Diskrete Optimierung

Sommersemester 2005

Dozent: Maurice Cochand
Assistent: Gabrio Caimi

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Algorithmen

Kapitel 1

Grundlagen

1.1 Lineare Programmierung

In diesem Kapitel werden nur die für die Vorlesung wichtigen Elemente und Resultate der linearen Programmierung zusammengefasst. Für eine ausführliche Beschreibung und für die Beweise der erwähnten Resultate wird auf das Buch V. Chvátal: Lineare Programmierung, Freeman, 1983 verwiesen.

1.1.1 Das Problem der linearen Programmierung

Seien $c_1, c_2, \ldots, c_n$ reelle Zahlen und $x_1, \ldots, x_n$ reelle Variablen. Die Funktion f, definiert durch

$$f(x_1, \ldots, x_n) := c_1 x_1 + \cdots + c_n x_n$$

wird eine *lineare Funktion* genannt. Ist zusätzlich eine reelle Zahl b gegeben, so heisst die Gleichung

$$f(x_1, \ldots, x_n) = b$$

eine *lineare Gleichung* und die Ungleichungen

$$f(x_1, \ldots, x_n) \leq b, \quad f(x_1, \ldots, x_n) \geq b$$

lineare Ungleichungen. Lineare Gleichungen und lineare Ungleichungen werden auch *lineare Restriktionen* genannt.

Ein *lineares Programm (LP)* ist definiert durch

(i) eine lineare Zielfunktion, die es zu maximieren oder minimieren gilt,

(ii) lineare Restriktionen,

(iii) Nichtnegativitätsbedingungen auf einzelnen Variablen.

Im folgenden werden wir ein lineares Programm (LP) in folgender Form schreiben:

$$\max \sum_{j=1}^{n} c_j x_j \tag{1.1}$$

$$\text{so dass:} \quad \sum_{j=1}^{n} a_{ij} x_j \leq b_i \qquad i \in I_1 := \{1, \ldots, m'\} \tag{1.2}$$

$$\sum_{j=1}^{n} a_{ij} x_j = b_i \qquad i \in I_2 := \{m'+1, \ldots, m\} \tag{1.3}$$

$$x_j \geq 0 \qquad j \in J_1 := \{1, \ldots, n'\} \tag{1.4}$$

$$x_j \text{ frei} \qquad j \in J_2 := \{n'+1, \ldots, n\} \tag{1.5}$$

Man bemerke, dass eine lineare Restriktion der Form $f(x_1, \ldots, x_n) \geq b'$ auch als eine Restriktion (1.2) geschrieben werden kann, nämlich $-f(x_1, \ldots, x_n) \leq -b'$, und dass ein LP mit einer zu minimierenden Zielfunktion $f(x_1, \ldots, x_n)$ einem LP entspricht, bei dem $-f(x_1, \ldots, x_n)$ maximiert wird.

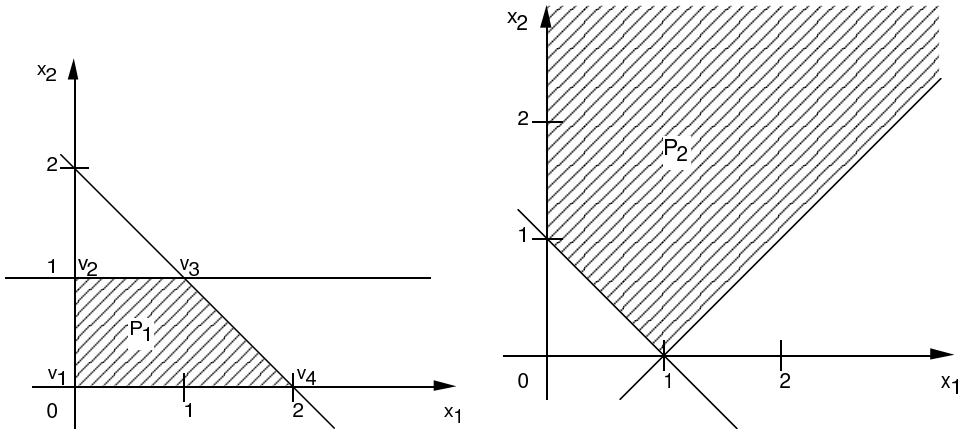
Eine *zulässige Lösung* des obigen LP's ist ein Vektor $x = (x_1, \ldots, x_n)$, der (1.2) - (1.5) erfüllt. Die Menge aller zulässigen Lösungen $P := \{x | x \text{ erfüllt } (1.2) \text{ bis } (1.5)\}$ wird dann als *Zulässigkeitsbereich* bezeichnet. Falls eine zulässige Lösung x zusätzlich noch

