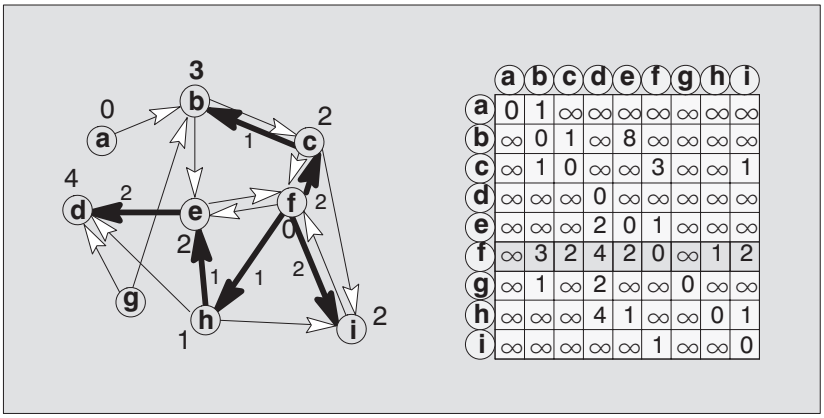
INSTITUT FÜR VERKEHRSWIRTSCHAFT, STRASSENWESEN UND STÄDTEBAU

Univ.–Prof. Dr.–Ing. Robert Schnüll

UNIVERSITÄT HANNOVER

Studienarbeit

Kritische Analyse von Routensuchverfahren



Martin Rose

Matr.–Nr. 1463002

Betreuung: Dr.–Ing. Hans–Martin Heck

Erstprüfer: Univ.–Prof. Dr.–Ing. Robert Schnüll
Zweitprüfer: Dr.–Ing. Hans–Martin Heck

Inhalt

1. Einleitung 1

1.1. Aufgaben der Routensuchverfahren im Verkehrswesen 1

1.2. Anforderungen an Routensuchverfahren im Verkehrswesen 3

1.3. Zur Analyse bestehender Routensuchverfahren 3

2. Grundlagen der Graphentheorie 5

2.1. Graphen 5

2.2. Datenstrukturen für Graphen 9

2.2.1. Kantenlisten 9

2.2.2. Adjazenz– und Inzidenzmatrizen 9

2.2.3. Adjazenzlisten 10

3. Abstraktion eines Verkehrsnetzes auf einen Graph 11

4. Grundlegende Routensuchverfahren 13

4.1. Überblick 13

4.1.1. Einteilung der Routensuchverfahren nach der Startknotenanzahl 14

4.1.2. Aufgabenstellung zur Bestimmung kürzester Wege 15

4.1.3. Entwicklungsgeschichte der Routensuchverfahren 16

4.2. Darstellung und Analyse ausgewählter Routensuchverfahren 22

4.2.1. Ungewichtete Graphen 22

4.2.1.1. Breitensuche nach Moore 22

4.2.2. Kürzeste Wege von einem Knoten aus 26

4.2.2.1. Algorithmus von Moore für kürzeste Wege von einem Knoten aus 26

4.2.2.2. Algorithmus von Dijkstra für kürzeste Wege von einem Knoten aus 31

4.2.3. Kürzeste Wege zwischen je zwei Knoten 35

4.2.3.1. Algorithmus von Floyd für kürzeste Wege zwischen je zwei Knoten 35

4.2.3.2. Wegealgebren 36

5. Weiterentwicklung der Routensuchverfahren 38
5.1. Verbesserung der Minimumsuche in Baumalgorithmen 38
5.2. Verbesserung der Routensuche durch Parallelisierung 40
5.3. Fazit bezüglich der Effizienzanforderung an die Routensuche 40

6. Zum Problem der Modellierung von Verkehrsnetzen 41

7. Literaturverzeichnis 44
7.1. Routensuchverfahren im Verkehrswesen 44
7.2. Graphentheorie 44
7.3. Allgemeine Probleme mit Routen 45
7.4. Kürzeste Wege von einem Knoten aus 45
7.5. Kürzeste Wege zwischen je zwei Knoten 45
7.6. Effizienzsteigerung der klassischen Routensuchverfahren 46

1. Einleitung

Routensuchverfahren können zu den wichtigsten grundlegenden Algorithmen in den Modellen des Verkehrswesens gezählt werden. Mit ihrer Hilfe werden optimale Wege für den Wechsel von Verkehrsteilnehmern und –gütern zwischen Orten innerhalb eines Verkehrsnetzes bezüglich eines vorgegebenen Kriteriums ermittelt. Solche Ortswechsel bilden die Basis des Verkehrs und somit einer Vielzahl von Verkehrsmodellen.

Im Verkehrswesen kommen Routensuchverfahren meist als ein wesentlicher Teil von Verkehrsmodellen zur Umlegung der Verkehrsnachfrage auf Verkehrsnetze vor. Sie können aber auch für sich allein auftreten, wie beispielsweise bei Leitsystemen, die in Kraftfahrzeuge integriert sind, um den Fahrer auf dem günstigsten Weg durch ein Verkehrsnetz zu einem gewünschten Ziel zu führen.

Die Entwicklung effektiver Routensuchverfahren vor allem für große Netze wird zum einen durch die Entwicklung der Computer mit einer immer schnelleren Verarbeitung großer Datenmengen und zum anderen durch die mathematischen Grundlagen der Graphentheorie bestimmt.

In der Graphentheorie werden Routensuchverfahren ”Algorithmen zur Bestimmung kürzester Wege” genannt. Sie zeichnen sich durch ihre vielseitige Anwendbarkeit in beliebigen Bereichen aus, in denen das topologische Problem der Suche optimaler Wege auftritt. So lassen sich beispielsweise in Kommunikationsnetzen die bestmöglichen nicht direkten Schaltungen von Ferngesprächen über mehrere Städte ermitteln, die effektivste Nutzung eines Roboters zum Löten von Platinen herleiten oder der minimale Verbrauch an Kabeln für die Versorgung eines Hochhauses bestimmen. Für den effizienten Einsatz von Parallelrechnern und für schnelles Editieren in Graphiksystemen bilden Kürzeste–Wege–Algorithmen eine Grundlage. In der Graphentheorie treten die Algorithmen zur Bestimmung der kürzesten Wege oft auch als Teil eines anderen Verfahrens auf. Als praktisches Beispiel sei hier die optimale Beladung eines Containerseeschiffes erwähnt, das auf einer möglichst kurzen Rundreise seine Fracht in einer Vielzahl von Häfen unterschiedlicher Länder möglichst schnell entladen soll.

1.1. Aufgaben der Routensuchverfahren im Verkehrswesen

Im Verkehrswesen werden Routensuchen traditionell innerhalb von statischen Verkehrsumlegungsmodellen verwendet. Üblicherweise werden hiermit für langfristige Verkehrsplanungen eines Planungsraums die Verkehrsstärken der Verkehrsnetze unterschiedlicher Planfälle berechnet. Da diese Berechnungen für zukünftige Verkehrsnachfragen erfolgen, sollte die Umlegung eine möglichst zuverlässige, realistische Beschreibung des Wegewahlverhaltens der Verkehrsteilnehmer bieten.

Neuere Verkehrsumlegungsmodelle für längerfristige Verkehrsplanungen sind dynamische Modelle, die die Abhängigkeit von den zeitlich bereits vorhandenen Verkehrsbelastungen mitberücksichtigen, um so eine realistischere Beschreibung der Verkehrsstärke zu erreichen. Da der Verkehr und das Verhalten der Verkehrsteilnehmer einen Prozeß darstellt, scheint die notwendige dynamische Modellierung den entstehenden hohen Aufwand zu rechtfertigten. Ein Beispiel für eine solches Verfahren ist das Modellkon-

zept DRUM [Se94] zur dynamischen Routensuche und Umlegung, bei dem neben den zeitinvarianten Netzwiderständen auch die unterschiedliche Netzkenntnis und Schätzfähigkeit der Verkehrsteilnehmer unterschiedlicher verhaltenshomogener Personengruppen berücksichtigt wird.

Ein sehr wichtiger Aspekt bei der Abbildung des Verkehrs und der Wegewahl der Verkehrsteilnehmer ist das Zusammenwirken aller Verkehrsmittel im Straßenverkehr. So kann die bisher übliche Umlegung des Verkehrs für jeweils nur ein Verkehrsmittel (meist motorisierter Individualverkehr) auf das Verkehrsnetz nicht sehr realistisch sein. Die Aufteilung der Verkehrsmittel durch den Modal−Split hat nicht nur zur Folge, daß die Verkehrsströme für ausschließlich ein Verkehrsmittel sich im Modell nicht gegenseitig beeinflussen können, sondern auch, daß das Umsteigen eines Verkehrsteilnehmers von einem Verkehrsmittel in das andere nicht darstellbar ist. Daher ist eine Aufhebung des unnatürlichen Modal−Splits in den Umlegungsmodellen notwendig. Inwieweit dies Auswirkungen auf die Routensuche hat, wird unter anderem im letzten Kapitel über Probleme bei der Modellierung von Verkehrsnetzen diskutiert.

Seit einigen Jahren wird der Straßenverkehr immer mehr durch ”intelligente” verkehrsabhängige Beeinflussungssysteme gesteuert und gelenkt. Dabei hat sich gezeigt, wie wichtig effiziente Routensuchverfahren auch in diesen Bereichen des Verkehrswesens sind. So gibt es beispielsweise Leitsysteme für Fahrzeuge, die während der Fahrt den Verkehrsteilnehmern abhängig vom aktuellen Verkehrszustand im Verkehrsnetz auf dem kürzesten Weg zu einem gewünschten Ziel führen sollen. Hierzu ist eine ständige kurzfristige Berechnung des kürzesten Weges unabdingbar.

Auch bei verkehrsabhängigen Beeinflussungssystemen, die durch stationäre Anlagen an der Strecke den Verkehr steuern und lenken, sollten auf Routensuchverfahren nicht verzichtet werden. So hat unter anderem Ploss in [Pl93] eindrucksvoll gezeigt, daß verkehrsabhängige Beeinflussungssysteme prinzipiell eine ständige Nachfrage− und Umlegungsberechnung aus den aktuell ermittelten Verkehrsdaten benötigen, um die notwendigen globalen Auswirkungen einer (lokalen) Beeinflussung abschätzen zu können. Da all diese Verkehrsbeeinflussungssysteme eine kurzfristige Routensuche benötigen, sollten die verkehrsabhängigen Routensuchverfahren möglichst schnell sein.

Die Installationen von verkehrsabhängigen Beeinflussungssystemen sind sehr aufwendig und kostenintensiv. Daher sollte vor der Einrichtung solcher Systeme ihre Wirkungsweise auf den Verkehr in einem Simulationsmodell untersucht werden. Simulationsmodelle sind meist dynamische mikroskopische Modelle, die den Verkehrsablauf durch Beschreibung des Verhaltens von Einzelfahrzeugen darstellen. Ein simuliertes Einzelfahrzeug oder ein Fahrzeugpulk muß an einem beliebigen Startpunkt im Modell erzeugt und möglichst realistisch durch das Verkehrsnetz zu einem Ziel geführt werden. Aus diesem Grund ist innerhalb der Simulation für jede Fahrt eines modellierten Fahrzeugs oder Fahrzeugpulks mindestens einmal eine Routensuche durchzuführen. Ebenso wie dynamische Modelle zur Verkehrsumlegung sind auch Simulationsmodelle für Verkehrsabläufe meist sehr rechenaufwendig. Da die Routensuche einen wesentlichen Anteil an der Gesamtrechenzeit der Modelle besitzt, sollte ihre Rechenzeit minimiert werden.

1.2. Anforderungen an Routensuchverfahren im Verkehrswesen

Aus den Aufgaben von Routensuchverfahren im Verkehrswesen, die im letzten Abschnitt 1.1 beschrieben wurden, ergeben sich zwei grundsätzliche Anforderungen an diese Verfahren.

Die erste grundsätzliche Anforderung an die Routensuchverfahren im Verkehrswesen ist nicht verkehrsspezifisch, sondern ein allgemeines Ziel für die Anwendung des Routensuchalgorithmus auf große Netze: Es ist die schnellstmögliche Rechenzeit zur Lösung des Kürzeste–Wege–Problems.

In Anbetracht immer leistungsfähigerer Rechner scheint eine solche Forderung zwar nicht relevant zu sein, in dieser Arbeit sollte jedoch klar werden, daß die Rechenzeit bei großen Netzen in erster Linie gerade nicht von der Geschwindigkeit des Rechners abhängt, sondern von der Zeitkomplexität des Rechenverfahrens. Die geschichtliche Entwicklung der Kürzeste–Wege–Algorithmen in der Graphentheorie wird daher auch ausschließlich durch die Verringerung der Rechenzeit bestimmt.

Die zweite grundsätzliche Anforderung an die Routensuche ist bei genauer Betrachtung keine Anforderung an das Routensuchverfahren, sondern an das Umlegungsmodell, in das das Routensuchverfahren als Berechnungskomponente zur Umlegung der Verkehrsnachfrage auf ein Verkehrsnetz integriert ist: Es ist die möglichst zuverlässige und realistische Modellierung des Wegewahlverhaltens der Verkehrsteilnehmer.

Dieses Hauptziel der Verkehrsumlegung bestimmt die Kriterien der Wegewahl, jedoch nicht die Routensuche an sich. Die Routensuche ermittelt einen optimalen Weg gemäß eines Widerstandes der Strecken zwischen dem Start– und Zielort des gesuchten Weges. Dieser Widerstand richtet sich nach den Kriterien der Wegewahl, wie etwa Fahrzeit, Weglänge, Umsteigemöglichkeiten oder Kosten. Auch die meisten Alternativroutenverfahren stellen lediglich eine bestimmte Veränderung der Widerstände dar. Je nachdem, wie realistisch das Verkehrsverhalten beschrieben werden soll, können auch sehr komplexe Kriterien entwickelt werden. Die Ermittlung geeigneter Kriterien ist Aufgabe des Umlegungsmodells, nicht aber der Routensuche, die lediglich nach bestimmten Kriterien, die vorher ermittelt wurden, optimale Wege berechnet.

1.3. Zur Analyse bestehender Routensuchverfahren

Die Idee der Abstraktion eines Verkehrsnetzes auf einen Graph hat unter anderem zu einer effizienten Lösungsmöglichkeit für die Suche optimaler Wege im Netz geführt. Die Graphentheorie beschreibt mit einem Graph die allgemeingültigen topologischen Zusammenhänge einer beliebig strukturierten Menge. In der Allgemeingültigkeit liegt ihre Stärke, da die Graphentheorie so Lösungen in Form von möglichst optimalen Algorithmen für sehr viele Bereiche, wie etwa dem Verkehrswesen oder der Elektrotechnik zur Verfügung stehen kann. Ziel der vorliegenden Arbeit soll es sein, ausgehend von den klassischen Verfahren zur Bestimmung kürzester Wege in Graphen den aktuellen Stand der Routensuchverfahren zu analysieren. Dabei kann in erster Linie nur die Untersuchung der Effizienz dieser Graphenalgorithmen im Vordergrund stehen.

Die spezielle Anwendung der Kürzesten–Wege–Algorithmen auf Probleme des Verkehrswesens führt zu keinen anderen Lösungen als in der Graphentheorie, da das Verkehrsnetz gerade vorher auf einen Graph abstrahiert wurde. Die Routensuche wird kaum von Problemen, wie der Darstellung mehrerer Verkehrsmittel in einem Verkehrsnetz oder die Beschreibung realistischer Wegewahlverhalten der Verkehrsteilnehmer betroffen. Nur über die Krirterien, nach denen der Graphenalgorithmus die optimalen Wege suchen soll, ist ein Einfluß solcher Probleme auf die Routensuche darstellbar. Die Bestimmung der Kriterien für die Widerstände innerhalb der Routensuche geschieht im Verkehrswesen meist in den Umlegungsmodellen bevor die Routensuche durchgeführt wird. Sie ist daher *nicht* explizit Bestandteil der Analyse der Routensuchverfahren.

Es wird sich in dieser Arbeit zeigen, daß die Algorithmen zur Bestimmung der kürzesten Wege in einem Graph, also die Routensuchverfahren, mittlerweile als optimal angesehen werden können. Dies trifft zumindest für die Bestwegverfahren zu, wo nur die optimalen Wege gesucht werden. Alternativroutenverfahren, die auch schlechtere Wege untersuchen, werden in dieser Arbeit nicht untersucht, zumal sie auf den Bestwegverfahren basieren.

Die derzeit und künftig zu lösenden Probleme im Verkehrswesen sind nicht vorrangig die Entwicklung effizienter Algorithmen für abstrakte Datenstrukturen (dies ist Hauptaufgabe der Informatiker), sondern eine möglichst gute Modellierung realer Verkehrsprobleme. Diese Aufgabe der sinnvollen Abstraktion eines praktischen Verkehrsproblems in eine korrekte Modellform, wie beispielsweise die Abbildung eines realistischen Wegewahlverhaltens von Verkehrsteilnehmern, kann dem Verkehrsingenieur, mit seiner Ausbildung und Erfahrung in Fragen des Verkehrs, nicht abgenommen werden.

Wegen der Wichtigkeit der Modellierung von Verkehrsproblemen und deren Auswirkungen zum Beispiel auf den Verlauf der Routensuche wird im letzten Kapitel dieser Arbeit das Modellierungsproblem diskutiert. Zuvor erfolgt die Analyse der Routensuchverfahren. So ist die Arbeit in die folgenden Kapitel unterteilt:

- Kapitel 2 beschreibt die Grundlagen der Graphentheorie, soweit sie zum Verständnis bei der Analyse der Kürzeste–Wege–Algorithmen benötigt werden.
- Kapitel 3 erläutert kurz die Abstraktion eines Verkehrsnetzes auf einen Graph, um zu zeigen, auf welche Weise der in Kapitel 2 definierte Graph für Verkehrszwecke verwendet werden kann und in welcher Abstraktionsform das Verkehrsnetz hierzu abgebildet werden muß.
- In Kapitel 4 werden dann die klassischen Routensuchverfahren erläutert. Ausgehend von der geschichtlichen Entwicklung sind vor allem die Grundmethoden und Unterschiede der wichtigsten Kürzeste–Wege–Algorithmen dargestellt.
- Kapitel 5 zeigt schließlich die Entwicklung der Effizienzsteigerungen der Algorithmen zur Bestimmung kürzester Wege in Graphen auf.
- Kapitel 6 beschäftigt sich mit generellen Modellierungsproblemen im Verkehrswesen und geht dabei im besonderen auf den Einsatz von objektorientierten Methoden ein. Es wird diskutiert, inwieweit die Routensuche durch neue Modellierungen des Verkehrsnetzes beeinträchtigt wird.

2. Grundlagen der Graphentheorie

Die Graphentheorie ist ein Teilgebiet der kombinatorischen Mathematik. Obwohl ihre Anfänge schon über 250 Jahren zurückliegen, hat das Interesse an der Graphentheorie erst in den fünfziger Jahren in der Mathematik und vor allem im Transport− und Verkehrswesen durch die dort auftretenden topologischen Probleme, wie das der Bestimmung kürzester Wege, stark zugenommen. Es gibt seitdem ständig neue Anwendungen und Veröffentlichungen zu diesem Thema. In diesem Kapitel können und sollen nur die Grundlagen, die für Probleme der Routensuche wichtig sind, kurz erläutert werden. Für weitergehende Informationen zu Graphen sei auf die zahlreiche Grundlagenliteratur, wie [Ah74], [Cl94], [Br94] oder [Ju94] verwiesen.

2.1. Graphen

Ein **Graph** ist eine Menge aus wohlunterscheidbaren Elementen mit gleichen Eigenschaften, der eine Struktur aufgeprägt wird. Die aufgeprägte Struktur wird durch eine Relation zwischen den Elementen der Menge beschrieben. Eine Folge, ein Baum oder auch eine unstrukturierte Menge sind Sonderfälle eines Graphen.

