Kombinatorische Optimierung

M. Grötschel

Manuskript einer im WS 1997/98

an der TU Berlin im Fachbereich Mathematik

gehaltenen Vorlesung

Teil I

Kapitel 1 bis 8

Dezember 1997

Vorwort

Bei dem vorliegenden Skript handelt es sich um die Ausarbeitung der vierstündigen Vorlesung "Kombinatorische Optimierung", die ich im WS 1997/98 am Fachbereich Mathematik der TU Berlin gehalten habe.

Ziel der Vorlesung ist eine Einführung in die kombinatorische Optimierung. Dieses Gebiet baut in seinen Hauptsträngen im wesentlichen auf drei Pfeilern auf: der Graphentheorie, der Informatik und der linearen Optimierung.

Die Graphentheorie stellt in vielen Bereichen die Begriffe zur Beschreibung kombinatorischer Optimierungsprobleme zur Verfügung. Viele ihrer Sätze und Beweise sind algorithmischer Natur und ermöglichen, schnelle Verfahren zur Lösung kombinatorischer Optimierungsprobleme anzugeben. Kenntnisse der Graphentheorie sind nützlich, zum Verständnis der Vorlesung aber nicht erforderlich. Alle benötigten Begriffe und Konzepte werden in der Vorlesung eingeführt.

Aus der Informatik benötigen wir Grundkenntnisse der Komplexitätstheorie und Kenntnisse über Datenstrukturen zum Entwurf effizienter Algorithmen. Die für uns wichtigen Konzepte der Komplexitätstheorie werden in der Vorlesung informell erläutert. Datenstrukturen werden kurz gestreift. Sie werden hauptsächlich in den Übungen diskutiert.

Viele kombinatorische Optimierungsprobleme haben eine Interpretation als lineare oder lineare (gemischt-) ganzzahlige Programme. Auf Dualitätstheorie und Verfahren zur Lösung linearer Programme wird häufig Bezug genommem. Grundkenntnisse dieses Gebietes werden vorausgesetzt.

Die Vorlesung umfaßt sowohl die Behandlung polynomial lösbarer kombinatorischer Probleme (Auffindung minimaler aufspannender Bäume, kürzester Wege, maximaler Flüsse, kostenminimaler Flüsse ...) als auch die Untersuchung $\mathcal{NP}$-vollständige Probleme (Knapsack-Problem, Travelling-Salesman-Problem, Max-Cut-Problem, Reihenfolgeprobleme, ...). Für letzteren Problemkreis werden sowohl Techniken zum Entwurf von primalen und dualen Heuristiken (mit Gütegarantie) als auch exakte Verfahren angegeben. Viele Beispiele realer Anwendungen sollen die besondere Praxisrelevanz dieses Gebietes unterstreichen.

Es gibt kein einzelnes Buch, das den gesamten, in dieser Vorlesung abgehandelten Themenkreis abdeckt. Daher sind in die einzelnen Kapitel Literaturhinweise eingearbeitet worden. Lesenswerte Einführungen in die kombinatorische Optimierung (bzw. gewisse Teilbereiche dieses Gebietes) sind die im Literaturverzeichnis von Kapitel 1 aufgeführten Bücher D. Jungnickel (1994), G. L. Nemhauser und L. A. Wolsey (1988), A. Schrijver (1986) und der Übersichtsartikel M. Grötschel und L. Lovász (1995).

Dies ist ein Vorlesungsskript und kein Buch. Obwohl ich mit der gebotenen Sorgfalt geschrieben habe, war nicht genügend Zeit für intensives Korrekturlesen und das Einarbeiten umfassender Literaturhinweise. Die daher vermutlich vorhandenen Fehler bitte ich zu entschuldigen (und mir wenn möglich mitzuteilen). Das Thema wird nicht erschöpfend behandelt. Das Manuskript enthält nur die wesentlichen Teile der Vorlesung.

Dezember 1997 — M. Grötschel

Inhaltsverzeichnis

Kapitel 1

Ganzzahlige und gemischt-ganzzahlige Optimierung

In diesem einführenden Kapitel wollen wir zunächst an die verschiedenen linearen Programme, die wir in der Vorlesung "Lineare Optimierung" kennengelernt haben, erinnern und dann daraus Formulierungen von ganzzahligen und gemischt-ganzzahligen Optimierungsproblemen ableiten. In Abschnitt 1.2 wird eine Reihe von Beispielen ganzzahliger und gemischt-ganzzahliger Programme diskutiert.

1.1 Problemformulierungen

Mit $\mathbb{R}$($\mathbb{Q}$) bezeichnen wir den Körper der reellen (rationalen) Zahlen, das Symbol $\mathbb{K}$ bezeichnet einen beliebigen dieser beiden angeordneten Körper. Wenn nichts anderes gesagt wird, gehen wir immer davon aus, daß gegebene Daten aus $\mathbb{K}$ sind. Sind Matrizen $A \in \mathbb{K}^{(p,r)}$, $B \in \mathbb{K}^{(p,s)}$, $C \in \mathbb{K}^{(q,r)}$, $D \in \mathbb{K}^{(q,s)}$ und Vektoren $a \in \mathbb{K}^r$, $b \in \mathbb{K}^s$, $c \in \mathbb{K}^p$, $d \in \mathbb{K}^q$ gegeben, dann bezeichnen wir bekanntlich die Aufgabe, einen Vektor $\binom{x}{y} \in \mathbb{K}^{r+s}$ zu finden, der $Ax+By=c$, $Cx+Dy \leq d$, $x \geq 0$ erfüllt und dessen Wert a^Tx+b^Ty so groß (oder klein) wie möglich ist, als **lineares Optimierungsproblem** oder kurz **lineares Programm (LP)**. Wir schreiben dafür abkürzend

$$
\begin{aligned}
&\max a^Tx+b^Ty \\
&\quad Ax+By=c \\
&\quad Cx+Dy \leq d \\
&\quad x \geq 0.
\end{aligned} \tag{1.1}
$$

Aufgabe (1.1) ist die Formulierung eines LP in seiner allgemeinsten Form. Bei den meisten Anwendungsfällen treten Spezialfälle von (1.1) auf. Desgleichen sind alle Algorithmen zur Lösung linearer Programme auf bestimmte Typen linearer Programme zugeschnitten, von denen wir allerdings jeweils gezeigt haben, daß sich (1.1) in diese Spezialtypen auf einfache Weise transformieren läßt. Wir erinnern daran, daß der Simplexalgorithmus lineare Programme der Form

$$
\begin{aligned}
&\max c^Tx \\
&\quad Ax=b \\
&\quad x \geq 0
\end{aligned} \tag{1.2}
$$

löst (mit $A \in \mathbb{K}^{(m,n)}$, $b \in \mathbb{K}^m$, $c \in \mathbb{K}^n$). LP's des Typs (1.2) heißen lineare Programme in **Standardform**. Die Ellipsoidmethode wird angewendet auf LP's der Form

$$\begin{aligned} \max\ & c^T x \\ & Ax \leq b. \end{aligned} \tag{1.3}$$

Der Karmarkar-Algorithmus ist für noch speziellere lineare Programme konzipiert.

Treten zusätzlich zu den Nebenbedingungen der linearen Programme (1.1), (1.2), (1.3) Forderungen nach Ganzzahligkeit aller oder eines Teils der Variablen auf, so nennt man die LP's ganzzahlig oder gemischt-ganzzahlig. Genauer, eine Aufgabe der Form (vergleiche (1.1))

$$\begin{aligned} \max\ & a^T x + b^T y \\ & Ax + By = c \\ & Cx + Dy \leq d \\ & x \geq 0 \\ & x \in \mathbf{Z}^r, y \in \mathbf{Z}^s \end{aligned} \tag{1.4}$$

heißt **ganzzahliges lineares Optimierungsproblem** oder kurz **ganzzahliges Programm**. Zur Abkürzung schreiben wir manchmal GLP oder auch IP (Kurznotation für: Integer Program). Teilen wir alle in (1.1) auftretenden Matrizen und Vektoren dimensionsverträglich auf, so ist

$$\begin{aligned} \max\ & a_1^T x_1 + a_2^T x_2 + b_1^T y_1 + b_2^T y_2 \\ & A_1 x_1 + A_2 x_2 + B_1 y_1 + B_2 y_2 = c \\ & C_1 x_1 + C_2 x_2 + D_1 y_1 + D_2 y_2 \leq d \\ & x_1, x_2 \geq 0 \\ & x_1 \in \mathbf{Z}^{r_1}, y_1 \in \mathbf{Z}^{s_1} \end{aligned} \tag{1.5}$$

ein (allgemeines) **gemischt-ganzzahliges lineares Optimierungsproblem** oder kurz **gemischt-ganzzahliges Programm** (GGLP oder MIP = Mixed Integer Program). Natürlich treten die Programme (1.4) und (1.5) selten in dieser allgemeinsten Form auf. Z. B. werden wir bei den ganzzahligen Programmen meistens auf Programme der Form

$$\begin{aligned} \max\ & c^T x \\ & Ax = b \\ & x \geq 0 \\ & x \text{ ganzzahlig} \end{aligned} \tag{1.6}$$

$$\begin{aligned} \max\ & c^T x \\ & Ax \leq b \\ & x \text{ ganzzahlig} \end{aligned} \tag{1.7}$$

$$
(1.8) \qquad \begin{aligned} &\max c^T x \\ &Ax \le b \\ &x \ge 0 \\ &x \text{ ganzzahlig} \end{aligned}
$$

stoßen. Dürfen die Variablen in einem linearen Programm nur die Werte 0 oder 1 annehmen, so spricht man von einem **binären linearen Programm** oder **0/1-Programm** (BLP). 0/1-Programme kommen meistens in folgender Form vor

$$
(1.9) \qquad \begin{aligned} &\max c^T x \\ &Ax \le b \\ &x \in \{0,1\}^n \,. \end{aligned}
$$

Man beachte, daß zu (1.9) die Nichtnegativitätsbedingung $x \ge 0$ hinzugefügt werden kann, ohne daß an der Menge der zulässigen Lösungen von (1.9) etwas geändert wird.

1.2 Beispiele und Anwendungen

Ganzzahlige bzw. gemischt-ganzzahlige Optimierungsprobleme treten immer dann auf, wenn gewisse Produkte oder Resourcen nicht beliebig geteilt werden können. Eine Fluggesellschaft kann nicht $\frac{7}{2}$ Flugzeuge kaufen sondern nur 3 oder 4, und ebenso macht es keinen Sinn, $\frac{29}{3}$ Lokomotiven zu produzieren. Die zusätzlich auftretende Forderung nach Ganzzahligkeit von Lösungvektoren führt zu erheblichen theoretischen und rechnerischen Problemen. Heutzutage kann man lineare Programme fast beliebiger Größenordnung lösen, dagegen können nicht selten ganzzahlige Programme mit wenigen hundert Variablen und Nebenbedingungen selbst in Hunderten von Stunden CPU-Zeit nicht gelöst werden. Auf eine theoretische Erklärung dieses Phänomens werden wir später noch eingehen. Zunächst wollen wir mehrere anwendungsnahe Beispiele diskutieren. Einige Beispiele sind dem Buch Brucker (1975) entnommen, in dem der Leser weitere Anwendungen und Literaturhinweise finden kann wie auch in den Büchern Ahuja, Magnanti und Orlin (1993) und Nemhauser und Wolsey (1988). Einige konkrete Anwendungen sind auch in Bachem, Jünger und Schrader (1995) dargestellt.

Diese Beispiele sind für den akademischen Gebrauch aufbereitet, um sie in wenigen Zeilen darstellen zu können. Die realen Anwendungsprobleme, die hinter diesen Skizzen zu erkennen sind, sind wesentlich komplizierter. Nähere Erläuterungen hierzu erfolgen in der Vorlesung.

(1.10) Das Rucksackproblem (englisch: **Knapsack Problem).** Für den Stückguttransport von einem Ort A zu einem Ort B stehen n verschiedene Güterarten in unbegrenzter Menge zur Verfügung. Für den Transport eines Stücks der Güterart i, das ein Gewicht von $a_i > 0$ besitzt, werden c_i Geldeinheiten gezahlt. Die Kapazität eines Lastwagens, der zum Transport der Güter eingesetzt werden soll, beträgt b Gewichtseinheiten.

Wie hat man den Lastwagen zu beladen, damit der Umsatz maximal wird? Dieses Problem führt zum folgenden ganzzahligen linearen Optimierungsproblem:

$$\begin{aligned} &\max c_1x_1 + c_2x_2 + \ldots + c_nx_n \\ &\quad a_1x_1 + a_2x_2 + \ldots + a_nx_n \le b \\ &\qquad\qquad x_i \ge 0 \text{ und ganzzahlig für } \; i = 1, 2, \ldots, n. \end{aligned}$$

Ein GLP mit dieser Struktur (Vorzeichenbeschränkungen, Ganzzahligkeitsbedingungen und nur eine Ungleichung, üblicherweise mit nicht-negativen Koeffizienten) heißt **Rucksackproblem**.

□

(1.11) Bestimmung einer optimalen Auswahl von Werbekampagnen. Unter einer Reihe von Werbekampagnen soll eine Auswahl derart getroffen werden, daß

- ein vorgegebenes Werbebudget von 700 000 DM nicht überschritten wird,
- möglichst viele potentielle Käufer durch die Werbung angesprochen werden.

Die möglichen Werbekampagnen, ihre Kosten und die Anzahl potentieller Käufer, die angesprochen werden, sind in der folgenden Tabelle aufgeführt.

Nummer der Kampagne	1	2	3	4	5
Art der Kampagne	Postwurfsendungen	Sonderangebote	Fernsehwerbung	Magazinwerbung	Verkäufertraining
Kosten in DM	100 000	40 000	300 000	250 000	100 000
Anzahl potentieller Käufer, die angesprochen werden	200 000	50 000	400 000	300 000	75 000

Dann läßt sich das gestellte Problem wie folgt als binäres lineares Optimierungsproblem formulieren:

$$\begin{aligned} &\max 200000x_1 + 50000x_2 + 400000x_3 + 300000x_4 + 75000x_5 \\ &\quad 100000x_1 + 40000x_2 + 300000x_3 + 250000x_4 + 100000x_5 \le 700000 \\ &\quad x_1, x_2, x_3, x_4, x_5 \in \{0, 1\}. \end{aligned}$$

□

Schaut man sich die mathematische Modellierung von Problem (1.11) an, so wird man feststellen, daß (1.11) ein Spezialfall von (1.10) ist. (1.11) ist ein sogenanntes **binäres** (oder 0/1-) **Rucksackproblem**. Wir werden dieses und damit verwandte Probleme ausführlich in einem späteren Kapitel behandeln.

(1.12) Ein Problem der Investitionsplanung. Ein Unternehmen plant, in zwei wichtigen Zweigbetrieben Investitionsprojekte durchzuführen. In jedem dieser Betriebe hat man drei verschiedene Investitionsmöglichkeiten. Diese Möglichkeiten sind jedoch dadurch eingeschränkt, daß

1. maximal 85 Geldeinheiten für alle Investitionsprojekte zur Vefügung stehen,
2. in den Zweigbetrieben nicht mehr als 8 bzw. 10 Arbeitskräfte zur Betreuung der Projekte zur Verfügung stehen.

Für jedes Projekt sind bekannt

- seine Kosten,
- die Anzahl der zur Betreuung notwendigen Arbeitskräfte,
- der Gegenwartswert aller durch die Realisation des Projektes bedingten zusätzlichen Erträge.

Diese Daten sind in der folgenden Tabelle enthalten.

	Betrieb A			Betrieb B		
Projekt Nr.	1	2	3	4	5	6
Bedarf an Arbeitskräften	3	4	6	3	4	6
Kapital-bedarf in GE	25	30	40	15	20	25
Erträge in GE	6	8	7	4	5	5

In welche Projekte hat man zu investieren, um den Gegenwartswert aller Erträge zu maximieren, wenn zugleich das nicht investierte Kapital p% an Zinsen im Planungszeitraum bringt? Diese Fragestellung führt zum folgenden binären gemischt-ganzzahligen Programm:

$$
\begin{aligned}
\max\ & 6x_1 + 8x_2 + 7x_3 + 4x_4 + 5x_5 + 5x_6 + \tfrac{p}{100}x_7 \\
& 25x_1 + 30x_2 + 40x_3 + 15x_4 + 20x_5 + 25x_6 + x_7 = 85 \\
& 3x_1 + 4x_2 + 6x_3 \le 8 \\
& 3x_4 + 4x_5 + 6x_6 \le 10 \\
& x_i \in \{0,1\}\ i = 1, 2, \dots, 6 \\
& x_7 \ge 0
\end{aligned}
$$

x_7 ist der Kapitalbetrag, der nicht investiert wird, und dann einen Ertrag von $\frac{p}{100}x_7$ erbringt.

Man kann nun das beschriebene Modell auf verschiedene Weise modifizieren und dadurch flexibler gestalten. Will man z. B. erreichen, daß nicht in beide Projekte 1 und 2 investiert wird, so hat man die zusätzliche Nebenbedingung $x_1 + x_2 \le 1$ hinzuzufügen. Eine Bedingung der Form $x_4 - x_5 \le 0$ bewirkt, daß mit dem Projekt 4 auch das Projekt 5 durchgeführt werden muß.

Das Modell läßt sich außerdem dadurch erweitern, daß man z. B. einen Transfer von maximal zwei Arbeitskräften von A nach B und umgekehrt zuläßt. Bezeichnet man mit b_{AB}

bzw. b_{BA} die Anzahl der von A nach B bzw. von B nach A transferierten Arbeitskräfte, so unterscheidet sich das neue Optimierungsmodell vom alten dadurch, daß man

1. die Zielfunktion durch
$$6x_1 + 8x_2 + 7x_3 + 4x_4 + 5x_5 + 5x_6 + \frac{p}{100} x_7 - 0.3 b_{AB} - 0.2 b_{BA}$$
(0.3 bzw. 0.2 sind die Transferkosten je Arbeitskraft) zu ersetzen hat,
2. anstelle der 2. und 3. Nebenbedingung
$$\begin{array}{rcl} 3x_1 + 4x_2 + 6x_3 \qquad\qquad\qquad + b_{BA} - b_{AB} & \leq & 8 \\ 3x_4 + 4x_5 + 6x_6 + b_{AB} - b_{BA} & \leq & 10 \end{array}$$
zu betrachten hat,
3. die zusätzlichen Restriktionen
$$0 \leq b_{AB} \leq 2\,, \quad 0 \leq b_{BA} \leq 2 \;\text{ und } b_{AB}, b_{BA} \text{ ganzzahlig}$$
hinzufügen muß.

□

(1.13) Ein Standortproblem. Ein Versandhaus kann an zwei verschiedenen Orten die Auslieferungslager L_1, L_2 errichten, von denen aus 3 verschiedene Kundengruppen D_1, D_2, D_3 beliefert werden sollen. Die nachfolgende Tabelle enthält

- Bau- und Erhaltungskosten der Auslieferungslager,
- für jedes Lager L_i und jede Kundengruppe D_j die Kosten, die entstehen, falls 1% der Güter für die Kundengruppe D_j durch das Lager L_i geliefert werden.

		Lieferkosten		
	Bau- und Erhaltungskosten	D_1	D_2	D_3
L_1	35	30	40	70
L_2	35	80	60	40

Wir nehmen an, daß die Nachfrage bei den Kunden zeitlich konstant ist und durch das Versandhaus aufgrund vertraglicher Verpflichtungen erfüllt werden muß. Die Summe aus Bau-, Liefer- und Unterhaltungskosten soll minimal werden. Welche Lager soll man unter diesen Voraussetzungen errichten? In welchem Umfang sollten die Kundengruppen aus den errichteten Lagern beliefert werden?

Obiges Problem führt zu einem gemischt-ganzzahligen linearen Optimierungsproblem der Form
$$\min\ 30x_{11} + 40x_{12} + 70x_{13} + 80x_{21} + 60x_{22} + 40x_{23} + 35y_1 + 35y_2$$

$$
\begin{array}{llllllll}
x_{11} & & & +x_{21} & & & & =100 \\
& x_{12} & & & +x_{22} & & & =100 \\
& & x_{13} & & & +x_{23} & & =100 \\
x_{11}+x_{12}+x_{13} & & & & & -My_1 & & \leq \quad 0 \\
& & & x_{21}+x_{22}+x_{23} & & & -My_2 & \leq \quad 0 \\
\end{array}
$$

$$
x_{ij} \geq 0 \;\; \text{für } i=1,2 \quad j=1,2,3
$$

$$
y_1, y_2 \in \{0,1\}
$$

Die Variablen x_{1j} bzw. x_{2j} geben an, wieviel Prozent der nach D_j gelieferten Mengen aus L_1 bzw. L_2 stammen.

$y_i = 1$ oder 0 bringt zum Ausdruck, ob das Lager L_i errichtet wird ($y_i = 1$) oder nicht ($y_i = 0$). M ist eine sehr große ganze Zahl. Ist $x_{1j} > 0$ für mindestens ein j, so folgt aus der 4. Nebenbedingung, daß $y_1 = 1$ sein muß. Ebenso folgt aus der 5. Nebenbedingung, daß Lager L_2 gebaut werden muß, falls ein Kunde durch L_2 beliefert werden soll. Ist dagegen $\sum_{j=1}^{3} x_{ij} = 0$, so sorgt die Zielfunktion dafür, daß in einer optimalen Lösung $y_i = 0$ sein muß. Die ersten drei Nebenbedingungen sorgen dafür, daß alle Kunden voll beliefert werden.

□

Das hier dargestellte Modell ist recht einfach, eine Lösung des zugehörigen GGLP läßt sich leicht finden, indem man für die drei Möglichkeiten $y_1 = 0$ und $y_2 = 1$, $y_1 = 1$ und $y_2 = 0$ sowie $y_1 = 1$ und $y_2 = 1$ das entsprechende lineare Programm löst und die Ergebnisse vergleicht. Bei realistischen Problemen dieser Art (die häufig und in vielfältiger Form auftreten) kann man jedoch von einer Vielzahl von Auslieferungslagern und Kundengruppen ausgehen. In solchen Fällen ist der soeben beschriebene Lösungsweg nicht mehr realisierbar.

(1.14) Ein Verschnittproblem. Aus Eisenstäben der Länge b sollen je k_i-mal die Längen $a_i \leq b$, $i = 1, \ldots, n$, so zugeschnitten werden, daß die Anzahl der angeschnittenen Stäbe möglichst klein wird. Die ganze Zahl m sei eine obere Schranke für die Anzahl der zu zerschneidenden Stäbe (z. B. $m = \sum_{i=1}^{n} k_i$). Setzen wir

$$
x_{ij} = d, \text{ falls Länge } a_i \text{ vom } j\text{-ten Stab } d\text{-mal abgeschnitten wird,}
$$

wobei $i = 1, \ldots, n$, $j = 1, \ldots, m$, so läßt sich das Problem durch das binäre lineare Optimierungsproblem

$$
\begin{array}{ll}
\min \sum_{j=1}^{m} y_j & \\
\sum_{j=1}^{m} x_{ij} = k_i & i = 1, \dots, n \\
\sum_{i=1}^{n} a_i x_{ij} \le b & j = 1, \dots, m \\
\sum_{i=1}^{n} x_{ij} - M y_j \le 0 & j = 1, \dots, m \\
x_{ij} \ge 0 \text{ und ganzzahlig} & i = 1, \dots, n;\ j = 1, \dots, m\,, \\
y_j \in \{0, 1\} & j = 1, \dots, m,
\end{array}
$$

in dem M eine (relativ zu den übrigen Daten) sehr große positive Zahl darstellt, ausdrücken. Die ersten beiden Systeme von Nebenbedingungen sorgen dafür, daß jede Länge k_i-mal zugeschnitten wird und nicht mehr Längen aus einem Stab zugeschnitten werden, als seine Länge zuläßt. Das letzte System bewirkt, daß jeder angeschnittene Stab gezählt wird. Ist nämlich mindestens ein $x_{ij} = 1$, so folgt $y_j = 1$.

□

Von viel größerer praktischer Bedeutung als das oben eingeführte eindimensionale Verschnittproblem sind zweidimensionale Verschnittprobleme. Sie treten z. B. in der Kleidungsproduktion und im Karosseriebau auf. Bei der Herstellung von Hemden, Anzügen, Babywäsche etc. werden von einem Modell sehr viele Stückzahlen aus u. U. unterschiedlichen Stoffen produziert. Die (nicht einfache) Aufgabe besteht darin, die benötigten Schnittmuster auf die verschiedenen Stoffbahnen so zu verteilen, daß der Verschnitt minimal ist. Analog sind die Grundformen der Karosserieteile auf die vorhandenen Bleche so zu plazieren, daß möglichst wenig Reste übrigbleiben. Wie würden Sie dieses Problem als ganz- bzw. gemischt-ganzzahliges Optimierungsproblem formulieren?

Die Aufgabe, einen Transportcontainer mit Behältern so zu beladen, daß möglichst wenig Leerraum verbleibt, kann man als dreidimensionales "Verschnittproblem" auffassen. Solche Probleme treten in der Praxis häufig auf und sind recht schwierig.

(1.15) Fixkostenprobleme (englisch: **Fixed Charge Problem).** Gegeben sei folgendes Optimierungsproblem aus dem Bereich der Produktionsplanung. Unter Einsatz von m Faktorgütern sollen n Güter produziert werden. c_j sei der Ertrag pro Mengeneinheit des produzierten Gutes j. Hierbei werden zugleich a_{ij} Mengeneinheiten des Faktorguts i verbraucht. Zur Verfügung stehen vom i-ten Faktorgut höchstens b_i Mengeneinheiten. Wieviele Mengeneinheiten x_j hat man von jedem Gut j $(j = 1, \dots, n)$ zu produzieren, damit die Summe der Erträge maximal wird?

Zur Beantwortung dieser Frage hat man das lineare Programm

$$\begin{aligned}
\max \sum_{j=1}^{n} c_j x_j & \\
\sum_{j=1}^{n} a_{ij} x_j \leq b_i & \qquad i = 1, 2, \ldots, m \\
x_j \geq 0 & \qquad j = 1, \ldots, n
\end{aligned}$$

zu lösen. Falls von einem Teil der Güter nur ganze Einheiten produziert werden können, kommen für diese Güter noch Ganzzahligkeitsbedingungen hinzu.

In diesem Modell geht man von der in vielen Fällen unrealistischen Annahme aus, daß die Erträge proportional zur Menge des produzierten Gutes sind. Mit der Herstellung eines Gutes entstehen jedoch in der Regel auch fixe Kosten, die berücksichtigt werden müssen. Dies geschieht, indem wir die Ertragsfunktionen

$$f_j(x_j) = c_j x_j$$

durch

$$g_j(x_j) = \begin{cases} c_j x_j - d_j & \text{falls } x_j > 0 \\ 0 & \text{falls } x_j = 0 \end{cases}$$

mit $d_j > 0$ ersetzen. Das lineare Programm geht in diesem Fall über in ein gemischt-ganzzahliges lineares Programm der Form

$$\begin{aligned}
\max \sum_{j=1}^{n} (c_j x_j - d_j y_j) & \\
\sum_{j=1}^{n} a_{ij} x_j \leq b_i & \qquad i = 1, \ldots, m \\
x_j - M y_j \leq 0 & \qquad j = 1, \ldots, n \\
x_j \geq 0 & \qquad j = 1, \ldots, n \\
y_j \in \{0, 1\} & \qquad j = 1, \ldots, n\,.
\end{aligned}$$

Wie im vorigen Beispiel ist M eine sehr große positive Zahl. Die Nebenbedingungen $x_j - My_j \leq 0$ bewirken zusammen mit der Zielfunktion, daß $y_j = 1$ dann und nur dann gilt, wenn $x_j > 0$ ist.

□

Literaturverzeichnis von Kapitel 1

R. W. Ahuja, T. L. Magnanti und J. B. Orlin (1993), *Network flows: Theory, algorithms and applications*, Prentice Hall, Englewood Cliffs, N.J..

A. Bachem, M. Jünger, R. Schrader (1995), *Mathematik in der Praxis*, Springer-Verlag, Berlin.

P. Brucker (1975), *Ganzzahlige Lineare Programmierung mit ökonomischen Anwendungen*, Mathematical System in Economics 16, Anton Hain Verlag, Meisenheim.

M. Grötschel und L. Lovász (1995), "Combinatorial Optimization", R. L. Graham et al. (Hrsg.) *Handbook of Combinatorics*, Vol. II, Kap. 28 , 1541-1597, North-Holland, Amsterdam.

D. Jungnickel (1994), *Graphen, Netzwerke und Algorithmen*, BI Wissenschaftsverlag, Mannheim (3. Auflage).

G. L. Nemhauser und L. A. Wolsey (1988), *Integer and Combinatorial Optimization*, Wiley, New York.

A. Schrijver (1986), *Theory of linear and integer programming*, Wiley, New York.

Kapitel 2

Kombinatorische Optimierung und Graphentheorie

In diesem Kapitel wollen wir spezielle ganzzahlige Optimierungsprobleme betrachten, insbesondere solche, die sich mit Hilfe graphentheoretischer Konzepte formulieren lassen. Ferner geben wir eine kurze Einführung in die Terminologie der Graphentheorie.

2.1 Kombinatorische Optimierungsprobleme

In (sehr) allgemeiner Form kann man ein kombinatorisches Optimierungsproblem wie folgt definieren:

(2.1) Allgemeines kombinatorisches Optimierungsproblem. *Gegeben seien eine endliche Menge $\mathcal{I}$ und eine Funktion $f : \mathcal{I} \to \mathbb{K}$, die jedem Element von $\mathcal{I}$ einen "Wert" zuordnet. Gesucht ist ein Element $I^* \in \mathcal{I}$, so daß $f(I^*)$ so groß (oder klein) wie möglich ist.*

□

Eine Problemformulierung dieser Art ist allerdings relativ sinnlos, da über ein Problem, das wie oben gegeben ist, kaum vernünftige mathematische Aussagen gemacht werden können. Auch algorithmisch ist (2.1) auf triviale Weise lösbar: man durchlaufe alle Elemente I von $\mathcal{I}$, werte die Funktion $f(I)$ aus und wähle das Element I^* mit dem größten (oder kleinsten) Wert $f(I^*)$ aus. Falls die Elemente $I \in \mathcal{I}$ algorithmisch bestimmbar und $f(I)$ auswertbar ist, hat der eben beschriebene Enumerationsalgorithmus eine sogenannte lineare Laufzeit, da jedes Element von $\mathcal{I}$ nur einmal betrachtet wird.

Die üblicherweise auftretenden kombinatorischen Optimierungsprobleme sind jedoch auf andere, wesentlich strukturiertere Weise gegeben. Die Menge $\mathcal{I}$ ist nicht durch explizite Angabe aller Elemente spezifiziert sondern implizit durch die Angabe von Eigenschaften, die die Elemente von $\mathcal{I}$ haben sollen. Ebenso ist die Funktion f nicht punktweise sondern durch "Formeln" definiert.

In dieser Vorlesung wollen wir uns hauptsächlich auf den folgenden Problemtyp konzentrieren.

(2.2) Kombinatorisches Optimierungsproblem mit linearer Zielfunktion. *Gegeben seien eine endliche Menge E (genannt* **Grundmenge***), eine Teilmenge $\mathcal{I}$ der Potenzmenge 2^E von E (die Elemente von $\mathcal{I}$ heißen* **zulässige Mengen** *oder* **zulässige Lösungen***) und eine Funktion $c : E \to \mathbb{K}$. Für jede Menge $F \subseteq E$ definieren wir ihren "Wert" durch*

$$c(F) := \sum_{e \in F} c(e),$$

und wir suchen eine Menge $I^* \in \mathcal{I}$, so daß $c(I^*)$ so groß (oder klein) wie möglich ist.

□

Zur Notationsvereinfachung werden wir in Zukunft einfach **kombinatorisches Optimierungsproblem** sagen, wenn wir ein Problem des Typs (2.2) meinen. Da ein derartiges Problem durch die Grundmenge E, die zulässigen Lösungen $\mathcal{I}$ und die Zielfunktion c definiert ist, werden wir kurz von einem kombinatorischen Optimierungsproblem $(E, \mathcal{I}, c)$ sprechen.

Die Zielfunktion haben wir durch Formulierung (2.2) bereits sehr speziell strukturiert. Aber Problem (2.2) ist algorithmisch immer noch irrelevant, falls wir eine explizite Angabe von $\mathcal{I}$ unterstellen. Wir werden nachfolgend (und im Verlaufe der Vorlesung noch sehr viel mehr) Beispiele des Typs (2.2) kennenlernen. Fast alle der dort auftretenden zulässigen Mengen lassen sich durch folgende Form charakterisieren.

$$\mathcal{I} = \{I \subseteq E \mid I \text{ hat Eigenschaft } \Pi\}.$$

Wir werden uns damit beschäftigen, welche Eigenschaften die Eigenschaft Π haben muß, damit die zugehörigen Probleme $(E, \mathcal{I}, c)$ auf einfache Weise gelöst werden können. Nehmen wir an, daß E insgesamt n Elemente enthält, dann führt natürlich jede Eigenschaft Π, die impliziert, daß $\mathcal{I}$ (relativ zu n) nur sehr wenige Elemente enthält, dazu, daß $(E, \mathcal{I}, c)$ einfach lösbar ist. Typischerweise haben jedoch die interessanten kombinatorischen Optimierungsprobleme eine Anzahl von Lösungen, die exponentiell in n ist, etwa $n!$ oder 2^n. Eine vollständige Enumeration der Elemente solcher Mengen ist offenbar auch auf den größten Rechnern (für z. B. $n \geq 40$) nicht durchführbar. Das Ziel der kombinatorischen Optimierung besteht — kurz und vereinfachend gesagt — darin, Algorithmen zu entwerfen, die (erheblich) schneller als die Enumeration aller Lösungen sind.

Kombinatorische und ganzzahlige Optimierungsprobleme stehen in enger Beziehung. Wie wir gleich sehen werden, sind die kombinatorischen und binären Optimierungsprobleme in einem gewissen Sinne äquivalent. Mit welchem Namen man sie belegt, hängt im wesentlichen von der eigenen Sichtweise des Problems bzw. der Anwendung ab, aus der es stammt. Tritt ein Problem direkt in der Form (1.9) auf, so wird es meist "binär" genannt, während das Wort "kombinatorisch" meistens für Probleme des Typs (2.2) reserviert ist. Wir werden jedoch nun zeigen, daß sich die eine Problemformulierung in die andere umformen läßt.

Zunächst wollen wir einige Begriffe einführen. Ist E eine endliche Menge, so bezeichnen wir mit

$$\mathbb{K}^E$$

den $\mathbb{K}$-Vektorraum der $|E|$-Tupel (oder Spaltenvektoren der Länge $|E|$) $x = (x_e)_{e \in E}$, so daß jede Komponente von x mit einem Element von E indiziert ist. Das heißt, jedem Element $e \in E$ ist eine Komponente x_e von x zugeordnet und umgekehrt. Ist $F \subseteq E$, so heißt der Vektor $\chi^F \in \mathbb{K}^E$ mit

$$\chi_e^F = \begin{cases} 1, & \text{falls } e \in F \\ 0, & \text{falls } e \notin F \end{cases} \tag{2.3}$$

der **Inzidenzvektor** von F (oder der charakteristische Vektor von F). Umgekehrt ist jeder 0/1-Vektor $x \in \mathbb{K}^E$ (d. h. jeder Vektor, dessen Komponenten 0 oder 1 sind) charakteristischer Vektor einer Teilmenge F_x von E, und zwar gilt

$$(2.4) \qquad F_x := \{e \in E \mid x_e = 1\}, \quad \chi^{F_x} = x.$$

Ist nun ein binäres Optimierungsproblem, z. B. der Form (1.9) gegeben, d. h. durch eine Matrix $A \in \mathbb{K}^{(m,n)}$ und Vektoren $b \in \mathbb{K}^m$, $c \in \mathbb{K}^n$, so setzen wir:

$$(2.5) \qquad \begin{aligned} E &:= \{1, 2, \dots, n\} \\ \mathcal{I} &:= \{F_x \subseteq E \mid x \in \{0,1\}^E, Ax \leq b\} \\ c(F_x) &:= c^T \chi^{F_x}. \end{aligned}$$

Offenbar ist dann $(E, \mathcal{I}, c)$ ein kombinatorisches Optimierungsproblem, dessen Optimallösungen Mengen F_x sind, so daß die zugehörigen Vektoren $x = \chi^{F_x}$ Optimallösungen von (1.9) sind. Ist umgekehrt ein kombinatorisches Optimierungsproblem $(E, \mathcal{I}, c)$ gegeben, so setzen wir

$$(2.6) \qquad P_{\mathcal{I}} := \operatorname{conv}\{\chi^I \in \mathbb{K}^E \mid I \in \mathcal{I}\}.$$

Nach Definition ist $P_{\mathcal{I}}$ die konvexe Hülle endlich vieler Vektoren, also ein Polytop. Die Ecken von $P_{\mathcal{I}}$ sind genau die Inzidenzvektoren der zulässigen Mengen $I \in \mathcal{I}$. Fassen wir die Funktion $c : E \to \mathbb{K}$ als Vektor $c \in \mathbb{K}^E$ auf, so ist jede optimale Ecklösung des linearen Programms

$$(2.7) \qquad \max\{c^T x \mid x \in P_{\mathcal{I}}\}$$

der Inzidenzvektor einer Optimallösung von $(E, \mathcal{I}, c)$ (und umgekehrt). Wir haben somit das kombinatorische Optimierungsproblem $(E, \mathcal{I}, c)$ sogar in ein lineares Programm umformuliert, jedoch hat diese Formulierung einige Haken.

Das Problem (2.7) ist insofern ein lineares Programm, als eine lineare Zielfunktion über einem Polytop optimiert werden soll. Üblicherweise sind aber lineare Programme durch die Angabe von Gleichungen und Ungleichungen spezifiziert. Wir wissen zwar aus der Vorlesung "Lineare Optimierung", daß jedes Polytop der Form $P_{\mathcal{I}}$ eine Darstellung durch Gleichungen und Ungleichungen hat, jedoch gibt es keine effiziente Methode, um eine derartige Darstellung zu berechnen. Weit häufiger gelingt es, relativ "einfache" Darstellungen von $(E, \mathcal{I}, c)$ als ganzzahliges Programm zu bestimmen. Wir werden später anhand vieler Einzelbeispiele genauer darauf eingehen, möchten jedoch jetzt schon darauf hinweisen, daß das Studium der Polyeder $P_{\mathcal{I}}$ des Typs (2.6) zu sehr interessanten Forschungsproblemen mit Anwendungsbezug führt. Im Kapitel "Polyedrische Kombinatorik" werden wir diesen Themenkreis ausführlich behandeln.

2.2 Grundbegriffe der Graphen-, Hypergraphen- und Matroidtheorie

Die meisten kombinatorischen Optimierungsprobleme lassen sich mit Hilfe graphentheoretischer Konzepte definieren. Die Terminologie und Notation in der Graphentheorie ist aber leider sehr uneinheitlich. Wir wollen daher hier einen kleinen Katalog wichtiger graphentheoretischer Begriffe und Bezeichnungen zusammenstellen und zwar in der Form, wie sie im weiteren benutzt werden sollen. Definitionen werden durch Fettdruck hervorgehoben. Nach einer Definition folgen gelegentlich (in Klammern) weitere Bezeichnungen, um auf alternative Namensgebungen in der Literatur hinzuweisen. Wir werden ferner noch einige der Grundbegriffe der Hypergraphen- und der Matroidtheorie auflisten.

Es gibt sehr viele Bücher über Graphentheorie. Zur Einführung in die mathematische Theorie empfehlen wir u. a. Aigner (1984), Bollobás (1979) oder Bondy und Murty (1976). Stärker algorithmisch orientiert und anwendungsbezogen sind z. B. Ebert (1981), Golumbic (1980) und Walther und Nägler (1987). Umfassende Information über Kombinatorik in Form von Übersichtsartikeln – darunter auch über Graphen, Hypergraphen, Matroide, Komplexitätstheorie und kombinatorische Optimierung – findet sich in Graham, Grötschel und Lovász (1995).

Graphen

Ein **Graph** G ist ein Tripel (V, E, Ψ) bestehend aus einer nicht-leeren Menge V, einer Menge E und einer **Inzidenzfunktion** $\Psi : E \to V^{(2)}$. Hierbei bezeichnet $V^{(2)}$ die Menge der ungeordneten Paare von (nicht notwendigerweise verschiedenen) Elementen von V. Ein Element aus V heißt **Knoten** (oder Ecke oder Punkt oder Knotenpunkt; englisch: vertex oder node oder point), ein Element aus E heißt **Kante** (englisch: edge oder line). Zu jeder Kante $e \in E$ gibt es also Knoten $u, v \in V$ mit $\Psi(e) = uv = vu$. (In der Literatur werden auch die Symbole $[u, v]$ oder $\{u, v\}$ zur Bezeichnung des ungeordneten Paares uv benutzt.)

Die Anzahl der Knoten eines Graphen heißt **Ordnung** des Graphen. Ein Graph heißt **endlich**, wenn V und E endliche Mengen sind, andernfalls heißt G **unendlich**. Wir werden uns nur mit endlichen Graphen beschäftigen und daher ab jetzt statt "endlicher Graph" einfach "Graph" schreiben. Wie werden versuchen, die natürliche Zahl n für die Knotenzahl und die natürliche Zahl m für die Kantenzahl eines Graphen zu reservieren.

Gilt $\Psi(e) = uv$ für eine Kante $e \in E$, dann heißen die Knoten $u, v \in V$ **Endknoten** von e, und wir sagen, daß u und v mit e **inzidieren** oder **auf** e **liegen**, daß e die Knoten u und v **verbindet**, und daß u und v **Nachbarn** bzw. **adjazent** sind. Wir sagen auch, daß zwei Kanten **inzident** sind, wenn sie einen gemeinsamen Endknoten haben. Eine Kante e mit $\Psi(e) = uu$ heißt **Schlinge**; Kanten e, f mit $\Psi(e) = uv = \Psi(f)$ heißt **parallel**, man nennt in diesem Falle die Kante mit den Endknoten u und v eine **Mehrfachkante**. Graphen, die weder Mehrfachkanten noch Schlingen enthalten, heißen **einfach**. Der einfache Graph, der zu jedem in G adjazenten Knotenpaar u, v mit $u \neq v$ genau eine u und v verbindende Kante enthält, heißt der G **unterliegende einfache Graph**. Mit $\Gamma(v)$ bezeichnen wir die Menge der Nachbarn eines Knotens v. Falls v in einer Schlinge enthalten ist, ist v

natürlich mit sich selbst benachbart. $\Gamma(W) := \bigcup_{v\in W}\Gamma(v)$ ist die Menge der Nachbarn von $W \subseteq V$. Ein Knoten ohne Nachbarn heißt **isoliert**.

Die Benutzung der Inzidenzfunktion Ψ führt zu einem relativ aufwendigen Formalimus. Wir wollen daher die Notation etwas vereinfachen. Dabei entstehen zwar im Falle von nicht-einfachen Graphen gelegentlich Mehrdeutigkeiten, die aber i. a. auf offensichtliche Weise interpretiert werden können. Statt $\Psi(e) = uv$ schreiben wir von nun an einfach $e = uv$ (oder äquivalent $e = vu$) und meinen damit die Kante e mit den Endknoten u und v. Das ist korrekt, solange es nur eine Kante zwischen u und v gibt. Gibt es mehrere Kanten mit den Endknoten u und v, und sprechen wir von der Kante uv, so soll das heißen, daß wir einfach eine der parallelen Kanten auswählen. Von jetzt an vergessen wir also die Inzidenzfunktion Ψ und benutzen die Abkürzung $G = (V, E)$, um einen Graphen zu bezeichnen. Manchmal schreiben wir auch E_G oder $E(G)$ bzw. V_G oder $V(G)$ zur Bezeichnung der Kanten- bzw. Knotenmenge eines Graphen G.

Zwei Graphen $G = (V, E)$ und $H = (W, F)$ heißen **isomorph**, wenn es eine bijektive Abbildung $\varphi : V \to W$ gibt, so daß $uv \in E$ genau dann gilt, wenn $\varphi(u)\varphi(v) \in F$ gilt. Isomorphe Graphen sind also — bis auf die Benamung der Knoten und Kanten — identisch.

Eine Menge F von Kanten heißt **Schnitt**, wenn es eine Knotenmenge $W \subseteq V$ gibt, so daß $F = \delta(W) := \{uv \in E \mid uv \in W, v \in V \setminus W\}$ gilt; manchmal wird $\delta(W)$ der **durch W induzierte Schnitt** genannt. Statt $\delta(\{v\})$ schreiben wir kurz $\delta(v)$. Ein Schnitt, der keinen anderen nicht-leeren Schnitt als echte Teilmenge enthält, heißt **Cokreis**. Wollen wir betonen, daß ein Schnitt $\delta(W)$ bezüglich zweier Knoten $s, t \in V$ die Eigenschaft $s \in W$ und $t \in V \setminus W$ hat, so sagen wir, $\delta(W)$ ist ein s **und** t **trennender Schnitt** oder kurz ein $[s, t]$**-Schnitt**.

Generell benutzen wir die eckigen Klammern [. , .], um anzudeuten, daß die Reihenfolge der Objekte in der Klammer ohne Bedeutung ist. Z. B. ist ein $[s, t]$-Schnitt natürlich auch ein $[t, s]$-Schnitt, da ja $\delta(W) = \delta(V \setminus W)$ gilt.

Wir haben oben Bezeichnungen wie $\Gamma(v)$ oder $\delta(W)$ eingeführt unter der stillschweigenden Voraussetzung, daß man weiß, bezüglich welches Graphen diese Mengen definiert sind. Sollten mehrere Graphen involviert sein, so werden wir, wenn Zweideutigkeiten auftreten können, die Graphennamen als Indizes verwenden, also z. B. $\Gamma_G(v)$ oder $\delta_G(V)$ schreiben. Analog wird bei allen anderen Symbolen verfahren.

Der **Grad** eines Knoten v (Bezeichnung: $\deg(v)$) ist die Anzahl der Kanten, mit denen er inzidiert, wobei Schlingen doppelt gezählt werden. Hat ein Graph keine Schlingen, so ist der Grad von v gleich $\delta(v)$. Ein Graph heißt k **-regulär**, wenn jeder Knoten den Grad k hat, oder kurz **regulär**, wenn der Grad k nicht hervorgehoben werden soll.

Sind W eine Knotenmenge und F eine Kantenmenge in $G = (V, E)$, dann bezeichnen wir mit $E(W)$ die Menge aller Kanten von G mit beiden Endknoten in W und mit $V(F)$ die Menge aller Knoten, die Endknoten mindestens einer Kante aus F sind.

Sind $G = (V, E)$ und $H = (W, F)$ zwei Graphen, so heißt der Graph $(V \cup W, E \cup F)$ die **Vereinigung** von G und H, und $(V \cap W, E \cap F)$ heißt der **Durchschnitt** von G und H. G und H heißen **disjunkt**, falls $V \cap W = \emptyset$, **kantendisjunkt**, falls $E \cap F = \emptyset$. Wir sprechen von einer **disjunkten** bwz. **kantendisjunkten Vereinigung** von zwei Graphen, wenn sie disjunkt bzw. kantendisjunkt sind.

Sind $G = (V, E)$ und $H = (W, F)$ Graphen, so daß $W \subseteq V$ und $F \subseteq E$ gilt, so heißt H **Untergraph** (oder Teilgraph) von G. Falls $W \subseteq V$, so bezeichnet $G - W$ den Graphen, den man durch **Entfernen** (oder Subtrahieren) aller Knoten in W und aller Kanten mit mindestens einem Endknoten in W gewinnt. $G[W] := G - (V \setminus W)$ heißt der **von W induzierte Untergraph** von G. Es gilt also $G[W] = (W, E(W))$. Für $F \subseteq E$ ist $G - F := (V, E \setminus F)$ der Graph, den man durch **Entfernen** (oder Subtrahieren) der Kantenmenge F enthält. Statt $G - \{f\}$ schreiben wir $G - f$, analog schreiben wir $G - w$ statt $G - \{w\}$ für $w \in V$. Ein Untergraph $H = (W, F)$ von $G = (V, E)$ heißt **aufspannend**, falls $V = W$ gilt.

Ist $G = (V, E)$ ein Graph und $W \subseteq V$ eine Knotenmenge, so bezeichnen wir mit $G \cdot W$ den Graphen, der durch **Kontraktion der Knotenmenge** W entsteht. Das heißt, die Knotenmenge von $G \cdot W$ besteht aus den Knoten $V \setminus W$ und einem neuen Knoten w, der die Knotenmenge W ersetzt. Die Kantenmenge von $G \cdot W$ enthält alle Kanten von G, die mit keinem Knoten aus W inzidieren, und alle Kanten, die genau einen Endknoten in W haben, aber dieser Endknoten wird durch w ersetzt (also können viele parallele Kanten entstehen). Keine der Kanten von G, die in $E(W)$ liegen, gehört zu $G \cdot W$. Falls $e = uv \in E$ und falls G keine zu e parallele Kante enthält, dann ist der Graph, der durch **Kontraktion von** e entsteht (Bezeichnung $G \cdot e$), der Graph $G \cdot \{u, v\}$. Falls G zu e parallele Kanten enthält, so erhält man $G \cdot e$ aus $G \cdot \{u, v\}$ durch Addition von so vielen Schlingen, die den neuen Knoten w enthalten, wie es Kanten in G parallel zu e gibt. Der Graph $G \cdot F$, den man durch **Kontraktion einer Kantenmenge** $F \subseteq E$ erhält, ist der Graph, der durch sukzessive Kontraktion (in beliebiger Reihenfolge) der Kanten aus F gewonnen wird. Ist e eine Schlinge von G, so sind $G \cdot e$ und $G - e$ identisch.

Ein einfacher Graph heißt **vollständig**, wenn jedes Paar seiner Knoten durch eine Kante verbunden ist. Offenbar gibt es — bis auf Isomorphie — nur einen vollständigen Graphen mit n Knoten. Dieser wird mit K_n bezeichnet. Ein Graph G, dessen Knotenmenge V in zwei disjunkte nicht-leere Teilmengen V_1, V_2 mit $V_1 \cup V_2 = V$ zerlegt werden kann, so daß keine zwei Knoten in V_1 und keine zwei Knoten in V_2 benachbart sind, heißt **bipartit** (oder paar). Die Knotenmengen V_1, V_2 nennt man eine **Bipartition** (oder 2-Färbung) von G. Falls G zu je zwei Knoten $u \in V_1$ und $v \in V_2$ genau eine Kante uv enthält, so nennt man G **vollständig bipartit**. Den — bis auf Isomorphie eindeutig bestimmten — vollständig bipartiten Graphen mit $|V_1| = m, |V_2| = n$ bezeichnen wir mit $K_{m,n}$.

Ist G ein Graph, dann ist das **Komplement** von G, bezeichnet mit $\overline{G}$, der einfache Graph, der dieselbe Knotenmenge wie G hat und bei dem zwei Knoten genau dann durch eine Kante verbunden sind, wenn sie in G nicht benachbart sind. Ist G einfach, so gilt $\overline{\overline{G}} = G$. Der **Kantengraph** (englisch: line graph) $L(G)$ eines Graphen G ist der einfache Graph, dessen Knotenmenge die Kantenmenge von G ist und bei dem zwei Knoten genau dann adjazent sind, wenn die zugehörigen Kanten in G einen gemeinsamen Endknoten haben.

Eine **Clique** in einem Graphen G ist eine Knotenmenge Q, so daß je zwei Knoten aus Q in G benachbart sind. Eine **stabile Menge** in einem Graphen G ist eine Knotenmenge S, so daß je zwei Knoten aus S in G nicht benachbart sind. Für stabile Mengen werden auch die Begriffe unabhängige Knotenmenge oder Coclique verwendet. Eine Knotenmenge K in G heißt **Knotenüberdeckung** (oder Überdeckung von Kanten durch Knoten), wenn jede Kante aus G mit mindestens einem Knoten in K inzidiert. Die größte Kardinalität (= Anzahl der Elemente) einer stabilen Menge (bzw. Clique) in einem Graphen bezeichnet man mit $\alpha(G)$ (bzw. $\omega(G)$); die kleinste Kardinalität einer Knotenüberdeckung mit $\tau(G)$.

Eine Kantenmenge M in G heißt **Matching** (oder Paarung oder Korrespondenz oder unabhängige Kantenmenge), wenn M keine Schlingen enthält und je zwei Kanten in M keinen gemeinsamen Endknoten besitzen. M heißt **perfekt**, wenn jeder Knoten von G Endknoten einer Kante des Matchings M ist. Ein perfektes Matching wird auch **1-Faktor** genannt. Eine Kantenmenge F in G heißt **k-Faktor**, wenn jeder Knoten von G in genau k Kanten aus F enthalten ist. Eine **Kantenüberdeckung** (oder Überdeckung von Knoten durch Kanten) ist eine Kantenmenge, so daß jeder Knoten aus G mit mindestens einer Kante diese Menge inzidiert. Die größte Kardinalität eines Matchings in G bezeichnet man mit $\nu(G)$, die kleinste Kardinalität einer Kantenüberdeckung mit $\rho(G)$.

Eine Zerlegung der Knotenmenge eines Graphen in stabile Mengen, die sogenannten **Farbklassen**, heißt **Knotenfärbung**; d. h., die Knoten werden so gefärbt, daß je zwei benachbarte Knoten eine unterschiedliche Farbe haben. Eine Zerlegung der Kantenmenge in Matchings heißt **Kantenfärbung**; die Kanten werden also so gefärbt, daß je zwei inzidente Kanten verschieden gefärbt sind. Eine Zerlegung der Knotenmenge von G in Cliquen heißt **Cliquenübedeckung** von G. Die minimale Anzahl von stabilen Mengen (bzw. Cliquen) in einer Knotenfärbung (bzw. Cliquenüberdeckung) bezeichnet man mit $\chi(G)$ (bzw. $\overline{\chi}(G)$), die minimale Anzahl von Matchings in einer Kantenfärbung mit $\gamma(G)$. Die Zahl $\gamma(G)$ heißt **chromatischer Index** (oder Kantenfärbungszahl), $\chi(G)$ **Färbungszahl** (oder Knotenfärbungszahl oder chromatische Zahl).

Ein Graph $G = (V, E)$ kann in die Ebene gezeichnet werden, indem man jeden Knoten durch einen Punkt repräsentiert und jede Kante durch eine Kurve (oder Linie oder Streckenstück), die die beiden Punkte verbindet, die die Endknoten der Kante repräsentieren. Ein Graph heißt **planar** (oder plättbar), falls er in die Ebene gezeichnet werden kann, so daß sich keine zwei Kanten (d. h. die sie repräsentierenden Kurven) schneiden — außer möglicherweise in ihren Endknoten. Eine solche Darstellung eines planaren Graphen G in der Ebene nennt man auch **Einbettung** von G in die Ebene.

Digraphen

Die Kanten eines Graphen haben keine Orientierung. In vielen Anwendungen spielen aber Richtungen eine Rolle. Zur Modellierung solcher Probleme führen wir gerichtete Graphen ein. Ein **Digraph** (oder **gerichteter Graph**) $D = (V, A)$ besteht aus einer (endlichen) nicht-leeren **Knotenmenge** V und einer (endlichen) Menge A von **Bögen** (oder gerichteten Kanten; englisch: arc). Eine **Bogen** a ist ein geordnetes Paar von Knoten, also

$a = (u, v)$, u ist der **Anfangs-** oder **Startknoten**, v der **End-** oder **Zielknoten** von a; u heißt **Vorgänger von** v, v **Nachfolger** von u, a **inzidiert mit** u und v. (Um exakt zu sein, müßten wir hier ebenfalls eine Inzidenzfunktion $\Psi = (t, h) : A \to V \times V$ einführen. Für einen Bogen $a \in A$ ist dann $t(a)$ der Anfangsknoten (englisch: tail) und $h(a)$ der Endknoten (englisch: head) von a. Aus den bereits oben genannten Gründen wollen wir jedoch die Inzidenzfunktion nur in Ausnahmefällen benutzen.) Wie bei Graphen gibt es auch hier **parallele Bögen** und **Schlingen**. Die Bögen (u, v) und (v, u) heißen **antiparallel**.

In manchen Anwendungsfällen treten auch "Graphen" auf, die sowohl gerichtete als auch ungerichtete Kanten enthalten. Wir nennen solche Objekte **gemischte Graphen** und bezeichnen einen gemischten Graphen mit $G = (V, E, A)$, wobei V die Knotenmenge, E die Kantenmenge und A die Bogenmenge von G bezeichnet.

Falls $D = (V, A)$ ein Digraph ist und $W \subseteq V, B \subseteq A$, dann bezeichnen wir mit $A(W)$ die Menge der Bögen, deren Anfangs- und Endknoten in W liegen, und mit $V(B)$ die Menge der Knoten, die als Anfangs- oder Endknoten mindestens eines Bogen in B auftreten. Unterdigraphen, induzierte Unterdigraphen, aufspannende Unterdigraphen, Vereinigung und Durchschnitt von Digraphen, das Entfernen von Bogen- oder Knotenmengen und die Kontraktion von Bogen- oder Knotenmengen sind genau wie bei Graphen definiert.

Ist $D = (V, A)$ ein Digraph, dann heißt der Graph $G = (V, E)$, der für jeden Bogen $(i, j) \in A$ eine Kante ij enthält, der D **unterliegende Graph**. Analog werden der D **unterliegende einfache Graph** und der **unterliegende einfach Digraph** definiert. Wir sagen, daß ein Digraph eine "ungerichtete" Eigenschaft hat, wenn der ihm unterliegende Graph diese Eigenschaft hat (z. B., D ist bipartit oder planar, wenn der D unterliegende Graph G bipartit oder planar ist). Geben wir jeder Kante ij eines Graphen G eine Orientierung, d. h., ersetzen wir ij durch einer der Bögen (i, j) oder (j, i), so nennen wir den so entstehenden Digraphen D **Orientierung** von G.

Ein einfacher Digraph heißt **vollständig**, wenn je zwei Knoten $u \neq v$ durch die beiden Bögen $(u, v), (v, u)$ verbunden sind. Ein **Turnier** ist ein Digraph, der für je zwei Knoten $u \neq v$ genau einen der Bögen (u, v) oder (v, u) enthält. (Der einem Turnier unterliegende Graph ist also ein vollständiger Graph; jedes Turnier ist die Orientierung eines vollständigen Graphen.)

Für $W \subseteq V$ sei $\delta^+(W) := \{(i, j) \in A \mid i \in W, j \notin W\}, \delta^-(W) := \delta^+(V \setminus W)$ und $\delta(W) := \delta^+(W) \cup \delta^-(W)$. Die Bogenmenge $\delta^+(W)$ (bzw. $\delta^-(W)$) heißt **Schnitt**. Ist $s \in W$ und $t \notin W$, so heißt $\delta^+(W)$ auch **(s, t)-Schnitt**. (Achtung: in einem Digraphen ist ein (s, t)-Schnitt kein (t, s)-Schnitt!) Statt $\delta^+(\{v\}), \delta^-(\{v\}), \delta(\{v\})$ schreiben wir $\delta^+(v), \delta^-(v), \delta(v)$, Der **Außengrad** (**Innengrad**) von v ist die Anzahl der Bögen mit Anfangsknoten (Endknoten) v. Die Summe von Außengrad und Innengrad ist der **Grad** von v. Ein Schnitt $\delta^+(W), \emptyset \neq W \neq V$, heißt **gerichteter Schnitt**, falls $\delta^-(W) = \emptyset$, d. h. falls $\delta(W) = \delta^+(W)$. Ist $r \in W$, so sagen wir auch, daß $\delta^+(W)$ ein **Schnitt mit Wurzel** r ist.

Ketten, Wege, Kreise, Bäume

Das größte Durcheinander in der graphentheoretischen Terminologie herrscht bei den Begriffen Kette, Weg, Kreis und bei den damit zusammenhängenden Namen. Wir haben uns für folgende Bezeichnungen entschieden.

In einem Graphen oder Digraphen heißt eine endliche Folge $W = (v_0, e_1, v_1, e_2, v_2, \cdots, e_k, v_k), k \geq 0$, die mit einem Knoten beginnt und endet und in der Knoten und Kanten (Bögen) alternierend auftreten, so daß jede Kante (jeder Bogen) e_i mit den beiden Knoten v_{i-1} und v_i inzidiert, eine **Kette**. Der Knoten v_0 heißt **Anfangsknoten**, v_k **Endknoten** der Kette; die Knoten $v_1, \ldots, v_{k-1}$ heißen **innere Knoten**; W wird auch $[v_0, v_i]$-**Kette** genannt. Die Zahl k heißt **Länge** der Kette (= Anzahl der Kanten bzw. Bögen in w). Falls (in einem Digraphen) alle Bögen e_i der Kette W der Form (v_{i-1}, v_i) (also gleichgerichtet) sind, so nennt man W **gerichtete Kette** bzw. (v_0, v_k)**-Kette**. Ist $W = (v_o, e_1, v_1, \ldots, e_k, v_k)$ eine Kette, und sind i, j Indizes mit $0 \leq i < j \leq k$, dann heißt die Kette $(v_i, e_{i+1}, v_{i+1}, \ldots, e_j, v_j)$ das $[v_i, v_j]$**-Segment** (bzw. (v_i, v_j)**-Segment**, wenn W gerichtet ist) von W. Jede (gerichtete) Kante, die zwei Knoten der Kette W miteinander verbindet, die aber nicht Element von W ist, heißt **Diagonale** (oder Sehne) von W.

Gibt es in einem Graphen keine parallelen Kanten, so ist eine Kette W bereits durch die Folge $(v_0, \ldots, v_k)$ ihrer Knoten eindeutig festgelegt. Desgleichen ist in einem Digraphen ohne parallele Bögen eine gerichtete Kette durch die Knotenfolge $(v_0, \ldots, v_k)$ bestimmt. Zur Bezeichnungsvereinfachung werden wir daher häufig von der Kette $(v_0, \ldots, v_k)$ in einem Graphen bzw. der gerichteten Kette $(v_0, \ldots, v_k)$ ein einem Digraphen sprechen, obgleich bei parallelen Kanten (Bögen) die benutzten Kanten (Bögen) hiermit nicht eindeutig festgelegt sind. Diese geringfügige Ungenauigkeit sollte aber keine Schwierigkeiten bereiten. Gelegentlich interessiert man sich mehr für die Kanten (Bögen) einer Kette, insbesondere wenn diese ein Weg oder ein Kreis (siehe unten) ist. In solchen Fällen ist es zweckmäßiger, eine Kette als Kantenfolge $(e_1, e_2, \ldots, e_k)$ zu betrachten. Ist C die Menge der Kanten (Bögen) eines Kreises oder eines Weges, so spricht man dann einfach vom Kreis oder Weg C, während $V(C)$ die Menge der Knoten des Kreises oder Weges bezeichnet. Je nach behandeltem Themenkreis wird hier die am besten geeignete Notation benutzt.

Eine Kette, in der alle Knoten voneinander verschieden sind, heißt **Weg**. Eine Kette, in der alle Kanten oder Bögen verschieden sind, heißt **Pfad**. Ein Weg ist also ein Pfad, aber nicht jeder Pfad ist ein Weg. Ein Weg oder Pfad in einem Digraphen, der eine gerichtete Kette ist, heißt **gerichteter Weg** oder **gerichteter Pfad**. Wie bei Ketten sprechen wir von $[u, v]$**-Wegen**, (u, v)**-Wegen** etc.

Im Englischen heißt Kette walk oder chain. Im Deutschen benutzen z. B. Domschke (1982), Hässig (1979) und Berge und Ghouila-Houri (1969) ebenfalls das Wort Kette, dagegen schreiben Aigner (1984) und Wagner (1970) hierfür "Kantenzug", während König (1936), Halin (1989) und Sachs (1970) "Kantenfolge" benutzen; Ebert (1981) schließlich nennt unsere Ketten "ungerichtete Pfade". Dieses Wirrwarr setzt sich bezüglich der Begriffe Pfad und Weg auf ähnliche Weise fort.

Eine Kette heißt **geschlossen**, falls ihre Länge nicht Null ist und falls ihr Anfangsknoten mit ihrem Endknoten übereinstimmt. Eine geschlossene (gerichtete) Kette, in der der Anfangsknoten und alle inneren Knoten voneinander verschieden sind, heißt **Kreis** (**gerichteter Kreis**). Die Anzahl der Kanten (Bögen) eines Kreises ist seine **Länge**.

Eine (gerichtete) Kette, die jede Kante (jeden Bogen) eines Graphen (Digraphen) genau einmal enthält, heißt (gerichteter) **Eulerpfad**. Ein geschlossener Eulerpfad heißt **Eulertour**. Ein **Eulergraph** (Eulerdigraph) ist ein Graph (Digraph), der eine (gerichtete) Eulertour enthält.

Ein (gerichteter) Kreis (Weg) der Länge $|V|$ (bzw. $|V|-1$) heißt (gerichteter) **Hamiltonkreis** (**Hamiltonweg**). Ein Graph (Digraph), der einen (gerichteten) Hamiltonkreis enthält, heißt **hamiltonsch**. Manchmal sagen wir statt Hamiltonkreis einfach **Tour**.

Ein **Wald** ist ein Graph, der keinen Kreis enthält. Ein zusammenhängender Wald heißt **Baum**. Ein Baum in einem Graphen heißt **aufspannend**, wenn er alle Knoten des Graphen enthält. Ein **Branching** B ist ein Digraph, der ein Wald ist, so daß jeder Knoten aus B Zielknoten von höchstens einem Bogen von B ist. Ein zusammenhängendes Branching heißt **Arboreszenz**. Eine **aufspannende Arboreszenz** ist eine Arboreszenz in einem Digraphen D, die alle Knoten von D enthält. Eine Arboreszenz enthält einen besonderen Knoten, genannt **Wurzel**, von dem aus jeder andere Knoten auf genau einem gerichteten Weg erreicht werden kann. Arboreszenzen werden auch **Wurzelbäume** genannt. Ein Digraph, der keinen gerichteten Kreis enthält, heißt **azyklisch**.