$$c^T x \equiv \sum_{j=1}^{n} c_j x_j$$

über alle zulässigen Lösungen maximiert, so ist x eine *optimale Lösung* des LP's.

1.1.2 Polyeder

Ein *Polyeder* $P \subseteq \mathbb{R}^n$ ist der Durchschnitt von endlich vielen Halbräumen, wobei ein *Halbraum* eine Menge der Form $H := \{x \in \mathbb{R}^n | a^T x \leq b'\}$ mit $a \in \mathbb{R}^n, b' \in \mathbb{R}$ ist.



(a) P_1 (b) P_2

Abbildung 1.1: Beispiele von Polyedern: $P_1 := \{x \in \mathbb{R} | x_2 \leq 1; x_1 + x_2 \leq 2; x_1, x_2 \geq 0\}$ ist beschränkt und $P_2 := \{x \in \mathbb{R} | x_2 \leq 1; x_1 + x_2 \leq 2; x_1, x_2 \geq 0\}$ ist unbeschränkt

Falls also $P \subseteq \mathbb{R}^n$ ein Polyeder ist, so existieren eine Matrix A und ein Vektor b, so dass $P = \{x \in \mathbb{R}^n | Ax \leq b\}$ ist. Man bemerke, dass der Zulässigkeitsbereich eines LP's ein Polyeder ist.

Betrachtet man das Polyeder P_1 der Abbildung 1.1, so ist P_1 auch vollständig durch die Vektoren $v^1, \ldots, v^4$ beschrieben. Wir werden sehen, dass eine ähnliche Beschreibung für jedes Polyeder existiert. Dazu benötigen wir folgende Definitionen:

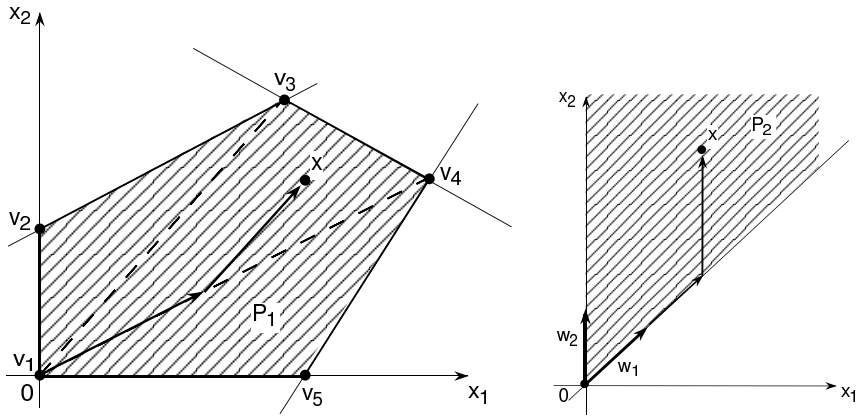
$$CONV(\{v^1, \ldots, v^M\}) := \Bigg\{ x \in \mathbb{R}^n | \, x = \sum_{i=1}^{M} \lambda_i v^i, \quad \text{für } \lambda_i \geq 0, \; i = 1, \ldots, M \quad \text{mit } \sum_{i=1}^{M} \lambda_i = 1 \Bigg\} \tag{1.6}$$

$$CONE(\{v^1, \ldots, v^M\}) := \Bigg\{ x \in \mathbb{R}^n | \, x = \sum_{i=1}^{M} \lambda_i v^i, \quad \text{für } \lambda_i \geq 0, \; i = 1, \ldots, M \Bigg\} \tag{1.7}$$