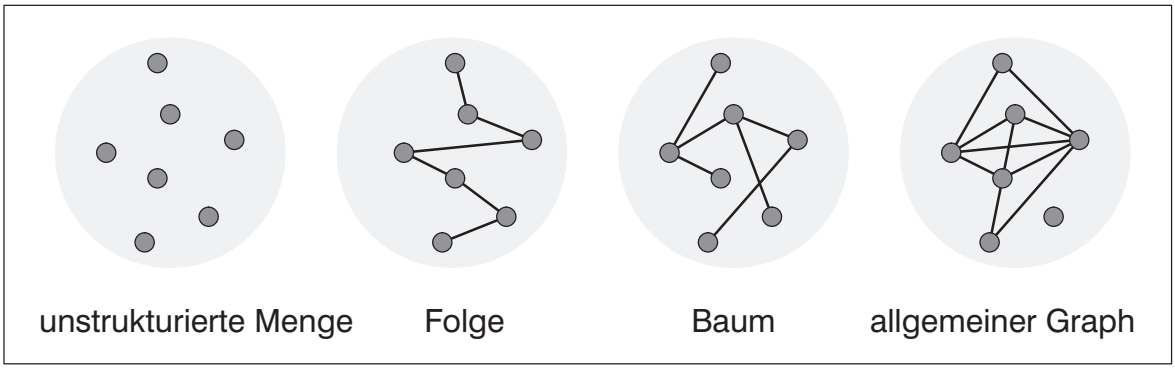


Bild 1: Unterschiedliche Ausprägungen eines Graphen

Definition: Ein Graph G besteht aus einer Menge V und einer Relation E in V.

$$G := (V;E) \qquad mit \quad E \subseteq V \times V \quad (1)$$

Die Elemente der Menge V heißen **Knoten** (englisch "node" oder "vertex"). Einem Knoten können Eigenschaften und Methoden zugeordnet werden. So können beispielsweise den Stationen eines Stadtbahnnetzes als Knoten eines Graphen der Stationsname und Eigenschaften, wie etwa Größe oder Stationsart zugeordnet werden. Die Anzahl der Knoten wird mit $|V|$ angeben.

Ein Element der Relation E, also ein geordnetes Paar zweier Knoten, heißt **Kante** (englisch "edge"). Die Kante $e = (u,v) \neq (v,u)$ hat den Anfangsknoten u und den Endknoten v. u und v sind mit e **inzident** und v ist **adjazent** (benachbart) zu u. Die Bezeichnung $|E|$ gibt die Anzahl der Kanten in E an.

Da Graphen unabhängig von einer geometrischen Anordnung sind, lassen sie sich auf vielfache Weise darstellen. Meistens wird eine Darstellung gewählt, in der die Knoten als Kreise und die Kanten als Strecken mit Pfeilen gekennzeichnet sind. Häufig benutzt man auch die sogenannte Adjazenzmatrix. Dies ist eine boolsche $(n \times n)-$Matrix $A=(a_{ij})$. Die Einträge a_{ij} beschreiben die Kanten von Knoten i zu Knoten j.

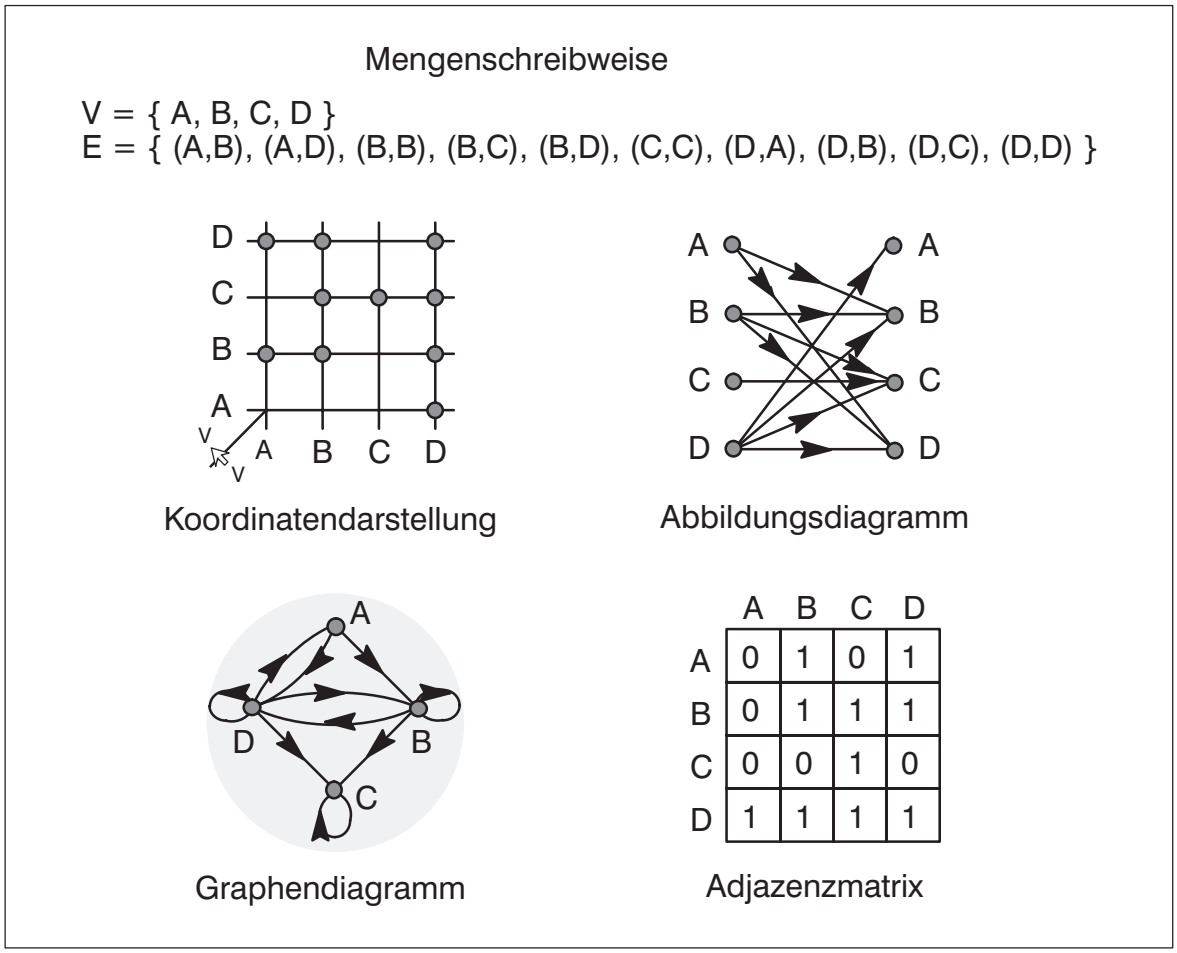


Bild 2: Fünf Darstellungsformen eines Graphen

In der Literatur zur Graphentheorie taucht sehr oft die Unterscheidung von gerichteten und ungerichteten Kanten auf. Da Kanten zumindest nach Definition (1) geordnete Paare sind, haben sie zwangläufig eine Richtung und demnach gibt es keine ungerichteten Kanten. Ungerichtete Graphen sind symetrische Graphen; das heißt, wenn es in der Relation E eines Graphen $G=(V;E)$ eine Kante (u,v) vom Knoten u zum Knoten v gibt, so existiert in E auch die Kante (v,u).

In vielen Anwendungen werden zwei unterschiedliche Mengen benötigt, um ein Problem zu beschreiben. So könnte beispielweise der Wunsch bestehen, in einem Straßenverkehrsnetz neben der Zuordnung von Knotenpunktattributen zu den Knoten des Graphen auch die Strecken zwischen den Knotenpunkten detaillierter beschreiben zu wollen. Hierzu müssen die Kanten eines Graphen als eigenständige Objekte eingeführt werden, also als Elemente einer zweiten Menge (so sind auch ungerichtete Kanten möglich).

Verschiedene Mengen in einem Graph können zum einen als Teilmengen der Knoten des Graphen aufgefaßt werden, zum anderen in Form mehrerer Mengen, zwischen denen eine Relation besteht. Eine Struktur zwischen zwei Mengen ist durch einen **bipartiten Graph** darstellbar.

Definition: Ein bipartiter Graph G besteht aus zwei Mengen V_1 und V_2 und einer Vereinigung E der binären Relationen auf V_1 und V_2 sowie auf V_2 und V_1.

$$G := (V_1, V_2; E) \qquad mit \quad E \subseteq V_1 \times V_2 \cup V_2 \times V_1 \tag{2}$$

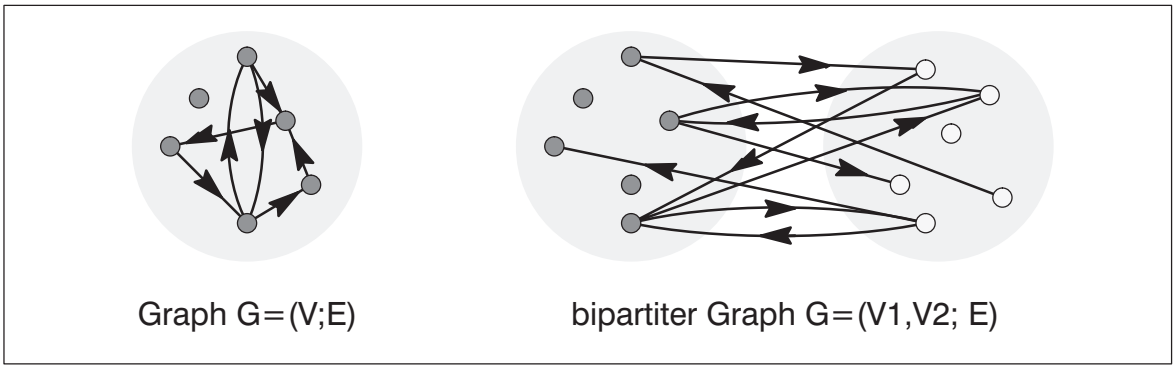


Bild 3: Graph und bipartiter Graph

Jeder Graph $G'=(V';E')$, dessen Knotenmenge V' eine Teilmenge von V eines Graphen $G=(V;E)$ ist, und dessen Relation E' genau die Kanten aus E enthält, von denen beide Endknoten in V' enthalten sind, heißt **Teilgraph** (englisch "subgraph").

Besitzt ein Graph nur Knoten aber keine Kanten ($E=\{\}$), so heißt der Graph **leer**, sind dagegen alle möglichen Kanten vorhanden ($E=V\times V$), so ist der Graphen ein **vollständiger** Graph. Graphen mit relativ wenig Kanten heißen **licht** und Graphen mit relativ vielen Kanten heißen **dicht**.

Für jeden Knoten v eines Graphen definiert man den **Grad** $d(v)$ (englisch "degree") von v als die Anzahl der mit v inzidenten Kanten. Er setzt sich aus einem Eingangsgrad (englisch "indegree") $d_{in}(v)$ und einem Ausgangsgrad (englisch "outdegree") $d_{out}(v)$ zusammen. Der Eingangsgrad ist als Anzahl der Kanten mit dem Endknoten v, und der Ausgangsgrad ist als die Anzahl der Kanten mit dem Anfangsknoten v definiert.

Eine Folge von $n+1$ Knoten $v_0, v_1, \ldots, v_n$ aus V heißt **Kantenzug**, wenn jeweils zwei hintereinanderliegende Knoten der Folge durch eine Kante $e_i = \{v_i, v_{i+1}\}$ für $i = 0, \ldots, n-1$ verbunden sind. Die Länge des Kantenzugs ist n. Ist der Anfangsknoten der Folge mit dem Endknoten der Folge identisch ($v_0 = v_n$), so spricht man von einem geschlossenen Kantenzug. Sind die Kanten eines Kantenzugs paarweise verschieden, so spricht man von einem **Weg** oder im geschlossenen Fall von einem **Zyklus**. Dabei ist darauf zu achten, daß die Folge der Knoten nur in Richtung der Kanten möglich ist. Wenn zusätzlich die Knoten des Weges bezeihungsweise des Zyklus paarweise verschieden sind, so ergibt sich ein einfacher Weg oder Pfad beziehungsweise ein einfacher Zyklus.

Zwei Knoten *u* und *v* eines Graphen *G* heißen verbindbar, wenn es einen Kantenzug von *u* nach *v* gibt. Sind je zwei Knoten von *G* verbindbar, so heißt *G* **zusammenhängend**. Ist der Graph nicht zusammenhängend, so besteht er aus mehreren Zusammenhangskomponenten, in denen jeweils wieder alle Knoten miteinander verbunden sind. Sind zwei Knoten *u* und *v* in derselben Zusammenhangskomponente, so gibt es offensichtlich einen Weg kürzester Länge zwischen *u* und *v*. Man sagt dann, *u* und *v* haben den **Abstand** *dist = dist(v,u)* (englisch "distance").

Ein Graph, der keine Zyklen hat, heißt **azyklisch**. Ein zusammenhängender azyklischer Graph, bei dem man von einem Knoten zu jedem anderen Knoten über genau einen Weg gelangt, heißt **Baum**. Ein Baum mit *n* Knoten hat *n−1* Kanten. Jeder Knoten eines Baumes mit dem Ausgangsgrad 0 heißt Blatt und mit dem Eingangsgrad 0 heißt Wurzel. Ein **Wald** ist ein Graph, dessen Zusammenhangskomponenten Bäume sind.

Ein **Spannbaum** ist ein Baum, der alle Knoten eines Graphen oder einer Zusammenhangskomponente enthält. Ein sehr wichtiges Thema in der Graphentheorie sind "minimal" zusammenhängende Graphen. Hierzu gehört beispielsweise ein Algorithmus zur Bestimmung eines minimalen Spannbaumes als eine Basis der Kürzesten−Wege−Suche.

Zur Suche der kürzesten Wege im Sinne der Routensuche sind Gewichtungen der Kanten notwendig, um etwa die Zeit oder Weglänge festzuhalten, die man für das "Durchqueren" der Kante benötigt. Graphen mit gewichteten Kanten werden **gewichtete Graphen** genannt. Die Gewichtung für einen Graph *G=(V;E)* erfolgt im allgemeinen durch eine Gewichtsfunktion $l: E \rightarrow \mathbb{R}$, die jeder Kante *e* einen reelen Wert *l(e)* zuordnet.

Ist *W* ein Weg in *G*, so ist die Länge *l(W)* des Weges *W* definiert als die Summe der Gewichtungen aller Kanten von *W*:

$$l(W) = \sum_{i=1}^{k} l(e_i)$$

Der Weg *W'* mit der minimalsten Länge *l(W')* aller Wege eines gewichteten Graphen vom Knoten *u* zum Knoten *v* heißt **kürzester Weg** von *u* nach *v*. Die Länge des kürzesten Weges von *u* nach *v* beziehungweise deren minimaler Abstand wird durch folgende Funktion angegeben:

$$dist(u,v) = \begin{cases} 0 & \text{falls } u = v \\ \min\{(l(W) : \text{ W ist ein Weg von u nach v}\} & \text{falls ein solcher Weg existiert} \\ \infty & \text{sonst} \end{cases}$$

2.2. Datenstrukturen für Graphen

Es gibt ein Vielzahl möglicher Datenstrukturen für Graphen. In diesem Abschnitt werden drei sinnvolle Datenstrukturen für Graphen, nämlich Kantenlisten, Adjazenz– oder Inzidenzmatrizen und Adjazenzlisten, erläutert.

2.2.1. Kantenlisten

Die **Kantenliste** eines Graphen $G=(V;E)$ mit m Kanten besteht aus einer Folge von m geordneten Paaren (u_i, v_i), also m Kanten $e_i = (u_i, v_i) \in E$ mit dem Anfangsknoten $u_i \in V$ und dem Endknoten $v_i \in V$ (siehe Abbildung 4a). Die Bewertungen der Kanten können hierbei in einer entsprechenden Liste gleicher Länge abgelegt werden.

Ein großer Vorteil der Kantenlisten als Darstellung von Graphen in einem Computer ist ihr geringer Speicherplatzbedarf. So können Kantenlisten sehr gut zum Schreiben eines Graphen auf sekundäre Speicher benutzt werden. Außerdem können besondere sequentielle Algorithmen gerade mit dieser Darstellung des Graphen schnell arbeiten. Da jedoch bei dieser Variante der Beschreibung die Struktur des Graphen kaum berücksichtigt wird, ist die Kantenliste für die meisten Anwendungen unhandlich und langsam.

Die folgenden Darstellungen nutzen die Adjazenz– beziehungsweise Inzidenzeigenschaften von Knoten und Kanten aus.

2.2.2. Adjazenz– und Inzidenzmatrizen

Eine **Adjazenzmatrix** ist eine boolsche (n x n)–Matrix $\mathbf{A} = (a_{ij})$. Die Einträge a_{ij} beschreiben die Kanten von Knoten i zu Knoten j. Besteht eine Verbindung zwischen den Knoten i und j, so ist a_{ij} gleich 1, sonst gleich 0. Handelt es sich bei dem Graph um einen ungerichteten Graph, so ist die Matrix symmetrisch (siehe Abbildung 4b). Um die Kantenbewertungen abzulegen wird oft anstatt der boolschen Matrix eine reele Matrix verwendet, wo an jeder Stelle a_{ij} die Bewertung steht (∞ für nicht vorhandene Kante). Diese Matrix, die der Fahrtenmatrix im Verkehrswesen gleicht, wird oft Strukturmatrix genannt.

Bei einer **Inzidenzmatrix** $\mathbf{A} = (a_{ij})$ wird jeder Knoten durch eine Zeile und jede Kante durch eine Spalte beschrieben. Wenn die Kante j den Knoten i verläßt, ist $a_{ij}=-1$. Wenn die Kante j auf den Knoten i zeigt, ist $a_{ij}=1$. Ansonsten ist $a_{ij}=0$.

Im Computer haben Adjazenz– bzw. Inzidenzmatrizen den Vorteil von schnellen Zugriffs– und Berechnungszeiten. Sie haben allerdings zwei entscheidene Nachteile. Zum einen ist die Größe des Graphen durch die Festlegung der Matrixgröße begrenzt. Dies kann man nur durch aufwendige Darstellungen einer Matrix beheben. Zum anderen ist eine Adjazenz– bzw. Inzidenzmatrix nur für relativ dichte Graphen vorteilhaft, da die meisten Einträge bei lichten Graphen Null sind. Nullen zu speichern nimmt unnötig viel Speicherplatz in Anspruch. Obwohl Adjazenzlisten aufwendiger zu programmieren sind, kann man mit ihnen die beiden erwähnten Nachteile beheben.

2.2.3. Adjazenzlisten

Die Grundidee einer **Adjazenzliste** ist, jeden Knoten in einem Graph mit seinen Adjazenzknoten direkt zu verbinden. Dazu werden die Knoten in einer Liste abgelegt. An jedem Element dieser Knotenliste "hängt" eine Kantenliste. Jeder Knoten ist zu den Kanten, die als Elemente in der angehängten Kantenliste sind, inzident. (siehe Abbildung 4c).

Im Rechner wird die Knotenliste meist als doppelt verkettete Liste realisiert. Mit Hilfe von Verweisen, wie Zeiger oder Indexe, werden die Kantenlisten an die Knotenlistenelemente (die Anfangsknoten der Kanten) "gehängt". Zu jedem Element der Kantenliste gehört ein Verweis auf den Endknoten der Kante, der zu dem Anfangsknoten adjazent ist.

Eine Adjazenzliste ist dynamisch und benötigt nur soviel Speicherplatz, wie Knoten und Kanten im Netz vorhanden sind. Diese Datenstruktur ist viel komplexer als eine Matrix. Sie ist aber für lichte Graphen, wie sie im Verkehrswesen zum überwiegenden Teil auftreten, sehr gut geeignet.

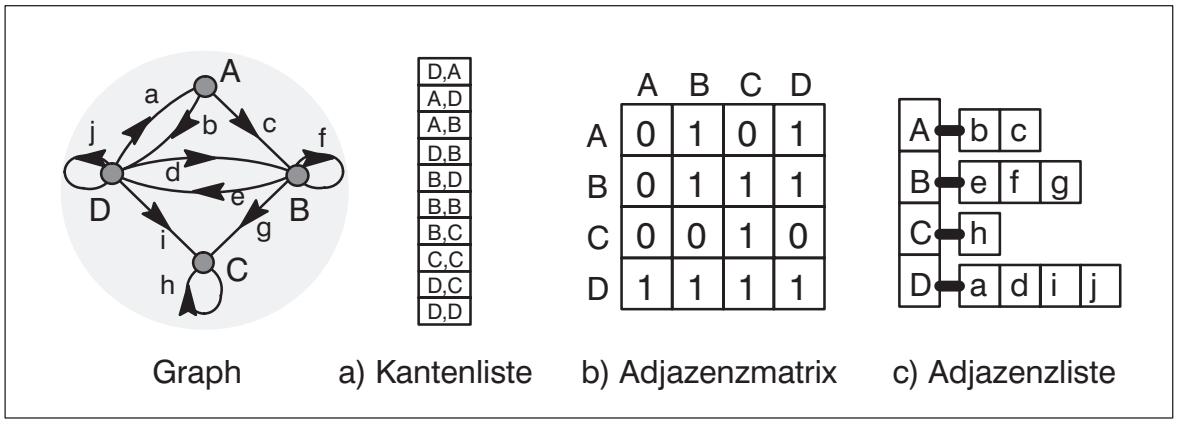


Bild 4: Mögliche Datenstrukturen eines Graphen

3. Abstraktion eines Verkehrsnetzes auf einen Graph

Um das Routensuchproblem innerhalb eines Verkehrsnetzes mit Hilfe effizienter Algorithmen zur Bestimmung kürzester Wege lösen zu können, ist es unbedingt erforderlich, das Verkehrsnetz auf eine geeignete Form zu abstrahieren. Als günstige Abstraktion eines Netzes für das Kürzeste−Wege−Problem hat sich ein gewichteter Graph $G=(V;E)$ erwiesen. Dabei werden die Kanten der Relation E in der Knotenmenge V durch eine Gewichtungsfunktion $l: E \rightarrow \mathbb{R}$ bewertet. Prinzipiell bestehen zwei Möglichkeiten, ein Verkehrsnetz auf einen solchen Graph abzubilden.