Ein Graph heißt **zusammenhängend**, falls es zu jedem Paar von Knoten s, t einen $[s, t]$-Weg in G gibt. Ein Digraph D heißt **stark zusammenhängend**, falls es zu je zwei Knoten s, t von D sowohl einen gerichteten (s, t)-Weg als auch einen gerichteten (t, s)-Weg in D gibt. Die **Komponenten** (**starken Komponenten**) eines Graphen (Digraphen) sind die bezüglich Kanteninklusion (Bogeninklusion) maximalen zusammenhängenden Untergraphen von G (maximalen stark zusammenhängenden Unterdigraphen von D). Eine Komponente heißt **ungerade Komponente**, falls ihre Knotenzahl ungerade ist, andernfalls heißt sie **gerade Komponente**.

Sei $G = (V, E)$ ein Graph. Eine Knotenmenge $W \subseteq V$ heißt **trennend**, falls $G - W$ unzusammenhängend ist. Für Graphen $G = (V, E)$, die keinen vollständigen Graphen der Ordnung $|V|$ enthalten, setzen wir $\kappa(G) := \min\{|W| \mid W \subseteq V \text{ ist trennend}\}$. Die Zahl $\kappa(G)$ heißt **Zusammenhangszahl** (oder Knotenzusammenhangszahl) von G. Für jeden Graphen $G = (V, E)$, der einen vollständigen Graphen der Ordnung $|V|$ enthält, setzen wir $\kappa(G) := |G| - 1$. Falls $\kappa(G) \geq k$, so nennen wir G **k-fach knotenzusammenhängend** (kurz: **k-zusammenhängend**). Ein wichtiger Satz der Graphentheorie (Satz von Menger) besagt, daß G k-fach zusammenhängend genau dann ist, wenn jedes Paar $s, t, s \neq t$, von Knoten durch mindestens k knotendisjunkte $[s, t]$-Wege miteinander verbunden ist. (Eine Menge von $[s, t]$-Wegen heißt **knotendisjunkt**, falls keine zwei Wege einen gemeinsamen inneren Knoten besitzen und die Menge der in den $[s, t]$-Wegen enthaltenen Kanten keine parallelen Kanten enthält.)

Eine Kantenmenge F eines Graphen $G = (V, E)$ heißt **trennend**, falls $G - F$ unzusammenhängend ist. Für Graphen G, die mehr als einen Knoten enthalten, setzen wir $\lambda(G) := \min\{|F| \mid F \subseteq E \text{ trennend}\}$. Die Zahl $\lambda(G)$ heißt **Kantenzusammenhangszahl**. Für Graphen G mit nur einem Knoten setzen wir $\lambda(G) = 0$. Falls $\lambda(G) \geq k$, so nennen wir G $k-$**fach kantenzusammenhängend** (kurz: k-**kantenzusammenhängend**). Eine Version des Menger'schen Satzes besagt, daß G k-kantenzusammenhängend genau dann ist, wenn jedes Paar $s, t, s \neq t$, von Knoten durch mindestens k kantendisjunkte $[s, t]$-Wege verbunden ist. Für Graphen G mit mindestens einem Knoten sind die Eigenschaften "G ist zusammenhängend", "G ist 1-kantenzusammenhängend" äquivalent.

Analoge Konzepte kann man in Digraphen definieren. Man benutzt hierbei den Zusatz "stark", um den "gerichteten Zusammenhang" zu kennzeichnen. Wir sagen, daß ein Digraph $D = (V, A)$ **stark** k-**zusammenhängend** (bzw. stark k-**bogenzusammenhängend**) ist, falls jedes Knotenpaar $s, t, s \neq t$ durch mindestens k knotendisjunkte (bzw. bogendisjunkte) (s, t)-Wege verbunden ist. Wir setzen $\vec{\kappa}(D) := \max\{k \mid D \text{ stark } k\text{-zusammenhängend }\}$ und $\vec{\lambda}(D) := \max\{k \mid D \text{ stark } k\text{-bogenzusammenhängend}\}$; $\vec{\lambda}(D)$ heißt die **starke Zusammenhangszahl** von D, $\vec{\lambda}(D)$ die **starke Bogenzusammenhangszahl** von D.

Ein Kante e von G heißt **Brücke** (oder Isthmus), falls $G - e$ mehr Komponenten als G hat. Ein Knoten v von G heißt **Trennungsknoten** (oder Artikulation), falls die Kantenmenge E von G so in zwei nicht-leere Teilmengen E_1 und E_2 zerlegt werden kann, daß $V(E_1) \cap V(E_2) = \{v\}$ gilt. Ist G schlingenlos mit $|V| \geq 2$, dann ist v ein Trennungsknoten genau dann, wenn $\{v\}$ eine trennende Knotenmenge ist, d. h. wenn $G - v$ mehr Komponenten als G besitzt. Ein zusammenhängender Graph ohne Trennungsknoten wird **Block** genannt. Blöcke sind entweder Schlingen oder Graphen mit 2 Knoten, die durch eine Kante oder mehrere parallele Kanten verbunden sind oder, falls $|V| \geq 3$, 2-zusammenhängende Graphen. Ein **Block eines Graphen** ist ein Untergraph, der ein Block und maximal bezüglich dieser Eigenschaft ist. Jeder Graph ist offenbar die Vereinigung seiner Blöcke.

Hypergraphen

Sei V eine endliche Menge und $\mathcal{E} = (E_i \mid i \in I)$ eine endliche Familie von Teilmengen von V. $\mathcal{E}$ heißt endlicher **Hypergraph auf** auf V, falls $E_i \neq \emptyset$ für alle $i \in I$ und $\bigcup_{i \in I} E_i = V$ gilt. Das Paar $(V, \mathcal{E})$ heißt **Hypergraph**. $|V|$ ist die **Ordnung** des Hypergraphen, die Elemente von V heißen **Knoten**, die Mengen E_i **Kanten** (oder Hyperkanten). Hyperkanten sind also "Kanten", die mehr als zwei Knoten besitzen können.

Die Kanten mit genau einem Element heißen **Schlingen**. Ein Hypergraph heißt **einfach**, wenn alle Kanten voneinander verschieden sind. In diesem Falle ist $\mathcal{E}$ eine Teilmenge von 2^V. (2^V bezeichnet die Potenzmenge von V.)

Jeder Graph ohne isolierte Knoten kann also als Hypergraph aufgefaßt werden, dessen Kanten höchstens 2 Elemente enthalten. Ein einfacher Graph ohne isolierte Knoten ist ein einfacher Hypergraph mit $|E_i| = 2$ für alle $i \in I$.

Es ist gelegentlich nützlich, graphentheoretische Objekte als Hypergraphen aufzufassen. Sei z. B. $D = (V, A)$ ein Digraph, und seien $s, t \in V$ zwei festgewählte Knoten. Sei $B \subseteq A$ die Menge der Bögen, die auf mindestens einem (s, t)-Weg liegen. Seien $\mathcal{E}_1 := \{P \subseteq B \mid P \text{ ist die Bogenmenge eines } (s, t)\text{-Weges }\}$, $\mathcal{E}_2 := \{C \subseteq B \mid C \text{ ist ein } (s, t)\text{-Schnitt in } (V, B)\}$, dann sind $(B, \mathcal{E}_1)$ und $(B, \mathcal{E}_2)$ zwei interessante Hypergraphen, die uns in der Theorie der Blocker wiederbegegnen werden. In dieser Theorie sind besonders **Antiketten** (oder Clutter) von Interesse. Das sind Hypergraphen $(V, \mathcal{E})$, so daß für je zwei Kanten $E, F \in \mathcal{E}$ weder $E \subseteq F$ noch $F \subseteq E$ gilt. Der oben definierte Hypergraph $(B, \mathcal{E}_1)$ ist für jeden Digraphen D eine Antikette, dies gilt jedoch nicht für den Hypergraphen $(B, \mathcal{E}_2)$. Die Theorie der Hypergraphen ist ausführlich in dem Buch Berge (1989) dargestellt.

Matroide, Unabhängigkeitssysteme

Ist E eine endliche Menge und $\mathcal{I} \subseteq 2^E$ eine Menge von Teilmengen von E, dann heißt das Paar $(E, \mathcal{I})$ **Unabhängigkeitssystem** auf E, falls es die folgenden Axiome erfüllt:

(I.1) $\emptyset \in \mathcal{I}$

(I.2) $I \subseteq J \in \mathcal{I} \Longrightarrow I \in \mathcal{I}$.

Manchmal sagen wir auch einfach, daß $\mathcal{I}$ ein Unabhängigkeitssystem ist. Die Elemente von $\mathcal{I}$ heißen **unabhängige Mengen**, die Mengen in $2^E \setminus \mathcal{I}$ sind **abhängige Mengen**. $(E, \mathcal{I} \setminus \{\emptyset\})$ ist also ein einfacher Hypergraph. Beispiele graphentheoretisch interessanter Unabhängigkeitssysteme sind etwa die Menge aller Matchings $\{M \subseteq E \mid M \text{ Matching }\}$ oder die Menge aller stabilen Knotenmengen $\{S \subseteq V \mid S \text{ stabil }\}$ eines Graphen $G = (V, E)$ oder die Menge aller Branchings $\{B \subseteq A \mid (V, B) \text{ Branching }\}$ eines Digraphen $D = (V, A)$.

Ein Unabhängigkeitssystem $(E, \mathcal{I})$ heißt **Matroid**, falls gilt

(I.3) $I, J \in \mathcal{I}, |I| = |J| + 1 \Longrightarrow \exists e \in I \setminus J \text{ mit } J \cup \{e\} \in \mathcal{I}$.

Ist $(E, \mathcal{I})$ ein Unabhängigkeitssystem und $F \subseteq E$, dann heißt jede Menge $B \subseteq F$ mit $B \in \mathcal{I}$, die in keiner weiteren Menge mit diesen beiden Eigenschaften enthalten ist, **Basis** von F. Die Zahl

$$r(F) := \max\{|B| \mid B \text{ Basis von } F\}$$

ist der **Rang** von F bezüglich $(E, \mathcal{I})$. Für Matroide $(E, \mathcal{I})$ folgt aus (I.3), daß für jede Menge $F \subseteq E$ alle Basen von F die gleiche Kardinalität besitzen.

Eine abhängige Menge $C \subseteq E$, die die Eigenschaft hat, daß $C \setminus \{e\}$ unabhängig ist für alle $e \in C$, heißt **Zirkuit** (oder Kreis) des Unabhängigkeitssytems $(E, \mathcal{I})$.

Ein klassisches Beispiel für Matroide ist das folgende. Sei $G = (V, E)$ ein Graph. Dann ist $\mathcal{I} := \{F \subseteq E \mid (V, F) \text{ Wald}\}$ das Unabhängigkeitssystem eines Matroids auf E. Dieses Matroid wird das **graphische Matroid** bezüglich G genannt. Die Zirkuits von $(E, \mathcal{I})$ sind die Kreise von G, die Basen einer Kantenmenge F sind die Vereingungen von aufspannenden Bäumen der Komponenten von $(V(F), F)$.

Ist E eine Menge und $\mathcal{I} \subseteq 2^E$ ein Mengensystem, so nennen wir eine Menge $M \in \mathcal{I}$ **maximal** (bzw. **minimal**) bezüglich $\mathcal{I}$, wenn es keine Menge $M' \in \mathcal{I}$ gibt mit $M \subset M'$ (bzw. $M' \subset M$). $M \in \mathcal{I}$ heißt ein **größtes** (bzw. **kleinstes**) Element von $\mathcal{I}$, wenn $|M'| \leq |M|$ (bzw. $|M'| \geq |M|$) gilt für alle $M' \in \mathcal{I}$. So sind z. B. bezüglich eines Unabhängigkeitssystems $(E, \mathcal{I})$ und einer Menge $F \subseteq E$ die Basen von F genau die maximalen Elemente von $\mathcal{I}_F := \{I \in \mathcal{I} \mid I \subseteq F\}$. Nicht jede Basis ist auch ein größtes Element von $\mathcal{I}_F$. (Z. B. sind nicht alle maximalen Matchings (maximalen stabilen Knotenmengen) in einem Graphen größte Matchings (größte stabile Mengen).) Ist $(E, \mathcal{I})$ jedoch ein Matroid, so sind alle maximalen Elemente von $\mathcal{I}_F$ auch größte Elemente von $\mathcal{I}_F$. (In einem zusammenhängenden Graphen sind z. B. alle maximalen Wälder aufspannende Bäume.)

Gute Einführungen in die Matroidtheorie sind in den Büchern Oxley (1994), Truemper (1992) und Welsh (1976) zu finden.

Liste einiger in der Graphentheorie verwendeter Symbole

G	bezeichnet meistens einen Graphen
D	bezeichnet meistens einen Digraphen
n	Anzahl der Knoten eines Graphen (wenn möglich)
m	Anzahl der Kanten eines Graphen (wenn möglich)
$\alpha(G)$	Stabilitätszahl = maximale Kardinalität einer stabilen Menge in G
$\omega(G)$	Cliquenzahl = maximale Kardinalität einer Clique in G
$\tau(G)$	Knotenüberdeckungszahl = minimale Kardinalität einer Knotenüberdeckung von G
$\rho(G)$	Kantenüberdeckungszahl = minimale Kardinalität einer Kantenüberdeckung von G
$\nu(G)$	Matchingzahl = maximale Kardinalität eines Matchings in G
$\chi(G)$	Färbungszahl = chromatische Zahl = minimale Anzahl von stabilen Mengen in einer Knotenfärbung von G
$\overline{\chi}(G)$	Cliquenüberdeckungszahl = minimale Anzahl von Cliquen in einer Cliquenüberdeckungs von G
$\gamma(G)$	Kantenfärbungszahl = chromatischer Index = minimale Anzahl von Matchings in einer Kantenfärbung von G
$\Delta(G)$	maximaler Grad eines Knotens von G
$\delta(G)$	minimaler Grad eines Knotens von G
$\delta(W)$	der durch die Knotenmenge W induzierte Schnitt
$\delta^+(W)$	Menge aller Bögen mit Anfangsknoten in W und Endknoten in $V \setminus W$
$\delta^-(W)$	Menge aller Bögen mit Endknoten in W und Anfangsknoten in $V \setminus W$
$\kappa(G)$	Zusammenhangszahl = maximales κ, so daß G κ-zusammenhängend ist.
$\vec{\kappa}(D)$	starke Zusammenhangszahl = maximales κ, so daß G stark κ-zusammenhängend ist.
$\lambda(G)$	Kantenzusammenhangszahl = maximales κ, so daß G κ-kantenzusammenhängend ist.
$\vec{\lambda}(G)$	starke Bogenzusammenhangszahl = maximales κ, so daß D stark κ-bogenzusammenhängend ist.
$\Gamma(W)$	Menge der Nachbarn der Knotenmenge W
$G[W]$	der von der Knotenmenge W induzierte Untergraph von G

$G - W$	der aus G durch Entfernung der Knotenmenge W entstehende Graph
$G - F$	der aus G durch Entfernung der Kantenmenge F entstehende Graph
$G \cdot W$	der aus G durch Kontaktion der Knotenmenge W entstehende Graph
$G \cdot FW$	der aus G durch Kontaktion der Kantenmenge F entstehende Graph
$\deg(v)$	Grad des Knoten v
$E(W)$	Menge aller Kanten von G mit beiden Endknoten in W
$A(W)$	Menge alle Bögen von D mit Anfangs- und Endknoten in W
$V(F)$	Menge aller Knoten aus G (bzw. D) die Endknoten (bzw. Anfangs- oder Endknoten) mit mindestens einer Kante (bzw. eines Bogens) aus F sind
K_n	vollständiger Graph mit n Knoten
$K_{m,n}$	vollständig bipartiter Graph mit $\|V_1\| = \|V_2\| = n$.
$L(G)$	Kantengraph von G
$\overline{G}$	Komplement von G

2.3 Einige Beispiele

In diesem Abschnitt wollen wir mehrere Beispiele von kombinatorischen Optimierungsproblemen und einige ihrer Anwendungen auflisten. Diese Sammlung ist nicht im geringsten vollständig, sondern umfaßt nur einige in der Literatur häufig diskutierte oder besonders anwendungsnahe Probleme. Wir benutzen dabei gelegentlich englische Namen, die mittlerweile auch im Deutschen zu Standardbezeichnungen geworden sind. Fast alle der nachfolgend aufgeführten "Probleme" bestehen aus mehreren eng miteinander verwandten Problemtypen. Wir gehen bei unserer Auflistung so vor, daß wir meistens zunächst die graphentheoretische Formulierung geben und dann einige Anwendungen skizzieren.

(2.8) Kürzeste Wege. Gegeben seien ein Digraph $D = (V, A)$ und zwei verschiedene Knoten $u, v \in V$, stelle fest, ob es einen gerichteten Weg von u nach v gibt. Falls das so ist, und falls "Entfernungen" $c_{ij} \geq 0$ für alle $(i, j) \in A$ bekannt sind, bestimme einen kürzesten gerichteten Weg von u nach v (d. h. einen (u, v)-Weg P, so daß $c(P)$ minimal ist). Dieses Problem wird üblicherweise **Problem des kürzesten Weges** (shortest path problem) genannt. Zwei interessante Varianten sind die folgenden: Finde einen kürzesten (u, v)-Weg gerader bzw. ungerader Länge (d. h. mit gerader bzw. ungerader Bogenzahl).

Das Problem des kürzesten Weges gibt es auch in einer ungerichteten Version. Hier sucht man in einem Graphen $G = (V, E)$ mit Entfernungen $c_e \geq 0$ für alle $e \in E$ bei gegebenen Knoten $u, v \in V$ einen kürzesten $[u, v]$-Weg. Analog kann man nach einem kürzesten Weg gerader oder ungerader Länge fragen.

Natürlich kann man in allen bisher angesprochenen Problemen, das Wort "kürzester" durch "längster" ersetzen und erhält dadurch **Probleme der längsten Wege** verschiedener Arten. Hätten wir beim Problem des kürzesten Weges nicht die Beschränkung $c_{ij} \geq 0$ für die Zielfunktionskoeffizienten, wären die beiden Problemtypen offensichtlich äquivalent. Aber so sind sie es nicht! Ein Spezialfall (Zielfunktion $c_e = 1$ für alle $e \in E$) des Problems des längsten Weges ist das Problem zu entscheiden, ob ein Graph einen hamiltonschen Weg von u nach v enthält.

$\square$

Anwendungen dieses Problems und seiner Varianten sind offensichtlich. Eine Anwendung aus der Wirtschafts- und Sozialgeographie, die nicht unbedingt im Gesichtsfeld von Mathematikern liegt, sei hier kurz erwähnt. Bei Fragen der Raumordnung und Landesplanung werden sehr umfangreiche Erreichbarkeitsanalysen angestellt, um Einzugsbereiche (bzgl. Straßen-, Nahverkehrs- und Bundesbahnanbindung) festzustellen. Auf diese Weise werden Mittel- und Oberzentren des ländlichen Raumes ermittelt und Versorgungsgrade der Bevölkerung in Bezug auf Ärzte, Krankenhäuser, Schulen etc. bestimmt. Ebenso erfolgen Untersuchungen bezüglich des Arbeitsplatzangebots. Alle diese Analysen basieren auf einer genauen Ermittlung der Straßen-, Bus- und Bahnentfernungen (in Kilometern oder Zeiteinheiten) und Algorithmen zur Bestimmung kürzester Wege in den "Verbindungsnetzwerken". Einen Eindruck von derartigen Analysen vermittelt

> "ROLAND-Report 3: Erreichbarkeitsanalysen für Raumordnung und Landesplanung" verfaßt von R. Derenbach, K. Horn & U. Schaaf und herausgegeben vom Bayerischen Staatsministerium für Landesentwicklung und Umweltfragen, der Bundesforschungsanstalt für Landeskunde und Raumordnung und der DATUM e. V., Bonn-Bad Godesberg, 1974, 6105/675.

(2.9) Das Zuordnungsproblem (assignment problem)**.** Gegeben sei ein bipartiter Graph $G = (V, E)$ mit Kantengewichten $c_e \in \mathbb{K}$ für alle $e \in E$, gesucht ist ein Matching in G maximalen Gewichts. Man nennt dieses Problem das **Matchingproblem in bipartiten Graphen** oder kurz **bipartites Matchingproblem**. Haben die beiden Knotenmengen in der Bipartition von V gleiche Kardinalität und sucht man ein perfektes Matching minimalen Gewichts, so spricht man von einem **Zuordnungsproblem**. Es gibt noch eine weitere Formulierung des Zuordnungsproblems. Gegeben sei ein Digraph $D = (V, A)$, der auch Schlingen haben darf, mit Bogengewichten (meistens wird unterstellt, daß D vollständig ist und Schlingen hat), gesucht ist eine Bogenmenge minimalen Gewichts, so daß jeder Knoten von D genau einmal Anfangs- und genau einmal Endknoten eines Bogens aus B ist. (B ist also eine Menge knotendisjunkter gerichteter Kreise, so daß jeder Knoten auf genau einem Kreis liegt.) Wir wollen dieses Problem **gerichtetes Zuordnungsproblem** nennen.

$\square$

Das Zuordnungsproblem hat folgende "Anwendung". Gegeben seien n Männer und n Frauen, für $1 \leq i, j \leq n$ sei c_{ij} ein "Antipathiekoeffizient". Gesucht ist eine Zuordnung von Männern zu Frauen (Heirat), so daß die Summe der Antipathiekoeffizienten minimal ist. Dieses Problem wird häufig **Heiratsproblem** genannt.

Das Matchingproblem in bipartiten Graphen kann man folgendermaßen interpretieren. Ein Betrieb habe m offene Stellen und n Bewerber für diese Position. Durch Tests hat man herausgefunden, welche Eignung Bewerber i für die Stelle j hat. Diese "Kompetenz" sei mit c_{ij} bezeichnet. Gesucht wird eine Zuordnung von Bewerbern zu Positionen, so daß die

“Gesamtkompetenz” maximal wird.

Das Zuordnungsproblem und das Matchingproblem in bipartiten Graphen sind offenbar sehr ähnlich, die Beziehungen zwischen dem Zuordnungsproblem und seiner gerichteten Version sind dagegen nicht ganz so offensichtlich. Dennoch sind diese drei Probleme in folgendem Sinne “äquivalent”: man kann sie auf sehr einfache Weise ineinander transformieren, d. h. mit einem schnellen Algorithmus zur Lösung des einen Problems kann man die beiden anderen Probleme lösen, ohne komplizierte Transformationsalgorithmen einzuschalten.

Transformationstechniken, die einen Problemtyp in einen anderen überführen, sind außerordentlich wichtig und zwar sowohl aus theoretischer als auch aus praktischer Sicht. In der Theorie werden sie dazu benutzt, Probleme nach ihrem Schwierigkeitsgrad zu klassifizieren (siehe Kapitel 3), in der Praxis ermöglichen sie die Benutzung eines einzigen Algorithmus zur Lösung der verschiedensten Probleme und ersparen daher erhebliche Codierungs- und Testkosten. Anhand der drei vorgenannten Probleme wollen wir nun derartige Transformationstechniken demonstrieren.

Bipartites Matchingsproblem $\longrightarrow$ Zuordnungsproblem. Angenommen wir haben ein Matchingproblem in einem bipartiten Graphen und wollen es mit einem Algorithmus für Zuordnungsprobleme lösen. Das Matchingproblem ist gegeben durch einen bipartiten Graphen $G = (V, E)$ mit Bipartition V_1, V_2 und Kantengewichten $c_e \in \mathbb{K}$ für alle $e \in E$. O. B. d. A. können wir annehmen, daß $m = |V_1| \leq |V_2| = n$ gilt. Zur Menge V_1 fügen wir $n - m$ neue Knoten W (künstliche Knoten) hinzu. Wir setzen $V_1' := V_1 \cup W$. Für je zwei Knoten $i \in V_1'$ und $j \in V_2$, die nicht in G benachbart sind, fügen wir eine neue (künstliche) Kante ij hinzu. Die Menge der so hinzugefügten Kanten nennen wir E', und den Graphen $(V_1' \cup V_2, E \cup E')$ bezeichnen wir mit G'. G' ist der vollständige bipartite Graph $K_{n,n}$. Wir definieren neue Kantengewichte c_e' wie folgt:

$$c_e' := \begin{cases} 0 & \text{falls } e \in E' \\ 0 & \text{falls } e \in E \text{ und } c_e \leq 0 \\ -c_e & \text{falls } e \in E \text{ und } c_e > 0 \end{cases}$$

Lösen wir das Zuordnungsproblem bezüglich G' mit den Gewichten c_e', $e \in E \cup E'$, so erhalten wir ein perfektes Matching M' minimalen Gewichts bezüglich c'. Es ist nun einfach zu sehen, daß

$$M := \{e \in M' \mid c_e' < 0\}$$

ein Matching in G ist, das maximal bezüglich der Gewichtsfunktion c ist.

Zuordnungsproblem $\longrightarrow$ gerichtetes Zuordnungsproblem. Wir zeigen nun, daß man das Zuordnungsproblem mit einem Algorithmus für das gerichtete Zuordnungsproblem lösen kann. Gegeben sei also ein bipartiter Graph $G = (V, E)$ mit Bipartition V_1, V_2 und Kantengewichten c_e. Es gelte $V_1 = \{u_1, u_2, \ldots, u_n\}$, $V_2 = \{v_1, v_2, \ldots, v_n\}$. Wir definieren einen Digraphen $D = (W, A)$ mit $W = \{w_1, \ldots, w_n\}$. Zwei Knoten w_i, $w_j \in W$ sind genau dann durch einen Bogen (w_i, w_j) verbunden, wenn $u_i v_j \in E$ gilt. Das Gewicht $c'((w_i, w_j))$

des Bogens (w_i, w_j) sei das Gewicht $c(u_i v_j)$ der Kante $u_i v_j$. Ist B eine minimale Lösung des gerichteten Zuordnungsproblems bezüglich D und c', so ist

$$M := \{u_i v_j \in E \mid (w_i, w_j) \in B\}$$

offenbar ein minimales perfektes Matching in G bezüglich der Gewichtsfunktion c. Es ist ebenfalls sofort klar, daß das gerichtete Zuordnungsproblem bezüglich D eine Lösung genau dann hat, wenn G ein perfektes Matching enthält.

Gerichtetes Zuordnungsproblem $\longrightarrow$ bipartites Matchingproblem. Schließlich wollen wir noch vorführen, daß man das **gerichtete Zuordnungsproblem auf das Matchingproblem** in bipartiten Graphen zurückführen kann. Gegeben sei also ein Digraph $D = (W, A)$ mit $W = \{w_1, \ldots, w_n\}$ und Bogengewichten $c((w_i, w_j))$ für alle $(w_i, w_j) \in A$. Wir definieren einen bipartiten Graphen $G = (V, E)$ mit Bipartition $V_1 = \{u_1, \ldots, u_n\}$, $V_2 = \{v_1, \ldots, v_n\}$ und Kantenmenge $E := \{u_i v_j \mid (w_i, w_j) \in A\}$. Es seien

$$z := n(\max\{|c((w_i, w_j))| : (w_i, w_j) \in A\}) + 1$$

und

$$c'(u_i v_j) := -c((w_i, w_j)) + z.$$

Nach Konstruktion gilt, daß jedes Matching in G mit k Kanten ein geringeres Gewicht hat als ein Matching mit $k+1$ Kanten, $k = 0, \ldots, n-1$. Daraus folgt, daß es eine Lösung des gerichteten Zuordnungproblems bezüglich D genau dann gibt, wenn jedes maximale Matching M bezüglich G und c' perfekt ist. Ist dies so, dann ist

$$B := \{(w_i, w_j) \in A \mid u_i v_j \in M\}$$

eine minimale Lösung des gerichteten Zuordnungsproblems mit Gewicht $c(B) = -c'(M) + nz$.

(2.10) Das Matchingproblem. Die Grundversion dieses Problems ist die folgende. Gegeben sei ein Graph $G = (V, E)$ mit Kantengewichten c_e für alle $e \in E$. Ist ein Matching M von G maximalen Gewichts $c(M)$ gesucht, so heißt dieses Problem **Matchingproblem**. Sucht man ein perfektes Matching minimalen Gewichts, so wird es **perfektes Matchingproblem** genannt.

Diese Probleme können wie folgt verallgemeinert werden. Gegeben seien zusätzlich nichtnegative ganze Zahlen b_v für alle $v \in V$ (genannt Gradbeschränkungen) und u_e für alle $e \in E$ (genannt Kantenkapazitäten). Ein (perfektes) **b-Matching** ist eine Zuordnung x_e von nichtnegativen ganzen Zahlen zu den Kanten $e \in E$, so daß für jeden Knoten $v \in V$ die Summe der Zahlen x_e über die Kanten $e \in E$, die mit v inzidieren, höchstens (exakt) b_v ist. Das **unkapazitierte (perfekte) b-Matchingproblem** ist die Aufgabe ein (perfektes) b-Matching (x_e) $e \in E$ zu finden, so daß $\sum_{e \in E} c_e x_e$ maximal (minimal) ist.

Sollen die ganzzahligen Kantenwerte x_e für alle $e \in E$ zusätzlich noch die Kapazitätsschranken $0 \leq x_e \leq u_e$ erfüllen, so spricht man von einem **(perfekten) u-kapazitierten b-Matchingproblem**.

$\square$

Wir geben nun eine Formulierung des u-kapazitierten b-Matchingproblems als ganzzahliges lineares Programm. (Im Vektor $x = (x_e)_{e \in E} \in \mathbb{K}^E$ fassen wir die Werte zusammen, die den Kanten von G zugeordnet werden.) Es sei

$$
(2.11) \qquad
\begin{array}{rrll}
 & & \max c^T x & \\
\text{(i)} & x(\delta(v)) = & \sum\limits_{e \in \delta(v)} x_e \leq b_v & \text{für alle } v \in V \\
\text{(ii)} & & 0 \leq x_e \leq u_e & \text{für alle } e \in E \\
\text{(iii)} & & x_e \in \mathbf{Z} & \text{für alle } e \in E.
\end{array}
$$

Die Ungleichungssysteme (2.11) (i) nennt man aus naheliegenden Gründen **Gradbeschränkungen**. Es ist einfach einzusehen (und sei dem Leser überlassen), daß die zulässigen Lösungen von (2.11) genau den u-kapazitierten b-Matchings von G entsprechen. Ein ganzzahliges lineares Programm, das das unkapazitierte b-Matchingproblem modelliert, erhält man dadurch, daß man in (2.11) (ii) die oberen Schranken u_e weglässt (bzw. $u_e = \infty$ setzt). Formulierungen der "perfekten" Versionen dieser Probleme als ganzzahlige lineare Programme ergeben sich durch Ersetzen von max durch min in der Zielfunktion und ersetzen des "$\leq$" Zeichens in (2.11) (i) durch "$=$".

Setzt man $u_e = 1$ (oder $u_e \geq 1$ beliebig) für alle $e \in E$ und $b_v = 1$ für alle $v \in V$ (also $b = \mathbf{1}$), so sind die zulässigen Lösungen von (2.11) genau die Inzidenzvektoren der Matchings von G. Das unkapazitierte (perfekte) $\mathbf{1}$-Matchingproblem ist also nichts anderes als das (perfekte) Matchingproblem.

An dieser Stelle wollen wir noch eine — nicht offensichtliche — Problemtransformation vorführen. Und zwar wollen wir zeigen, daß die Aufgabe, in einem ungerichteten Graphen $G = (V, E)$ mit Kantengewichten $c_e \geq 0$ für alle $e \in E$ einen kürzesten Weg ungerader Länge zwischen zwei Knoten $u, v \in V$ zu bestimmen, mit einem Algorithmus für das perfekte Matchingproblem gelöst werden kann. Und zwar konstruieren wir aus G einen neuen Graphen G' wie folgt. Nehmen wir an, daß $V = \{v_1, \ldots, v_n\}$ gilt. Die Graphen $G_1 = (U, E_1)$ mit $U := \{u_1, \ldots, u_n\}$ und $G_2 = (W, E_2)$ mit $W := \{w_1, \ldots, w_n\}$ seien knotendisjunkte isomorphe Bilder (also Kopien) von G, so daß die Abbildungen $v_i \mapsto u_i$ und $v_i \mapsto w_i$, $i = 1, \ldots, n$ Isomorphismen sind. Aus G_2 entfernen wir die Bilder der Knoten u und v, dann verbinden wir die übrigen Knoten $w_i \in W$ mit ihren isomorphen Bildern $u_i \in U$ durch eine Kante $u_i w_i$. Diese neuen Kanten $u_i w_i$ erhalten das Gewicht $c(u_i w_i) = 0$. Die Kanten aus G_1 und $G_2 - \{u, v\}$, die ja Bilder von Kanten aus G sind, erhalten das Gewicht ihrer Urbildkanten. Der Graph G' entsteht also aus der Vereinigung von G_1 mit $G_2 - \{u, v\}$ unter Hinzufügung der Kanten $u_i w_i$, siehe Abbildung 2.1. Man überlegt sich leicht, daß jedes perfekte Matching in G' einem ungeraden $[u, v]$-Weg in G entspricht und daß jedes minimale perfekte Matching in G' einen minimalen ungeraden $[u, v]$-Weg bestimmt.

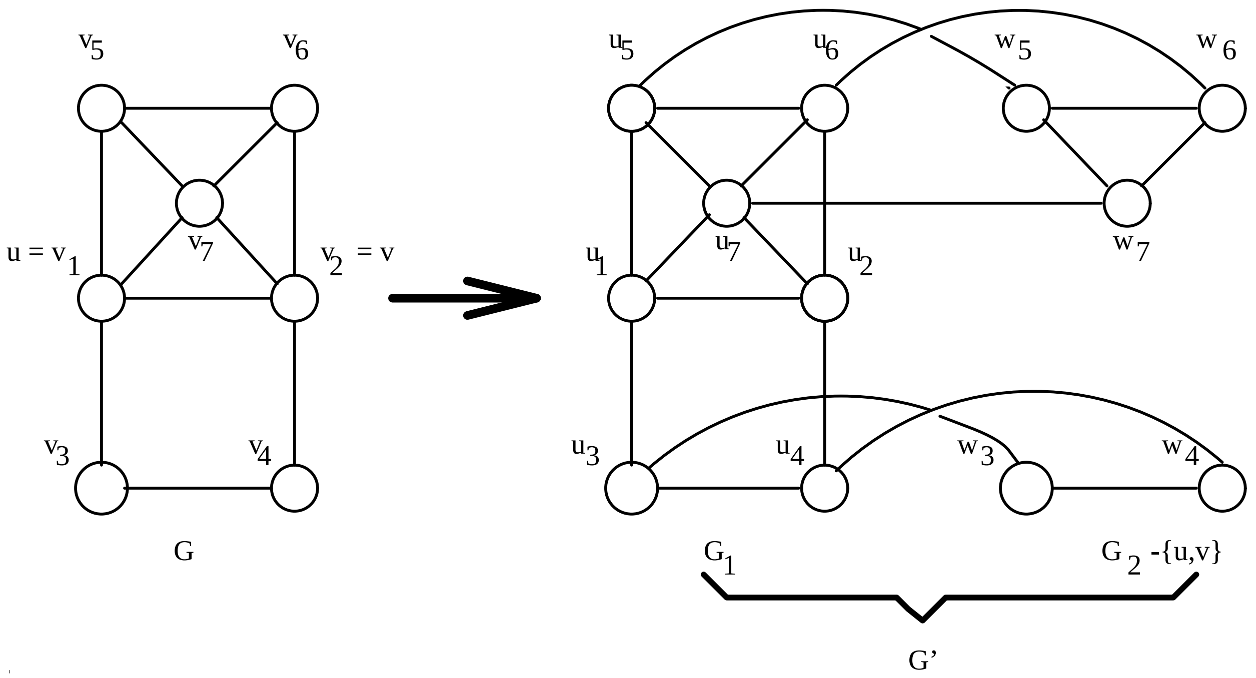

Abb. 2.1

In Abbildung 2.1 entspricht z. B. dem ungeraden $[u, v]$-Weg (u, v_7, v_6, v) das perfekte Matching $M = \{u_1 u_7, w_7 w_6, u_6 u_2, u_3 w_3, u_4 w_4, u_5 w_5\}$ und umgekehrt.

Hausaufgabe. Finden Sie eine ähnliche Konstruktion, die das Problem, einen kürzesten $[u, v]$-Weg gerader Länge zu bestimmen, auf ein perfektes Matchingproblem zurückführt!

(2.12) Wälder, Bäume, Branchings, Arboreszenzen. Gegeben sei ein Graph $G = (V, E)$ mit Kantengewichten $c_e \in \mathbb{K}$ für alle $e \in E$. Die Aufgabe, einen Wald $W \subseteq E$ zu finden, so daß $c(W)$ maximal ist, heißt **Problem des maximalen Waldes**.

Die Aufgabe einen Baum $T \subseteq E$ zu finden, der G aufspannt und dessen Gewicht $c(T)$ minimal ist, heißt **Problem des minimalen aufspannenden Baumes** (minimum spanning tree problem). Diese beiden Probleme haben auch eine gerichtete Version.

Gegeben sei ein Digraph $D = (V, A)$ mit Bogengewichten $c_a \in \mathbb{K}$ für alle $a \in A$. Die Aufgabe, ein Branching $B \subseteq A$ maximalen Gewichts zu finden, heißt **maximales Branching-Problem**, die Aufgabe, eine Arboreszenz (mit vorgegebener Wurzel r) von D minimalen Gewichts zu finden heißt **minimales Arboreszenz-Problem** (r-**Arboreszenz-Problem**).

□

Die im folgenden Punkt zusammengefaßten Probleme gehören zu den am meisten untersuchten und anwendungsreichsten Problemen.

(2.13) Routenplanung. Gegeben seien n Städte und Entfernungen c_{ij} zwischen diesen, gesucht ist eine Rundreise (Tour), die durch alle Städte genau einmal führt und minimale Länge hat. Haben die Entfernungen die Eigenschaft, daß $c_{ij} = c_{ji}$ gilt, $1 \leq i < j \leq n$, so nennt man dieses Problem **symmetrisches Travelling-Salesman-Problem (TSP)**, andernfalls heißt es **asymmetrisches TSP**. Graphentheoretisch läßt sich das TSP wie folgt formulieren. Gegeben sei ein vollständiger Graph (oder Digraph) G mit Kantengewichten (oder Bogengewichten), gesucht ist ein (gerichteter) hamiltonscher Kreis minimaler Länge. Beim TSP geht man durch jeden Knoten genau einmal, beim (gerichteten) **Chinesischen Postbotenproblem** (Chinese postman problem) durch jede Kante (jeden Bogen) mindestens einmal, d. h. in einem Graphen (Digraphen) mit Kantengewichten (Bogengewichten) wird eine Kette (gerichtete Kette) gesucht, die jede Kante (jeden Bogen) mindestens einmal enthält und minimale Länge hat.

Zu diesen beiden Standardproblemen gibt es hunderte von Mischungen und Varianten. Z. B., man sucht eine Kette, die durch einige vorgegebene Knoten und Kanten mindestens einmal geht und minimale Länge hat; man legt verschiedene Ausgangspunkte (oder Depots) fest, zu denen man nach einer gewissen Streckenlänge wieder zurückkehren muß, etc. Eine relativ allgemeine Formulierung ist die folgende. Gegeben ist ein gemischter Graph mit Knotenmenge V, Kantenmenge E und Bogenmenge A. Ferner sind eine Menge von Depots $W \subseteq V$, von denen aus Reisen gestartet werden müssen, eine Menge $U \subseteq V$ von Knoten, die mindestens einmal besucht werden müssen, und eine Menge $B \subseteq E \cup A$ von Kanten und Bögen, die mindestens einmal durchlaufen werden müssen. Gesucht sind geschlossene Ketten von Kanten und gleichgerichteten Bögen, so daß jede dieser Folgen mindestens (oder genau) einen der Knoten aus W enthält und die Vereinigung dieser Ketten jeden Knoten aus U und jede Kante (Bogen) aus B mindestens einmal enthält und minimale Länge hat.

□

Anwendungen dieser Probleme in der Routenplanung von Lieferwagen, von Straßenkehrmaschinen, der Müllabfuhr, von Speditionen etc. sind offensichtlich. Aber auch bei der Steuerung von NC-Maschinen (zum automatischen Bohren, Löten oder Schweißen) oder der Verdrahtung von Leiterplatten (z. B. von Testbussen) tritt das TSP (oder eine seiner Varianten) auf. Abbildung 2.2 zeigt eine Leiterplatte, durch die 441 Löcher gebohrt werden müssen. Links unten ist der Startpunkt, an den der Bohrkopf nach Beendigung des Arbeitsvorganges zurückkehrt, damit eine neue Platte in die Maschine eingelegt werden kann. Abbildung 2.2 zeigt eine optimale Lösung dieses 442-Städte-TSP. Die Bohrmaschine muß eine Weglänge von 50.069 Einheiten zurückzulegen.

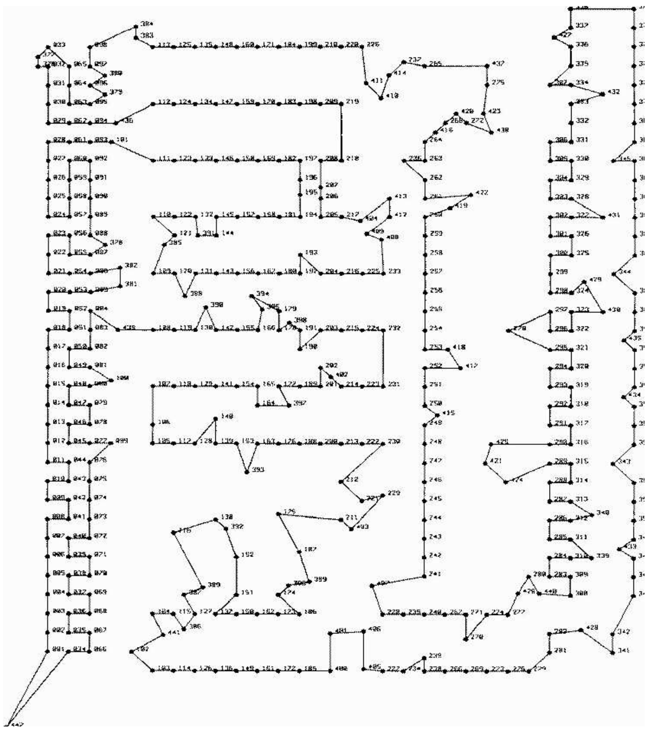

Abb. 2.2

Abbildung 2.3 zeigt 666 Städte auf der Weltkugel. Wählt man die "Luftliniendistanz" (bezüglich eines Großkreises auf der Kugel) als Entfernung zwischen zwei Städten, so zeigt Abbildung 2. 3 eine kürzeste Rundreise durch die 666 Orte dieser Welt. Die Länge dieser Reise ist 294 358 km lang. Abbildungen 2. 2 und 2. 3 sind Grötschel und Holland (1991) entnommen.

In Abbildung 2.4 sind die eingezeichneten Punkte Standorte von Telefonzellen in der holländischen Stadt Haarlem. Der Stern in der Mitte ist das Postamt. Die Aufgabe ist hier, eine Routenplanung für den sich wöchentlich wiederholenden Telefonzellenwartungsdienst zu machen. Die einzelnen Touren starten und enden am Postamt (diese Verbindungen sind nicht eingezeichnet) und führen dann so zu einer Anzahl von Telefonzellen, daß die Wartung aller Telefonzellen auf der Tour innerhalb einer Schicht durchgeführt werden kann.

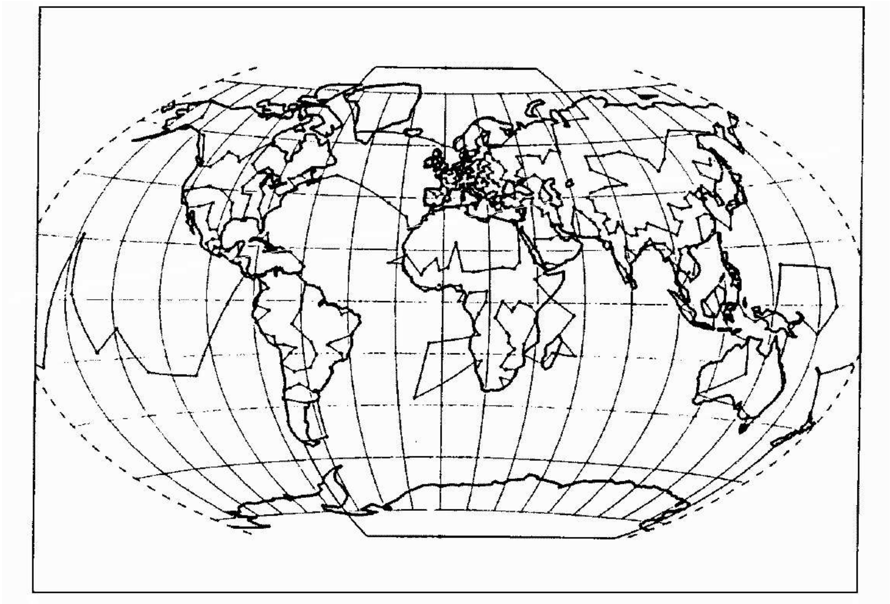

Abb. 2.3

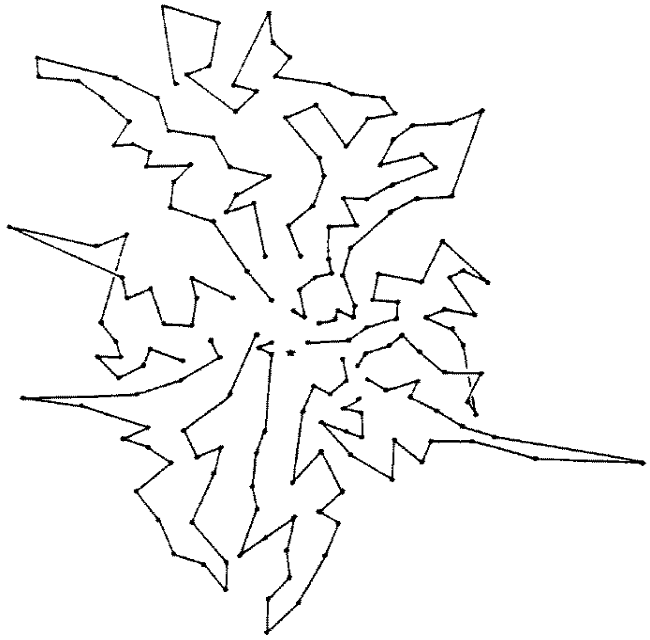

Abb. 2.4

Als Travelling-Salesman-Problem lassen sich auch die folgenden Anwendungsprobleme formulieren:

- Bestimmung einer optimalen Durchlaufreihenfolge der Flüssigkeiten (Chargen) in einer Mehrproduktenpipeline,
- Bestimmung der optimalen Verarbeitungsfolge von Lacken in einer Großlackiererei,

bei diesen beiden Problemen sollen Reinigungszeiten minimiert werden,

– Bestimmung einer Reihenfolge des Walzens von Profilen in einem Walzwerk, so daß die Umrüstzeiten der Walzstraße minimiert werden,
– Bestimmung der zeitlichen Reihenfolge von archäologischen Fundstätten (Grablegungsreihenfolge von Gräbern in einem Gräberfeld, Besiedlungsreihenfolge von Orten) aufgrund von Ähnlichkeitsmaßen (Distanzen), die durch die aufgefundenen Fundstücke definiert werden.

Umfangreiche Information über das TSP, seine Varianten und Anwendungen kann man in dem Sammelband Lawler, Lenstra, Rinnooy Kan und Shmoys (1985) finden.

(2.14) Stabile Mengen, Cliquen, Knotenüberdeckungen. Gegeben sei ein Graph $G = (V, E)$ mit Knotengewichten $c_v \in \mathbb{K}$ für alle $v \in V$. Das **Stabile-Mengen-Problem** ist die Aufgabe, eine stabile Menge $S \subseteq V$ zu suchen, so daß $c(S)$ maximal ist, das **Cliquenproblem** die Aufgabe, eine Clique $Q \subseteq V$ zu suchen, so daß $c(Q)$ maximal ist, und das **Knotenüberdeckungsproblem** die Aufgabe, eine Knotenüberdeckung $K \subseteq V$ zu suchen, so daß $c(K)$ minimal ist.

$\square$

Die drei oben aufgeführten Probleme sind auf triviale Weise ineinander überführbar. Ist nämlich $S \subseteq V$ eine stabile Menge in G, so ist S eine Clique im komplementären Graphen $\overline{G}$ von G und umgekehrt. Also ist das Stabile-Menge-Problem für G mit Gewichtsfunktion c nicht anders als das Cliquenproblem für $\overline{G}$ mit derselben Gewichtsfunktion und umgekehrt. Ist ferner $S \subseteq V$ eine stabile Menge in G, so ist $V \setminus S$ eine Knotenüberdeckung von G. Daraus folgt, daß zu jeder gewichtsmaximalen stabilen Menge S die zugehörige Knotenüberdeckung $V \setminus S$ gewichtsminimal ist und umgekehrt. Das Stabile-Menge-Problem, das Cliquenproblem und das Knotenüberdeckungsproblem sind also drei verschiedene Formulierungen einer Aufgabe. Anwendungen dieser Probleme finden sich z. B. in folgenden Bereichen:

– Einsatzplanung von Flugzeugbesatzungen
– Auslegung von Fließbändern
– Investitionsplanung
– Zuordnung von Wirtschaftsprüfern zu Prüffeldern
– Entwurf von optimalen fehlerkorrigierenden Codes
– Schaltkreisentwurf
– Standortplanung
– Wiedergewinnung von Information aus Datenbanken
– Versuchsplanung
– Signalübertragung.

Aber auch das folgende Schachproblem kann als Stabile-Menge-Problem formuliert werden: Bestimme die maximale Anzahl von Damen (oder Türmen, oder Pferden etc.), die auf einem $n \times n$ Schachbrett so plaziert werden können, daß keine eine andere schlägt.

(2.15) Färbungsprobleme. Gegeben sei ein Graph $G = (V, E)$. Zusätzlich seien Knotengewichte b_v für alle $v \in V$ gegeben. Die Aufgabe, eine Folge von (nicht notwendigerweise verschiedenen) stabilen Mengen $S_1, \ldots, S_t$ von G zu suchen, so daß jeder Knoten in mindestens b_v dieser stabilen Mengen enthalten und t minimal ist, heißt (gewichtetes) **Knotenfärbungsproblem** oder kurz **Färbungsproblem**. Beim (gewichteten) **Kantenfärbungsproblem** sind statt Knotengewichten Kantengewichte c_e, $e \in E$, gegeben und gesucht ist eine Folge von (nicht notwendigerweise verschiedenen) Matchings $M_1, \ldots, M_s$, so daß jede Kante in mindestens c_e dieser Matchings enthalten und s so klein wie möglich ist.

$\square$

Das Färbungsproblem stammt aus der Geographie. Bei der Kartenherstellung will man die Länder einer Landkarte so einfärben, daß je zwei benachbarte Länder verschiedene Farben haben und möglichst wenige Farben benutzt werden. Zur graphentheoretischen Formulierung dieser Landkartenfärbungsaufgabe definiert man den folgenden Graphen G. Jedem Land ordnet man einen Knoten zu. Je zwei Knoten werden durch eine Kante verbunden, falls die zugehörigen Länder eine gemeinsame Grenze haben.

Hat man eine Färbung der Länder, so daß je zwei benachbarte Länder verschieden gefärbt sind, so entspricht jede Gruppe von Ländern gleicher Farbe einer stabilen Menge in G. Hat man umgekehrt eine Zerlegung der Knotenmenge von G in stabile Mengen, so kann man jeweils die Länder, die zu den Knoten einer stabilen Menge gehören mit derselben Farbe belegen und erhält dadurch eine zulässige Landkartenfärbung. Das Landkartenfärbungsproblem ist also das Knotenfärbungsproblem des zugehörigen Graphen mit $b_v = 1$ für alle $v \in V$.

Die Aufgabe, in einer geographischen Region die Sendefrequenzen von Rundfunksendern so zu verteilen, daß sich die Sender gegenseitig nicht stören und alle Rundfunkteilnehmer, die für sie gedachten Programme auch empfangen können, kann man als Färbungsproblem (mit weiteren Nebenbedingungen) formulieren.

(2.16) Schnitt-Probleme. Gegeben sei ein Graph $G = (V, E)$ mit Kantengewichten $c_e \in \mathbb{K}$ für alle $e \in E$. Das Problem, einen Schnitt $\delta(W)$ in G zu finden mit maximalem Gewicht $c(\delta(W))$, heißt **Max-Cut-Problem**. Sind alle Kantengewichte c_e nicht-negativ, so nennt man das Problem, einen Schnitt minimalen Gewichts in G zu finden, **Min-Cut-Problem**.

$\square$

Das Min-Cut-Problem ist in der Theorie der Netzwerkflüsse sehr wichtig (siehe Kapitel 7). Das Max-Cut-Problem hat z. B. eine interessante Anwendung in der Physik, und zwar kann man beim Studium magnetischer Eigenschaften von Spingläsern im Rahmen des Ising Modells die Aufgabe, einen Grundzustand (energieminimale Konfiguration bei 0° K) zu bestimmen, als Max-Cut-Problem formulieren. Ich will diese Anwendung kurz skizzieren. Ein **Spinglas** besteht aus nichtmagnetischem Material, das an einigen Stellen durch magnetische Atome "verunreinigt" ist. Man interessiert sich für die Energie des Systems und die Orientierung der magnetischen Atome (Verunreinigungen) bei 0° K, also für den sogenannten (gefrorenen) Grundzustand des Spinglases. Dieser Grundzustand ist experimentell nicht herstellbar, und die Physiker haben unterschiedliche, sich z. T. widersprechende Theorien über einige Eigenschaften dieses Grundzustandes.

Mathematisch wird dieses Problem wie folgt modelliert. Jeder Verunreinigung i wird ein Vektor $S_i \in \mathbb{R}^3$ zugeordnet, der (bei einem gegebenen Bezugssytem) die Orientierung des Atomes im Raum, d. h. den magnetischen Spin, beschreibt. Zwischen zwei Verunreinigungen i, j besteht eine magnetische Interaktion, die durch

$$H_{ij} = J(r_{ij}) S_i \cdot S_j$$

beschreiben wird, wobei $J(r_{ij})$ eine Funktion ist die vom Abstand r_{ij} der Verunreinigungen abhängt und $S_i \cdot S_j$ das innere Produkt der Vektoren S_i, S_j ist. In der Praxis wird J (bei gewissen physikalischen Modellen) wie folgt bestimmt

$$J(r_{ij}) := \cos(\mathrm{K} r_{ij}) / r_{ij}^3 ,$$

wobei K eine materialabhängige Konstante ist (z. B. $K = 2.4 \times 10^8$). Die gesamte Energie einer Spinkonfiguration ist gegeben durch

$$H = -\sum J(r_{ij}) S_i \cdot S_j + \sum F \cdot S_i ,$$

wobei F ein äußeres magnetisches Feld ist. (Der Einfachheit halber nehmen wir im folgenden an $F = 0$.) Ein Zustand minimaler Energie ist also dadurch charakterisiert, daß $\sum J(r_{ij}) S_i \cdot S_j$ maximal ist.

Das hierdurch gegebene Maximierungsproblem ist mathematisch kaum behandelbar. Von Ising wurde folgende Vereinfachung vorgeschlagen. Statt jeder beliebigen räumlichen Orientierung werden jeder Verunreinigung nur zwei Orientierungen erlaubt: "Nordpol oben" oder "Nordpol unten". Die dreidimensionalen Vektoren S_i werden dann in diesem Modell durch Variable s_i mit Werten in $\{1, -1\}$ ersetzt, wobei unter Physikern Übereinstimmung darüber besteht, daß dieses Ising-Modell das wahre Verhalten gewisser Spingläsern gut widerspiegelt. Das obige Maximierungsproblem lautet dann bezüglich des Ising Modells

$$\max\{\sum J(r_{ij}) s_i s_j \mid s_i \in \{-1, 1\}\} .$$

Der Schritt zum Max-Cut-Problem ist nun leicht. Wir definieren einen Graphen $G = (V, E)$, wobei jeder Knoten aus V eine Verunreinigung repräsentiert, je zwei Knoten i, j sind durch eine Kante verbunden, die das Gewicht $c_{ij} = -J(r_{ij})$ trägt. (Ist r_{ij} groß, so ist nach Definition c_{ij} sehr klein, und üblicherweise werden Kanten mit kleinen Gewichten c_{ij} gar nicht berücksichtigt). Eine Partition von V in V_1 und V_2 entspricht einer Orientierungsfestlegung der Variablen, z. B. $V_1 := \{i \in V \mid i$ repräsentiert eine Verunreinigung mit Nordpol oben $\}$, $V_2 := \{i \in V \mid$ der Nordpol von i ist unten $\}$. Bei gegebenen Orientierungen der Atome (Partition V_1, V_2 von V) ist die Energie des Spinglaszustandes also wie folgt definiert

$$\sum_{i \in V_1, j \in V_2} c_{ij} - \sum_{i,j \in V_1} c_{ij} - \sum_{i,j \in V_2} c_{ij} .$$

Der Zustand minimaler Energie kann also durch Maximierung des obigen Ausdrucks bestimmt werden. Addieren wir zu diesem Ausdruck die Konstante $C := \sum_{i,j \in V} c_{ij}$, so folgt daraus, daß der Grundzustand eines Spinglases durch die Lösung des Max-Cut Problems

$$\max\{\sum_{i \in V_1} \sum_{j \in V_2} c_{ij} \mid V_1, V_2 \text{ Partition von } V\}$$

bestimmt werden kann. Eine genauere Darstellung und die konkrete Berechnung von Grundzuständen von Spingläsern (sowie weitere Literaturhinweise) kann man in Barahona, Grötschel, Jünger, Reinelt (1988) finden. Dieses Paper beschreibt auch eine Anwendung des Max-Cut-Problems im VLSI-Design: Die Lagenzuweisung von Leiterbahnen, so daß Anzahl der Kontaktlöcher minimal ist.

(2.17) Standortprobleme. Probleme dieses Typs tauchen in der englischsprachigen Literatur z. B. unter den Namen Location oder Allocation Problems, Layout Planning, Facilities Allocation, Plant Layout Problems oder Facilities Design auf. Ihre Vielfalt ist (ähnlich wie bei (2.13)) kaum in wenigen Zeilen darstellbar. Ein (relativ allgemeiner) Standardtyp ist der folgende. Gegeben sei ein Graph (oder Digraph), dessen Knoten Städte, Wohnbezirke, Bauplätze, mögliche Fabrikationsstätten etc. repräsentieren, und dessen Kanten Verkehrsverbindungen, Straßen, Kommunikations- oder Transportmöglichkeiten etc. darstellen. Die Kanten besitzen "Gewichte", die z. B. Entfernungen etc. ausdrücken. Wo sollen ein Krankenhaus, ein Flughafen, mehrere Polizei- oder Feuerwehrstationen, Warenhäuser, Anlieferungslager, Fabrikationshallen ... errichtet werden, so daß ein "Optimalitätskriterium" erfüllt ist? Hierbei tauchen häufig Zielfunktionen auf die nicht linear sind. Z. B. soll ein Feuerwehrdepot so stationiert werden, daß die maximale Entfernung vom Depot zu allen Wohnbezirken minimal ist; drei Auslieferungslager sollen so errichtet werden, daß jedes Lager ein Drittel der Kunden bedienen kann und die Summe der Entfernungen der Lager zu ihren Kunden minimal ist bzw. die maximale Entfernung minimal ist.

$\square$

Im Verlauf der Vorlesung werden wir noch einige spezielle Standortprobleme und ihre Anwendungen kennenlernen. Vorab möchte ich kurz von einer sehr wichtigen Anwendung berichten, bei der Standort- und Routenplanungsprobleme in riesigen Größenordnungen auftreten. Es handelt sich um Plazierungs- und Verdrahtungsprobleme im VLSI-Design. Diese Probleme können derzeit nicht mit exakten Optimierungsverfahren gelöst werden. Hier ist ein Gebiet, bei dem der Entwurf schneller und "guter" Heuristiken von großer Bedeutung ist.

Wir gehen davon aus, daß die Logik einer elektrischen Schaltung oder eines Speicherchips entworfen wurde. Man kann sich das so vorstellen, daß dazu Basiseinheiten (sogenannte Zellen) verwendet werden, die miteinander und mit der "Außenwelt" über Kontaktstellen (sogenannte Pins) verbunden werden können. Jede Zelle hat mehrere Pins. Der Ingenieur, der die Schaltungslogik entwirft, entscheidet also, welche Pins mit welchen anderen verbunden werden sollen. Wir gehen davon aus, daß die Logik festgelegt ist und daß nun die Schaltung physikalisch (auf einem Chip oder einer Leiterplatte) realisiert werden soll.

Man kann nicht davon sprechen, daß bei der Realisierung ein einziges Ziel verfolgt werden soll (typischer OR-Anwendungsfall!). Als ein Hauptziel wird i. a. angesehen, den Chip so zu realisieren, daß die Signalverarbeitung in möglichst kurzer Zeit korrekt durchgeführt wird. Es ist schwierig, dieses Ziel mathematisch zu formulieren, da ja sehr viele verschiedene Signale und Signalkombinationen als Input auftreten und je nach der Komplexität der dadurch hervorgerufenen Stromflüsse unterschiedliche Signalverarbeitungszeiten induzieren können. Daher behilft man sich damit zu versuchen, die Länge der Verbindungen zu minimieren und hier speziell der Verbindungen zwischen Hauptkomponenten eines Chips. Auch dieses Ziel ist i. a. praktisch nicht erreichbar. I. a. wird daher als Kompromiß die Hilfsgröße "Chipfläche" minimiert, da man davon ausgehen kann, daß mit einer geringen Fläche auch kurze Verdrahtungslängen und somit kurze Laufzeiten verbunden sind. Ziel der Realisierung ist also — im wesentlichen —, die Schaltung auf einer möglichst kleinen (oder einer vorgegebenen) Fläche unterzubringen.

Jede Zelle hat bestimmte physikalische Ausmaße, i. a. sind Zellen rechteckig und haben eine gewisse Breite und Höhe, und ihre Pinpositionen sind definiert, siehe Abbildung 2.5 (a). Die Aufgabe bei der Realisierung besteht also darin, die Zellen so zu plazieren, daß möglichst wenig Fläche verbraucht wird. Bei dieser groben Zielsetzung ist jedoch noch die Verdrahtungsfrage außer Acht gelassen worden. Die Verdrahtung der Pins geschieht wie folgt. Es werden Mengen von Pins festgelegt, die miteinander verbunden werden sollen. Diese Mengen heißen in der VLSI-Sprache "Netze". Das heißt, es wird ein Hypergraph definiert, bei dem die Pins die Knoten und die Netze die Hyperkanten (Kanten können mehr als 2 Knoten haben) sind. Die Pins mit den Nummern 1, 2, 3, 4, 5 in Abbildung 2.5 (b) bilden ein Netz. Die physikalische Verdrahtung der Pins eines Netzes geschieht dadurch, daß ein Baum bestimmt wird, der alle Pins des Netzes enthält. Eine solche Verdrahtung des Netzes aus Abbildung 2.5 (b) ist in Abbildung 2.5 (c) gezeigt. I. a. wird in sogenannten "Kanälen" (gewisse freigelassene Flächen) verdrahtet, wobei die Verbindungen nur horizontal oder vertikal verlaufen dürfen.

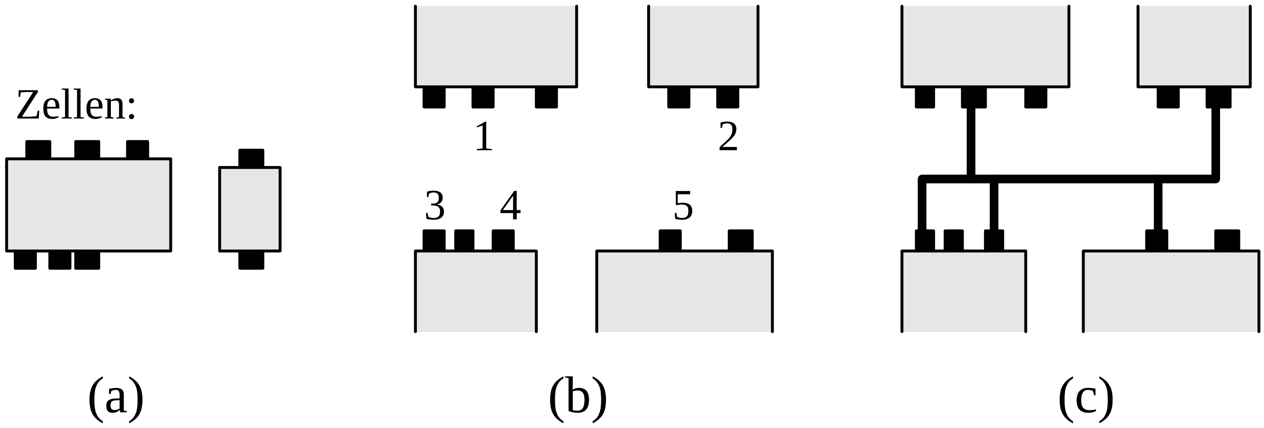

Abb. 2.5

Bei der Plazierung der Zellen muß also so viel Platz zwischen den Zellen gelassen werden, daß alle Netze technisch verdrahtet werden können. (Drähte müssen gewisse Abstände voneinander haben, um Kurzschlüsse zu vermeiden.) Bei gegebener Plazierung besteht also die Aufgabe, eine Verdrahtung zu finden, die mit den freigelassenen Kanälen auskommt. Hierbei sind eine Reihe weiterer Nebenbedingungen zu berücksichtigen. Elektrische Verbindungen dürfen sich nicht schneiden. Sie werden auf den zwei (oder mehr) Seiten (Lagen, layers) des Chips ausgeführt, wobei immer auf einer Lage alle Verbindungen in einer Richtung verlaufen. Bei zweiseitigen Chips verlaufen sie auf einer Seite horizontal auf der anderen vertikal. Die Verbindungen zwischen zwei Seiten werden durch Bohrungen (Durchkontaktierungen) vorgenommen, wobei ein Layout eines Chips so zu entwerfen ist, daß möglichst wenig Durchkontaktierungen vorgenommen werden müssen. Dies sind bei weitem noch nicht alle Wünsche, Ziele und Bedingungen, die bei der technischen Realisierung berücksichtigt werden müssen. Aber ich glaube, es ist bereits jetzt jedem klar, daß das Chip-Design eine reichlich komplexe Aufgabe ist — insbesondere wenn man bedenkt, daß durchaus einige hunderttausend Pins und Verbindungen vorkommen können.