$CONE(\{v^1, \ldots, v^M\})$ ist der *Kegel*, der von $v^1, \ldots, v^M$ *aufgespannt* wird, während $CONV(\{v^1, \ldots, v^M\})$ die *konvexe Hülle* von $v^1, \ldots, v^M$ ist. Ein Vektor

$$x = \sum_{i=1}^{M} \lambda_i v^i, \text{ mit } \sum_{i=1}^{M} \lambda_i = 1, \ \lambda_i \geq 0, \ i = 1, \ldots, M$$

ist eine *Konvexkombination* der Vektoren $v^1, \ldots, v^M$.



(a) $P_1 = CONV(\{v^1, \ldots, v^5\})$ (b) $P_2 = CONE(\{w^1, w^2\})$

Abbildung 1.2: Konvexe Hülle und aufgespannter Kegel: in (a) ist x eine Konvexkombination von $\{v^1, v^3, v^4\}$ und in (b) ist $x \in CONE(\{w^1, w^2\})$

Satz 1.1 (Minkowski) *Sei $P := \{x \in \mathbb{R}^n | Ax \leq b\} \neq \emptyset$ ein Polyeder. Dann existieren endlich viele Vektoren $v^1, \ldots, v^M$ und $w^1, \ldots, w^N$ in $\mathbb{R}^n, M > 0, N \geq 0$, so dass*

$$P = CONV(\{v^1, \ldots, v^M\}) + CONE(\{w^1, \ldots, w^N\}). \tag{1.8}$$

Das heisst, $x \in P$ dann und nur dann, falls reelle Zahlen $\lambda_i \geq 0, i = 1, \ldots, M$, und $\mu_j \geq 0, j = 1, \ldots, N$ existieren mit

$$\sum_{i=1}^{M} \lambda_i = 1, \text{ und } x = \sum_{i=1}^{M} \lambda_i v^i + \sum_{j=1}^{N} \mu_j w^j \tag{1.9}$$

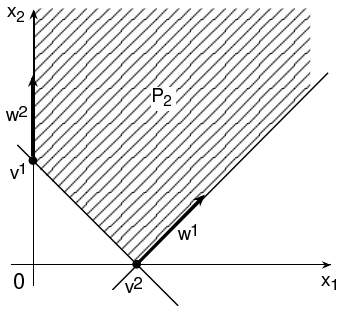


Abbildung 1.3: $P = CONV(\{v^1, v^2\}) + CONE(\{w^1, w^2\})$

Auch die Umkehrung gilt:

Satz 1.2 (Weyl) *Seien $v^1, \ldots, v^M$ und $w^1, \ldots, w^N$ Vektoren in $\mathbb{R}^n, M > 0, N \geq 0$, dann ist*

$$P := CONV(\{v^1, \ldots, v^M\}) + CONE(\{w^1, \ldots, w^N\})$$

ein Polyeder, d. h. es existieren eine Matrix A und ein Vektor b, so dass das Polyeder geschrieben werden kann als $P = \{x \in \mathbb{R}^n | Ax \leq b\}$.