Bei der ersten Möglichkeit werden ausgewählte Orte, wie Knotenpunkte oder Ortschaften, im Verkehrsnetz auf die Knoten des Graphen abgebildet. Die Knotenmenge V enthält also feste Orte im Verkehrsnetz, denen Namen und Eigenschaften zugewiesen werden können. Die Relation E beschreibt die Erreichbarkeit eines Knoten von einem anderen. Eine Kante $e = (u,v)$ der Relation E besagt demnach: "Der Knoten v ist direkt vom Knoten u erreichbar". Je nachdem, für welches Kriterium die kürzesten Wege gesucht werden sollen, ordnet die Gewichtsfunktion $l: E \rightarrow \mathbb{R}$ jeder Kante e eine reele Zahl zu. Diese Zahl kann beispielsweise die Länge einer Strecke zwischen zwei Knotenpunkten, die Reisezeit oder vielleicht die Betriebskosten für das "Durchqueren" der Kante e darstellen. Diese erste Möglichkeit der Abstraktion eines Verkehrsnetzes auf einen Graph wird im Verkehrswesen oft "knotenpunktorientiert" genannt.

Bei der zweiten prinzipiell denkbaren Abstraktion eines Verkehrsnetzes auf einen Graph $G = (V;E)$ umfaßt die Knotenmenge V die Strecken des Verkehrsnetzes. Den Knoten können Namen und Eigenschaften von Strecken zugeordnet werden, wie etwa die Steigung, die Breite oder die Anzahl der Fahrspuren. Die Kanten der Relation E beschreiben die Erreichbarkeit einer Strecke von einer anderen. So können beispielsweise Abbiegebeziehungen erfaßt werden. Diese zweite Abstraktion des Verkehrsnetzes wird im Verkehrswesen meist "streckenorientiert" genannt.

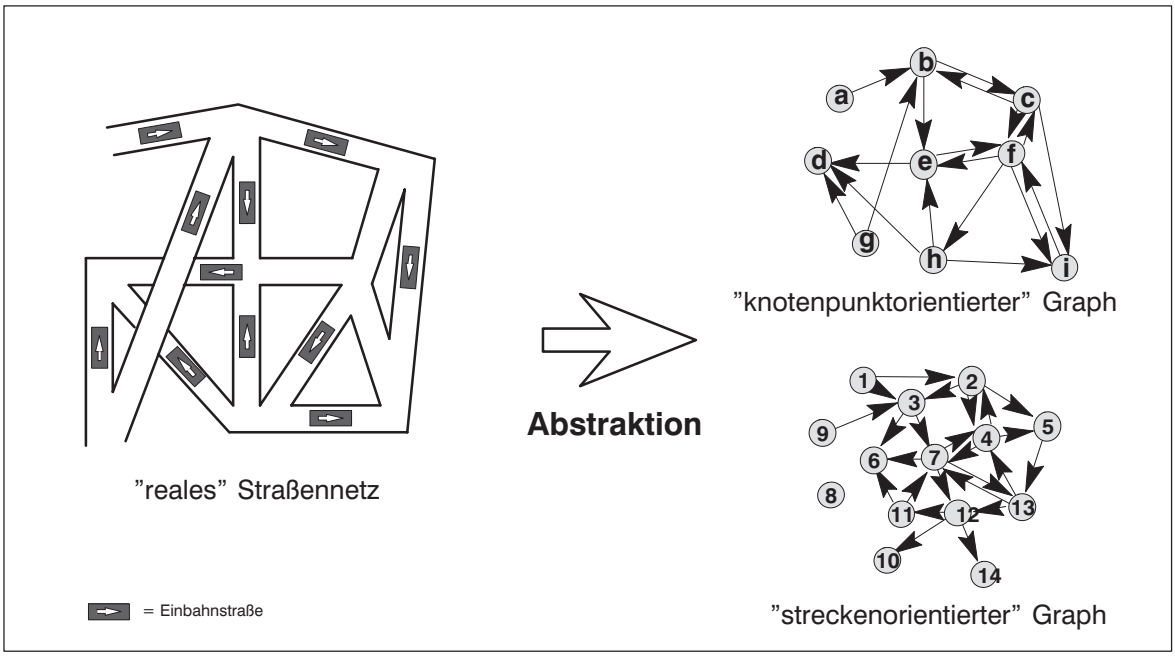


Bild 5: Abstraktion eines Verkehrsnetzes auf einen Graph

In der Verkehrsplanung kann es vorkommen, daß weiterführende Eigenschaften des Verkehrsnetzes in den Graph mit aufgenommen werden sollen, die jedoch seiner Definition wiedersprechen. Denkbar ist etwa der Wunsch nach Abbildung der Abbiegebeziehungen in einem knotenpunktorientierten Graph. Oft werden dann "Tricks" (wie eine "Knotenpunktauflösung") oder auch unzulässige Modifikationen am Graph vorgenommen, anstatt eine grundlegend andere Form der Abstraktion zu untersuchen, die das gegebene Verkehrsproblem besser darstellen kann, als ein gewichteter Graph $G=(V;E)$. Im letzten Kapitel dieser Arbeit werden solche Modellierungsprobleme von Verkehrsnetzen diskutiert.

Da die klassischen Kürzeste–Wege–Algorithmen mit einfach definierten Graphen arbeiten und komplexere Graphendarstellungen eher zur Unübersichtlichkeit führen, basiert die Beschreibung der Routensuchverfahren in den nächsten zwei Kapiteln auf dem oben erläuterten gewichteten Graph $G = (V;E)$ mit einer Knotenmenge V, einer Relation E in V und einer Kantengewichtungsfunktion $l: E \rightarrow \mathbb{R}$.

Es sollte dabei beachtet werden, daß sich für normale Verkehrsnetze zusammenhängende, lichte und zyklische Graphen ergeben. Dies ergibt sich aus der notwendigen Erreichbarkeit eines jeden Ortes in einem Verkehrsnetz von jedem anderen und der relativ wenigen Ortsverbindungen im Verhältnis zu den Orten. Die Kantengewichtung ist für das Routensuchproblem im Verkehrswesen meist positiv.

4. Grundlegende Routensuchverfahren

4.1. Überblick

Nachdem ein Verkehrsnetz auf einen gewichteten Graph $G=(V; E)$ abgebildet wurde, kann die Routensuche im Sinne der Bestimmung kürzester Wege durchgeführt werden. Die Beschränkung von Routensuchen auf Kürzeste–Wege–Algorithmen, wie sie in der vorliegenden Arbeit verwendet werden, ist nicht zwingend. So ist beispielsweise gerade das älteste Problem, das durch die Abstraktion auf einen Graph gelöst werden konnte, interessanterweise eine Aufgabe bezüglich einer bestimmten Route, aber nicht eines kürzesten Weges. Bei diesem Problem handelt es sich um das Königsberger Brückenproblem. Das Problem behandelt die Frage, ob man es schafft, seinen sonntäglichen Spaziergang durch Königsberg so zu gestalten, daß man auf einem geschlossenen Weg jede der sieben Pregelbrücken nur einmal überquert, um schließlich wieder am Ausgangspunkt des Spaziergangs anzukommen (siehe Abbildung 6 links). 1736 hat Euler [Eu36] mit Hilfe der Abstraktion eines ungerichteten Graphen die Unmöglichkeit eines solchen Rundgangs bewiesen und dabei die ersten allgemeingültigen Sätze der Graphentheorie aufgestellt.

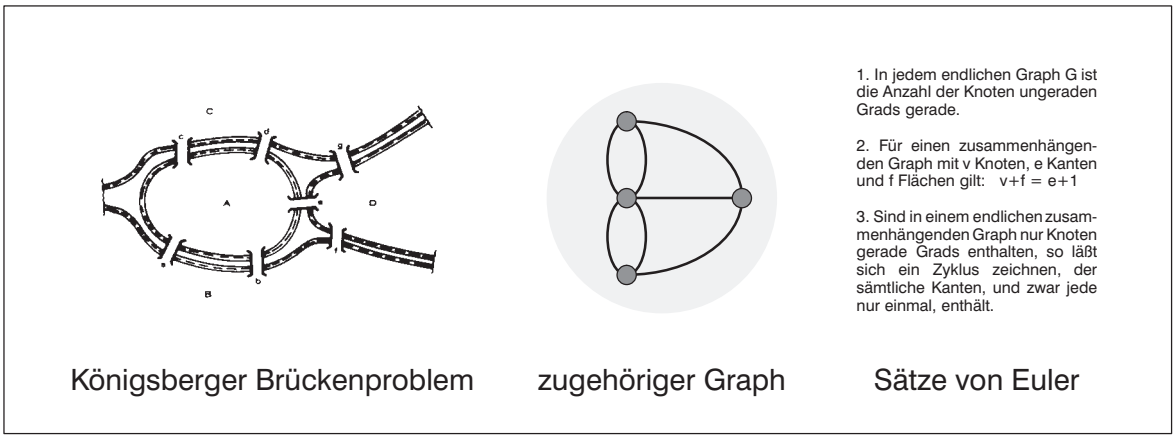


Bild 6: Königsberger Brückenproblem, gelöst 1736 durch Euler

Es gibt eine Vielzahl weiterer Graphenprobleme mit Routen, wie etwa das Chinesische Postbotenproblem [Kw62], Hamiltonsche Kreise [Ber89] oder das für Speditionen sehr wichtige Travelling–Salesman–Problem [La85], das sich mit kürzesten Rundreisen durch vorgegebene Orte beschäftigt. Bei den letztgenannten, zur Zeit noch nicht optimal gelösten Problem ist die Bestimmung kürzester Wege als Teilalgorithmus notwendig.

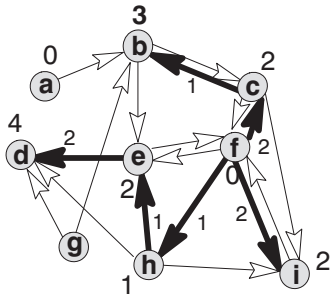
In den folgenden Abschnitten werden die Kürzeste–Wege–Algorithmen zunächst prinzipiell nach der Anzahl der Startknoten der gesuchten Wege eingeteilt, dann die konkrete Aufgabenstellung und die zur Lösung wichtigsten Grundgleichungen dargestellt. Mit der anschließenden Beschreibung der Entwicklungsgeschichte der Routensuchverfahren soll ein Einblick in den Ablauf und die Probleme beim Entwurf von Kürzeste–Wege–Algorithmen gegeben werden. Im Abschnitt 4.2 werden dann einige grundlegende Routensuchverfahren anschaulich beschrieben und analysiert.

4.1.1. Einteilung der Routensuchverfahren nach der Startknotenanzahl

Routensuchverfahren lassen sich prinzipiell nach der Anzahl der Startknoten der kürzesten Wege unterscheiden. Dabei gibt es zwei grundsätzliche Betrachtungsweisen. Die erste geht davon aus, daß der optimale Weg von einem Knoten zu allen anderen Knoten des Graphen gesucht wird. In der Graphentheorie wird ein solcher Algorithmus "Kürzeste Wege von einem Knoten aus" genannt (englisch "single source shortest paths"). Im Verkehrswesen werden diese Routensuchverfahren wegen der Struktur der Ergebnisse häufig "Baumalgorithmen" bezeichnet. "Folgealgorithmen", also Algorithmen, die exakt die lineare Folge des Weges zwischen zwei Knoten bestimmen, existieren nicht, da bei der Suche nach dem kürzesten Weg automatisch die kürzesten Wege zu den übrigen Knoten des Graphen gefunden werden – so ergibt sich die Baumstruktur.

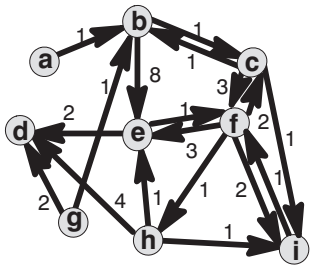
Die zweite grundsätzliche Betrachtungsweise in bezug auf die Startknoten geht davon aus, daß die gesamten kürzesten Wege zwischen allen Knoten in einem Algorithmus gefunden werden. In der Graphentheorie heißen diese Algorithmen "Kürzeste Wege zwischen je zwei Knoten" (englisch "all pairs shortest paths"). Im Verkehrswesen werden sie wegen der Ergebnisstruktur "Matrixalgorithmen" genannt (obwohl "Waldalgorithmen" konsequenter wäre). Diese Verfahren haben den Vorteil, daß sie anschaulicher als die "Baumalgorithmen" sind. So gestattet beispielsweise die Wegealgebra [Ca71], alle kürzesten Wege durch den bekannten Gauß–Seidel–Gleichungslöser zu bestimmen.

Oft ist es notwendig, die optimalen Wege zu mehreren Startknoten zu suchen. In diesem Fall werden die "Kürzeste–Wege–Algorithmen von einem Knoten aus" mehrmals aufgerufen. Es ergibt sich dann ein Wald mit Bäumen kürzester Wege. Versuche, während des Programmablaufs Teile des schon bekannten Kürzeste–Wege–Waldes zu nutzen, um die Zeit zur Bestimmung noch ausstehender Bäume zu minimieren, haben lediglich zu einer Verringerung der Rechenzeit um einen geringen Faktor geführt.



	a	b	c	d	e	f	g	h	i
a	0	1	∞	∞	∞	∞	∞	∞	∞
b	∞	0	1	∞	8	∞	∞	∞	∞
c	∞	1	0	∞	∞	3	∞	∞	1
d	∞	∞	∞	0	∞	∞	∞	∞	∞
e	∞	∞	∞	2	0	1	∞	∞	∞
f	∞	3	2	4	2	0	∞	1	2
g	∞	1	∞	2	∞	∞	0	∞	∞
h	∞	∞	∞	4	1	∞	∞	0	1
i	∞	∞	∞	∞	∞	1	∞	∞	0

"Baumalgorithmus" für Startknoten f



	a	b	c	d	e	f	g	h	i
a	0	1	2	8	6	4	∞	5	3
b	∞	0	1	7	5	3	∞	4	2
c	∞	1	0	6	4	2	∞	3	1
d	∞	∞	∞	0	∞	∞	∞	∞	∞
e	∞	4	3	2	0	1	∞	2	3
f	∞	3	2	4	2	0	∞	1	2
g	∞	1	2	2	7	5	0	6	3
h	∞	5	4	3	1	2	∞	0	1
i	∞	4	3	5	3	1	∞	2	0

"Matrixalgorithmus"

Bild 7: "Kürzeste Wege von einem Knoten aus" und "Kürzeste Wege zwischen je zwei Knoten"

4.1.2. Aufgabenstellung zur Bestimmung kürzester Wege

Die konkrete mathematische Formulierung der Aufgabe zur Bestimmung kürzester Wege in einem Graph lautet wie folgt:

Gegeben sei ein Graph $G = (V; E)$ mit einer Knotenmenge V, einer Relation $E \subseteq V \times V$ und eine Gewichtungsfunktion $l: E \rightarrow \mathbb{R}$, die jeder Kante $e \in E$ genau einen reelen Wert zuordnet.

Gesucht sind die kürzesten Wege W' von einem oder allen Knoten $v_1 \in V$ zu allen übrigen Knoten $v_i \in V\,(i = 2,3,...,|V|)$ des Graphen G, sodaß der Abstand $dist(v_1,v_i)$ zwischen allen Knoten v_1 und v_i minimal wird:

$$dist(v_1, v_i) = \begin{cases} 0 & \text{falls } v_1 = v_i \\ \min\{(l(W) : \text{ } W \text{ ist ein Weg von } v_1 \text{ nach } v_i\} & \text{falls ein solcher Weg existiert} \\ \infty & \text{sonst} \end{cases}$$

$$\text{mit } \; l(W) = \sum_{j=1}^{k} l(e_j)\,, \;\; k = \text{Anzahl der Kanten } e_j \text{ von } W$$

Die Lösung dieser Aufgabe stellt ein Optimierungsproblem dar. Im linearen Fall kann das Problem mit der "Simplex−Methode" gelöst werden, wie es beispielsweise Dantzig für Transportprobleme gezeigt hat [Da51]. Im wesentlich häufiger auftretenden nichtlinearen Fall wird ein dynamisches Optimierungsverfahren benötigt. Eine rekursive Methode für eine dynamische Optimierung hat Bellman in seinem 1957 eingereichten Artikel "On a Routing Problem" [Bel58] gezeigt und dabei die für die Suche kürzester Wege grundlegenden "Bellman−Gleichungen" aufgestellt:

$$dist(v_1, v_i) = \min\Big(l(v_i, v_j) + dist(v_1, v_j)\Big) \quad \text{mit} \quad i = 2,3,...,|V|$$
$$dist(v_1, v_1) = 0$$

Diese Gleichungen besagen, daß wenn $l_1(v_1,v_k) = (v_1, v_2,..., v_i,..., v_k)$ ein kürzester Weg von v_1 nach v_k ist, $l_1^*(v_i,v_k) = (v_i,..., v_k)$ ein kürzester Weg von v_i nach v_k sein muß. Wäre nämlich beipielsweise $l_u^*(v_i,v_k) = (v_i,..., u,..., v_k)$ *mit* $u \in l_1(v_1,v_k)$ kürzer als l_1^*, dann könnte ein Gesamtweg $l_u(v_1,v_k) = l_1(v_1, v_i) \cup l_u^*(v_i,v_k)$ mit $l_u(v_1,v_k) < l_1(v_1, v_k)$ konstruiert werden, was jedoch der definitionsgemäßen Optimalität von $l_1(v_1,v_k)$ widersprechen würde.

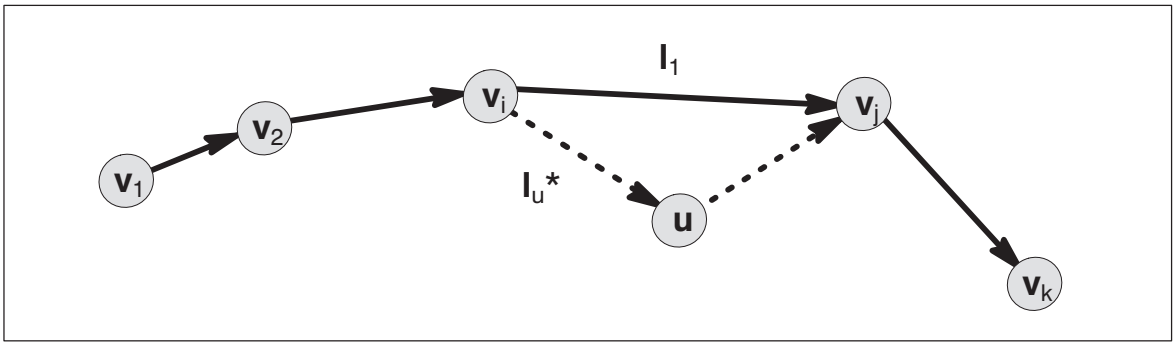
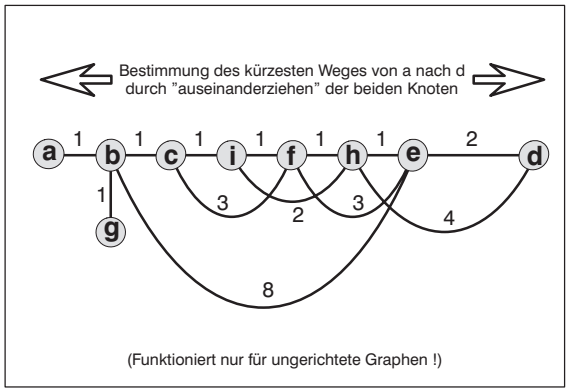


Bild 8: Skizze zur Erläuterung der Bellman–Gleichungen

4.1.3. Entwicklungsgeschichte der Routensuchverfahren

Dieser Abschnitt zeigt kurz die Entwicklungsgeschichte von Algorithmen zur Bestimmung kürzester Wege auf, sodaß die prinzipiellen Probleme beim Entwurf dieser Algorithmen deutlich werden. Einen umfassenden Überblick über die Geschichte der Routensuchverfahren bis Mitte der sechziger Jahre wird in [St66] und [Ri72] aufgezeigt. Dort sind unter anderem auch die Vorläufer der Routensuchverfahren beschrieben, wie etwa das "Fadenmodell" von Minty, wo Start und Ziel einer Route in einem Straßennetz "auseinander gezogen werden", oder Verfahren mit Analogrechnern zur Unterstützung des Verkehrsplaners bei der Schätzung kürzester Wege.