Der Entwurf des Layouts derartiger Schaltungen geschieht heute in der Praxis in mehreren Stufen. Zunächst wird heuristisch eine Plazierung vorgenommen, die über "Hilfszielfunktionen", die "Verdrahtungsproblematik" berücksichtigt. Ausgehend von einer gegebenen Plazierung werden dann sukzessive die einzelnen Netze (mit heuristischen Verfahren) verdrahtet (Routing). Dabei kann es geschehen, daß man feststellt, daß die bei der Plazierung vorgesehenen Verdrahtungskanäle nicht ausreichen, um alle Verbindungen zu ziehen. Die Plazierung wird daraufhin (meistens lokal) geändert, und es wird neu verdrahtet. Das Verfahren wird so lange wiederholt bis alle Zellen untergebracht und alle Netze verdrahtet sind. Anschließend wird "nachoptimiert", d. h. es wird versucht, noch ein wenig zu "drehen", zu "quetschen" und zu "schieben", um die Schaltung auf eine kleinere Fläche zu bringen. Dies nennt man auch Kompaktierung.

Das Gesamtverfahren der Plazierung und Verdrahtung wird also zerlegt in Teilprobleme, die nacheinander gelöst und häufig durch interaktiven menschlichen Eingriff korrigiert und modifiziert werden, um das Ziel zu erreichen, Chips mit möglichst wenig "toter Fläche" zu entwerfen. Es ist klar, daß man durch diese Kombination von Heuristiken und die Zerlegung in Stufen selten eine optimale Flächenausnutzung erreicht. Jedoch sind hier die Anzahlen der Variablen so groß und die zu berücksichtigenden Nebenbedingungen technischer Art so komplex, daß derzeit keine Möglichkeit besteht, eine optimale Plazierung und Verdrahtung "auf einen Schlag" zu bestimmen.

Ein guter Überblick über diesen Themenkreis ist in dem Buch Th. Lengauer (1990) gegeben. Hier finden sich viele Details über die technischen Rahmenbedingungen und die benutzten Verfahren zur Plazierung und Verdrahtung.

(2.18) Lineare Anordnungen und azyklische Subdigraphen. Gegeben sei ein vollständiger Digraph $D_n = (V, A)$ mit Bogengewichten $c((i,j))$ für alle $(i,j) \in A$. Das Problem, eine lineare Reihenfolge der Knoten, sagen wir $i_1, \ldots, i_n$, zu bestimmen, so daß die Summe der Gewichte der Bögen, die konsistent mit der linearen Ordnung sind (also $\sum_{p=1}^{n-1} \sum_{q=p+1}^{n} c((i_p, i_q)))$, maximal ist, heißt **Linear-Ordering-Problem**. Das **Azyklische-Subdigraphen-Problem** ist die Aufgabe, in einem Digraphen $D = (V, A)$ mit Bogengewichten eine Bogenmenge $B \subseteq A$ zu finden, die keinen gerichteten Kreis enthält und deren Gewicht maximal ist. Beim **Feedback-Arc-Set-Problem** sucht man eine Bogenmenge minimalen Gewichts, deren Entfernung aus dem Digraphen alle gerichteten Kreise zerstört.

□

Die drei in (2.18) genannten Probleme sind auf einfache Weise ineinander transformierbar. Diese Probleme haben interessante Anwendungen z. B. bei der Triangulation von Input-Output-Matrizen, der Rangbestimmung in Turniersportarten, im Marketing und der Psychologie. Weitergehende Information findet sich in Grötschel, Jünger und Reinelt (1984) und Reinelt (1985). Einige konkrete Anwendungsbeispiele werden in den Übungen behandelt.

(2.19) Entwurf kostengünstiger und ausfallsicherer Telekommunikationsnetzwerke. Weltweit werden heute die Kupferkabel, die Telefongespräche etc. übertragen, durch Glasfaserkabel ersetzt. Da Glasfaserkabel enorm hohe Übertragungskapazitäten haben, wurden anfangs die stark "vermaschten" Kupferkabelnetzwerke durch Glasfasernetzwerke mit Baumstruktur ersetzt. Diese Netzwerkstrukturen haben jedoch den Nachteil, daß beim Ausfall eines Verbindungskabels (z. B. bei Baggerarbeiten) oder eines Netzknoten (z. B. durch einen Brand) große Netzteile nicht mehr miteinander kommunizieren können. Man ist daher dazu übergegangen, Telekommunikationsnetzwerke mit höherer Ausfallsicherheit wie folgt auszulegen. Zunächst wird ein Graph $G = (V, E)$ bestimmt; hierbei repräsentiert V die Knotenpunkte, die in einem Telekommunikationsnetz verknüpft werden sollen, und E stellt die Verbindungen zwischen Knoten dar, die durch das Ziehen eines direkten (Glasfaser-) Verbindungskabels realisiert werden können. Gleichzeitig wird geschätzt, was das Legen einer direkten Kabelverbindung kostet. Anschließend wird festgelegt, welche Sicherheitsanforderungen das Netz erfüllen soll. Dies wird so gemacht, daß man für je zwei Knoten bestimmt, ob das Netz noch eine Verbindung zwischen diesen beiden Knoten besitzen soll, wenn ein, zwei, drei ... Kanten oder einige andere Knoten ausfallen. Dann wird ein Netzwerk bestimmt, also eine Teilmenge F von E, so daß alle Knoten miteinander kommunizieren können, alle Sicherheitsanforderungen erfüllt werden und die Baukosten minimal sind. Mit Hilfe dieses Modells (und zu seiner Lösung entwickelter Algorithmen) werden derzeit z. B. in den USA Glasfasernetzwerke für sogenannte LATA-Netze entworfen und ausgelegt, siehe Grötschel, Monma, Stoer (1992) und (1995) und Stoer (1992). Abbildung 2.6 (a) zeigt (von einer großen Stadt in den USA) das Netzwerk der möglichen direkten Kabelverbindungen, Abbildung 2.6 (b) zeigt eine optimale Lösung. Hierbei sind je zwei durch ein Quadrat gekennzeichnete Knoten gegen den Ausfall eines beliebigen Kabels geschützt (d. h. falls ein Kabel durchschnitten wird, gibt es noch eine (nicht notwendig direkte sondern auch über Zwischenknoten verlaufende) Verbindung zwischen je zwei dieser Knoten, alle übrigen Knotenpaare wurden als relativ unwichtig erachtet und mußten nicht gegen Kabelausfälle geschützt werden.

□

Abb. 2.6a

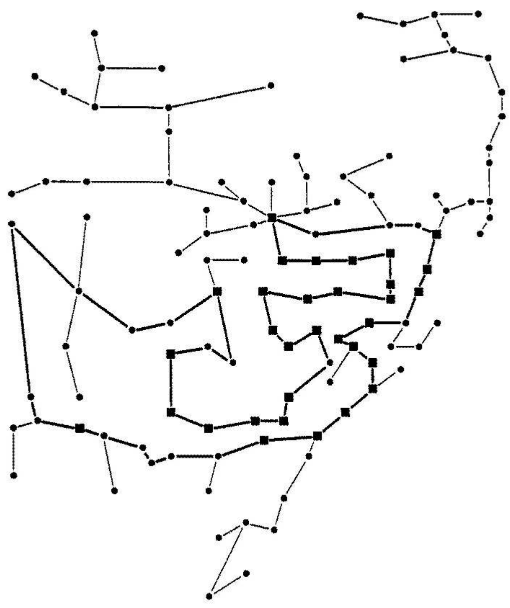

Abb. 2.6b

Literaturverzeichnis von Kapitel 2

M. Aigner (1984), *Graphentheorie: Eine Entwicklung aus dem 4-Farben-Problem,* Teubner Verlag, Stuttgart.

F. Barahona, M. Grötschel, M. Jünger und G. Reinelt (1988), "An application of combinatorial optimization to statistical physics and circuit layout design", Operations Research 36 (1988) 493-513.

C. Berge (1989), *Hypergraphs: Combinatorics of Finite Sets,* North-Holland, Amsterdam.

C. Berge, A. Ghouila-Houri (1969), *Programme, Spiele, Transportnetze,* Teubner, Leipzig.

B. Bollobás (1979), *Graph Theory: An Introductory Course,* Springer Verlag, New York.

J. A. Bondy und U. S. R. Murty (1976), *Graph Theory with Applications,* American Elsevier, New York and Macmillan, London.

W. Domschke (1982), *Logistik: Rundreisen und Touren,* Oldenbourg, München.

J. Ebert (1981), *Effiziente Graphenalgorithmen,* Wiesbaden.

M. C. Golumbic (1980), *Algorithmic Graph Theory and Perfect Graphs,* Academic Press, New York.

R. L. Graham, M. Grötschel und L. Lovász (1995), *Handbook of Combinatorics,* Volume I, 1-1018, North-Holland, Amsterdam.

R. L. Graham, M. Grötschel und L. Lovász (1995), *Handbook of Combinatorics,* Volume II, 1019-2198, North-Holland, Amsterdam.

M. Grötschel, M. Jünger und G. Reinelt (1984), "A cutting plane algorithm for the linear ordering problem", Operations Research 32, 1195-1220.

M. Grötschel und O. Holland (1991), "Solution of large-scale symmetric travelling salesman problems", Mathematical Programming 51, 141-202.

M. Grötschel, C. Monma und M. Stoer (1992), "Computational results with a cutting plane algorithm for designing communication network with low connectivity constraints", Operations Research 40, 309-330.

M. Grötschel, C. Monma und M. Stoer (1995), "Design of survivable networks", Ball, M. O. et al. (Hrsg.): *Handbook in Operations Research and Management Science* Vol. 7: Network Models (1995) Elsevier Science B.V., 617-672..

R. Halin (1989), *Graphentheorie* , Wissenschaftliche Buchgesellschaft, Darmstadt.

K. Hässig (1979), *Graphentheoretische Methoden des Operations Research,* Teubner, Stuttgart.

D. König (1936), *Theorie der endlichen und unendlichen Graphen,* Akademische Verlagsgesellschaft, Leipzig, (mehrfach auf deutsch und in englischer Übersetzung nachgedruckt).

E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan und D. B. Shmoys (1985), *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization,* Wiley, Chichester.

Th. Lengauer (1990), *Combinatorial Algorithms for Integrated Circuit Layout,* Teubner, Stuttgart und Wiley, Chichester.

J. K. Lenstra (1976), *Sequencing by Enumerative Methods,* Dissertation, Mathematisch Centrum, Amsterdam.

J. G. Oxley (1992), *Matroid Theory,* Oxford University Press, Oxford.

G. Reinelt (1985), *The Linear Ordering Problem: Algorithms and Applications,* Heldermann Verlag, Berlin.

H. Sachs (1970), *Einführung in die Theorie der endlichen Graphen,* Teubner, Leipzig, 1970, ebenfalls erschienen: Hanser, München, 1971.

M. Stoer (1992), *Design of survivable networks,* Springer Verlag, Heidelberg, Lecture Notes for Mathematics.

K. Truemper (1992), *Matroid Decomposition,* Academic Press, Boston.

K. Wagner (1970), *Graphentheorie,* BI Wissenschaftsverlag, Mannheim.

H. Walther und G. Nägler (1987), *Graphen, Algorithmen, Programme* , VEB Fachbuchverlag, Leipzig.

D. J. A. Welsh (1976), *Matroid Theory* , Academic Press, London.

Kapitel 3

Komplexität, Speicherung von Daten

In diesem Kapitel führen wir einige der Grundbegriffe der Komplexitätstheorie ein, die für die kombinatorische Optimierung von Bedeutung sind, insbesondere die Klassen $\mathcal{P}$ und $\mathcal{NP}$ und das Konzept der $\mathcal{NP}$-Vollständigkeit. Die Darstellung erfolgt auf informellen Niveau. Gute Bücher zur Einführung in die Komplexitätstheorie sind Garey und Johnson (1979) und Papadimitriou (1994), weiterführende Aspekte werden z. B. in Wagner und Wechsung (1986) und van Leeuwen (1990) behandelt. Shmoys und Tardos (1995) ist ein guter Übersichtsartikel.

Komplexitätstheorie kann man nicht ohne Grundkenntnisse in Kodierungstechniken betreiben. Noch wichtiger aber sind Datenstrukturen beim Entwurf effizienter Algorithmen. Diesen Themenkreis streifen wir kurz in Abschnitt 3.3. Zur Vertiefung dieses Gebietes empfehlen wir Aho, Hopcroft und Ulman (1974), Cormen, Leiserson und Rivest (1990), Mehlhorn (1984), Meinel (1991), Ottmann und Widmayer (1990) und Tarjan (1983).

3.1 Probleme, Komplexitätsmaße, Laufzeiten

In der Mathematik (und nicht nur hier) kann das Wort "Problem" sehr verschiedene Bedeutungen haben. Für unsere Zwecke benötigen wir eine (einigermaßen) präzise Definition. Ein **Problem** ist eine allgemeine Fragestellung, bei der mehrere Parameter offen gelassen sind und für die eine Lösung oder Antwort gesucht wird. Ein Problem ist dadurch definiert, daß alle seine Parameter beschrieben werden und daß genau angegeben wird, welche Eigenschaften eine Antwort (Lösung) haben soll. Sind alle Parameter eines Problems mit expliziten Daten belegt, dann spricht man im Englischen von einem "problem instance". Im Deutschen hat sich hierfür bisher kein Standardbegriff ausgeprägt. Es sind u. a. die Wörter Einzelfall, Fallbeispiel, Problembeispiel, Probleminstanz oder Problemausprägung gebräuchlich. Ebenso wird auch das (allgemeine) Problem manchmal als Problemtyp oder Problemklasse bezeichnet. In diesem Skript werden wir keiner starren Regel folgen. Aus dem Kontext heraus dürfte i. a. klar sein, was gemeint ist.

Das Travelling-Salesman-Problem ist in diesem Sinne ein Problem. Seine offenen Parameter sind die Anzahl der Städte und die Entfernungen zwischen diesen Städten. Eine Entfernungstabelle in einem Autoatlas definiert ein konkretes Beispiel für das Travelling-Salesman-Problem.

Wir sagen, daß ein Algorithmus Problem Π **löst**, wenn er für jedes Problembeispiel $\mathcal{I} \in \Pi$ eine Lösung findet. Das Ziel des Entwurfs von Algorithmen ist natürlich, möglichst "effiziente" Verfahren zur Lösung von Problemen zu finden.

Um dieses Ziel mit Inhalt zu füllen, müssen wir den Begriff “Effizienz” meßbar machen. Mathematiker und Informatiker haben hierzu verschiedene **Komplexitätsmaße** definiert. Wir werden uns hier nur mit den beiden für uns wichtigsten Begriffen **Zeit-** und **Speicherkomplexität** beschäftigen. Hauptsächlich werden wir über Zeitkomplexität reden.

Es ist trivial, daß die Laufzeit eines Algorithmus abhängt von der “Größe” eines Problembeispiels, d. h. vom Umfang der Eingabedaten. Bevor wir also Laufzeitanalysen anstellen können, müssen wir beschreiben, wie wir unsere Problembeispiele darstellen, bzw. kodieren wollen. Allgemein kann man das durch die Angabe von **Kodierungsschemata** bewerkstelligen. Da wir uns jedoch ausschließlich mit Problemen beschäftigen, die mathematisch darstellbare Strukturen haben, reicht es für unsere Zwecke aus, Kodierungsvorschriften für die uns interessierenden Strukturen anzugeben. Natürlich gibt es für jede Struktur beliebig viele Kodierungsmöglichkeiten. Wir werden die geläufigsten benutzen und merken an, daß auf diesen oder dazu (in einem spezifizierbaren Sinn) äquivalenten Kodierungsvorschriften die gesamte derzeitige Komplexitätstheorie aufbaut.

Ganze Zahlen kodieren wir **binär**. Die binäre Darstellung einer nicht-negativen ganzen Zahl n benötigt $\lceil \log_2(|n|+1)\rceil$ Bits (oder Zellen). Hinzu kommt ein Bit für das Vorzeichen. Die **Kodierungslänge** $\langle n\rangle$ einer ganzen Zahl n ist die Anzahl der zu ihrer Darstellung notwendigen Bits, d.h.,

$$\langle n\rangle := \lceil \log_2(|n|+1)\rceil + 1. \tag{3.1}$$

Jede rationale Zahl r hat eine Darstellung $r = \frac{p}{q}$ mit $p, q \in \mathbf{Z}$, p und q teilerfremd und $q > 0$. Wir nehmen an, daß jede rationale Zahl so dargestellt ist und können daher sagen, daß die Kodierungslänge von $r = \frac{p}{q}$ gegeben ist durch

$$\langle r\rangle := \langle p\rangle + \langle q\rangle.$$

Wir werden im weiteren auch sagen, daß wir eine ganze oder rationale Zahl r in einem Speicherplatz (oder Register) speichern, und wir gehen davon aus, daß der Speicherplatz für r genau die benötigten $\langle r\rangle$ Zellen besitzt.

Die Kodierungslänge eines Vektors $x = (x_1, \dots, x_n)^T \in \mathbf{Q}^n$ ist

$$\langle x\rangle := \sum_{i=1}^{n} \langle x_i\rangle,$$

und die Kodierungslänge einer Matrix $A \in \mathbf{Q}^{(m,n)}$ ist

$$\langle A\rangle := \sum_{i=1}^{m}\sum_{j=1}^{n} \langle a_{ij}\rangle.$$

Für diese Vorlesung besonders wichtig sind die Datenstrukturen zur Kodierung von Graphen und Digraphen. Auf diese werden wir in Abschnitt 3.3 genauer eingehen.

Sind alle Kodierungsvorschriften festgelegt, so müssen wir ein Rechnermodell entwerfen, auf dem unsere Speicher- und Laufzeitberechnungen durchgeführt werden sollen. In der Komplexitätstheorie benutzt man hierzu i. a. Turing-Maschinen oder RAM-Maschinen. Wir wollen auf diese Rechnermodelle nicht genauer eingehen. Wir nehmen an, daß der Leser weiß, was Computer sind und wie sie funktionieren, und unterstellen einfach, daß jeder eine naive Vorstellung von einer "vernünftigen" Rechenmaschine hat. Dies reicht für unsere Zwecke aus.

Wir stellen uns den Ablauf eines Algorithmus A (der in Form eines Rechnerprogramms vorliegt) auf einer Rechenanlage wie folgt vor. Der Algorithmus soll Problembeispiele $\mathcal{I}$ des Problems Π lösen. Alle Problembeispiele liegen in kodierter Form vor. Die Anzahl der Zellen, die notwendig sind, um $\mathcal{I}$ vollständig anzugeben, nennen wir die **Kodierungslänge** oder **Inputlänge** $\langle \mathcal{I} \rangle$ von $\mathcal{I}$. Der Algorithmus liest diese Daten und beginnt dann Operationen auszuführen, d. h. Zahlen zu berechnen, zu speichern, zu löschen, usw.. Die Anzahl der Zellen, die während der Ausführung des Algorithmus A mindestens einmal benutzt wurden, nennen wir den **Speicherbedarf von A zur Lösung von $\mathcal{I}$**. Üblicherweise schätzt man den Speicherbedarf eines Algorithmus A dadurch, daß man die Anzahl der von A benutzten Speicherplätze bestimmt und diesen Wert mit der größten Anzahl von Zellen multipliziert, die einer der Speicherplätze beim Ablauf des Algorithmus benötigte.

Die **Laufzeit von A zur Lösung von $\mathcal{I}$** ist (etwas salopp gesagt) die Anzahl der elementaren Operationen, die A bis zur Beendigung des Verfahrens ausgeführt hat. Dabei wollen wir als **elementare Operationen** zählen:

Lesen, Schreiben und Löschen,

Addieren, Subtrahieren, Multiplizieren, Dividieren und Vergleichen

von rationalen (oder ganzen) Zahlen. Da ja zur Darstellung derartiger Zahlen mehrere Zellen benötigt werden, muß zur genauen Berechnung der Laufzeit jede elementare Operation mit den Kodierungslängen der involvierten Zahlen multipliziert werden. Die **Laufzeit von A zur Lösung von $\mathcal{I}$** ist die Anzahl der elementaren Rechenoperationen, die A ausgeführt hat, um eine Lösung von $\mathcal{I}$ zu finden, multipliziert mit der Kodierungslänge der bezüglich der Kodierungslänge größten ganzen oder rationalen Zahl, die während der Ausführung des Algorithmus aufgetreten ist. Wir tun also so, als hätten wir alle elementaren Operationen mit der in diesem Sinne größten Zahl ausgeführt und erhalten somit sicherlich eine Abschätzung der "echten" Laufzeit nach oben.

(3.2) Definition. *Sei A ein Algorithmus zur Lösung eines Problems Π.*

(a) *Die Funktion $f_A : \mathbf{N} \longrightarrow \mathbf{N}$ definiert durch*

$$f_A(n) := \max\{\textit{Laufzeit von } A \textit{ zur Lösung von } \mathcal{I} \mid \mathcal{I} \in \Pi \textit{ und } \langle \mathcal{I} \rangle \leq n\}$$

heißt **Laufzeitfunktion** *von A.*

(b) *Die Funktion $s_A : \mathbf{N} \longrightarrow \mathbf{N}$ definiert durch*

$$s_A(n) := \max\{\textit{Speicherbedarf von } A \textit{ zur Lösung von } \mathcal{I} \mid \mathcal{I} \in \Pi \textit{ und } \langle \mathcal{I} \rangle \leq n\}$$

heißt **Speicherplatzfunktion** *von A.*

(c) *Der Algorithmus A hat eine* **polynomiale Laufzeit** *(kurz: A ist ein* **polynomialer Algorithmus***), wenn es ein Polynom* $p : \mathbf{N} \longrightarrow \mathbf{N}$ *gibt mit*

$$f_A(n) \leq p(n) \quad \text{für alle } n \in \mathbf{N}.$$

Wir sagen f_A *ist von der Ordnung höchstens* n^k *(geschrieben* $f_A = O(n^k)$*), falls das Polynom* p *den Grad* k *hat.*

(d) *Der Algorithmus A hat* **polynomialen Speicherplatzbedarf***, falls es ein Polynom* $q : \mathbf{N} \longrightarrow \mathbf{N}$ *gibt mit*

$$s_A(n) \leq q(n) \quad \text{für alle } n \in \mathbf{N}.$$

□

Wir werden uns im ersten Teil der Vorlesung mit Problemen beschäftigen, für die polynomiale Algorithmen existieren. Später werden wir Probleme behandeln, die in einem noch zu präzisierenden Sinne "schwieriger" sind und für die (bisher) noch keine polynomialen Verfahren gefunden worden sind.

Eine triviale Bemerkung sei hier gemacht. Ein Algorithmus, dessen Speicherplatzfunktion nicht durch ein Polynom beschränkt werden kann, kann keine polynomiale Laufzeit haben, da nach Definition die Benutzung eines Speicherplatzes in die Berechnung der Laufzeitfunktion eingeht.

(3.3) Hausaufgabe. *Bestimmen Sie die Laufzeitfunktion und die Speicherplatzfunktion des folgenden Algorithmus:*

Input: *ganze Zahl* n

$k := \langle n \rangle$
DO $i = 1$ *TO* k*:*
$n := n \cdot n \cdot n$
END
Gib n *aus.*

□

3.2 Die Klassen $\mathcal{P}$ und $\mathcal{NP}$, $\mathcal{NP}$-Vollständigkeit

Wir wollen nun einige weitere Begriffe einführen, um zwischen "einfachen" und "schwierigen" Problemen unterscheiden zu können. Wir werden dabei zunächst — aus technischen Gründen — nur Entscheidungsprobleme behandeln und später die Konzepte auf (die uns eigentlich interessierenden) Optimierungsprobleme erweitern. Ein **Entscheidungsproblem** ist ein Problem, das nur zwei mögliche Antworten besitzt, nämlich "ja" oder "nein". Die Fragen "Enthält ein Graph einen Kreis?", "Enthält ein Graph einen hamiltonschen Kreis?", "Ist die Zahl n eine Primzahl?" sind z. B. Entscheidungsprobleme. Da wir uns nicht mit (für uns unwichtigen) Feinheiten der Komplexitätstheorie beschäftigen wollen, werden wir im weiteren nur solche Entscheidungsprobleme als Entscheidungsprobleme betrachten, für die Lösungsalgorithmen mit endlicher Laufzeitfunktion existieren.

Die Klasse aller derjenigen Entscheidungsprobleme, für die ein polynomialer Lösungsalgorithmus existiert, wird mit $\mathcal{P}$ bezeichnet. Diese Definition ist recht informell. Wenn wir genauer wären, müßten wir $\mathcal{P}$ relativ zu einem Kodierungsschema und zu einem Rechnermodell definieren. Die Definition würde dann etwa wie folgt lauten. Gegeben sei ein Kodierungsschema E und ein Rechnermodell M, Π sei ein Entscheidungsproblem, wobei jedes Problembeispiel aus Π durch das Kodierungsschema E kodiert werden kann. Π gehört zur Klasse $\mathcal{P}$ (bezüglich E und M), wenn es einen auf M implementierbaren Algorithmus zur Lösung der Problembeispiele aus Π gibt, dessen Laufzeitfunktion auf M polynomial ist. Wir wollen im weiteren derartig komplizierte und unübersichtliche Definitionen vermeiden und werden auf dem (bisherigen) informellen Niveau bleiben in der Annahme, die wesentlichen Anliegen ausreichend klar machen zu können.

Wir werden im Abschnitt 3.3 sehen, daß das Problem "Enthält ein Graph einen Kreis?" zur Klasse $\mathcal{P}$ gehört. Aber trotz enormer Anstrengungen sehr vieler Forscher ist es noch nicht gelungen, das Problem "Enthält ein Graph einen hamiltonschen Kreis" in polynomialer Zeit zu lösen.

Diese Frage ist "offenbar" schwieriger. Um zwischen den Problemen in $\mathcal{P}$ und den "schwierigeren" formal unterscheiden zu können, sind die folgenden Begriffe geprägt worden.

Wir sagen – zunächst einmal informell –, daß ein Entscheidungsproblem Π zur Klasse $\mathcal{NP}$ gehört, wenn es die folgende Eigenschaft hat: Ist die Antwort für ein Problembeispiel $\mathcal{I} \in \Pi$ "ja", dann kann die Korrektheit der Antwort in polynomialer Zeit überprüft werden.

Bevor wir etwas genauer werden, betrachten wir ein Beispiel. Wir wollen herausfinden, ob ein Graph einen hamiltonschen Kreis enthält. Enthält er einen, und jemand hat ihn gefunden und alle Kanten des Kreises rot angestrichen, dann können wir auf einfache Weise überprüfen, ob die rot angemalten Kanten tatsächlich einen hamiltonschen Kreis bilden. Bilden sie einen solchen Kreis, so haben wir die Korrektheit der "ja"-Antwort in polynomialer Zeit verifiziert.
Nun die ausführliche Definition.

(3.4) Definition. *Ein Entscheidungsproblem Π gehört zur Klasse $\mathcal{NP}$, wenn es die folgenden Eigenschaften hat:*

(a) *Für jedes Problembeispiel $\mathcal{I} \in \Pi$, für das die Antwort "ja" lautet, gibt es mindestens ein Objekt Q, mit dessen Hilfe die Korrektheit der "ja"-Antwort überprüft werden kann.*

(b) *Es gibt einen Algorithmus, der Problembeispiele $\mathcal{I} \in \Pi$ und Zusatzobjekte Q als Input akzeptiert und der in einer Laufzeit, die polynomial in $\langle \mathcal{I} \rangle$ ist, überprüft, ob Q ein Objekt ist, aufgrund dessen Existenz eine "ja"-Antwort für $\mathcal{I}$ gegeben werden muß.*

$\square$

Die Probleme "Hat ein Graph G einen Kreis?", "Hat ein Graph G einen hamiltonschen Kreis?" sind somit in $\mathcal{NP}$. Hat nämlich G einen Kreis oder hamiltonschen Kreis, so wählen wir diesen als Objekt Q. Dann entwerfen wir einen polynomialen Algorithmus, der für einen Graphen G und eine zusätzliche Kantenmenge Q entscheidet, ob Q ein Kreis oder hamiltonscher Kreis von G ist. Auch die Frage "Ist $n \in \mathbf{N}$ eine zusammengesetzte Zahl?" ist in $\mathcal{NP}$, denn liefern wir als "Objekt" zwei Zahlen $\neq 1$, deren Produkt n ist, so ist n keine Primzahl. Die Überprüfung der Korrektheit besteht somit in diesem Fall aus einer einzigen Multiplikation.

Die obige Definition der Klasse $\mathcal{NP}$ enthält einige Feinheiten, auf die ich ausdrücklich hinweisen möchte.

- Es wird nichts darüber gesagt, wie das Zusatzobjekt Q zu finden ist. Es wird lediglich postuliert, daß es existiert, aber nicht, daß man es z. B. mit einem polynomialen Algorithmus finden kann.
- Die Laufzeit des Algorithmus in (b) ist nach Definition polynomial in $\langle \mathcal{I} \rangle$. Da der Algorithmus Q lesen muß, folgt daraus, daß die Kodierungslänge von Q durch ein Polynom in der Kodierungslänge von $\mathcal{I}$ beschränkt sein muß. Auf die Frage "Hat die Gleichung $x^2 + y^2 = n^2$ eine Lösung x, y?" ist "$x = n$ und $y = 0$" ein geeignetes Zusatzobjekt Q, aber weder "$x = y = n\sqrt{0.5}$" ($\sqrt{0.5}$ kann nicht endlich binär kodiert werden) noch "$x = \frac{n2^{2^n}}{2^{2^n}}, y = 0$" (die Kodierungslänge von x ist exponentiell in der Inputlänge des Problems) wären als Zusatzobjekt Q geeignet, um die Korrektheit der "ja"-Antwort in polynomialer Zeit verifizieren zu können.
- Ferner ist die Definition von $\mathcal{NP}$ unsymmetrisch in "ja" und "nein". Die Definition impliziert nicht, daß wir auch für die Problembeispiele mit "nein"-Antworten Objekte Q und polynomiale Algorithmen mit den in (3.4) (a) und (b) spezifizierten Eigenschaften finden können.

Wir sagen, daß die Entscheidungsprobleme, die Negationen von Problemen aus der Klasse $\mathcal{NP}$ sind, zur Klasse co-$\mathcal{NP}$ gehören. Dies sind Probleme komplementär zu den $\mathcal{NP}$-Problemen. Zu co-$\mathcal{NP}$ gehören folglich die Probleme "Hat G keinen Kreis?", "Hat G keinen hamiltonschen Kreis?", "Ist $n \in \mathbf{N}$ eine Primzahl?". Es ist bekannt, daß das erste und das letzte dieser drei Probleme ebenfalls zu $\mathcal{NP}$ gehören. Diese beiden Probleme gehören also zu $\mathcal{NP} \cap$ co-$\mathcal{NP}$. Vom Problem "Hat G keinen hamiltonschen Kreis?" weiß man nicht, ob es zu $\mathcal{NP}$ gehört. Niemand hat bisher Objekte Q und einen Algorithmus finden können, die den Forderungen (a) und (b) aus (3.4) genügen.

Das Symbol $\mathcal{NP}$ ist abgeleitet vom Begriff "nichtdeterministischer polynomialer Algorithmus". Dies sind – grob gesagt – Algorithmen, die am Anfang "raten" können, also einen nichtdeterministischen Schritt ausführen können und dann wie übliche Algorithmen ablaufen. Ein nichtdeterministischer Algorithmus "löst" z. B. das hamiltonsche Graphenproblem wie folgt. Am Anfang rät er einen hamiltonschen Kreis. Gibt es keinen, so hört das Verfahren auf. Gibt es einen, so überprüft er, ob das geratene Objekt tatsächlich ein hamiltonscher Kreis ist. Ist das so, so antwortet er mit "ja".

Trivialerweise gilt $\mathcal{P} \subseteq \mathcal{NP}$, da für Probleme in $\mathcal{P}$ Algorithmen existieren, die ohne Zusatzobjekte Q in polynomialer Zeit eine "ja"- oder "nein"-Antwort liefern. Also gilt auch $\mathcal{P} \subseteq$ co-$\mathcal{NP}$. Eigentlich sollte man meinen, daß Algorithmen, die raten können, mächtiger sind als übliche Algorithmen. Trotz gewaltiger Forschungsanstrengungen in den vergangenen 25 Jahren ist die Frage, ob $\mathcal{P} = \mathcal{NP}$ gilt oder nicht, immer noch ungelöst. Meiner Meinung nach ist dieses Problem eines der wichtigsten offenen Probleme der heutigen Mathematik und Informatik. Jeder, der sich mit diesem Problem beschäftigt hat, glaubt, daß $\mathcal{P} \neq \mathcal{NP}$ gilt. Könnte diese Vermutung bestätigt werden, so würde das – wie wir gleich sehen werden – bedeuten, daß für eine sehr große Zahl praxisrelevanter Probleme niemals wirklich effiziente Lösungsalgorithmen gefunden werden können. Wir werden uns also mit der effizienten Auffindung suboptimaler Lösungen zufrieden geben und daher auf den Entwurf von Heuristiken konzentrieren müssen. Deswegen wird auch im weiteren Verlauf der Vorlesung viel Wert auf die Untersuchung und Analyse von Heuristiken gelegt.

Wir haben gesehen, daß $\mathcal{P} \subseteq \mathcal{NP} \cap \text{co-}\mathcal{NP}$ gilt. Auch bezüglich der Verhältnisse dieser drei Klassen zueinander gibt es einige offene Fragen.

$$\begin{aligned} &\text{Gilt} \quad \mathcal{P} = \mathcal{NP} \cap \text{co-}\mathcal{NP}? \\ &\text{Gilt } \mathcal{NP} \ = \text{co-}\mathcal{NP}? \end{aligned}$$

Aus $\mathcal{NP} \neq \text{co-}\mathcal{NP}$ würde $\mathcal{P} \neq \mathcal{NP}$ folgen, da offenbar $\mathcal{P} = \text{co-}\mathcal{P}$ gilt.

Wir wollen nun innnerhalb der Klasse $\mathcal{NP}$ eine Klasse von besonders schwierigen Problemen auszeichnen.

(3.5) Definition. *Gegeben seien zwei Entscheidungsprobleme* Π *und* Π'. *Eine* **polynomiale Transformation** *von* Π *in* Π' *ist ein polynomialer Algorithmus, der, gegeben ein (kodiertes) Problembeispiel* $\mathcal{I} \in \Pi$, *ein (kodiertes) Problembeispiel* $\mathcal{I}' \in \Pi'$ *produziert, so daß folgendes gilt:*

Die Anwort auf $\mathcal{I}$ *ist genau dann "ja", wenn die Antwort auf* $\mathcal{I}'$ *"ja" ist.*

$\square$

Offenbar gilt folgendes. Ist Π in Π' polynomial transformierbar und gibt es einen polynomialen Algorithmus zur Lösung von Π', dann kann man auch Π in polynomialer Zeit lösen. Man transformiert einfach jedes Problembeispiel aus Π in eine Problembeispiel aus Π' und wendet den Algorithmus für Π' an. Da sowohl der Transformationsalgorithmus als auch der Lösungsalgorithmus polynomial sind, hat auch die Kombination beider Algorithmen eine polynomiale Laufzeit.

Nun kommen wir zu einem der wichtigsten Begriffe dieser Theorie, der spezifiziert, welches die schwierigsten Probleme in der Klasse $\mathcal{NP}$ sind.

(3.6) Definition. *Ein Entscheidungsproblem* Π *heißt* $\mathcal{NP}$**-vollständig**, *falls* $\Pi \in \mathcal{NP}$ *und falls jedes andere Problem aus* $\mathcal{NP}$ *polynomial in* Π *transformiert werden kann.*

$\square$

Jedes $\mathcal{NP}$-vollständige Entscheidungsproblem Π hat also die folgende Eigenschaft. Falls Π in polynomialer Zeit gelöst werden kann, dann kann auch jedes andere Problem aus $\mathcal{NP}$ in polynomialer Zeit gelöst werden; in Formeln:

$$\Pi \ \mathcal{NP}\text{-vollständig und } \Pi \in \mathcal{P} \Longrightarrow \mathcal{P} = \mathcal{NP}.$$

Diese Eigenschaft zeigt, daß – bezüglich polynomialer Lösbarkeit – kein Problem in $\mathcal{NP}$ schwieriger ist als ein $\mathcal{NP}$-vollständiges. Natürlich stellt sich sofort die Frage, ob es überhaupt $\mathcal{NP}$-vollständige Probleme gibt. Dies hat Cook (1971) in einer für die Komplextiätstheorie fundamentalen Arbeit beweisen. In der Tat sind (leider) fast alle praxisrelevanten Probleme $\mathcal{NP}$-vollständig. Ein Beispiel: Das hamiltonsche Graphenproblem "Enthält G einen hamiltonschen Kreis?" ist $\mathcal{NP}$-vollständig.

Ein interessantes Problem, dessen Status in einem gewissen Sinne noch offen ist, ist das Primzahlproblem "Ist $n \in \mathbf{N}$ prim?". Man kann zeigen, daß es in $\mathcal{NP} \cap \text{co-}\mathcal{NP}$ liegt, aber es nicht bekannt, ob dieses Problem in $\mathcal{P}$ liegt oder $\mathcal{NP}$-vollständig ist.

Wir wollen nun Optimierungsprobleme in unsere Betrachtungen einbeziehen. Aus jedem Optimierungsproblem kann man wie folgt ein Entscheidungsproblem machen. Ist Π ein Maximierungsproblem (Minimierungsproblem), so legt man zusätzlich zu jedem Problembeispiel $\mathcal{I}$ noch eine Schranke, sagen wir B, fest und fragt

Gibt es für $\mathcal{I}$ eine Lösung, deren Wert nicht kleiner (nicht größer) als B ist?

Aus dem Travelling-Salesman-Problem wird auf diese Weise ein Entscheidungsproblem, indem man fragt "Enthält das Problembeispiel eine Rundreise, deren Länge nicht größer als B ist?".

Wir nennen ein Optimierungsproblem **$\mathcal{NP}$-schwer**, wenn das (wie oben angegebene) zugeordnete Entscheidungsproblem $\mathcal{NP}$-vollständig ist. Diese Bezeichnung beinhaltet die Aussage, daß alle $\mathcal{NP}$-schweren Optimierungsprobleme mindestens so schwierig sind wie die $\mathcal{NP}$-vollständigen Probleme. Könnten wir nämlich ein $\mathcal{NP}$-schweres Problem (wie das Travelling-Salesman-Problem) in polynomialer Zeit lösen, dann könnten wir auch das zugehörige Entscheidungsproblem in polynomialer Zeit lösen. Wir berechnen den Wert w einer Optimallösung und vergleichen ihn mit B. Ist bei einem Maximerungsproblem (Minimierungsproblem) $w \geq B$ ($w \leq B$), so antworten wir "ja", andernfalls "nein".

Häufig kann man Entscheidungsprobleme dazu benutzen, um Optimierungsprobleme zu lösen. Betrachten wir als Beispiel das TSP-Entscheidungsproblem und nehmen wir an, daß alle Problembeispiele durch ganzzahlige Entfernungen zwischen den Städten gegeben sind. Seien s und t die kleinste bzw. größte Zahl, die als Entfernung vorkommt. Da jede Rundreise n Kanten enthält, ist keine Rundreise kürzer als ns und keine länger als nt. Wir fragen nun den Algorithmus zur Lösung des TSP-Entscheidungsproblems, ob es eine Rundreise gibt, deren Länge nicht größer als $\frac{n(t-s)}{2}$ ist. Ist das so, fragen wir, ob es eine Rundreise gibt, deren Länge höchstens $\frac{n(t-s)}{4}$ ist, andernfalls fragen wir, ob es eine Rundreise gibt mit Länge höchstens $\frac{3n(t-s)}{4}$. Wir fahren auf diese Weise fort bis wir das Intervall ganzer Zahlen, die als mögliche Länge einer kürzesten Rundreise in Frage kommen, auf eine Zahl reduziert haben. Diese Zahl muß dann die Länge der kürzesten Rundreise sein. Insgesamt haben wir zur Lösung des Optimierungsproblems das zugehörige TSP-Entscheidungsproblem ($\lceil \log_2(n(t-s)) \rceil + 1$)-mal aufgerufen, also eine polynomiale Anzahl von Aufrufen eines polynomialen Algorithmus vorgenommen. Dieser Algorithmus findet also in polynomialer Zeit die Länge einer kürzesten Rundreise – bei einem gegebenen polynomialen Algorithmus für das zugehörige Entscheidungsproblem. (Überlegen Sie sich, ob — und gegebenenfalls wie — man eine kürzeste Rundreise finden kann, wenn man ihre Länge kennt!)

Dem aufmerksamen Leser wird nicht entgangen sein, daß die oben beschriebene Methode zur Reduktion des Travelling-Salesman-Problems auf das zugehörige Entscheidungsproblem nichts anderes ist als das bekannte Verfahren der binären Suche.

Mit diesem oder ähnlichen "Tricks" lassen sich viele Optimierungsprobleme durch mehrfachen Aufruf von Algorithmen für Entscheidungsprobleme lösen. Wir nennen ein Optimerungsproblem Π **$\mathcal{NP}$-leicht**, falls es ein Entscheidungsproblem Π' in $\mathcal{NP}$ gibt, so daß Π durch polynomial viele Aufrufe eines Algorithmus zur Lösung von Π' gelöst werden kann. $\mathcal{NP}$-leichte Probleme sind also nicht schwerer als die Probleme in $\mathcal{NP}$. Unser Beispiel oben zeigt, daß das TSP auch $\mathcal{NP}$-leicht ist.

Wir nennen ein Optimierungsproblem **$\mathcal{NP}$-äquivalent**, wenn es sowohl $\mathcal{NP}$-leicht als auch $\mathcal{NP}$-schwer ist. Diese Bezeichnung ist im folgenden Sinne gerechtfertigt. Ein $\mathcal{NP}$-äquivalentes Problem ist genau dann in polynomialer Zeit lösbar, wenn $\mathcal{P} = \mathcal{NP}$ gilt. Wenn jemand einen polynomialen Algorithmus für das TSP findet, hat er damit gleichzeitig $\mathcal{P} = \mathcal{NP}$ bewiesen.

Wir wollen im weiteren dem allgemein üblichen Brauch folgen und die feinen Unterschiede zwischen den oben eingeführten Bezeichnungen für Entscheidungs- und Optimierungsprobleme nicht so genau nehmen. Wir werden häufig einfach von **$\mathcal{NP}$-vollständigen Optimierungsproblemen** sprechen, wenn diese $\mathcal{NP}$-schwer sind. Die Begriffe $\mathcal{NP}$-leicht und $\mathcal{NP}$-äquivalent werden wir kaum gebrauchen, da sie für unsere Belange nicht so wichtig sind.

In der nachfolgenden Tabelle haben wir einige der Beispiele von kombinatorischen Optimierungsproblemen aufgelistet, die wir in den Abschnitten 2.3 und 1.2 eingeführt haben und die $\mathcal{NP}$-schwer sind.

– das symmetrische Travelling Salesman Problem

– das asymmetrische Travelling Salesman Problem

– das Chinesische Postbotenproblem für gemischte Graphen

– fast alle Routenplanungsprobleme (2.13)

– das Stabile-Mengen-Problem

– das Cliquenproblem

– das Knotenüberdeckungsproblem

– das Knotenfärbungsproblem

– das Kantenfärbungsproblem

– das Max-Cut-Problem

– die meisten Standortprobleme (2.17)

– das Linear-Ordering-Problem

– das azyklische Subdigraphenproblem

– das Feedback-Arc-Set-Problem

– das allgemeine gemischt-ganzzahlige Optimierungsproblem (1.5)

– das ganzzahlige Optimierungsproblem (1.4)

– das 0/1-Optimierungsproblem (1.9)

– das Rucksackproblem (1.10)

– das Fixkostenproblem (1.15)

– das Maschinenbelegungsproblem (1.16)

Einige hundert weitere $\mathcal{NP}$-vollständige bzw. $\mathcal{NP}$-schwere Probleme und einige tausend Varianten von diesen sind in dem bereits zitierten Buch von Garey & Johnson (1979) aufgeführt. Probleme, die mit diesem Themenkreis zusammenhängen, werden auch in einer seit 1981 bestehenden, fortlaufenden Serie von Aufsätzen von D. S. Johnson mit dem Titel "The $\mathcal{NP}$-completeness column: an ongoing guide" im *Journal of Algorithms* behandelt.

Der wichtigste Aspekt der hier skizzierten Theorie ist, daß man zwischen "einfachen" und "schwierigen" Problemen zu unterscheiden lernt, und daß man – sobald man weiß, daß ein Problem schwierig ist – andere Wege (Heuristiken, etc.) als bei Problemen in $\mathcal{P}$ suchen muß, um das Problem optimal oder approximativ zu lösen. In dieser Vorlesung soll versucht werden, einige der Methoden zu beschreiben, mit denen man derartige Probleme angreifen kann.

3.3 Datenstrukturen zur Speicherung von Graphen

Wir gehen davon aus, daß jeder weiß, wie man auf Computern Zahlen, Vektoren und Matrizen speichert. Viele der Probleme, die wir in dieser Vorlesung behandeln, sind auf Graphen oder Digraphen definiert. Wir wollen daher hier einige Methoden skizzieren, mit deren Hilfe man Graphen und Digraphen speichern kann, und ihre Vor- und Nachteile besprechen. Kenntnisse dieser Datenstrukturen sind insbesondere dann wichtig, wenn man laufzeit- und speicherplatzeffiziente (oder gar -optimale) Codes von Algorithmen enwickeln will.

Kanten- und Bogenlisten

Die einfachste Art, einen Graphen oder Digraphen zu speichern, ist die **Kantenliste** für Graphen bzw. die **Bogenliste** für Digraphen. Ist $G = (V, E)$ ein Graph mit $n = |V|$

Knoten und $m = |E|$ Kanten, so sieht eine Kantenliste wie folgt aus:

$$n, m, a_1, e_1, a_2, e_2, a_3, e_3, \ldots, a_m, e_m,$$

wobei a_i, e_i die beiden Endknoten der Kante i sind. Die Reihenfolge des Aufführens der Endknoten von i bzw. den Kanten selbst ist beliebig. Bei Schleifen wird der Endknoten zweimal aufgelistet.

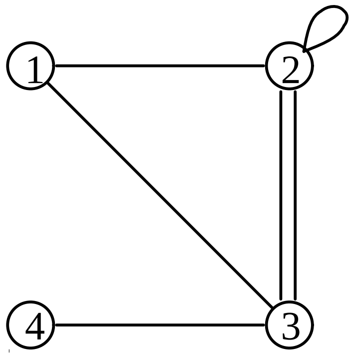

Abb. 3.1

Eine mögliche Kantenliste für den Graphen aus Abbildung 3.1 ist die folgende:

$$4, 6, 1, 2, 2, 3, 4, 3, 3, 2, 2, 2, 1, 3.$$

Bei der Bogenliste eines Digraphen verfahren wir genauso; wir müssen jedoch darauf achten, daß ein Bogen immer zunächst durch seinen Anfangs- und dann durch seinen Endknoten repräsentiert wird.

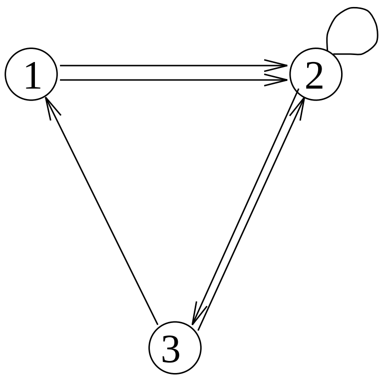

Abb. 3.2

Eine Bogenliste des Digraphen aus Abbildung 3.2 ist

$$3, 6, 1, 2, 2, 3, 3, 2, 2, 2, 1, 2, 3, 1.$$

Haben die Kanten oder Bögen Gewichte, so repräsentiert man eine Kante (einen Bogen) entweder durch Anfangsknoten, Endknoten, Gewicht oder macht eine Kanten- bzw. Bogenliste wie oben und hängt an diese noch eine Liste mit den m Gewichten der Kanten $1, 2, \ldots, m$ an.

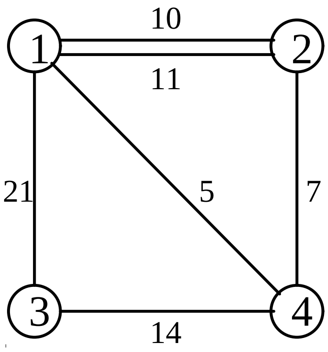

Abb. 3.3

Der gewichtete Graph aus Abbildung 3.3 ist in den beiden folgenden Kantenlisten mit Gewichten gespeichert:

$$4, 6, 1, 2, 10, 1, 2, 11, 2, 4, 7, 4, 3, 14, 3, 1, 21, 1, 4, 5$$

$$4, 6, 1, 2, 1, 2, 2, 4, 3, 4, 1, 4, 1, 3, 11, 10, 7, 14, 5, 21.$$

Der Speicheraufwand einer Kanten- bzw. Bogenliste beträgt $2(m+1)$ Speicherplätze, eine Liste mit Gewichten erfordert $3m+2$ Speicherplätze.

Adjazenzmatrizen

Ist $G = (V, E)$ ein ungerichteter Graph mit $V = \{1, 2, \ldots, n\}$, so ist die symmetrische (n, n)-Matrix $A = (a_{ij})$ mit

$$\begin{aligned} a_{ji} = a_{ij} &= \text{ Anzahl der Kanten, die } i \text{ und } j \text{ verbinden, falls } i \neq j \\ a_{ii} &= 2 \cdot (\text{Anzahl der Schleifen, die } i \text{ enthalten}),\ i = 1, \ldots, n \end{aligned}$$

die **Adjazenzmatrix** von G. Aufgrund der Symmetrie kann man etwa die Hälfte der Speicherplätze sparen. Hat ein Graph keine Schleifen (unser Normalfall), dann genügt es, die obere (oder untere) Dreiecksmatrix von A zu speichern. Man macht das z. B. in der Form

$$a_{12}, a_{13}, \ldots, a_{1n}, a_{23}, a_{24}, \ldots, a_{2n}, a_{34}, \ldots, a_{n-1,n}.$$

Die Adjazenzmatrix des Graphen in Abbildung 3.1 sieht wie folgt aus:

$$\begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 2 & 2 & 0 \\ 1 & 2 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

Hat ein Graph Kantengewichte, und ist er einfach, so setzt man

$$a_{ij} = \text{ Gewicht der Kante } ij,\ \ i, j = 1, \ldots, n.$$

Ist eine Kante ij nicht im Graphen G enthalten, so setzt man

$$a_{ij} = 0, \quad a_{ij} = -\infty \quad \text{oder} \quad a_{ij} = +\infty,$$

je nachdem, welche Gewichtszuordnung für das Problem sinnvoll ist und die Benutzung dieser Kante ausschließt. Wiederum braucht man von der so definierten Matrix A nur die obere Dreiecksmatrix zu speichern.

Die Adjazenzmatrix $A = (a_{ij})$ eines Digraphen $D = (V, E)$ mit $V = \{1, 2, \ldots, n\}$ ohne Schleifen ist definiert durch

$$\begin{aligned} a_{ii} &= 0, \quad i = 1, 2, \ldots, n \\ a_{ij} &= \text{Anzahl der Bögen in } E \text{ mit Anfangsknoten } i \text{ und Endknoten } j,\ i \neq j. \end{aligned}$$

Bogengewichte werden wie bei Adjazenzmatrizen von Graphen behandelt. Der Speicheraufwand von Adjazenzmatrizen beträgt n^2 bzw. $\binom{n}{2}$ Speicherplätze, falls bei einem ungerichteten Graphen nur die obere Dreiecksmatrix gespeichert wird.

Adjazenzlisten

Speichert man für einen Graphen $G = (V, E)$ die Anzahl der Knoten und für jeden Knoten $v \in V$ seinen Grad und die Namen der Nachbarknoten, so nennt man eine solche Datenstruktur **Adjazenzliste** von G. Für den Graphen aus Abbildung 3.1 sieht die Adjazenzliste wie folgt aus

$$\begin{array}{cccl} & \text{Knotennummer} & & \\ & \downarrow & & \\ 4 & 1 & 2 & 2,\ 3 \\ & 2 & 5 & 1,\ 3,\ 3,\ 2,\ 2 \\ & 3 & 4 & 1,\ 2,\ 2,\ 4 \\ & 4 & 1 & 3 \\ & & \uparrow & \\ & & \text{Grad} & \end{array}$$

Die Liste der Nachbarknoten eines Knoten v heißt **Nachbarliste** von v. Jede Kante ij, $i \neq j$, ist zweimal repräsentiert, einmal auf der Nachbarliste von i, einmal auf der von j. Bei Gewichten hat man wieder zwei Möglichkeiten. Entweder man schreibt direkt hinter jeden Knoten j auf der Nachbarliste von i das Gewicht der Kante ij, oder man legt eine kompatible Gewichtsliste an.

Bei Digraphen geht man analog vor, nur speichert man auf der Nachbarliste eines Knoten i nur die Nachfolger von i. Man nennt diese Liste daher auch **Nachfolgerliste**. Ein Bogen wird also nur einmal in der Adjazenzliste eines Digraphen repräsentiert. (Wenn es für das Problem günstiger ist, kann man natürlich auch Vorgängerlisten statt Nachfolgerlisten oder beides anlegen.)

Zur Speicherung einer Adjazenzliste eines Graphen braucht man $2n+1+2m$ Speicherplätze, für die Adjazenzliste eines Digraphen werden $2n + 1 + m$ Speicherplätze benötigt.

Kanten- bzw. Bogenlisten sind die kompaktesten aber unstrukturiertesten Speicherformen. Will man wissen, ob G die Kante ij enthält, muß man im schlechtesten Fall die gesamte Liste durchlaufen und somit $2m$ Abfragen durchführen. Eine derartige Frage benötigt bei der Speicherung von G in einer Adjazenzmatrix nur eine Abfrage, während man bei einer Adjazenzliste die Nachbarliste von i (oder j) durchlaufen muß und somit im schlechtesten Fall $n-1$ (bzw. Grad von i) Abfragen durchführen muß.

Für dünn besetzte Graphen ist die Speicherung in einer Adjazenzmatrix i. a. zu aufwendig. Es wird zu viel Platz vergeudet. Außerdem braucht fast jeder Algorithmus, der für Adjazenzlisten konzipiert ist, mindestens $O(n^2)$ Schritte, da er ja jedes Element von A mindestens einmal anschauen muß, um zu wissen, ob die zugehörige Kante im Graphen ist oder nicht. Mit Hilfe von Adjazenzlisten kann man dagegen dünn besetzte Graphen häufig sehr viel effizienter bearbeiten. Das Buch Aho, Hopcroft & Ullman (1974) informiert sehr ausführlich über dieses Thema.

Wir wollen hier als Beispiel nur einen einzigen einfachen, aber vielseitig und häufig anwendbaren Algorithmus zur Untersuchung von Graphen erwähnen: das Depth-First-Search-Verfahren (kurz DFS-Verfahren bzw. auf deutsch: Tiefensuche).

Wir nehmen an, daß ein Graph $G = (V, E)$ gegeben ist und alle Knoten unmarkiert sind. Alle Kanten seien unbenutzt. Wir wählen einen Startknoten, sagen wir v, und markieren ihn. Dann wählen wir eine Kante, die mit v inzidiert, sagen wir vw, gehen zu w und markieren w. Die Kante vw ist nun benutzt worden. Allgemein verfahren wir wie folgt. Ist x der letzte von uns markierte Knoten, dann versuchen wir eine mit x inzidente Kante xy zu finden, die noch nicht benutzt wurde. Ist y markiert, so suchen wir eine weitere mit x inzidente Kante, die noch nicht benutzt wurde. Ist y nicht markiert, dann gehen wir zu y, markieren y und beginnen von neuem (y ist nun der letzte markierte Knoten). Wenn die Suche nach Kanten, die mit y inzidieren und die noch nicht benutzt wurden, beendet ist (d. h. alle Kanten, auf denen y liegt, wurden einmal berührt), kehren wir zu x zurück und fahren mit der Suche nach unbenutzten Kanten, die mit x inzidieren fort, bis alle Kanten, die x enthalten, abgearbeitet sind. Diese Methode nennt man Tiefensuche, da man versucht, einen Knoten so schnell wie möglich zu verlassen und "tiefer" in den Graphen einzudringen.

Eine derartige Tiefensuche teilt die Kanten des Graphen in zwei Teilmengen auf. Eine Kante xy heißt **Vorwärtskante**, falls wir bei der Ausführung des Algorithmus von einem markierten Knoten x entlang xy zum Knoten y gegangen sind und y markiert haben. Andernfalls heißt xy **Rückwärtskante**. Man überlegt sich leicht, daß die Menge der Vorwärtskanten ein Wald von G ist, der in jeder Komponente von G einen aufspannenden Baum bildet. Ist der Graph zusammenhängend, so nennt man die Menge der Vorwärtskanten **DFS-Baum** von G. Mit Hilfe von Adjazenzlisten kann die Tiefensuche sehr effizient rekursiv implementiert werden.

(3.7) Depth-First-Search.

Input: Graph $G = (V, E)$ in Form einer Adjazenzliste, d. h. für jeden Knoten $v \in V$ ist eine Nachbarliste $N(v)$ gegeben.

Output: Kantenmenge T (= DFS-Baum, falls G zusammenhängend ist)

Alle Knoten $v \in V$ seien unmarkiert.

1. Setze $T := \emptyset$.
2. Für alle $v \in V$ führe aus:
   Ist v unmarkiert, dann CALL SEARCH(v).
   END.
3. Gib T aus.

Rekursives Unterprogramm

PROCEDURE SEARCH(v)

1. Markiere v.
2. Für alle Knoten $w \in N(v)$ führe aus:
   3. Ist w unmarkiert, setze $T := T \cup \{vw\}$ und CALL SEARCH(w).

   END.

END SEARCH.

□

In Algorithmus (3.7) wird im Hauptprogramm jeder Knoten einmal berührt, und im Unterprogramm jede Kante genau zweimal. Hinzu kommt die Ausgabe von T. Die Laufzeit des Verfahrens ist also $O(|V|+|E|)$. Diese Laufzeit könnte man niemals bei der Speicherung von G in einer Adjazenzmatrix erreichen.

Mit Hilfe des obigen Verfahrens können wir unser mehrmals zitiertes Problem "Enthält G einen Kreis?" lösen. Offensichtlich gilt: G enthält genau dann einen Kreis, wenn $E \setminus T \neq \emptyset$ ist. Wir haben somit einen polynomialen Algorithmus zur Lösung des Kreisproblems gefunden. Der DFS-Baum, von (3.7) produziert, hat einige interessante Eigenschaften, die man dazu benutzen kann, eine ganze Reihe von weiteren Graphenproblemen sehr effizient zu lösen. Der hieran interessierte Leser sei z. B. auf das Buch Aho, Hopcroft & Ullman (1974) verwiesen.

Literaturverzeichnis von Kapitel 3

A. V. Aho, J. E. Hopcroft und J. D. Ullman (1974), *The Design and Analysis of Computer Algorithms,* Addison-Wesley, Reading.

T. H. Cormen, C. E. Leiserson und R. L. Rivest (1990), *Introduction to Algorithms,* MIT Press, Cambridge, Mass., (wurde mehrfach neu aufgelegt).

S. A. Cook (1971), *The complexity of theorem proving procedures, in: Proceedings of the Third Annual ACM Symposium on Theory of Computing (Shaker Heights, Ohio, 1971),* ACM, New York, 151-158.

M. G. Garey und D. S. Johnson (1979), *Computers and Intractability: A Guide to the Theory of $\mathcal{NP}$-completeness,* Freeman, San Francisco.

K. Mehlhorn (1984), *Data Structures and Algorithms, Vol. 1-3,* EATCS Monographie, Springer-Verlag, 1984, (dreibändige Monographie, Band I liegt auch auf deutsch im Teubner-Verlag (1986) vor).

Ch. Meinel (1991), *Effiziente Algorithmen,* Fachbuchverlag, Leipzig.

T. Ottmann und P. Widmayer (1990), *Algorithmen und Datenstrukturen,* BI Wissenschaftsverlag, Mannheim.

C. H. Papadimitriou (1994), *Computational Complexity,* Addison-Wesley, Amsterdam.

D. B. Shmoys und É. Tardos (1995), "Computational Complexity", R. L. Graham et al. (Hrsg.) *Handbook of Combinatorics*, Kap. 29 , 1599-1645, North-Holland, Amsterdam.

R. E. Tarjan (1983), *Data Structures and Network Algorithms,* SIAM, Philadelphia.

J. van Leeuwen (1990), *Algorithms and Complexity,* Handbook of Theoretical Computer Science, Vol. A, Elsevier, Amsterdam.

K. Wagner und G. Wechsung (1986), *Computational Complexity*, VEB Deutscher Verlag der Wissenschaften, Berlin.

Kapitel 4

Minimale Bäume, maximale Branchings

4.1 Bäume und Wälder

Das Problem, in einem Graphen mit Kantengewichten einen aufspannenden Baum minimalen Gewichts oder einen Wald maximalen Gewichts zu finden, haben wir bereits in (2.12) eingeführt. Beide Probleme sind sehr effizient lösbar und haben vielfältige Anwendungen. Einen umfassenden Überblick über die Geschichte dieser Probleme, ihre Anwendungen und die bekannten Lösungsverfahren gibt der Aufsatz Graham und Hell (1985).

Wir wollen hier jedoch nur einige dieser Lösungsmethoden besprechen. Zunächst wollen wir uns überlegen, daß die beiden Probleme auf sehr direkte Weise äquivalent sind.

Angenommen wir haben einen Algorithmus zur Lösung eines Maximalwald-Problems, und wir wollen in einem Graphen $G = (V, E)$ mit Kantengewichten c_e, $e \in E$, einen minimalen aufspannenden Baum finden, dann gehen wir wie folgt vor. Wir setzen

$$\begin{aligned} M &:= \max\{|c_e| \mid e \in E\} + 1, \\ c'_e &:= M - c_e \end{aligned}$$

und bestimmen einen maximalen Wald W in G bezüglich der Gewichtsfunktion c'. Falls G zusammenhängend ist, ist W ein aufspannender Baum, denn andernfalls gäbe es eine Kante $e \in E$, so daß $W' := W \cup \{e\}$ ein Wald ist, und wegen $c'_e > 0$, wäre $c'(W') > c'(W)$. Aus der Definition von c' folgt direkt, daß W ein minimaler aufspannender Baum von G bezüglich c ist. Ist W nicht zusammenhängend, so ist auch G nicht zusammenhängend, also existiert kein aufspannender Baum.

Haben wir einen Algorithmus, der einen minimalen aufspannenden Baum in einem Graphen findet, und wollen wir einen maximalen Wald in einem Graphen $G = (V, E)$ mit Kantengewichten c_e, $e \in E$, bestimmen, so sei $K_n = (V, E_n)$ der vollständige Graph mit $n = |V|$ Knoten und folgenden Kantengewichten

$$\begin{aligned} c'_e &:= -c_e && \text{für alle } e \in E \text{ mit } c_e > 0, \\ c'_e &:= M && \text{für alle } e \in E_n \setminus \{e \in E \mid c_e > 0\}, \end{aligned}$$

wobei wir z. B. setzen

$$M := n \cdot (\max\{|c_e| \mid e \in E\} + 1)\,.$$

Ist B ein minimaler aufspannender Baum von K_n bezüglich der Kantengewichte c', dann ist offenbar aufgrund unserer Konstruktion $W := B \setminus \{e \in B \mid c'_e = M\}$ ein Wald in G maximalen Gewichts.

Der folgende sehr einfache Algorithmus findet einen maximalen Wald.

(4.1) GREEDY-MAX.

Input: *Graph $G = (V, E)$ mit Kantengewichten $c(e)$ für alle $e \in E$.*

Output: *Wald $W \subseteq E$ mit maximalem Gewicht $c(W)$.*

1. *(Sortieren): Ist k die Anzahl der Kanten von G mit positivem Gewicht, so numeriere diese k Kanten, so daß gilt $c(e_1) \geq c(e_2) \geq \ldots \geq c(e_k) > 0$.*
2. *Setze $W := \emptyset$.*
3. FOR $i = 1$ TO k DO:
   Falls $W \cup \{e_i\}$ keinen Kreis enthält, setze $W := W \cup \{e_i\}$.
4. *Gib W aus.*

$\square$

(4.2) Satz. *Der Algorithmus GREEDY-MAX arbeitet korrekt.*

Beweis. Hausaufgabe! $\square$

Versuchen Sie, einen direkten Beweis für die Korrektheit von Algorithmus (4.1) zu finden. Im nachfolgenden Teil dieses Abschnitts und in Kapitel 5 werden wir Sätze angeben, aus denen Satz (4.2) folgt.