Für das beschränkte Polyeder P_1 der Abbildung 1.2 ist $P_1 = CONV(\{v^1, \ldots, v^5, x\}) = CONV(\{v^1, \ldots, v^5\})$. Für die Darstellung von P_1 als konvexe Hülle von Vektoren ist x nicht notwendig, da x bereits eine Konvexkombination von $v^1, \ldots, v^5$ ist. Hingegen kann keiner der Vektoren $v^1, \ldots, v^5$, der sogenannten *Eckpunkte*, weggelassen werden:
Sei P ein Polyeder. Der Vektor $x \in P$ ist ein *Eckpunkt* von P, falls x nicht Konvexkombination zweier anderer Vektoren aus P ist, d.h. $x \in P$ ist *Eckpunkt*, falls aus $x = \lambda x' + (1 - \lambda)x'', 0 \leq \lambda \leq 1$ und $x', x'' \in P, x' \neq x''$, folgt $\lambda = 0$ oder $\lambda = 1$.
Ist $P := \{x \in \mathbb{R}^n | x$ erfüllt (1.2) bis (1.5)$\}$, so kann ein Eckpunkt auch wie folgt charakterisiert werden:

Satz 1.3 *Folgende Aussagen sind äquivalent:*

(i) x ist ein Eckpunkt.

(ii) x ist zulässig und es existieren Teilmengen $I' \subseteq I_1, J' \subseteq J_1$, so dass x die einzige Lösung des folgenden Gleichungssystems ist:

$$\sum_{j=1}^{n} a_{ij}x_j = b_i, \quad i \in I' \cup I_2$$
$$x_j = 0, \qquad j \in J'.$$

(iii) Es existiert ein Zielfunktionsvektor c', so dass x die einzige optimale Lösung des LP's, „$\max c'^T x$, so dass x erfüllt (1.2) bis (1.5)“, ist.

Beispiel 1.4 Sei $P_1 := \{x \in \mathbb{R}^2 | x_2 \leq 1; x_1 + x_2 \leq 2; x_1, x_2 \geq 0\}$ das Polyeder der Abbildung 1.1. Die Punkte v^2 und v^3 sind offensichtlich Eckpunkte, also muss auch (ii) und (iii) gelten. Effektiv ist v^2 einzige Lösung des Gleichungssystems „$x_2 = 1; x_1 = 0$“ und v^3 einzige Lösung von „$x_2 = 1; x_1 + x_2 = 2$“. Betrachte nun das LP „$\max c^T x$ so dass $x \in P_1$“. Wählt man für $c^T := (-1, 1)$, so ist v^2 die einzige optimale Lösung, wählt man $c^T := (1, 2)$, so ist v^3 die einzige optimale Lösung. ◇

Man bemerke, dass Polyeder existieren, die keine Eckpunkte besitzen:

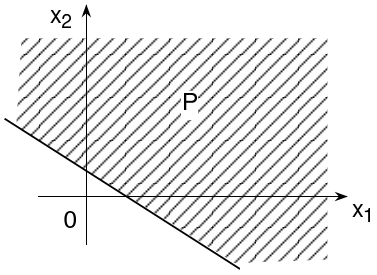


Abbildung 1.4: Polyeder ohne Eckpunkte

Falls aber ein Polyeder mindestens einen Eckpunkt besitzt, so gilt:

Satz 1.5 *Sei $P \neq \emptyset$ ein Polyeder, das Eckpunkte besitzt, so gilt:*

(i) Es existieren endlich viele Eckpunkte $v^1, \ldots, v^N$ von P. Falls P beschränkt ist (es existieren $a, d \in \mathbb{R}^n$ so dass $P \subseteq \{x \in \mathbb{R}^n | a \leq x \leq d\}$), so ist $P = CONV(\{v^1, \ldots, v^N\})$.

(ii) Jedes LP „$\max c^T x$ so dass $x \in P$“ besitzt eine optimale Lösung die ein Eckpunkt ist, sofern das LP eine optimale Lösung besitzt.

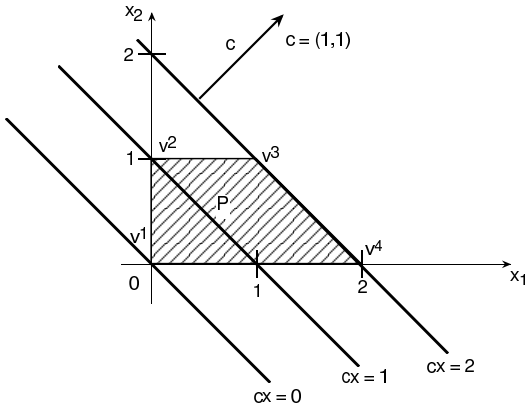


Abbildung 1.5: Alle Punkte auf der Strecke $[v^3, v^4]$ sind optimal, v^3, v^4 sind die optimalen Lösungen, die Eckpunkte sind.