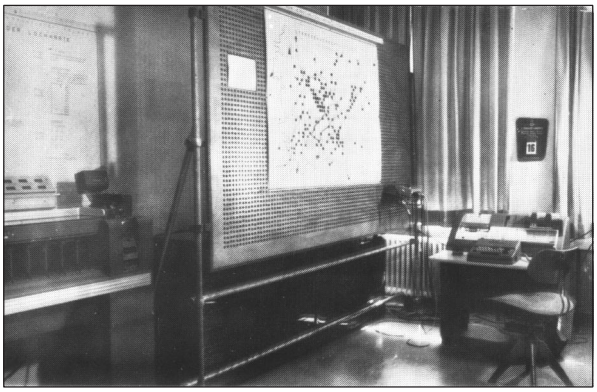


Bild 9: Fadenmodell von Minty und Analogrechner zur Routenumsetzung [Ri72]

Die klassischen Routensuchverfahren sind interessanterweise alle im sehr kurzen Zeitraum von Mitte der fünfziger bis Anfang der sechziger Jahre entstanden. An den grundsätzlichen Methoden, die dabei entwickelt wurden, hat sich seitdem nichts Wesentliches geändert. Die Entwicklung der Routensuchverfahren gerade in dieser Zeit läßt sich einerseits durch die damals häufiger werdenden Probleme im Verkehrs–, Transport– und Nachrichtenwesen erklären. Andererseits erhöhte das Aufkommen der elektronischen Rechner die Rechtfertigung solcher Verfahren, da schnell deutlich wurde, daß die zunächst als Handrechenmethoden genutzten Verfahren durch den Einsatz von Rechnern in absehbarer Zeit sehr viel effizienter und in viel größerem Umfang realisierbar sein würden. So konnten zum Beispiel erstmals größere Verkehrsnetze berechnet werden.

1954 hat Shimbel auf einem "Symposium of Information Networks" in Brooklyn die von ihm entwickelte "Min–Additions–Methode" vorgestellt [Sh55]. Sie war zur Lösung des Problems der Entfernungsberechnung zwischen diskreten Netzpunkten gedacht. Dabei verwendete er die Strukturmatrix. Das Verfahren stellt die Grundlage aller späteren Matrixalgorithmen, besonders auch der Wegealgebra, dar.

Shimbels Verfahren besteht aus einer Folge von Matrizenmultiplikationen, wobei für beliebige reele und unendliche Zahlen x und y gilt:

$$\begin{aligned} x + y &= \min(x, y) \\ x \cdot y &= \textit{algebraische Summe von x und y} \\ \infty \cdot x &= x \cdot \infty = \infty \end{aligned} \tag{3}$$

Es muß dabei beachtet werden, daß ein solches Matrixprodukt nicht kommutativ ist.

Bei dem Verfahren wird eine Ausgangsmatrix $\mathbf{M}^1 = (m_{ij})^1$, dessen Diagonalelemente $(m_{ii})^1 = 0$ und alle übrigen Einträge $(m_{ij})^1 = \infty$ sind, mit der Strukturmatrix $\mathbf{S} = (s_{ij})$ multipliziert. Die Ergebnismatrix $\mathbf{M}^2$ wird, wie alle folgenden Ergebnismatrizen ($\mathbf{M}^3$, $\mathbf{M}^4$, ..., $\mathbf{M}^k$) wieder mit $\mathbf{S}$ multipliziert:

$$\boldsymbol{M}^2 = \boldsymbol{M}^1 \cdot \boldsymbol{S} \rightarrow \boldsymbol{M}^3 = \boldsymbol{M}^2 \cdot \boldsymbol{S} \rightarrow \ldots \rightarrow \boldsymbol{M}^k = \boldsymbol{M}^{k-1} \cdot \boldsymbol{S}$$

$$m_{ij}^l = \sum_{i=1}^{n} m_{ij}^{l-1} \cdot s_{ji} = m_{1j}^{l-1} \cdot s_{j1} + m_{2j}^{l-1} \cdot s_{j2} + \ldots + m_{nj}^{l-1} \cdot s_{jn}$$

Mit Definition (3) ergibt sich folgende Minimierung für m_{ij}^l:

$$m_{ij}^l = \min\left\{(m_{1j}^{l-1} + s_{j1}), (m_{2j}^{l-1} + s_{j2}), \ldots, (m_{nj}^{l-1} + s_{jn})\right\}$$

Bei jeder Multiplikation wird das Ergebnis weiter minimiert, da für jeden Weg zwischen zwei Knoten (Einträge der Matrix) durch Einfügen eines weiteren Knotens in den Weg der Abstand eventuell verbessert werden kann. Dieser Vorgang der Minimierung entspricht dem Prinzip der Bellman−Gleichungen. Das Verfahren wird beendet, wenn keine weitere Verbesserung in der Ergebnismatrix auftritt. Die Ergebnismatrix enthält sämtliche minimalen Abstände zwischen je zwei Knoten. Ein kürzester Weg zwischen zwei Knoten ist durch Rückrechnung bestimmbar.

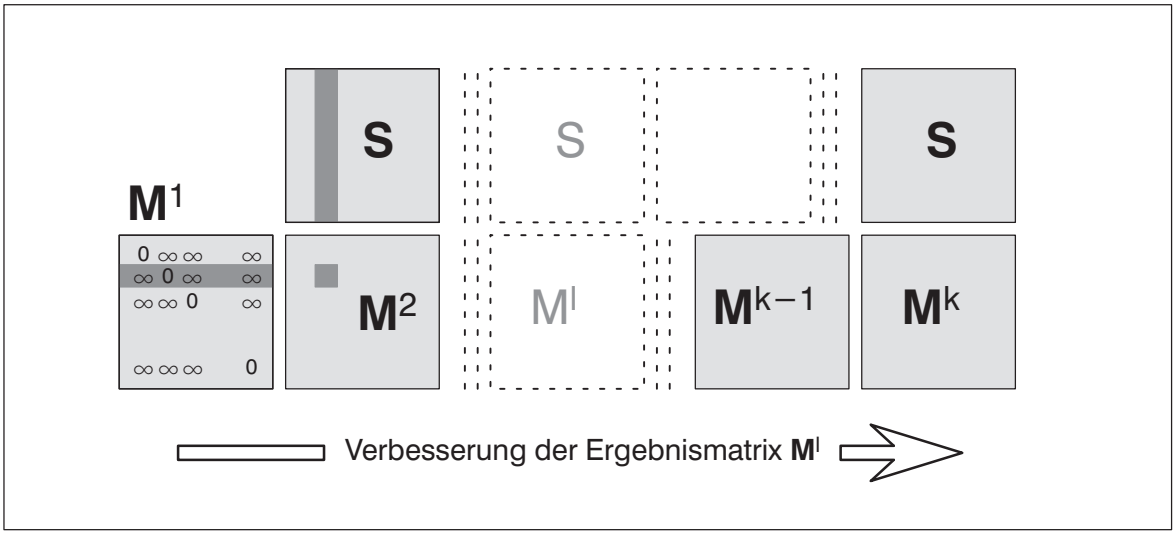


Bild 10: "Min−Additions−Methode" von Shimbel (1954)

Eine entscheidende Rolle bei der Entwicklung von Routensuchverfahren und einer Vielzahl weiterer Transport− und Verteilungsprobleme hat eine Gruppe der RAND Corporation in Santa Monica eingenommen, wo unter anderem auch Bellman beschäftigt war. Zu dieser Gruppe gehörten Dantzig, Ford, Fulkerson und Johnson. Sie haben sehr viele Publikationen in Zeitschriften und Büchern, wie [FoFu62], veröffentlicht.

In diesem Rahmen hat Ford 1956 in [Fo56] die erste Routensuche mit einem Baum kürzester Wege von einem Startknoten aus vorgestellt. Obwohl sein Baumalgorithmus aus den Untersuchungen der Dualität des Kürzeste−Wege− mit dem Max−Flow−Min−

Cut–Problem [FoFu56] entstand, ist sein Verfahren identisch mit dem von Shimbel, bis auf den Unterschied, daß Ford nur die kürzesten Wege zu einem statt zu allen Startknoten sucht. So wird das Ergebnis nicht in einer Matrix gespeichert, sondern in einer Liste. Sie entspricht der Zeile eines Startknoten in der Ergebnismatrix $\mathbf{M}^l$ (Abbildung 11).

Bei den sogenannten "Baumalgorithmen" ist das Ergebnis meist eine solche Liste minimaler Abstände zum Startknoten. Der Baum kürzester Wege wird, wie der Wald kürzester Wege bei den "Matrixalgorithmen" durch Rückrechnung bestimmt. Daß heißt, ausgehend vom Endknoten des Weges werden jeweils die beiden größten Listeneinträge voneinander abgezogen. Entspricht die Subtraktion der Gewichtung einer zu den Knoten inzidenten Kante, so ist diese Kante eine Kante des kürzesten Weges. Angelangt am Startknoten ist der Baum rückwärts auf dem kürzesten Weg einmal durchlaufen worden.

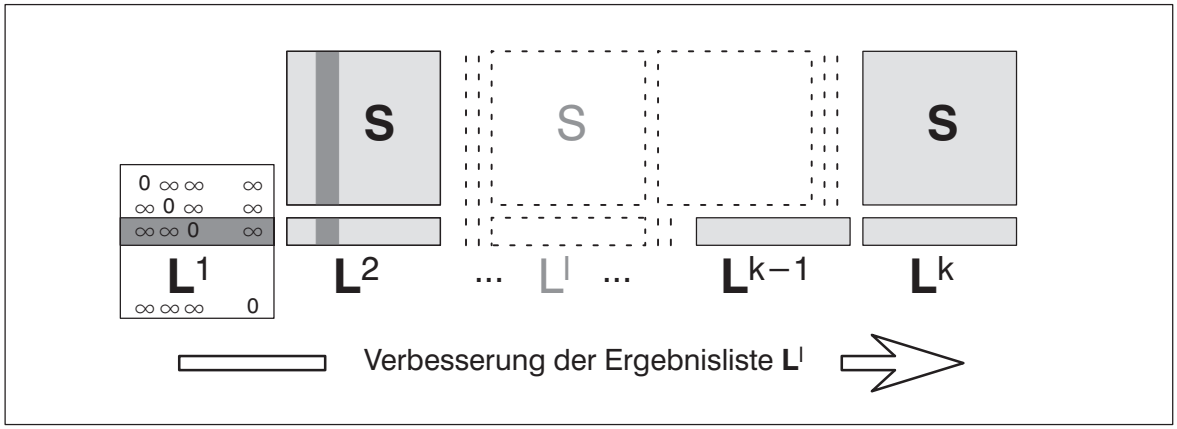


Bild 11: Erster Baumalgorithmus von Ford (1956)

Die Verfahren von Shimbel und Ford sind sehr ineffektiv sowohl in bezug auf die Rechenzeit ($O(n^4)$) als auch auf den benötigten Speicherplatz. So bleiben unter anderem die Vorteile der Matrixdarstellung ungenutzt. Daher wurden diese Grundverfahren der Routensuche in der Folgezeit immer weiter verbessert. Shimbel selbst entwickelte 1958 eine Verbesserung seines Verfahrens, die für jede Multiplikation eine gesamte Matrix weniger abspeichern muß. Der Rechenaufwand verringerte sich jedoch nicht.

1957 stellte Moore auf dem "International Symposium on The Theory of Switching" einen Baumalgorithmus vor, der durch Einführung von Markierungen an bestimmten Knoten im Graph während der Laufzeit eine entscheidene Verbesserung der Routensuche erreicht [Mo59]. Er erläuterte dieses als Handrechenmethode gedachte, aber auch auf elektronische Rechner übertragbare Verfahren an zwei ausgewählten Beispielen.

Im ersten Beispiel (Abbildung 12 links) wird der minimale Abstand von einem Startknoten (A) zu einem Endknoten (B) in einem ungewichteten Graph auf einfache Weise bestimmt. Zunächst wird der Startknoten mit 0 markiert. Dann wird in einem Schritt *1* die zu diesem Knoten adjazenten Knoten gesucht und mit *1* markiert. Zu den zuletzt mit *(i–1)* markierten Knoten werden in jedem weiteren Schritt *i* die benachbarten und noch nicht markierten Knoten mit *i* markiert. Der Algorithmus endet, wenn entweder der

Endknoten oder aber alle vom Startknoten aus erreichbaren Knoten markiert sind. Das Ergebnis in Moores Beispiel lautet ”AFDCMLB”.

Die Verallgemeinerung dieses Algorithmus auf einen Wald mit Bäumen minimaler Abstände führt zum bekannten Algorithmus der Breitensuche in einem Graph (siehe Abschnitt 4.2.1.1).

Die in seinem ersten Beispiel beschriebene Methode überträgt Moore auf die Suche nach dem kürzesten Weg zwischen den gleichen Start− und Endknoten (A und B) in dem gleichen Graph, wie vorher, wobei seine Kanten nun gewichtet sind (Abbildung 12 rechts). Die Markierung der Knoten *v* mit normalen Zahlen aus einer einfachen Reihenfolge wird ersetzt durch die Markierung mit der Summe aus der Markierung des Knoten *u*, zu dem der Knoten *v* adjazent ist. Diese Markierung wird nur dann durchgeführt, wenn die Summe kleiner ist, als die Markierung, die der Knoten *v* eventuell schon besitzt. Dieser Vorgang entspricht der Anwendung der Bellman−Gleichungen.[1)]

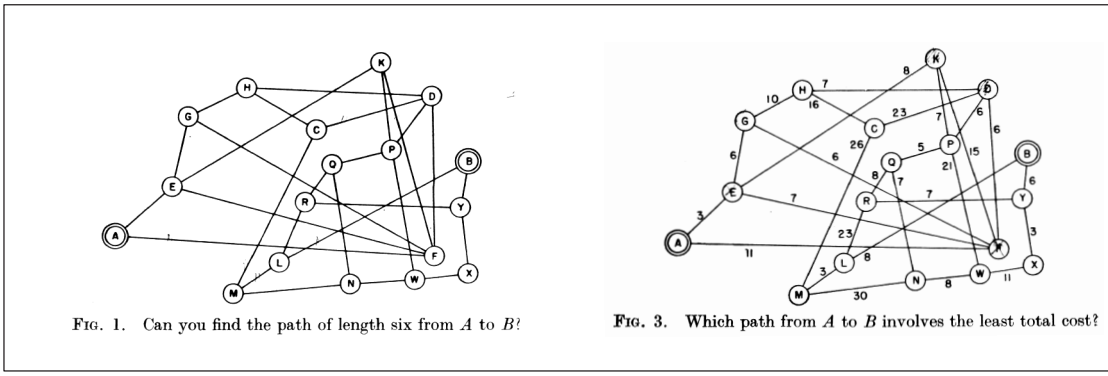


FIG. 1. Can you find the path of length six from *A* to *B*?

FIG. 3. Which path from *A* to *B* involves the least total cost?

Bild 12: Beispiele von Moore zur Erläuterung der Breitensuche und seiner Routensuche [Mo59]

Durch die Markierungen, die Moore eingeführt hat, wird das gesuchte Minimum nicht mehr global im Graph gesucht, sondern nur noch lokal an einer ”Suchfront”, die aus den zuletzt markierten Knoten, den Kandidaten besteht. Diese Einschränkung des Suchbereichs auf eine Kandidatenmenge bewirkt eine schnellere Rechenzeit (siehe 4.2.2.1).

Der Algorithmus hat jedoch ein Manko: Durch die willkürliche Wahl eines Knoten aus der Kandidatenmenge kann es oft vorkommen, daß bereits bearbeitete Knoten mehrmals durch neue Markierungen optimiert werden und so zu einer unwirtschaftlichen Rechenzeit führen. Wegen dieses sich wiederholenden Durchlaufs durch den Graph, der erst beendet werden kann, wenn sich keine Knotenmarkierung mehr verbessert, werden solche Baumalgorithmen oft ”Iterationsverfahren” genannt.

Die sogenannten ”iterationslosen Verfahren”, in denen zwar auch Iterationen verwendet werden (weswegen der Name eher irreführend als hilfreich ist), die aber das Manko der mehrmaligen Markierung schon bearbeiteter Knoten beheben, entstanden in den folgenden zwei Jahren nach Moores Beitrag 1957. Sie zeichnen sich dadurch aus, daß für den jeweils nächsten Rechenschritt aus der Kandidatenmenge immer genau der Knoten mit der minimalsten Markierung ausgewählt wird. Da die Markierung eines Knoten dem

1) Das Ergebnis der Routensuche im zweiten Beispiel ist übrigens "AEKPQRYB".

Abstand zum Startknoten entspricht, garantieren die Bellman – Gleichungen, daß der Weg vom Startknoten zu diesem minimal markierten Kandidatenknoten ein minimaler Weg ist, und der Knoten somit im weiteren Verlauf nicht mehr bearbeitet werden muß. Darüberhinaus kann aus demselben Grund der Algorithmus bereits abgeschlossen werden, wenn ein gesuchter Endknoten gefunden wurde.

Die "iterationslosen Routensuchverfahren" wurden unter anderem von Dantzig [Da59], Dijkstra [Di59], Minty [Po60], Whiting und Hillier [Wh60] entwickelt. Sie unterscheiden sich durch die unterschiedliche Suche nach dem Kandidatenknoten mit dem geringsten Abstand zum Startknoten. Zu Beginn der Entwicklung von "Iterationslosen Verfahren" wurde eine einfache Liste verwendet, in der der nächste Kandidatenknoten gesucht werden mußte. Zur Suche des minimalen Elementes wurden unterschiedliche Methoden und Hilfen verwendet. Dantzig [Da59] schaffte mit einer Eingrenzung der Liste auf potentielle Kandidaten eine quadratische Rechenzeit für den Baumalgorithmus, brauchte aber für eine hierzu notwendige Vorsortierung der Knoten einen Rechenaufwand von $n^2 \cdot \log^2 n$ Schritte.

In seinem Artikel [Di59] im ersten Band der Zeitschrift "Numerische Mathematik" stellt Dijkstra 1959 zwei Algorithmen vor, von denen der zweite eine Routensuche mit einer bisher nicht erreichten quadratischen Rechenzeit war. Dieser Algorithmus zur Bestimmung der kürzesten Wege von einem Knoten aus ist in der Graphentheorie zum Standardverfahren für eine Bestwegesuche geworden. Fast alle neueren Veröffentlichungen zur Routensuche nehmen auf diesen Algorithmus bezug. Beispielsweise stellt der von Dial 1969 veröffentlichte Algorithmus *MOORE* [Dia69] eine Ausprägung des Verfahren von Dijkstra dar.

Ebenso wie Moore erläutert Dijkstra in seinem Artikel zunächst ein ähnliches Problem, bevor er eine Lösung für die Routensuche angibt. Dieses ähnliche Problem ist die Suche nach einem minimalen Spannbaum in einem Graph. Das Problem wurde vorher schon von Kruskal 1956 in [Kr56] gelöst. Dijkstra, der sich direkt auf ihn bezieht, zeigt eine schnellere Lösung. Er verteilt dazu die Knoten und die Kanten in zwei beziehungsweise drei disjunkte Knoten – und Kantenmengen und sorgt dafür, daß sie zu jedem Zeitpunkt disjunkt bleiben. So hält er den Speicherplatz so klein, wie möglich. Wichtiger ist jedoch, daß nur noch eine kleine Anzahl der wirklich relevanten Knoten betrachtet werden muß. Diese kann dann schnellstmöglich durchsucht werden.

Bei der Vorstellung seines Routensuchalgorithmus bezieht sich Dijkstra auf den bereits oben beschriebenen Algorithmus von Ford [Fo56], nicht auf den von Moore. Er überträgt die Methode, die er für die Bestimmung des minimalen Spannbaumes eines zusammenhängenden Graphen verwendet hat, auf sein Routensuchverfahren. Dabei benötigt er nun jeweils drei disjunkte Mengen für die Knoten und Kanten. Die Knoten sind aufgeteilt auf die Menge A für die Knoten, die bereits zum kürzesten Weg gehören, auf die Kandidatenmenge B und auf die Menge C für die restlichen Knoten. Die Kantenmengen I, II und III bestehen aus den Kanten, die den Baum bereits bestimmter kürzester Wege

bilden, aus den Kanten, von denen als nächstes eine zu den kürzesten Wegen gehören wird, und aus den Restkanten.

Zunächst sind alle Knoten in der Menge C und alle Kanten in der Menge III. Nachdem der Startknoten in die Menge A übertragen wurde, wiederholen sich im Algorithmus nur noch zwei Schritte:

Schritt 1:
Knoten, die zu dem zuletzt nach A geschriebenen Knoten adjazent sind, kommen nach B, wenn sie vorher in C waren – gleichzeitig kommen die inzidenten Kanten in die Menge II. Bei adjazenten Knoten, die schon in der Menge B stehen, ersetzt die neue inzidente Kante die alte in II, wenn der Widerstand dadurch verbessert wird (Anwendung der Bellman–Gleichungen).

