermitteln, aber da diese Verfahren sehr ähnlich sind, soll hier nur Dijkstra eben kurz erwähnt werden. Während der FIFO-Algorithmus zu den Label-Correcting-Methoden gehört, gehört ein entsprechender Algorithmus nach Dijkstra zu den Label-Setting-Methoden. Welche Methode effizienter ist, hängt von der jeweiligen Implementierung ab, bzw. von der Größe des Graphen, sowie des Verhältnises der Anzahl der Kanten zu der Anzahl der Knoten ab.

4 All Pairs Shortest Path

Um die kürzesten Entfernungen zwischen allen Knoten erhalten, kann man natürlich die Verfahren des „single source shortest path“-Problems für alle Knoten wiederholen. Dies ist sogar üblich. Allerdings haben dies beiden oben vorgestellten Algorithmen maximale Laufzeiten von $O(n^2)$, bzw. $O(n^3)$, bei n-facher Wiederholung liegt die Laufzeit dementsprechend bei $O(n^3)$ und $O(n^4)$
Eine alternative Lösung dieses Problems stellen hingegen die so genannten Matrix-Algorithmen dar, von denen exemplarisch der Tripel-Algoritmus dargestellt werden soll. Dieser hat eine Laufzeit von $\Theta(n^3)$.

4.1 Der Tripel-Algorithmus

Zum Graphen G=(V,E), kommt noch eine Kostenmatrix $C(G)=(c_{ij})$, sowie eine Vorgängermatrix VG(G), die am Ende des Verfahrens die Routingtabelle enthalten soll. C(G) wird hierbei im Laufe des Verfahrens um die Kosten einer indirekten Erreichung des Knotens j von i aus erweitert.
Bei der Durchführung des Verfahren werden Tripel (i,j,k) aus den 3 Werten Starknoten, Endknoten und Kosten der Entfernung dahin gehend überprüft, ob es

[1] Erweiterung von D'Esopo, der eigentliche FIFO-Algortihmus, fügt Konten immer am Ende ein.

von i nach j einen noch kürzeren Weg gibt, als bisher bekannt. Wenn das der Fall ist, werden die Informationen entsprechend geändert.

```
for (int j=1;j<n, j++)
{
 for (int i=1;i<n, i++)
 {
  for (int k=1;k<n, k++)
  {
   su=c[i][j]+ c[j][k];
   if (su<c[i][k] && i!=j && k!=j)
   {
    c[i][k]=su;
    vg[i][k]=vg[j][k];
   }
  }
 }
}
```

Es werden zunächst alle Knoten durchgegangen, diese Knoten sind die Startknoten. Für jeden dieser Knoten werden nacheinander alle anderen Knoten als Endknoten ausgewählt. Für jede dieser Strecken wird überprüft, ob diese Strecke länger ist, als über einen Zwischenknoten. Als dieser Zwischenknoten werden nacheinander alle Knoten ausgewählt, die nicht der Start- oder Endknoten sind. Ist eine Strecke über einen dieser Zwischenknoten (die Strecken über einen Zwischenknoten kann natürlich noch weiter Zwischenknoten enthalten, dies ist sogar normal) kürzer als der bekannte, wird die Routing-Tablle und die Kostenmatrix entsprechend geändert.

5 Entfernung zwischen zwei Punkten

Manchmal möchte man nur die minimale Entfernung und den kürzesten Weg zwischen einem Anfangs- und einem Endpunkt wissen. Um diese Aufgabe zu lösen bedient man sich der Bellmanschen Optimalitätsgleichung. Diese besagt, dass eine Lösung des Gesamtproblems sich aus Lösungen von Teilproblemen zusammensetzt. Was nicht anderes bedeutet, als das auf dem Weg ein Zwischenpunkte gesetzt wird und der minimale Weg zwischen dem Anfangs- und dem Zwischenpunkt, sowie dem Zwischen- und dem Endpunkt gefunden werden sollen, wobei die Länge dieser Strecke wieder über die Wähl eines Zwischenpunktes berechnet wird. Als Zwischenpunkte werden natürlich nacheinander alle anderen Knoten genommen. Wie man sieht ist diese Lösung dem Tripel-Algorithmus sehr ähnlich.

6 Routing auf EDV-Netzwerken

Das Routing auf EDV-Netzwerken bereitet einige Probleme, denn die angeschlossenen Stationen können nicht über alle Wege zu allen Stationen verfügen, da immer nur ein Teil der Stationen für eine einzelne Station wichtig ist. So greift z.B. ein Rechner eines normalen Internetnutzers immer nur auf einen sehr kleinen Teil der anderen Stationen zu, die Anzahl und die Orte der angeschlossenen Stationen ändern sich ständig. Es ist nicht immer eine richtige Bewertung der Verbindungen gewährleistet. Leitungen können ausfallen, bzw. verstopft sein. Es gibt sehr viele parallele Kanten/Verbindungen. Würden jeweils alle Stationen immer aktualisiert werden, so wäre dies ein immenser Datentransfer und jeder Rechner müsste ständig seine komplette Routing-Tabelle aktualisieren, was auch zu einem erhöhten Rechenaufwand auf dem Rechner führt. Wobei das genaue Routing auch uninteressant ist, da die meisten Rechner

nur eine geringe Anzahl von Anschlüssen verfügt, die auch zumeist in völlig unterschiedliche Richtungen führen. Für ein Datenpaket, das von Europa nach Amerika geschickt werden soll, ist es in erster Linie einmal interessant, wie es von einem Punkt in Europa zu einem Punkt in Europa kommt, von dem es nach Amerika geschickt werden kann. Zwischen Amerika und Europa bestehen im Vergleich zu Verbindungen in Europa vergleichsweise wenige Verbindungen. Und in Amerika ist es erst interessant, wie das Paket wirklich seinen Empfänger erreicht. Erschwerend kommt hinzu, dass Netzwerkleitungen immer eine beschränkte Kapazität oder Datendurchfluss haben. Es ist nicht auszuschließen, dass einmal mehr Daten über einen Strang des Netzwerkes geschickt werden sollen, als wirklich auf die Leitung passen. Dies nennt man dann einen Flaschenhals. Es ist dann an dieser Stelle notwendig, dass Pakete gepuffert werden und zeitversetzt weiter geschickt werden.
Ein Algorithmus, der gewährleistet, dass die Routing-Tabelle eines Routers wirklich aktuell ist, ist der Netchange-Algorithmus, der im folgendem vorgestellt werden soll.