Der obige Algorithmus heißt "Greedy-Max" ("greedy" bedeutet "gierig" oder "gefräßig"), weil er versucht, das bezüglich der Zielfunktionskoeffizienten jeweils "Beste" zu nehmen, das im Augenblick noch verfügbar ist. Wir werden später noch andere Algorithmen vom Greedy-Typ kennenlernen, die bezüglich anderer Kriterien das "Beste" wählen. Der Greedy-Algorithmus funktioniert auf analoge Weise für das Minimum Spanning Tree Problem.

(4.3) GREEDY-MIN.

Input: *Graph $G = (V, E)$ mit Kantengewichten $c(e)$ für alle $e \in E$.*

Output: *Maximaler Wald $T \subseteq E$ mit minimalem Gewicht $c(T)$.*

1. *(Sortieren): Numeriere die m Kanten des Graphen G, so daß gilt $c(e_1) \leq c(e_2) \leq \ldots \leq c(e_m)$.*
2. *Setze $T := \emptyset$.*
3. FOR $i = 1$ TO m DO:
   Falls $T \cup \{e_i\}$ keinen Kreis enthält, setze $T := T \cup \{e_i\}$.
4. *Gib T aus.*

$\square$

Aus Satz (4.2) und unseren Überlegungen zur Reduktion des Waldproblems auf das Baumproblem und umgekehrt folgt.

(4.4) Satz. *Algorithmus (4.3) liefert einen maximalen Wald T (d. h. für jede Zusammenhangskomponente $G' = (V', E')$ von G ist $T \cap E'$ ein aufspannender Baum), dessen*

Gewicht $c(T)$ minimal ist. Ist G zusammenhängend, so ist T ein aufspannender Baum von G minimalen Gewichts $c(T)$.

□

Die Laufzeit von Algorithmus (4.1) bzw. (4.3) kann man wie folgt abschätzen. Mit den gängigen Sortierverfahren der Informatik (z. B. HEAP-SORT) kann man die Kanten von E in $O(k \log_2 k)$ bzw. $O(m \log_2 m)$ Schritten in der geforderten Weise ordnen. In Schritt 3 ruft man k- bzw. m-mal ein Unterprogramm auf, das überprüft, ob eine Kantenmenge einen Kreis besitzt oder nicht. Durch Benutzung geeigneter Datenstrukturen kann man einen derartigen Aufruf in höchstens $O(n)$ Schritten abarbeiten. Daraus folgt, daß Schritt 3 in höchstens $O(mn)$ Schritten ausgeführt werden kann. Dies ist auch die Gesamtlaufzeit des Verfahrens. Mit speziellen "Implementierungstricks" kann die Gesamtlaufzeit sogar auf $O(m \log m)$ Schritte reduziert werden. In der Literatur wird Algorithmus (4.3) häufig **Kruskal-Algorithmus** genannt.

Einen gewichtsminimalen aufspannenden Baum kann man übrigens auch mit folgendem Verfahren finden.

(4.5) "Dualer" Greedy-Algorithmus.

Input: *Zusammenhängender Graph $G = (V, E)$ mit Kantengewichten $c(e)$ für alle $e \in E$.*

Output: *Aufspannender Baum $T \subseteq E$ minimalen Gewichts $c(T)$.*

1. *(Sortieren): Numeriere die m Kanten des Graphen G, so daß gilt $c(e_1) \geq c(e_2) \geq \ldots \geq c(e_m)$.*
2. *Setze $T := E$.*
3. FOR $i = 1$ TO m DO*:*
   Falls $T \setminus \{e_i\}$ zusammenhängend ist, setze $T := T \setminus \{e_i\}$.
4. *Gib T aus.*

□

Der Korrektheitsbeweis für Algorithmus (4.5) bleibt einer Übungsaufgabe überlassen.

Wie bereits erwähnt, gibt es eine Vielzahl weiterer Verfahren zur Bestimmung minimaler aufspannender Bäume. Ein gemeinsames Skelett für mehrere dieser Algorithmen kann wie folgt skizziert werden.

(4.6) Algorithmus.

Input: *Zusammenhängender Graph $G = (V, E)$ mit Kantengewichten $c(e)$ für alle $e \in E$.*

Output: *Aufspannender Baum T von G minimalen Gewichts.*

1. *(Initialisierung):*
   FOR ALL $i \in V$ DO*:*
   Setze $V_i := \{i\}$ und $T_i := \emptyset$.
2. DO $|V| - 1$ TIMES*:*
   (a) Wähle eine nicht-leere Menge V_i.

(b) *Wähle eine Kante $uv \in E$ mit $u \in V_i$, $v \in V \setminus V_i$ und $c(uv) \leq c(pq)$ für alle $pq \in E$ mit $p \in V_i$, $q \in V \setminus V_i$.*
(c) *Bestimme j so daß $v \in V_j$.*
(d) *Setze $V_i := V_i \cup V_j$; $V_j := \emptyset$.*
(e) *Setze $T_i := T_i \cup T_j \cup \{uv\}$; $T_j := \emptyset$.*

3. *Gib diejenige Kantenmenge T_i mit $T_i \neq \emptyset$ aus.*

□

Algorithmus (4.3) ist ein Spezialfall von Algorithmus (4.6). Überlegen Sie sich wieso!

(4.7) Satz. *Algorithmus (4.6) bestimmt einen aufspannenden Baum minimalen Gewichts.*

Beweis. Wir zeigen durch Induktion über $p = |T_1| + \ldots + |T_n|$ daß G einen minimalen aufspannenden Baum T enthält mit $T_i \subseteq T$ für alle i. Ist $p = 0$, so ist nichts zu zeigen. Sei uv eine Kante, die bei einem Durchlauf von Schritt 2 in (b) gewählt wurde. Nach Induktionsvoraussetzung sind alle bisher bestimmten Mengen T_i in einem minimalen aufspannenden Baum T enthalten. Gilt $uv \in T$, so sind wir fertig. Ist $uv \notin T$, so enthält $T \cup \{uv\}$ einen Kreis. Folglich muß es eine Kante $rs \in T$ geben mit $r \in V_i$, $s \in V \setminus V_i$. Aufgrund unserer Wahl in (b) gilt $c(uv) \leq c(rs)$. Also ist $T := (T \setminus \{rs\}) \cup \{uv\}$ ebenfalls ein minimaler aufspannender Baum und der neue Baum T erfüllt unsere Bedingungen. Die Korrektheit des Algorithmus folgt aus dem Fall $p = n - 1$. □

Die Laufzeit des Algorithmus (4.6) hängt natürlich sehr stark von den Datenstrukturen ab, die man zur Ausführung des Schrittes 2 implementiert. Wir können an dieser Stelle nicht ausführlich auf Implementierungstechniken eingehen und verweisen hierzu auf Mehlhorn (1984), Vol 2, Kapitel IV, Abschnitt 8. Hier wird gezeigt, daß bei geeigneten Datenstrukturen eine Laufzeit von $O(n \log\log m)$ Schritten erreicht werden kann. Für planare Graphen ergibt sich sogar eine $O(n)$-Laufzeit.

Spanning-Tree-Algorithmen werden häufig als Unterprogramme zur Lösung von Travelling Salesman Problemen benötigt. Speziell ist hier eine Implementation dieser Algorithmen für vollständige Graphen erforderlich. Der nachfolgende Algorithmus läßt sich gerade für diesen Fall vollständiger Graphen einfach implementieren und hat sowohl empirisch wie theoretisch günstige Rechenzeiten aufzuweisen. Dieses Verfahren, das offensichtlich ebenfalls eine Spezialisierung von (4.6) ist, wird häufig PRIM-Algorithmus genannt.

(4.8) PRIM-Algorithmus.

Input: *Zusammenhängender Graph $G = (V, E)$ mit Kantengewichten $c(e)$ für alle $e \in E$.*

Output: *Aufspannender Baum T minimalen Gewichts $c(T)$.*

1. *Wähle $w \in V$ beliebig, setze $T := \emptyset$, $W := \{w\}$, $V := V \setminus \{w\}$.*
2. *Ist $V = \emptyset$, dann gib T aus und STOP.*
3. *Wähle eine Kante uv mit $u \in W$, $v \in V$ so daß $c(uv) = \min\{c(e) \mid e \in \delta(W)\}$.*

4. Setze

$$T := T \cup \{uv\}$$
$$W := W \cup \{v\}$$
$$V := V \setminus \{v\}$$

und gehe zu 2.

□

Das PRIM-Verfahren kann für den vollständigen Graphen K_n so implementiert werden, daß seine Laufzeit $O(n^2)$ beträgt, was offenbar bezüglich der Ordnung (also bis auf Multiplikation mit Konstanten und lineare Terme) bestmöglich ist, da ja jede der $\frac{n(n-1)}{2}$ Kanten mindestens einmal überprüft werden muß. Bereits bei dieser Überprüfung sind $O(n^2)$ Schritte notwendig. Nachfolgend finden Sie eine Liste eines PASCAL-Programms für Algorithmus (4.8), das für die Bestimmung minimaler aufspannender Bäume im K_n konzipiert ist.

(4.9) PASCAL-Implementierung von Algorithmus (4.8).

```
PROGRAM prim(inp, outp);

/**********************************************************************
 *                                                                    *
 *        Prim's Algorithm to Determine a Minimum Spanning Tree       *
 *        in a Complete Graph With n Nodes                            *
 *                                                                    *
 *        (G. Reinelt)                                       *
 *                                                                    *
 *--------------------------------------------------------------------*
 *                                                                    *
 *  Input:                                                            *
 *                                                                    *
 *  There are four ways to input the edge weights of the complete     *
 *  graph K_n. In any case we assume that first two numbers are given:   *
 *                                                                    *
 *      n = number of nodes                                           *
 *   mode = input mode                                                *
 *                                                                    *
 *  Mode specifies the input mode for the edge weights. All edge weights  *
 *  have to be integers.                                              *
 *                                                                    *
 *  Mode=0 : The full matrix of edge weights is given. The entries are    *
 *           stored row by row. The lower diagonal and the diagonal   *
 *           entries are ignored.                                     *
 *                                                                    *
 *  Mode=1 : The matrix of edge weights is given as upper triangular  *
 *           matrix. The entries are stored row by row.               *
 *                                                                    *
 *  Mode=2 : The matrix of edge weights is given as lower triangular  *
 *           matrix. The entries are stored row by row.               *
 *                                                                    *
 *  Mode=3 : The edge weights are given in an edge list of the        *
 *           form: 1st endnode, 2nd endnode, edge weight. Edges which *
 *           are not present are assumed to have 'infinite' weight.   *
 *           The input is ended if the first endnode is less than 1.  *
 *                                                                    *
```

```
 ************************************************************************/

CONST  max_n  = 100;            { maximum number of nodes }
       max_n2 = 4950;           { max_n choose 2 = number of edges of K_n}
                                { to process larger graphs only max_n and
                                  max-n2 have to be changed }
       inf    = maxint;         { infinity }

TYPE   arrn2 = ARRAY[1..max_n2] OF integer;
       arrn  = ARRAY[1..max_n]  OF integer;

VAR    i, j,
       mode,                    { input mode of weights:
                                    0  : full matrix
                                    1  : upper triangular matrix
                                    2  : lower triangular matrix
                                    3  : edge list              }
       min,                     { minimum distance }
       ind,                     { index of entering edge }
       newnode,                 { entering tree node }
       t1, t2,                  { entering tree edge }
       outnodes,                { number of nodes not in tree }
       c,
       weight,                  { weight of tree }
       nchoose2,
       n         : integer;     { number of nodes }
       w         : arrn2;       { vector of weights }
       dope,                    { dope vector for index calculations }
       dist,                    { shortest distances to non-tree nodes }
       in_t,                    { in-tree node of shortest edge }
       out_t     : arrn;        { out-tree node of shortest edge }
                                { minimum tree is also stored in in_t & out_t}
       connected : boolean;     { true <=> input graph is connected? }
       inp,                     { input file }
       outp      : text;        { output file }

BEGIN {MAIN PROGRAM}

{====== Input of complete graph =====}

reset(inp);
rewrite(outp);

{- number of nodes -}
writeln(outp,'Enter number of nodes:');
read(inp,n);

IF (n<1) OR (n>max_n) THEN
 BEGIN
   writeln(outp,'Number of nodes too large or not positive!');
   HALT;
 END;

{- initialize dope vector -}
nchoose2 := (n * (n-1)) DIV 2;
FOR i:=1 TO nchoose2 DO
  w[i] := inf;
dope[1] := -1;
FOR i:=2 TO n DO
  dope[i] := dope[i-1] + n - i;
```

```
{- input mode -}
writeln(outp,'Enter input mode:');
read(inp,mode);

{- edge weights -}
CASE mode OF
   0 : {* full matrix *}
       BEGIN
         FOR i:=1 TO n DO
           FOR j:=1 TO n DO
             BEGIN
               read(inp, c);
               IF i<j THEN w[dope[i]+j] := c;
             END
       END;
   1 : {* upper triangular matrix *}
       BEGIN
         FOR i:=1 TO nchoose2 DO
           read(inp, w[i]);
       END;
   2 : {* lower triangular matrix *}
       BEGIN
         FOR i:=2 TO n DO
           FOR j:=1 TO i-1 DO
             BEGIN
               read(inp, c);
               w[dope[j]+i] := c;
             END;
       END;
   3 : {* edge list *}
       BEGIN
         read(inp, i, j, c);
         WHILE (i>0) DO
          BEGIN
            IF (i<1) OR (i>n) OR
               (j<1) OR (j>n)
             THEN BEGIN
                    writeln(outp,'Input error, node out of range!');
                    HALT;
                  END;
            IF i<j
              THEN w[dope[i]+j] := c
              ELSE w[dope[j]+i] := c;
            read(inp, i, j, c);
          END;
       END;
 ELSE: {* invalid mode *}
       BEGIN
         writeln(outp,'Invalid input mode!');
         HALT;
       END;
  END;{OF CASE}

{====== Initialization ======}

connected := true;
outnodes  := n-1;
weight    := 0;
FOR i:=1 TO outnodes DO
 BEGIN
   in_t[i]  := 1;
   out_t[i] := i+1;
```

```
   dist[i]  := w[i];
 END;

{====== Prim's Algorithm ======}

WHILE (outnodes > 1) AND connected DO
 BEGIN
   {- determine entering node -}
   min := inf;
   ind := 0;
   FOR i:=1 TO outnodes DO
     IF dist[i] < min
      THEN BEGIN
             min := dist[i];
             ind := i;
           END;

   IF ind = 0
    THEN connected := false
    ELSE BEGIN
           {- augment tree -}
           weight          := weight + min;
           newnode         := out_t[ind];
           t1              := in_t[ind];
           t2              := out_t[ind];
           c               := dist[ind];
           in_t[ind]       := in_t[outnodes];
           out_t[ind]      := out_t[outnodes];
           dist[ind]       := dist[outnodes];
           in_t[outnodes]  := t1;
           out_t[outnodes] := t2;
           dist[outnodes]  := c;
           outnodes        := outnodes - 1;
           {- update dist[] and in_t[] -}
           FOR i:=1 TO outnodes DO
            BEGIN
              IF newnode < out_t[i]
                THEN c := w[dope[newnode]+out_t[i]]
                ELSE c := w[dope[out_t[i]]+newnode];
              IF c < dist[i]
              THEN BEGIN
                     in_t[i] := newnode;
                     dist[i] := c;
                   END;
            END;
         END;
 END;
{- insert the last edge -}
IF connected
 THEN IF dist[1]>=inf
       THEN connected := false
       ELSE weight    := weight + dist[1];

{====== Output of minimum spanning tree ======}

writeln(outp);
IF NOT connected
 THEN writeln(outp,'The graph is disconnected.')
 ELSE BEGIN
        writeln(outp,'Minimum spanning tree:');
        writeln(outp,'=====================');
        writeln(outp);
```

```
    FOR i:=n-1 DOWNTO 1 DO
      writeln(outp, in_t[i]:5, ' - ', out_t[i]:3,
                '  (', dist[i]:1,')');
    writeln(outp);
    writeln(outp,'Weight: ', weight:6);
    writeln(outp);
  END;
END.
```

Wir wollen nun noch ein Beispiel angeben, das die Vorgehensweise der Algorithmen (4.3), (4.5) und (4.8) verdeutlicht.

(4.10) Beispiel. Wir betrachten den in Abbildung 4.1 dargestellten Graphen.

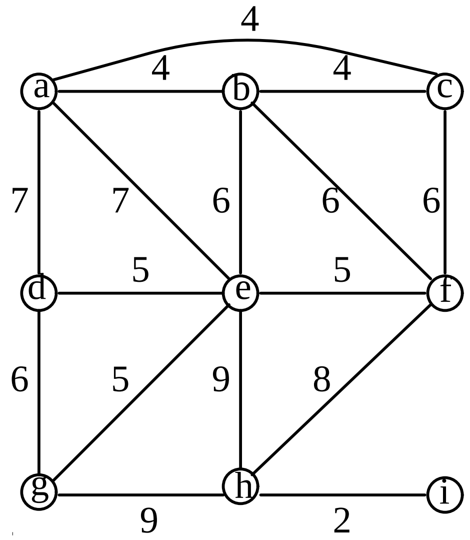

Abb. 4.1

Wir wenden Algorithmus (4.3) an. Zunächst sortieren wir die Kanten in nicht absteigender Reihenfolge *hi*, *bc*, *ab*, *ac*, *de*, *ef*, *eg*, *be*, *bf*, *cf*, *dg*, *ad*, *ae*, *hf*, *he*, *hg*. In Schritt 3 von (4.3) werden die in der Abbildung 4.2 gezeichneten Kanten ausgewählt.

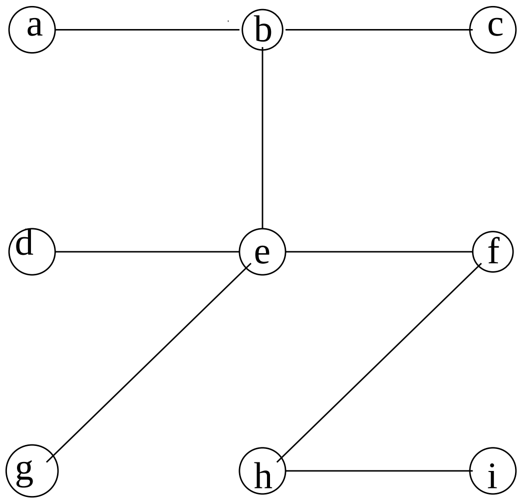

Abb. 4.2

Den Prim-Algorithmus (4.8) starten wir mit dem Knoten $w = a$. Es ergibt sich der in Abbildung 4.3 gezeichnete minimale aufspannende Baum. □

Wie Beispiel (4.10) zeigt, muß ein minimaler Baum nicht eindeutig bestimmt sein. Überlegen Sie sich bitte, wie man feststellen kann, ob ein minimaler aufspannender Baum eindeutig ist.

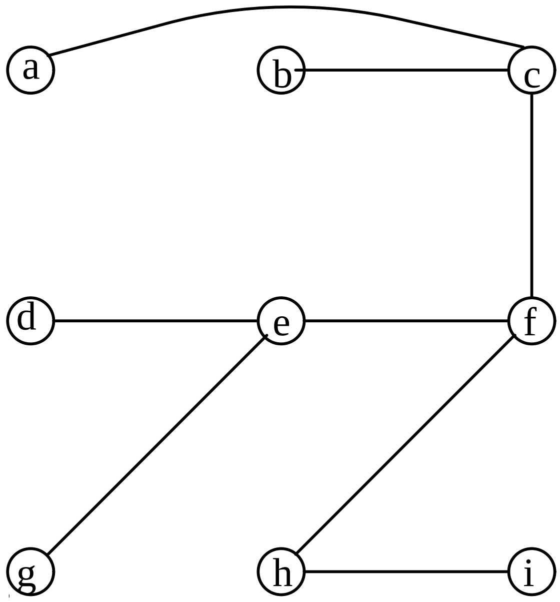

Abb. 4.3

4.2 Branchings und Arboreszenzen

Die Aufgaben, in einem Digraphen $D = (V, A)$ mit Bogengewichten $c_{ij}\ \forall\ (i, j) \in A$ ein Branching mit maximalem Gewicht bzw. eine Arboreszenz minimalen Gewichts zu finden, sind trivialerweise äquivalent. Wir wollen nun zeigen, daß man ein maximales Branching in polynomialer Zeit konstruieren kann. Der Algorithmus hierzu wurde unabhängig voneinander von Bock (1971), Chu & Liu (1965) und Edmonds (1967) entdeckt. Er ist erheblich komplizierter als die Algorithmen zur Bestimmung maximaler Wälder bzw. minimaler aufspannender Bäume. Insbesondere der Korrektheitsbeweis erfordert einigen Aufwand. Wir wollen bei der Beschreibung des Algorithmus die Inzidenzfunktion $\Psi : A \to V \times V$ eines Digraphen benutzen, da mit ihrer Hilfe die Technik des Schrumpfens von Knotenmengen etwas klarer beschrieben werden kann. Falls $a \in A$, schreiben wir $\Psi(a) := (t(a), h(a))$ um den Anfangs- und den Endknoten von a anzugeben.

(4.11) Der Branching-Algorithmus.

Input: *Ein schlingenfreier Digraph* $D = (V, A)$ *mit Bogengewichten* $c(a)$ *für alle* $a \in A$. $\Psi = (t, h) : A \to V \times V$ *sei die Inzidenzfunktion von* D.

Output: *Ein Branching* B, *dessen Gewicht maximal ist.*

1. *(Initialisierung):*
   Setze $D_0 := D$, $V_0 := V$, $A_0 := A$, $\Psi_0 := (t_0, h_0) = \Psi = (t, h)$, $B_0 := \emptyset$,
   $c_0 := c$, $i := 0$.
   Alle Knoten aus V_0 *seien unmarkiert.*

Phase I *(Greedy-Auswahl und Schrumpfung von Kreisen):*

2. *Sind alle Knoten aus* V_i *markiert, gehe zu 11.*
3. *Wähle einen unmarkierten Knoten* $v \in V_i$.
4. *Gilt* $c_i(a) \leq 0$ *für alle* $a \in \delta^-(v) \subseteq A_i$, *markiere* v *und gehe zu 2.*
5. *Unter allen Bögen aus* $\delta^-(v) \subseteq A_i$, *wähle einen Bogen* b *mit maximalem (positiven) Gewicht* $c_i(b)$.
6. *Ist* $B_i \cup \{b\}$ *ein Branching, setze* $B_i := B_i \cup \{b\}$, *markiere* v *und gehe zu 2.*
7. *Ist* $B_i \cup \{b\}$ *kein Branching, dann bestimme den gerichteten Kreis, sagen wir* C_i, *der von* b *und einigen Bögen aus* B_i *gebildet wird.* W_i *sei die Knotenmenge von* C_i.
8. *Schrumpfe die Knotenmenge* W_i, *d. h. ersetze die Knotenmenge* W_i *durch einen neuen Knoten* w_i *(genannt Pseudoknoten) und definiere einen neuen Digraphen* $D_{i+1} := (V_{i+1}, A_{i+1})$ *mit Inzidenzfunktion* Ψ_{i+1} *und Gewichten* c_{i+1} *wie folgt:*

$$\begin{aligned} V_{i+1} &:= (V_i \setminus W_i) \cup \{w_i\} \\ A_{i+1} &:= A_i \setminus \{a \in A_i \mid t_i(a), h_i(a) \in W_i\} \end{aligned}$$

$$\left.\begin{aligned} \Psi_{i+1}(a) &:= (w_i, h_i(a)) \\ c_{i+1}(a) &:= c_i(a) \end{aligned}\right\} \quad \textit{falls} \quad t_i(a) \in W_i \quad \textit{und} \quad h_i(a) \notin W_i$$

$$\left.\begin{aligned} \Psi_{i+1}(a) &:= \Psi_i(a) \\ c_{i+1}(a) &:= c_i(a) \end{aligned}\right\} \quad \textit{falls} \quad t_i(a) \notin W_i \quad \textit{und} \quad h_i(a) \notin W_i$$

$$\left.\begin{array}{r} \Psi_{i+1}(a) := (t_i(a), w_i) \\ c_{i+1}(a) := c_i(a) + c_i(b_i) - c_i(a_i) \end{array}\right\} \quad \textit{falls} \quad t_i(a) \notin W_i \quad \textit{und} \quad h_i(a) \in W_i$$

wobei in der letzten Zuweisung b_i ein Bogen des Kreises C_i mit $c(b_i) = \min\{c(b) \mid b \in C_i\}$ ist und a_i der Bogen des Kreises C_i ist, der $h_i(a)$ als Endknoten hat.

9. *Alle Knoten in $V_{i+1} \setminus \{w_i\}$ behalten die Markierung, die sie in D_i hatten. Der Pseudoknoten w_i sei unmarkiert.*

10. *Setze $B_{i+1} := B_i \setminus C_i$, $i := i + 1$ und gehe zu 2.*

Phase II *(Rekonstruktion eines maximalen Branchings):*
(Bei Beginn von Phase II sind alle Knoten des gegenwärtigen Digraphen D_i markiert, und die Bogenmenge B_i ist ein Branching in D_i mit maximalem Gewicht $c_i(B_i)$.)

11. *Falls $i = 0$, STOP! (Das Branching B_0 ist ein optimales Branching des gegebenen Digraphen $D = D_0$)*

12. *Falls der Pseudoknoten w_{i-1} von D_i die Wurzel einer Arboreszenz des Branchings B_i ist, dann setze*

$$B_{i-1} := B_i \cup (C_{i-1} \setminus \{b_{i-1}\}),$$

wobei b_{i-1} ein Bogen des Kreises C_{i-1} ist, der unter den Bögen aus C_{i-1} minimales Gewicht $c_{i-1}(b_{i-1})$ hat.

Setze $i := i - 1$ und gehe zu 11.

13. *Gibt es einen Bogen b in B_i mit $h_i(b) = w_{i-1}$ (d. h. ist w_{i-1} keine Wurzel), dann bestimme den Bogen $a_{i-1} \in C_{i-1}$ mit $h_{i-1}(b) = h_{i-1}(a_{i-1})$ und setze*

$$B_{i-1} := B_i \cup (C_{i-1} \setminus \{a_{i-1}\}).$$

Setze $i := i - 1$ und gehe zu 11.

□

Bevor wir die Korrektheit des Algorithmus beweisen, besprechen wir ein Beispiel.

(4.12) Beispiel. Wir betrachten den in Abbildung 4.4 dargestellten Digraphen $D = (V, A)$ mit Knotenmenge $1, \ldots, 6$ und den eingetragenen Bogengewichten.

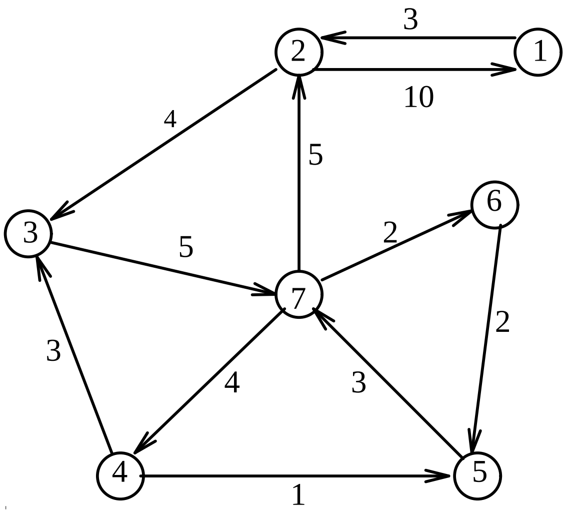

Abb. 4.4

Wir durchlaufen nun die einzelnen Schritte von Algorithmus (4.11). Die Bögen bezeichnen wir nicht mit einem Namen, sondern mit ihrem jeweiligen Anfangs- und Endknoten. In Schritt 1 initialisieren wir wie angegeben.

2. – ("–" heißt, die angegebene Bedingung ist nicht erfüllt, und wir machen nichts.)
3. Wir wählen Knoten 1.
4. –
5. $b = (2, 1)$.
6. $B_0 := \{(2, 1)\}$, wir markieren 1 und gehen zu 2.
2. –
3. Wir wählen Knoten 2.
4. –
5. $b = (7, 2)$.
6. $B_0 := \{(2, 1), (7, 2)\}$, wir markieren 2.
2. –
3. Wir wählen Knoten 3.
4. –
5. $b = (2, 3)$.
6. $B_0 := \{(2, 1), (7, 2), (2, 3)\}$, wir markieren 3.
2. –
3. Wir wählen Knoten 7.
4. –
5. $b = (3, 7)$.
6. –
7. $B_0 \cup \{(3, 7)\}$ enthält den gerichteten Kreis $C_0 = (3, 7, 2)$; es ist $W_0 = \{2, 3, 7\}$.
8. Der neue Digraph $D_1 = (V_1, A_1)$ mit $V_1 = \{w_0, 1, 4, 5, 6\}$ sieht wie in Abb. 4.5 gezeigt aus und hat die in Abb. 4.5 angegebenen Gewichte c_1. Gewichtsänderungen ergeben

sich nur bei Bögen mit Anfangsknoten in $V_0 \setminus W_0$ und Endknoten in W_0, also

$$\begin{aligned} c_1((1,w_0)) &:= c_0((1,2)) + c_0((2,3)) - c_0((7,2)) = 3+4-5 = 2,\\ c_1((4,w_0)) &:= c_0((4,3)) + c_0((2,3)) - c_0((2,3)) = 3+4-4 = 3,\\ c_1((5,w_0)) &:= c_0((5,7)) + c_0((2,3)) - c_0((3,7)) = 3+4-5 = 2. \end{aligned}$$

Markierungen von Knoten sind durch ein $\times$ angedeutet.

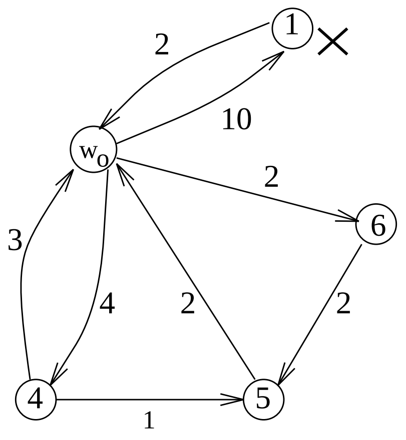

Abb. 4.5

9. In D_1 ist nur Knoten 1 markiert.
10. $B_1 = \{(w_0, 1)\}$.
2. –
3. Wir wählen Knoten 4.
4. –
5. $b = (w_0, 4)$.
6. $B_1 = \{(w_0,1),(w_0,4)\}$, wir markieren 4.
2. –
3. Wir wählen Knoten w_0.
4. –
5. $b = (4, w_0)$.
6. –
7. $B_1 \cup \{(4,w_0)\}$ enthält den Kreis $C_1 = \{(4,w_0)(w_0,4)\}$ mit $W_1 = \{4, w_0\}$.
8. Der neue Digraph $D_2 = (V_2, A_2)$ hat die Knotenmenge $V_2 = \{w_1, 1, 5, 6\}$ und ist in Abbildung 4.6 mit seinen Bogengewichten und Markierungen dargestellt. Die Bogengewichtsänderungen ergeben sich aus:

$$\begin{aligned} c_2((1,w_1)) &:= c_1((1,w_0)) + c_1((4,w_0)) - c_1((4,w_0)) = 2-3+3 = 2,\\ c_2((5,w_1)) &:= c_1((5,w_0)) + c_1((4,w_0)) - c_1((4,w_0)) = 2-3+3 = 2. \end{aligned}$$

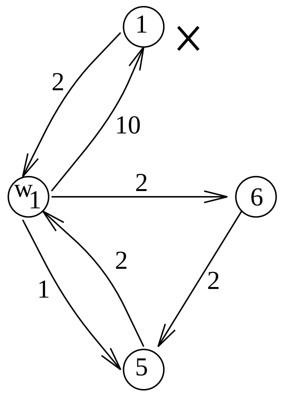

Abb. 4.6

9. In D_2 ist Knoten 1 markiert.
10. $B_2 = \{(w_1, 1)\}$.
2. –
3. Wir wählen Knoten 5.
4. –
5. $b = (6, 5)$.
6. $B_2 := \{(w_1, 1), (6, 5)\}$, wir markieren 5.
2. –
3. Wir wählen Knoten 6.
4. –
5. $b = (w_1, 6)$.
6. $B_2 := \{(w_1, 1), (6, 5), (w_1, 6)\}$, wir markieren 6.
2. –
3. Wir wählen Knoten w_1.
4. –
5. $b = (5, w_1)$.
6. –
7. $B_2 \cup \{(5, w_1)\}$ enthält den Kreis $C_2 = \{(w_1, 6, 5)\}$ mit $W_2 = \{w_1, 5, 6\}$.
8. Der neue Digraph $D_3 = (V_3, A_3)$ mit $V_3 = \{1, w_2\}$ ist in Abbildung 4.7 dargestellt.

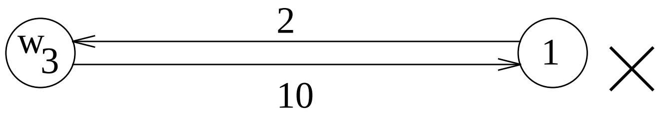

Abb. 4.7

9. In D_3 ist Knoten 1 markiert.
10. $B_3 = \{(w_2, 1)\}$.
2. –
3. Wir wählen Knoten w_1.
4. –

5. $b = (1, w_2)$.
6. –
7. $B_3 \cup \{(1, w_2)\}$ enthält den Kreis $C_3 = \{(1, w_2), (w_2, 1)\}$ mit $W_3 = \{1, w_2\}$.
8. und 9. Der neue Digraph $D_4 = (V_4, A_4)$ besteht aus nur einem unmarkierten Knoten w_3 und enthält keine Bögen.
10. $B_4 = \emptyset$.
2. –
3. Wir wählen Knoten w_3.
4. Wir markieren w_3.
2. Alle Knoten sind markiert.

Wir beginnen nun mit Phase II, der Rekonstruktion des optimalen Branching B von D aus den Branchings B_i, $i = 4, 3, 2, 1, 0$.

11. $i = 4$
12. w_3 ist Wurzel (des leeren Branchings B_4).
    $B_3 := B_4 \cup (C_3 \setminus \{b_{i-1}\}) = \emptyset \cup (\{(1, w_2)(w_2, 1)\} \setminus \{(1, w_2)\}) = \{(w_2, 1)\}$.
11. $i = 3$.
12. w_2 ist Wurzel von B_3.
    $B_2 = B_3 \cup (C_2 \setminus \{b_2\}) = \{(w_1, 1)\} \cup (\{(w_1, 6), (6, 5), (5, w_1)\} \setminus \{(w_1, 6)\}) = \{(w_1, 1), (6, 5), (5, w_1)\}$.
    (Man beachte hier, daß der Bogen $(w_2, 1) \in B_3 \subseteq A_3$ dem Bogen $(w_1, 1) \in A_2$ entspricht.)
11. $i = 2$.
12. –
13. Der Bogen $b = (5, w_1) \in A_2$ entspricht dem Bogen $(5, w_0) \in A_1$, also hat $a_1 = a_{i-1} = (4, w_0) \in C_1$ denselben Endknoten wie b.
    $B_1 := B_2 \cup (C_1 \setminus \{a_1\}) = \{(w_0, 1), (6, 5), (5, w_0)\} \cup (\{(4, w_0), (w_0, 4)\} \setminus \{(4, w_0)\}) = \{(w_0, 1), (6, 5), (5, w_0), (w_0, 4)\}$.
11. $i = 1$.
12. –
13. Der Bogen $b = (5, w_0) \in B_1 \subseteq A_1$ entspricht dem Bogen $(5, 7) \in A_0$, also haben $a_0 = (3, 7) \in C_0$ und b denselben Endknoten.
    $B_0 := B_1 \cup (C_0 \setminus \{a_0\}) = \{(2, 1), (6, 5), (5, 7), (7, 4)\} \cup \{(2, 3), (3, 7), (7, 2)\} \setminus \{(3, 7)\} = \{(2, 1), (6, 5), (5, 7), (7, 4), (2, 3), (7, 2)\}$.
11. $i = 0$, STOP, B_0 ist optimal.

Die Phase II des "Aufblasens" stellen wir noch einmal graphisch in Abbildung 4.8 dar. Das optimale Branching $B = B_0 \subseteq A$ hat das Gewicht $2 + 3 + 4 + 5 + 4 + 10 = 28$.

$\square$

(4.13) Beispiel. Wir betrachten den in Abbildung 4.9 dargestellten Digraphen mit den eingezeichneten Bogengewichten. Wir durchlaufen den Branching-Algorithmus nicht mehr so explizit, wie im vorangegangenen Beispiel, sondern stellen nur noch die Schrumpfungsphasen graphisch dar.

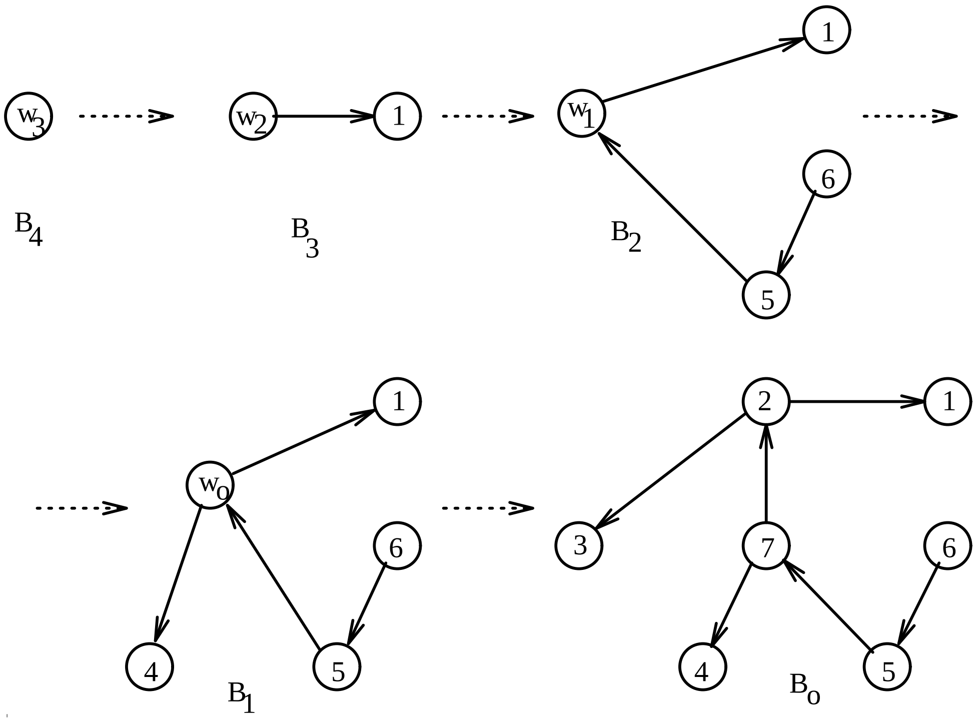

Abb. 4.8

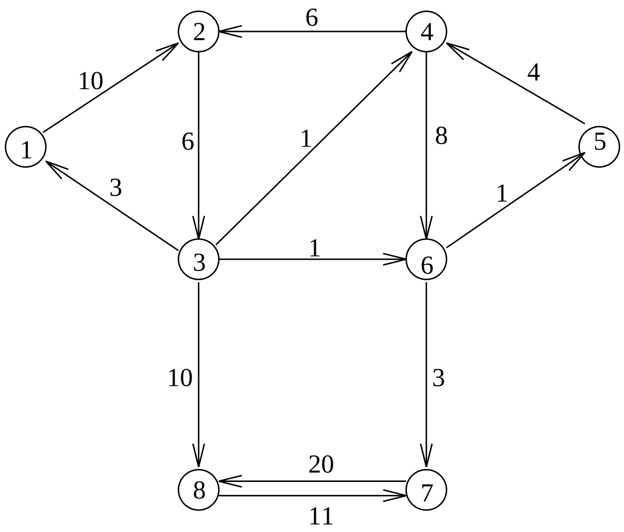

Abb. 4.9

Wir wählen in Schritt 3 nacheinander die Knoten 1, 2, 3 und erhalten den gerichteten Kreis $C_0 = (1, 2, 3)$ mit der Knotenmenge $W_0 = \{1, 2, 3\}$, die zu einem neuen Knoten w_0 wie in Abbildung 4.10 gezeigt, geschrumpft wird.

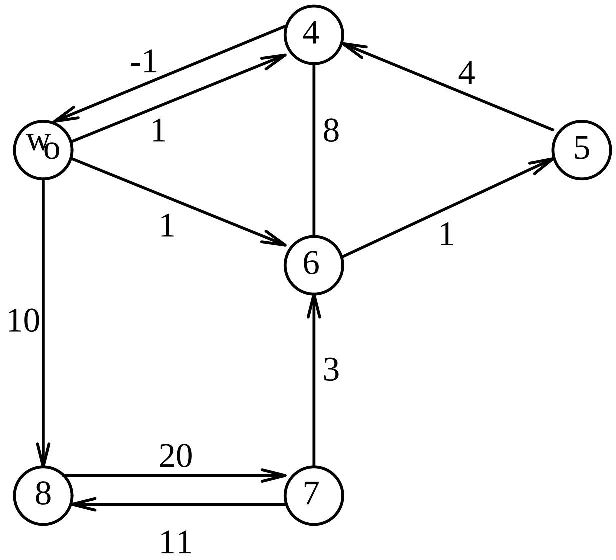

Abb. 4.10

Nun wählen wir in Schritt 3 nacheinander die Knoten w_0, 4, 5, 6. (Da alle in w_0 endenden Bögen negatives Gewicht haben, wird w_0 in Schritt 4 markiert, aber kein Bogen mit Endknoten w_0 gewählt.) Es ergibt sich der Kreis $C_1 = (4,5,6)$ mit $W_1 = \{4,5,6\}$. Der Schrumpfungsprozeß ergibt den in Abbildung 4.11 gezeigten Digraphen.

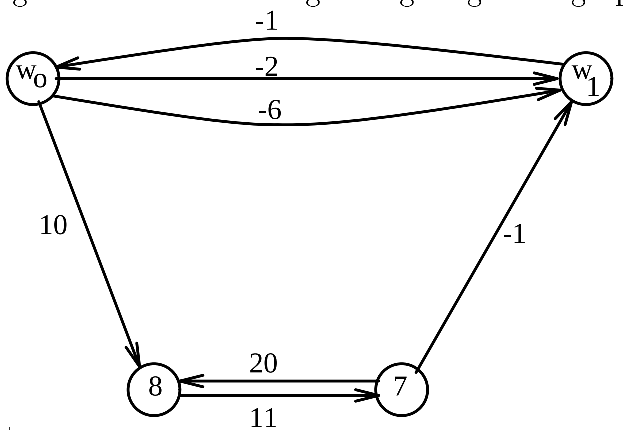

Abb. 4.11

In Schritt 3 wählen wir nun nacheinander die Knoten w_1, 7, 8. Es ergibt sich der gerichtete Kreis $C_2 = (7,8)$ mit $W_2 = \{7,8\}$. Wir schrumpfen W_2 zu w_2, wie in Abbildung 4.12 gezeigt.

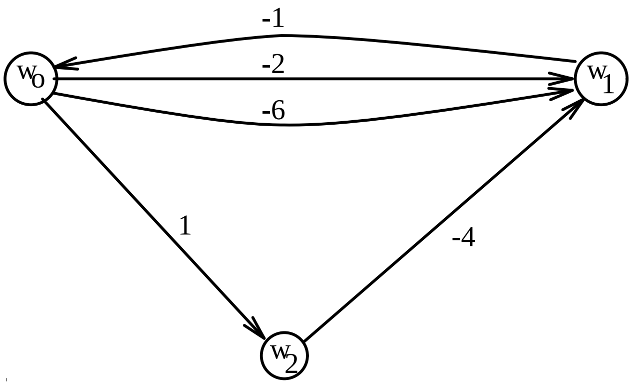

Abb. 4.12

Nun ist lediglich w_2 noch nicht markiert. Wir erhalten den Bogen (w_0, w_2), der ein optimales Branching B_3 des in Abbildung 4.12 gezeigten Digraphen D_3 ist. Alle Knoten von D_3 sind markiert. Wir beginnen die Phase II und rekonstruieren das optimale Branching von D. Dieser Vorgang ist in Abbildung 4.13 dargestellt. Das optimale Branching besteht aus den Bögen $(1,2), (2,3), (3,8), (8,7), (5,4)$ und $(4,6)$.

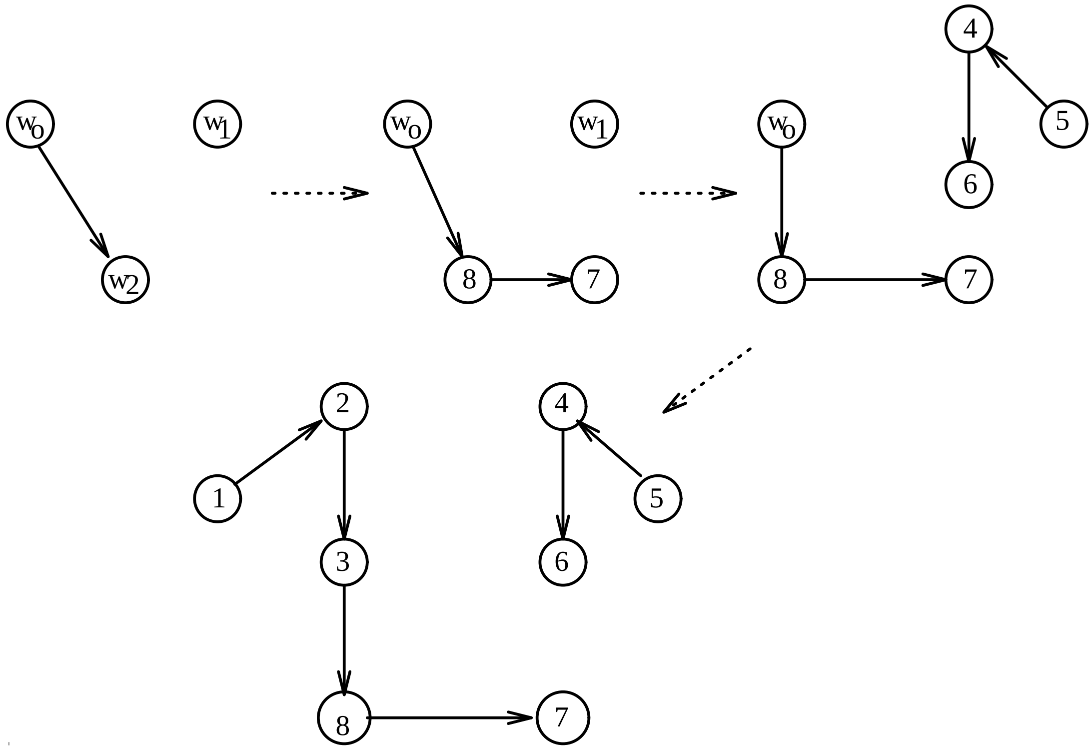

Abb. 4.13

□

Der Branching-Algorithmus (4.11) kann so implementiert werden, daß seine Laufzeit für einen Digraphen mit m Bögen und n Knoten $O(m \log n)$ bzw. bei Digraphen mit vielen Bögen $O(n^2)$ beträgt, siehe hierzu Tarjan (1977) und Camerini, Fratta und Maffioli

(1979). Im letztgenannten Artikel werden einige inkorrekte Implementierungsdetails des Tarjan-Aufsatzes richtiggestellt. Zum Abschluß des Kapitels wollen wir die Korrektheit des Branching-Algorithmus (4.11) beweisen.

(4.14) Lemma. *Bei der i-ten Iteration der Phase I sei in Schritt 7 bei der Addition des Bogens b zum Branching B_i der gerichtete Kreis $C_i \subseteq B_i \cup \{b\}$ mit Knotenmenge W_i gefunden worden. Dann existiert in $D_i = (V_i, A_i)$ ein bezüglich der Gewichtsfunktion c_i maximales Branching B^* mit folgenden Eigenschaften:*

(a) $|B^* \cap ((\bigcup_{v \in W_i} \delta^-(v)) \setminus C_i)| \leq 1$,

(b) $|B^* \cap C_i| = |C_i| - 1$.

Beweis. Wir nehmen an, daß Behauptung (a) falsch ist, und wählen ein maximales Branching B^* von D_i maximalen Gewichts, so daß die Anzahl der Bögen in $\Delta := (B^* \cap \bigcup_{v \in W_i} \delta^-(v)) \setminus C_i$ minimal ist. Zur Notationsvereinfachung nehmen wir an, daß $C_i = (1, 2, \ldots, r)$ gilt. Es sei $\Delta = \{(u_j, v_j) \mid j = 1, \ldots, k\}$ (nach Annahme gilt $k \geq 2$), und wir können o. B. d. A. annehmen, daß die Knoten $v_j \in W_i$ so numeriert sind, daß $v_p < v_q$ gilt, falls $p < q$.

Wir untersuchen nun die Mengen $B_j^* := (B^* \setminus \{(u_j, v_j)\}) \cup \{(v_j - 1, v_j)\}$ für $j = 1, \ldots, k$. Wäre für irgendeinen Bogen $(u_j, v_j) \in \Delta$ die Menge B_j^* ein Branching, so wäre aufgrund der Auswahlvorschriften in Schritt 5 $c_i(B_j^*) - c_i(B^*) = c_i((v_j - 1, v_j)) - c_i((u_j, v_j)) \geq 0$. D. h. B_j^* wäre ein maximales Branching mit weniger Bögen in $\delta^-(W_i)$ als B^*, ein Widerspruch zu unserer Minimalitätsannahme. Folglich enthält B_j^* einen Kreis, der $(v_j - 1, v_j)$ enthält. Da in B_j^* keine zwei Kanten einen gemeinsamen Endknoten besitzen, ist dieser Kreis gerichtet. Daraus folgt, daß B_j^* — und somit B^* — einen gerichteten $(v_j, v_j - 1)$-Weg P_j enthält. Enthielte P_j nur Bögen des Kreises C_i, so wäre $P_j = C_i - (v_j - 1, v_j)$; daraus folgte, daß außer v_j kein weiterer Knoten aus W_i Endknoten eines Bogens aus Δ sein könnte. Dies widerspräche unserer Annahme $k \geq 2$. Mithin enhält P_j mindestens einen Bogen aus Δ. Sei d der letzte (gesehen in Richtung v_j nach $v_j - 1$) Bogen von P_j, der in Δ liegt. Dann muß d der Bogen (u_{j-1}, v_{j-1}) sein, da jeder gerichtete Weg in B^* von irgendeinem Knoten $v \in V \setminus \{v_{j-1}, v_{j-1}+1, v_{j-1}+2, \ldots, v_j - 1\}$ zum Knoten v_j den Weg $(v_{j-1}, v_{j-1}+1, \ldots, v_j - 1)$ enthalten muß und v_{j-1} in B^* nur über den Bogen (u_{j-1}, v_{j-1}) zu erreichen ist. Mithin enthält B^* einen gerichteten Weg $\overline{P}_j$ von v_j nach v_{j-1}. Diese Aussage gilt für $j = 1, \ldots, k$ (wobei $v_0 = v_k$ zu setzen ist). Setzen wir diese Wege wie folgt zusammen

$$v_k, \overline{P}_k, v_{k-1}, \overline{P}_{k-1}, v_{k-2}, \ldots, v_2 \overline{P}_2, v_1, \overline{P}_1, v_0 = v_k,$$

so ergibt sich eine geschlossene gerichtete Kette. Diese Kette enthält einen gerichteten Kreis, und dieser Kreis ist eine Teilmenge von B^*. Widerspruch! Damit ist der Beweis von (a) erledigt.

Sei nun B^* ein maximales Branching, das (a) erfüllt und für das $|C_i \cap B^*|$ maximal ist. Enthält B^* keinen Bogen aus $\delta^-(W_i)$, so ist nach (a) $B^* \cap (A(W_i) \cup \delta^-(W_i)) \subseteq C_i$. Trivialerweise muß dann $|B^* \cap C_i| = |C_i| - 1$ gelten. Enthält B^* einen Bogen aus $\delta^-(W_i)$, sagen wir a, dann sei a_i der Bogen aus C_i mit $h_i(a_i) = h_i(a)$. Gilt $|B^* \cap C_i| < |C_i| - 1$, so ist $B' := (B^* \setminus \{a\}) \cup \{a_i\}$ ein Branching, für das aufgrund von Schritt 5 gilt $c_i(B') \geq c_i(B^*)$. Dies widerspricht unserer Annahme, daß $|C_i \cap B^*|$ maximal ist. Damit ist auch (b) gezeigt. □

(4.15) Satz. *Zu jedem Digraphen $D = (V, A)$ mit Gewichten $c(a)$ für alle $a \in A$ liefert Algorithmus (4.11) ein maximales Branching.*

Beweis. Führen wir die Iteration der Phase I zum letzten Mal (sagen wir k-ten Mal) aus, so haben alle Bögen d des gegenwärtigen Branchings $B_k \subseteq A_k$ die Eigenschaft $c_k(d) \geq c_k(a)$ für alle Bögen a mit $h_k(a) = h_k(d)$ (wegen Schritt 5). Für die Knoten $v \in V_k$, die nicht Endknoten eines Bogens aus B_k sind, gilt wegen Schritt 4 $c_k(a) \leq 0$ für alle $a \in \delta^-(v)$. B_k ist also offensichtlich ein maximales Branching in D_k, das nur Bögen mit positivem c_k-Gewicht enthält. Durch Induktion zeigen wir nun: Ist B_i ein maximales Branching von D_i bezüglich c_i mit $c_i(b) > 0 \; \forall \, b \in B_i$, so ist das in den Schritten 12 oder 13 definierte Branching B_{i-1} maximal in D_{i-1} bezüglich c_{i-1} mit $c_{i-1}(b) > 0 \; \forall \, b \in B_{i-1}$. Der Induktionsanfang für $i = k$ ist durch die obige Bemerkung gegeben.

Wir nehmen an, daß die Behauptung für i, $0 < i \leq k$, richtig ist und wollen zeigen, daß sie auch für $i - 1$ gilt. Sei also B_i das durch den Algorithmus gefundene maximale Branching von D_i bezüglich c_i, sei $C_{i-1} \subseteq B_{i-1} \cup \{b\}$ der in Schritt 7 gefundene gerichtete Kreis mit Knotenmenge W_{i-1}, sei $b_{i-1} \in C_{i-1}$ ein Bogen minimalen c_{i-1}-Gewichts, und sei B^* ein bezüglich c_{i-1} maximales Branching von D_{i-1}, das die Bedingungen (a) und (b) von Lemma (4.14) erfüllt.

1. Fall: $\{a\} = B^* \cap \delta^-(W_{i-1})$. Dann hat B^* die Form $B^* = B' \cup \{a\} \cup (C_{i-1} \setminus \{a_{i-1}\})$, wobei $B' \subseteq A_{i-1} \setminus (\delta^-(W_{i-1}) \cup A(W_{i-1}))$ und $h_{i-1}(a) = h_{i-1}(a_{i-1})$ gilt, und $B' \cup \{a\}$ ist ein Branching in D_i. Da B_i ein maximales Branching von D_i ist, gilt

$$\begin{aligned} c_{i-1}(B^*) &= c_{i-1}(B' \cup \{a\}) + c_{i-1}(C_{i-1} \setminus \{a_{i-1}\}) \\ &= c_i(B' \cup \{a\}) - c_{i-1}(b_{i-1}) + c_{i-1}(a_{i-1}) + c_{i-1}(C_{i-1} \setminus \{a_{i-1}\}) \\ &\leq c_i(B_i) + c_{i-1}(C_{i-1} \setminus \{b_{i-1}\}) \end{aligned}$$

Enthält B_i keinen Bogen aus $\delta^-(W_{i-1})$, so ist $B_i \cup (C_{i-1} \setminus \{b_{i-1}\})$, das in Schritt 12 konstruierte Branching B_{i-1}. Aus $c_i(B_i) = c_{i-1}(B_i)$ folgt $c_{i-1}(B_{i-1}) \geq c_{i-1}(B^*)$ und damit die Maximalität von B_{i-1} in D_{i-1} bezüglich c_{i-1}. Enthält B_i einen Bogen $b \in \delta^-(W_{i-1})$ und ist $d_{i-1} \in C_{i-1}$ mit $h_{i-1}(d_{i-1}) = h_{i-1}(b)$, so gilt $B_{i-1} = B_i \cup (C_{i-1} \setminus \{d_{i-1}\})$ und folglich

$$\begin{aligned} c_{i-1}(B_{i-1}) &= c_i(B_i) - c_{i-1}(b_{i-1}) + c_{i-1}(d_{i-1}) + c_{i-1}(C_{i-1} \setminus \{d_{i-1}\}) \\ &= c_i(B_i) + c_{i-1}(C_{i-1} \setminus \{b_{i-1}\}) \geq c_{i-1}(B^*). \end{aligned}$$

Also ist B_{i-1} maximal. □

2. Fall: $B^* \cap \delta^-(W_{i-1}) = \emptyset$. Der Beweis verläuft analog zum 1. Fall.

Sei also B_i maximal in D_i bezüglich der Gewichtsfunktion c_i. Sei $C_{i-1} \subseteq B_{i-1} \cup \{b\}$ der in Schritt 7 gefundene gerichtete Kreis mit Knotenmenge W_{i-1}, und sei B^* ein bezüglich c_{i-1} maximales Branching von D_{i-1}, das eine minimale Zahl von Bögen aus $\delta^-(W_{i-1})$ enthält. Ein solches Branching hat nach Lemma (4.14) (a) höchstens einen Bogen in $\delta^-(W_{i-1}$. Wir setzen $B^* = B' \cup B'' \cup B'''$ mit

$$\begin{aligned} B' &:= B^* \setminus (A(W_{i-1}) \cup \delta^-(W_{i-1}), \\ B'' &:= B^* \cap A(W_{i-1}), \\ B''' &:= B^* \cap \delta^-(W_{i-1}). \end{aligned}$$

Nach Annahme ist B''' entweder leer oder eindeutig. Sei B_i das durch den Algorithmus bestimmte Branching von D_i, das nach Induktionsannahme in D_i maximal bezüglich der Gewichte c_i ist (und dessen Bögen positive c_i-Gewichte haben). Sei C_{i-1} der Schritt 7 der Iteration $i-1$ gefundene gerichtete Kreis, und b_{i-1} sei ein Bogen des Kreises C_{i-1} mit kleinstem (positivem) c_{i-1}-Gewicht.

1. Fall: $B''' = \emptyset$. Nach Lemma (4.14) (b) ist dies gleichwertig mit $c_i(a) \leq 0$ für alle $a \in \delta^-(w_{i-1}) \subseteq A_i$. Also ist $B_i \cap \delta^-(w_{i-1}) = \emptyset$. Aufgrund der Auswahl der Bögen in Schritt 5 gilt $c_{i-1}(C_{i-1} \setminus \{b_{i-1}\}) \geq c_{i-1}(B'')$. Ferner ist B' ein Branching in D_i, woraus $c_i(B') \leq c_i(B_i)$ folgt. Da w_{i-1} eine Wurzel in B_i ist, gilt nach Schritt 12 $B_{i-1} = B_i \cup (C_{i-1} \setminus \{b_{i-1}\})$. Daraus ergibt sich

$$\begin{aligned} c_{i-1}(B_{i-1}) &= c_{i-1}(B_i) + c_{i-1}(C_{i-1} \setminus \{b_{i-1}\}) = c_i(B_i) + c_{i-1}(C_{i-1} \setminus \{b_{i-1}\}) \\ &\geq c_i(B') + c_{i-1}(B'') = c_{i-1}(B') + c_{i-1}(B'') = c_{i-1}(B^*). \end{aligned}$$

Mithin ist B_{i-1} c_{i-1}-maximal in D_{i-1}.

2. Fall: $B''' = \{d\}$. Es sei d_{i-1} der Bogen in C_{i-1} mit $h_{i-1}(d_{i-1}) = h_{i-1}(d)$. Nach Lemma (4.14) (b) ist $B''' \neq \emptyset$ gleichwertig damit, daß mindestens ein Bogen aus $\delta^-(w_{i-1})$ positives c_i-Gewicht hat. Aus den Vorschriften in (4.11) folgt, daß w_{i-1} keine Wurzel von B_i sein kann. Das heißt, es gibt einen Bogen $\overline{a}_i \in B_i$ mit $h_i(\overline{a}_i) = w_{i-1}$. Ferner sei a_{i-1} der Bogen in C_{i-1} mit $h_{i-1}(\overline{a}_i) = h_{i-1}(a_{i-1})$. Nach Schritt 13 ist dann $B_{i-1} = B_i \cup (C_{i-1} \setminus \{a_{i-1}\})$. Ferner sei b_{i-1} ein Bogen aus C_{i-1} mit kleinstem c_{i-1}-Gewicht. Aufgrund der Auswahlvorschriften in Schritt 5 gilt $c_{i-1}(C_{i-1}) \geq c_{i-1}(B'') + c_{i-1}(d)$. Ferner ist $B' \cup \{d\}$ ein Branching in D_i, woraus $c_i(B_i) \geq c_i(B') + c_i(d)$ folgt. Daraus ergibt sich nun

$$\begin{aligned} c_{i-1}(B^*) &= c_{i-1}(B') + c_{i-1}(d) + c_{i-1}(B'') \\ &= c_i(B') + c_i(d) - c_{i-1}(b_{i-1}) + c_{i-1}(d_{i-1}) + c_{i-1}(B'') \\ &\leq c_i(B_i) - c_{i-1}(b_{i-1}) + c_{i-1}(d_{i-1}) + c_{i-1}(B'') \\ &= c_{i-1}(B_i) + c_{i-1}(b_{i-1}) - c_{i-1}(a_{i-1}) - c_{i-1}(b_{i-1}) + c_{i-1}(d_{i-1}) + c_{i-1}(B'') \\ &\leq c_{i-1}(B_i) - c_{i-1}(a_{i-1}) + c_{i-1}(C_{i-1}) \\ &= c_{i-1}(B_{i-1}). \end{aligned}$$

Damit ist die Behauptung bewiesen. □

Literaturverzeichnis von Kapitel 4.

F. Bock (1971), "An algorithm to construct a minimum directed spanning tree in a directed network, in: B. Avi-Itzhack (ed.) Developements in Operations Research", Gordon and Breach, New York (1971) 29–44.

P. M. Camerini, L. Fratta und F. Maffioli (1979), "A Note on Finding Optimum Branchings", *Networks* 9 (1979) 309–312.

Y. J. Chu, und T. H. Liu (1965), "On the shortest arborescence of a directed graph", Scientia Sinica 4 (1965) 1396–1400.

J. Edmonds (1967a), "Optimum branching", Journal of Research of the National Bureau of Standards B 71 (1967a) 233–240.

R. L. Graham und P. Hell (1982), "On the History of the Minimum Spanning Tree Problem", Annals of the History of Computing 7 (1985), 43–57.

R. E. Tarjan (1977), "Finding Optimum Branchings", *Networks* 7 (1977) 25–35.

Kapitel 5

Matroide und Unabhängigkeitssysteme

Wir werden nun einige sehr allgemeine Klassen von kombinatorischen Optimierungsproblemen kennenlernen. Diese enthalten die im vorigen Kapitel betrachteten Probleme. Die in Kapitel 4 besprochenen Algorithmen sind ebenfalls auf die neuen Probleme übertragbar. Allerdings treten bei der algorithmischen Behandlung dieser Probleme einige Schwierigkeiten auf, die wir eingehend diskutieren müssen.

Gute Bücher über Matroidtheorie sind Oxley (1992), Truemper (1992) und Welsh (1976), ein neuerer Übersichtsartikel Welsh (1995). Optimierungsprobleme auf Matroiden werden z. B. in Bixby und Cunningham (1995) ausführlich behandelt.

5.1 Unabhängigkeitssysteme

E bezeichnet im folgenden immer eine endliche Menge. 2^E ist die Menge aller Teilmengen von E.

(5.1) Definition. *Eine Menge $\mathcal{I} \subseteq 2^E$ heißt* **Unabhängigkeitssystem** *(oder* **monotones Mengensystem***) auf E, wenn $\mathcal{I}$ die folgenden Axiome erfüllt:*

(I.1) $\emptyset \in \mathcal{I}$,

(I.2) $F \subseteq G \in \mathcal{I} \Longrightarrow F \in \mathcal{I}$.

Häufig wird auch das Paar $(E, \mathcal{I})$ Unabhängigkeitssystem genannt. Die Teilmengen von E, die in $\mathcal{I}$ enthalten sind, heißen **unabhängige Mengen**, *alle übrigen Teilmengen von E heißen* **abhängige Mengen**.

$\square$

Mit Unabhängigkeitssystemen sind auf kanonische Weise andere Mengensysteme gegeben, die wir nun kurz einführen wollen.

Die bezüglich Mengeninklusion minimalen abhängigen Teilmengen von E heißen **Zirkuits** (oder **Kreise**), d. h. $C \subseteq E$ ist ein Zirkuit, wenn C abhängig ist und wenn C keine von sich selbst verschiedene abhängige Teilmenge enthält. Die Menge aller Zirkuits (bzgl. eines Unabhängigkeitssystems $\mathcal{I}$) wird mit $\mathcal{C}$ bezeichnet.

Ist $F \subseteq E$, so heißt jede Teilmenge von F, die unabhängig ist und die in keiner anderen unabhängigen Teilmenge von F enthalten ist, **Basis** von F, d. h.

$$B \text{ Basis von } F \Longleftrightarrow (B, B' \in \mathcal{I}, B \subseteq B' \subseteq F \Longrightarrow B = B').$$

Die Menge aller Basen der Grundmenge E heißt **Basissystem** (bezgl. $\mathcal{I}$) und wird mit $\mathcal{B}$ bezeichnet.