Schritt 2:
Der Knoten aus B mit dem minimalen Abstand zum Startknoten wird zur Menge A hinzugefügt und ist in der nächsten Iteration Ausgangsknoten. Die inzidente Kante wird zur Menge I mit dem Baum kürzester Wege hinzugefügt. Ist der Endknoten zur Menge A hinzugefügt worden, so kann der Algorithmus beendet werden.

Menge B wird als Sequenzliste, die nur die Knoten aufnimmt, die wirklich Kandidat sind, implementiert. So muß das minimale Element nur noch in der kleinstmöglichen Kandidatenliste gesucht werden. Der Gesamtrechenaufwand reduziert sich so auf n^2 Schritte.

Mit dem Wissen, daß der Bereich, in dem der minimale Kandidatenknoten gesucht wird, möglichst klein sein muß, bestand die Aufgabe aller späteren Baumalgorithmen nur noch darin, diese Minimumsuche effizienter zu gestalten. Somit war das Grundproblem der Baumalgorithmen gelöst, da eine weitere Effizienzsteigerung nur noch eine Aufgabe von Suchen und Sortieren der Kandidatenknoten sein konnte.

Für Kürzeste Wege zwischen je zwei Knoten wurden nach Shimbel zunächst nur einige weniger gute Matrixalgorithmen entworfen, wie etwa der von Narahari Pandit, den Farbey, Land und Murchland später zum "Cascade–Algorithmus" [Fa67] weiterentwickelten. Eine Effizienz, wie bei den Baumalgorithmen schien für Matrixalgorithmen nicht erreichbar.

1962 veröffentlichte Floyd jedoch einen nur elf (!) Zeilen kurzen Quellcode für einen Matrixalgorithmus (siehe Algorithmus 7), der nur drei Schleifen und keine Markierungen oder ähnliches braucht, aber ebenso schnell ist, wie der Aufruf des Baumalgorithmus von Dijkstra für jeden Knoten des Graphen [Fl62]. Floyd übertrug die Resultate von Warshalls Analyse zur Rechnung mit Adjazenzmatrizen [Wa62] auf die Strukturmatrizen im Problem der Bestimmung kürzester Wege zwischen je zwei Knoten.

Vom Matrixalgrithmus nach Shimbel unterscheidet sich Floyds Verfahren letztendlich nur durch den Wechsel der äußeren von den drei notwendigen Schleifen nach innen. Anstatt n Matrixprodukte wie bei Shimbel zu berechnen, wird bei Floyd nacheinander jeder der n Matrixeinträge einmal konstant festgehalten und sein Einfluß auf die eventuelle Minimierung von Wegen zu anderen Knoten mit Hilfe der Bellman–Gleichungen be-

stimmt. So ergibt sich ein einziger Durchlauf durch eine Strukturmatrix, was eine enorme Reduzierung der Rechenzeit auf n^3 Schritte und einen minimalen Speicherplatzverbrauch bedeutet.

4.2. Darstellung und Analyse ausgewählter Routensuchverfahren

Im letzten Abschnitt wurde ein Abriß der Entwicklungsgeschichte von Kürzeste−Wege−Algorithmen dargestellt. Dabei hat sich insbesondere bei Moore und Dijkstra gezeigt, wie sinnvoll es sein kann, Ideen aus anderen Problemstellungen zu übernehmen. Moore hat aus der leichtverständlichen Darstellung der Breitensuche die Markierungsstrategie auf die Routensuche übertragen und verringerte damit den Aufwand des Verfahrens. Dies gelang Dijkstra durch Einführung einer Sequenzliste für die Kandidatenknoten, die er vorher für die Konstruktion eines minimalen Spannbaumes einleuchtend erklären konnte. Floyd hat durch Verwendung der Rechnung mit boolschen Matrizen nach Warshall in wenigen Zeilen einen Matrixalgorithmus zur Suche aller kürzesten Wege aufgezeigt, der heute noch zu den schnellsten zählt. Allgemein zeichnen sich die klassischen Routensuchverfahren durch ihre Kürze und relative Einfachheit aus.

Im Folgenden werden ausgewählte Algorithmen analysiert. Sie sollten als exemplarische Beispiele für die klassischen Routensuchverfahren verstanden werden. Zunächst wird der Breitensuchalgorithmus nach Moore für die Suche minimaler Abstände von Knoten in einem ungewichteten Graph vorgestellt. Um den Unterschied der "Iterationsverfahren" und der "Iterationslosen Verfahren" für die Bestimmung der kürzesten Wege in einem positiv gewichteten Graph deutlich zu machen, werden die Routensuchverfahren von Moore und Dijkstra ausführlich beschrieben. Als exemplarisches Beispiel und zur Demonstration der Einfachheit seines Matrixalgorithmus wird schließlich das Routensuchverfahren vom Floyd erläutert.

Zu jedem Algorithmus ist ein möglichst allgemeinverständlicher Quellcode angegeben. Es werden jeweils seine spezifischen Aufgaben, die Ein− und Ausgabe, die notwendigen Datenstrukturen, eventuelle Besonderheiten und sein Zeit− und Speicheraufwand erläutert. Alle Verfahren werden anhand eines immer wiederkehrenden Beispiels erklärt.

4.2.1. Ungewichtete Graphen

Im traditionellen Sinne stellt die Suche eines kürzesten Weges in einem ungewichteten Graph kein Routensuchverfahren dar. Weil jedoch mit solch einer relativ einfachen Breitensuche in linearer Zeit gezeigt werden kann, ob überhaupt ein Weg zwischen zwei Knoten existiert, hat so ein Algorithmus bei der Lösung von Topologieproblemen eine große Bedeutung. Die Grundlagen für die Breitensuche hat Moore in [Mo57] gezeigt.

4.2.1.1. Breitensuche nach Moore

Die Aufgabe der Breitensuche ist die Suche minimaler Abstände von Knoten in einem ungewichteten Graph. Gesucht sind also Wege von einem Startknoten zu einem Endknoten, die möglichst wenig weitere Knoten enthalten. Hilfreich und ebenfalls korrekt ist die Vorstellung von Baumalgorithmen für einen gewichteten Graph, bei dem alle Kanten gleich bewertet sind.

Im allgemeinen Fall benötigt der Breitensuchalgorithmus als Eingabe lediglich einen Graph $G=(V;E)$. Das Ergebnis besteht aus einem Wald mit gerichteten Kanten, in dem jeder Baum die minimalen Abstände der Knoten einer Zusammenhangskomponente beschreibt, und einer Numerierung zu den Knoten, um den Wald in der richtigen Reihenfolge zu durchlaufen. Ist, wie in dem Beitrag von Moore, nur der minimale Abstand zwischen genau zwei Knoten des Graphen gefragt, so muß dem Algorithmus als zusätzliche Eingabe der Start− und der Endknoten angegeben werden. Der Algorithmus liefert dann einen Baum mit dem Startknoten als Wurzel. Entspricht eines der Blätter dem Endknoten, so war die Suche nach dem geringsten Abstand erfolgreich, ansonsten sind Start− und Endknoten nicht in einer Zusammenhangskomponente.

Breitensuche nach Moore

Eingabe: Graph G(V; E)
Ausgabe: Menge "Breitensuchwald" mit Kanten zur Beschreibung der Spannbäume
Numerierung b(v) zu den Knoten aus V zur Beschreibung der Reihenfolge (der Ebenen) in den Bäumen

```
/*..INITIALISIERUNG.....................................................*/
  Breitensuchwald = {};   /* Spannbaume als Ergebnis der Breitensuche
  Kandidatenmenge = {};   /* Knoten, die bereits gefunden wurden
  i               = 0;    /* Baumebene = Abstand Knoten vom Wurzelknoten

  Markiere alle Knoten v mit "nicht erreicht" und
  ordne ihnen die Numerierung(v) = ∞ zu;

/*..SCHLEIFE über die Wurzeln aller Spannbäume der Breitensuche........*/
  Schleife, solange die Kandidatenmenge leer ist und
  noch ein Knoten mit der Numerierung(v) = ∞ existiert:
    i = 0;
    Füge einen Knoten v mit Numerierung(v) = ∞ in die Kandidatenmenge ein;
   /* Soll ein Baum von einem bestimmten Knoten aus aufbebaut werden,
      so muss v an dieser Stelle genau dem Startknoten entsprechen */
    Markiere diesen Knoten v mit "erreicht";

/*..SCHLEIFE, um Kandidaten zu nummerieren & ihre Nachfolger zu finden.*/
    Schleife, solange die Kandidatenmenge nicht leer ist:
      Nehme irgendeinen Knoten v aus der Kandidatenmenge heraus;
      Numerierung(v)=i; i=i+1;

/*..SCHLEIFE, um Nachfolgeknoten zu finden und Baum weiter aufzubauen..*/
      Schleife über alle Nachbarknoten w vom Knoten v,
      die noch nicht mit "erreicht" markriert sind:
        Füge die Kante (v,w) in den Breitensuchwald ein;
        Füge den Nachbarknoten w in die Kandidatenmenge ein;
        Markiere den Nachbarknoten w mit "erreicht";
      Schleifenende;
    Schleifenende;
  Schleifenende;
```

Algorithmus 1: Breitensuche in einem ungewichteten Graph nach Moore

Als Ergebnis sind nur die Ebenen der Breitensuchbäume interessant. So haben alle Knoten einer Ebene *i* denselben minimalen Abstand *i* zum Wurzelknoten. Während die Zuordnung der Knoten zu einer Ebene in jedem Fall gleich bleibt, sind die verbindenden Kanten in einem begrenzten Rahmen willkürlich angeordnet. Die Begrenzung liegt darin, daß im Breitensuchbaum nur Kanten innerhalb einer Ebene oder zwischen zwei direkt übereinanderliegenden Ebenen auftreten können.

Die Datenstruktur des Graphen G kann beliebig sein (etwa eine Adjazenzmatrix). Die Menge der Kanten, die zum Wald gehören, und die Menge der Knoten mit Markierung und Numerierung werden vorteilhaft als Schlange implementiert; daß heißt, als Liste, bei der an beiden Enden Elemente hinzugefügt oder entfernt werden können.

Der Algorithmus besteht aus drei ineinander geschachtelten Iterationsschleifen. Die äußere Schleife läuft über die Anzahl der möglichen Spannbäume des Waldes. Im Fall, daß nur ein bestimmter minimaler Abstand gesucht wird, wird diese Schleife nur einmal durchlaufen. Die zweite Schleife läuft solange, bis ein minimaler Spannbaum des ungewichteten Graphen aufgebaut ist. In jedem Durchlauf wird einer der zuletzt markierten Knoten (der nächsthöheren Ebene im Baum) ausgewählt und als Ausgangsknoten für die Markierung weiterer Knoten an die innerste Schleife übergeben. Diese dritte Schleife sucht die noch nicht markierten adjazenten Knoten zu diesem Ausgangsknoten und fügt die entsprechenden inzidenten Kanten in den Breitensuchwald ein.

Betrachtet man den Verlauf der Breitensuche anhand eines konkreten Beispiels, wie in Abbildung 13, wo der Weg mit minimalem Abstand vom Startknoten *a* zum Endknoten *d* gesucht wird, so wird einem das Prinzip des Algorithmus recht schnell klar:

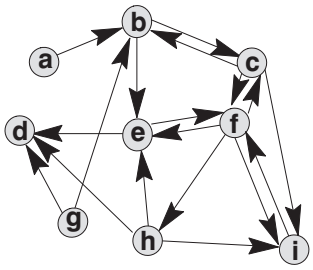
Der Startknoten *a* wird mit 0 markiert und in die Kandidatenmenge gefügt. Als einziger Kandidat wird der Knoten *a* wieder herausgenommen (er repäsentiert dann die Knoten der Stufe 0 im Breitensuchbaum) und ist Ausgangsknoten für die Suche der Kandidaten der Ebene 1. Der einzige adjazente Knoten, der noch nicht markiert wurde, ist im Fall des Knoten *a* nur Knoten *b*. Dieser wird also mit 1 markiert und in die Kandidatenmenge aufgenommen. Gleichzeitig wird die Kante *(a,b)* in den Breitensuchbaum gefügt. Die adjazenten Knoten zu *b* sind *c* und *e*. Sie werden mit 2 markiert und in die Kandidatenmenge gefügt. Mit den Einfügen der Kanten *(b,c)* und *(b,e)* erreicht der Baum die Ebene 2. Würde im nächsten Schritt aus der Kandidatenmenge zuerst der Knoten *e* gewählt und seine adjazenten Knoten *d* und *f* gefunden, so könnte der Algorithmus für die Suche des Weges von *a* nach *d* mit dem Einfügen der Kante *(e,d)* in den Baum beendet werden.

In Abbildung 13 wurde zufällig zunächst der Kandidatenknoten *c* ausgewählt. Es ist erkennbar, daß in diesem Fall die Kante *(c,f)* einfügt wird. Wäre *e* zuerst gewählt worden, so wäre die *Kante (e,f)* eingefügt worden. Die Struktur, die die Kanten des Baumes bilden, ist also willkürlich, bis auf die Beschränkung, daß die Kanten nur aus Knoten einer Ebene oder zweier direkt übereinanderliegenden Ebenen bestehen können. Daß *d*, *f* und *i* die Knoten der Ebene 3 sind, wird sich in keinem Fall der Breitensuche ändern.

Eine allgemeine Breitensuche würde als Element der vierten Ebene noch den Knoten *h* einfügen. Da *g* nicht von *a* erreicht werden kann, ist der Graph nicht streng zusammen-

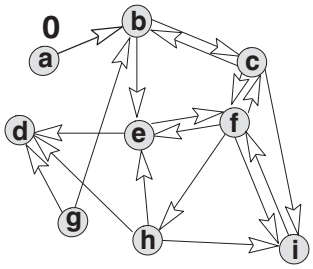
hängend. Der allgemeine Breitensuchalgorithmus würde daher noch einen zweiten Baum mit der Wurzel *g* und den beiden Blättern *b* und *d* konstruieren.

Breitensuche nach Moore



Aufgabe:
Gegeben seien ein Anfangsknoten (a) und ein Endknoten (d) in einem ungewichteten Graph, zwischen denen der Weg mit den wenigsten Zwischenknoten gesucht wird.

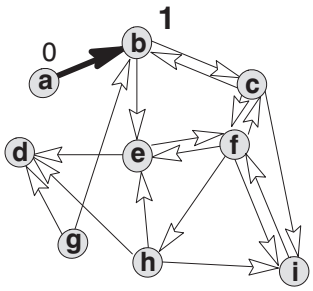
Vorsicht!
Diese Aufgabenstellung beschreibt keine komplette Breitensuche.
Für eine kompette Breitensuche müßte der Algorithmus in diesem Beispiel jedoch nur noch drei Schritte weiter laufen. Er hätte dann auf der vierten Stufe des Baumes den Knoten (f) angehängt und einen zweiten Spannbaum mit der Wurzel (g) und den Blättern (d) und (b) aufgebaut.



Kandidatenmenge: a

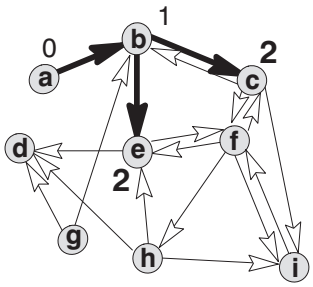
Ebene / Breitensuchbaum:

Ebene	
0	a



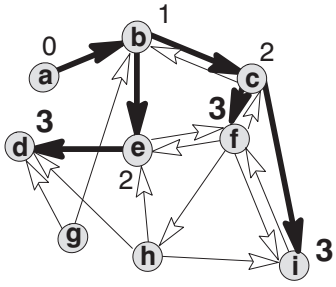
Kandidatenmenge: b

Ebene	
0	a
1	b



Kandidatenmenge: e, c

Ebene	
0	a
1	b
2	e c



Kandidatenmenge: i, d, f

Ebene	
0	a
1	b
2	e c
3	d f i

Bild 13: Beispiel zur Breitensuche nach Moore

Die allgemeine Breitensuche fügt jeden Knoten des Graphen nur einmal irgendwo in den Breitensuchwald ein. Da alle abgehenden Kanten der Knoten einmal verwendet werden, um eventuelle neue Kandidaten zu finden, ist der Rechenaufwand für diesen Algorithmus $O(|V|+|E|)$. Daß heißt, es werden insgesamt soviele Schritte durchgeführt, wie Knoten und Kanten im Graph enthalten sind. Die Breitensuche findet also mit einem optimalen linearen Zeitaufwand $O(n)$ eine Lösung. Der Speicheraufwand ist durch die beiden Warteschlangen minimal.

4.2.2. Kürzeste Wege von einem Knoten aus

Die Breitensuche entspricht der Bestimmung kürzester Wege in einem gewichteten Graph, bei dem jede Kante genau gleich bewertet wird. Aus der Gleichbewertung ergibt sich ein großer Vorteil für das Breitensuchverfahren: Ist ein Knoten einmal gefunden worden, so ist der Abstand zum Startknoten, zur Wurzel des Breitensuchbaums, minimal. Da alle Kantengewichtungen gleich sind, ist es unmöglich im weiteren Verlauf des Algorithmus mit Knoten die noch nicht markiert sind, einen Umweg zu finden, der einen geringeren Abstand zum Startknoten hat. Dieser Umweg müßte über Knoten "tieferer" Ebenen (mit höherer Numerierung) führen. Da die Numerierung der Ebenen identisch mit dem minimalen Abstand zum Startknoten sind, ist dies unmöglich.

In den Algorithmen zur Bestimmung kürzester Wege in Graphen, deren Kanten beliebig gewichtet werden können, kann nicht ausgeschlossen werden, daß bisher gefundene minimale Wege durch andere verkürzt werden. Da die Ebenen im Baum kürzester Wege nicht mehr identisch mit dem Abstand zum Startknoten sind, können Umwege über Knoten tieferer Ebenen zu optimalen Wegen führen (Abbildung 15 rechts).

Die lineare Rechenzeit des Breitensuchalgorithmus wird somit bei einem Algorithmus zur Bestimmung kürzester Wege nur in Sonderfällen erreichbar sein. Ein solcher Sonderfall liegt zum Beispiel vor, wenn der Graph azyklisch ist. In diesem (für das Verkehrswesen uninteressanten) Fall erreicht man wieder die Zeitkomplexität $O(|V|+|E|)$.

Im folgenden wird der Algorithmus von Moore als ein "Iterationsverfahren" und der Algorithmus von Dijkstra als ein "iterationsloses Verfahren" untersucht. Beide Algorithmen erkennen keine negative Zyklen und sind somit auf positiv gewichtete Graphen beschränkt. Ein Verfahren, das negative Zyklen erkennt, also für beliebig gewichtete Graphen funktioniert, ist der Algorithmus von Bellman−Ford (siehe [Ju94], 113). Da im Verkehrswesen fast ausschließlich positive Gewichtungen verwendet werden, wird er hier nicht erläutert.

4.2.2.1. Algorithmus von Moore für kürzeste Wege von einem Knoten aus

Das Routensuchverfahren von Moore berechnet die kürzesten Wege von einem Startknoten zu allen anderen Knoten eines positiv gewichteten Graphen.

Als Eingabe benötigt der Algorithmus einen Graph $G=(V; E)$, eine positive Kantengewichtungsfunktion $l : E \rightarrow \mathbb{R}^+$ sowie einen Startknoten aus V. Das Ergebnis besteht aus den beiden Listen für die Kanten des Baumes kürzester Wege vom Startknoten aus und für die minimalen Abstände vom Startknoten zu allen anderen.

Als verwendete Datenstruktur für den Graph ist wie bei allen Baumalgorithmen eine Adajzenzliste am vorteilhaftesten. Die Kandidatenmenge kann als einfache Liste implementiert werden, da ein neuer Kandidat beliebig gewählt wird. Die Liste für die minimalen Abstände hat die konstante Länge von $|V|$, da die Abstände für alle Knoten angegeben werden. Die Datenstruktur für den Baum kürzester Wege ist uninteressant, da sie hier nur zur Verdeutlichung im folgenden Beispiel dient. Normalerweise wird der Weg durch die bereits in Abschnitt 4.1.3 beschriebene Rückrechnung bestimmt.