6.1 Der Netchange-Algorithmus

Tajibnapis Netchange – Algorithmus berechnet Routing-Tabellen, die gemäss dem „minimum – hop“ Maß optimal sind. Er basiert auf folgenden Annahmen:

- Die Knoten kennen die Größe des Netzes (N).
- Die Kanäle erfüllen die FIFO-Eigenschaft.
- Knoten werden über Ausfälle und Reparaturen ihrer inzidenten Kanäle (die sie mit ihren Nachbarn verbinden) benachrichtigt.
- Die Kosten eines Weges sind gleich der Anzahl der Kanäle in dem Weg.

Ausfall und Reparatur von Knoten wird nicht betrachtet: es wird stattdessen angenommen, dass der Ausfall eines Knotens von seinem Nachbarn als Ausfall des verbindenden Kanals beobachtet wird.
Der Algorithmus führt in jedem Knoten u eine Tabelle $NB_u[v]$, die zu jedem Ziel v einen Nachbar von u enthält, zu dem Pakete für v weitergeleitet werden.
Es kann nicht gefordert werden, dass die Berechnung dieser Tabellen innerhalb einer endlichen Anzahl von Schritten in allen Fällen terminiert, da wiederholte Ausfälle und Reparaturen von Kanälen erneute Berechnungen auf unbestimmte Zeit nach sich ziehen.
Die Auswahl eines Nachbars, zu dem Pakete für das Ziel v weitergeleitet werden, basiert auf Schätzungen der Distanz von jedem Knoten zu v. Der bevorzugte Nachbar ist immer der Nachbar mit dem niedrigsten Schätzwert für diese Distanz.

```
Netchange Algorithmus für den Knoten u:

var  Neigh:  set of Nodes;    (*die Nachbarn von u*)
     C:      array of 0…n     (*C[v] enthält geschätztes c(u,v)*)
     Nb:     array of Nodes;  (*Nb[v] ist der anzusprechende Nachbar für v*)
     ndis:   array of o…n;    (*ndis [w,v] enthält geschätztes c(w,v)*)
```

```
Initialisation:
Begin
    forall w ? Neigh, v ? V do ndis[w,v]:=n;
    forall v ? V do
        begin
        D[v]=N;
        NB[v]:= 8
        End;
    D[u]=0;
    Nb[u]=local;
    Forall w ? Neigh do send <mydist, u, 0> to w
End

Procedure Recompute
Begin
    If v=u
        Then
            Begin
            D[v]:=0;
            Nb[v]=local;
        End;
        Else
            Begin  (*Abschätzen der Entfernung zu v*)
            D=1+ min{ndis[w,v]: w ? Neigh};
            If d<N
                then
                    Begin
                    D[v]=d; Nb[v]=w with 1+ndis[w,v]=d
                    End
                Else
                    Begin
                    D[v]=n;
                    Nb[v] = 8
                    End
            End
    If D[w] was changed then
        Forall x ? Neigh do send <mydist, v, D[v]> to x
End

(* Bearbeiten einer <mydist, u, 0> Nachricht vom Nachbar w: *)
Begin
    Recieve <mydist, v, d> from w;
    ndis [w,v]=d;
    Recompute(v);
End

(*Beim Ausfall der Verbindung uw:*)
Begin
    Receive <fail,w>; Neigh=Neigh\{w};
    Forall v ? V do Recompute (v);
End

(* Bei Reparatur der Verbindung w *)
Begin
    Recieve <reapair, w>;    Neigh=Neigh ? {w};
    Forall v ? V do
        Begin
        ndis[w,v]=n;
        send <mydist, v, D[v]> to w
    End
End
```

7 Literatur

Ingo Wegener (1999), Theoretische Informatik – eine algorithmische Einführung, 2. Auflage, Teubner

Wolfgang Domschke (1981), Logistik: Transport, Oldenbourg

Wolfgang Domschke (1997), Logistik: Rundreisen und Touren, 4. Auflage, Oldenbourg

Susanne Albers (2000), Grundvorlesung Datenstrukturen WS 2000/2001, Skript zu einer Vorlesung an der Uni Dortmund

Andreas Jakoby (1999), Paralle Algorithmen, Kapitel 2: Packet-Routing auf Array-Netzwerken, Skript zu einer Vorlesung an der Uni Lübeck

O. Drobnik (2001), Verteilte Systeme und Telematik I, Kapitel 4: Routing - Algorithmen: Grundlagen, Vorlesungsfolien zu einer Vorlesung an der Uni Frankfurt

Peter Merz (2003), Moderne Heuristische Optimierungsverfahren, Kapitel 1: Optimierungsprobleme, Skript zu einer Vorlesung an der Uni Tübingen

Knud Bielefeld (1993), Lösung von Rundreisproblemen mit engen Zeitschranken, Diplomarbeit im Studiengang Wirtschaftsinformatik an der Fachhochschule Flensburg.