Für jede Menge $F \subseteq E$ heißt die ganze Zahl

$$r(F) := \max\{|B| \mid B \text{ Basis von } F\} \tag{5.2}$$

Rang von F. Die Rangfunktion r ist also eine Funktion, die 2^E in die nicht-negativen ganzen Zahlen abbildet.

Offenbar induziert jedes Unabhängigkeitssystem $\mathcal{I}$ auf E ein eindeutig bestimmtes Zirkuitsystem, ein eindeutig bestimmtes Basissystem und eine eindeutig bestimmte Rangfunktion. Es gilt auch die Umkehrung.

Zirkuitsysteme und Basissysteme sind nach Definition **Antiketten (Clutter)**, d. h. Systeme von Mengen, so daß keine zwei Mengen ineinander enthalten sind.

Ist $\mathcal{B} \neq \emptyset$ eine Antikette auf E, so ist

$$\mathcal{I} := \{I \subseteq E \mid \exists\, B \in \mathcal{B} \text{ mit } I \subseteq B\}$$

ein Unabhängigkeitssystem auf E, und $\mathcal{B}$ ist das zu $\mathcal{I}$ gehörige Basissystem.

Ist $\mathcal{C} \neq \{\emptyset\}$ eine Antikette auf E, so ist

$$\mathcal{I} := \{I \subseteq E \mid I \text{ enthält kein Element von } \mathcal{C}\}$$

ein Unabhängigkeitssystem, und $\mathcal{C}$ ist das zu $\mathcal{I}$ gehörige Zirkuitsystem.

Die oben definierte Rangfunktion hat folgende Eigenschaften. Sie ist **subkardinal**, d. h. für alle $F \subseteq E$ gilt

$$r(F) \leq |F|,$$

sie ist **monoton**, d. h. für alle $F, G \subseteq E$ gilt

$$F \subseteq G \Longrightarrow r(F) \leq r(G),$$

und sie ist **stark subadditiv**, d. h. für alle $F \subseteq E$, für alle ganzen Zahlen $k \geq 1$ und für alle Familien $(F_i, i \in K)$ von Teilmengen von F mit der Eigenschaft, daß $|\{i \in K \mid e \in F_i\}| = k$ für alle $e \in F$, gilt

$$k \cdot r(F) \leq \sum_{i \in K} r(F_i).$$

Ist $r : 2^E \rightarrow \mathbf{Z}_+$ eine subkardinale, monotone, stark subadditive Funktion, so ist

$$\mathcal{I} := \{I \subseteq E \mid r(I) = |I|\}$$

ein Unabhängigkeitssystem, dessen Rangfunktion die Funktion r ist.

Unabhängigkeitssysteme $\mathcal{I}$ auf einer Grundmenge E definieren also mathematische Strukturen, die äquivalent durch Zirkuitsysteme, Basissysteme oder Rangfunktionen gegeben werden können. Unabhängigkeitssysteme sind sehr allgemeine Objekte und besitzen zu wenig Struktur, um tiefliegende Aussagen über sie machen zu können.

Sei für jedes Element $e \in E$ ein "Gewicht" $c_e \in \mathbb{K}$ gegeben. Für $F \subseteq E$ setzen wir wie üblich

$$c(F) := \sum_{e \in F} c_e.$$

Das Problem, eine unabhängige Menge $I^* \in \mathcal{I}$ zu finden, so daß $c(I^*)$ maximal ist, heißt **Optimierungsproblem über einem Unabhängigkeitssystem** $\mathcal{I}$, d. h. wir suchen

$$\max\{c(I) \mid I \in \mathcal{I}\}. \tag{5.3}$$

Offenbar macht es hier keinen Sinn, Gewichte c_e zu betrachten, die nicht positiv sind. Denn wenn $I^* \in \mathcal{I}$ optimal ist, gilt $I' := I^* \setminus \{e \in E \mid c_e \leq 0\} \in \mathcal{I}$ (wegen (I.2)) und $c(I') \geq c(I^*)$, also ist I' ebenfalls eine optimale unabhängige Menge. Deswegen werden wir uns im weiteren bei Optimierungsproblemen über Unabhängigkeitssystemen auf Gewichtsfunktionen beschränken, die positiv oder nicht-negativ sind.

Bei Optimierungsproblemen über Basissystemen ist es dagegen auch sinnvoll negative Gewichte zuzulassen. Ist ein Basissystem $\mathcal{B}$ auf der Grundmenge E gegeben und sind $c_e \in \mathbb{K}$ Gewichte, so nennen wir

$$\min\{c(B) \mid B \in \mathcal{B}\} \tag{5.4}$$

Optimierungsproblem über einem Basissystem.

(5.5) Beispiele.
(a) Sei $G = (V, E)$ ein Graph. Eine Knotenmenge $S \subseteq V$ heißt **stabil (Clique)**, falls je zwei Knoten aus S nicht benachbart (benachbart) sind. Die Menge der stabilen Knotenmengen (Cliquen) ist ein Unabhängigkeitssystem auf V. Die Aufgabe — bei gegebenen Knotengewichten — eine gewichtsmaximale stabile Menge (Clique) zu finden, ist ein Optimierungsproblem über einem Unabhängigkeitssystem, vergleiche (2.14).

(b) Ein **Wald** in einem Graphen $G = (V, E)$ ist eine Kantenmenge, die keinen Kreis enthält. Die Menge aller Wälder bildet ein Unabhängigkeitssystem auf E. Das Problem, einen maximalen Wald in G zu finden, war Hauptthema von Kapitel 4, siehe auch (2.12). Offenbar ist in einem zusammenhängenden Graphen G die Menge der aufspannenden Bäume genau die Menge der Basen des Unabhängigkeitssystems der Wälder. Das Problem, einen minimalen aufspannenden Baum zu finden, ist somit ein Optimierungsproblem über einem Basissystem.

(c) Ein **Matching** in einem Graphen $G = (V, E)$ ist eine Kantenmenge $M \subseteq E$, so daß jeder Knoten aus V in höchstens einer Kante aus M enthalten ist. Die Menge aller Matchings ist ein Unabhängigkeitssystem auf E. Das Matchingproblem (2.10) ist also ein Optimierungsproblem über einem Unabhängigkeitssystem. Die Aufgabe, in einem vollständigen Graphen mit Kantengewichten ein minimales perfektes Matching zu finden, ist ein Optimierungsproblem über einem Basissystem.

(d) Gegeben sei ein vollständiger Digraph $D_n = (V, A)$ mit n Knoten und Bogenlängen c_{ij} für alle $(i, j) \in A$. Eine **Tour (gerichteter Hamilton-Kreis)** ist ein gerichteter Kreis in D_n, der jeden Knoten enthält. Die Aufgabe, eine Tour mit minimalem Gewicht zu finden, heißt **asymmetrisches Travelling-Salesman-Problem**, siehe (2.13). Die Menge $\mathcal{T}$ aller Touren ist kein Unabhängigkeitssystem, jedoch eine Antikette, also Basissystem eines Unabhängigkeitssystems. Das asymmetrische TSP ist also ein Optimierungsproblem über einem Basissystem. Wir können es aber auch als Optimierungsproblem über einem Unabhängigkeitssystem auffassen. Dies geht wie folgt. Setzen wir

$$\tilde{\mathcal{T}} := \{I \subseteq A \mid \exists T \in \mathcal{T} \text{ mit } I \subseteq T\},$$
$$c'_{ij} := \max\{|c_{ij}| \mid (i, j) \in A\} + 1 - c_{ij},$$

so ist $\tilde{\mathcal{T}}$ ein Unabhängigkeitssystem, und jede Lösung von $\max\{c'(I) \mid I \in \tilde{\mathcal{T}}\}$ ist eine Tour, die — bezüglich der Gewichte c_{ij} — minimales Gewicht hat. (Auf die gleiche Weise kann man viele andere Optimierungsprobleme über Basissystemen in Optimierungsprobleme über Unabhängigkeitssystemen überführen.) Ebenso ist das symmetrische TSP ein Optimierungsproblem über einem Basissystem.

(e) Gegeben sei ein gerichteter Graph $D = (V, A)$. Ein **Branching** in D ist eine Bogenmenge $B \subseteq A$, die keinen Kreis (im ungerichteten Sinne) enthält, und die die Eigenschaft hat, daß jeder Knoten $v \in V$ Endknoten von höchstens einem Bogen aus B ist. Die Menge aller Branchings ist ein Unabhängigkeitssystem auf A, siehe Kapitel 4 und (2.12). Das Problem, in einem vollständigen Digraphen eine minimale aufspannende Arboreszenz zu finden, ist ein Optimierungsproblem über einem Basissystem.

$\square$

Überlegen Sie sich, welche der übrigen Beispiele aus Abschnitt 2.3 als Optimierungsprobleme über Unabhängigkeits- oder Basissystemen aufgefaßt werden können und welche nicht.

Verschiedene praktische Fragestellungen führen auch zu Optimierungsproblemen über Zirkuits. Sind die Elemente $e \in E$ durch Gewichte c_e bewertet, so kann man das Problem untersuchen, ein Zirkuit $C \in \mathcal{C}$ zu finden, das minimales Gewicht $c(C)$ hat. Die Aufgabe, in einem Graphen einen kürzesten Kreis zu bestimmen, ist z. B. von diesem Typ.

Allgemeiner noch ist folgende Frage von Interesse. Wir sagen, daß eine Menge $Z \subseteq E$ eine **Zyklus** ist, wenn Z die Vereinigung von paarweise disjunkten Zirkuits ist, d. h. wenn es Zirkuits $C_1, \ldots, C_k$ gibt mit $C_i \cap C_j = \emptyset$, $1 \leq i < j \leq k$, so daß $Z = \bigcup_{i=1}^{k} C_i$. Sind die Elemente $e \in E$ mit Gewichten c_e belegt, so sucht man nach einem Zyklus maximalem Gewichts. Das Chinesische Postbotenproblem (siehe (2.13)) und das Max-Cut-Problem (2.16) sind z. B. von diesem Typ. Aus Zeitgründen können wir auf Optimierungsprobleme über Zirkuits bzw. Zyklen nicht eingehen, siehe hierzu Barahona und Grötschel (1986) und Grötschel und Truemper (1989).

5.2 Matroide

Wie die Beispiele aus dem vorigen Abschnitt zeigen, enthalten Optimierungsprobleme über

Unabhängigkeitssystemen sowohl polynomial lösbare als auch $\mathcal{NP}$-vollständige Probleme. Man wird daher nicht erwarten können, daß für diese Probleme eine "gute" Lösungstheorie existiert. Wir wollen nun eine Spezialklasse von Unabhängigkeitssystemen einführen, für die es so etwas gibt. In einem noch zu präzisierenden Sinn (siehe Folgerung (5.16)) ist dies die Klasse der Unabhängigkeitssysteme, für die der Greedy-Algorithmus (siehe (4.1)) eine Optimallösung liefert.

(5.6) Definition. *Ein* **Matroid** *M besteht aus einer Grundmenge E zusammen mit einem Unabhängigkeitssystem $\mathcal{I} \subseteq 2^E$, das eine der folgenden äquivalenten Bedingungen erfüllt:*

(I.3) $\quad I, J \in \mathcal{I}, |I| = |J| - 1 \Longrightarrow \exists\, j \in J \setminus I$ *mit* $I \cup \{j\} \in \mathcal{I}$,

(I.3′) $\quad I, J \in \mathcal{I}, |I| < |J| \Longrightarrow \exists\, K \subseteq J \setminus I$ *mit* $|I \cup K| = |J|$ *so daß* $I \cup K \in \mathcal{I}$,

(I.3″) $\quad F \subseteq E$, *B, B' Basen von F* $\Longrightarrow |B| = |B'|$.

□

Das heißt also, das Unabhängigkeitssystem eines Matroids auf E ist ein Mengensystem $\mathcal{I} \subseteq 2^E$, das die Axiome (I.1), (I.2) und eines der Axiome (I.3), (I.3′), (I.3″) erfüllt. Ein solches System erfüllt automatisch auch die beiden übrigen der drei Axiome (I.3), (I.3′), (I.3″).

Ein Wort zur Terminologie! Nach der obigen Definition ist ein Matroid M ein Paar $(E, \mathcal{I})$ mit den oben aufgeführten Eigenschaften. Wenn klar ist, um welche Grundmenge E es sich handelt, spricht man häufig auch einfach von dem Matroid $\mathcal{I}$, ohne dabei E explizit zu erwähnen. Man sagt auch, M (bzw. $\mathcal{I}$) ist ein Matroid auf E.

In Definition (5.6) haben wir von den "äquivalenten Bedingungen" (I.3), (I.3′), (I.3″) gesprochen. Diese Äquivalenz muß natürlich bewiesen werden. Da wir hier jedoch keine Vorlesung über Matroide halten wollen, können wir auf die Beweise (aus Zeitgründen) nicht eingehen. Das gleiche gilt für die nachfolgend gemachten Aussagen über Zirkuit-, Basissysteme und Rangfunktionen. Beweise der hier gemachten Aussagen findet der interessierte Leser z.B. in Welsh (1976).

Wie wir bereits gesehen haben, können Unabhängigkeitssysteme über Zirkuits, Basen oder Rangfunktionen beschrieben werden. Ist $M = (E, \mathcal{I})$ ein Matroid und sind $\mathcal{C}, \mathcal{B}, r$ das zugehörige Zirkuit-, Basissystem bzw. die zugehörige Rangfunktion, so ist natürlich $\mathcal{I}$ durch die Angabe von $\mathcal{C}$, $\mathcal{B}$ oder r eindeutig beschrieben. Gelegentlich werden daher auch die Paare $(E, \mathcal{C})$, $(E, \mathcal{B})$, (E, r) als Matroide bezeichnet, meistens dann, wenn es bei einer speziellen Untersuchung sinnvoller erscheint, mit Zirkuits, Basen oder Rangfunktionen statt mit Unabhängigkeitssystemen zu arbeiten.

Sicherlich induzieren nicht alle Zirkuit- oder Basissysteme oder alle Rangfunktionen Unabhängigkeitssysteme von Matroiden. Diese Antiketten bzw. Funktionen müssen spezielle Eigenschaften haben, damit das zugehörige Unabhängigkeitssystem das Axiom (I.3) erfüllt. Einige solcher Eigenschaften wollen wir kurz auflisten.

(5.7) Satz. *(a) Eine Antikette $\mathcal{C} \subseteq 2^E$, $\mathcal{C} \neq \{\emptyset\}$, ist das* **Zirkuitsystem eines Matroids** *auf E genau dann, wenn eine der beiden folgenden äquivalenten Bedingungen erfüllt ist:*

(C.1) $C_1, C_2 \in \mathcal{C}$, $C_1 \neq C_2$, $z \in C_1 \cap C_2 \Longrightarrow \exists\, C_3 \in \mathcal{C}$ *mit* $C_3 \subseteq (C_1 \cup C_2) \setminus \{z\}$,

(C.1′) $C_1, C_2 \in \mathcal{C}$, $C_1 \neq C_2$, $y \in C_1 \setminus C_2 \Longrightarrow \forall\, x \in C_1 \cap C_2 \, \exists\, C_3 \in \mathcal{C}$ *mit* $y \in C_3 \subseteq (C_1 \cup C_2) \setminus \{x\}$.

(b) Eine Antikette $\mathcal{B} \subseteq 2^E$, $\mathcal{B} \neq \emptyset$, *ist das* **Basissystem eines Matroids** *auf* E *genau dann, wenn das folgende Axiom erfüllt ist:*

(B.1) $B_1, B_2 \in \mathcal{B}$, $x \in B_1 \setminus B_2 \Longrightarrow \exists\, y \in B_2 \setminus B_1$ *mit* $(B_1 \cup \{y\}) \setminus \{x\} \in \mathcal{B}$.

(c) eine Funktion $r : 2^E \to \mathbf{Z}$ *ist die* **Rangfunktion eines Matroids** *auf* E *genau dann, wenn eines der beiden folgenden äquivalenten Axiomensysteme erfüllt ist:*

(R.1) $r(\emptyset) = 0$,

(R.2) $F \subseteq E$, $e \in E \Longrightarrow r(F) \leq r(F \cup \{e\}) \leq r(F) + 1$,

(R.3) $F \subseteq E$, $f, g \in E$ *mit* $r(F \cup \{g\}) = r(F \cup \{f\}) = r(F)$ $\Longrightarrow r(F \cup \{g, f\}) = r(F)$,

beziehungsweise

(R.1′) $F \subseteq E \Longrightarrow 0 \leq r(F) \leq |F|$, *(r ist subkardinal)*

(R.2′) $F \subseteq G \subseteq E \Longrightarrow r(F) \leq r(G)$, *(r ist monoton)*

(R.3′) $F, G \subseteq E \Longrightarrow r(F \cup G) + r(F \cap G) \leq r(F) + r(G)$. *(r ist submodular)*

□

Es gibt noch einige hundert weitere äquivalente Definitionen von Matroiden (die Äquivalenz ist — auch in den obigen Fällen — nicht immer offensichtlich). Wir wollen uns jedoch mit den obigen begnügen.

Ist $\mathcal{I}_1$ ein Matroid auf einer Grundmenge E_1 und $\mathcal{I}_2$ ein Matroid auf E_2, so heißen $\mathcal{I}_1$ und $\mathcal{I}_2$ **isomorph**, falls es eine bijektive Abbildung $\varphi : E_1 \to E_2$ gibt mit

$$\varphi(F) \text{ ist unabhängig in } \mathcal{I}_2 \Longleftrightarrow F \text{ ist unabhängig in } \mathcal{I}_1.$$

Eine Teilmenge $F \subseteq E$ heißt **abgeschlossen** (bezüglich $\mathcal{I}$), falls gilt

$$r(F) < r(F \cup \{e\}) \ \text{ für alle } e \in E \setminus F.$$

Die Matroidtheorie kann man als eine gemeinsame Verallgemeinerung gewisser Aspekte der Graphentheorie und der linearen Algebra ansehen. Die beiden Beispiele, aus denen die Matroidtheorie entstanden ist, wollen wir daher zuerst vorstellen.

(5.8) Beispiele.

(a) **Graphische Matroide**

Das in (5.5) (b) definierte Unabhängigkeitssystem der Wälder eines Graphen $G = (V, E)$ ist ein Matroid. Ist $F \subseteq E$, so zerfällt (V, F) in Zusammenhangskomponenten $G_1 = (V_1, F_1), \ldots, G_k = (V_k, F_k)$. Die Basen der Zusammenhangskomponenten $G_1, \ldots, G_k$ sind die aufspannenden Bäume von $G_1, \ldots, G_k$. Jede Vereinigung von aufspannenden Bäumen von $G_1, \ldots, G_k$ ist eine Basis von F. Die Zirkuits von $\mathcal{I}$ sind die Kreise des Graphen G (daher der Name!). Der Rang in einer Menge $F \subseteq E$ ist gegeben durch

$$r(F) = |V(F)| - \text{Anzahl } k \text{ der Komponenten von } (V, F),$$

wobei $V(F)$ die Menge aller Knoten $v \in V$ bezeichnet, die in mindestens einer Kante aus F enthalten sind.

Die Matroide, die wie oben angegeben auf einem Graphen definiert werden können (bzw. isomorph zu solchen sind), heißen **graphische Matroide**.

(b) **Matrix-Matroide**

Sei $A = (a_{ij})$ eine (m, n)-Matrix über einem beliebigen Körper K mit Spaltenvektoren $A_{\cdot 1}, \ldots, A_{\cdot n}$. $E = \{1, \ldots, n\}$ sei die Menge der Spaltenindizes von A. Eine Teilmenge $F \subseteq E$ heißt unabhängig, wenn die Vektoren $A_{\cdot j}$, $j \in F$, linear unabhängig in K^m sind. Da jede Teilmenge einer linear unabhängigen Menge wiederum linear unabhängig ist, ist das so definierte Mengensystem $\mathcal{I}$ offenbar ein Unabhängigkeitssystem. Die Menge aller Basen von E ist die Menge aller $B \subseteq E$, so daß die Vektoren $A_{\cdot j}$, $j \in B$, eine Basis des durch die Spaltenvektoren von A aufgespannten linearen Teilraums von K^m bilden. Das Basisaxiom (B.1) ist in diesem Falle aufgrund des Steinitz'schen Austauschsatzes erfüllt. (Dieser Satz war die Motivation für (B.1)). Matroide, die auf die hier angegebene Weise konstruiert werden können, heißen **Matrix-Matroide**. Die Rangfunktion von $\mathcal{I}$ entspricht der aus der linearen Algebra bekannten Rangfunktion des K^m beschränkt auf die Spalten von A. Ist $\mathcal{I}$ ein Matroid auf E, und gibt es einen Körper K und eine (m, n)-Matrix A über K, so daß $\mathcal{I}$ zu dem Matrix-Matroid bezüglich A isomorph ist, dann heißt $\mathcal{I}$ **repräsentierbar** (über K). (Natürlich kann man hier etwas verallgemeinern und anstelle von Körpern Schiefkörper oder andere geeignete Objekte betrachten und Repräsentierbarkeit über diesen studieren.) Matroide, die über dem zweielementigen Körper $GF(2)$ repräsentierbar sind, heißen **binär**. Eine umfassende Untersuchung dieser wichtigen Klasse von Matroiden findet sich in Truemper (1992). Matroide, die über allen Körpern repräsentierbar sind, nennt man **regulär**. $\square$

Es folgt nun eine Liste weiterer interessanter Matroide.

(5.9) Beispiele.

(a) **Cographische Matroide**

Gegeben sei ein Graph $G = (V, E)$. Ein **Cokreis** ist eine Kantenmenge, deren Entfernung aus G die Komponentenzahl erhöht und die (mengeninklusionsweise) minimal bezüglich dieser Eigenschaft ist. Ein **Schnitt** ist eine Kantenmenge der Form

$$\delta(W) = \{ij \in E \mid i \in W, j \in V \setminus W\}, \ W \subseteq V.$$

Jeder Cokreis ist offenbar ein Schnitt, und es ist einfach einzusehen, daß die Cokreise gerade die minimalen nicht-leeren Schnitte von G sind. Sind $\delta(W)$ und $\delta(W')$ verschiedene Cokreise und ist die Kante ij in beiden enthalten, so ist $\delta(W \triangle W')$ ein Schnitt der ij nicht enthält. Ist $\delta(W \triangle W')$ kein Cokreis, so enthält er einen Cokreis, der natürlich ij auch nicht enthält. Daraus folgt, daß die Antikette der Cokreise das Axiom (C.1) erfüllt und somit das Zirkuitsystem eines Matroids auf E ist. Ein Matroid, das isomorph zu einem so definierten Matroid ist, heißt **cographisch**. Ist $G = (V, E)$ ein zusammenhängender Graph und M das cographische Matroid auf G, so ist das Basissystem $\mathcal{B}$ von M gegeben durch

$$\mathcal{B} = \{B \subseteq E \mid \exists \text{ aufspannender Baum } T \subseteq E \text{ mit } B = E \setminus T\}.$$

(b) **Uniforme Matroide**

Sei E eine Menge mit n Elementen, dann ist die Menge aller Teilmengen von E mit höchstens k Elementen ein Matroid auf E. Dieses Matroid heißt **uniform** und ist durch die Angabe von k und n bis auf Isomorphie eindeutig bestimmt. Dieses Matroid wird mit $U_{k,n}$ bezeichnet. Das Basissystem von $U_{k,n}$ wird durch die Menge der Teilmengen von E mit genau k Elementen gebildet. Das Zirkuitsystem von $U_{k,n}$ besteht aus den Teilmengen von E mit $k+1$ Elementen. Die Matroide $U_{n,n}$ (d. h. die Matroide, in denen alle Mengen unabhängig sind) heißen **frei** (oder trivial). Für freie Matroide gilt $\mathcal{C} = \emptyset$, $\mathcal{B} = \{E\}$, $r(F) = |F| \ \forall \ F \subseteq E$. Das uniforme Matroid $U_{2,4}$ ist das kleinste nicht binäre Matroid.

(c) **Partitionsmatroide**

Sei E eine endliche Menge, und $E_1, \ldots, E_k$ seien nicht-leere Teilmengen von E mit $E_i \cap E_j = \emptyset$, $i \neq j$, und $\bigcup_{i=1}^k E_i = E$. Seien $b_1, \ldots, b_k$ nicht-negative ganze Zahlen, dann ist

$$\mathcal{I} := \{I \subseteq E \mid |I \cap E_i| \leq b_i, i = 1, \ldots, k\}$$

ein Matroid auf E, genannt **Partitionsmatroid**.

(d) **Transversalmatroide**

Sei E eine endliche Menge, $(E_i, i \in I)$ sei eine endliche Familie von Teilmengen von E. Eine Teilmenge $T \subseteq E$ ist eine **teilweise Transversale** (oder partielles Repräsentantensystem) von $(E_i, i \in I)$, falls es eine Indexmenge $J \subseteq I$ gibt mit $|J| = |T|$ und eine Bijektion

$\pi : T \to J$, so daß $t \in E_{\pi(t)}$ für alle $t \in T$. Die Menge aller teilweisen Transversalen ist das Unabhängigkeitssystem eines Matroids auf E. (Dieses Ergebnis ist nicht trivial und hatte einen wesentlichen Einfluß auf die Transversaltheorie.)

$\square$

Wir wollen nun noch einige konkrete Beispiele von Matroiden angeben und ihre Basis-, Zirkuitsysteme etc. explizit auflisten.

Betrachten wir den folgenden in Abbildung 5.1 dargestellten Graphen $G = (V, E)$ mit $E = \{1, 2, \ldots, 13\}$. Das Zirkuitsystem $\mathcal{C}$ des graphischen Matroids auf E ist gegeben durch die Menge aller Kreise in G, d. h.

$$\mathcal{C} = \{\{1, 2, 3\}, \{4, 5, 6, 7\}, \{9, 10, 11\}, \{11, 12, 13\}, \{9, 10, 12, 13\}\}.$$

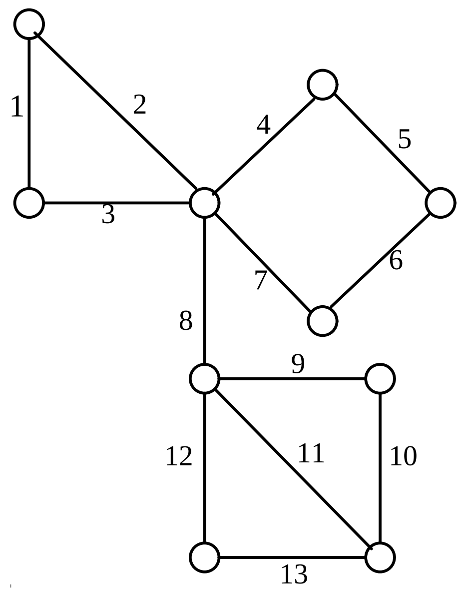

Abb. 5.1

Das Zirkuitsystem $\mathcal{C}^*$ des cographischen Matroids auf E ist gegeben durch die Menge aller minimalen Schnitte, d. h.

$$\begin{aligned}\mathcal{C}^* = &\{\{1, 2\}, \{1, 3\}, \{2, 3\}, \{4, 5\}, \{4, 6\}, \{4, 7\}, \{5, 6\}, \{5, 7\}, \{6, 7\},\\ &\{8\}, \{9, 10\}, \{9, 11, 12\}, \{9, 11, 13\}, \{10, 11, 12\}, \{10, 11, 13\}, \{12, 13\}\}\,.\end{aligned}$$

Das Basissystem $\mathcal{B}$ des graphischen Matroids des folgenden Graphen $G = (V, E)$ (siehe Abbildung 5.2) mit $E = \{1, 2, 3, 4\}$ ist gegeben durch

$$\mathcal{B} = \{\{1, 2, 3\}, \{1, 2, 4\}, \{1, 3, 4\}\},$$

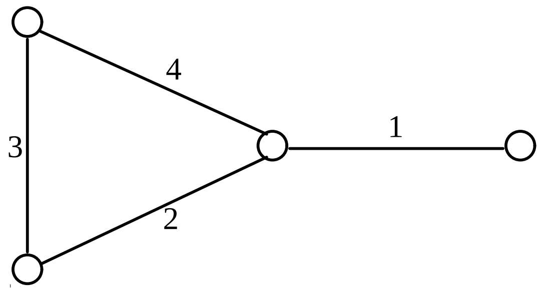

Abb. 5.2

und das Basissystem $\mathcal{B}^*$ des cographischen Matroids bezüglich G ist die folgende Antikette

$$\mathcal{B}^* = \{\{4\},\{3\},\{2\}\}.$$

Das graphische Matroid des in Abbildung 5.3 dargestellten Graphen ist das uniforme Matroid $U_{2,3}$. Uniforme Matroide können also auch isomorph zu graphischen sein. Das uniforme Matroid $U_{2,4}$ ist jedoch nicht graphisch.

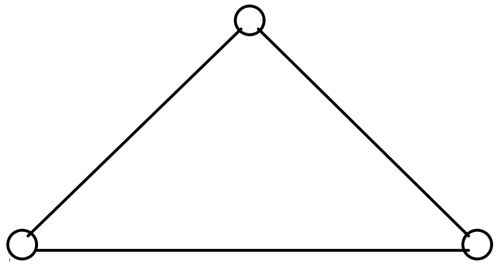

Abb. 5.3

Betrachten wir die Matrix

$$A = \begin{pmatrix} 1 & -1 & 1 & -1 \\ 0 & 1 & 1 & 0 \end{pmatrix}$$

als Matrix über dem Körper $\mathbb{R}$ oder $\mathbb{Q}$. Das Zirkuitsystem $\mathcal{C}$ des Matrix-Matroids M bezüglich A ist offenbar gegeben durch

$$\mathcal{C} = \{\{1,2,3\},\{2,3,4\},\{1,4\}\},$$

und das Basissystem $\mathcal{B}$ des Matrix-Matroids M ist

$$\mathcal{B} = \{\{1,2\},\{1,3\},\{2,3\},\{2,4\},\{3,4\}\}.$$

Wir wollen nun noch einen einfachen aber interessanten Zusammenhang zwischen Unabhängigkeitssystemen und Matroiden erwähnen.

(5.10) Satz. *Jedes Unabhängigkeitssystem ist als Durchschnitt von Matroiden darstellbar, d. h. ist $\mathcal{I}$ ein Unabhängigkeitssystem auf E, dann gibt es Matroide $\mathcal{I}_1, \ldots, \mathcal{I}_k$ auf E mit*

$$\mathcal{I} = \bigcap_{i=1}^{k} \mathcal{I}_i \,.$$

Beweis. Sei $\mathcal{C}$ das zu $\mathcal{I}$ gehörige Zirkuitsystem. Jedes Zirkuit $C \in \mathcal{C}$ definiert eine Antikette $\{C\} \subseteq 2^E$, die trivialerweise das Axiom (C.1) aus (5.7) erfüllt. Also ist das zu dem Zirkuitsystem $\{C\}$ gehörige Unabhängigkeitssystem $\mathcal{I}_C$ ein Matroid. Wir behaupten nun

$$\mathcal{I} = \bigcap_{C \in \mathcal{C}} \mathcal{I}_C .$$

Ist $I \in \mathcal{I}$, so ist kein Zirkuit $C \in \mathcal{C}$ in I enthalten, folglich ist $I \in \mathcal{I}_C$ für alle $C \in \mathcal{C}$. Sei umgekehrt $I \in \mathcal{I}_C$ für alle $C \in \mathcal{C}$, so heißt dies, daß kein Zirkuit $C \in \mathcal{C}$ in I enthalten ist, und somit, daß I ein Element von $\mathcal{I}$ ist. $\square$

Die im Beweis von Satz (5.10) angegebene Konstruktion zur Darstellung eines Unabhängigkeitssystems als Durchschnitt von Matroiden produziert i. a. eine riesige Zahl von Matroiden, die das Gewünschte leisten. Häufig kommt man mit viel weniger Matroiden aus.

Betrachten wir z. B. die Menge der Branchings $\mathcal{I} \subseteq 2^A$ in einem Digraphen $D = (V, A)$. Man kann einfach zeigen, daß das zugehörige Zirkuitsystem $\mathcal{C}$ aus den inklusionsminimalen Mengen der Vereinigung der folgenden Antiketten $\mathcal{C}_1$ und $\mathcal{C}_2$ besteht:

$$\begin{aligned}
\mathcal{C}_1 &:= \{C \subseteq A \mid |C| = 2, \text{ und die Endknoten der beiden Bögen in } C \text{ sind identisch}\},\\
\mathcal{C}_2 &:= \{C \subseteq A \mid C \text{ ist ein Kreis (die Bogenrichtungen spielen keine Rolle)}\}.
\end{aligned}$$

$\mathcal{C}_1$ ist das Zirkuitsystem eines Partitionsmatroids auf A, dessen Unabhängigkeitssystem gegeben ist durch $\{B \subseteq A \mid |B \cap \delta^-(v)| \leq 1 \ \forall \ v \in V\}$, und $\mathcal{C}_2$ ist das Zirkuitsystem des graphischen Matroids auf D (hierbei wird D als Graph aufgefaßt, d. h. die Bogenrichtungen werden ignoriert). Daraus folgt, daß das Unabhängigkeitssystem der Branchings Durchschnitt von 2 Matroiden ist. Für den vollständigen Digraphen D mit n Knoten hätte die Konstruktion im Beweis von Satz (5.10) insgesamt

$$n\binom{n-1}{2} + \sum_{k=2}^{n} \binom{n}{k}(k-1)!$$

verschiedene Matroide geliefert.

Das Unabhängigkeitssystem $\tilde{\mathcal{T}}$ der Teilmengen von Touren im vollständigen Digraphen D_n mit n Knoten, siehe (5.5) (d), ist als Durchschnitt von 3 Matroiden darstellbar (Hausaufgabe). Also ist das asymmetrische Travelling-Salesman-Problem als Optimierungsproblem über dem Durchschnitt von 3 Matroiden formulierbar.

5.3 Orakel

Um Eigenschaften von Matroiden oder Unabhängigkeitssystemen überprüfen zu können, muß man sich natürlich fragen, wie man Matroide geeignet darstellt, um z. B. ein Computerprogramm schreiben zu können, das Matroide als Input akzeptiert.

Betrachten wir zum Beispiel das in (5.3) formulierte Optimierungsproblem $\max\{c(I) \mid I \in \mathcal{I}\}$ über einem Unabhängigkeitssystem $\mathcal{I}$ auf E (Spezialfall: $(E, \mathcal{I})$ ist ein Matroid). Ist $\mathcal{I}$ als Liste aller unabhängigen Mengen gegeben, so ist das Optimierungsproblem völlig trivial. Wir durchlaufen die Liste, rechnen für jede Menge I der Liste den Wert $c(I)$ aus und wählen eine Menge I^*, so daß $c(I^*)$ maximal ist. Die Laufzeit dieses trivialen Algorithmus ist linear in der Inputlänge des Unabhängigkeitssystems.

Viele der Matroide und Unabhängigkeitssysteme, die wir in den voraufgegangenen Abschnitten definiert haben, sind jedoch in wesentlich kompakterer Form gegeben. Zum Beispiel ist ein Matrix-Matroid (5.8) (b) durch eine Matrix A (mit der Information, daß $I \subseteq E$ unabhängig genau dann ist, wenn die Spalten $A_{.i}$, $i \in I$, linear unabhängig sind) gegeben, ein graphisches Matroid (5.8) (a) durch einen Graphen $G = (V, E)$ (zusammen mit der Information, daß $I \subseteq E$ unabhängig ist genau dann, wenn I keinen Kreis enthält), ein cographisches Matroid (5.9) (a) durch einen Graphen $G = (V, E)$ (zusammen mit der Information, daß $C \subseteq E$ ein Zirkuit ist genau dann, wenn C ein Cokreis ist). Würden wir die Inputlänge des Matroids als die Länge der Kodierung der Matrix A (für (5.8) (b)) bzw. als die Länge der Kodierung des Graphen G (für (5.8) (a)) definieren, hätte unser trivialer Enumerationsalgorithmus exponentielle Laufzeit in der Kodierungslänge dieses kompakten Inputs.

Die Frage ist also: Was ist eine "geeignete" Kodierung eines Matroids? Motiviert durch die gerade angegebenen Beispiele könnte man glauben, daß die Angabe einer Liste aller unabhängigen Mengen eine unökonomische Methode sei. Jedoch geht es im allgemeinen nicht viel besser. Man kann nämlich zeigen, daß es mindestens

$$2^{2^{n/2}} \text{ Matroide mit } n \text{ Elementen}$$

gibt. Daraus folgt, daß es bei jeder beliebigen Kodierungsart immer Matroide gibt, deren Kodierung eine Länge hat, die mindestens $2^{n/2}$ beträgt.

Aufgrund dieser Tatsache hat sich ein anderes Konzept der Repräsentation von Matroiden durchgesetzt und als außerordentlich nützlich erwiesen. Matroide werden durch "Orakel" dargestellt.

Wir treffen folgende Definition. Die **Kodierungslänge eines Matroids** $M = (E, \mathcal{I})$ ist die Anzahl der Elemente von E. Das Matroid selbst ist in einem Orakel "versteckt", wobei das Orakel als ein Computerprogramm (Subroutine) interpretiert werden kann, das Fragen eines speziellen Typs beantwortet. Wir geben einige Beispiele (diese gelten natürlich nicht nur für Matroide sondern analog auch für Unabhängigkeitssysteme) und gehen davon aus, daß wir die Grundmenge E kennen.

(5.11) Beispiele für Orakel.

(a) **Unabhängigkeitsorakel**

Für jede Menge $F \subseteq E$ können wir das Orakel fragen, ob F unabhängig ist. Das Orakel antwortet mit "ja" oder "nein".

(b) **Zirkuitorakel**

Für jede Menge $F \subseteq E$ können wir das Orakel fragen, ob F ein Zirkuit ist. Das Orakel antwortet mit "ja" oder "nein".

(c) **Basisorakel**

Für jede Menge $F \subseteq E$ können wir das Orakel fragen, ob F eine Basis von E ist. Das Orakel antwortet mit "ja" oder "nein".

(d) **Rangorakel**

Für jede Menge $F \subseteq E$ können wir das Orakel fragen, wie groß der Rang von F ist. Die Antwort des Orakels ist "$r(F)$".

□

Man sieht sofort, daß dieses theoretische Orakelkonzept dem Konzept von Unterprogrammen der Programmierung entspricht.

Wenn wir also Algorithmen betrachten wollen, die Eigenschaften von Matroiden überprüfen, so gehen wir immer davon aus, daß ein Matroid (oder Unabhängigkeitssystem) durch die Grundmenge E und ein Orakel gegeben ist, das im Verlaufe des Algorithmus befragt werden kann. Bei der Komplexitätsanalyse von Algorithmen für Matroide (Unabhängigkeitssysteme) wird dann ein Orakelaufruf (Unterprogrammaufruf) als ein Schritt gezählt. Achtung! Die Laufzeit beim Lesen der Antwort wird wie üblich berechnet. Gibt also ein Orakel eine Antwort, die z. B. $2^{|E|}$ Speicherplätze benötigt, so hat der Algorithmus automatisch eine exponentielle Laufzeit. Bei den in (5.11) angegebenen Orakeln kommt so ein Fall allerdings nie vor! Man nennt Verfahren, die Orakel aufrufen können, **Orakelalgorithmen**. Ist ihre Laufzeit in dem oben angegebenen Sinne polynomial in $|E|$, so sagt man, daß die Verfahren **orakelpolynomial** sind.

Hat man einen Algorithmus, dessen Laufzeit orakelpolynomial in $|E|$ ist, so ist der Algorithmus für alle die Matroide (Unabhängigkeitssysteme) im üblichen Sinne polynomial, für die das Orakel durch einen in $|E|$ polynomialen Algorithmus realisiert werden kann.

Dies ist zum Beispiel für Matrix-Matroide der Fall. Ein Unabhängigkeitsorakel kann man bei einer gegebenen Matrix durch Rangbestimmung mit Gauß-Elimination realisieren (analog die drei anderen Orakel). Dies gilt auch für graphische und cographische Matroide, die durch einen Graphen $G = (V, E)$ gegeben sind. Zum Beispiel kann man die Unabhängigkeit einer Menge F im graphischen Matroid (F enthält keinen Kreis) durch Depth-First-Search testen; auch die übrigen drei Orakel kann man durch Algorithmen realisieren deren Laufzeit polynomial in $|E|$ ist.

Eine wichtige Frage ergibt sich sofort. Sind die verschiedenen Orakel algorithmisch "äquivalent"? Das heißt, kann man ein Orakel durch ein anderes Orakel in orakelpolynomialer Zeit simulieren und umgekehrt? Zum Beispiel: Ist ein Unabhängigkeitsorakel gegeben, kann man dann einen Algorithmus entwerfen, der nur dieses Orakel benutzt und orakelpolynomial ist, der für jede Menge $F \subseteq E$ korrekt entscheidet, ob F ein Zirkuit ist oder nicht?

Für Matroide und die vier oben angegebenen Orakel wurde diese Frage von Hausmann und Korte (1981) wie folgt beantwortet.

(5.12) Satz. *Sei $M = (E, \mathcal{I})$ ein beliebiges Matroid.*

(a) *Das Unabhängigkeitsorakel und das Rangorakel sind bezüglich M "äquivalent".*

(b) *Das Basisorakel und das Zirkuitorakel können durch das Unabhängigkeitsorakel sowie durch das Rangorakel in orakelpolynomialer Zeit simuliert werden.*

(c) *Es gibt keine anderen orakelpolynomialen Beziehungen zwischen diesen Orakeln.*

$\square$

Man kann diesen Satz graphisch durch Abbildung 5.4 veranschaulichen. Ein Pfeil in diesem Diagramm besagt, daß das Orakel am Ende des Pfeils durch polynomial viele Aufrufe des Orakels an der Spitze des Pfeils simuliert werden kann. Ist zwischen zwei Kästchen kein Pfeil (in irgendeiner der beiden Richtungen), so besagt dies auch, daß es keine orakelpolynomiale Simulation dieser Art gibt. Z. B. kann man das Basisorakel nicht durch das Zirkuitorakel in orakelpolynomialer Zeit simulieren und umgekehrt. Dies zeigt man dadurch, daß man eine Klasse von Beispielen angibt, bei denen zur Entscheidung der Frage, ob eine Menge I eine Basis (ein Zirkuit) ist, eine Anzahl von Aufrufen des Zirkuitorakels (Basisorakels) notwendig ist, die exponentiell in $|E|$ ist.

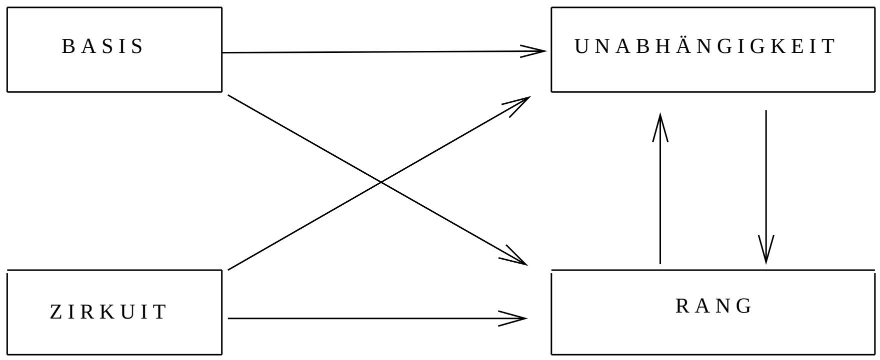

Abb. 5.4

Hausmann und Korte (1980) haben dieselbe Fragestellung auch für allgemeine Unabhängigkeitssysteme untersucht, die — wie wir in Abschnitt 5.1 gesehen haben — äquivalent durch Zirkuitsysteme, Basissysteme oder Rangfunktionen gegeben werden können. Der entsprechende Satz ist bildlich in Abbildung 5.5 veranschaulicht (Interpretation wie bei Abbildung 5.4).

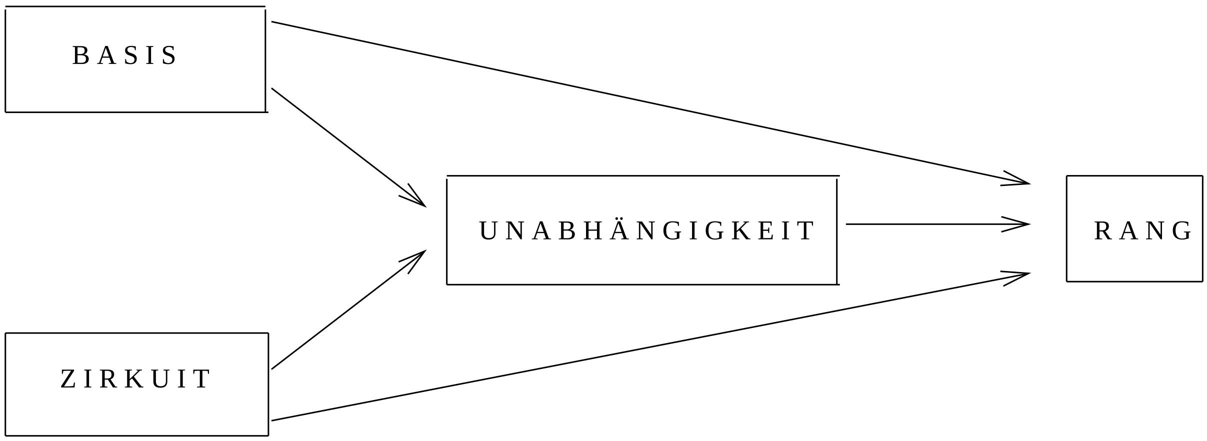

Abb. 5.5

Insbesondere folgt, daß bei Unabhängigkeitssystemen das Rangorakel das stärkste ist. Durch dieses lassen sich die übrigen Orakel in orakelpolynomialer Zeit simulieren. Jedoch sind hier keine zwei Orakel algorithmisch "äquivalent" (sie sind natürlich logisch äquivalent).

5.4 Optimierung über Unabhängigkeitssystemen

Wir wollen nun die Frage untersuchen, ob bzw. wie gut ein Optimierungsproblem der Form (5.3)

$$\max\{c(I) \mid I \in \mathcal{I}\},$$

wobei $\mathcal{I}$ ein Unabhängigkeitssystem auf einer Grundmenge E ist, gelöst werden kann. Wir betrachten dazu den folgenden trivialen Algorithmus, vergleiche (4.1).

(5.13) Greedy-Algorithmus für Unabhängigkeitssysteme.

Input: *Grundmenge* $E = \{1, \ldots, n\}$ *mit Gewichten* $c_i \in \mathbb{K}$ *für alle* $i \in E$. *Ferner ist ein Unabhängigkeitssystem* $\mathcal{I} \subseteq 2^E$ *durch ein Unabhängigkeitsorakel (siehe (5.11) (a)) gegeben.*

Output: *Eine unabhängige Menge* $I_g \in \mathcal{I}$.

1. *Sortiere die Gewichte in nicht aufsteigender Reihenfolge, (d. h. nach Beendigung von Schritt 1 können wir annehmen, daß* $c_1 \geq c_2 \geq \ldots \geq c_n$ *gilt).*
2. *Setze* $I := \emptyset$.
3. FOR $i = 1$ TO n DO:
   Ist $c_i \leq 0$, *gehe zu 4.*
   Ist $I \cup \{i\}$ *unabhängig (Orakelaufruf), dann setze* $I := I \cup \{i\}$.
4. *Setze* $I_g := I$ *und gib* I_g *aus.*

□

Der Greedy-Algorithmus durchläuft (nach der Sortierung in Schritt 1) die Elemente von E genau einmal, entweder er nimmt ein Element in die Menge I auf, oder er verwirft es für immer. Die am Ende des Verfahrens gefundene Lösung I_g nennen wir **Greedy-Lösung**. Für eine Menge $F \subseteq E$ setzen wir

$$r_u(F) := \min\{|B| \mid B \text{ Basis von } F\} \tag{5.14}$$

und nennen $r_u(F)$ den **unteren Rang von** F. Ist $(E, \mathcal{I})$ ein Matroid, so gilt nach (I.3″) natürlich $r_u(F) = r(F)$ für alle $F \subseteq E$. Das folgende Resultat wurde von Jenkyns (1976) bewiesen.

(5.15) Satz. *Sei $\mathcal{I}$ ein Unabhängigkeitssystem auf E, seien I_g eine Greedy-Lösung von (5.3) und I_0 eine optimale Lösung von (5.3), dann gilt*

$$\min_{F \subseteq E} \frac{r_u(F)}{r(F)} \leq \frac{c(I_g)}{c(I_0)} \leq 1,$$

und für jedes Unabhängigkeitssystem gibt es Gewichte $c_i \in \{0, 1\}$, $i \in E$, so daß die erste Ungleichung mit Gleichheit angenommen wird.

Beweis. Wir können o. B. d. A. annehmen, daß $c_i > 0$ für $i = 1, \ldots, n$ und daß $c_1 \geq c_2 \geq \ldots \geq c_n$ gilt. Aus technischen Gründen führen wir ein Element $n + 1$ ein mit $c_{n+1} = 0$. Wir setzen

$$E_i := \{1, \ldots, i\}, i = 1, \ldots, n + 1$$

und

$$q := \min_{F \subseteq E} \frac{r_u(F)}{r(F)}\,.$$

Es gilt dann offenbar

$$(1) \qquad c(I_g) = \textstyle\sum_{i=1}^n |I_g \cap E_i|(c_i - c_{i+1}),$$

$$(2) \qquad c(I_0) = \textstyle\sum_{i=1}^n |I_0 \cap E_i|(c_i - c_{i+1}).$$

Da $I_0 \cap E_i \subseteq I_0$, gilt $I_0 \cap E_i \in \mathcal{I}$, und somit $|I_0 \cap E_i| \leq r(E_i)$. Die Vorgehensweise des Greedy-Algorithmus impliziert, daß $|I_g \cap E_i|$ eine Basis von E_i ist, also daß $I_g \cap E_i \geq r_u(E_i)$ gilt. Daraus folgt

$$|I_g \cap E_i| \geq |I_0 \cap E_i| \frac{r_u(E_i)}{r(E_i)} \geq |I_0 \cap E_i|\, q, i = 1, \ldots, n$$

und somit

$$\begin{aligned} c(I_g) &= \sum_{i=1}^n |I_g \cap E_i|(c_i - c_{i+1}) \\ &\geq \sum_{i=1}^n (|I_0 \cap E_i|\, q)(c_i - c_{i+1}) \\ &= q \sum_{i=1}^n |I_0 \cap E_i|(c_i - c_{i+1}) \\ &= c(I_0) q. \end{aligned}$$

Die Ungleichung $1 \geq \frac{c(I_g)}{c(I_0)}$ ist trivial. Damit haben wir die Gültigkeit der Ungleichungskette bewiesen.

Sei nun $F \subseteq E$ mit $q = \frac{r_u(F)}{r(F)}$. Wir können annehmen, daß $F = \{1, \ldots, k\}$ gilt und daß $B = \{1, \ldots, p\} \subseteq F$ eine Basis von F ist mit $|B| = r_u(F)$. Wir setzen $c_i = 1$, $i = 1, \ldots, k$, und $c_i = 0$, $i = k+1, \ldots, n$. Dann liefert der Greedy-Algorithmus die Greedy-Lösung $I_g = B$ mit $c(I_g) = r_u(F)$ während für jede Optimallösung I_0 gilt $c(I_0) = r(F)$. Also wird für diese spezielle 0/1-Zielfunktion die linke Ungleichung in der Aussage des Satzes mit Gleichheit angenommen. $\square$

(5.16) Folgerung. *Sei $\mathcal{I}$ ein Unabhängigkeitssystem auf E. Dann sind aqüivalent:*

(a) $(E, \mathcal{I})$ ist ein Matroid.

(b) Für alle Gewichte $c \in \mathbb{K}^E$ liefert der Greedy-Algorithmus (5.13) eine Optimallösung von (5.3).

(c) Für alle Gewichte mit 0/1-Koeffizienten liefert der Greedy-Algorithmus (5.13) eine Optimallösung von (5.3).

$\square$

(5.16) wurde von verschiedenen Autoren unabhängig voneinander bewiesen. Edmonds (1971) zeigte insbesondere wie man den Greddy-Algorithmus für Matroide als ein Verfahren zur Lösung spezieller linearer Programme deuten kann. Er liefert in der Tat auch duale Lösungen. Wir können hier jedoch nicht auf diese Zusammenhänge eingehen.

Da der Algorithmus (4.1) GREEDY-MAX offenbar eine Spezialisierung des Greedy-Algorithmus (5.13) für das graphische Matroid ist, liefert Folgerung (5.16) einen weiteren Beweis von Satz (4.2).

Satz (5.15) ist ein Prototyp von Abschätzungssätzen für die Qualität von heuristischen Lösungen, wie wir sie später noch mehrfach kennenlernen werden. Der Greedy-Algorithmus (5.13) ist offenbar orakelpolynomial. Immer dann, wenn wir den Orakelaufruf (Unabhängigkeitstest) in Schritt 2 durch einen polynomialen Algorithmus realisieren können, ist der Greedy-Algorithmus ein polynomialer Algorithmus (im üblichen Sinne). Eine solche polynomiale Realisation ist z. B. auf triviale Weise für das Cliquenproblem, Stabile-Menge-Problem, Travelling-Salesman-Problem und azyklische Subdigraphenproblem möglich. Würde der Greedy-Algorithmus auch in diesen Fällen immer Optimallösungen liefern, hätten wir einen Beweis für $\mathcal{P}$=$\mathcal{NP}$ gefunden, da alle gerade aufgelisteten Probleme $\mathcal{NP}$-vollständig sind. Wir können also nicht erwarten, daß der Greedy-Algorithmus immer optimale Antworten produziert. Die Frage, wie schlecht im schlechtest möglichen Falle eine Greedy-Lösung ist, beantwortet Satz (5.15) auf bestmögliche Weise. Er gibt eine sogenannte **worst-case-Schranke** an, die besagt, daß jede Greedy-Lösung nicht schlechter ist als der Quotient

$$q := \min_{F \subseteq E} \frac{r_u(F)}{r(F)},$$

den man auch **Rangquotient** nennt. Etwas positiver ausgedrückt, der Rangquotient ist eine **Gütegarantie** für die Lösungsqualität des Greedy-Algorithmus. Der Rangquotient ist im allgemeinen nicht einfach auszurechnen. Eine Abschätzung wird gegeben durch.

(5.17) Satz. *Sei $\mathcal{I}$ ein Unabhängigkeitssystem auf E, und $(E, \mathcal{I}_i)$ $i = 1, \ldots, k$ sei eine minimale Zahl von Matroiden mit $\mathcal{I} = \bigcap_{i=1}^{k} \mathcal{I}_i$, dann gilt*

$$\min_{F \subseteq E} \frac{r_u(F)}{r(F)} \geq \frac{1}{k}.$$

□

Daraus folgt zum Beispiel: Ist $\mathcal{I}$ ein Unabhängigkeitssystem, das Durchschnitt von 2 Matroiden ist, dann ist der Wert der Greedy-Lösung mindestens halb so groß wie der Wert der Optimallösung von (5.3). Insbesondere liefert also der Greedy-Algorithmus für Branchingprobleme Lösungen, deren Wert mindestens die Hälfte des Optimums beträgt.

Für das Branching-Problem haben wir in Kapitel 4 einen polynomialen Lösungsalgorithmus kennengelernt. Es gibt also offenbar auch Probleme über Unabhängigkeitssystemen, die keine Matroide sind und für die effiziente Optimierungsverfahren existieren. Ein sehr tiefliegendes Resultat ist der folgende von Edmonds (1979) und Lawler (1975) gefundene Satz.

(5.18) Satz. *Seien $(E, \mathcal{I}_1)$ und $(E, \mathcal{I}_2)$ zwei Matroide gegeben durch Unabhängigkeitsorakel, dann gibt es einen Algorithmus, der für jede beliebige Zielfunktion c das Problem $\max\{c(I) \mid I \in \mathcal{I}_1 \cap \mathcal{I}_2\}$ in orakelpolynomialer Zeit löst.*

□

Der Algorithmus, der den Beweis des obigen Satzes liefert, ist relativ kompliziert, und sein Korrektheitsbeweis benötigt Hilfsmittel aus der Matroidtheorie, die uns hier nicht zur Verfügung stehen. Deshalb verzichten wir auf eine Angabe und Analyse dieses (sehr interessanten) Verfahrens.

Man fragt sich nun natürlich sofort, ob auch über dem Durchschnitt von 3 Matroiden in (orakel-)polynomialer Zeit optimiert werden kann. Dies geht (vermutlich) i. a. nicht. Man kann zeigen, daß das Optimierungsproblem für Unabhängigkeitssysteme, die Durchschnitte von 3 Matroiden sind, Probleme enthält, die $\mathcal{NP}$-vollständig sind (z. B. das asymmetrische TSP).

Wir wollen uns nun noch kurz dem Optimierungsproblem (5.4) über Basissystemen zuwenden. Wie bei Bäumen und Wäldern vorgeführt (siehe Abschnitt 4.1), kann man ein Maximierungsproblem des Typs (5.3) mit Hilfe einer polynomialen Transformation in ein Minimierungsproblem (5.4) überführen und umgekehrt. Analog läßt sich auch aus (5.13) ein Minimierungsalgorithmus ableiten.

(5.19) Greedy-Minimierungsalgorithmus für Basissysteme.

Input: *Grundmenge $E = \{1, \ldots, n\}$ mit Gewichten $c_i \in \mathbb{K}$ für alle $i \in E$, ein Unabhängigkeitssystem $\mathcal{I} \subseteq 2^E$ gegeben durch ein Unabhängigkeitsorakel.*

Output: *Eine Basis $B_g \in \mathcal{I}$.*

1. *Sortiere die Gewichte in nicht absteigender Reihenfolge, (nach Beendigung von Schritt 1 gelte $c_1 \leq c_2 \leq \ldots \leq c_n$).*

2. *Setze $I := \emptyset$.*
3. FOR $i = 1$ TO n DO*:*
   Ist $I \cup \{i\} \in \mathcal{I}$ (Orakelaufruf), dann setze $I := I \cup \{i\}$.
4. *Setze $B_g := I$ und gib B_g aus.*

□

Offenbar ist B_g eine Basis der Grundmenge E des Unabhängigkeitssystems $\mathcal{I}$, gehört also zum zugehörigen Basissystem $\mathcal{B}$. Folgerung (5.16) impliziert:

(5.20) Satz. *Sei $\mathcal{I}$ ein Unabhängigkeitssystem auf E und $\mathcal{B}$ das zugehörige Basissystem. Dann sind äquivalent:*

(a) $(E, \mathcal{I})$ ist ein Matroid.

(b) Für alle Gewichte $c \in \mathbb{K}^E$ liefert der Greedy-Algorithmus (5.19) eine Optimallösung von $\min\{c(B) \mid B \in \mathcal{B}\}$.

(c) Für alle Gewichte $c \in \{0,1\}^E$ liefert der Greedy-Algorithmus (5.19) eine Optimallösung von $\min\{c(B) \mid B \in \mathcal{B}\}$.

□

Sortiert man in Schritt 1. von (5.19) die Gewichte in nicht aufsteigender Reihenfolge, so erhält man offenbar eine Basis maximalen Gewichts. Satz (5.20) gilt analog, wenn man "min" durch "max" ersetzt.

Leider kann man die schöne Abschätzung aus Satz (5.15) für die Gütegarantie von (5.13) nicht ohne weiteres auf Algorithmus (5.19) übertragen. In der Tat gibt es keine universelle Konstante, die bezüglich eines Problems die Qualität des Ergebnisses von (5.19) unabhängig von den Zielfunktionskoeffizienten beschränkt. Betrachten wir z. B. das Cliquen-Problem auf dem in Abbildung 5.6 dargestellten Graphen

Abb. 5.6

mit der Gewichtsfunktion $c_1 = 1$, $c_2 = M > 2$, $c_3 = 2$. Das zugehörige Unabhängigkeitssystem hat die Basen $\{1,2\}$ und $\{3\}$. Bei Anwendung von (5.19) erhalten wir immer die Greedylösung $B_g = \{1,2\}$ mit Gewicht $1 + M$, während die Optimalbasis B_0 das Gewicht 2 hat. Der Quotient aus B_g und B_0 geht also gegen ∞ falls $M \to \infty$.

5.5 Ein primal-dualer Greedy-Algorithmus

Die Matroidtheorie ist aufgrund ihres technischen Apparates besonders gut dazu geeignet, eine gemeinsame Verallgemeinerung des primalen und des dualen Greedy-Algorithmus zur Berechnung minimaler aufspannender Bäume (siehe (4.3) und (4.5)) zu formulieren. Grundlegend hierfür ist der folgende

(5.21) Satz und Definition. *Sei M ein Matroid auf einer Grundmenge E mit Basissystem $\mathcal{B}$. Setze $\mathcal{B}^* := \{E \setminus B \mid B \in \mathcal{B}\}$. Dann ist $\mathcal{B}^*$ das Basissystem eines Matroids M^* auf E. M^* wird das zu M* **duale Matroid** *genannt.*

□

Offensichlich gilt für ein Basissystem

$$B^{**} = \mathcal{B}$$

d. h. das zu M^* duale Matroid ist das Matroid M. Wir haben bereits ein Matroid und das dazu duale Matroid kennengelernt. Ist nämlich M das graphische Matroid auf einem Graphen G (siehe (5.8)(a)), so ist das cographische Matroid auf G (siehe (5.9)(a)) das zu M duale Matroid M^*.

Für Matrix-Matroide (siehe (5.8)(b)) kann man auf einfache Weise das duale Matroid konstruieren. Wir nehmen an, daß M durch die $m \times n$-Matrix A (über irgendeinem Körper) definiert ist. O.B.d.A. können wir annehmen, daß A vollen Zeilenrang m hat. Mit Hilfe der Gauß-Elimination bringen wir A in **Standardform**, d. h. wir formen A so um, daß $A = (I, B)$ gilt, wobei I die $m \times m$-Einheitsmatrix ist. (Möglichweise sind hierzu Zeilen- und Spaltenverbindungen erforderlich.) Man kann nun zeigen, daß das zu M duale Matroid M^* durch die Matrix

$$(-B^T, I)$$

definiert ist. Hierbei ist I eine $(n-m, n-m)$-Einheitsmatrix.

In der Matroidtheorie hat sich die folgende Sprechweise eingebürgert. Ist M ein Matroid und A eine Basis oder ein Zirkuit des dualen Matroids M^*, so nennt man A auch eine **Cobasis** oder ein **Cozirkuit** von M. Analog heißt der Rang von A bezüglich der Rangfunktion von M^* auch der **Corang** von A. Ein **Zyklus** ist die Vereinigung von paarweise disjunkten Zirkuits, analog ist ein **Cozyklus** die Vereinigung paarweise disjunkter Cozirkuits. Die nachfolgenden (einfachen) Beobachtungen sind für die nachfolgenden Betrachtungen wichtig.

(5.22) Satz.

(a) Ein System $\mathcal{C}$ von Teilmengen von E ist genau dann die Zirkuitmenge eines Matroids M auf E, wenn $\mathcal{C}$ genau die minimalen, nichtleeren Teilmengen $C \subseteq E$ enthält, so daß gilt: $|C \cap C^*| \neq 1$ für alle Cozirkuits C^* von M.

(b) Sei B eine Basis (Cobasis) eines Matroids M, und sei $e \in E \setminus B$, dann enthält $B \cup \{e\}$ genau ein Zirkuit (Cozirkuit) $\mathcal{C}_e$, der e enthält. C_e heißt der durch e und B erzeugte fundamentale **Zirkuit** *(Cozirkuit).*

□

(5.23) Satz. *Sei M ein Matroid auf E mit Gewichten c_e für alle $e \in E$. Sei B eine Basis von M. Dann sind die folgenden Aussagen äquivalent:*

(i) B ist eine gewichtsminimale Basis.

(ii) Für jedes Element $e \in E \setminus B$ gilt

$c_e \geq c_f$ *für alle* f *aus dem von* e *und* B *erzeugten fundamentalen Zirkuit* K_e.

(iii) Für jeden Cozirkuit C *gilt*

$$\min\{c_e \mid e \in C\} = \min\{c_e \mid e \in B \cap C\}$$

(iv) $E \setminus B$ *ist eine gewichtsmaximale Cobasis.*

(v) Für jedes Element $e \in B$ *gilt*

$c_e \leq c_f$ *für alle* f *aus dem von* e *und* $E \setminus B$ *erzeugten fundamentalen Cozirkuit* C_e.

(vi) Für jeden Zirkuit K *gilt*

$$\max\{c_e \mid e \in K\} = \max\{c_e \mid e \in K \cap (E \setminus B)\}.$$

Beweis. (i) $\Leftrightarrow$ (iv). Trivial.

(i) $\Rightarrow$ (ii). Angenommen, es existieren $e \in E \setminus B$ und $f \in K_e$ mit $c_e < c_f$. Dann ist $(B \setminus \{f\}) \cup \{e\}$ eine Basis mit kleinerem Gewicht als B. Widerspruch !

(ii) $\Rightarrow$ (iii). Angenommen, es existieren ein Cozirkuit C und $e \in C \setminus B$ mit $c_e < \min\{c_f \mid f \in B \cap C\}$. Sei K_e der durch e mit B erzeugte fundamentale Zirkuit. Dann existiert nach (5.22)(a) ein Element $f \in (K_e \setminus \{e\}) \cap C$. Aber dann gilt für $e \in E \setminus B$ und $f \in B : c_e < c_f$. Widerspruch!