Kürzeste Wege nach Moore

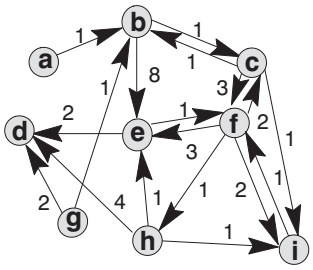
Eingabe: Graph G(V; E) mit positiver Kostenfunktion l: $E \rightarrow \mathbb{R}^+$
Anfangsknoten v1 aus der Menge V
Ausgabe: Menge "KuerzesteWegeBaum" mit Kanten des Baumes kürzester Wege
Liste d(v1, ..., vn), die die Abstände vom Anfangsknoten enthält

```
/*..INITIALISIERUNG.................................................*/
  KuerzesteWegeBaum = {};     /* Baum der kuerzesten Wege als Ergebnis
  Kandidatenmenge   = {};     /* Knoten, die bereits gefunden wurden

  Füge den Anfangsknoten v1 in die Kandidatenmenge;
  Ordne dem Anfangsknoten den Abstand d(v1) = 0 zu;
  Ordne alle übrigen Knoten den Abstand d(v) = ∞ zu;

/*..SCHLEIFE, um die Distanz der Kandidaten & ihre Nachfolger zu finden*/
  Scheife, solange die Kandidatenmenge nicht leer ist:
    Nehme irgendeinen Knoten v aus der Kandidatenmenge heraus;

/*..SCHLEIFE, um Nachfolgeknoten zu finden und Baum weiter aufzubauen..*/
    Schleife über alle Nachbarknoten w von Knoten v:
      Füge den Nachbarknoten w in die Kandidatenmenge;

      Wenn d(w) > d(v) + l(v,w), dann:
        d(w) = d(v) + l(v,w);
        Füge die Kante (v,w) in den KuerzesteWegeBaum;
        Nehme ggf. den vorherigen Weg zum Knoten w rückwärts bis zu
       einem Verzweigungspunkt aus dem KuerzesteWegeBaum wieder heraus;

    Schleifenende;
  Schleifenende
```

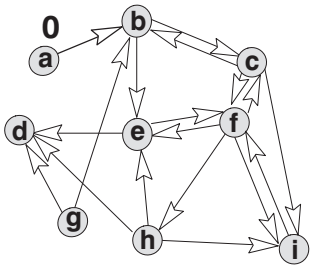
Algorithmus 2: Moore: Bestimmung der kürzesten Wege von einem Knoten aus

Der Algorithmus von Moore zur Bestimmung kürzester Wege von einem Knoten aus verläuft im Prinzip analog zum Breitensuchalgorithmus. Da hier in jedem Fall nur ein Baum aufgebaut werden soll, besteht er aus zwei Schleifen. Die äußere Schleife für eventuelle weitere Bäume fällt weg. Ansonsten unterscheiden sich die beiden Algorithmen explizit nur dadurch, daß im Gegensatz zur Breitensuche die Markierung der Knoten nicht durch eine fortlaufende Numerierung mit normalen Zahlen geschieht, sondern durch das Verbessern der Markierungen (die zunächst für alle Knoten alle auf ∞ gesetzt werden) mit Hilfe der Bellman−Gleichungen.

Moore: Kürzeste Wege von einem Knoten aus

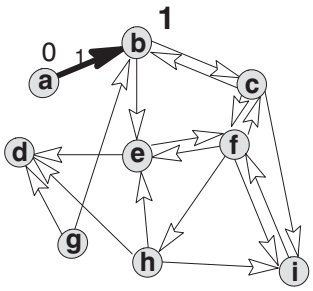


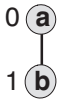
Aufgabe:
Gegeben seien ein Anfangsknoten (a) und ein Endknoten (d) in einem gewichteten Graph, zwischen denen der Weg mit der minimalen Gesamtbewertung gesucht werden soll.

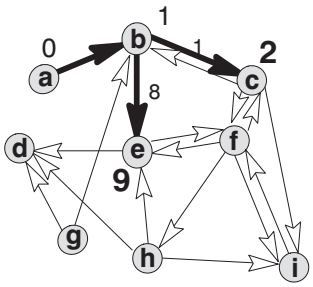


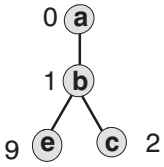
Kandidatenmenge

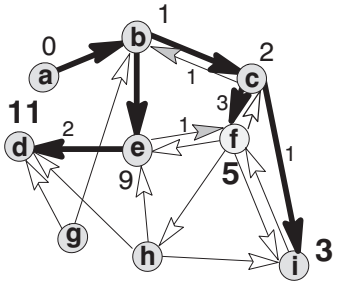
KürzesteWegeBaum

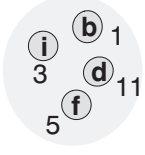












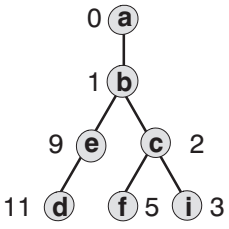


Bild 14: Beispiel zu Moores Bestimmung der kürzesten Wege von einem Knoten aus (1. Teil)

Jeder neu zu betrachtende Nachbarknoten eines Elementes der Kandidatenmenge erfordert einen eigenen Rechenschritt. In dieser Darstellung erfolgt eine Zusammenfassung bei der Bearbeitung der Nachbarknoten aller momentaner Kandidatenknoten, sodaß die Anzahl der Schritte sehr viel größer ist, als dies hier scheint.

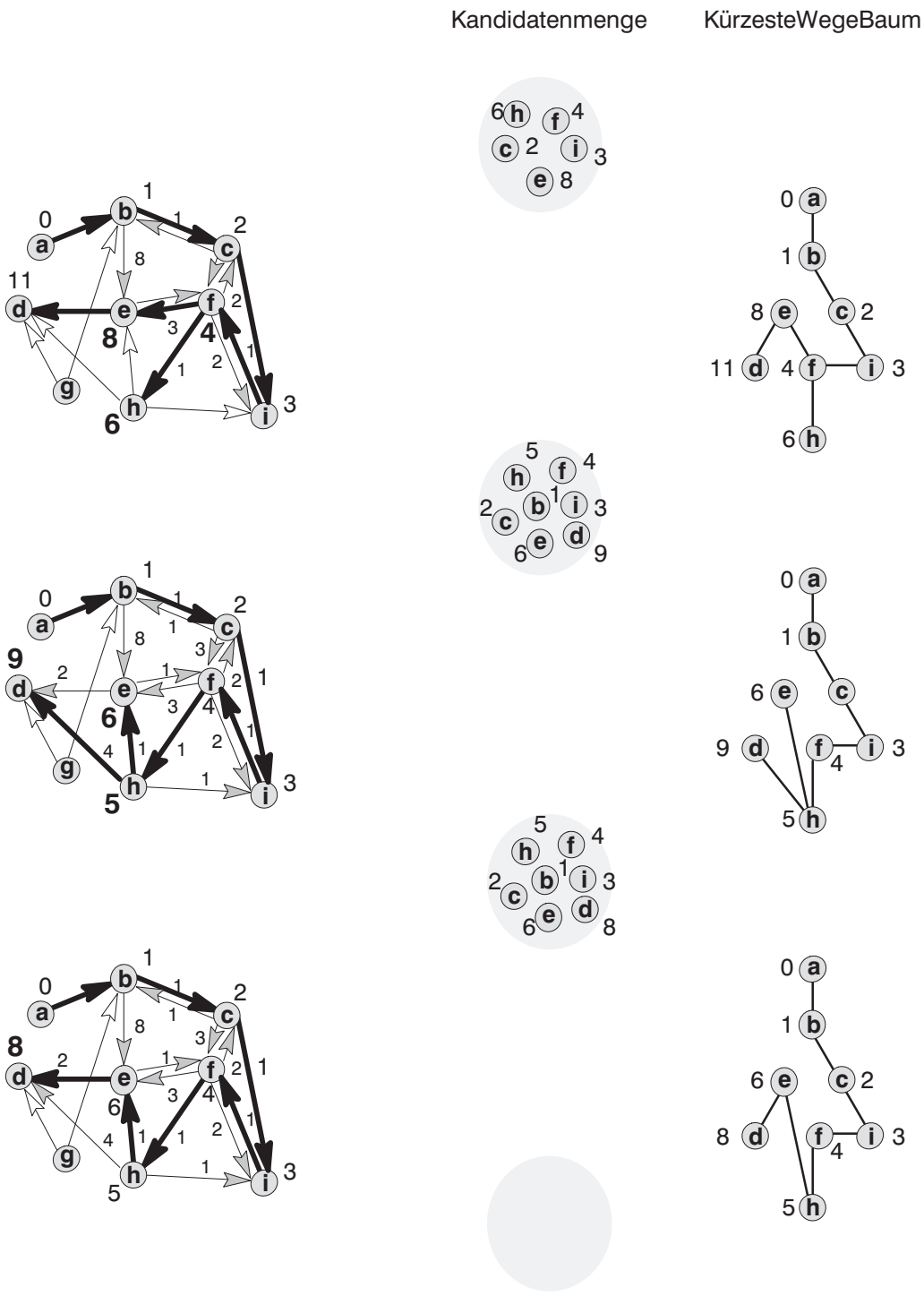


Bild 15: Beispiel zu Moores Bestimmung der kürzesten Wege von einem Knoten aus (2. Teil)

Implizit besteht ein Unterschied im sehr unterschiedlichen Laufzeitverhalten. Durch die unterschiedliche Kantenbewertung kann im Kürzeste−Wege−Algorithmus bei beliebiger Wahl des nächsten Kandidatenknoten, nicht mehr garantiert werden, daß später gewählte Knoten zu einem günstigeren Weg führen, und daß es somit zu einer ständigen Wiederholung der Verbesserung der schon als fertig erkannter Knoten kommt.

In dem Beispiel der Abbildungen 14 und 15 werden das Laufzeitverhalten und die erläuterten Unterschiede deutlich:

Bis zum Einfügen der Kanten *(a,b)* und *(b,c)* verhält sich das Routensuchverfahren für gewichtete Graphen wie die Breitensuche, da die Kanten mit *1* bewertet wurden. Beim Einfügen der Kante *(b,e)* erkennt man die hohe Markierung *9* für den Knoten *e*, die durch die Addition der Gewichtung von *(b,e)* und der Markierung von *b* entstanden ist (*8+1=9*). Unten auf Abbildung 14 erkennt man rechts, bis auf die unterschiedlichen Bewertungen der Knoten, den gleichen Baum. Im Graphen links erkennt man, daß unter anderem auch die Kanten *(c,e)* und *(e,f)* untersucht wurden, aber zu keiner Verbesserung der Markierung von *b* oder *f* geführt haben (graue Pfeile). Da der Knoten *d* erreicht wurde, scheint es, daß der Algorithmus beendet werden könnte. Dies ist aber nicht der Fall, da *11* nicht die minimale Weglänge von *a* nach *d* ist, wie es im weiteren Verlauf des Algorithmus deutlich wird.

Ohne auf die Einzelheiten von Abbildung 15 einzugehen, können einige Punkte festgestellte werden:

1. Die Kandidatenmenge wird mit der Zeit immer größer, da immer mehr Verbesserungen in den schon durchsuchten Bereichen gefunden werden.
2. Der Baum kürzester Wege weicht stark von der klaren Einteilung ab, die bei der Breitensuche zu erkennen war. Es treten Kanten mit Knoten zweier weit entfernter Ebenen auf und es ergeben sich Wege von tieferen zu höheren Ebenen.
3. Die Strukturen, die die Kanten des Baumes in jedem Schritt erzeugen, variieren stark.
4. Der lineare Zeitaufwand, der am Anfang der Abbildung 15 bereits erreicht war, wird weit überschritten. Die Zeitkomplexität beträgt $O(|V| \cdot |E|)$, wobei $|E|$ die Anzahl der Kanten ist.
5. Der Algorithmus wird erst beendet, wenn keine Verbessung mehr möglich ist.

Bis auf den Punkt 2, der jedem Routensuchverfahren für beliebig gewichtete Graphen zugeorndet werden kann, zeigen die Punkte das Verhalten eines sogenannten "Iterationsverfahren": Durch die willkürliche Wahl des nächsten Kandidatenknoten können ungünstige Kombinationen, die zu ständigen Verbesserungen eines Großteils des Graphen führen, nicht ausgeschlossen werden. In unserem Beispiel muß erst der Knoten *h* von *i* über *f* aus markiert sein, um den minimalen Abstand von *e* und *d* berechnen zu können. Da *h* erst sehr spät gefunden werden kann, optimiert der Algorithmus in der Zeit bis zum Finden von *h* unnötigerweise beliebig andere Bereiche im Graph.

Im ungünstigsten Fall ist der Graph ein dichter Graph, somit $|E|$ fast $|V|^2$ und die Zeitkomplexität für den Gesamtalgorithmus fast $|V|^3$. Sollte der Algorithmus zufällig die

Kandidaten in der richtigen Reihenfolge auswählen, so würden, wie bei der Breitensuche, genau $|V|+|E|$ Schritte benötigt, also eine lineare Zeitkomplexität *O(n)*. Dies ist bei den großen Kandidatenmengen der "Iterationsverfahren" kaum der Fall. Bei den "Iterationslosen Verfahren" ist dieser Zufall der richtigen Kandidaten−Reihenfolgen der Regelfall, da immer der Kandidatenknoten mit dem geringsten Abstand zum Startknoten gewählt wird. Der Algorithmus von Dijkstra stellt ein solches "Iterationsloses Verfahren" dar.

4.2.2.2. Algorithmus von Dijkstra für kürzeste Wege von einem Knoten aus

Das Routensuchverfahren von Dijkstra berechnet, ebenso wie das von Moore, die kürzesten Wege von einem Startknoten zu allen anderen Knoten eines positiv gewichteten Graphen. Auch die Ein− und Ausgabe ist identisch zum Algorithmus zur Bestimmung der kürzesten Wege nach Moore. Nur die Datenstruktur der Kandidatenmenge unterscheidet sich. Wo bei Moore eine einfache Liste genügte, ist jetzt eine Prioritätsschlange notwendig, bei der immer auf das Element mit der höchsten Priorität zugegriffen wird. Im Grundalgorithmus von Dijkstra wird dabei eine verketten Liste verwendet.

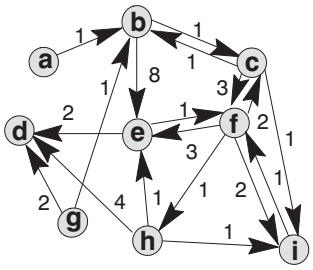
Kürzeste Wege nach Dijkstra

Eingabe: Graph G(V; E) mit positiver Kostenfunktion l: E→$\mathbb{R}^+$
Anfangsknoten v1 aus der Menge V
Ausgabe: Menge "KuerzesteWegeBaum" mit Kanten des Baumes kürzester Wege
Liste d(v1, ..., vn), die die Abstände vom Anfangsknoten enthält

```
/*..INITIALISIERUNG.....................................................*/
  KuerzesteWegeBaum = {};     /* Baum kuerzester Wege als Ergebnis
  Kandidatenliste   = {};     /* Sequenzliste der Kandidatenknoten
  Ausgangsmenge     = V;      /* Knoten, die noch nicht gefunden wurden

  Ueberfuehre den Anfangsknoten v aus der Ausgangsmenge
  in die Kandidatenliste und ordne ihm d(v)=0 zu;
  Ordne alle übrigen Knoten d(v) = ∞ zu;

/*..SCHLEIFE, um den kuerzesten Weg um jeweils ein Knoten zu erweitern.*/
  Scheife, solange die Kandidatenliste nicht leer ist und
  der Endknoten noch nicht erreicht:
    Nehme einen Knoten v aus Kandidatenliste, für den d(v) minimal ist;

/*..SCHLEIFE, um die Nachbarknoten des aktuellen Knoten abzuarbeiten...*/
    Schleife über alle Nachbarknoten w von Knoten v:

      Liegt w in der Ausgangsmenge,
        so ist d(w) = l(v,w) und fuege w in die Kandidatenliste;
      Liegt w in der Kandidatenliste,
        so ist d(w) = min {d(w), d(v) + l(v,w)};
      Im Falle eines neuen kuerzesten Weges, ersetze den alten
      KuerzesteWegeBaum durch den neuen mit der Kante (v,w);

    Schleifenende;
  Schleifenende
```

Algorithmus 3: Dijkstra: Bestimmung der kürzesten Wege von einem Knoten aus (traditionell)

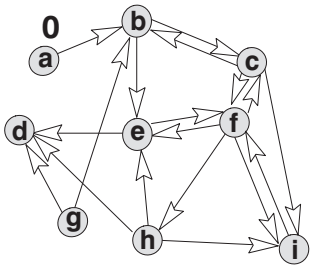
Dijkstra: Kürzeste Wege von einem Knoten aus



Aufgabe:
Gegeben seien ein Anfangsknoten (a) und ein Endknoten (d) in einem gewichteten Graph, zwischen denen der Weg mit der minimalen Gesamtbewertung gesucht werden soll.

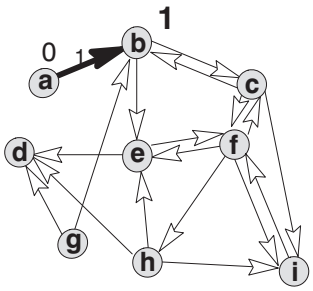
Kandidatenliste

KürzesterWegeBaum



a
0

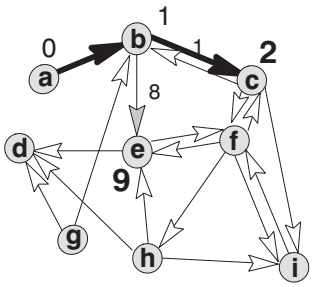
0 a



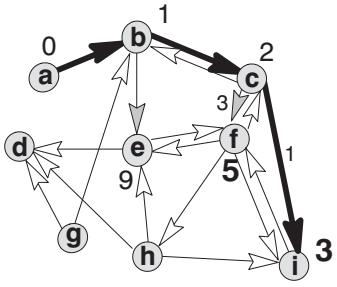
b
1

0 a
1 b

c e
2 9



0 a
1 b
c 2



i f e
3 5 9

0 a
1 b
c 2
i 3

Bild 16: Beispiel zu Dijkstras Bestimmung der kürzesten Wege von einem Knoten aus (1. Teil)

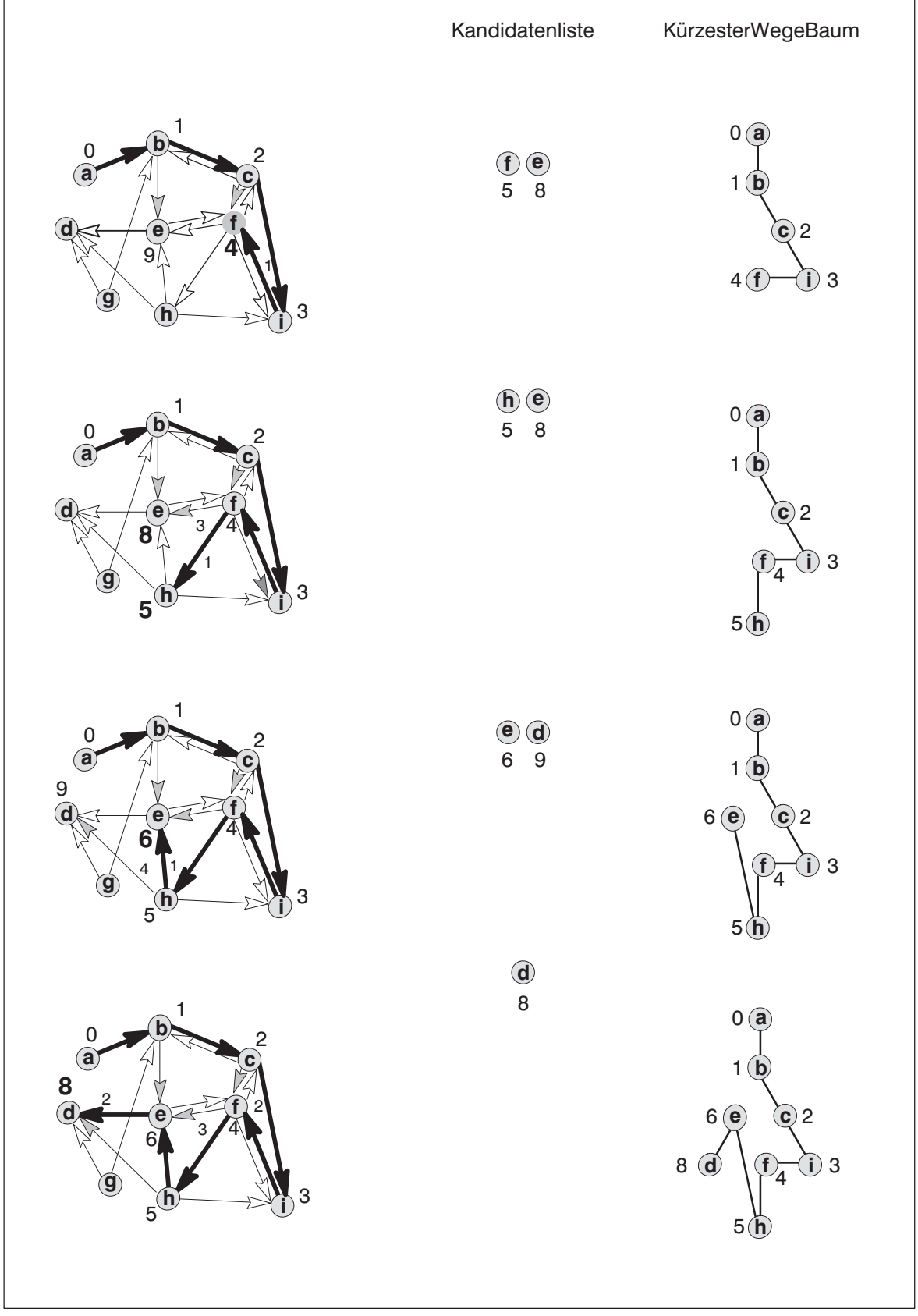


Bild 17: Beispiel zu Dijkstras Bestimmung der kürzesten Wege von einem Knoten aus (2. Teil)

Der Algorithmus von Dijkstra unterscheidet sich von Moores Routensuchverfahren dadurch, daß seine Kantenmenge eine dynamische Kantenliste minimaler Größe ist, aus der jeweils der Knoten herausgenommen wird, der den geringsten Abstand zum Startknoten hat. Außerdem wird die Ausgangsmenge explizit betrachtet, was aber zu keinem weiteren Speicherplatzverbrauch führt, da diese Menge im Laufe des Algorithmus mit dem gleichen Maß abnimmt, wie die Menge der bereits fertigen Knoten zunimmt.