1.1.3 Dualität

Betrachte folgendes LP:

$$\text{(PLP)} \qquad z^* = \max\ 4x_1 + x_2 + 6x_3 + 3x_4 = f(x_1, x_2, x_3, x_4) \tag{1.10}$$

$$x_1 - x_2 - x_3 + 3x_4 = 1 \tag{1.11}$$

$$5x_1 + x_2 + 3x_3 + 8x_4 \leq 55 \tag{1.12}$$

$$-x_1 + 2x_2 + 3x_3 - 5x_4 \leq 3 \tag{1.13}$$

$$x_1,\ x_2,\ x_4 \geq 0 \tag{1.14}$$

$$x_3 \quad \text{frei} \tag{1.15}$$

Weiter sei $Q := \{x \in \mathbb{R}^4 | x \text{ erfüllt } (1.14), (1.15)\} \supseteq P := \{x \in \mathbb{R}^4 | x \text{ erfüllt } (1.11) - (1.15)\}$.
Anstatt das LP zu lösen, wollen wir versuchen, den Wert von z^* abzuschätzen. Um eine untere Schranke zu erhalten, genügt es, eine zulässige Lösung zu kennen; z. B. ist $x' := (1, 0, 0, 0)$ zulässig, also ist $z^* \geq f(x') = 4$, oder $x'' := (3, 0, 2, 0)$ impliziert $z^* \geq f(x'') = 24$. (Sicher ist ein systematisches Berechnen einer optimalen Lösung effizienter als probieren, wir interessieren uns hier aber für Schranken für z^*.)
Wir wollen nun obere Schranken suchen. Multipliziert man die Restriktion

(1.12) mit 2, so erhält man für alle $x \in P$

$$\begin{array}{rcrcrcrcrcl} 110 & & \geq & 10x_1 & + & 2x_2 & + & 6x_3 & + & 16x_4 & \\ & & \underset{\text{da } x_1,x_2,x_4\geq 0}{\geq} & 4x_1 & + & x_2 & + & 6x_3 & + & 3x_4 & = & f(x_1,x_2,x_3,x_4) \end{array}$$

Somit ist $f(x) \leq 110$ für alle $x \in P$, also $z^* \leq 110$. Falls wir die Restriktionen (1.12) und (1.13) zusammenzählen, erhalten wir für alle $x \in P$:

$$\begin{array}{rcrcrcrcrcl} 58 & \geq & 4x_1 & + & 3x_2 & + & 6x_3 & + & 3x_4 & & \\ & \underset{\text{da } x_2\geq 0}{\geq} & 4x_1 & + & x_2 & + & 6x_3 & + & 3x_4 & = & f(x_1,x_2,x_3,x_4) \end{array}$$

also $z^* \leq 58$. Für gewisse Linearkombinationen der Restriktionen (1.11) - (1.13) können wir auf diese Weise obere Schranken für z^* erhalten. Betrachten wir eine solche Linearkombination mit Koeffizienten y_1, y_2, y_3 und $y_2, y_3 \geq 0$:

$$\begin{array}{rcll} x_1 - x_2 - x_3 + 3x_4 = 1 & | & \cdot y_1 \quad \text{frei} & \text{(aus 1.11)} \\ 5x_1 + x_2 + 3x_3 + 8x_4 \leq 55 & | & \cdot y_2 \geq 0 & \text{(aus 1.12)} \\ -x_1 + 2x_2 + 3x_3 - 5x_4 \leq 3 & | & \cdot y_3 \geq 0 & \text{(aus 1.13)} \end{array}$$