(iii) $\Rightarrow$ (i). Sei B' eine gewichtsminimale Basis, so daß $|B \cap B'|$ so groß wie möglich ist. Angenommen, es existiert ein Element, $f \in B' \setminus B$. Sei $\mathcal{C}_f$ der von f mit $E \setminus B'$ erzeugte fundamentale Cozirkuit. Da B eine Basis ist, gilt $B \cap C_f \neq \emptyset$. Sei $g \in C_f \cap B$ mit $c_g = \min\{c_e \mid e \in C_f \cap B\}$. Nach (iii) gilt wegen $f \in C_f : c_f \geq c_g$. Nun aber ist $B'' := (B' \setminus \{f\}) \cup \{g\}$ eine Basis von M mit $c(B'') \leq c(B')$ und $|B \cap B''| > |B \cap B'|$. Widerspruch ! Daraus folgt $B = B'$, und somit ist B gewichtsminimal.

(iv) $\Rightarrow$ (v) $\Rightarrow$ (vi) $\Rightarrow$ (iv) folgt aus Dualitätsgründen. $\square$

(5.24) Primal-dualer Greedy-Algorithmus.

Input: *Grundmenge E mit Gewichten c_e für alle $e \in E$ und ein Matroid $M = (E, \mathcal{I})$ durch ein Unabhängigkeitsorakel.*

Output: *eine gewichtsminimale Basis B von M und eine gewichtsmaximale Cobasis B^* von M.*

(Terminologie: Wir nennen eine Menge S durch eine Menge F **überdeckt**, falls $F \cap S \neq \emptyset$.)

1. *Setze $B := B^* := \emptyset$.*
2. *Führe einen (beliebigen) der beiden folgenden Tests aus:*
   (a) *Sind alle Cozirkuits von M durch B überdeckt, STOP.*
   (Gib B und $B^* := E \setminus B$ aus.)
   (b) *Sind alle Zirkuits von M durch B^* überdeckt, STOP.*
   (Gib B^* und $B := E \setminus B^*$ aus.)
3. *Führe eine (beliebige) der folgenden beiden Operationen (a), (b) aus:*
   (a) (i) *Wähle einen nicht-leeren Cozyklus C von M, der nicht durch B überdeckt ist.*
   (ii) *Bestimme $f \in C$ mit $c_f = \min\{c_e \mid e \in C\}$*
   (iii) *Setze $B := B \cup \{f\}$ und gehe zu 2.*
   (b) (i) *Wähle einen nicht-leeren Zyklus Z von M, der nicht durch B^* überdeckt ist.*
   (ii) *Bestimme $g \in Z$ mit $c_g = \max\{c_e \mid e \in Z\}$*
   (iii) *Setze $B^* := B^* \cup \{g\}$ und gehe zu 2.*

(5.25) Satz. *Der obige Algorithmus (5.24) funktioniert.*

Beweis. Wir beweisen durch Induktion nach $|B| + |B^*|$, daß die jeweils während des Ablaufs des Verfahrens konstruierten Mengen B und B^* in einer minimalen Basis bzw. maximalen Cobasis enthalten sind.
Für den Induktionsanfang $B = B^* = \emptyset$ ist die Behauptung trivial.

Wir nehmen an, daß der Satz gilt, wenn $|B| + |B^*| \leq k$, d.h., wenn wir Schritt 2 und 3 höchstens k-mal ausgeführt haben. Wir beginnen jetzt mit der $(k+1)$-ten Ausführung von Schritt 3 und führen (a) aus. Sei B die bisher konstruierte Menge und B' eine gewichtsminimale Basis mit $B \subseteq B'$. Wir wählen in (i) einen Cozyklus C und in (ii) ein Element $f \in C$. Gilt $f \in B'$, so ist alles klar. Anderenfalls sei $g \in C \cap B'$ ein Element, das mit f auf einem Cozirkuit liegt. Dann muß wegen (ii) gelten $c_g \geq c_f$. $B'' := (B' \setminus \{g\}) \cup \{f\}$ ist damit eine Basis mit $c(B'') \leq c(B')$ und $\{f\} \cup B \subseteq B''$. Also ist $B \cup \{f\}$ in einer gewichtsminimalen Basis enthalten.

Führen wir (b) aus, so folgt die Behauptung analog.

Stellen wir in 2 (a) fest, daß B eine Basis ist, so ist nach Induktion $c(B)$ minimal und some $E \setminus B$ eine maximale Cobasis. Analog schließen wir im Falle, daß B^* in 2 (b) als Cobasis erkannt wird.

$\square$

Algorithmus (5.24) ist aufgrund der vielen Freiheitsgrade ein äußerst flexibel einsetzbares Instrument zur Konstruktion optimaler Basen und Cobasen.

Literaturverzeichnis von Kapitel 5

F. Barahona und M. Grötschel (1986), "On the cycle polytope of a binary matroid", Journal of Combinatorial Theory B 40, 40–62.

R. E. Bixby und W. H. Cunningham (1995), "Matroid Optimization and Algorithms", R. L. Graham et al. (Hrsg.) *Handbook of Combinatorics*, Vol. I, Kap. 11 , 551–609 North-Holland, Amsterdam.

J. Edmonds (1971), "Matroids and the greedy algorithm", Mathematical Programming 1 (1971) 127–136.

J. Edmonds (1979), "Matroid intersection", Annals of Discrete Mathematics 4 (1979) 39–49.

M. Grötschel und K. Truemper (1989), "Decomposition and optimization over cycles in binary matroids", Journal of Combinatorial Theory B 46, 306–337.

D. Hausmann und B. Korte (1980), "The relative strength of oracles for independence systems, in: J. Frehse, D. Pallaschke and U. Trottenberg, eds., Special topics of applied mathematics, functional analysis, numerical analysis, and optimization", North-Holland, Amsterdam (1980) 195–211.

D. Hausmann und B. Korte (1981), "Algorithmic versus axiomatic definitions of matroids", Mathematical Programming Study 14 (1981) 98–111.

T. A. Jenkyns (1976), "The Efficacy of the Greedy Algorithm, in: F. Hoffman, L. Lesniak, R. Mullin et.al. (Eds.):", Proceedings of the 7th Southeastern Conference on Combinatorics, Graph Theory and Computing, Baton Rouge .

E. L. Lawler (1975), "Matroid intersection algorithms", Mathematical Programming 9 (1975) 31–56.

J. G. Oxley (1992), *Matroid Theory,* Oxford University Press, Oxford.

K. Truemper (1992), *Matroid Decomposition,* Academic Press, Boston .

D. J. A. Welsh (1976), *Matroid Theory,* Academic Press, London.

D. J. A. Welsh (1995), "Matroids: Fundamental Concepts", R. L. Graham et al. (Hrsg.) *Handbook of Combinatorics*, Vol. I, Kap. 9 , 481–526 North-Holland, Amsterdam.

Kapitel 6

Kürzeste Wege

Wir wollen uns nun mit der Aufgabe beschäftigen, in einem Digraphen mit Bogengewichten einen kürzesten gerichteten Weg von einem Knoten zu einem anderen oder allen anderen zu finden. Wir beschränken uns auf Digraphen, da derartige Probleme in ungerichteten Graphen auf einfache Weise auf gerichtete Probleme reduziert werden können. Denn ist ein Graph $G = (V, E)$ mit Kantenlängen $c(e) \geq 0$ für alle $e \in E$ gegeben, so ordnen wir diesem den Digraphen $D = (V, A)$ mit $A = \{(i, j), (j, i) \mid ij \in E\}$ und $c((i, j)) := c((j, i)) := c(ij)$ zu. Den (ungerichteten) $[u, v]$-Wegen in G entsprechen dann die gerichteten (u, v)-Wege bzw. (v, u)-Wege in D und umgekehrt. Einander entsprechende Wege in G und D haben nach Definition gleiche Längen. Also liefert uns ein kürzester (u, v)-Weg (oder ein kürzester (v, u)-Weg) in D einen kürzesten $[u, v]$-Weg in G.

Kürzeste-Wege-Probleme spielen in der kombinatorischen Optimierung eine große Rolle. Es ist daher nicht überraschend, daß es zu diesem Problemkreis eine außerordentlich umfangreiche Literatur und sehr viele Lösungsvorschläge gibt. Wenn man dann noch Variationen hinzunimmt wie: Berechnung längster Wege, zuverlässiger Wege, Wege maximaler Kapazität, der k kürzesten Wege, Wege mit gerader oder ungerader Bogenzahl etc., so liefert das den Stoff einer gesamten Vorlesung. Wir wollen in dieser Vorlesung lediglich drei Algorithmen (für unterschiedliche Spezialfälle) behandeln. Der Leser, der sich für umfassendere Darstellungen interessiert, sei auf die Bücher Ahuja, Magnanti und Orlin (1993), Lawler (1976), Mehlhorn (1984), Domschke (1972), Syslo, Deo und Kowalik (1983) verwiesen. Es werden derzeit immer noch neue Algorithmen oder Modifikationen bekannter Algorithmen entdeckt, die aus theoretischer oder praktischer Sicht schneller als die bekannten Verfahren sind oder sonstige Vorzüge haben.

Es gibt keinen Algorithmus zur Bestimmung eines kürzesten (s, t)-Weges, der nicht (zumindest implizit) auch alle übrigen kürzesten Wege von s nach v, $s \neq v \neq t$, berechnet. Die Algorithmen für Kürzeste-Wege-Probleme kann man in zwei Kategorien einteilen, und zwar solche, die negative Bogenlängen zulassen, und solche, die nur nichtnegative Bogenlängen behandeln können. Von jedem der beiden Typen stellen wir einen Vertreter vor. Ferner wollen wir noch einen Algorithmus behandeln, der die kürzesten Wege zwischen allen Knoten berechnet.

6.1 Ein Startknoten, nichtnegative Gewichte

Das Verfahren, das wir nun darstellen wollen, ist mehrfach entdeckt worden. Es wird allgemein nach E. W. Dijkstra (1959) benannt. Wir gehen davon aus, daß ein Digraph $D = (V, A)$ mit "Gewichten" bzw. "Längen" oder "Entfernungen" $c(a) \geq 0$ für alle $a \in A$

gegeben ist. Ferner seien ein Startknoten s und möglicherweise ein Endknoten t gegeben. Das Verfahren findet einen kürzesten gerichteten Weg von s zu allen anderen Knoten bzw. einen kürzesten (s,t)-Weg.

Der Algorithmus wird häufig als **Markierungsmethode** bezeichnet. (Warum, wird aus dem weiteren klar.) Seine Idee kann man wie folgt beschreiben.

Wir beginnen im Startknoten s, markieren s und ordnen s die **permanente Distanz** Null (= Länge des kürzesten Weges von s zu sich selbst) zu. Alle übrigen Knoten v seien unmarkiert und erhalten als **temporäre Distanz** (= Länge des kürzesten bisher gefundenen (s,v)-Weges) $+\infty$ oder die Länge des Bogens (s,v), falls dieser in D existiert, zugeordnet. Der unmarkierte Knoten, sagen wir u, mit der kleinsten temporären Distanz ist der Knoten, der am nächsten zu s liegt. Da alle Bogenlängen nicht-negativ sind, ist der Bogen (s,u) der küzeste Weg von s nach u. Wir markieren daher u und erklären die temporäre Distanz von u als **permanent**, weil wir den (global) kürzesten (s,u)-Weg gefunden haben. Nun bestimmen wir alle Nachfolger v von u und vergleichen die temporäre Distanz von v mit der permanenten Distanz von u plus der Länge des Bogens (u,v). Ist diese Summe kleiner als die bisherige temporäre Distanz, wird sie die neue temporäre Distanz, weil der bisher bekannte Weg von s nach v länger ist als der Weg von s über u nach v. Wir wählen nun wieder eine kleinste der temporären Distanzen, erklären sie als permanent, da der bisher gefundene Weg durch Umwege über andere Knoten nicht verkürzt werden kann, markieren den zugehörigen Knoten und fahren so fort bis entweder alle Knoten oder der gesuchte Endknoten t markiert sind. Etwas formaler kann man diesen Algorithmus wie folgt aufschreiben.

(6.1) DIJKSTRA-Algorithmus.

Input: *Digraph $D=(V,A)$, Gewichte $c(a)\geq 0$ für alle $a\in A$, ein Knoten $s\in V$ (und möglicherweise ein Knoten $t\in V\setminus\{s\}$).*

Output: *Kürzeste gerichtete Wege von s nach v für alle $v\in V$ und ihre Länge, (bzw. ein kürzester (s,t)-Weg).*

Datenstrukturen:

DIST(v) *(= Länge des kürzesten (s,v)-Weges),*

VOR(v) *(= Vorgänger von v im kürzesten (s,v)-Weg).*

1. *Setze:*

$$\begin{array}{ll} \mathrm{DIST}(s) := 0 & \\ \mathrm{DIST}(v) := c((s,v)) & \forall\, v\in V \text{ mit } (s,v)\in A \\ \mathrm{DIST}(v) := +\infty & \forall\, v\in V \text{ mit } (s,v)\notin A \\ \mathrm{VOR}(v) := s & \forall\, v\in V\setminus\{s\} \end{array}$$

   Markiere s, alle übrigen Knoten seien unmarkiert.

2. *Bestimme einen unmarkierten Knoten u, so daß* $\mathrm{DIST}(u)=\min\{\mathrm{DIST}(v)\mid v$ *unmarkiert*$\}$. *Markiere u. (Falls $u=t$, gehe zu 5.)*

3. *Für alle unmarkierten Knoten v mit $(u, v) \in A$ führe aus:*

$$\text{Falls } \quad \mathrm{DIST}(v) > \mathrm{DIST}(u) + c((u,v)) \text{ setze:}$$
$$\mathrm{DIST}(v) := \mathrm{DIST}(u) + c((u,v)) \text{ und } \mathrm{VOR}(v) := u.$$

4. *Sind noch nicht alle Knoten markiert, gehe zu 2.*
5. *Für alle markierten Knoten v ist $\mathrm{DIST}(v)$ die Länge eines kürzesten (s, v)-Weges. Falls v markiert ist und $\mathrm{DIST}(v) < +\infty$, so ist $\mathrm{VOR}(v)$ der Vorgänger von v in einem kürzesten (s, v)-Weg, d. h. durch Rückwärtsgehen bis s kann ein kürzester (s, v)-Weg bestimmt werden. (Brechen wir das Verfahren nicht in Schritt 2 ab und gilt am Ende $\mathrm{DIST}(v) = +\infty$, so heißt das, daß es in D keinen (s, v)-Weg gibt.)*

$\square$

Zur Notationsvereinfachung für den nachfolgenden Beweis bezeichnen wir mit $\mathrm{DIST}_k(v)$ den Wert der in (6.1) berechneten Distanzfunktion nach dem k-ten Durchlaufen der Schritte 2, 3 und 4. Für den DIJKSTRA-Algorithmus gilt aufgrund der Auswahlvorschrift nach der k-ten Markierungsphase folgendes. Sind die Knoten in der Reihenfolge $v_1, v_2, \ldots, v_k$ markiert worden, so gilt $\mathrm{DIST}_k(v_1) \leq \ldots \leq \mathrm{DIST}_k(v_k) \leq \mathrm{DIST}_k(v)$ für alle bisher unmarkierten Knoten v.

(6.2) Satz. *Der Dijkstra-Algorithmus arbeitet korrekt.*

Beweis. Wir zeigen durch Induktion über die Anzahl k markierter Knoten folgendes. Ist v markiert, so enthält $\mathrm{DIST}_k(v)$ die Länge eines kürzesten (s, v)-Weges, ist v unmarkiert, so enthält $\mathrm{DIST}_k(v)$ die Länge eines kürzesten (s, v)-Weges, wobei als innere Knoten nur markierte Knoten zugelassen sind. (Falls $\mathrm{DIST}_k(v) = +\infty$, so wird dies als Nichtexistenz eines (s, v)-Weges bzw. eines (s, v)-Weges über markierte innere Knoten interpretiert). Hieraus folgt offenbar die Behauptung.

Ist nur ein Knoten (also s) markiert, so ist unsere Behauptung aufgrund der Definition in Schritt 1 korrekt. Wir nehmen nun an, daß die Behauptung richtig ist für k markierte Knoten und daß das Verfahren in Schritt 2 einen $(k+1)$-sten Knoten, sagen wir u, markiert und Schritt 3 durchlaufen hat. Nach Induktionsvoraussetzung ist $\mathrm{DIST}_k(u)$ die Länge eines kürzesten (s, u)-Weges, der als innere Knoten nur die ersten k markierten Knoten benutzen darf. Gäbe es eine kürzeren gerichteten Weg, sagen wir P, von s nach u, so müßte dieser einen Bogen von einem markierten Knoten zu einem bisher nicht markierten Knoten enthalten. Sei (v, w) der erste derartige Bogen auf dem Weg P. Der Teilweg $\overline{P}$ des Weges P von s nach w ist also ein (s, w)-Weg, dessen innere Knoten markiert sind. Folglich gilt nach Induktionsvoraussetzung $\mathrm{DIST}_{k+1}(w) \leq c(\overline{P})$. Aus $\mathrm{DIST}_{k+1}(u) \leq \mathrm{DIST}_{k+1}(w)$ und der Nichtnegativität der Bogenlängen folgt $\mathrm{DIST}_{k+1}(u) \leq c(\overline{P}) \leq c(P)$, ein Widerspruch.

Es bleibt noch zu zeigen, daß für die derzeit unmarkierten Knoten v der Wert $\mathrm{DIST}_{k+1}(v)$ die Länge eines kürzesten (s, v)-Weges ist, der nur markierte innere Knoten enthalten darf. Im Update-Schritt 3 wird offenbar die Länge eines (s, v)-Weges über markierte Knoten verschieden von u verglichen mit der Länge eines (s, v)-Weges über markierte Knoten, der als vorletzten Knoten den Knoten u enthält. Angenommen es gibt einen (s, v)-Weg P über markierte Knoten (inclusive u), dessen vorletzter Knoten w verschieden von u ist und dessen Länge geringer ist als die kürzeste Länge der oben betrachteten Wege. Da $\text{DIST}_{k+1}(w)$ die Länge eines kürzesten (s, w)-Weges ist und es einen solchen, sagen wir P', gibt, der nur markierte Knoten enthält, die verschieden von u sind (w wurde vor u markiert), kann der (s, w)-Weg auf P nicht kürzer als P' sein, also ist P nicht kürzer als die Länge von $P' \cup \{(w, v)\}$. Widerspruch. □

In der Datenstruktur VOR merken wir uns zu jedem Knoten v seinen Vorgänger in einem kürzesten (s, v)-Weg. Einen kürzesten (s, v)-Weg erhält man also in umgekehrter Reihenfolge durch die Knotenfolge

$$\text{VOR}(v), \text{VOR}(\text{VOR}(v)), \ldots, \text{VOR}(\text{VOR}(\ldots \text{VOR}(v) \ldots))$$

Durch VOR ist offenbar eine Arboreszenz mit Wurzel s in D definiert. Daraus folgt sofort:

(6.3) Satz. *Sei $D = (V, A)$ ein Digraph mit nichtnegativen Bogengewichten und $s \in V$, dann gibt es eine Arboreszenz B mit Wurzel s, so daß für jeden Knoten $v \in V$, für den es einen (s, v)-Weg in D gibt, der (eindeutig bestimmte) gerichtete Weg in B von s nach v ein kürzester (s, v)-Weg ist.*

□

An dieser Stelle sei darauf hingewiesen, daß der PRIM-Algorithmus (4.8) und der DIJKSTRA-Algorithmus (6.1) (im wesentlichen) identische Algorithmen sind. Sie unterscheiden sich lediglich bezüglich einer Gewichtstransformation. In Schritt 3 von (4.8) wird $\min\{c(e) \mid e \in \delta(W)\}$ gesucht, in Schritt 2 von (6.1) wird auch ein derartiges Minimum gesucht, jedoch sind vorher in Schritt 3 die Gewichte der Bögen des Schnittes modifiziert worden.

Den DIJKSTRA-Algorithmus kann man ohne Schwierigkeiten so implementieren, daß seine Laufzeit $O(|V|^2)$ beträgt. Bei Digraphen mit geringer Bogenzahl kann die Laufzeit durch Benutzung spezieller Datenstrukturen beschleunigt werden, siehe hierzu z.B. Ahuja, Magnanti und Orlin (1993).

6.2 Ein Startknoten, beliebige Gewichte

Das Problem, einen kürzesten Weg in einem Digraphen mit beliebigen Bogengewichten zu bestimmen, ist trivialerweise äquivalent zum Problem, einen längsten Weg in einem Digraphen mit beliebigen Bogengewichten zu finden. Gäbe es für das letztere Problem einen polynomialen Algorithmus, so könnte man in polynomialer Zeit entscheiden, ob ein Digraph einen gerichteten hamiltonschen Weg enthält. Dieses Problem ist aber $\mathcal{NP}$-vollständig, also ist das Kürzester-Weg-Problem für beliebige Gewichte $\mathcal{NP}$-schwer.

Andererseits kann man dennoch in beliebig gewichteten Digraphen kürzeste Wege finden, wenn die negativen Gewichte “gut verteilt” sind oder der Digraph bestimmte Eigenschaften hat. Der DIJKSTRA-Algorithmus funktioniert bei negativen Gewichten nicht (im Induktionsschritt des Beweises von (6.2) wurde von der Nichtnegativität explizit Gebrauch gemacht). Wir wollen nun auf ein Verfahren eingehen, das unabhängig voneinander von Moore (1959) und Bellman (1958) vorgeschlagen wurde. Zu diesem Verfahren gibt es eine Vielzahl von Verbesserungsvorschlägen (siehe hierzu z. B. Lawler (1976), Syslo, Deo und Kowalik (1983) oder Glover, Klingman und Phillips (1985)).

Die Idee hinter diesem Verfahren läßt sich wie folgt beschreiben. Wir wollen vom Startknoten s aus zu allen anderen Knoten v einen kürzesten (s, v)-Weg bestimmen. Wir initialisieren DIST(v) wieder mit $+\infty$ oder mit $c((s, v))$ (DIST(v) enthält also die Länge des kürzesten zur Zeit bekannten (s, v)-Weges mit einer bestimmten Eigenschaft) und setzen wie in (6.1) VOR(v) $= s$. Nun versuchen wir sukzessive die Distanzen DIST(v) zu reduzieren. Finden wir einen Bogen (u, v) mit $\text{DIST}(u) + c((u, v)) < \text{DIST}(v)$, so setzen wir $\text{DIST}(v) := \text{DIST}(u) + c((u, v))$ und $\text{VOR}(v) := u$. Wir führen diese Iteration so lange fort, bis kein Wert DIST(u) mehr reduziert werden kann. Die verschiedenen Versionen des Moore-Bellman-Algorithmus unterscheiden sich durch die Art wie diese Basisiteration ausgeführt wird (d. h. in welcher Reihenfolge die Knoten und Bögen (u. U. mehrfach) abgearbeitet werden).

Wir wollen uns zunächst überlegen, unter welchen Umständen der MOORE-BELLMAN-Algorithmus bei allgemeinen Gewichten nicht funktioniert. Wir betrachten den Digraphen aus Abbildung 6.1 mit den dort eingetragenen Gewichten.

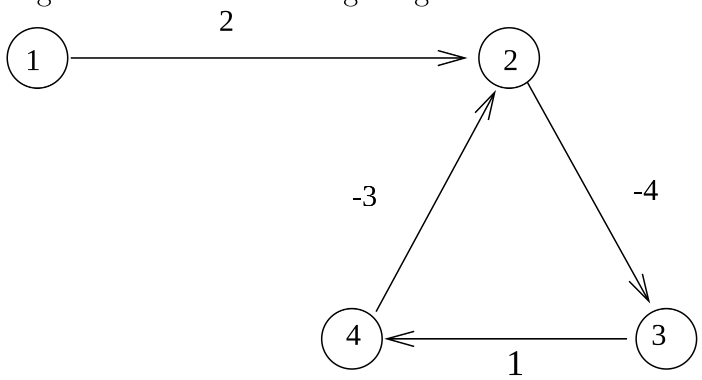

Abb. 6.1

Wir initialisieren mit $\text{DIST}(1) = 0$, $\text{DIST}(2) = 2$, $\text{DIST}(3) = \text{DIST}(4) = +\infty$, $\text{VOR}(i) = 1$, $i = 1, 2, 3, 4$. Wir stellen fest, daß $\text{DIST}(3) > \text{DIST}(2) + c((2, 3)) = -2$ und setzen $\text{DIST}(3) = -2$, $\text{VOR}(3) = 2$. Wir setzen analog $\text{DIST}(4) = \text{DIST}(3) + c((3, 4)) = -1$, $\text{VOR}(4) = 3$. Nun gilt $\text{DIST}(2) = 2 > \text{DIST}(4) + c((4, 2)) = -4$, also setzen wir $\text{DIST}(2) = -4$. Was ist passiert? Der kürzeste Weg von 1 nach 2 besteht offensichtlich aus dem Bogen (1,2) und hat die Länge 2. Wenden wir unser Verfahren an, so stellen wir fest, daß wir von 1 nach 4 mit der Weglänge -1 gelangen können. Dieser Weg enthält den Knoten 2. Aber nun können wir von 4 nach 2 zurückgehen, und unsere gerichtete Kette von 1 nach 2, nach 3, nach 4 und wieder zu 2 hat eine geringere Länge als der direkte Weg von 1 nach 2. Der Grund für diese Wegverkürzung liegt darin, daß wir einen Kreis, hier den Kreis (2,3,4), entdeckt haben, dessen Gesamtlänge negativ ist. Laufen wir nun noch einmal durch diesen Kreis, so können wir die "Weglänge" noch weiter verkürzen, d. h. unser Verfahren wird eine immer kleinere "Weglänge" produzieren und nicht enden. Nennen wir einen gerichteten Kreis C **negativ**, wenn sein Gewicht $c(C)$ negativ ist, so zeigt die obige Überlegung, daß negative Kreise in einem Digraphen zum Scheitern des Verfahrens führen. Hat ein Digraph überhaupt keinen gerichteten Kreis, ist er also azyklisch, so gibt es insbesondere keine negativen Kreise, und das MOORE-BELLMAN-Verfahren funktioniert.

(6.4) MOORE-BELLMAN-Algorithmus für azyklische Digraphen.

Input: *Azyklischer Digraph $D = (V, A)$, Gewichte $c(a)$ für alle $a \in A$ (auch negative Gewichte sind zugelassen), ein Knoten $s \in V$.*

Output: *Kürzeste gerichtete Wege von s nach v für alle $v \in V$ und ihre Länge.*

Datenstrukturen:

DIST(v), VOR(v) $\forall\, v \in V$. *O. B. d. A. nehmen wir an, daß $V = \{1, 2, \ldots, n\}$ gilt und alle Bögen die Form (u, v) mit $u < v$ haben.*

1. *Setze:*

$$\mathrm{DIST}(v) := \begin{cases} 0 & \text{falls } s = v \\ \infty & \text{falls } s \neq v \text{ und } (s,v) \notin A \\ c((s,v)) & \text{andernfalls} \end{cases}$$

$$\mathrm{VOR}(v) := s.$$

2. DO $v = s + 2$ TO n:
3. DO $u = s + 1$ TO $v - 1$:
   Falls $(u, v) \in A$ *und* $\mathrm{DIST}(u) + c((u, v)) < \mathrm{DIST}(v)$ *setze*
   $\mathrm{DIST}(v) := \mathrm{DIST}(u) + c((u, v))$ *und* $\mathrm{VOR}(v) := u$.
   END *3.*

   END *2.*
4. *Falls* $\mathrm{DIST}(v) < +\infty$*, so enthält* $\mathrm{DIST}(v)$ *die Länge des kürzesten gerichteten Weges von s nach v, und aus* VOR *kann ein kürzester (s, v)-Weg entnommen werden. Falls* $\mathrm{DIST}(v) = +\infty$*, so existiert in D kein (s, v)-Weg.*

□

(6.5) Satz. *Algorithmus (6.4) funktioniert für beliebige azyklische Digraphen D und beliebige Bogengewichte.*

Beweis. Nach Voraussetzung haben alle Bögen in D die Form (u, v) mit $u < v$. Folglich gibt es in D keinen (s, v)-Weg für $v < s$. Nach Definition ist die Länge eines (s, s)-Weges gleich Null. Ferner enthält jeder (s, v)-Weg mit $v > s$ nur innere Knoten u mit $s < u < v$. Es gibt höchstens einen $(s, s+1)$-Weg, nämlich den Bogen $(s, s+1)$, falls er in D existiert, also enthält $\mathrm{DIST}(v)$ für $1 \leq v \leq s + 1$ die Länge eines kürzesten (s, v)-Weges in D.

Ist $v \geq s + 1$, so folgt durch Induktion über die Schleifenindizes der Schleife 2, daß $\mathrm{DIST}(u)$ die Länge eines kürzesten (s, u)-Weges für $1 \leq u \leq v$ enthält. Aus formalen

Gründen lassen wir Schleife 2 mit $v = s + 1$ beginnen. Dadurch wird kein Wert $\mathrm{DIST}(u)$ in Schritt 3 geändert. Für $v = s + 1$ ist somit nach obiger Bemerkung die Behauptung korrekt. Sei also die Behauptung für v richtig und betrachten wir den Knoten $v + 1$. Nach Induktionsvoraussetzung enthält $\mathrm{DIST}(u)$, $1 \le u \le v$, die Länge eines kürzesten (s, u)-Weges. Da ein $(s, v + 1)$-Weg entweder von s direkt nach $v + 1$ führt (das Gewicht dieses Weges ist in $\mathrm{DIST}(v + 1)$ gespeichert) oder zunächst zu Zwischenknoten u im Intervall $s < u \le v$ und dann auf einen Bogen nach $v + 1$ führt, ist also die Länge des kürzesten $(s, v + 1)$-Weges gegeben durch das Minimum der folgenden beiden Werte:

$$c((s, v + 1)) = \mathrm{DIST}(v + 1),$$

$$\text{Länge des kürzesten } (s, u)\text{-Weges } + c((u, v + 1)) = \mathrm{DIST}(u) + c((u, v + 1)).$$

Dieses Minimum wird offenbar bei Ausführung der Schleife 3 für $v + 1$ berechnet. Daraus folgt die Behauptung. □

Da das Verfahren (6.4) im wesentlichen aus zwei Schleifen besteht, die beide über maximal $n - 2$ Indizes laufen, ist die Laufzeit des Verfahrens $O(n^2)$.

Wir geben nun den MOORE-BELLMAN-Algorithmus in zwei verschiedenen Varianten an:

(6.6) MOORE-BELLMAN-Algorithmus.

Input: *Digraph $D = (V, A)$, Gewichte $c(a)$ für alle $a \in A$ (können auch negativ sein), ein Knoten $s \in V$.*

Output: *Kürzeste gerichtete Wege von s nach v für alle $v \in V$ und ihre Länge, falls D keinen negativen Kreis enthält.*

Datenstrukturen: $\mathrm{DIST}(v)$, $\mathrm{VOR}(v)$ *für alle $v \in V$ (wie in (6.1))*

1. *Setze:*

$$\begin{aligned} \mathrm{DIST}(s) &:= 0, \\ \mathrm{DIST}(v) &:= \infty \ \forall\, v \in V \setminus \{s\} \\ \mathrm{VOR}(v) &:= s \quad \forall\, v \in V. \end{aligned}$$

YEN-VARIANTE

Wir nehmen hier zur Vereinfachung der Darstellung o. B. d. A. an, daß $V = \{1, \ldots, n\}$ und $s = 1$ gilt.

2. DO $m = 0$ TO $n - 2$:
3. *Falls m gerade:* DO $v = 2$ TO n:
4. DO $u = 1$ TO $v - 1$:
   Falls $(u, v) \in A$ *und* $\mathrm{DIST}(u) + c((u, v)) < \mathrm{DIST}(v)$,
   setze $\mathrm{DIST}(v) := \mathrm{DIST}(u) + c((u, v))$ *und* $\mathrm{VOR}(v) := u$.
   END *4.*
   END *3.*
5. *Falls m ungerade:* DO $v = n - 1$ TO *1* BY -1:

6. DO $u = n$ TO $v + 1$ BY -1:

Falls $(u, v) \in A$ *und* $\mathrm{DIST}(u) + c((u, v)) < \mathrm{DIST}(v)$,

setze $\mathrm{DIST}(v) := \mathrm{DIST}(u) + c((u, v))$

und $\mathrm{VOR}(v) := u$.

END *6.*

END *5.*

END *2.*

Gehe zu 7.

D'ESOPO-PAPE-VARIANTE

2'. Initialisiere eine Schlange Q und setze s in Q.

3'. Hole das erste Element aus der Schlange, sagen wir u.

4'. Für alle Bögen (u, v), die in u beginnen, führe aus:

5'. Falls $\mathrm{DIST}(u) + c((u, v)) < \mathrm{DIST}(v)$

a) *setze* $\mathrm{DIST}(v) := \mathrm{DIST}(u) + c((u, v))$ *und* $\mathrm{VOR}(v) := u$,

b) *Falls v noch nicht in Q war, setze v an das Ende von Q,*

c) *Falls v schon in Q war, aber gegenwärtig nicht in Q ist, setze v an den Anfang von Q.*

END *4'.*

6'. Ist die Schlange nicht leer gehe zu 3', andernfalls zu 7.

7. Falls $\mathrm{DIST}(v) < +\infty$, *so enthält* $\mathrm{DIST}(v)$ *die Länge eines kürzesten (s, v)-Weges, und aus* $\mathrm{VOR}(v)$ *kann wie üblich ein kürzester (s, v)-Weg rekonstruiert werden. Ist* $\mathrm{DIST}(v) = +\infty$, *so gibt es in D keinen (s, v)-Weg.* □

Es ist intuitiv einsichtig, daß das MOORE-BELLMAN-Verfahren ein korrektes Ergebnis liefert, falls keine negativen Kreise vorliegen. Ebenso leuchtet ein, daß die D'ESOPO-PAPE-Variante eine Spezialisierung dieses Verfahrens ist mit einer konkreten Angabe der Bearbeitungsreihenfolge. Wir wollen nun noch die Korrektheit der YEN-Variante vorführen.

(6.7) Satz. *Die YEN-Variante des MOORE-BELLMAN-Verfahrens arbeitet korrekt, falls D keinen negativen gerichteten Kreis enthält.*

Beweis. Wir geben dem Vektor DIST eine Interpretation, aus der die Korrektheit einfach folgt. Wir haben in (6.6) angenommen, daß $s = 1$ gilt und die Knoten mit $1, 2, \ldots, n$ bezeichnet sind. Wir nennen einen Bogen (u, v) einen **Aufwärtsbogen**, falls $u < v$ gilt, andernfalls heißt (u, v) **Abwärtsbogen**. Wir sprechen von einem **Richtungswechsel**, wenn in einem (s, v)-Weg ein Abwärtsbogen auf einen Aufwärtsbogen folgt oder umgekehrt. Da $s = 1$, ist der erste Bogen immer ein Aufwärtsbogen, also ist der erste Richtungswechsel immer aufwärts nach abwärts. Um einfacher argumentieren zu können, bezeichnen wir mit $\mathrm{DIST}(v, m)$ den Inhalt des Vektors $\mathrm{DIST}(v)$ nach Beendigung der m-ten Iteration der äußeren Schleife. Wir behaupten nun, daß $\mathrm{DIST}(v, m)$, $0 \leq m \leq n - 2$, die Länge des kürzesten aller der $(1, v)$-Wege enthält, bei denen nur m Richtungswechsel erlaubt sind. Da ein $(1, v)$-Weg höchstens $n - 1$ Bögen und somit höchstens $n - 2$ Richtungswechsel besitzt, folgt der Satz aus dem Beweis unserer Behauptung.

Wir beweisen unsere Behauptung durch Induktion über m. Für $m = 0$ ist der Durchlauf der Schritte 3 und 4 nichts anderes als Algorithmus (6.4) (angewendet auf $s = 1$), dessen Korrektheit wir in Satz (6.5) bewiesen haben. $\mathrm{DIST}(v, 0)$ enthält somit die Länge des kürzesten $(1, v)$-Weges ohne Richtungswechsel, die Behauptung für $m = 0$ ist richtig.

Nehmen wir nun an, daß unsere Behauptung für $m \geq 0$ richtig ist und daß wir Schleife 2 zum $(m + 1)$-sten Male durchlaufen haben. Wir müssen zwei Fälle unterscheiden: $m + 1$ gerade oder ungerade. Wir führen den Fall $m+1$ ungerade vor, der andere Fall folgt analog. Die Menge der (s, v)-Wege mit höchstens $m + 1$ Richtungswechseln besteht aus folgenden Wegen:

(a) (s, v)-Wege mit höchstens m Richtungswechseln,

(b) (s, v)-Wege mit genau $m + 1$ Richtungswechseln.

Die Minimallänge der Wege in (a) kennen wir bereits, sie ist nach Induktionsvoraussetzung $\mathrm{DIST}(v, m)$. Da nach Annahme der erste Bogen eines jeden Weges ein Aufwärtsbogen ist, ist für einen (s, v)-Weg mit $m + 1$ Richtungswechseln und $m + 1$ ungerade, der letzte Bogen ein Abwärtsbogen. Zur Bestimmung des Minimums in (b) führen wir eine weitere Induktion über $u = n, n - 1, \ldots, v + 1$ durch. Da jeder Weg, der in n endet mit einem Aufwärtsbogen aufhört, gibt es keinen (s, n)-Weg mit genau $m+1$ Richtungswechseln, also gilt $\mathrm{DIST}(n, m) = \mathrm{DIST}(n, m+1)$. Nehmen wir nun an, daß wir wissen, daß $\mathrm{DIST}(w, m+1)$ für $n \geq w \geq u > v + 1$ die Länge eines kürzesten (s, w)-Weges mit höchstens $m + 1$ Richtungswechseln ist. Zur Bestimmung der Länge eines kürzesten $(s, u-1)$-Weges müssen wir die Länge eines kürzesten $(s, u - 1)$-Weges mit höchsten m Richtungswechseln (diese ist in $\mathrm{DIST}(u - 1, m)$ gespeichert) vergleichen mit der Länge eines kürzesten $(s, u - 1)$-Weges mit genau $m + 1$ Richtungswechseln. Im letzteren Falle ist der letzte Bogen eines solchen $(s, u - 1)$-Weges ein Abwärtsbogen. Der vorletzte Knoten w erfüllt also $u - 1 < w \leq n$. Ist für $u - 1 < w \leq n$ P ein kürzester (s, w)-Weg, und hat P höchstens m Richtungswechsel, so ist, falls $(w, u - 1) \in A$, $P \cup \{(w, u - 1)\}$ ein $(s, u - 1)$-Weg mit höchstens m Richtungswechseln. Hat P $m + 1$ Richtungswechsel, so ist der letzte Bogen in P nach obiger Bemerkung ein Abwärtsbogen, also hat auch $P \cup \{(w, u-1)\}$ genau $m+1$ Richtungswechsel. Nach Induktionsvoraussetzung befindet sich in $\mathrm{DIST}(w, m + 1)$ für $u - 1 < w \leq n$ die Länge eines kürzesten (s, w)-Weges mit höchstens $m + 1$ Richtungswechseln, also ist $\min\{\mathrm{DIST}(w, m+1) + c(w, u-1) \mid u-1 < w \leq n\}$ nicht größer als die Länge des kürzesten $(s, u - 1)$-Weges mit genau $m + 1$ Richtungswechseln. Dieses Minimum wird offenbar in der Schleife 6 bestimmt und mit $\mathrm{DIST}(u - 1, m)$ verglichen. Damit ist gezeigt, daß der Induktionsschritt der zweiten Induktion korrekt ist. Hieraus folgt die Korrektheit der ersten Induktion, und wir sind fertig. $\square$

Wir haben festgestellt, daß die beiden Varianten des MOORE-BELLMAN-Verfahrens korrekt arbeiten, wenn der gegebene Digraph keine negativen Kreise enthält, aber haben bisher verschwiegen, wie man das effektiv entdeckt. Wie man das bei der D'ESOPO-PAPE-Variante auf einfache Weise machen kann — ohne andere Algorithmen einzuschalten — ist mir nicht bekannt. Bei der YEN-Variante gibt es eine simple Modifikation, die das Gewünschte leistet.

(6.8) Bemerkung. Nehmen wir an, daß jeder Knoten des Digraphen D von $s = 1$ auf einem gerichteten Weg erreicht werden kann. D enthält einen negativen Kreis genau dann, wenn bei einer zusätzlichen Ausführung der Schleife 2 der YEN-Variante (also für $m = n - 1$) der Wert DIST(v) für mindestens einen Knoten $v \in V$ geändert wird.

□

Der Beweis dieser Bemerkung sei dem Leser überlassen. Im nächsten Abschnitt werden wir auf das Thema "negative Kreise" noch einmal zurückkommen.

Die YEN-Variante des MOORE-BELLMAN-Algorithmus hat, da drei Schleifen über maximal n Indizes ineinander geschaltet sind, eine Laufzeit von $O(n^3)$. Für die D'ESOPO-PAPE-Variante gibt es (konstruierte) Beispiele mit exponentieller Laufzeit (siehe Übungsaufgabe). Dennoch hat sie sich in der Praxis als sehr schnell erwiesen und ist fast immer der YEN-Variante überlegen. Sind alle Gewichte positiv und sollen kürzeste (s, v)-Wege für alle $v \in V$ bestimmt werden, so ist die DIJKSTRA-Methode für Digraphen mit vielen Bögen (d. h. $O(n^2)$ Bögen) die bessere Methode; bei Digraphen mit wenigen Bögen haben extensive Testläufe gezeigt, daß die D'ESOPO-PAPE-Variante in der Praxis günstigstere Laufzeiten erbringt.

6.3 Kürzeste Wege zwischen allen Knotenpaaren

Natürlich kann man kürzeste Wege zwischen je zwei Knotenpaaren eines Digraphen D dadurch bestimmen, daß man das DIJKSTRA- oder das MOORE-BELLMAN-Verfahren n-mal anwendet, d. h. jeder Knoten wird einmal als Startknoten gewählt. Bei Benutzung der DIJKSTRA-Methode (nicht-negative Gewichte vorausgesetzt) hätte dieses Verfahren eine Laufzeit von $O(n^3)$, falls negative Gewichte vorkommen, müßte die YEN-Variante verwendet werden, was zu einer Laufzeit von $O(n^4)$ führt. Es gibt jedoch einen extrem einfachen $O(n^3)$-Algorithmus, der das gleiche leistet. Dieses Verfahren geht auf Floyd (1962) zurück.

(6.9) FLOYD-Algorithmus.

Input: *Digraph $D = (V, A)$, $V = \{1, \dots, n\}$ mit Gewichten $c(a)$ (können auch negativ sein), für alle $a \in A$.*

Output: *Eine (n, n)-Matrix $W = (w_{ij})$, so daß für $i \neq j$ w_{ij} die Länge des kürzesten (i, j)-Weges und w_{ii} die Länge eines kürzesten gerichteten Kreises, der i enthält, ist (eine Matrix mit diesen Eigenschaften nennt man* **Kürzeste-Weglängen-Matrix***) und eine (n, n)-Matrix $P = (p_{ij})$, so daß p_{ij} der vorletzte Knoten eines kürzesten (i, j)-Weges (bzw. (i, i)-Kreises) ist.*

1. DO $i = 1$ TO n:
   DO $j = 1$ TO n:

$$w_{ij} := \begin{cases} c((i,j)) & \text{falls } (i,j) \in A \\ \infty & \text{andernfalls} \end{cases}$$

$$p_{ij} := \begin{cases} i & \text{falls } (i,j) \in A \\ 0 & \text{andernfalls (bedeutet, zur Zeit kein Weg bekannt)} \end{cases}$$

END
END.

2. DO $l = 1$ TO n*:*
DO $i = 1$ TO n*:*
DO $j = 1$ TO n*:*
Falls $w_{ij} > w_{il} + w_{lj}$,
setze $w_{ij} := w_{il} + w_{lj}$ *und* $p_{ij} := p_{lj}$.
(Falls $i = j$ *und* $w_{ii} < 0$*, kann abgebrochen werden.)*
END
END
END.

3. Gib W *und* P *aus.*

□

Für zwei Knoten i, j kann der in P gespeicherte kürzeste (i, j)-Weg wie folgt bestimmt werden. Setze

$$v_q := p_{ij},\ v_{q-1} := p_{i,v_q},\ v_{q-2} := p_{i,v_{q-1}}, \dots, i := p_{i,v_1}$$

d. h. wir iterieren so lange bis ein Knoten v_s der Knoten i ist, dann ist

$$(i, v_1, v_2, \dots, v_q, j)$$

ein kürzester (i, j)-Weg. Überzeugen Sie sich, daß dies stimmt!

(6.10) Satz. *Sei* $D = (V, A)$ *ein Digraph mit beliebigen Bogengewichten* $c(a)$ *für alle* $a \in A$*. Sei* W *die* (n, n)*-Matrix, die vom FLOYD-Algorithmus produziert wird, dann gilt:*

(a) Der FLOYD-Algorithmus liefert genau dann eine Kürzeste-Weglängen-Matrix W*, wenn* D *keinen negativen gerichteten Kreis enthält.*

(b) D *enthält genau dann einen negativen gerichteten Kreis, wenn ein Hauptdiagonalelement von* W *negativ ist.*

Beweis. Zur Notationsvereinfachung bezeichnen wir die Anfangsmatrix W aus Schritt 1 mit W^0, die Matrix W nach Beendigung des l-ten Durchlaufs von der äußeren Schleife von 2 mit W^l. Durch Induktion über $l = 0, 1, \dots, n$ zeigen wir, daß W^l genau dann die Matrix der kürzesten Längen von (i, j)-Wegen (bzw. (i, i)-Kreisen) ist, bei denen die Knoten $1, \dots, l$ als innere Knoten auftreten können, wenn D keinen negativen Kreis in der Knotenmenge $1, \dots, l$ besitzt. Ist letzteres der Fall, so gilt $w_{ii}^l < 0$ für ein $i \in \{1, \dots, l\}$.

Für $l = 0$ ist die Behauptung offenbar richtig. Angenommen, sie ist für $l \geq 0$ richtig, und wir haben die äußere Schleife von 2 zum $(l + 1)$-sten Male durchlaufen. Bei diesem Durchlauf haben wir folgenden Test ausgeführt.

$$\text{Falls } w_{ij}^l > w_{i,l+1}^l + w_{l+1,j}^l, \text{ dann } w_{ij}^{l+1} := w_{i,l+1}^l + w_{l+1,j}^l,$$

d. h. wir haben die (nach Induktionsvoraussetzung) kürzeste Länge eines (i, j)-Weges über die Knoten $1, \dots, l$ verglichen mit der Summe der kürzesten Längen eines $(i, l + 1)$-Weges und eines $(l+1, j)$-Weges jeweils über die Knoten $1, \ldots, l$. Die letztere Summe repräsentiert also die Länge eines kürzesten (i, j)-Weges über $1, \ldots, l+1$, der den Knoten $l+1$ enthält. Falls diese Summe kleiner als w_{ij}^l ist, setzen wir $w_{ij}^{l+1} := w_{i,l+1}^l + w_{l+1,j}^l$, andernfalls $w_{ij}^{l+1} = w_{ij}^l$. Daraus folgt die Behauptung, es sei denn, $w_{ij}^l > w_{i,l+1}^l + w_{l+1,j}^l$ und die Verkettung, sagen wir K des $(i, l+1)$-Weges mit dem $(l+1, j)$-Weg ist gar kein Weg, d. h. K ist eine gerichtete (i, j)-Kette, die einen Knoten mindestens zweimal enthält. Die Kette K enthält natürlich einen (i, j)-Weg, sagen wir $\overline{K}$, und $\overline{K}$ geht aus K dadurch hervor, daß wir die in K vorkommenden gerichteten Kreise entfernen. Der Knoten $l+1$ ist nach Konstruktion in einem der Kreise enthalten, also ist $\overline{K}$ ein (i, j)-Weg, der nur Knoten aus $1, \ldots, l$ enthält, d. h. $w_{ij}^l \leq c(\overline{K})$. Aus $c(K) = w_{i,l+1}^l + w_{l+1,j}^l < w_{ij}^l$ folgt, daß mindestens einer der aus K entfernten gerichteten Kreise eine negative Länge hat. Für jeden Knoten i dieses negativen Kreises muß folglich $w_{ii}^{l+1} < 0$ gelten. Daraus folgt die Behauptung. □

Der FLOYD-Algorithmus liefert also explizit einen Kreis negativer Länge, falls ein solcher existiert.

(6.11) Folgerung. *Für einen Digraphen D mit Bogengewichten, der keine negativen gerichteten Kreise enthält, kann ein kürzester gerichteter Kreis in $O(n^3)$ Schritten bestimmt werden.*

Beweis. Wir führen den FLOYD-Algorithmus aus. Nach Beendigung des Verfahrens ist in w_{ii}, $i = 1, \ldots, n$ die Länge eines kürzesten gerichteten Kreises, der den Knoten i enthält, verzeichnet. Wir wählen einen Wert w_{ii}, der so klein wie möglich ist, und rekonstruieren aus der Matrix P, wie oben angegeben, den gerichteten Kreis, der i enthält. Dieser ist ein kürzester gerichteter Kreis in D. Diese "Nachbearbeitung" erfordert lediglich $O(n)$ Operationen, also ist die worst-case-Komplexität des FLOYD-Algorithmus auch die Laufzeitschranke für das Gesamtverfahren. □

Wendet man Algorithmus (6.9) auf Entfernungstabellen in Straßenatlanten an, so wird man feststellen, daß es häufig Städte i, j, k gibt mit $c_{ij} + c_{jk} < c_{ik}$. Die Entfernungen genügen also nicht der Dreiecksungleichung. Warum ist das wohl so?

Transformieren wir einen ungerichteten Graphen G in einen gerichteten Graphen D, indem wir jeder Kante ij die beiden Bögen (i, j) und (j, i) mit dem Gewicht von ij zuordnen, so können wir natürlich durch Anwendung unseres Verfahrens auf D auch kürzeste Wege bzw. Kreise in G bestimmen. Man beachte jedoch, daß ein negatives Kantengewicht $c(ij)$ in G automatisch zu einem negativen gerichteten Kreis (i, j) (j, i) in D führt. Mit unseren Methoden können wir also nur kürzeste Wege und Kreise in Graphen mit nichtnegativen Kantengewichten bestimmen.

Es sei an dieser Stelle jedoch darauf hingewiesen, daß auch in Graphen mit negativen Kantengewichten kürzeste Wege und Kreise bestimmt werden können, falls kein Kreis negativ ist. Dies geschieht mit Methoden der Matching-Theorie, auf die wir hier aus Zeitgründen nicht eingehen können.

Literaturverzeichnis von Kapitel 6

R. W. Ahuja, T. L. Magnanti und J. B. Orlin (1993), *Network flows: Theory, algorithms and applications,* Prentice Hall, Englewood Cliffs, N.J..

R. E. Bellman (1958), "On a routing problem", Quart. Appl. Math. 16, 87–90.

E. W. Dijkstra (1959), "A note on two problems in connexion with graphs", Numer. Math. 1, 269–271.

W. Domschke (1972), *Kürzeste Wege in Graphen,* Mathematical Systems in Economics 2, Verlag A. Hain, Meisenheim am Glan.

R. W. Floyd (1962), "Algorithm 97, Shortest path", Comm. ACM 5, 345.

E. L. Lawler (1976), *Combinatorial Optimization: Networks and Matroids,* Holt, Rinehart and Winston, New York.

K. Mehlhorn (1984), *Data structures and algorithms,* Springer-Verlag, Berlin.

E. F. Moore (1959), "The shortest path through a maze", Proc. Int. Symp. on Theory of Switching Part II, Cambridge, Mass., Harvard University Press, 285–292.

M. M. Syslo, N. Deo und J. S. Kowalik (1983), *Discrete optimization algorithms,* Prentice Hall, Englewood Cliffs, N.J..

Kapitel 7

Maximale Flüsse in Netzwerken

In diesem Kapitel behandeln wir ein in sowohl theoretischer als auch praktischer Hinsicht außerordentlich interessantes Gebiet: Flüsse in Netzwerken. Es war früher üblich und wird aus Traditionsgründen häufig weiter so gehandhabt, einen Digraphen mit Bogengewichten bzw. -kapazitäten ein **Netzwerk** zu nennen. Sind zusätzlich zwei Knoten s und t ausgezeichnet, so spricht man von einem (s, t)-Netzwerk. Wir wollen diese Bezeichnung hier nicht übernehmen, sondern nur im Kapiteltitel andeuten. Es wird sich zeigen, daß die Netzwerkflußtheorie als ein Bindeglied zwischen der linearen und der ganzzahligen Optimierung aufgefaßt werden kann. Netzwerkflußprobleme sind (ganzzahlige) lineare Programme, für die sehr schnelle kombinatorische Lösungsmethoden existieren. Der Dualitätssatz der linearen Programmierung hat hier eine besonders schöne Form.

Netzwerkflüsse haben folgenden Anwendungshintergrund. Wir betrachten ein Rohrleitungssystem (Abwasserkanäle, Frischwasserversorgung), bei dem die Rohre gewisse Kapazitäten (z. B. maximale Durchflußmenge pro Minute) haben. Einige typische Fragen lauten: Was ist die maximale Durchflußmenge durch das Netzwerk? Welche Wassermenge kann man maximal pro Minute vom Speicher zu einem bestimmten Abnehmer pumpen? Wieviel Regenwasser kann das System maximal aufnehmen? Ähnliches gilt für Telefonnetzwerke. Hier sind die "Rohre" die Verbindungen zwischen zwei Knotenpunkten, die Kapazitäten die maximalen Anzahlen von Gesprächen die über eine Verbindung geführt werden können. Man interessiert sich z. B. für die maximale Gesprächszahl, die parallel zwischen zwei Orten (Ländern) geführt werden können.

Darüber hinaus treten Netzwerkflußprobleme in vielfältiger Weise als Unter- oder Hilfsprobleme bei der Lösung komplizierter Anwendungsprobleme auf. Insbesondere werden Netzwerkflußalgorithmen sehr häufig als Separierungsroutinen bei Schnittebenenverfahren eingesetzt.

Das klassische Werk der Netzwerkflußtheorie ist das Buch Ford und Fulkerson (1962). Es ist auch heute noch lesenswert. Es gibt unzählige Veröffentlichungen zur Theorie, Algorithmik und den Anwendungen der Netzwerkflußtheorie. Durch neue algorithmische Ansätze (Präfluß-Techniken, Skalierungsmethoden) und effiziente Datenstrukturen sind gerade in letzter Zeit wieder sehr viele Artikel zu diesem Thema erschienen. Gute Darstellungen hierzu sind in den umfangreichen und sehr informative Übersichtsartikeln Ahuja, Magnanti und Orlin (1989), Goldberg, Tardos und Tarjan (1990) und Frank (1995) zu finden. Ein sehr empfehlenswertes Buch ist Ahuja, Magnanti und Orlin (1993). Die beiden Handbücher Ball et al. (1995a, 1995b) enthalten recht umfassende Informationen zur Theorie, Algorithmik und zu den Anwendungen von Netzwerken.

7.1 Das Max-Flow Min-Cut Theorem

Im folgenden sei $D = (V, A)$ ein Digraph mit Bogenkapazitäten $c(a) \in \mathbb{K}$, $c(a) \geq 0$ für alle $a \in A$. Ferner seien s und t zwei voneinander verschiedene Knoten aus V. Der Knoten s heißt **Quelle** (englisch: source), und t heißt **Senke** (englisch: sink). Eine Funktion $x : A \to \mathbb{K}$ (bzw. ein Vektor $x \in \mathbb{K}^A$) heißt **zulässiger** (s,t)**-Fluß**, wenn die folgenden beiden Bedingungen erfüllt sind:

(7.1) $$0 \leq x_a \leq c_a \quad \text{für alle } a \in A \qquad \text{(Kapazitätsbedingungen)}$$

(7.2) $$\sum_{a \in \delta^-(v)} x_a = \sum_{a \in \delta^+(v)} x_a \quad \text{für alle } v \in V \setminus \{s,t\} \text{ (Flußerhaltungsbedingungen)}$$

Ist $x \in \mathbb{K}^A$ ein zulässiger (s,t)-Fluß, dann heißt

(7.3) $$\mathrm{val}(x) := \sum_{a \in \delta^+(s)} x_a - \sum_{a \in \delta^-(s)} x_a$$

der **Wert** des (s,t)-Flusses x.

Wir werden uns in diesem Abschnitt damit beschäftigen, eine Charakterisierung des maximalen Wertes eines (s,t)-Flusses zu finden. In den nächsten beiden Abschnitten werden wir Algorithmen zur Bestimmung eines maximalen Flusses angeben.

Wir erinnern daran, daß ein (s,t)**-Schnitt** in D eine Bogenmenge der Form $\delta^+(W) = \delta^-(V \setminus W) = \{(i,j) \in A \mid i \in W, j \in V \setminus W\}$ mit $s \in W \subseteq V$ und $t \in V \setminus W$ ist. Die **Kapazität eines Schnittes** $\delta^+(W)$ ist wie üblich mit $c(\delta^+(W)) = \sum_{a \in \delta^+(W)} c_a$ definiert. Aus der "Rohrleitungsanwendung" wird der Name "Schnitt" klar. Durchschneiden wir alle Rohre irgendeines Schnittes, d. h. entfernen wir alle Bögen eines (s,t)-Schnittes aus dem Digraphen, dann kann kein Fluß mehr von s nach t "fließen". Diese triviale Beobachtung liefert:

(7.4) Lemma. *(a) Sei $s \in W$, $t \notin W$, dann gilt für jeden zulässigen (s,t)-Fluß x:*

$$\mathrm{val}(x) = \sum_{a \in \delta^+(W)} x_a - \sum_{a \in \delta^-(W)} x_a.$$

(b) Der maximale Wert eines zulässigen (s,t)-Flusses ist höchstens so groß wie die minimale Kapazität eines (s,t)-Schnittes.

Beweis. (a) Aus der Flußerhaltungsbedingung (7.2) folgt

$$\begin{aligned}\mathrm{val}(x) &= \sum_{a \in \delta^+(s)} x_a - \sum_{a \in \delta^-(s)} x_a = \sum_{v \in W} \Big(\sum_{a \in \delta^+(v)} x_a - \sum_{a \in \delta^-(v)} x_a \Big) \\ &= \sum_{a \in \delta^+(W)} x_a - \sum_{a \in \delta^-(W)} x_a.\end{aligned}$$

(b) Seien $\delta^+(W)$ ein beliebiger (s,t)-Schnitt und x ein zulässiger (s,t)-Fluß, dann gilt wegen (a), (7.1) und (7.2)

$$\mathrm{val}(x) = \sum_{a\in\delta^+(W)} x_a - \sum_{a\in\delta^-(W)} x_a \leq \sum_{a\in\delta^+(W)} c_a = c(\delta^+(W)).$$

$\square$

Wir werden später einen kombinatorischen Beweis dafür angeben, daß der maximale Wert eines (s,t)-Flusses gleich der minimalen Kapazität eines (s,t)-Schnittes ist. Hier wollen wir diesen Satz aus der Dualitätstheorie der linearen Programmierung ableiten. Dazu schreiben wir die Aufgabe, einen maximalen (s,t)-Flußwert in D zu finden, als lineares Programm. Dieses lautet wie folgt:

$$\max \sum_{a\in\delta^+(s)} x_a - \sum_{a\in\delta^-(s)} x_a \quad (= x(\delta^+(s)) - x(\delta^-(s)))$$

$$\begin{aligned} x(\delta^-(v)) - x(\delta^+(v)) = \sum_{a\in\delta^-(v)} x_a - \sum_{a\in\delta^+(v)} x_a &= 0 \text{ für alle } v \in V\setminus\{s,t\}, \\ 0 \leq x_a \leq c_a & \text{ für alle } a \in A. \end{aligned} \tag{7.5}$$

Jede zulässige Lösung von (7.5) ist also ein zulässiger (s,t)-Fluß, und jede optimale Lösung ein optimaler(s,t)-Fluß. Um das zu (7.5) duale lineare Programm aufschreiben zu können, führen wir für jeden Knoten $v \in V\setminus\{s,t\}$ eine Dualvariable z_v und für jeden Bogen $a \in A$ eine Dualvariable y_a ein. Das folgende lineare Programm ist dann zu (7.5) dual.

$$\begin{array}{lllll}
\min \sum_{a\in A} c_a y_a & & & & \\
y_a + z_v - z_u & \geq & 0 & \text{falls} & a = (u,v) \in A(V\setminus\{s,t\}) \\
y_a \qquad\; - z_u & \geq & -1 & \text{falls} & a = (u,s), u \neq t \\
y_a + z_v & \geq & 1 & \text{falls} & a = (s,v), v \neq t \\
y_a \qquad\; - z_u & \geq & 0 & \text{falls} & a = (u,t), u \neq s \\
y_a + z_v & \geq & 0 & \text{falls} & a = (t,v), v \neq s \\
y_a & \geq & 1 & \text{falls} & a = (s,t) \\
y_a & \geq & -1 & \text{falls} & a = (t,s) \\
\qquad y_a & \geq & 0 & \text{für alle} & a \in A
\end{array}$$

Führen wir zusätzlich die Variablen z_s und z_t ein und setzen sie mit 1 bzw. 0 fest, so kann man dieses LP äquivalent, aber etwas kompakter wie folgt schreiben:

$$\begin{aligned} \min \sum_{a\in A} c_a y_a & \\ y_a + z_v - z_u \geq 0 & \quad \text{für alle } a = (u,v) \in A \\ z_s = 1 & \\ z_t = 0 & \\ y_a \geq 0 & \quad \text{für alle } a \in A. \end{aligned} \tag{7.6}$$

Wir benutzen nun (7.5) und (7.6), um folgenden berühmten Satz, der auf Ford und Fulkerson (1956) und Elias, Feinstein und Shannon (1956) zurückgeht, zu beweisen.

(7.7) Das Max-Flow Min-Cut Theorem. *Gegeben seien ein Digraph $D = (V, A)$ mit Bogenkapazitäten $c_a \in \mathbb{K}$, $c_a \geq 0$ für alle $a \in A$, und zwei verschiedene Knoten $s, t \in V$. Dann ist der maximale Wert eines (s,t)-Flusses gleich der minimalen Kapazität eines (s,t)-Schnittes.*

Beweis. Aufgrund von Lemma (7.4) genügt es zu zeigen, daß es einen (s,t)-Schnitt gibt, dessen Kapazität gleich dem maximalen Flußwert ist. Da alle Variablen beschränkt sind und der Nullfluß zulässig ist, hat (7.5) eine optimale Lösung. Sei also x^* ein optimaler zulässiger (s,t)-Fluß mit Wert $\operatorname{val}(x^*)$. Aufgrund des Dualitätssatzes der linearen Programmierung gibt es eine Lösung, sagen wir y_a^*, $a \in A$ und z_v^*, $v \in V$, des zu (7.5) dualen Programms (7.6) mit $\operatorname{val}(x^*) = \sum_{a \in A} c_a y_a^*$. Wir setzen: $W := \{u \in V \mid z_u^* > 0\}$ und zeigen, daß $\delta^+(W)$ ein (s,t)-Schnitt mit $c(\delta^+(W)) = \operatorname{val}(x^*)$ ist.

Offenbar gilt $s \in W$, $t \notin W$, also ist $\delta^+(W)$ ein (s,t)-Schnitt. Ist $a = (u, v) \in \delta^+(W)$, dann gilt $z_u^* > 0$, $z_v^* \leq 0$ und folglich $y_a^* \geq z_u^* - z_v^* > 0$. Aufgrund des Satzes vom schwachen komplementären Schlupf muß dann in der zu y_a^* gehörigen Ungleichung $x_a \leq c_a$ des primalen Programms (7.5) Gleichheit gelten. Also erfüllt der optimale (s,t)-Fluß x^* die Gleichung $x_a^* = c_a$. Ist $a = (u, v) \in \delta^-(W)$, so gilt $z_v^* > 0$, $z_u^* \leq 0$ und somit (da $y_a^* \geq 0$) $y_a^* - z_u^* + z_v^* \geq z_v^* - z_u^* > 0$. Die Ungleichung ist also "locker". Der Satz vom komplementären Schlupf impliziert nun, daß die zugehörige Primalvariable $x_a \geq 0$ nicht positiv sein kann. Also gilt $x_a^* = 0$. Daraus folgt

$$c(\delta^+(W)) = x^*(\delta^+(W)) - x^*(\delta^-(W) = x^*(\delta^+(s)) - x^*(\delta^-(s)) = \operatorname{val}(x^*).$$

$\square$

Man beachte, daß der obige Beweis des Max-Flow Min-Cut Theorems konstruktiv ist. Aus jeder optimalen Lösung des dualen linearen Programms (7.6) können wir in polynomialer Zeit einen (s,t)-Schnitt konstruieren, der den gleichen Wert wie die Optimallösung von (7.6) hat und somit ein (s,t)-Schnitt in D minimaler Kapazität ist. Aus jedem (s,t)-Schnitt $\delta^+(W)$ können wir durch

$$\begin{aligned} y_a &:= 1 \quad \text{für alle } a \in \delta^+(W) \\ y_a &:= 0 \quad \text{für alle } a \in A \setminus \delta^+(W) \\ z_v &:= 1 \quad \text{für alle } v \in W \\ z_v &:= 0 \quad \text{für alle } v \in V \setminus W \end{aligned}$$

auch eine Lösung von (7.6) konstruieren, und daraus folgt, daß das lineare Programm (7.6) immer auch ganzzahlige optimale Lösungen hat. Wir können somit also das ganzzahlige Programm bestehend aus (7.6) plus Ganzzahligkeitsbedingungen für y_a und z_v durch das im Beweis von (7.7) angegebene Verfahren lösen.

Wir werden im nächsten Abschnitt zeigen, daß auch das LP (7.5) immer ganzzahlige Optimallösungen hat, wenn alle Kapazitäten ganzzahlig sind. Die diesem Beweis unterliegende Konstruktion ist der Startpunkt für effiziente Algorithmen zur Lösung des Maximalflußproblems.