Hieraus ergeben sich zwei Vorteile. Zum einen ist mit dem Entfernen des Knoten aus der Kandidatenliste, der den geringsten Abstand zum Startknoten hat, ebenso wie bei der Breitensuche sichergestellt, daß dieser Knoten nicht weiter betrachtet werden muß. Ein mehrmaliges Iterieren, wie bei der Routensuche von Moore, entfällt somit. Zum anderen wird durch die Implementierung der Kandidatenmenge als dynamische Liste, die nur noch die zu untersuchenden Kandidaten beinhaltet, minimaler Speicherplatz benötigt und eine schnelle Suche des Knoten mit dem geringsten Abstand zum Knoten garantiert.

Betrachtet man das Beispiel der Abbildungen 16 und 17 zur Beschreibung des Verhaltens des Algorithmus von Dijkstra im Vergleich zum bereits bekannten Beispiel für Moores Routensuchverfahren in den Abbildungen 14 und 15, so erkennt man schon im dritten Schritt, wie ”zielstrebig” Dijkstras Algorithmus den kürzesten Weg verfolgt. Bis zum Erreichen des Endknoten *d* wird in jedem Schritt der korrekte Baum kürzester Wege bis zum dem Knoten konstruiert, der zuletzt aus der Kandidatenmenge genommen wurde. Spätere Veränderungen des Baumes, wie bei Moore, treten nicht auf.

In den Lehrbüchern zur Graphentheorie, wie etwa in [Ju94] oder [Br94] findet man häufig eine Darstellung des Algorithmus, die nach wie vor der Originalbeschreibung von Dijkstra in [Di59] entsprechen, aber wesentlich kürzer sind. Das Nachvollziehen dieser Varianten ist etwas komplizierter, weil die leichtverständliche Einteilung in die jeweils drei disjunkten Knoten− und Kantenmengen nicht mehr erkennbar ist.

Neuere Darstellung des Algorithmus von Dijkstra

Eingabe: Graph G(V; E) mit positiver Kostenfunktion l: E→$\mathbb{R}^+$
Anfangsknoten v1 aus der Menge V
Ausgabe: Liste d(v1, ..., vn), die die Abstände vom Anfangsknoten enthält

```
/*..INITIALISIERUNG.....................................................*/
  Ausgangsliste = V;      /* Knoten, die noch nicht gefunden wurden
  Ordne dem Anfangsknoten v den Abstand d(v) = 0 zu;
  Ordne alle übrigen Knoten den Abstand d(v) = ∞ zu;

/*..SCHLEIFE, um den kuerzesten Weg um jeweils einen Knoten zu erweitern.*/
  Scheife, solange die Ausgangliste nicht leer ist und
  der Endknoten noch nicht erreicht:
    Nehme einen Knoten v aus der Ausgangsliste, für das d(v) minimal ist;
    Für alle Nachbarknoten w von Knoten v in der Ausgangsliste ist der
    Abstand d(w) = min {d(w), d(v) + l(v,w)};
  Schleifenende
```

Algorithmus 4: Dijkstra: Bestimmung der kürzesten Wege von einem Knoten aus (neuere Version)

Da die günstigste Reihenfolge der Kandidatenknoten nicht, wie bei der Breitensuche implizit mit Erreichen eines neuen Knoten gegeben ist, sondern erst mit einer Minimumsuche gefunden werden muß, wird die lineare Rechenzeit von $|V|+|E|$ Schritten nicht erreicht. Das Suchen des nächsten Kandidaten in der einfachen Prioritätsschlange erfordert einen Mehraufwand von höchstens $|V|$ Schritten. So ergibt sich ein Gesamtaufwand mit der Zeitkomplexität von $O(|V|^2+|E|) = O(|V|^2)$.

4.2.3. Kürzeste Wege zwischen je zwei Knoten

Mit der Bestimmung kürzester Wege zwischen je zwei Knoten wird ein Wald mit n Bäumen kürzester Wege von jeweils einem der n Knoten des Graphen berechnet. Das heißt, um alle möglichen optimalen Wege eines Graphen zu bestimmen, muß lediglich für jeden Knoten ein Baumalgorithmus aufgerufen werden. Matrixalgorithmen, die in einem Algorithmus alle kürzesten Wege eines Graphen berechnen, lassen sich durch ihre Anschaulichkeit und ihre für dichtbesetzte Graphen fast optimale Speicherplatzausnutzung rechtfertigen. Der herausragende Algorithmus dieser Art ist der von Floyd.

Wegealgebren, die es erlauben, fast so, wie in den allgemein bekannten Algebren, mit den Strukturmatrizen zu arbeiten, als wäre sie Bestand eines Gleichungssystems, werden kurz im dann folgenden Abschnitt erläutert.

4.2.3.1. Algorithmus von Floyd für kürzeste Wege zwischen je zwei Knoten

Floyds Algorithmus berechnet die minimalen Abstände aller Knoten eines gewichteten Graphen in $|V|^3$ Schritten. Benötigt wird nur der Graph in Form der Strukturmatrix. Innerhalb des Algorithmus wird die eingegebene Strukturmatrix durch direktes Ändern der Einträge in eine Ausgabematrix mit den minimalen Abständen zwischen je zwei Knoten umgewandelt. Dies geschieht durch drei ineinander verschachtelte Schleifen.

Kürzeste Wege zwischen je zwei Knoten (Floyd–Warshall)

```
procedure shortest path (m,n); value n; interger n; array m;
comment  Initially m[i,j] is the length of a direct link from point i of
a network to point j. If no direct link exists, m[i,j] is initially 1010.
At completion, m[i,j] is the length of the shortest path from i to j. If
none exists, m[i,j] is 1010. Reference: Warshall, S. A Theorem on Boolean
matrices. J, ACM 9(1962), 11–12
begin
integer i,j,k; real inf,s; inf:=1010;
for i:=1 step 1 until n do
for j:=1 step 1 until n do
if m[j,i] < inf then
for k:=1 step 1 until n do
if m[i,k] < inf then
begin s:= m[j,i] + m[i,k];
if s < m[j,k] then m[j,k]:=s
end
end shortest path
```

komplett zitiertes Original [Fl62]

Algorithmus 5: Floyd–Warshall: Bestimmung der kürzesten Wege zwischen je zwei Knoten

Im Beispiel von Abbildung 18 kann man das Durchlaufen der Schleifen wie folgt erkennen: Die beiden äußeren Schleifen i und j sorgen dafür, daß die Einträge der Matrix spaltenweise von links oben nach rechts unten nacheinander betrachtet werden (graues Band). Zu jedem betrachteten Eintrag wird mit der inneren Schleife k die entsprechende Zeile (die adjazenten Knoten) durchlaufen. Jeder Eintrag x dieser Zeile wird zu jedem Eintrag y in der Spalte des betrachteten Knoten addiert und die entstehende Summe $x+y$ an die Stelle der Spalte von x und der Zeile von y geschrieben, falls der dort schon vorhandene Eintrag nicht kleiner ist.

Als Beispiel wird hier die Berechnung des Abstandes 9 der Kante *(a,e)* im Matrixzustand rechts oben in der Abbildung 18 erläutert: Die 0 unter dem grauen Band stellt den Eintrag des aktuell betrachteten Elementes dar. Der Eintrag 9 der Kante *(a,e)* ergibt sich aus der Addition der 8 in der Zeile und der 1 in der Spalte der betracheten 0. Anstatt der 9 betrug der Eintrag für *(a,e)* vorher ∞. Er wurde also (mit den Bellman−Gleichungen) verbessert.

4.2.3.2. Wegealgebren

Die Bellman−Gleichungen bilden ein nichtlineares Gleichungssystem. Shimbel hat es geschafft, das Problem der Suche von kürzesten Wegen in einem Graph mit Hilfe der Strukturmatrix algebraisch zu lösen.

Es ist möglich, eine Algebra für Adjazenz− und Strukturmatrixen zu entwickeln, mit denen sich nicht nur kürzeste Wege berechnen lassen, sondern viele weitere topologische Wegeprobleme. Daher heißt diese, besonders von Carré [Ca71] analysierte Algebra "Wegealgebra".

Die Wegprobleme lassen sich wie beim bekannten Lösen von Gleichungssystemen mit Matrixmultiplikationen berechnen. Hierzu können Gleichungslöser verwendet werden.

Grundlage hierfür bildet der Ansatz von Shimbel in Definition (3):

$$x + y = \min(x, y)$$
$$x \cdot y = x + y \text{ (}\textit{im Sinne der bekannten Algebra fuer reele Zahlen}\text{)}$$
$$\infty \cdot x = x \cdot \infty = \infty$$

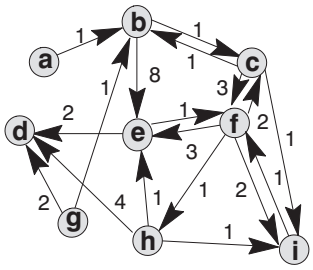
Eine wichtige Einschränkung ist die Nichtkommutativität. Wenn sie nicht gegeben wäre, kõnnen die Regeln der bekannten Matrixalgebra 1:1 auf die Wegealgebra übertragen. So ist zum Beispiel die Berechnung einer inversen Strukturmatrix nicht immer möglich.

Möglich sind jedoch durch Ersetzen der Operatoren "+" und "$\cdot$" der Wegealgebra in folgende Operatoren der booleschen und allgemeinen Matrixalgebra Lösungen mit Hilfe von Gleichungssystemen:

"$\vee$" und "$\wedge$" führen zur Bestimmung der Existenz von Wegen,
"+" und "$\cdot$" zur Anzahl der Wege,
"min()" und "+" zu kürzesten Wegen sowie
"max()" und "+" zu längsten Wegen.

So ist beispielsweise die Bestimmung kürzester Wege auf die Lösung von Gleichungssystemen mit dem Jacobi− oder dem Gauß−Seidel−Verfahren rückführbar.

Floyd–Warshall: Kürzeste Wege zwischen je zwei Knoten



Aufgabe:
Gegeben sei ein gewichteter Graph, für den die kürzesten Wege zwischen allen Knoten bestimmt werden sollen.

	a	b	c	d	e	f	g	h	i
a	0	1	∞	∞	∞	∞	∞	∞	∞
b	∞	0	1	∞	8	∞	∞	∞	∞
c	∞	1	0	∞	∞	3	∞	∞	1
d	∞	∞	∞	0	∞	∞	∞	∞	∞
e	∞	∞	∞	2	0	1	∞	∞	∞
f	∞	∞	2	∞	3	0	∞	1	2
g	∞	1	∞	2	∞	∞	0	∞	∞
h	∞	∞	∞	4	1	∞	∞	0	1
i	∞	∞	∞	∞	∞	1	∞	∞	0

	a	b	c	d	e	f	g	h	i
a	0	1	**2**	∞	**9**	∞	∞	∞	∞
b	∞	0	1	∞	8	∞	∞	∞	∞
c	∞	1	0	∞	∞	3	∞	∞	1
d	∞	∞	∞	0	∞	∞	∞	∞	∞
e	∞	∞	∞	2	0	1	∞	∞	∞
f	∞	∞	2	∞	3	0	∞	1	2
g	∞	1	∞	2	∞	∞	0	∞	∞
h	∞	∞	∞	4	1	∞	∞	0	1
i	∞	∞	∞	∞	∞	1	∞	∞	0

	a	b	c	d	e	f	g	h	i
a	0	1	**2**	∞	**9**	∞	∞	∞	∞
b	∞	0	1	∞	8	∞	∞	∞	∞
c	∞	1	0	∞	**9**	3	∞	∞	1
d	∞	∞	∞	0	∞	∞	∞	∞	∞
e	∞	∞	∞	2	0	1	∞	∞	∞
f	∞	∞	2	∞	3	0	∞	1	2
g	∞	1	**2**	2	**9**	∞	0	∞	∞
h	∞	∞	∞	4	1	∞	∞	0	1
i	∞	∞	∞	∞	∞	1	∞	∞	0

	a	b	c	d	e	f	g	h	i
a	0	1	2	**11**	9	**5**	∞	∞	**3**
b	∞	0	1	**10**	8	**4**	∞	∞	**2**
c	∞	1	0	**9**	9	3	∞	∞	1
d	∞	∞	∞	0	∞	∞	∞	∞	∞
e	∞	∞	∞	2	0	1	∞	∞	∞
f	∞	**3**	2	**5**	3	0	∞	1	2
g	∞	1	2	2	9	**5**	0	∞	**3**
h	∞	∞	∞	**3**	1	**2**	∞	0	1
i	∞	∞	∞	∞	∞	1	∞	∞	0

	a	b	c	d	e	f	g	h	i
a	0	1	2	**9**	**7**	5	∞	**6**	3
b	∞	0	1	**8**	**6**	4	∞	**5**	2
c	∞	1	0	**7**	**5**	3	∞	**4**	1
d	∞	∞	∞	0	∞	∞	∞	∞	∞
e	∞	**4**	**3**	2	0	1	∞	**2**	**3**
f	∞	3	2	5	3	0	∞	1	2
g	∞	1	2	2	**8**	5	0	**6**	3
h	∞	**5**	**4**	3	1	2	∞	0	1
i	∞	**4**	**3**	**6**	**4**	1	∞	**2**	0

	a	b	c	d	e	f	g	h	i
a	0	1	2	**8**	**6**	**4**	∞	**5**	3
b	∞	0	1	**7**	**5**	**3**	∞	**4**	2
c	∞	1	0	**6**	**4**	**2**	∞	**3**	1
d	∞	∞	∞	0	∞	∞	∞	∞	∞
e	∞	4	3	2	0	1	∞	2	3
f	∞	3	2	**4**	**2**	0	∞	1	2
g	∞	1	2	2	**7**	5	0	6	3
h	∞	5	4	3	1	2	∞	0	1
i	∞	4	3	**5**	**3**	1	∞	2	0

Bild 18: Beispiel zu Floyd–Warshalls Bestimmung der kürzesten Wege zwischen allen Knoten

5. Weiterentwicklung der Routensuchverfahren

Mit der Entwicklung der klassischen Routensuchverfahren standen Anfang der sechziger Jahre die gesamten grundsätzlichen Methoden zur Bestimmung kürzester Wege zur Verfügung. In der Weiterentwicklung der Verfahren ging und geht es nur noch um die Verbesserung ihrer Effizienz.

Es gibt prinzipiell drei Möglichkeiten, um die Verfahren von Dijkstra und Floyd, die zur Zeit ihrer Vorstellung die schnellsten Algorithmen zur Bestimmung kürzester Wege darstellten, hinsichtlich ihrer Effizienz zu verbessern. Die erste Möglichkeit besteht in der effizienterern Suche des Kandidaten mit minimalem Abstand innerhalb der "iterationslosen Verfahren". Eine zweite Möglichkeit liegt in der Parallelisierung der Verfahren auf Multiprozessorrechnern.

Schließlich besteht die dritte Möglichkeit in der Ausnutzung vorteilhafter Eigenschaften von speziellen Graphen. So hat beispielsweise Braun [Br80] für den zusammenhängenden, lichten Graphen eines allgemeinen Verkehrsnetzes bei der Aufgabe der Verkehrsumlegung erkannt, daß nur kürzeste Wege zwischen den Knoten für Quellbezirke zu Knoten für Zielbezirke bestimmt werden müssen. Die Konstruktion des Waldes hat Braun beschleunigt, indem er Teilbäume, die in dem Algorithmus bereits bestimmt waren, für die Konstruktion neuer Bäume genutzt hat. Mit dieser sehr aufwendigen Lösung hat Braun ein Routensuchverfahren entworfen, das dreimal schneller war, als die damals bekannten. Da auf nicht absehbare Zeit jedes Jahr für den gleichen Preis fast doppelt so schnelle Rechner wie im Jahr zuvor zu bekommen sind, können solche konstanten Verbesserungen kaum relevant sein. Wichtig ist gerade für große Netze die Optimierung der Zeitkomplexität, da mit ihr beiepielsweise schon bei 1000 Knoten eine Verbesserung um über das 300−fache möglich sein könnte ($O(n^2) \rightarrow O(n \log n)$).

In den folgenden zwei Absschnitten wird die Entwicklung der Effizienzsteigerung von kürzesten−Wege−Algorithmen durch Verbesserung der Zeitkomplexität bei der Minimumsuche und auf Parallelrechner dargestellt. Im Anschluß wird ein Fazit zur grundlegenden Anforderung an die Routensuchverfahren, möglichst effizient zu sein, gezogen.

5.1. Verbesserung der Minimumsuche in Baumalgorithmen

Bei der Analyse der Breitensuche hat sich gezeigt, daß das optimale Laufzeitverhalten von Algorithmen zur Bestimmung kürzester Wege von einem Startknoten aus $O(|E|+|V|)$ ist, und daß dieses Optimum erreicht wird, wenn garantiert werden kann, daß während der Kürzesten−Wege−Suche beim Finden eines neuen Knoten gleichzeitig der minimale Abstand des Knoten vom Startknoten gefunden wird.

Die Vorraussetzung, daß mit dem Erreichen eines Knoten sein minimaler Abstand vom Startknoten aus gefunden wird, ist im speziellen Fall der Breitensuche durch die gleiche Gewichtung aller Kanten erfüllt. Im allgemeinen Fall beliebig positiv gewichteter Kanten wird diese Voraussetzung von "iterationslosen" Baumalgorithmen durch die Suche des minimalen Elementes in der Kandidatenmenge erfüllt.

Um, wie bei der Breitensuche, zu einer optimalen Laufzeit mit höchstens $|E|+|V|$ Schritten zu gelangen, muß ein "iterationsloser" Baumalgorithmus das minimale Ele-

ment der Kandidatenmenge in konstanter Zeit finden. Ein Suchalgorithmus für eine unsortierte Menge mit der Zeitkomplexität *O(1)* existiert nicht, da er immer sofort das richtige Element finden müßte. Eine notwendige Sortierung einer Menge für einen konstanten Zugriff ist nur in konstanter Zeit möglich, wenn die Menge schon sortiert ist.

Die Aufgabe der Effizienzsteigerung von Kürzeste−Wege−Algorithmen besteht somit in der Entwicklung einer geschickten Suche, Sortierung oder Suche mit Sortierung für die Kandidatenmenge der "iterationslosen" Verfahren nach dem minimalen Absatnd. Die Komplexität der Laufzeit dieses Suchens und/oder Sortierens muß möglichst nahe an *O(1)* liegen, damit die Gesamtlaufzeit fast linear wird.

Dijkstra hat es mit der Darstellung der Kandidatenmenge als einfache Prioritätswarteschlange in Form einer Liste geschafft, den Aufwand der Suche auf auf die lineare Ordnung O(|V|) zu beschränken und kommt so zu einem Gesamtaufwand von $|V|^2 + |E| = |V|^2$ Schritten zu gelangen. Das Einfügen eines Knoten in die Schlange ist dabei konstant, da eine beliebige Position für das Element ausgewählt wird. Das Herausnehmen ist identisch mit dem linearen Durchsuchen der Liste von einem Ende bis zum anderen. Für eine weitere Effizienzsteigerung der Baumalgorithmen bedeutet dies, das die Suche oder die Sortierung der Kandidaten schneller als linear sein muß.

1972 gelangte Spira eine erste entscheidende Verbesserung der Minimumsuche [Sp73] durch Darstellung der Prioritätsschlange als ein Heap. Ein Heap ist ist ein fast vollständig binärer Baum mit der Eigenschaft, daß die Werte der Knoten für eine Minimumsuche auf dem Weg von der Wurzel zu einem Blatt monoton steigend sind. Beim Einfügen wird ein neuer Knoten sofort an die richtige Stelle des Baums mit log|V| Schritten einsortiert und kann dann mit log|V| wieder herausgeholt werden, da der Baum nur noch von der Wurzel zu einem Blatt durchsucht werden muß. So ergibt sich ein Gesamtaufwand von $O(|V| \cdot \log^2 |V|)$.

Der Algorithmus von Dijkstra mit Heap

```
/*..INITIALISIERUNG.....................................................*/
  Prioritaetsschlange = {}; /* Prioritätsschlange mit Prioritätsfunktion

  Fuege den Ausgangsknoten v zur Prioritaetsschlange;
  Ordne dem Aufangsknoten v den Abstand d(v) = 0 zu;
  Ordne alle übrigen Knoten den Abstand d(v) = ∞ zu;

/*..SCHLEIFE, um den kuerzesten Weg um jeweils ein Knoten zu erweitern.*/
  Scheife, solange die Prioritaetssschlange nicht leer ist:
    v = minPrioritaetssschlange;
    DELETEMIN (Prioritaetssschlange);
    Schleife ueber alle Knoten w:
      Wenn d(w) = ∞ ist,
      dann ist d(w) = d(v) + l(v,w) und
           fuege W mit Prioritaet d(w) in die Prioritaetssschlange ein;
      Sonst: Wenn d(v) + l(v,w) < d(w),
             dann aendere die Priorität auf d(w) = d(v) + l(v,w)
    Schleifenende
  Schleifenende
```

Algorithmus 6: Kürzeste Wegebestimmung nach Dijkstra mit Heap

Johnson erreicht 1977 mit einer neuen, komplizierten Darstellung einer Prioritätsschlange eine Komplexität von $O(\min(|V|^{1+1/k} + |E|,\ (|V| + |E|) \cdot \log|V|))$ für einen Baumalgorithmus [Jo77].

Fredman zeigte 1976 erstmals die theoretische Möglichkeit einer Komplexität von $O(n^{5/2})$ für einen Algorithmus zur Bestimmung der kürzesten Wege zwischen je zwei Knoten [Fr76]. Vierzehn Jahre später findet er zusammen mit Dan E. Willard eine Möglichkeit der Zeitkomplexität $O(|E| + |V| \log|V| / \log\log|V|)$ für eine neue Version des Algorithmus von Dijkstra [Fr90]. Hierzu wird ebenfalls ein spezieller Heap verwendet. Der verwendete Fibonacci−Heap (oder AF−Heap) ist ein Wald heapgeordneter Bäume mit jeweils disjunkten Knotenmengen und verhält sich im Gegensatz zu anderen Heaps sehr viel dynamischer. Ohne hier auf die Einzelheiten einzugehen (siehe [Ot90], 435−444), läßt sich zeigen, daß die Zugriffszeit die Komplexität $O(\log|V|)$ hat, und durch eine amortisierte Zeit jede andere Operation auf den Fibonacci−Heap konstant ausgeführt werden kann.

Die letzten beiden Algorithmen, die hier erwähnt werden, arbeiten ebenfalls mit Fibonacci−Heaps und erreichen mit $O(|E| \cdot |V| + |V|^2 \log|V|)$ für alle kürzesten Wege in einem Graph das bisherige Optimum. Obwohl der eine Algorithmus von Young, Tarjan und Orlin am MIT [Yo91] unabhängig von anderen (Karger, Koller und Phillips an der Standford University [Ka91]) entwickelt wurde, kommen beide zur gleichen Laufzeit.

5.2. Verbesserung der Routensuche durch Parallelisierung

Da mit Hilfe der Fibonacci−Heaps eine logrithmische und somit fast konstante Zeit für das Finden des Kandidatenknoten minimalen Abstands erreicht werden konnte, ist der Einsatz von Parallelrechern für diese Suche kaum zu gerechtfertigt. Mit der Parallelisierung ist jedoch eine Reduzierung der, in sequenziell Baumalgorithmen nicht veränderbaren Zeitkomplexität von $O(|E| + |V|)$, auf einen logarithmischen Durchlauf möglich. So wird beispielsweise in [Kav94] eine Zeitkomplexität von $O(\log n \log\log n)$ angegeben.

5.3. Fazit bezüglich der Effizienzanforderung an die Routensuche

Mit der Entwicklung von fast linearen Baumalgorithmen auf Seriellrechnern und von fast konstanten Verfahren auf Parallelrechnern ist die Grenze der Effizenzsteigerung von Algorithmen zur Bestimmung kürzester Wege in einem gewichteten Graph erreicht.

Die Untersuchung weiterer Effizienzsteigerung dieser Kürzeste−Wege−Algorithmen ist daher im Verkehrswesen nicht weiter erforderlich.

Erforderlich für jeden Verkehrsplaner ist jedoch das Wissen vom prinzipiellen Ablauf der Routensuchverfahren, um die effizienten Kürzeste−Wege−Algorithmen bei guter Dokumentation der Ein− und Ausgabeparameter optimal einsetzen zu können. Er muß wissen, nach welchen Kriterien kürzeste, beste, beziehungsweise optimale Wege berechnet werden sollen, und er muß in der Lage sein, diese Kriterien (etwa im Umlegungsmodell) korrekt zu formulieren. Die Routensuchverfahren bestimmen nach diesen Kriterien lediglich optimale Wege − allerdings sehr schnell.

6. Zum Problem der Modellierung von Verkehrsnetzen

Die grundsätzliche Anforderung an die Verkehrsumlegungsmodelle, in die die Routensuchverfahren eingebettet sind, ist die zuverlässige und realistische Modellierung des Wegewahlverhaltens der Verkehrsteilnehmer. Hiermit ist auf der einen Seiten eine möglichst umfassende Modellierung des Verkehrsverhaltens notwendig, auf der anderen Seite können die Routensuchverfahren jedoch nur dann effektiv (wieder–) verwendet werden, wenn ihnen ein Graph sowie ein Kriterium in Form einer Kantengewichtung übergeben wird. Diesen vermeintlichen Widerspruch der Modellierung gilt es in Verkehrsumlegungsmodellen zu überwinden.

Betrachtet man den Verkehrsablauf in einem Straßennetz, der in einem Modell abgebildet werden soll, so erkennt man Knotenpunkte, Straßen und Verkehrsteilnehmer, die sich in dem Verkehrsnetz bewegen. Diese drei Komponenten müssen für ein realistisches Verhalten möglichst genau modelliert werden.

Die zur Zeit im Verkehrswesen übliche Anpassung der Verkehrsbeschreibung an ein spezielles Problem führt im Fall der Routensuchverfahren dazu, daß die Verkehrsteilnehmer in Form kontinuierlicher Verkehrsströme und die Knotenpunkte oder Straßen als Knoten im Graph beschrieben werden müssen. Bilden die Knotenpunkte die Knoten im Graph, so können die knotenpunktverbindenden Straßen nicht als eigenständige Elemente des Verkehrnetzes abgebildet werden. Umgekehrt sind bei der Abbildung der Straßen als Knoten die Eigenschaften der Knotenpunkte nicht darstellbar.

Sollen sowohl die Eigenschaften der Knotenpunkte, als auch der knotenverbindenden Straßen im Modell abgebildet werden, so kann dies durch einen bipartiten Graph erfolgen. Er beschreibt dann in der einen Menge die Knotenpunkten mit ihren speziellen Eigenschaften und in der anderen Menge die Straßen zwischen den Knotenpunkten mit Eigenschaften, wie Breite, Steigung oder Kurvigkeit. Für die Relation müßte die Bedingung gelten, daß im gerichteten Fall jeweils nur eine Kante von einem Element der ersten Menge zu einem Element der zweiten Menge existiert, oder im ungerichteten Fall nur jeweils zwei.

Für bipartite Graphen existieren jedoch keine Algorithmen zur Bestimmung kürzester Wege. Eine Möglichkeit dies zu beheben, wäre natürlich die Entwicklung einer neuen effizienten Routensuche für bipartite Graphen. Mit einem bipartiten Graph kann in vielen Fällen aber auch noch kein sehr realistisches Wegewahlverhalten der Verkehrsteilnehmer im Modell dargestellt werden. Weitere Modifikationen des Graphen würden dann zu wieder neuen Entwicklungen von Routensuchverfahren für die neue Darstellung des Verkehrs führen. Eine solche prozedurale Modellierung ist daher wegen ihrer mangelnden Flexibilität und kaum einer Wiederverwendungsmöglichkeit gerade für größere Verkehrsmodelle nicht sinnvoll.

Sinnvoll ist besonders für komplexere Probleme eine objektorientierte Modellierung. Die Elemente des betrachteten Verkehrsnetzes, wie die Knotenpunkte, die knoten-

punktverbindenden Straßen und die Verkehrsteilnehmer mit ihren Verkehrsmitteln werden als Objekte gewünschter Form modelliert (So ist beispielsweise auch ein kontinuierlicher Verkehrsstrom als Objekt beschreibbar). Jedem Objekt werden Eigenschaften und Methoden zugeordnet. Wenn diese Objekte hinsichtlich der verkehrsplanerischen Ziele möglichst korrekt beschrieben sind, werden die notwendigen Relationen zwischen den Objekten modelliert. Hierfür können unter anderem auch Graphen verwendet werden. Objekte und Relationen bilden die Struktur des Modells. Bei richtiger Modellierung ist der Verkehr auf diese Weise im Modell hinreichend abgebildet.

Die objektorientierte Modellierung bietet die Möglichkeit, auf einfache Weise neue völlig unterschiedliche Verkehrsmittel in Verkehrsmodelle einzuführen, zu ändern oder wieder herauszunehmen. Jedes Verkehrsmittel hat (auch wenn es beispielsweise in der Form eines Verkehrsstroms auftritt) seine speziellen Eigenschaften und Methoden. Durch die speziellen Relationen jedes Verkehrsmittels zum Straßennetz, zu den anderen Verkehrsteilnehmern und Verkehrsmittel sowie zu den Anlagen der Steuerungs– und Leitsysteme kann es mit seinem individuellen Verhalten im Zusammenhang mit seiner Umgebung objektorientiert dargestellt werden.

Für Aufgaben, wie der Suche kürzester Wege in einem Graph, sollten Standardalgorithmen in Funktionsbibliotheken zur Verfügung stehen, die Probleme in abstrakter Form möglichst effektiv lösen. Durch die einmalige Festlegung einer minimalen Schnittstelle der Algorithmen sind die Ein– und Ausgabeparameter eindeutig beschrieben. Ein Standardalgorithmus zur Bestimmung kürzester Wege erfordert als Eingabe einen Graph $G=(V; E)$, eine Gewichtungsfunktion und eventuell einen Anfangs– und einen Endknoten.

Um in dem oben erläuterten objektorientierten Verkehrsmodell einen kürzesten Weg zu suchen, ist eine Transformation der vorhandenen Darstellung des Verkehrsnetzes auf die Darstellung eines Graphen G durchzuführen und aus den Eigenschaften der Objekte und Relationen eine Gewichtungsfunktion ”zusammenzustellen”, die das gewünschte Kriterium, nach dem optimiert werden soll, beschreibt.

Die Transformation von einer Darstellung des Verkehrsnetzes zu einer anderen kann objektorientiert geschehen. Daß heißt, während der Transformation werden bestimmte Objektmethoden aufgerufen. Da jedes Objekt seine eigenen Methoden und Eigenschaften ”kennt”, kann jedes Objekt individuell bei der Transformation verwendet oder geändert werden.

In der Routensuche spielt die Modellierung der Verkehrsteilnehmer und der verkehrsmittel direkt keine Rolle, da die Eigenschaften von Objekten in diesem Verfahren überhaupt nicht betrachtet werden. Es wird nur die Gewichtung der Kanten betrachtet. Komplexere Zusammenhänge im Verkehrsmodell, wie das gegenseitig bedingte Verhalten der Verkehrsteilnehmer können durch eine komplexe Bestimmung des Kriteriums mit Berücksichtigung vieler Einflüsse der Verkehrsteilnehmer, Knotenpunkte und Straßen für die Routensuche dargestellt werden. Günstiger erscheint jedoch ein Konzept bei

dem mehrere unterschiedliche Kriterien entwickelt werden, deren Auswirkungen nach einer Bestimmung der kürzesten Wege für jeweils ein Kriterium, innerhalb des Verkehrsmodells (Umlegungsmodells) in Beziehung zueinander gesetzt werden kann. So sind nicht nur die Zusammenhänge zwischen unterschiedlichen Verkehrsmitteln darstellbar, sondern beispielsweise auch realistische Alternativroutenverfahren, wo bestimmte Personengruppen nach völlig unterschiedlichen Kriterien eine Route suchen. Auch für dynamische Modelle, wie DRUM, muß das Standardverfahren zur Suche kürzester Wege mehrmals, jeweils nach den Auswirkungen der Routensuche im Rechenschritt zuvor, bestimmt werden.

Löst man sich also von der Vorstellung, daß innerhalb einer Verkehrsumlegung nur eine Routensuche nach nur einem Kriterium möglich ist, so wird für Umlegungsprobleme, in denen beispielsweise unterschiedliche Verkehrsarten oder Alternativrouten betrachtet werden, die Lösung mit einem einzigen Standardalgorithmus zur effizienten Bestimmung kürzester Wege in einem Graph möglich. Es muß lediglich eine korrekte Transformation auf einen Graph durchgeführt und die Kriterien der Optimierung aus den Objekten und den Relationen des Verkehrsmodells zusammengestellt werden.

Die Analyse und Bestimmung der korrekten und sinnvollen Kriterien, nach denen die Routensuche das Optimum finden soll, kann nicht Aufgabe der Informatik sein, sondern sollte den Verkehrsplanern überlassen werden, da nur sie sich in ihrer speziellen Tätigkeit mit den Zusammenhängen im Verkehr beschäftigen und auskennen. Die Verkehrsbauingenieure müssen jedoch die Prinzipien der Routensuche kennen, um sie optimal in die notwendigen Verkehrsmodelle integrieren zu können.

7. Literaturverzeichnis

7.1. Routensuchverfahren im Verkehrswesen

[Br80] Braun, J.:
Adaptive Ermittlung kürzester Routen in Verkehrsnetzen.
Bock und Herchen, Bad Honnef, 1980

[Pl92] Ploss, G.:
Ein Dynamisches Verfahren zur Schätzung von Verkehrsbeziehungen aus Querschnittszählungen.
Fachgebiet Verkehrsplanung und Verkehrswesen,
Technische Universität München, 1992

[Ri72] Ribbeck, K.F.:
Routensuchen in Straßennetzen.
Forschung Straßenbau und Straßenverkehrstechnik, Heft 134, Bonn, 1972

[Se94] Serwill, D.:
DRUM: Modellkonzept zur dynamischen Routensuche und Umlegung.
Berichte 43, Institut für Stadtbauwesen, RWTH Aachen, 1994

[St66] Stairs, S.W.:
Bibliography of the Shortest Route Problem.
LSE−TNT−l London, 1966

7.2. Graphentheorie

[Ah74] Aho, A.V.; Hopcroft, J.E.; Ullman, J.D.:
The Design and Analysis of Computer Algorithms.
Addison−Wesley, 1974

[Ber89] Berge, C.:
Graphs.
North Holland, second revised edition, 1989

[Br94] Brandstädt, A.:
Graphen und Algorithmen.
B.G. Teubner, Stuttgart, 1994

[Cl94] Clark, J.:
Graphentheorie: Grundlagen und Anwendungen.
Spektrum, Akademischer Verlag, Heidelberg, Berlin, Oxford, 1994

[Eu36] Euler, L.:
The Solution of a problem to the Geometry of Position.
Commentarii Academiae Scientiarum Imperialis, 128−140, 1736

[Ju94] Jungnickel, D.:
Graphen, Netzwerke und Algorithmen.
3. Auflage, BI−Wissenschaftsverlag, Mannheim, Leipzig, 1994

7.3. Allgemeine Probleme mit Routen

[FoFu56] Ford, L.R., jr; Fulkerson D.R.:
Maximal Flow Through a Network.
Paper P−605, RAND Corporation, Santa Monica, (1954),
Canadian Journal of Mathematics 8, 399−404, 1956

[FoFu62] Ford, L.R., jr; Fulkerson D.R.:
Flows in Networks.
Princeton University Press, Princeton, New Jewrsey, 1962

[Kr56] Kruskal, J.B., jr.:
On the Shortest Spanning Subtree of a Graph and the Travelling Salesman Problem.
Proccedings of the American Mathematics Society 7, 48−50, 1956

[Kw62] Kwan, M.−K.:
Graphic Programming using Odd and Even Points.
Chines. Math. 1, 273−277, 1962

[La85] Lawler, E.L.; Lenstra, J.K.; Rinnooy Kan, A.H.G.; Shmoys, D.B.:
Traveling Salesman Problem: A Guided Tour of Combinatorical Optimization.
Wiley & Sons, New York, 1990

7.4. Kürzeste Wege von einem Knoten aus

[Da59] Dantzig, G.B.:
On the Shortest Route Through a Network.
Paper P−1345, RAND Corporation, Santa Monica, (1958),
Management Science 6, 187−190, 1959

[Di59] Dijkstra, E.W.:
A note on two problems in connexion with graphs.
Numerische Mathematik 1, Springer, Berlin, 269−271, 1959

[Dia69] Dial, R.B.:
Algorithm 360: Shortest Path Forest with Topological Ordering.
Communications of the Association of Computing Mach. 12, 632−633, 1969

[Fo56] Ford, L.R., jr:
Network Flow Theory.
Paper P−923, RAND Corporation, Santa Monica, 1956

[Mo59] Moore, E.F.:
The Shortest Path Through a Maze.
Proceedings of the International Symposium on the Theory of Switching,
The Annals of the Computing Laboratory of Harvard University 30, Part II,
Harvard University Press, 1959

[Po60] Pollak, M.; Wiebenson, W.:
Solutions of the Shortest−Path−Problem − a Review.
Operations Research 8, 224−230, 1960

[Wh60] Whiting, P.D.; Hillier, I.A.:
A Method for Finding a Shortest Route Through a Road network.
Operations Research, Quarterly, 37−40, 1960

7.5. Kürzeste Wege zwischen je zwei Knoten

[Bel58] Bellman, R.:
On a Routing Problem.
Quarterly of Applied Mathematics 16, 87−90, 1958

[Ca71] Carré, P. A.:
An Algebra for Network Routing Problems.
J. Inst. math. Appl., 7, 273−294, 1971

[Da51] Dantzig, G.B.:
Application of the Simplex Method to a Transportartion Problem.
Activity Ananlysis of Production and Allocation.
New York Cowles Commission Monograph 13, 359−373, 1951

[Fa67] Farbey, B.A.; Land, A.H.; Murchland, J.:
The Cascade Algorithm for Finding all Shortest Distances in a Direced Graph.
Management Science 14, 19−28, 1967

[Fl62] Floyd, R.W.:
Algorithm 97 (SHORTEST PATH).
Communications of the Association of Computing Machinery 5, 345, 1962

[Sh55] Shimbel, A.:
Structure in Communication Nets.
Proceeding: Symposium of Information Networks, (1954),
Polytechnik Institut of Brooklyn, Ed. Jerome Fox, 199−203, 1955

[Jo62] Warshall, S.:
A Theorem on Boolean matrics.
Journal of the Association for Computer Mach., 9, 11−12, 1962

7.6. Effizienzsteigerung der klassischen Routensuchverfahren

[Fr76] Fredman, M.L.:
New Bounds on the Complexity of the Shortest Path Problem.
SIAM Journal of Computing 5, 83−89, 1976

[Fr90] Fredman, M.L.; Dan E. Willard:
Trans−dichotomous Algorithms for Minimum Spanning Trees and Shortest Paths
Proceedings of the 31th Anunual IEEE Conference on Foundations of Computer Science (FOCS), 719−725, 1990

[Jo77] Johnson, D.B.:
Efficent Algorithms for Shortest Paths in Sparse Networks.
Journal of the Association for Computer Mach., 24, 1−13, 1977

[Ka91] Karger, D.R.; Koller, D.; Phillips, St.J.:
Finding the Hidden Path: Time Bounds for All Shortest Paths.
Proceedings of the 32th Anunual IEEE Conference on Foundations of Computer Science (FOCS), 560−568, 1991

[Kav94] Kavvadias, D.; Pantziou, G.; Spirakis, P.; Zaroliagis, C.:
Hammock−on−Ears Decomposition: A Technique for the Efficient Parallel Solution of Shortest Paths and other Problems
Paper P−923, RAND Corporation, Santa Monica, 1956

[Ot90] Ottmann, T.; Widmayer, P.:
Algorithmen und Datenstrukturen.
BI−Wissenschaftsverlag, Mannheim, Leipzig, 1990

[Sp73] Spira, P.M..:
A new Algorithm for Finding all Shortest Paths in a Graph of Positive Arcs in Average Time $O(n^2 \log^2 n)$.
SIAM Journal of Computing 2, 28−32, 1973

[Yo91] Young, N.E.; Tarjan, R.E.; Orlin, J.B.:
Faster Parametric Shortest Path and Minimum−Balance Algorithms
Networks, 21, 205−221, 1991