Das Max-Flow Min-Cut Theorem hat vielfältige Anwendungen in der Graphentheorie und kombinatorischen Optimierung. Aus Zeitgründen können wir an dieser Stelle nicht darauf eingehen.

7.2 Der Ford-Fulkerson-Algorithmus

Wir haben gesehen, daß das Maximalflußproblem und das Minimalschnittproblem lineare Programme sind, folglich können wir sie effizient lösen. Das heißt, in der Praxis dürfte der Simplexalgorithmus (7.5) und (7.6) in kurzer Zeit gute Lösungen liefern, während theortisch die Ellipsoidmethode eine Laufzeit garantiert, die polynomial in der Inputlänge $|V| + |A| + \sum_{a \in A} \langle c_a \rangle$ ist. Für diese besonderen linearen Programme gibt es jedoch effiziente kombinatorische Algorithmen und Spezialversionen des Simplexalgorithmus, die außerordentlich schnell arbeiten. Wir werden drei dieser Verfahren vorstellen.

Das erste dieser Verfahren geht auf Ford und Fulkerson zurück. Die Idee hinter dieser Methode kann wie folgt erläutert werden. Man starte mit dem zulässigen (s,t)-Fluß $x_a = 0$ für alle $a \in A$. Hat man einen zulässigen (s,t)-Fluß, dann versuche man im gegebenen Digraphen einen gerichteten Weg von s nach t zu finden, auf dem zusätzlich ein positiver Wert durch das Netzwerk “geschoben” werden kann. Geht dies, so erhöht man den gegenwärtigen Fluß und fährt fort. Die Suche nach einem gerichteten (s,t)-Weg, der die Erhöhung des Flußwertes erlaubt, führt allerdings nicht direkt zum gewünschten Erfolg. Betrachten wir z. B. den Digraphen D in Abbildung 7.1, bei dem die erste Zahl des zu einem Bogen gehörenden Zahlenpaares den gegenwärtigen Flußwert des Bogens anzeigt und die zweite Zahl die Kapazität des Bogen angibt.

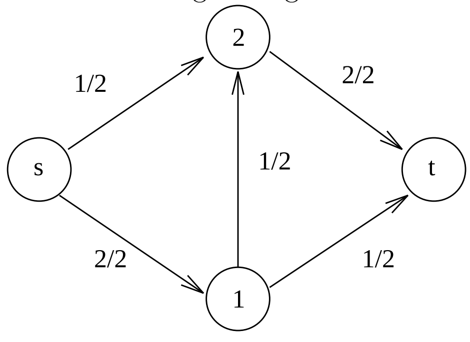

Abb. 7.1

Der gegenwärtige Fluß hat den Wert 3, und offenbar hat der Maximalfluß den Wert 4. Es gibt im Digraphen von Abbildung 7.1 aber keinen gerichteten (s,t)-Weg auf dem der gegenwärtige Fluß verbessert werden könnte. Auf allen drei gerichteten (s,t)-Wegen ist mindestens ein Bogenfluß an seiner maximalen Kapazität. Eine Möglichkeit, den Fluß entlang eines *ungerichteten* Weges zu erhöhen, haben wir jedoch. Wir betrachten den $[s,t]$-Weg P mit den Bögen $(s,2)$, $(1,2)$, $(1,t)$ und erhöhen den Fluß der Bögen $(s,2)$, $(1,t)$ um jeweils 1, erniedrigen den Fluß durch (1,2) um 1. Dadurch wird weder eine der Kapazitätsbedingungen (7.1) noch eine der Flußerhaltungsbedingungen verletzt, aber der Flußwert um 1 erhöht. Wir treffen daher folgende Definition.

(7.8) Definition. *Sei $D=(V,A)$ ein Digraph mit Bogengewichten c_a für alle $a \in A$, seien $s,t,v \in V$, $s \neq t$, und sei x ein zulässiger (s,t)-Fluß in D. In einem (ungerichteten) $[s,v]$-Weg P nennen wir einen Bogen (i,j), der auf P in Richtung s nach v verläuft,* **Vorwärtsbogen**, *andernfalls heißt (i,j)* **Rückwärtsbogen**. *P heißt* **augmentierender** $[s,v]$**-Weg** *(bezüglich des (s,t)-Flusses x), falls $x_{ij} < c_{ij}$ für jeden Vorwärtsbogen (i,j) gilt und $x_{ij} > 0$ für jeden Rückwärtsbogen (i,j) gilt. Wenn wir nur* **augmentierender Weg** *sagen, so meinen wir immer einen augmentierenden $[s,t]$-Weg.*

□

Im oben angegebenen Weg P des in Abbildung 7.1 gezeigten Digraphen ist $(1,2)$ ein Rückwärtsbogen, $(s,2)$ und $(1,t)$ sind Vorwärtsbögen. P selbst ist augmentierend bezüglich des gegebenen Flusses. Der folgende Satz liefert ein Optimalitätskriterium.

(7.9) Satz. *Ein (s,t)-Fluß x in einem Digraphen D mit Bogenkapazitäten ist genau dann maximal, wenn es in D keinen bezüglich x augmentierenden $[s,t]$-Weg gibt.*

Beweis. Ist P ein bezüglich x augmentierender $[s,t]$-Weg, dann sei

$$\varepsilon := \min \begin{cases} c_{ij} - x_{ij} & \text{falls } (i,j) \in P \text{ Vorwärtsbogen}, \\ x_{ij} & \text{falls } (i,j) \in P \text{ Rückwärtsbogen}. \end{cases} \tag{7.10}$$

Setzen wir

$$x'_{ij} := \begin{cases} x_{ij} + \varepsilon & \text{falls } (i,j) \in P \text{ Vorwärtsbogen}, \\ x_{ij} - \varepsilon & \text{falls } (i,j) \in P \text{ Rückwärtsbogen}, \\ x_{ij} & \text{falls } (i,j) \in A \setminus P, \end{cases} \tag{7.11}$$

dann ist offenbar x'_{ij} ein zulässiger (s,t)-Fluß mit $\text{val}(x') = \text{val}(x) + \varepsilon$. Also kann x nicht maximal sein.

Angenommen x besitzt keinen augmentierenden Weg. Dann sei W die Knotenmenge, die aus s und denjenigen Knoten $v \in V$ besteht, die von s aus auf einem bezüglich x augmentierenden $[s,v]$-Weg erreicht werden können. Definition (7.8) impliziert $x_a = c_a$ für alle $a \in \delta^+(W)$ und $x_a = 0$ für alle $a \in \delta^-(W)$. Daraus ergibt sich $\text{val}(x) = x(\delta^+(W)) - x(\delta^-(W)) = x(\delta^+(W)) = c(\delta^+(W))$. Aufgrund von Lemma (7.4) (b) ist somit x maximal. □

Aus Satz (7.9) folgt nun direkt, daß das lineare Programm (7.5) ganzzahlige Optimallösungen hat, falls alle Kapazitäten ganzzahlig sind.

(7.12) Satz. *Sei $D=(V,A)$ ein Digraph mit ganzzahligen Bogenkapazitäten $c_a \geq 0$, und seien $s,t \in V$, $s \neq t$. Dann gibt es einen maximalen (s,t)-Fluß, der ganzzahlig ist.*

Beweis. Wir führen einen Induktionsbeweis über die Anzahl der "Additionen" augmentierender Wege. Wir starten mit dem Nullfluß. Haben wir einen ganzzahligen Flußvektor und ist dieser nicht maximal, so bestimmen wir den Wert ε durch (7.10). Nach Voraussetzung ist ε ganzzahlig, und folglich ist der neue durch (7.11) festgelegte Flußvektor ebenfalls ganzzahlig. Bei jeder Augmentierung erhöhen wir den Flußwert um mindestens eins. Da der maximale Flußwert endlich ist, folgt die Behauptung aus (7.9). $\square$

Wir können nun den Ford-Fulkerson-Algorithmus angeben

(7.13) FORD-FULKERSON-Algorithmus.

Input: *Digraph* $D = (V, A)$ *mit Bogenkapazitäten* $c_a \in \mathbb{K}$, $c_a \geq 0$ *für alle Bögen* $a \in A$ *und zwei Knoten* $s, t \in V$, $s \neq t$.

Output: *Ein zulässiger* (s,t)*-Fluß* x *mit maximalem Wert* $\mathrm{val}(x)$ *und ein kapazitätsminimaler* (s,t)*-Schnitt* $\delta^+(W)$.

1. *(Initialisierung) Sei* $x = (x_{ij}) \in \mathbb{K}^A$ *ein zulässiger* (s,t)*-Fluß, z. B.* $x_{ij} = 0$ *für alle* $(i,j) \in A$. *Lege folgende Datenstrukturen an:*

$$\begin{aligned}
W \;& \textit{(= Menge der markierten Knoten)} \\
U \;& \textit{(= Menge der markierten aber noch nicht überprüften Knoten)} \\
\mathrm{VOR} \;& \textit{(=}\ (n-1)\textit{-Vektor, in dem der Vorgänger eines Knoten auf einem} \\
& \quad \textit{augmentierenden}\ [s,v]\textit{-Weg gespeichert wird)} \\
\mathrm{EPS} \;& \textit{(=}\ (n-1)\textit{-Vektor, zur sukzessiven Berechnung von (7.10).).}
\end{aligned}$$

Markieren und Überprüfen

2. *Setze* $W := \{s\}$, $U := \{s\}$, $\mathrm{EPS}(s) := +\infty$
3. *Ist* $U = \emptyset$, *dann gehe zu 9.*
4. *Wähle einen Knoten* $i \in U$ *aus und setze* $U := U \setminus \{i\}$.
5. *Führe für alle Bögen* $(i,j) \in A$ *mit* $j \notin W$ *folgendes aus:*

   Ist $x_{ij} < c_{ij}$, *dann setze*
   $\mathrm{VOR}(j) := +i$, $\mathrm{EPS}(j) := \min\{c_{ij} - x_{ij}, \mathrm{EPS}(i)\}$, $W := W \cup \{j\}$, $U := U \cup \{j\}$.
6. *Führe für alle Bögen* $(j,i) \in A$ *mit* $j \notin W$ *folgendes aus:*

   Ist $x_{ji} > 0$, *dann setze*
   $\mathrm{VOR}(j) := -i$, $\mathrm{EPS}(j) := \min\{x_{ji}, \mathrm{EPS}(i)\}$, $W := W \cup \{j\}$, $U := U \cup \{j\}$.
7. *Gilt* $t \in W$, *gehe zu 8, andernfalls zu 3.*

Augmentierung

8. *Konstruiere einen augmentierenden Weg und erhöhe den gegenwärtigen Fluß um* $\mathrm{EPS}(t)$, *d. h. bestimme* $j_1 = |\mathrm{VOR}(t)|$, *falls* $\mathrm{VOR}(t) > 0$, *setze* $x_{j_1 t} := x_{j_1 t} + \mathrm{EPS}(t)$, *andernfalls setze* $x_{t j_1} := x_{t j_1} - \mathrm{EPS}(t)$. *Dann bestimme* $j_2 := |\mathrm{VOR}(j_1)|$, *falls* $\mathrm{VOR}(j_1) > 0$, *setze* $x_{j_2 j_1} := x_{j_2 j_1} + \mathrm{EPS}(t)$, *andernfalls* $x_{j_1 j_2} := x_{j_1 j_2} - \mathrm{EPS}(t)$ *usw. bis der Knoten* s *erreicht ist. Gehe zu 2.*

Bestimmung eines minimalen Schnittes

9. *Der gegenwärtige (s,t)-Fluß x ist maximal und $\delta^+(W)$ ist ein (s,t)-Schnitt minimaler Kapazität. STOP.*

$\square$

Aus den Sätzen (7.9) und (7.12) folgt, daß Algorithmus (7.13) für ganzzahlige Kapazitäten korrekt arbeitet und nach endlicher Zeit abbricht. Sind die Daten rational, so kann man (wie üblich) alle Kapazitäten mit dem kleinsten gemeinsamen Vielfachen ihrer Nenner multiplizieren. Man erhält so ein äquivalentes ganzzahliges Maximalflußproblem. Also funktioniert (7.13) auch bei rationalen Daten. Läßt man (zumindest theoretisch) auch irrationale Kapazitäten zu, so kann man Beispiele konstruieren, bei denen Algorithmus (7.13) nicht nach endlicher Zeit abbricht. Aber auch bei ganzzahligen Daten gibt es Probleme. Ein Durchlauf der Markierungs- und Überprüfungsphase und der Augmentierungsphase kann offenbar in $0(m)$, $m = |A|$, Schritten durchgeführt werden. Nach jedem Durchlauf wird der Flußwert um mindestens 1 erhöht. Ist also v der Wert des maximalen (s,t)-Flusses, so ist die Laufzeit von (7.13) $0(m \cdot v)$. Diese Laufzeit ist nicht polynomial in $n + m + \sum_{a \in A} \langle c_a \rangle$, und wenn man die im Verfahren (7.13) noch nicht exakt spezifizierten Schritte ungeschickt ausführt, kann man tatsächlich zu exorbitanten Laufzeiten kommen. Allerdings haben Edmonds und Karp (1972) gezeigt:

(7.14) Satz. *Falls in Algorithmus (7.13) jeder Augmentierungsschritt entlang eines augmentierenden $[s,t]$-Weges mit minimaler Bogenzahl durchgeführt wird, dann erhält man einen Maximalfluß nach höchstens $\frac{mn}{2}$ Augmentierungen. Also ist die Gesamtlaufzeit dieser Version des Verfahrens (7.13) $0(m^2 n)$.*

$\square$

Satz (7.14) gilt für beliebige (auch irrationale) Bogenkapazitäten. Es ist in diesem Zusammenhang interessant zu bemerken, daß praktisch jeder, der Verfahren (7.13) implementiert, die Edmonds-Karp-Vorschrift einhält. Üblicherweise arbeitet man die Knoten in Breadth-First-Strategie ab. Dies führt zu augmentierenden Wegen minimaler Bogenzahl. Das heißt, man implementiert die Menge U der markierten und noch nicht abgearbeiteten Knoten als Schlange. Wird ein Knoten in Schritt 5 oder 6 zu U hinzugefügt, so kommt er an das Ende der Schlange. In Schritt 4 wird immer der Knoten $i \in U$ gewählt, der am Anfang der Schlange steht.

(7.15) Beispiel. Wir betrachten den in Abbildung 7.2 dargestellten Digraphen. Die erste Zahl des Zahlenpaares bei einem Bogen gibt den gegenwärtigen Fluß durch den Bogen an, die zweite die Kapazität des Bogens. In Abbildung 7.2 starten wir also mit einem Fluß des Wertes 10.

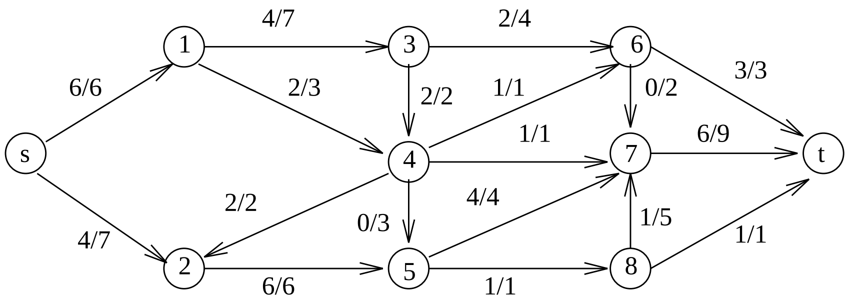

Abb. 7.2

Wir führen einen Durchlauf der Markierungs- und Überprüfungsphase vor. Im weiteren sei

$$\text{VOR} = (\text{VOR}(1), \text{VOR}(2), \ldots, \text{VOR}(8), \text{VOR}(t))$$
$$\text{EPS} = (\text{EPS}(1), \text{EPS}(2), \ldots, \text{EPS}(8), \text{EPS}(t)).$$

Das Verfahren beginnt wie folgt:

2. $W := \{s\}$, $U := \{s\}$.
3. –
4. Wir wählen $s \in U$, $U := \emptyset$.
5. $W := \{s, 2\}$, $U := \{2\}$
   $\text{VOR} = (-, +s, -, -, -, -, -, -, -)$
   $\text{EPS} = (-, 3, -, -, -, -, -, -, -)$.
6. –
7. –
3. –
4. Wir wählen $2 \in U$, $U := \emptyset$.
5. –
6. $W := \{s, 2, 4\}$, $U := \{4\}$
   $\text{VOR} = (-, +s, -, -2, -, -, -, -, -)$
   $\text{EPS} = (-, 3, -, 2, -, -, -, -, -)$.
7. –
3. –
4. Wir wählen $4 \in U$, $U := \emptyset$.
5. $W := \{s, 2, 4, 5\}$, $U := \{5\}$
   $\text{VOR} = (-, +s, -, -2, +4, -, -, -, -)$
   $\text{EPS} = (-, 3, -, 2, 2, -, -, -, -)$.
6. $W := \{s, 2, 4, 5, 1, 3\}$, $U := \{5, 1, 3\}$
   $\text{VOR} = (-4, +s, -4, -2, +4, -, -, -, -)$
   $\text{EPS} = (2, 3, 2, 2, 2, -, -, -, -)$.
7. –
3. –
4. Wir wählen $5 \in U$, $U := \{1, 3\}$.

5. –
6. –
7. –
3. –
4. Wir wählen $1 \in U$, $U := \{3\}$.
5. –
6. –
7. –
3. –
4. Wir wählen $3 \in U$, $U := \emptyset$.
5.
6. –
7. –
3. –
5. $W := \{s, 2, 4, 5, 1, 3, 6\}$, $U := \{6\}$
   $\mathrm{VOR} = (-4, +s, -4, -2, +4, +3, -, -, -)$
   $\mathrm{EPS} = (2, 3, 2, 2, 2, 2, -, -, -)$.
6. –
7. –
3. –
4. Wir wählen $6 \in U$, $U := \emptyset$.
5. $W := \{s, 2, 4, 5, 1, 3, 6, 7\}$, $U := \{7\}$
   $\mathrm{VOR} = (-4, +s, -4, -2, +4, +3, +6, -, -)$
   $\mathrm{EPS} = (2, 3, 2, 2, 2, 2, 2, -, -)$.
6. –
7. –
3. –
4. Wir wählen $7 \in U$, $U := \emptyset$.
5. $W := \{s, 2, 4, 5, 1, 3, 6, 7, t\}$, $U := \{t\}$
   $\mathrm{VOR} = (-4, +s, -4, -2, +4, +3, +6, -, +7)$
   $\mathrm{EPS} = (2, 3, 2, 2, 2, 2, 2, -, 2)$.
6. (Hier wird noch 8 markiert, das ist aber irrellevant, da t bereits markiert ist)
7. $t \in W$
8. Es gilt

$$\begin{aligned}
\mathrm{VOR}(t) &= +7 \\
\mathrm{VOR}(7) &= +6 \\
\mathrm{VOR}(6) &= +3 \\
\mathrm{VOR}(3) &= -4 \\
\mathrm{VOR}(4) &= -2 \\
\mathrm{VOR}(2) &= +s
\end{aligned}$$

also ist der augmentierende Weg mit $\mathrm{EPS}(t) = 2$ der folgende $(s, 2)$, $(4, 2)$, $(3, 4)$, $(3, 6)$, $(6, 7)$, $(7, t)$. Der neue Fluß ist in Abbildung 7.3 dargestellt Dieser (s, t)-Fluß ist maximal, ein (s, t)-Schnitt minimaler Kapazität ist $\delta^+(\{s, 2\})$, ein anderer $\delta^+(\{s, 1, 2, 3, 4, 5\})$. $\blacksquare$

$\square$

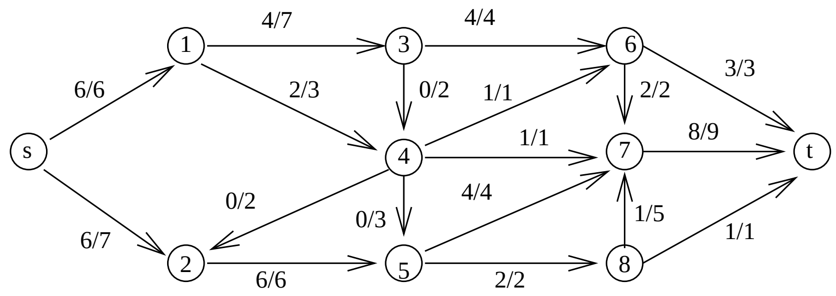

Abb. 7.3

7.3 Der Dinic-Malhorta-Kumar-Maheshwari-Algorithmus.

In diesem Abschnitt beschreiben wir einen Algorithmus zur Lösung des Maximalflußproblems, dessen Grundgerüst auf Dinic (1970) zurückgeht. Die wichtigste Phase dieses Algorithmus wurde in Malhorta, Pramodh Kumar und Maheshwari (1978) entscheidend verbessert. Wir wollen den im nachfolgenden dargestellten Algorithmus nach den vier oben aufgeführten Autoren benennen. Zur Abkürzung schreiben wir einfach **DMKM-Algorithmus**. Aus Zeitgründen wird nur eine relativ informelle Beschreibung des DMKM-Algorithmus gegeben. Sie basiert auf Syslo, Deo und Kowalik (1983). Eine sehr detaillierte Analyse des Verfahrens mit allen notwendigen Korrektheitsbeweisen und einer sorgfältigen Abschätzung der Laufzeit kann man in Mehlhorn (1984) finden.

Der DMKM-Algorithmus verfolgt die folgende generelle Strategie. Statt wie im Ford-Fulkerson-Algorithmus im gesamten Digraphen D mit gegebenem (s, t)-Fluß x einen augmentierenden $[s, t]$-Weg zu suchen, wird aus D und x ein azyklischer Digraph N (genannt geschichtetes Netzwerk) konstruiert, der mit weniger Zeitaufwand zu bearbeiten ist. Ein besonderes Merkmal dieser Konstruktion ist, daß N genau dann den Knoten t nicht enthält, wenn der gegenwärtige (s, t)-Fluß x maximal ist. Ist t in N enthalten, so versucht man einen möglichst großen (s, t)-Fluß in N zu finden. Nach Konstruktion kann jeder (s, t)-Fluß in N zum (s, t)-Fluß x in D “augmentiert” werden, und man kann ein neues geschichtetes Netzwerk N bestimmen. Da die Bestimmung eines maximalen (s, t)-Flusses in N zu aufwendig ist, begnügt man sich mit der Bestimmung eines sogenannten saturierten (s, t)-Flusses in N. Ein solcher ist relativ leicht zu finden. Das Verfahren zur Auffindung eines saturierten (s, t)-Flusses in N garantiert, daß ein (s, t)-Fluß mit positivem Wert in N gefunden wird, falls ein solcher existiert. Daraus folgt bereits, daß das Verfahren funktioniert. Eine weitere Besonderheit ist die Methode zur sukzessiven Verbesserung des (s, t)-Flusses in N bis zur Erreichung eines saturierten (s, t)-Flusses. Hier wird nicht bogenorientiert gearbeitet; man versucht stattdessen, durch die vorhandenen Knoten soviel Flußwert wie möglich “durchzuschieben”. Dabei bestimmt man einen Flußwert, der mit Sicherheit zum gegenwärtigen

Fluß “augmentiert” werden kann; ferner ist dieser Fluß relativ einfach auf die Bögen zu verteilen. Bevor wir die Phasen des DMKM-Algorithmus einzeln beschreiben, fassen wir das formale Konzept des Verfahrens zusammen.

(7.16) DMKM-Algorithmus (Überblick).

Input: *Digraph $D = (V, A)$ mit nichtnegativen Bogenkapazitäten c_a für alle $a \in A$, zwei Knoten $s, t \in V$.*

Output: *Maximaler (s, t)-Fluß x.*

1. *Setze $x_a = 0$ für alle $a \in A$.*
2. *Konstruiere aus D und x ein geschichtetes Netzwerk $N = (W, B)$ mit Bogenkapazitäten $\overline{c}_a$ für alle $a \in B$.*
3. *Ist $t \notin W$, STOP. Der gegenwärtige (s, t)-Fluß ist maximal.*
4. *Bestimme einen saturierten (s, t)-Fluß $\overline{x}$ in $N = (W, B)$.*
5. *Erhöhe den Fluß x durch $\overline{x}$ und gehe zu 2.*

$\square$

Saturierte (s, t)-Flüsse in geschichteten Netzwerken

Wir beginnen mit der Darstellung eines wichtigen Teils des DMKM-Algorithmus, der Bestimmung eines saturierten Flussses in einem geschichteten Netzwerk.

Es sei $N = (W, B)$ ein Digraph mit nicht-negativen Bogenkapazitäten $\overline{c}(a)$ für alle $a \in B$, und s, t seien zwei voneinander verschiedene Knoten von N. Zusätzlich nehmen wir an, daß N azyklisch ist, also keinen gerichteten Kreis enthält, und daß die Knotenmenge W in “Schichten” $V_1, \ldots, V_k$ mit folgenden Eigenschaften zerlegt werden kann:

$$
\begin{aligned}
V_1 &= \{s\},\\
V_i &= \{v \in W \mid \exists\, u \in V_{i-1} \text{ mit } (u, v) \in B\} \quad i = 2, \ldots, k-1,\\
V_k &= \{t\},\\
B &= \bigcup_{i=1}^{k-1} \{(u, v) \in B \mid u \in V_i, v \in V_{i+1}\}.
\end{aligned}
\tag{7.17}
$$

Einen derartigen Digraphen nennnen wir ein **geschichtetes Netzwerk**. In einem geschichteten Netzwerk, sind also Knoten aus V_i nicht durch einen Bogen verbunden, ebenso gibt es für Knoten in V_i und in V_j mit $i < j - 1$ keinen Bogen, der sie miteinander verbindet. Für jeden Knoten $v \in V_i$ haben alle (s, v)-Wege die Länge $i - 1$, speziell haben alle (s, t)-Wege die Länge $k - 1$. Für einen Bogen $(u, v) \in B$ gibt es immer einen Index i mit $u \in V_i$, $v \in V_{i+1}$.

Ist x ein (s, t)-Fluß in einem (beliebigen) Digraphen $D = (V, A)$ mit Bogenkapazitäten $c(a)$, so nennen wir den **Bogen** $a \in A$ **saturiert**, falls $x_a = c_a$ gilt. Ein (s, t)-**Weg** P heißt **saturiert** (bzgl. x), falls mindestens ein Bogen aus P saturiert ist. Der (s, t)-**Fluß** x in D heißt **saturiert**, wenn jeder (s, t)-Weg in D saturiert ist. Man beachte, daß jeder maximale Fluß saturiert ist, daß aber nicht jeder saturierte Fluß auch maximal ist (vergleiche Abbildung 7.1). Wir zeigen nun, wie man einen saturierten Fluß in einem geschichteten Netzwerk bestimmen kann.

Sei also $N = (W, B)$ ein geschichtetes Netzwerk mit Schichten $V_1, \ldots, V_k$ und Bogenkapazitäten $\overline{c}(a) \geq 0$. Ferner sei x ein zulässiger (s,t)-Fluß in N. Im DMKM-Algorithmus wird für jeden Knoten $v \in W \setminus \{s,t\}$ der maximale zusätzliche Flußwert, der durch den Knoten v geschickt werden kann, bestimmt. Diesen Wert nennen wir **Potential** von v und bezeichnen ihn mit $\text{POT}(v)$. Er kann wie folgt bestimmt werden.

$$(7.18) \qquad \text{POT}(v) := \min\{\sum_{a\in\delta^-(v)} (\overline{c}(a) - x(a)), \sum_{a\in\delta^+(v)} (\overline{c}(a) - x(a))\},\ v \in W \setminus \{s,t\}.$$

Unter allen Knoten $v \in W \setminus \{s,t\}$ wird ein Knoten mit minimalem Potential bestimmt. Dieser Knoten, sagen wir r, wird **Referenzknoten** genannt, d. h.

$$(7.19) \qquad \text{POT}(r) = \min\{\text{POT}(v) \mid v \in W \setminus \{s,t\}\}.$$

Abbildung 7.4 zeigt ein geschichtetes Netzwerk mit 5 Schichten und 10 Knoten; es gilt $s = 1$, $t = 10$. Wie üblich gibt die erste Zahl eines Zahlenpaares bei einem Bogen den gegenwärtigen Fluß durch den Bogen und die zweite Zahl die Bogenkapazität an. Aus (7.18) folgt

$$\begin{aligned} &\text{POT}(2) = 4,\ \text{POT}(3) = 4,\ \text{POT}(4) = 5,\ \text{POT}(5) = 3,\\ &\text{POT}(6) = 9,\ \text{POT}(7) = 4,\ \text{POT}(8) = 4,\ \text{POT}(9) = 5. \end{aligned}$$

Der Referenzknoten ist somit eindeutig bestimmt, es ist der Knoten 5.

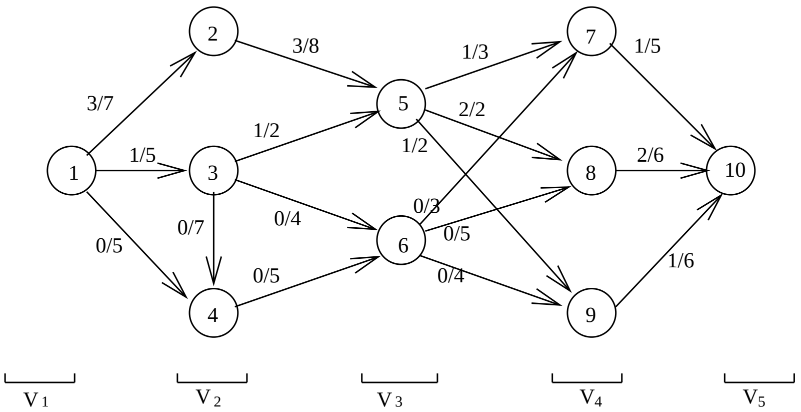

Abb. 7.4

Wir wollen nun vom Referenzknoten r ausgehend den zusätzlichen Flußwert $\text{POT}(r)$ durch das gesamte Netzwerk schicken. Wir nehmen an, daß er in der Schicht V_i liegt. Zunächst verteilen wir $\text{POT}(r)$ auf die Bögen, die aus r hinausgehen. Dies ist wegen $\text{POT}(r) \leq \sum_{a\in\delta^+(r)}(\overline{c}(a) - x(a))$ möglich. Der zusätzliche Fluß $\text{POT}(r)$ kommt also in der Schicht V_{i+1} an. Für jeden Knoten $v \in V_{i+1}$ verteilen wir die Flußmenge, die in v

ankommt auf die Bögen, die aus v herausgehen. Aufgrund von (7.18) ist auch dies möglich. Damit gelangt der gesamte Wert POT(r) zur Schicht V_{i+2}, und wir fahren so fort bis der Fluß POT(r) den Zielknoten $t \in V$ erreicht. Das Verteilen von POT(r) auf die verschiedenen Bögen erfolgt durch einfache Listenverarbeitung (z. B. Breadth-First).

Nun gehen wir rückwärts, um aus s den Flußwert POT(r) "herauszuziehen". Wir verteilen den Flußwert POT(r) auf alle Bögen, die in r enden. Für alle Bögen $v \in V_{i-1}$ verteilen wir den Anteil des Flußwertes POT(r), der rückwärts in v angekommen ist, auf die Bögen, die in v enden. Die Wahl des Referenzknoten r garantiert, daß dies möglich ist. Und wir machen auf diese Weise weiter, bis wir zur Quelle s gelangt sind. Durch dieses "Durchschieben" und "Herausziehen" wird also der Wert des Ausgangsflusses in N um POT(r) erhöht. Wenden wir dieses Verfahren auf unser Beispielnetzwerk aus Abbildung 7.4 an, so erhalten wir den in Abbildung 7.5 dargestellten Fluß.

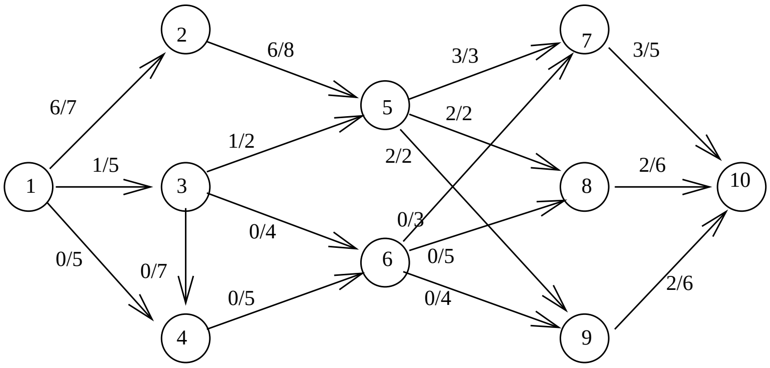

Abb. 7.5

Alle Bögen mit $x_a = \overline{c}_a$ sind saturiert. Ist der gegenwärtige (s,t)-Fluß in N nicht saturiert, muß es noch einen nicht saturierten (s,t)-Weg in N geben. Kein nicht saturierter (s,t)-Weg in N kann einen saturierten Bogen enthalten. Daher können wir alle saturierten Bögen aus N entfernen. Aufgrund unseres Vorgehens gilt für den Referenzknoten r, daß entweder alle Bögen aus $\delta^-(r)$ oder alle Bögen aus $\delta^+(r)$ saturiert sind (in unserem Beispiel aus Abbildung 7.5 sind alle Bögen aus $\delta^+(5)$ saturiert). Falls es also einen noch nicht saturierten (s,t)-Weg geben sollte, kann dieser den Knoten r nicht enthalten. Wir können also auch den Knoten r und alle weiteren Bögen, die r als Anfangs- oder Endknoten enthalten, aus N entfernen. Die Herausnahme von r und der saturierten Bögen kann bewirken, daß ein Knoten $v \in W$ auf keinem (s,t)-Weg in dem so reduzierten Digraphen liegt. (Dies gilt in Abbildung 7.5 nach Wegnahme der saturierten Bögen und des Knoten 5 für den Knoten 2.) Alle derartigen Knoten und die mit ihnen inzidenten Bögen können ebenfalls aus N entfernt werden. Wir entfernen Knoten und Bögen so lange, bis jede Menge $\delta^-(v)$ und $\delta^+(v)$ der noch verbleibenden Knoten v mindestens eine nicht saturierte Kante enthält. Werden s oder t entfernt, können wir aufhören, denn dann ist der gegenwärtige Fluß offenbar saturiert. Damit ist eine Iteration des Verfahrens zur Bestimmung eines saturierten (s,t)-Flusses beschrieben.

Diese Herausnahme von Knoten und Bögen aus N induziert natürlich eine Änderung der Knotenpotentiale der vorhandenen Knoten. Wir können somit mit den reduzierten Netzwerken von neuem beginnen, durch (7.18) die Potentiale bestimmen und mit der obigen Methode fortfahren.

Das geschichtete Netzwerk, das aus dem Netzwerk aus Abbildung 7.5 durch die obige Reduktionsmethode entsteht, ist in Abbildung 7.6 dargestellt.

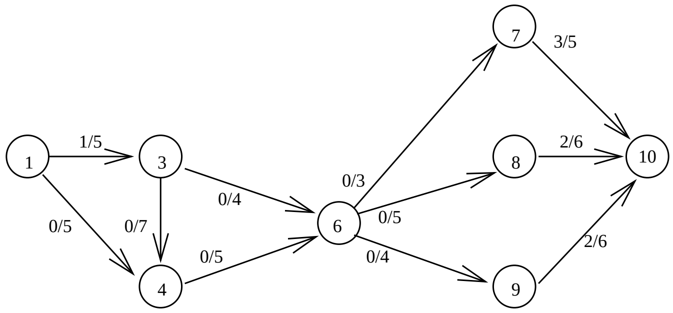

Abb. 7.6

Man beachte, daß die in Abbildung 7.6 angegebenen Flußwerte keinen zulässigen Fluß des Digraphen 7.6 darstellen. Fügen wir die aus dem Digraphen entfernten Knoten und Bögen mit den für sie bereits bestimmten Flußwerten hinzu, so erhalten wir einen zulässigen Fluß im ursprünglichen Digraphen aus Abbildung 7.5. Die Berechnung der Potentiale der Knoten aus Abbildung 7.6 ergibt

$$\begin{aligned} &\mathrm{POT}(3) = 4,\ \mathrm{POT}(4) = 5,\ \mathrm{POT}(6) = 9,\\ &\mathrm{POT}(7) = 2,\ \mathrm{POT}(8) = 4,\ \mathrm{POT}(9) = 4. \end{aligned}$$

Als Referenzknoten müssen wir nun den Knoten 7 wählen und POT(7) aus $s = 1$ "herausziehen" bzw. in $t = 10$ "hineinschieben".

Damit ist das Verfahren zur Bestimmung eines saturierten (s,t)-Flusses in geschichteten Netzwerken beschrieben. Wir starten zunächst mit dem Nullfluß, bestimmen die Potentiale aller Knoten $v \in V \setminus \{s,t\}$, wählen das kleinste Potential mit dem Referenzknoten r und schieben POT(r) durch das Netzwerk. Dann reduzieren wir das Netzwerk um alle saturierten Bögen (und einige Knoten) und beginnen von neuem. Dies führen wir so lange durch, bis es im gegenwärtigen reduzierten Digraphen keinen Weg von s nach t mehr gibt. Da bei jeder Reduktion mindestens ein Knoten entfernt wird, ist dies nach spätestens $n-1$ Iterationen der Fall.

Bestimmung eines geschichteten Netzwerks

Wir wissen nun, wie in einem geschichteten Netzwerk ein saturierter (aber nicht notwendig maximaler) (s,t)-Fluß bestimmt werden kann. Diese Methode wollen wir uns zunutze machen, um in einem beliebigen Digraphen einen maximalen Fluß zu konstruieren. Dabei gehen wir in mehreren Stufen wie folgt vor.

Zunächst konstruieren wir aus dem gegebenen Digraphen D ein geschichtetes Netzwerk, sagen wir N. Vom Nullfluß ausgehend bestimmen wir einen saturierten (s,t)-Fluß x

in N. Diesen (s,t)-Fluß x können wir (nach Konstruktion) in einen (s,t)-Fluß in D transformieren. Aus D und dem derzeitigen Fluß x konstruieren wir ein neues geschichtetes Netzwerk N' und bestimmen wiederum einen saturierten (s,t)-Fluß x' in N'. Den Fluß x' kann man (wie bei der Addition augmentierender Wege) zu x hinzufügen und erhält einen (s,t)-Fluß in D mit Wert $\text{val}(x)+\text{val}(x')$. Nun bestimmen wir aus D und dem neuen (s,t)-Fluß ein weiteres geschichtetes Netzwerk und fahren so fort, bis aus D und dem gegenwärtigen (s,t)-Fluß kein neues geschichtetes Netzwerk mehr konstruiert werden kann, das den Knoten t enthält. Die Regeln zur Konstruktion des geschichteten Netzwerkes aus D und dem gegenwärtigen Fluß sind so gestaltet, daß es dann zu D und dem gegenwärtigen (s,t)-Fluß keinen augmentierenden (s,t)-Weg mehr gibt. Aus Satz (7.9) folgt dann, daß der gegenwärtige (s,t)-Fluß maximal ist. Aus der letzten Bemerkung ist klar, wie wir bei der Auswahl der Bögen vorzugehen haben. Die Schichten des Netzwerkes werden nach dem Breadth-First-Prinzip bestimmt.

Ist also $D=(V,A)$ mit Bogengewichten $c(a)$ für alle $a \in A$ ein Digraph, sind s, t zwei verschiedene Knoten aus D, und ist x ein zulässiger (s,t)-Fluß, dann wird aus D, c und x ein geschichtetes Netzwerk $N=(W,F)$ mit Kapazitäten $\overline{c}(a)$ wie folgt konstruiert:
Es seien

(7.20)
$$\begin{aligned} A_1 &:= \{(u,v) \mid (u,v) \in A \text{ und } x_{uv} < c_{uv}\},\\ A_2 &:= \{(v,u) \mid (u,v) \in A \text{ und } x_{uv} > 0\}, \end{aligned}$$

(Auf den Bögen aus A_1 kann der Fluß erhöht, auf denen aus A_2 erniedrigt werden. Ein Bogen kann sowohl in A_1 als auch in A_2 sein. Mit $A_1 \dot\cup A_2$ bezeichnen wir die "Vereinigung" von A_1 und A_2, bei der ein Bogen zweimal auftreten kann. Ist $(u,v) \in A$ mit $x_{uv} < c_{uv}$, so schreiben wir auch $(u,v)_1$, um den zu (u,v) gehörigen Bogen aus A_1 zu bezeichnen. Ist $(u,v) \in A$ mit $x_{uv} > 0$, so schreiben wir auch analog $(v,u)_2$, um den zugehörigen Bogen (v,u) aus A_2 zu bezeichnen.)

(7.21)
$$\begin{aligned} \overline{c}_{uv} &:= c_{uv} - x_{uv} &&\text{für alle } (u,v) \in A_1,\\ \overline{c}_{uv} &:= x_{vu} &&\text{für alle } (u,v) \in A_2. \end{aligned}$$

Nun sei $V_1 := \{s\}$, und ist V_i bestimmt, dann sei

(7.22)
$$V_{i+1} := \{v \in V \setminus (V_1 \cup \ldots \cup V_i) \mid \exists\, u \in V_i \text{ mit } (u,v) \in A_1 \dot\cup A_2\}.$$

Sobald eine Knotenmenge, sagen wir V_k, den Knoten t enthält, brechen wir ab und setzen $V_k := \{t\}$ und $W' := \bigcup_{i=1}^k V_i$. Ferner sei

$$F' := (A_1 \cap \bigcup_{i=1}^{k-1} (V_i \times V_{i+1})) \mathbin{\dot\cup} (A_2 \cap \bigcup_{i=1}^{k-1} (V_i \times V_{i+1})).$$

Der Digraph (W',F') ist ein geschichtetes Netzwerk mit den in (7.21) definierten Bogenkapazitäten. Man beachte, daß $\delta^-(v) \neq \emptyset$ für alle Knoten $v \in W' \setminus \{s\}$ gilt; es kann aber

Knoten $v \in W' \setminus \{t\}$ geben mit $\delta^+(v) = \emptyset$. Solche Knoten können wir entfernen. Wir tun dies sukzessive und erhalten das "endgültige" geschichtete Netzwerk (W, F) durch Entfernen von Knoten und Bögen wie im vorigen Abschnitt angegeben.

Betrachten wir den in Abbildung 7.7 dargestellen kapazitierten Digraphen mit dem angegebenen (s, t)-Fluß.

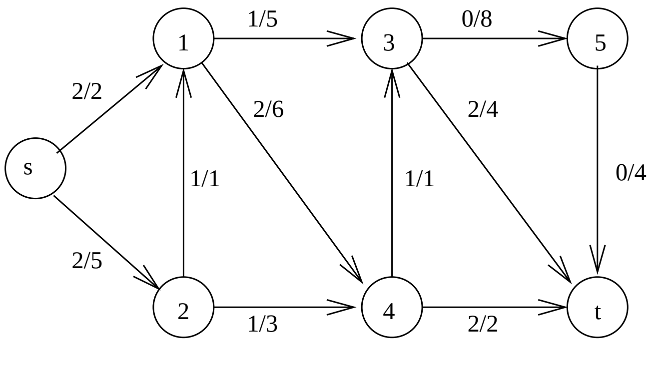

Abb. 7.7

In Abbildung 7.8 sind die in (7.20) definierten zugehörigen Bogenmengen A_1 und A_2 angegeben mit den durch (7.21) definierten Kapazitäten. Die Bögen aus A_2 sind gestrichelt gezeichnet, die aus A_1 mit durchgehenden Linien.

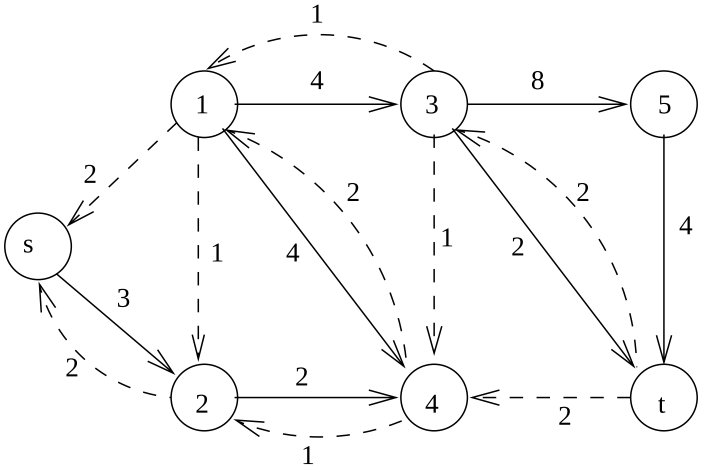

Abb. 7.8

Mit dem oben beschriebenen Verfahren erhalten wir nun das in Abbildung 7.9 dargestellte geschichtete Netzwerk mit 6 Schichten.

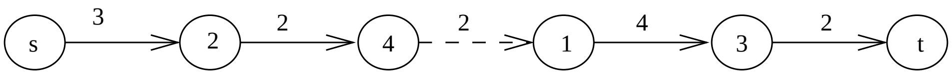

Abb. 7.9

Das geschichtete Netzwerk N ist so konstruiert, daß jeder zulässige (s, t)-Fluß in N eine Vereinigung augmentierender (s, t)-Wege in D ist, genauer:

(7.23) Satz. *Sei $D = (V, A)$ ein kapazitierter Digraph, und $N = (W, F)$ sei das oben aus*

D und dem zulässigen (s,t)-Fluß x konstruierte geschichtete Netzwerk mit den Kapazitäten $\overline{c}$, dann gilt:

(a) x ist ein maximaler (s,t)-Fluß in D genau dann, wenn $t \notin W$.

(b) Ist $\overline{x}$ ein zulässiger (s,t)-Fluß in N, dann ist $x' \in \mathbb{K}^A$ mit

$$x'_a := x_a + \overline{x}_{a_1} - \overline{x}_{a_2} \qquad \text{für alle } a = (u,v) \in A$$

ein zulässiger (s,t)-Fluß in D mit Wert $\operatorname{val}(x) + \operatorname{val}(\overline{x})$.

(Die Formel in (b) zur Berechnung von x'_{uv} ist wie folgt zu interpretieren. Ist $a = (u,v) \in A$ und $0 < x_{uv} < c_{uv}$, so ist $a_1 = (u,v) \in A_1$ und $a_2 = (v,u) \in A_2$ und die Formel ist wohldefiniert. Ist $x_a = 0$ so ist $(v,u) \notin A_2$ und $\overline{x}_{vu}$ ist als Null zu betrachten. Ist $x_{uv} = c_{uv}$, so gilt $(u,v) \notin A_1$ und $\overline{x}_{uv}$ ist als Null zu betrachten.)

Beweis. (b) Wir müssen zeigen, daß x' die Kapazitäts- und die Flußerhaltungsbedingungen (7.1), (7.2) erfüllt. Sei $(u,v) \in A$ beliebig, dann gilt

$$\begin{array}{llll}
0 \le x_{uv} - \overline{x}_{vu}, & \text{da} & x_{uv} & = \overline{c}_{vu} \ge \overline{x}_{vu} \\
\le x_{uv} + \overline{x}_{uv} - \overline{x}_{vu}, & \text{da} & \overline{x}_{uv} & \ge 0 \\
= x'_{uv} & & & \\
\le x_{uv} + \overline{x}_{uv}, & \text{da} & \overline{x}_{vu} & \ge 0 \\
\le c_{uv}, & \text{da} & \overline{c}_{uv} & = c_{uv} - x_{uv} \ge \overline{x}_{uv}.
\end{array}$$

Also ist (7.1) erfüllt. Sei nun $v \in V \setminus \{s,t\}$ ein beliebiger Knoten.

$$\begin{aligned}
x'(\delta^-(v)) - x'(\delta^+((v)) &= x(\delta^-(v)) + \overline{x}(\delta^-(v) \cap A_1) - \overline{x}(\delta^+(v) \cap A_2) \\
&\quad - x(\delta^+(v)) - \overline{x}(\delta^+(v) \cap A_1) + \overline{x}(\delta^-(v) \cap A_2) \\
&= (x(\delta^-(v)) - x(\delta^+(v))) + (\overline{x}(\delta^-(v)) - \overline{x}(\delta^+(v))) = 0 + 0 = 0.
\end{aligned}$$

Der Wert von x' ist offenbar $\operatorname{val}(x) + \operatorname{val}(\overline{x})$.

(a) Die Behauptung folgt direkt aus (b) und Satz (7.9). □

Damit ist der DMKM-Algorithmus vollständig beschrieben. Unter Verwendung geeigneter Datenstrukturen kann man folgendes zeigen (siehe Mehlhorn (1984)):

(7.24) Satz. *Der DMKM-Algorithmus findet in einem kapazitierten Digraphen mit n Knoten und m Bögen einen maximalen (s,t)-Fluß in $0(n^3)$ Schritten.*

□

Eine Implementation des DMKM-Algorithmus in PASCAL, in der alle oben informell dargestellten Details explizit ausgeführt sind, kann man in Syslo, Deo und Kowalik (1983) finden.

7.4 Ein generischer Präfluß-Algorithmus

In seiner Doktorarbeit aus dem Jahre 1987 und einigen z. T. vorher erschienenen gemeinsamen Veröffentlichungen mit anderen Autoren hat A. V. Goldberg einige neue und einige bekannte Ideen auf originelle Weise kombiniert und dabei die Grundlagen zu einer neuen Klasse von Maximalfluß-Algorithmen gelegt. In einer Vielzahl von Folgeveröffentlichungen verschiedener Autoren sind diese Ideen ergänzt und verfeinert worden, und die Laufzeiten der Algorithmen sind durch verbesserte Datenstrukturen verringert worden. Diese Algorithmen werden in der englischen Literatur häufig preflow-push- oder push-relabel-Algorithmen genannt. Wir nennen sie hier Präfluß-Algorithmen. Überblicke zu diesem Themenkreis finden sich u.a. in den Artikeln Ahuja, Magnanti und Orlin (1989) und Goldberg, Tardos und Tarjan (1990) sowie in dem Buch Ahuja, Magnanti und Orlin (1993). Implementierungswettbewerbe in letzter Zeit deuten an, daß die besten Präfluß–Algorithmen bei großen Graphen in der Praxis schneller sind als die bisher bekannten Verfahren, wobei nicht unbedingt die bezüglich der worst-case-Laufzeit schnellsten Algorithmen auch in der praktischen Ausführung am schnellsten sind. Die Entwicklung — speziell hinsichtlich praktisch effizienter Implementierungen — ist noch nicht abgeschlossen.

Es hat sich aus Notationsgründen eingebürgert, Präfluß–Algorithmen etwas anders darzustellen als die übrigen Maximalfluß–Verfahren. Wir folgen hier diesem Brauch.

Wie immer ist ein Digraph $D = (V, A)$ mit nichtnegativen Bogenkapazitäten $c(a)$ gegeben. (Wir lassen wie üblich zu, daß $c(a)$ auch den Wert ∞ annehmen kann.) Wir verlangen, daß $D = (V, A)$ **keine parallelen Bögen** besitzt, also **einfach** ist, und ferner, daß D zu jedem Bogen (u, v) auch seinen "Gegenbogen" (v, u) enthält. Digraphen mit letzterer Eigenschaft heißen **symmetrisch**.

Wenn (wie üblich) ein beliebiger Digraph gegeben ist, so können wir ihn in diese Standardform wie folgt transformieren. Parallele Bögen ersetzen wir durch einen einzigen Bogen, wobei wir die entsprechenden Bogenkapazitäten aufaddieren. Ist zu einem Bogen (u, v) noch kein Gegenbogen (v, u) vorhanden, so fügen wir (v, u) zum gegenwärtigen Digraphen mit der Kapazität $c(v, u) = 0$ hinzu. Jedem Maximalfluß in diesem transformierten symmetrischen und einfachen Digraphen entspricht offensichtlich ein Maximalfluß im ursprünglichen Digraphen und umgekehrt. Ab jetzt gehen wir daher in diesem Abschnitt davon aus, daß Digraphen symmetrisch und einfach sind.

Ein **Pseudofluß** auf $D = (V, A)$ ist eine Abbildung $x : A \to \mathbb{K}$ mit den folgenden Eigenschaften

$$x(u, v) \leq c(u, v) \quad \text{für alle} \quad (u, v) \in A \tag{7.25}$$

$$x(u, v) = -x(v, u) \quad \text{für alle} \quad (u, v) \in A \tag{7.26}$$

In der **Kapazitätsbeschränkung** (7.25) verzichtet man hier auf die übliche Nichtnegativitätsschranke, weil die **Antisymmetrie-Bedingung** (7.26) für den Pseudofluß eine notationstechnische Erleichterung bringt. Hinter (7.26) steckt keine tiefsinnige Überlegung!

Ist ein Pseudofluß x gegeben, so definiert man eine **Überschuß-Abbildung** $e : V \to \mathbb{K}$ (e steht für "excess") durch

$$(7.28) \qquad e(v) := \sum_{a \in \delta^-(v)} x(a) \quad \text{für alle} \quad v \in V.$$

Beim Vorliegen mehrerer Pseudoflüsse, sagen wir x und x', schreibt man $e_x(v)$ bzw. $e_{x'}(v)$, um zu kennzeichnen bzgl. welches Pseudoflusses der Überschuß definiert ist. Ist $e(v)$ nichtnegativ, so spricht man von einem **Überschuß** bei v, sonst von einem **Defizit**. Die **Residualkapazität** bezüglich x ist eine Abbildung $r := A \to \mathbb{K}$ definiert durch

$$(7.29) \qquad r(u,v) := c(u,v) - x(u,v) \quad \text{für alle} \quad (u,v) \in A.$$

Wiederum schreibt man $r_x(u,v)$, wenn man den betrachteten Pseudofluß hervorheben will. Der **Residualgraph** bezüglich x ist der Digraph $D_x = (V, A_x)$ mit $A_x := \{(u,v) \in A \mid r_x(u,v) > 0\}$.

Ein (s,t)-Präfluß oder kurz **Präfluß** ist ein Pseudofluß x, bei dem für alle Knoten $v \in V \setminus \{s,t\}$ der Überschuß $e_x(v)$ nichtnegativ ist. Die Flußerhaltungsbedingung (7.2) liest sich in der gegenwärtigen Notation $e(v) = 0$. Also ist ein (s,t)-Fluß ein Pseudo- (oder Prä-) Fluß mit $e(v) = 0$ für alle $v \in V \setminus \{s,t\}$.

Wir beschreiben nun einen generischen Präfluß-Algorithmus, in Anlehnung an Goldberg und Tarjan (1988).

(7.30) Definition. *Sei $D = (V,A)$ ein Digraph mit Kapazitäten $c(a) \geq 0$ für alle $a \in A$, seien $s,t \in V, s \neq t$, und sei x ein (s,t)-Präfluß . Eine* **Entfernungsmarkierung** *(distance labeling) ist eine Funktion $d : V \to \mathbf{Z}_+$ mit den Eigenschaften:*

$$\begin{aligned} &d(t) = 0, \quad d(s) = n = |V|, \\ &d(u) \leq d(v) + 1 \qquad \text{für alle Bögen} \quad (u,v) \in A_x \quad \text{des Residualdigraphen} \end{aligned}$$

$\square$

Hinter dieser Begriffsbildung liegt der folgende Gedanke. Wir setzen die "Länge" aller Bögen $a \in A_x$ des Residualdigraphen D_x mit 1 fest, nur die Länge von (s,t) wird n gesetzt. Für jeden Knoten $v \in V \setminus \{t\}$ können wir dann seine Entfernung $\text{dist}_{D_x}(v,t)$ zur Senke t berechnen, also die Länge des kürzesten (v,t)-Weges in D_x. Man kann zeigen, daß für alle $v \in V$ der Wert $d(v)$ eine untere Schranke dieser Enfernung $\text{dist}_{D_x}(v,t)$ ist.

Der generische Präfluß-Algorithmus hat zu jeder Zeit der Algorithmus-Ausführung einen Präfluß x und eine Entfernungsmarkierung d. Er schreibt x und d fort, wobei Schuboperationen (push) und Markierungsänderungen (relabel) vorgenommen werden. Um diese beschreiben zu können, treffen wir noch einige Definitionen. Wir nennen einen Knoten v **aktiv**, wenn $v \notin \{s,t\}$ und $e_x(v) > 0$ gilt. Ein Präfluß ist also ein Fluß, wenn kein Knoten aktiv ist. Ein Bogen heißt **erlaubt**, wenn $(u,v) \in A_x$ (also im Residualdigraphen) ist und $d(u) = d(v) + 1$ gilt.

Der generische Präfluß-Algorithmus beginnt mit dem Präfluß, der auf den Bögen $a \in \delta^+(s)$ den Wert $x(a) = c(a)$ und auf allen übrigen den Wert Null hat. Der Algorithmus führt dann in beliebiger Reihenfolge die Fortschreibungsoperationen aus. Formal kann man dies wie folgt beschreiben.

(7.31) Generischer Präfluß-Algorithmus.

Input: *(einfacher, symmetrischer) Digraph* $D = (V, A)$ *mit Kapazitäten* $c(a) \geq 0$ *für alle* $a \in A$ *und* $s, t \in V, s \neq t$.

Output: *maximaler* (s,t)*-Fluß* x.

Inititialisierung:

$$\begin{aligned}
\text{Setze}\quad x(s,v) &:= c(s,v) && \text{für alle} && (s,v) \in \delta^+(s),\\
x(v,s) &:= -c(s,v) && \text{für alle} && (v,s) \in \delta^-(s),\\
x(a) &:= 0 && \text{für alle} && a \in A \setminus (\delta^+(s) \cup \delta^-(s)).\\
\text{Setze}\quad e(v) &:= \sum_{a \in \delta^-(v)} x(a) && \text{für alle} && v \in V,\\
d(s) &:= n,\\
d(v) &:= 0 && \text{für alle} && v \in V \setminus \{s\}.
\end{aligned}$$

Schleife: *Solange es noch* **aktive** *Knoten gibt, wähle einen aktiven Knoten* u*. Falls* $r_x(u,v) = 0$ *und* $d(u) \leq d(v)$ *für alle Nachfolger* v *von* u *(d.h.* $v \in N^t(u)$*) gilt, führe RELABEL* (u) *aus. Andernfalls wähle einen Knoten* $V \in N^t(u)$*, so daß der Bogen* (u,v) *erlaubt ist, und führe PUSH* (u,v) *aus.*

RELABEL (u)**:** *Ersetze* $d(u)$ *durch* $\min\{d(v) \mid (u,v) \in A_x\} + 1$.

PUSH (u,v)**:** *Erhöhe den Fluß auf den Bogen* (u,v) *um den Werte* δ*, wobei* $0 \leq \delta \leq \min\{e_x(u), r_x(u,v)\}$ *gelten muß.*

□

Die PUSH-Operation schiebt also zusätzlichen Fluß durch einen Bogen. Im allgemeinen, und so machen wir es hier auch, wird man natürlich den Fluß um den größtmöglichen Wert $\delta := \min\{e(v), r(u,v)\}$ erhöhen; es gibt jedoch Skalierungsalgorithmen, bei denen kleinere Werte gewählt werden. Falls $\delta = r(u,v)$, so nennt man einen solchen Schub **saturierend**. Nach einem saturierenden Schub ist ein Bogen nicht mehr erlaubt, da der Fluß durch ihn die Kapazitätsgrenze erreicht hat und er aus dem Residualgraphen entfernt wird.

Die Entfernungsmarkierung von u wird dann erhöht, wenn u keinen erlaubten Bogen verläßt, weil jeder Bogen $(u,v) \in A$ entweder keine Kapazität mehr hat oder die Entfernungsmarken der Nachfolger von u zu hoch sind. Das Ziel der RELABEL-Operation ist, mindestens einen weiteren erlaubten Bogen zu erzeugen.

Der generische Präfluß-Algorithmus muß keineswegs mit der Entfernungsmarkierung $d(v) = 0$ für alle $v \in V \setminus \{s\}$ beginnen. In der Praxis hat es sich als außerordentlich nützlich erwiesen, die Werte $d(v)$ so gut wie möglich mit $\text{dist}_{D_x}(v,t)$ in Übereinstimmung zu bringen. Dies ist zwar rechnerisch teuer, lohnt sich jedoch. Man setzt zu Beginn einfach $d(v) = \text{dist}(v, t)$ für alle $v \in V \setminus \{s\}$ (durch breadth first search) fest. Dann schreibt man $d(v)$ durch die RELABEL-Operation fort. Man sollte aber $d(v)$ periodisch neu berechnen, um die Werte wieder in Einklang mit den Distanzen im gegenwärtigen Residualgraphen zu bringen.

Die Initialisierung erledigt verschiedene wichtige Aufgaben. Zunächst wird jeder Nachfolger von s mit einem positiven Überschuß versehen, so daß der Algorithmus mit einem aktiven Knoten beginnen kann. Ferner ist nach der Initialisierung keiner der Bögen aus $\delta^+(s)$ erlaubt, da alle PUSH-Operationen saturierend waren. Da alle (v, t)-Wege in D_x höchstens die Länge $n - 1$ haben, wird durch $d(s) = n$ sichergestellt, daß $d(u) \leq d(s) + 1$ für alle $(u, s) \in A_x$ gilt. Man beachte, daß D_x keinen gerichteten (s, t)-Weg enthält und damit $d(s) = n$ auch eine untere Schranke für $\text{dist}_{D_x}(s, t)$ ist. Da die Entfernungsmarkierungen im Verlaufe des Verfahrens nicht fallen, bleibt garantiert, daß D_x während der Ausführung des Präfluß-Algorithmus nie einen gerichteten (s, t)-Weg enthalten wird. Es wird also nie mehr nötig sein, aus s Fluß herauszuschieben.

Es ist für das Verständnis des Vorgehens sehr hilfreich, sich den Präfluß-Algorithmus anhand eines Rohrleitungssystems vorzustellen. Die Bögen des gegebenen Graphen repräsentieren Rohre, die flexible Verbindungsstücke haben und bewegt werden können. Die Knoten stellen die Rohrverbindungen dar. Eine Entfernungsmarkierung mißt, wie weit ein Knoten vom Boden entfernt ist. In diesem Netzwerk wollen wir Wasser von der Quelle zur Senke schicken, wobei Wasser in den Rohren jeweils nur abwärts fließen kann in Richtung der Senke. Gelegentlich gerät der Fluß in eine "lokale Senke", nämlich dann, wenn ein Knoten keinen Nachbarn hat, der tiefer liegt. In diesem Falle heben wir (durch ein RELABEL) den Knoten (auf eine Ebene höher als sein niedrigster Nachbar) an, und das Wasser kann wieder abfließen. Da wir die Knoten immer weiter anheben, fließt der verbleibende Überschuß (der die Senke nicht mehr erreichen kann) an den inneren Knoten des Netzwerkes irgendwann zurück zur Quelle. Der Algorithmus endet, wenn die maximale Wassermenge von der Quelle zur Senke fließt und kein Knoten aus $V \setminus \{s, t\}$ mehr Überschuß hat, also ein "richtiger" Fluß fließt.

(7.32) Beispiel. Wir betrachten das in Abb. 7.10 (a) angegebene Netzwerk mit den bei den Bögen aufgelisteten Kapazitäten. Dieser Digraph ist symmetrisch, aber die jeweiligen "Gegenpfeile" sind aus Übersichtlichkeitsgründen nicht gezeichnet. Diese haben alle die Kapazität 0.

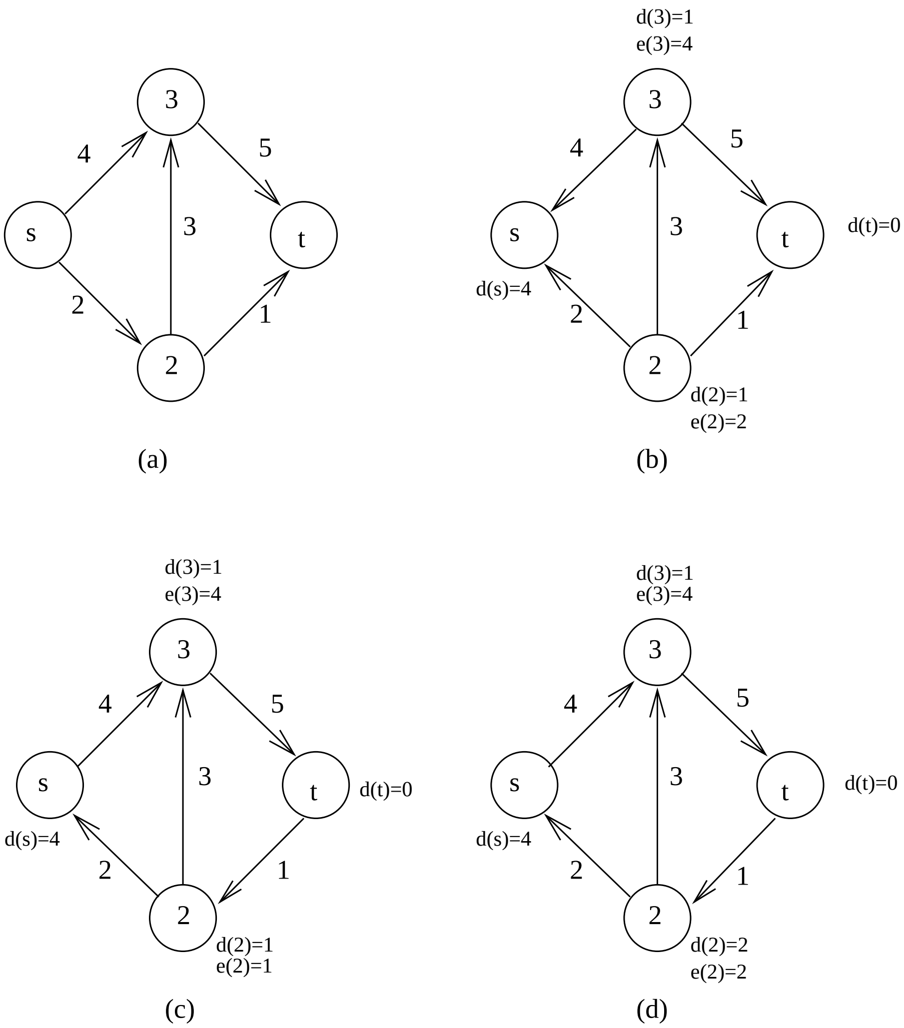

Abb. 7.10

Wir führen mit dem Netzwerk aus Abb. 7.10 (a) den Initialisierungsschritt aus. Daraus ergibt sich der in Abb. 7.10 (b) gezeichnete Residualgraph mir den angedeuteten Überschüssen $e(v)$ und den durch Kürzeste-Wege-Berechnung bestimmten Entfernungsmarkierungen $d(v)$. Die Knoten 2 und 3 sind aktiv. Wir wählen Knoten 2 aus. Da der Bogen $(2, t)$ eine Restkapazität von 1 hat und da $d(2) = d(t) + 1$ ist, ist der Bogen $(2, t)$ erlaubt. Da der Überschuß von Knoten 2 den Wert 2 hat, können wir einen Fluß vom Wert $\delta = \min\{r(2, t), e(2)\} = \min\{1, 2\} = 1$ durch den Bogen $(2, t)$ schieben. Diese Flußerhöhung reduziert den Überschuß von Knoten 2 auf 1, der Bogen (2,4) wird, da der Fluß durch ihn die Kapazitätsgrenze erreicht hat (der Schub also saturierend war) aus dem Residualgraphen entfernt, der Gegenbogen (t,2) wird mit Kapazität 1 hinzugefügt. Der neue Residualgraph ist in Abb. 7.10 (c) gezeigt. Der Knoten 2 ist immer noch aktiv. Wir können ihn also auswählen. Die Bögen $(2, s)$ und (2,3) haben positive Restkapazität, aber die Entfernungsbedingung für die Bögen (u, i) (d.h., $d(u) = d(i) + 1$) ist nicht erfüllt. Die Bögen sind also nicht erlaubt. Wir müssen daher ein RELABEL machen; damit erhält der

Knoten 2 die neue Entfernungsmarkierung $d(2) := \min\{d(3), d(s)\}+1 = \min\{1, 4\}+1 = 2$, siehe Abb. 7.10 (d). $\square$

Der Korrektheitsbeweis für den generischen Präfluß-Algorithmus basiert auf den folgenden Überlegungen. (Aus Zeitgründen geben wir die Beweise nicht vollständig an und verweisen auf die Übersichtsartikel Ahuja, Magnanti und Orlin (1989) und Goldberg, Tardos und Tarjan (1990) und die dort zitierte Originalliteratur.)

Wir gehen im weiteren davon aus, daß ein einfacher symmetrischer Digraph $D = (V, A)$ mit Bogenkapazitäten $c(a) \geq 0$ und zwei Knoten $s, t \in V, s \neq t$ gegeben sind und daß $n = |V|$ und $m = |A|$ gilt.

(7.33) Lemma. *In jedem beliebigen Stadium der Ausführung des generischen Präfluß-Algorithmus mit zugehörigem Präfluß x gibt es von jedem Knoten u mit positivem Überschuß $e_x(u) > 0$ einen gerichteten Weg von u zur Quelle s im Residualgraphen D_x.*

$\square$

Dieses Lemma impliziert, daß bei jeder RELABEL-Operation das Minimum nicht über der leeren Menge gebildet wird. Die nächsten Aussagen sind für die Laufzeit-Analyse von Bedeutung.

(7.34) Lemma. *In jedem Stadium der Ausführung des generischen Präfluß-Algorithmus gilt $d(v) \leq 2n - 1$ für jeden Knoten $v \in V$.*

$\square$

(7.35) Lemma.

(a) *Die Entfernungsmarkierung eines Knoten wird höchstens $(2n - 1)$ mal erhöht.*

(b) *Insgesamt werden höchstens $2n^2 - 1$ RELABEL-Operationen ausgeführt.*

(c) *Die Anzahl der saturierenden PUSH-Operationen ist höchstens nm.*

(d) *Die Anzahl der nichtsaturierenden PUSH-Operationen ist höchstens $2n^2m$.*

(7.36) Satz. *Der generische Präfluß-Algorithmus endet mit einem maximalen Fluß. Er benötigt $\mathcal{O}(n^2m)$ Fortschreibungsoperationen. Mit geeigneten Datenstrukturen ist das in $\mathcal{O}(n^2m)$ arithmetischen Operationen durchführbar.*

Es gibt eine Vielzahl von Arbeiten zur Laufzeitverbesserung des generischen Präfluß-Algorithmus. Man kann solche z.B. durch eine geschickte Auswahl der aktiven Knoten und die Benutzung geeigneter Datenstrukturen erreichen. Wird z.B. in der Schleife immer der Knoten mit dem höchsten Überschuß gewählt, so reduziert sich die Laufzeit auf $\mathcal{O}(n^3)$. Eine trickreiche Analyse dieses Verfahrens zeigt sogar, daß die Laufzeit $\mathcal{O}(n^2\sqrt{m})$ beträgt. Mit Skalierungsmethoden kann man eine Laufzeitreduktion auf $\mathcal{O}(n^2 \log U)$ bzw. $\mathcal{O}(nm + n^2\sqrt{\log U})$ erreichen, wobei $U := \max\{c(u, v) \mid (u, v) \in A\}$.

Ein detaillierteres Eingehen auf diese "Tricks" würde den Rahmen der Vorlesung sprengen. Wir verweisen hierzu auf die beiden mehrfach erwähnten Übersichtsartikel, das Buch von Ahuja, Magnanti und Orlin und die dort zitierte Originalliteratur.

Es sei noch auf einen fundamentalen Unterschied zwischen den Präfluß-Algorithmen und den anderen bisher besprochenen Maximalfluß-Algorithmen hingewiesen. Der Ford-Fulkerson und der DMKM-Algorithmus finden zunächst einen maximalen (s,t)-Fluß, dessen Maximalität sie durch Bestimmung eines (s,t)-Schnittes mit gleicher Kapazität nachweisen. Der Präfluß-Algorithmus findet "unterwegs" einen kapazitätsminimalen (s,t)-Schnitt (Überlegen Sie sich, wann das der Fall ist.) und muß anschließend durch Rückflußoperationen aus dem Präfluß einen Fluß machen. Ist man nur an der Bestimmung eines minimalen (s,t)-Schnittes interessiert, so kann man sich das "Rückschieben" sparen.

7.5 Einige Anwendungen

In diesem Abschnitt geht es nicht um praktische Anwendungen sondern Anwendungen der im Vorhergehenden angegebenen Sätze und Algorithmen zur Lösung anderer mathematischer (Optimierungs-)Probleme.

Matchings maximaler Kardinalität in bipartiten Graphen

In (2.9) haben wir das bipartite Matchingproblem kennengelernt. Wir wollen nun zeigen, wie man die Kardinalitätsversion dieses Problems, d. h. alle Kantengewichte sind 1, mit Hilfe eines Maximalflußverfahrens lösen kann.

Ist also $G = (V, E)$ ein bipartiter Graph mit Bipartition V_1, V_2, so definieren wir einen Digraphen $D = (W, A)$ wie folgt. Wir wählen zwei neue Knoten, sagen wir s und t, und setzen $W := V \cup \{s, t\}$. Die Bögen von D seien die folgenden. Ist $e = uv \in E$ eine Kante von G, so geben wir dieser die Richtung von V_1 nach V_2. Ist also $u \in V_1$ und $v \in V_2$, so wird aus $uv \in E$ der Bogen (u, v) andernfalls der Bogen (v, u). Ferner enthält D die Bögen (s, u) für alle $u \in V_1$ und die Bögen (v, t) für alle $v \in V_2$. Alle Bögen von D erhalten die Kapazität 1. Die Konstruktion von D aus G ist in Abbildung 7.11 an einem Beispiel dargestellt.

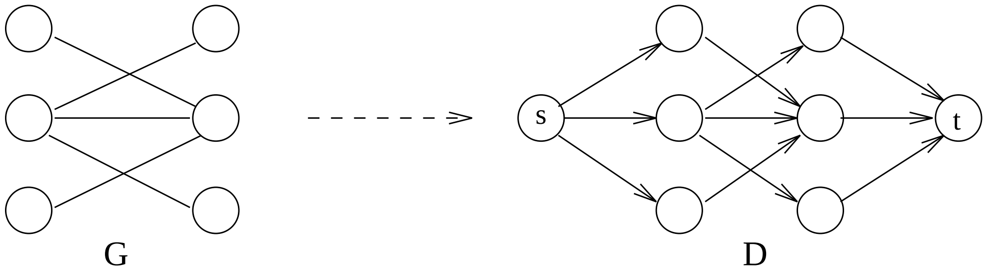

Abb. 7.11

(7.25) Satz. *Ist G ein bipartiter Graph und D der wie oben angegeben aus G konstruierte Digraph, dann ist der Wert eines maximalen (s,t)-Flusses x in D gleich dem Wert eines maximalen Matchings in D. Ferner kann ein maximales Matching M direkt aus x konstruiert werden.*

Beweis. Hausaufgabe. □

Zusammenhangsprobleme in Graphen und Digraphen

Mit Hilfe von Maximalflußalgorithmen können ferner für einen Digraphen die starke Zusammenhangszahl und die starke Bogenzusammenhangszahl in polynomialer Zeit bestimmt werden. Analog können in einem ungerichteten Graphen die Zusammenhangszahl und die Kantenzusammenhangszahl in polynomialer Zeit ermittelt werden.

Mehrere Quellen und Senken

Die Festlegung, daß wir in einem Digraphen einen Fluß von nur einer Quelle zu nur einer Senke schicken wollen, scheint auf den ersten Blick eine starke Einschränkung zu sein. Jedoch können Maximalflußprobleme mit mehreren Quellen und Senken sehr einfach auf das von uns behandelte Problem zurückgeführt werden.

Gegeben sei ein Digraph $D = (V, A)$ mit Bogenkapazitäten $c(a) \geq 0$ für alle $a \in A$. Ferner seien $S = \{s_1, \ldots, s_p\} \subseteq V$ Quellen und $T = \{t_1, \ldots, t_q\} \subseteq V$ Senken. Es gelte $S \cap T = \emptyset$. Ein zulässiger (S, T)**-Fluß** in D ist ein Vektor $x \in \mathbb{K}^A$ mit folgenden Eigenschaften

$$\begin{aligned} 0 \leq x_a \leq c_a \quad & \text{für alle } a \in A \\ x(\delta^-(v)) = x(\delta^+(v)) \quad & \text{für alle } v \in V \setminus (S \cup T). \end{aligned}$$

Der Wert eines zulässigen (S, T)-Flusses x ist definiert als

$$\operatorname{val}(x) := \sum_{s \in S} (x(\delta^+(s)) - x(\delta^-(s))).$$

Die Bestimmung eines maximalen (S, T)-Flusses in D kann wie folgt auf ein Maximalflußproblem in einem Digraphen $D' = (V', A')$ mit einer Quelle und einer Senke zurückgeführt werden. Wir wählen zwei neue Knoten s, t und setzen

$$V' := V \cup \{s, t\}.$$

Der Knoten s ist die Quelle, t ist die Senke von D'. Ferner sei

$$\begin{aligned} A' &:= A \cup \{(s, s_i) \mid i = 1, \ldots, p\} \cup \{(t_i, t) \mid i = 1, \ldots, q\} \\ c'(a) &:= c(a) \ \text{ für alle } a \in A \\ c(a) &:= M \ \text{ für alle } a \in A' \setminus A. \end{aligned}$$

Es reicht z. B. $M := \sum_{a \in A} c(a) + 1$ zu wählen. Man überlegt sich sofort, daß jedem zulässigen (s, t)-Fluß in D' ein zulässiger (S, T)-Fluß in D mit gleichem Wert entspricht. Folglich liefert ein maximaler (s, t)-Fluß in D' einen maximalen (S, T)-Fluß in D.

Literaturverzeichnis von Kapitel 7

R. K. Ahuja, T. L. Magnanti und J. B. Orlin (1989), "Network Flows", Kapitel 4 des Buches G. L. Nemhauser et al. (eds.), *Handbooks in OR & MS*, Vol. 1, North-Holland, Amsterdam.

M. O. Ball, T. L. Magnanti, C. L. Monma und G. L. Nemhauser (Hrsg.) (1995a), *Handbooks in Operations Research and Management Science; Vol. 8: Network routing,* North-Holland, Amsterdam.

M. O. Ball, T. L. Magnanti, C. L. Monma und G. L. Nemhauser (Hrsg.) (1995b), *Handbooks in Operations Research and Management Science; Vol. 7: Network models,* North-Holland, Amsterdam.

E. A. Dinic (1970), "Algorithm for solution of a problem of maximal flow in a network with power estimation", Soviet Math. Dokl. 11, 1277–1280.

J. Edmonds und R. M. Karp (1972), "Theoretical improvement in algorithmic efficiency of network flow problems", J. ACM 19, 248–264.

P. Elias, A. Feinstein und C. E. Shannon (1956), "Note on maximum flow through a network", IRE Trans. on Inform. Theroy 2 , 117–119.

L. R. Ford, Jr. und D. R. Fulkerson (1962), *Flows in Networks,* Princeton University Press, Princeton.

L. R. Ford, Jr. und D. R. Fulkerson (1956), "Maximal flow through a network", Canad. J. Math. 8, 399–404.

A. Frank (1995), "Connectivity and Network Flows", R. L. Graham et al. (Hrsg.) *Handbook of Combinatorics*, Kap. 2, 111–177, North-Holland, Amsterdam.

V. Goldberg (1987), "Efficient graph algorithms for sequential and parallel computers", Ph. D. Thesis, M.I.T., Cambridge; (erhältlich als Technical Report TR-374, Laboratory for Computer Science, M.I.T.).

V. Goldberg, É. Tardos und R. E. Tarjan (1990), "Network Flow Algorithms", in B. Korte et al. (eds.), *Paths, Flows, and VLSI-Layout*, Springer, Berlin.

V. Goldberg und R. E. Tarjan (1988), "A new approach to the maximum flow problem", J. Assoc. Comput. Mach. 35, 921-940.

V. M. Malhorta, M. Pramodh Kumar und N. Maheshwari (1978), "An $0(|V|^3)$ algorithm for finding the maximum flows in networks", Inform. Process. Lett. 7, 227–278..

K. Mehlhorn (1984), *Data Structures and Algorithms, Vol. 1-3,* EATCS Monographie, Springer-Verlag, 1984, (dreibändige Monographie, Band I liegt auch auf deutsch im Teubner-Verlag (1986) vor).

M. M. Syslo, N. Deo und J. S. Kowalik (1983), *Discrete optimization algorithms,* Prentice Hall, Englewood Cliffs, N.J..

Kapitel 8

Weitere Netzwerkflußprobleme

Das im vorhergehenden Kapitel behandelte Maximalflußproblem ist eines der Basisprobleme der Netzwerkflußtheorie. Es gibt noch weitere wichtige und anwendungsreiche Netzwerkflußprobleme. Wir können hier jedoch aus Zeitgründen nur wenige dieser Probleme darstellen und analysieren. Der Leser, der an einer vertieften Kenntnis der Netzwerkflußtheorie interessiert ist, sei auf die am Anfang von Kapitel 7 erwähnten Bücher und Übersichtsartikel verwiesen.

8.1 Flüsse mit minimalen Kosten

Häufig tritt das Problem auf, durch ein Netzwerk nicht einen maximalen Fluß senden zu wollen, sondern einen Fluß mit vorgegebenem Wert, der bezüglich eines Kostenkriteriums minimale Kosten verursacht. Wir wollen hier nur den Fall einer linearen Kostenfunktion behandeln, obwohl gerade auch konkave und stückweise lineare Kosten (bei Mengenrabatten) eine wichtige Rolle spielen.

Sind ein Digraph $D = (V, A)$ mit Bogenkapazitäten $c(a) \geq 0$ für alle $a \in A$ und Kostenkoeffizienten $w(a)$ für alle $a \in A$ gegeben, sind $s, t \in V$ zwei verschiedene Knoten und ist f ein vorgegebener Flußwert, dann nennt man die Aufgabe, einen (s, t)-Fluß x mit Wert f zu finden, dessen Kosten $\sum_{a \in A} w_a x_a$ minimal sind, ein **Minimalkosten-Netzwerkflußproblem**. Analog zur LP-Formulierung (7.5) des Maximalflußproblems kann man ein Minimalkosten-Flußproblem als lineares Programm darstellen. Offenbar ist jede Optimallösung des linearen Programms (8.1) ein kostenminimaler c-kapazitierter (s, t)-Fluß mit Wert f.

$$
\begin{aligned}
& \min \sum_{a \in A} w_a x_a \\
(8.1) \qquad & x(\delta^-(v)) - x(\delta^+(v)) = 0 \quad \forall\, v \in V \setminus \{s, t\} \\
& x(\delta^-(t)) - x(\delta^+(t)) = f \\
& 0 \leq x_a \leq c_a \quad \forall\, a \in A
\end{aligned}
$$

Minimalkosten-Flußprobleme kann man daher mit Algorithmen der linearen Optimierung lösen. In der Tat gibt es besonders schnelle Spezialversionen des Simplexalgorithmus für Probleme des Typs (8.1). Sie werden Netzwerk-Simplexalgorithmen genannt und nutzen die Tatsache aus, daß die Basen von (8.1) durch aufspannende Bäume im Digraphen D

repräsentiert werden können. Alle numerischen Unterprogramme wie Basis-Updates, Berechnung von reduzierten Kosten etc. können sehr effizient durch einfache kombinatorische Algorithmen erledigt werden.

Ein von A. Löbel (Konrad-Zuse-Zentrum, Berlin) implementierter Code dieser Art ist auf dem ZIB-Server verfügbar (URL: `http://www.zib.de/Optimization/Software/Mcf/`. Mit diesem Code namens MCF können Minimalkosten-Flußprobleme mit mehreren Millionen Variablen in wenigen Minuten gelöst werden.

Es gibt viele kombinatorische Spezialverfahren zur Lösung von Minimalkosten-Flußproblemen. Alle "Tricks" der kombinatorischen Optimierung und Datenstrukturtechniken der Informatik werden benutzt, um schnelle Lösungsverfahren für (8.1) zu produzieren. Ein Ende ist nicht abzusehen. Es gibt (zur Zeit) kein global bestes Verfahren weder bezüglich der beweisbaren Laufzeit noch in bezug auf Effizienz im praktischen Einsatz. Die Literatur ist allerdings voll mit Tabellen mit derzeitigen "Weltrekorden" bezüglich der worst-case-Laufzeit unter speziellen Annahmen an die Daten. Alle derzeit gängigen Verfahren können — gut implementiert — Probleme des Typs (8.1) mit zigtausenden von Knoten und Hunderttausenden oder gar Millionen von Bögen mühelos lösen.

Wir haben in dieser Vorlesung nicht genügend Zeit, um auf diese Details und Feinheiten einzugehen. Wir werden lediglich ein kombinatorisches Verfahren und die zugrundeliegende Theorie vorstellen. Um den Algorithmus und den Satz, auf dem seine Korrektheit beruht, darstellen zu können, führen wir einige neue Begriffe ein.

Sei x ein zulässiger (s,t)-Fluß in D und sei C ein (nicht notwendigerweise gerichteter) Kreis in D. Diesem Kreis C können wir offenbar zwei Orientierungen geben. Ist eine Orientierung von C gewählt, so nennen wir einen Bogen auf C, der in Richtung der Orientierung verläuft, **Vorwärtsbogen**, andernfalls nennen wir ihn **Rückwärtsbogen**. Ein Kreis C heißt **augmentierend** bezüglich x, wenn es eine Orientierung von C gibt, so daß $x_a < c_a$ für alle Vorwärtsbögen $a \in C$ und daß $0 < x_a$ für alle Rückwärtsbögen $a \in C$ gilt (vergleiche Definition (7.8)). Ein Kreis kann offenbar bezüglich beider, einer oder keiner Richtung augmentierend sein. Sprechen wir von einem augmentierenden Kreis C, so unterstellen wir fortan, daß eine Orientierung von C fest gewählt ist, bezüglich der C augmentierend ist.

Die Summe der Kostenkoeffizienten der Vorwärtsbögen minus der Summe der Kostenkoeffizienten der Rückwärtsbögen definieren wir als die **Kosten eines augmentierenden Kreises**. (Wenn ein Kreis in Bezug auf beide Orientierungen augmentierend ist, können die beiden Kosten verschieden sein!) Das zentrale Resultat dieses Abschnitts ist das folgende.

(8.2) Satz. *Ein zulässiger (s,t)-Fluß x in D mit Wert f hat genau dann minimale Kosten, wenn es bezüglich x keinen augmentierenden Kreis mit negativen Kosten gibt.*

Beweis. Wir beweisen zunächst nur die triviale Richtung. Gibt es einen augmentierenden Kreis C bezüglich x, so setzen wir:

$$(8.3) \qquad \varepsilon := \min \begin{cases} c_{ij} - x_{ij} & (i,j) \in C \text{ Vorwärtsbogen,} \\ x_{ij} & (i,j) \in C \text{ Rückwärtsbogen.} \end{cases}$$

Definieren wir

$$
(8.4) \qquad x'_{ij} := \begin{cases} x_{ij} + \varepsilon & \text{falls } (i,j) \in C \text{ Vorwärtsbogen,} \\ x_{ij} - \varepsilon & \text{falls } (i,j) \in C \text{ Rückwärtsbogen,} \\ x_{ij} & \text{falls } (i,j) \in A \setminus C, \end{cases}
$$

dann ist $x' \in \mathbb{R}^A$ trivialerweise ein zulässiger (s,t)-Fluß mit Wert $\text{val}(x') = f$. Hat der augmentierende Kreis C negative Kosten $\gamma < 0$, dann gilt offenbar $\sum_{(i,j)\in A} w_{ij}x'_{ij} = \sum_{(i,j)\in A} w_{ij}x_{ij} + \varepsilon\gamma$. Gibt es also einen augmentierenden Kreis bezüglich x mit negativen Kosten, dann kann x nicht kostenminimal sein. □

Um die umgekehrte Richtung zu beweisen, müssen wir etwas mehr Aufwand treiben, den wir allerdings direkt bei der Darstellung des Algorithmus benutzen können. Ist x ein zulässiger (s,t)-Fluß mit Wert f, dann definieren wir einen Digraphen (genannt **augmentierendes Netzwerk bezüglich** x) $N = (V, \overline{A}, \overline{c}, \overline{w})$ wie folgt: Es sei

$$
A_1 := \{a \in A \mid x_a < c_a\}, \quad A_2 := \{(v,u) \mid (u,v) \in A \text{ und } x_{uv} > 0\}.
$$

Ist $a \in A$, so schreiben wir a_1 bzw. a_2 um den zugehörigen Bogen aus A_1 bzw. A_2 zu bezeichnen. Schreiben wir für $a \in A$ eine Formel wie etwa $x'(a) = x(a) + \overline{x}(a_1) - \overline{x}(a_2)$ und ist einer der Bögen a_1 bzw. a_2 nicht definiert (d. h., es gilt entweder $x_a = c_a$ oder $x_a = 0$), dann ist der Wert $\overline{x}(a_1)$ bzw. $\overline{x}(a_2)$ als Null zu betrachten. Wir setzen $\overline{A} := A_1 \dot{\cup} A_2$ (man beachte, daß $\overline{A}$ parallele Bögen enthalten kann). Ferner sei für $\overline{a} \in \overline{A}$

$$
\overline{c}(\overline{a}) := \begin{cases} c(a) - x(a) & \text{falls } \overline{a} = a_1, \\ x(a) & \text{falls } \overline{a} = a_2, \end{cases}
$$

$$
\overline{w}(\overline{a}) := \begin{cases} w(a) & \text{falls } \overline{a} = a_1, \\ -w(a) & \text{falls } \overline{a} = a_2. \end{cases}
$$

In Abbildung 8.1 (a) ist ein Digraph mit einem (s,t)-Fluß x des Wertes 4 dargestellt. Die drei Zahlen bei einem Bogen a geben an: Fluß durch a / Kapazität von a / Kosten von a. Das augmentierende Netzwerk N bezüglich D und x ist in 8.2 (b) gezeichnet. Die beiden Zahlen bei einem Bogen a in 8.1 (b) geben an: Kapazität von a / Kosten von a.

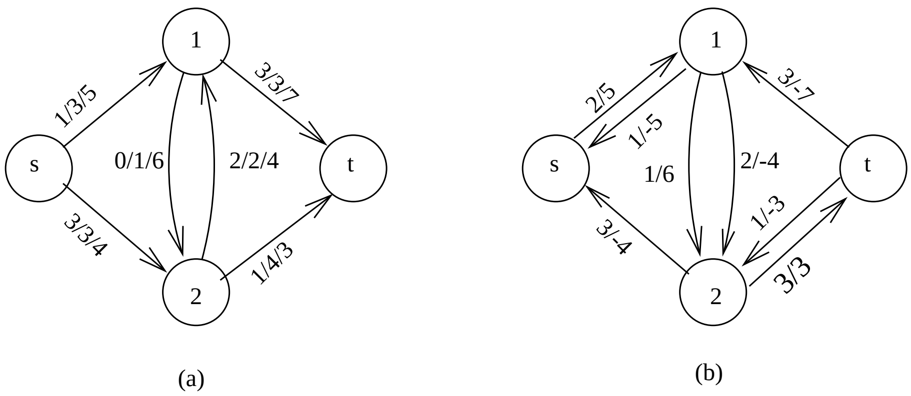

Abb. 8.1

Der Kreis $\{(2,t),(1,t),(2,1)\}$ in Abbildung 8.1 (a) ist ein augmentierender Kreis mit Kosten $3-7-4=-8$. Dieser Kreis entspricht in (b) dem gerichteten Kreis $\{(2,t),(t,1),(1,2)\}$, wobei von den beiden zwischen 1 und 2 parallel verlaufenden Bögen natürlich der mit negativen Kosten zu wählen ist. Aufgrund der Konstruktion von N ist folgende Beziehung offensichtlich:

(8.5) Lemma. *Ist $D=(V,A)$ ein Digraph mit Kapazitäten $c \in \mathbb{K}_+^A$ und Kosten $w \in \mathbb{K}^A$, ist x ein zulässiger (s,t)-Fluß in D und ist $N=(V,\overline{A},\overline{c},\overline{w})$ das zu D und x gehörige augmentierende Netzwerk, dann entspricht jeder augmentierende Kreis in D genau einem gerichteten Kreis in N. Die Kosten eines augmentierenden Kreises in D stimmen überein mit der Summe der Kostenkoeffizienten des zugehörigen gerichteten Kreises in N.*

□

Damit ist unser Exkurs zur Definition von augmentierenden Netzwerken beendet. Wir formulieren nun Theorem (8.2) unter Benutzung dieses neuen Konzepts um.

(8.6) Lemma. *Sei x ein zulässiger (s,t)-Fluß in D mit Wert f, und $N=(V,\overline{A},\overline{c},\overline{w})$ sei das bezüglich x und D augmentierende Netzwerk, dann gilt folgendes. Der Fluß x ist unter allen zulässigen (s,t)-Flüssen in D mit Wert f genau dann kostenminimal, wenn es in N keinen gerichteten Kreis mit negativen Kosten gibt.*

Beweis. Gibt es in N einen gerichteten Kreis mit negativen Kosten, so folgt analog zum Beweis des einfachen Teils von (8.2), daß x nicht minimal ist.

Nehmen wir umgekehrt an, daß x nicht kostenminimal ist, dann müssen wir in N einen gerichteten Kreis mit negativen Kosten finden. Sei also x' ein zulässiger (s,t)-Fluß in D mit Wert f und $w^T x' < w^T x$. Für jeden Bogen $\overline{a} \in \overline{A}$ setzen wir

$$\overline{x}(\overline{a}) := \begin{cases} \max\{0, x'(a) - x(a)\}, & \text{falls } \overline{a} = a_1, \\ \max\{0, x(a) - x'(a)\}, & \text{falls } \overline{a} = a_2, \end{cases}$$

d. h., es gilt $\overline{x}(a_1) - \overline{x}(a_2) = x'(a) - x(a)$. Wir weisen zunächst einige Eigenschaften von $\overline{x} \in \mathbb{K}^{\overline{A}}$ nach.

Behauptung 1. $\overline{x}$ ist ein zulässiger (s,t)-Fluß in N mit Wert 0 und $\overline{w}^T \overline{x} < 0$.

Beweis. Wir zeigen zunächst, daß $\overline{x}$ negative Kosten hat. Für alle $a \in A$ gilt offenbar

$$\overline{x}(a_1)\overline{w}(a_1) + \overline{x}(a_2)\overline{w}(a_2) = (x'(a) - x(a))w(a),$$

und daraus folgt:

$$\begin{aligned}
\sum_{\overline{a} \in \overline{A}} \overline{w}(\overline{a})\overline{x}(\overline{a}) &= \sum_{a \in A} (\overline{w}(a_1)\overline{x}(a_1) + \overline{w}(a_2)\overline{x}(a_2)) \\
&= \sum_{a \in A} (x'(a) - x(a))w(a) \\
&= w^T x' - w^T x < 0.
\end{aligned}$$

Der Vektor $\overline{x}$ erfüllt trivialerweise die Kapazitätsbedingungen $0 \leq \overline{x}(\overline{a}) \leq \overline{c}(\overline{a})$. Wir zeigen nun, daß $\overline{x}$ auch die Flußerhaltungsbedingungen für alle $v \in V$ erfüllt.

$$
\begin{aligned}
\overline{x}(\delta_N^+(v)) - \overline{x}(\delta_N^-(v)) &= \sum_{a \in \delta^+(v)} (\overline{x}(a_1) - \overline{x}(a_2)) - \sum_{a \in \delta^-(v)} (\overline{x}(a_1) - \overline{x}(a_2)) \\
&= \sum_{a \in \delta^+(v)} (x'(a) - x(a)) - \sum_{a \in \delta^-(v)} (x'(a) - x(a)) \\
&= \left(\sum_{a \in \delta^+(v)} x'(a) - \sum_{a \in \delta^-(v)} x'(a) \right) - \\
&\quad \left(\sum_{a \in \delta^+(v)} x(a) - \sum_{a \in \delta^-(v)} x(a) \right) \\
&= \begin{cases} 0 - 0 & \text{falls } v \in V \setminus \{s,t\} \\ \operatorname{val}(x') - \operatorname{val}(x) & \text{falls } v = s \\ -\operatorname{val}(x') + \operatorname{val}(x) & \text{falls } v = t \end{cases} \\
&= 0
\end{aligned}
$$

Daraus folgt, daß $\overline{x}$ die Flußerhaltungsbedingungen erfüllt und daß $\operatorname{val}(\overline{x}) = 0$ gilt. Damit ist Behauptung 1 bewiesen.

Behauptung 2. Es gibt zulässige (s,t)-Flüsse $x_1, \ldots, x_k \in R^{\overline{A}}$, $k \leq |\overline{A}|$, so daß folgendes gilt:

(a) $\overline{x}(\overline{a}) = \sum_{i=1}^{k} x_i(\overline{a})$ für alle $\overline{a} \in \overline{A}$.

(b) Für jedes $i \in \{1, \ldots, k\}$ gibt es einen gerichteten Kreis $\overline{C}_i$ in N und eine positive Zahl α_i, so daß $x_i(\overline{a}) = \alpha_i$ für alle $\overline{a} \in \overline{C}_i$ und $\overline{x}_i(\overline{a}) = 0$ für alle $\overline{a} \in \overline{A} \setminus \overline{C}_i$.

Beweis. Wir führen eine Induktion über die Anzahl p der Bögen $\overline{a} \in \overline{A}$ mit $\overline{x}(\overline{a}) \neq 0$ durch. Ist $p = 0$, so ist nichts zu zeigen. Sei also $p \geq 1$, und sei v_0 ein Knoten, so daß ein Bogen $(v_0, v_1) \in \overline{A}$ existiert mit $\overline{x}((v_0, v_1)) \neq 0$. Da in v_1 die Flußerhaltungsbedingung gilt, muß es einen Bogen $(v_1, v_2) \in \overline{A}$ geben mit $\overline{x}((v_1, v_2)) \neq 0$. Fahren wir so weiter fort, so erhalten wir einen gerichteten Weg $v_0, v_1, v_2, \ldots$. Da N endlich ist, muß irgendwann ein Knoten auftreten, der schon im bisher konstruierten Weg enthalten ist. Damit haben wir einen gerichteten Kreis $\overline{C}_p$ gefunden. Sei α_p der kleinste Wert $\overline{x}(\overline{a})$ der unter den Bögen $\overline{a}$ des Kreises $\overline{C}_p$ auftritt. Definieren wir

$$
\begin{aligned}
x_p(\overline{a}) &:= \alpha_p \quad \text{für alle } \overline{a} \in \overline{C}_p, \\
x_p(\overline{a}) &:= 0 \quad \text{sonst,}
\end{aligned}
$$

so ist $x_p \in \mathbb{K}^{\overline{A}}$ ein (s,t)-Fluß mit Wert 0. Setzen wir nun $\overline{x}_p := \overline{x} - x_p$, so ist $\overline{x}_p$ ein (s,t)-Fluß mit Wert 0, und die Zahl der Bögen $\overline{a} \in \overline{A}$ mit $\overline{x}_p(\overline{a}) \neq 0$ ist kleiner als p. Nach Induktionsvoraussetzung gilt die Behauptung für $\overline{x}_p$, und somit gilt sie nach Konstruktion auch für $\overline{x}$. Somit ist Behauptung 2 bewiesen.

Damit können wir den Beweis von (8.6) beenden. Für die nach Behauptung 2 existierenden (s,t)-Flüsse x_i gilt offenbar

$$\overline{w}^T\overline{x} = \sum_{i=1}^{k} \overline{w}^T x_i.$$

Da $\overline{w}^T\overline{x} < 0$ nach Behauptung 1 ist, muß einer der Werte $\overline{w}^T x_i$ kleiner als Null sein, daß heißt, wir haben in N einen gerichteten Kreis mit negativen Kosten gefunden. $\square$

Satz (8.2), bzw. Lemma (8.6) sind Optimalitätskriterien für zulässige (s,t)-Flüsse. Man kann beide Aussagen — was algorithmisch noch wichtiger ist — benutzen, um zu zeigen, daß Kostenminimalität erhalten bleibt, wenn man entlang Wegen minimaler Kosten augmentiert.

(8.7) Satz. *Sei $D = (V, A)$ ein Digraph mit gegebenen Knoten $s, t \in V$, Kapazitäten $c \in \mathbb{K}_+^A$ und Kosten $w \in \mathbb{K}^A$. Sei x ein zulässiger (s,t)-Fluß in D mit Wert f, der kostenminimal unter allen (s,t)-Flüssen mit Wert f ist, und sei $N = (V, \overline{A}, \overline{c}, \overline{w})$ das zugehörige augmentierende Netzwerk. Sei P ein (s,t)-Weg in N mit minimalen Kosten $\overline{w}(P)$, und sei $\overline{x}$ ein zulässiger (s,t)-Fluß in N, so daß $\overline{x}(\overline{a}) > 0$ $\forall\, \overline{a} \in P$ und $\overline{x}(\overline{a}) = 0$ $\forall\, \overline{a} \in \overline{A} \setminus P$, dann ist der Vektor $x' \in \mathbb{K}^A$ definiert durch*

$$x'(a) := x(a) + \overline{x}(a_1) - \overline{x}(a_2) \qquad \textit{für alle } a \in A$$

ein zulässiger (s,t)-Fluß in D mit Wert $f + \mathrm{val}(\overline{x})$, der kostenminimal unter allen Flüssen dieses Wertes in D ist.

Beweis. Trivialerweise ist $x' \in \mathbb{K}^A$ ein zulässiger (s,t)-Fluß mit Wert $f + \mathrm{val}(\overline{x})$. Wir zeigen, daß x' kostenminimal ist. Angenommen, dies ist nicht der Fall, dann gibt es nach Lemma (8.6) einen negativen gerichteten Kreis C' im bezüglich x' augmentierenden Netzwerk $N' = (V, A', c', w')$. Wir beweisen, daß dann auch ein negativer gerichteter Kreis in N bezüglich x existiert.

Wir bemerken zunächst, daß das augmentierende Netzwerk N' aus N dadurch hervorgeht, daß die Bögen aus $P \subseteq \overline{A}$ neue Kapazitäten erhalten und möglicherweise ihre Richtung und damit ihre Kosten ändern. Sei $\overline{B} \subseteq P$ die Menge der Bögen aus $\overline{A}$, die in N' eine andere Richtung als in N haben, und sei $B' := \{(v,u) \in A' \mid (u,v) \in \overline{B}\}$.

Wir untersuchen nun den gerichteten Kreis $C' \subseteq A'$ in N' mit negativen Kosten. Gilt $C' \cap B' = \emptyset$, so ist C' in $\overline{A}$ enthalten und somit ein negativer Kreis in N. Dann wäre x nach (8.6) nicht kostenoptimal, was unserer Voraussetzung widerspricht. Wir können daher annehmen, daß $C' \cap B' \neq \emptyset$ gilt.

Der Beweis verläuft nun wie folgt. Wir konstruieren aus dem (s,t)-Weg P und dem gerichteten Kreis C' einen (s,t)-Weg $Q \subseteq \overline{A}$ und einen gerichteten Kreis $K' \subseteq A'$ mit den Eigenschaften

$$\overline{w}(P) + w'(C') \geq \overline{w}(Q) + w'(K')$$
$$|C' \cap B'| > |K' \cap B'|$$

Durch iterative Wiederholung dieser Konstruktion erhalten wir nach höchstens $|B'|$ Schritten einen (s,t)-Weg in N und einen gerichteten Kreis in N', dessen Kosten negativ sind und der keinen Bogen aus B' enthält. Folglich ist dieser Kreis ein negativer Kreis in N. Widerspruch!

Die Konstruktion verläuft wie folgt. Da $C' \cap B' \neq \emptyset$, gibt es einen Bogen $(u,v) \in P$ mit $(v,u) \in C'$. Wir wählen denjenigen Bogen $(u,v) \in P$ der auf dem Weg von s nach t entlang P der erste Bogen mit $(v,u) \in C'$ ist. Wir unterscheiden zwei Fälle.

Fall 1: $C' \cap B'$ enthält mindestens zwei Bögen. Sei (y,x) der nächste Bogen auf dem gerichteten Kreis C' nach (v,u), der in B' ist. Wir konstruieren einen (s,t)-Pfad $\overline{P} \subseteq \overline{A}$ wie folgt. Wir gehen von s aus entlang P nach u, dann von u entlang C' nach y und von y entlang P nach t. Starten wir nun in v, gehen entlang P zu x und dann entlang C' nach v, so erhalten wir eine geschlossene gerichtete Kette $P' \subseteq A'$. Aus $\overline{w}_{uv} = -w'_{vu}$ und $\overline{w}_{xy} = -w'_{yx}$ folgt, daß $\overline{w}(P) + w'(C') = \overline{w}(\overline{P}) + w'(P')$ gilt. $\overline{P}$ ist die Vereinigung von gerichteten Kreisen $C'_1, \dots, C'_k \subseteq \overline{A}$ mit einem gerichteten (s,t)-Weg $Q \subseteq \overline{A}$, und P' ist die Vereinigung von gerichteten Kreisen $C'_{k+1}, \dots, C'_r \subseteq A'$. Da P kostenminimal in N ist, gilt $\overline{w}(P) \leq \overline{w}(Q)$, und somit gibt es wegen $\overline{w}(\overline{P}) + w'(P') = \overline{w}(Q) + \sum_{i=1}^{k} \overline{w}(C'_i) + \sum_{i=k+1}^{r} w'(C'_i)$ mindestens einen gerichteten Kreis in A', sagen wir K', der negative Kosten hat. Nach Konstruktion gilt $|K' \cap B'| < |C' \cap B'|$.

Fall 2: Der Bogen (u,v) ist der einzige Bogen auf P mit $(v,u) \in C' \cap B'$. Wir konstruieren einen gerichteten (s,t)-Pfad $\overline{P} \subseteq \overline{A}$ wie folgt. Wir starten in s und folgen P bis u, dann folgen wir dem gerichteten Weg von u entlang C' bis v und dann wieder dem gerichteten Weg von v entlang P bis t. Offenbar ist $\overline{P}$ in $\overline{A}$ enthalten und ein gerichteter (s,t)-Pfad in N. Aus $\overline{w}_{uv} = -w'_{vu}$ folgt direkt $\overline{w}(P) + w'(C') = \overline{w}(\overline{P})$. Der gerichtete (s,t)-Pfad $\overline{P}$ ist die Vereinigung eines (s,t)-Weges Q und einiger gerichteter Kreise $C'_1, \dots, C'_k$. Da P ein (s,t)-Weg in N mit minimalen Kosten ist, gilt $\overline{w}(Q) \geq \overline{w}(P)$, und aus $\overline{w}(Q) = \overline{w}(\overline{P}) - \sum_{i=1}^{k} \overline{w}(C'_1)$ folgt, daß mindestens einer der Kreise C'_i negativ ist. Da alle C'_i in $\overline{A}$ enthalten sind, enthält $\overline{A}$ einen negativen Kreis. Widerspruch!

$\square$

Damit können wir nun einen Algorithmus zur Lösung des Minimalkosten-Flußproblems angeben.

(8.8) Algorithmus.

Input: *Digraph $D = (V, A)$, mit Kapazitäten $c \in \mathbb{K}_+^A$ und Kosten $w \in \mathbb{K}^A$, zwei verschiedene Knoten $s, t \in V$ und ein Flußwert f.*

Output: *Ein zulässiger (s,t)-Fluß x mit Wert f, der kostenminimal unter allen zulässigen (s,t)-Flüssen mit Wert f ist, oder die Aussage, daß kein zulässiger (s,t)-Fluß mit Wert f existiert.*

1. *Setze $x(a) = 0$ für alle $a \in A$ (bzw. starte mit einem zulässigen (s,t)-Fluß mit Wert nicht größer als f).*

2. *Konstruiere das augmentierende Netzwerk $N = (V, \overline{A}, \overline{c}, \overline{w})$ bezüglich D und x.*
3. *Wende einen Kürzeste-Wege-Algorithmus (z. B. den Floyd-Algorithmus (6.9)) an, um im Digraphen $N = (V, \overline{A})$ mit den "Bogenlängen" $\overline{w}(\overline{a})$, $\overline{a} \in \overline{A}$ einen negativen gerichteten Kreis C zu finden. Gibt es keinen, dann gehe zu 5.*
4. *(Augmentierung entlang C)*
   Bestimme $\varepsilon := \min\{\overline{c}(\overline{a}) \mid \overline{a} \in C\}$, setze für $a \in A$

$$x(a) := \begin{cases} x(a) + \varepsilon & \text{falls } a_1 \in C \\ x(a) - \varepsilon & \text{falls } a_2 \in C \\ x(a) & \text{andernfalls} \end{cases}$$

   und gehe zu 2. (Hier erhalten wir einen Fluß mit gleichem Wert und geringeren Kosten.)
5. *Ist $\mathrm{val}(x) = f$, STOP, gib x aus.*
6. *Bestimme mit einem Kürzeste-Wege-Algorithmus (z. B. einer der Varianten des Moore-Bellman-Verfahrens (6.6), es gibt keine negativen Kreise!) einen (s,t)-Weg P in N mit minimalen Kosten $\overline{w}(P)$.*
7. *Gibt es in N keinen (s,t)-Weg, dann gibt es in D keinen zulässigen (s,t)-Fluß mit Wert f, STOP.*
8. *(Augmentierung entlang P)*
   Bestimme $\varepsilon' := \min\{\overline{c}(\overline{a}) \mid \overline{a} \in P\}$, $\varepsilon := \min\{\varepsilon', f - \mathrm{val}(x)\}$, setze für $a \in A$

$$x(a) := \begin{cases} x(a) + \varepsilon & \text{falls } a_1 \in P \\ x(a) - \varepsilon & \text{falls } a_2 \in P \\ x(a) & \text{andernfalls,} \end{cases}$$

   konstruiere das bezüglich x und D augmentierende Netzwerk $N = (V, \overline{A}, \overline{c}, \overline{w})$ und gehe zu 5.

□

Die Korrektheit des Algorithmus folgt unmittelbar aus (8.6) und (8.7). Wir wollen nun die Laufzeit abschätzen. Hat D nur nichtnegative Kosten $w(a)$ bzw. enthält D keinen augmentierenden Kreis mit negativen Kosten, so ist der Nullfluß eine kostenoptimaler Fluß mit Wert Null, und die Schleife über die Schritte 2, 3 und 4 braucht nicht durchlaufen zu werden. Sind alle Kapazitäten ganzzahlig, so wird der Flußwert in Schritt 8 um jeweils mindestens eine Einheit erhöht. Also sind höchstens f Aufrufe einen Kürzesten-Wege-Algorithmus erforderlich.

(8.9) Satz. *Ist $D = (V, A)$ ein Digraph mit ganzzahligen Kapazitäten $c(a)$ und nichtnegativen Kosten $w(a)$, und sind s, t zwei verschiedene Knoten und $f \in \mathbf{Z}_+$ ein vorgegebener Flußwert, so findet Algorithmus (8.8) in $\mathcal{O}(f|V|^3)$ Schritten einen kostenminimalen zulässigen (s,t)-Fluß mit Wert f, falls ein solcher existiert.*

□

Der Algorithmus ist in dieser Form nicht polynomial, da seine Laufzeit polynomial in $\langle f \rangle$ sein müsste. Ferner ist nicht unmittelbar klar, wie lange er läuft, wenn negative

Kosten erlaubt sind, da die Anzahl der Kreise mit negativen Kosten, auf denen der Fluß verändert werden muß, nicht ohne weiteres abgeschätzt werden kann. Diese Schwierigkeiten können durch neue Ideen (Augmentierung entlang Kreisen mit minimalen durchschnittlichen Kosten $\overline{w}(C)/|C|$, Skalierungstechniken) überwunden werden, so daß Versionen von Algorithmus (8.8) existieren, die polynomiale Laufzeit haben. Aus Zeitgründen können diese Techniken hier nicht dargestellt werden. Es sei hierzu wiederum auf die schon mehrfach erwähnten Übersichtsartikel und das Buch von Ahuja, Magnanti und Orlin verwiesen, die auch ausführlich auf die historische Entwicklung eingehen.

8.2 Netzwerke mit Flußmultiplikatoren

Gelegentlich werden Flüsse in Netzwerken nicht konserviert, es können Verluste (Sickerverluste bei Wasserleitungen durch undichte Stellen) oder Umwandlungen (Geldtausch, chemische Prozesse) auftreten. In solchen Fällen kann man diese Verluste oder Transformationen durch sogenannte “Multiplikatoren” berücksichtigen bzw. modellieren. Dies geschieht dadurch, daß man jedem Bogen $a \in A$ nicht nur eine Kapazität $c(a)$ und einen Kostenkoeffizienten $w(a)$ sondern noch eine Zahl $m(a)$, den **Flußmultiplikator**, zuordnet. Bei den Zwischenknoten bekommt dann die Flußerhaltungsbedingung die folgende Form:

$$\sum_{a \in \delta^+(v)} x(a) - \sum_{a \in \delta^-(v)} m(a)x(a) = 0 \quad \forall\, v \in V \setminus \{s,t\},$$

und für die Quelle s bzw. die Senke t gilt:

$$\begin{aligned} \sum_{a \in \delta^+(s)} x(a) - \sum_{a \in \delta^-(s)} m(a)x(a) &= f_s, \\ \sum_{a \in \delta^+(t)} x(a) - \sum_{a \in \delta^-(t)} m(a)x(a) &= -f_t. \end{aligned}$$

Beim klassischen Maximalflußproblem gilt $m(a) = 1 \; \forall\, a \in A$, woraus $f_s = f_t$ folgt, hier jedoch gilt i. a. $f_s \neq f_t$. Der Betrag $f_s - f_t$ wird üblicherweise **Verlust des Flusses** x genannt.

Es gibt nun verschiedene Optimierungsfragen. Z. B. kann man bei gegebenem Wert f_s den Wert f_t maximieren oder bei gegebenem Wert f_t den Wert f_s minimieren oder $f_t - f_s$ maximieren. Analog kann man natürlich auch bei gegebenen Kostenkoeffizienten $w(a)$ $\forall\, a \in A$ einen kostenminimalen Fluß bestimmen, wobei entweder f_s oder f_t vorgegeben sind. Ist letzteres der Fall, so erhalten wir ein LP der folgenden Form

$$\begin{aligned} \min & \sum_{a \in A} w(a)x(a) \\ & \sum_{a \in \delta^+(v)} x(a) - \sum_{a \in \delta^-(v)} m(a)x(a) = \begin{cases} 0 & \text{falls } v \in V \setminus \{s,t\}, \\ -f_t & \text{falls } v = t, \end{cases} \\ & 0 \le x_a \le c(a) \quad \text{für alle } a \in A. \end{aligned}$$

Ein interesantes Devisenspekulationsproblem läßt sich als Anwendung des Maximalflußproblems mit Flußmultiplikatoren betrachten. Wir wollen auf dem Devisenmarkt Währungen ineinander umtauschen und dabei einen Gewinn erzielen. Unser Plan ist der folgende. Wir starten mit einer noch zu bestimmenden Summe an DM, tauschen diese in Fremdwährungen um, diese möglicherweise mehrmals weiter um, und zum Schluß tauschen wir alles wieder in DM zurück. Von Ihren Urlaubsreisen wissen Sie vermutlich, daß Sie dabei fast immer Geld verlieren. Vielleicht aber kann man es mit Mathematik etwas besser machen. In den Tabellen 8.1 und 8.2 sind die Umtauschkurse vom 16.09.1994 bzw. 16.12.1994 (Quelle: Handelsblatt) zwischen 6 wichtigen Währungen aufgelistet. Kann man mit diesen Kursen die angestrebte "wunderbare Geldvermehrung" bewerkstelligen?

	DM	£	US $	SF	Yen	FF
DM	–	0.4103	0.6477	0.8297	64.0544	3.4171
£	2.4372	–	1.5785	2.0221	156.1136	8.3282
US $	1.5440	0.6335	–	1.2810	98.9000	5.2760
SF	1.2053	0.4945	0.7806	–	77.2053	4.1187
1000 Yen	15.6117	6.4056	10.1112	12.9525	–	53.3468
10 FF	2.9265	1.2007	1.8954	2.4280	187.4526	–

Wechselkurse vom 16.09.1994 *(Handelsblatt)*

Tabelle 8.1

	DM	£	US $	SF	Yen	FF
DM	–	0.4073	0.6365	0.8459	63.7770	3.4476
£	2.4552	–	1.5627	2.0768	156.5825	8.4644
US $	1.5707	0.6399	–	1.3285	100.1800	5.4151
SF	1.1822	0.4815	0.7524	–	75.3950	4.0756
1000 Yen	15.6796	6.3864	9.9800	13.2635	–	54.0569
10 FF	2.9006	1.1814	1.8462	2.4536	184.9903	–

Wechselkurse vom 16.12.1994 *(Handelsblatt)*

Tabelle 8.2

Wir machen zunächst einige zusätzliche Annahmen. Wir gehen in unserem Beispiel davon aus, daß der Umtausch kursneutral erfolgt, daß also keine Gebühren anfallen. Diese Annahme ist natürlich nur für Banken bzw. Devisenhändler erfüllt (falls überhaupt). Da jedoch Gebühren i. a. proportial zur Umtauschsumme anfallen, kann man diese durch Abschläge im Kurs auf direkte Weise berücksichtigen. Wir probieren nun eine Umtauschsequenz anhand der Tabellen 8.1 und 8.2 aus. Diese ist in Abbildung 8.2 dargestellt. Wir starten mit DM 1 Million, tauschen dann in US \$, dann in französische Franc und wieder zurück in DM.

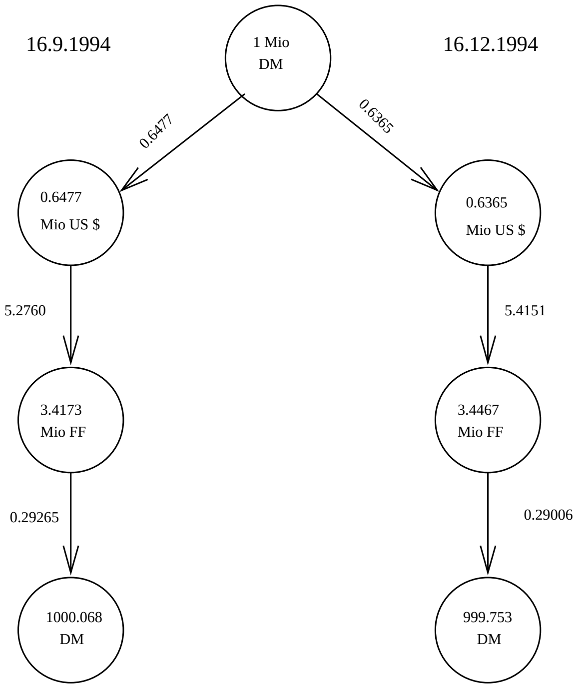

Abb. 8.2

Am 16.12.1994 ergibt sich dadurch ein Verlust von DM 247,– oder rund 0,02 % des eingesetzten Betrags, während sich am 16.09.1994 ein Gewinn von DM 63,– einstellt. Durch Erhöhung des eingesetzten Betrags kann man in unserem Modell den Gewinn beliebig erhöhen. Jedoch haben wir dabei nicht berücksichtigt, daß erhöhte Nachfrage bzw. erhöhtes Angebot zu Kursänderungen führt. In der Praxis ist es so, daß Devisenhändler die Umtauschkurse nur für gewisse maximale Geldbeträge garantieren und daß auch nur für wenige Minuten. Dies liegt daran, daß sich alle Devisenbörsen der Welt gleichzeitig beobachten und Kursschwankungen an einer Börse sofort zu Kursänderungen bei den anderen führen. Je nach Sachlage kann man die Umtauschsummenbeschränkungen dadurch berücksichtigen, daß man Bogenkapazitäten einführt oder daß man Umtauschbeschränkungen an jeder Devisenbörse festlegt. In unserem Beispiel folgen wir der zweiten Methode. Wir nehmen an, daß Tauschvorgänge, die folgenden Grenzen an einem Börsenplatz nicht überschreiten:

London	:	£	2 000 000,
New York	:	\$	4 000 000,
Zürich	:	SF	10 000 000,
Tokio	:	Yen	800 000 000,
Paris	:	FF	25 000 000,

keine Kursänderungen induzieren. Damit können wir unser Problem als das folgende lineare

Programm schreiben.

$$
(8.11)\qquad
\begin{aligned}
\max \sum_{a\in\delta^-(\mathrm{DM})} m(a)x(a) &- \sum_{a\in\delta^+(\mathrm{DM})} x(a)\\
\sum_{a\in\delta^+(v)} x(a) - \sum_{a\in\delta^-(v)} m(a)x(a) &= 0 \quad \forall\, v \in \{\pounds, \$, \mathrm{SF}, \mathrm{FF}, \mathrm{Yen}\}\\
\sum_{a\in\delta^+(v)} x(a) &\leq \begin{cases} 2.0 \text{ Mio} & \text{falls } v = \pounds\\ 4.0 \text{ Mio} & \text{falls } v = \$\\ 10.0 \text{ Mio} & \text{falls } v = \mathrm{SF}\\ 800.0 \text{ Mio} & \text{falls } v = \mathrm{Yen}\\ 25.0 \text{ Mio} & \text{falls } v = \mathrm{FF}\end{cases}\\
x_a &\geq 0 \quad \forall\, a \in A.
\end{aligned}
$$

Es gibt in der Literatur einige Vorschläge zur algorithmischen Behandlung der hier vorgestellten speziellen LP's. Aus Zeitgründen können wir darauf nicht eingehen. Wir haben das LP (8.11) für die zwei Probleme aus den Tabellen 8.1 und 8.2 mit dem Simplexalgorithmus gelöst. (Die Koeffizienten von (8.11) können z. B. wie folgt gelesen werden $m((\mathrm{DM}, \$))$ ist am 16.09.1994 gleich 0.6477 und am 16.12.1994 gleich 0.6365.) Es ergeben sich die in den Abbildungen 8.3 und 8.4 dargestellten Optimallösungen.

Man hätte also am 16.09.1994 unter Einsatz von 18,2 Mio DM einen Umtauschgewinn von DM 1004,– erzielen können. Durch die Summenbeschränkungen bei den einzelnen Währungen war kein höherer DM-Einsatz möglich.

Man beachte, daß natürlich aufgrund der im Modell nicht berücksichtigten Zeitbeschränkungen alle Umtauschaktionen innerhalb weniger Minuten (bzw. gleichzeitig) ausgeführt werden müssen, um den Gewinn zu realisieren. (Dieser wird natürlich durch die hohen Telefonkosten geschmälert.) Aber im Prinzip können diese Tauschvorgänge — auch nach Kursänderungen und damit verbundenen Neuberechnungen — immer wiederholt werden, so daß sich (theoretisch) relativ hohe Gewinne ansammeln können. Jedoch scheitert dieses Verfahren i. a. (zumindest für Nicht-Devisenhändler) am technisch nicht realisierbaren direkten Marktzugang. Nur wenige Händler haben eine Lizenz zur Durchführung dieser Transaktionen.

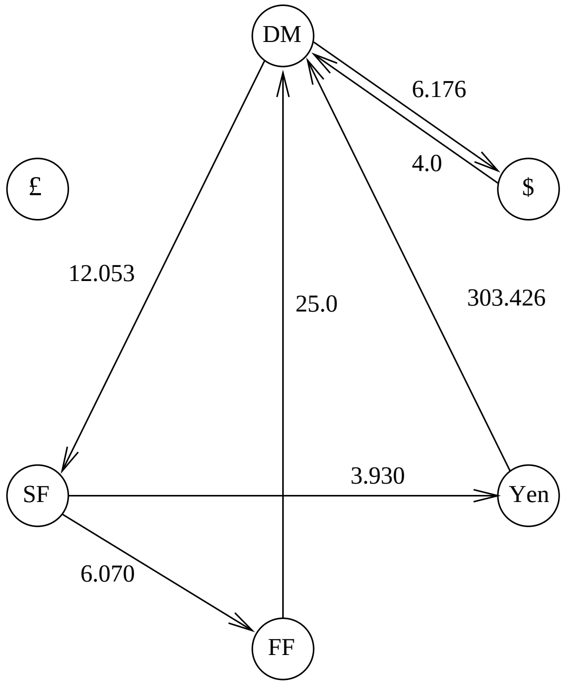

Abb. 8.3

Einsatz:	18 228 248
Gewinn:	1 004

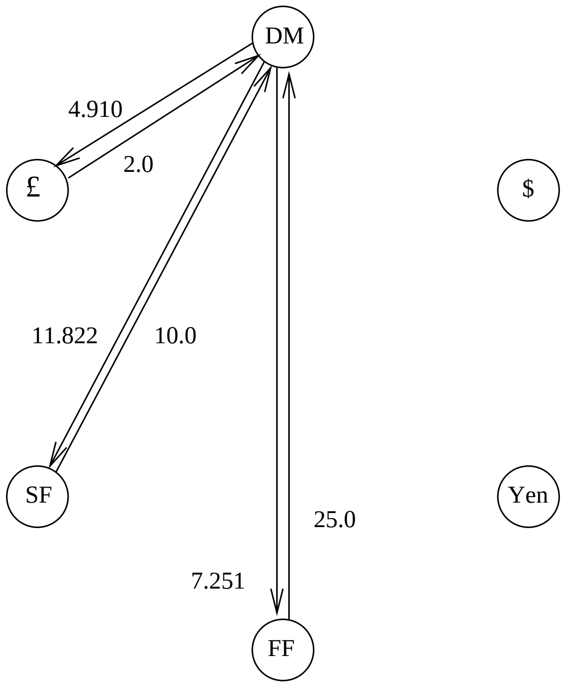

Abb. 8.4

Einsatz:	23 983 535
Gewinn:	365

8.3 Transsshipment-, Transport- und Zuordnungsprobleme

Wie beim Maximalflußproblem ist es natürlich möglich, Minimalkosten-Flußprobleme, bei denen mehrere Quellen und Senken vorkommen und bei denen von Null verschiedene untere Kapazitätsschranken für die Bögen vorkommen, zu lösen. Sie können auf einfache Weise auf das Standardproblem (8.1) reduziert werden.

Es gibt noch eine Reihe von Varianten des Minimalkosten-Flußproblems, die in der Literatur große Beachtung gefunden und viele Anwendungen haben. Ferner gibt es für alle dieser Probleme Spezialverfahren zu ihrer Lösung. Aus Zeitgründen können wir diese nicht behandeln. Wir wollen diese Probleme jedoch zumindest aus "Bildungsgründen" erwähnen und zeigen, wie sie in Minimalkosten-Flußprobleme transformiert werden können.

(8.12) Transsshipment-Probleme (Umladeprobleme). *Gegeben sei ein Digraph $D = (V, A)$, dessen Knotenmenge zerlegt sei in drei disjunkte Teilmengen V_a, V_n und V_u. Die Knoten aus V_a bezeichnen wir als* **Angebotsknoten**. *(Bei ihnen fließt ein Strom in das Netzwerk ein.) Die Knoten V_n bezeichnen wir als* **Nachfrageknoten** *(bei ihnen verläßt der Strom das Netz), und die Knoten V_u werden als* **Umladeknoten** *bezeichnet (hier wird der Fluß erhalten). Jedem Bogen $a \in A$ sind eine Kapazität $c(a)$ und ein Kostenkoeffizient $w(a)$ zugeordnet. Ferner sei bei jedem Angebotsknoten $v \in V_a$ die Menge $a(v)$ verfügbar, und bei jedem Nachfrageknoten die Menge $b(v)$ erwünscht. Die Aufgabe, einen Plan zu ermitteln, der Auskunft darüber gibt, von welchen Anbietern aus über welche Transportwege der Bedarf der Nachfrager zu decken ist, damit die Kosten für alle durchzuführenden Transporte minimiert werden, heißt* **Umladeproblem**.

□

Offenbar kann man ein Umladeproblem wie in (8.13) angegeben als lineares Programm schreiben.

$$
\begin{aligned}
&\min \sum w(a)x(a) \\
(8.13) \qquad & x(\delta^-(v)) - x(\delta^+(v)) = \begin{cases} 0 & \text{falls } v \in V_u \\ b(v) & \text{falls } v \in V_n \\ -a(v) & \text{falls } v \in V_a \end{cases} \\
& 0 \le x(a) \le c(a) \quad \forall\, a \in A.
\end{aligned}
$$

(8.13) ist offensichtlich höchstens dann lösbar, wenn $\sum_{v \in V_n} b(v) = \sum_{v \in V_a} a(v)$ gilt. Das lineare Programm (8.13) kann in ein Minimalkosten-Flußproblem (8.1) wie folgt transformiert werden. Wir führen eine (künstliche) Quelle s und eine künstliche Senke t ein. Die Quelle s verbinden wir mit jedem Knoten $v \in V_a$ durch einen Bogen (s, v) mit Kosten Null und Kapazität $a(v)$. Jeden Knoten $v \in V_n$ verbinden wir mit der Senke t durch einen Bogen (v, t) mit Kosten null und Kapazität $b(v)$. Offenbar liefert der kostenminimale (s, t)-Fluß in diesem neuen Digraphen mit Wert $\sum_{v \in V_n} b(v)$ eine optimale Lösung des Umladeproblems.

(8.14) Transportprobleme. Ein Transsshipment, bei dem $V_u = \emptyset$ gilt, heißt Transportproblem. Hier wird also von Erzeugern direkt zu den Kunden geliefert, ohne den Zwischenhandel einzuschalten.

□

(8.15) Zuordnungsproblem. Ein Transportproblem, bei dem $|V_a| = |V_n|$, $b(v) = 1$ $\forall\, v \in V_n$ und $a(v) = 1\ \forall\, v \in V_a$ gilt, heißt Zuordnungsproblem (vergleiche (2.9)).

□

Zur Lösung von Zuordnungs- und Transportproblemen gibt es besonders schnelle Algorithmen, die erheblich effizienter als die Algorithmen zur Bestimmung kostenminimaler Flüsse sind. Näheres hierzu wird in den Übungen behandelt. Aber auch in diesem Bereich kann man nicht — wie bei Algorithmen zur Bestimmung von Flüssen mit Minimalkosten – davon sprechen, daß irgendein Verfahren das schnellste (in der Praxis) ist. Immer wieder gibt es neue Implementationstechniken, die bislang unterlegene Algorithmen erheblich beschleunigen und anderen Verfahren überlegen erscheinen lassen. Siehe hierzu Ahuja, Magnanti und Orlin (1993).

Literaturverzeichnis von Kapitel 8

R. K. Ahuja, T. L. Magnanti und J. B. Orlin (1989), "Network Flows", Kapitel 4 des Buches G. L. Nemhauser et al. (eds.), *Handbooks in OR & MS*, Vol. 1, North-Holland, Amsterdam.

L. R. Ford, Jr. und D. R. Fulkerson (1962), *Flows in Networks*, Princeton University Press, Princeton.

V. Goldberg, É. Tardos und R. E. Tarjan (1990), "Network Flow Algorithms", in B. Korte et al. (eds.), *Paths, Flows, and VLSI-Layout*, Springer, Berlin.

T. C. Hu (1969), *Integer Programming and Network Flows*, Addison-Wesley, Reading, Massachusetts, deutsche Übersetzung (1972): *Ganzzahlige Programmierung und Netzwerkflüsse*, Oldenbourg, München.

P. A. Jensen und J. W. Barnes (1980), *Network Flow Programming*, Wiley & Sons, New York.

J. Kennington und R. Helgason (1980), *Algorithms for Network Programming*, Wiley & Sons, New York.

E. L. Lawler (1976), *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart & Winston, New York.