Dies ergibt nun folgende Ungleichung:

$$\begin{array}{c} (y_1 + 5y_2 - y_3)x_1 + (-y_1 + y_2 + 2y_3)x_2 \\ +(-y_1 + 3y_2 + 3y_3)x_3 + (3y_1 + 8y_2 - 5y_3)x_4 \leq y_1 + 55y_2 + 3y_3 \end{array} \tag{1.16}$$

Falls $x \in P$, so erfüllt es sicher die Restriktionen (1.11) bis (1.13) und somit auch (1.16). Man beachte, dass eine Gleichung mit einem beliebigen Koeffizienten, eine Ungleichung nur mit einem nicht–negativen Koeffizienten multipliziert werden darf. Die rechte Seite von (1.16) liefert sicher eine obere Schranke für z^*, falls die linke Seite von (1.16) grösser gleich $f(x)$ ist für alle $x \in Q \supseteq P$, das heisst, falls y_1, y_2, y_3 so gewählt wurden, dass für alle $x \in Q$ gilt:

$$\begin{array}{ccccccccc} (y_1 + 5y_2 - y_3)x_1 & + & (-y_1 + y_2 + 2y_3)x_2 & + & (-y_1 + 3y_2 + 3y_3)x_3 & + & (3y_1 + 8y_2 - 5y_3)x_4 & \geq \\ 4x_1 & & +1x_2 & & +6x_3 & & +3x_4 & \end{array}$$

Damit diese Ungleichung für alle $x \in Q$ gilt, muss zum Beispiel $(y_1 + 5y_2 - y_3)x_1 \geq 4x_1$ für alle $x_1 \geq 0$, also $(y_1 + 5y_2 - y_3) \geq 4$ sein, und $(-y_1 + 3y_2 + 3y_3)x_3 \geq 6x_3$ für alle x_3, also $(-y_1 + 3y_2 + 3y_3) = 6$ sein.

Zusammengefasst gilt also: Falls

$$y_1 \quad \text{frei} \tag{1.17}$$
$$y_2\ y_3 \geq 0 \tag{1.18}$$
$$y_1 + 5y_2 - y_3 \geq 4 \tag{1.19}$$
$$-y_1 + y_2 + 2y_3 \geq 1 \tag{1.20}$$
$$-y_1 + 3y_2 + 3y_3 = 6 \tag{1.21}$$
$$3y_1 + 8y_2 - 5y_3 \geq 3 \tag{1.22}$$

so ist $y_1+55y_2+3y_3 \geq z^*$. Man beachte, dass wir $(y_1, y_2, y_3) = (0, 1, 1)$ gewählt haben, um die obere Schranke 58 zu erhalten. Sucht man eine minimale obere Schranke auf diese Weise, so gilt es folgendes lineare Programm zu lösen:

$$\min\ y_1 + 55y_2 + 3y_3,\ \text{sodass}\ (y_1, y_2, y_3)\ \text{ erfüllt (1.17) bis (1.22).} \tag{DLP}$$

Dieses lineare Programm wird das zum (PLP) *duale LP* genannt. In diesem Kontext bezeichnet man dann das (PLP) als das *primale LP*.

Primal						Dual		
$\max \sum_{j=1}^{n} c_j x_j$						$\min \sum_{i=1}^{m} b_i y_i$		
$\sum_{j=1}^{n} a_{ij} x_j$	$\leq$	b_i	$i \in I_1$	$=$	$\{1, \ldots, m'\}$	y_i	$\geq$	0
$\sum_{j=1}^{n} a_{ij} x_j$	$=$	b_i	$i \in I_2$	$=$	$\{m'+1, \ldots, m\}$	y_i		frei
x_j	$\geq$	0	$j \in J_1$	$=$	$\{1, \ldots, n'\}$	$\sum_{i=1}^{m} a_{ij} y_i$	$\geq$	c_j
x_j		frei	$j \in J_2$	$=$	$\{n'+1, \ldots, n\}$	$\sum_{i=1}^{m} a_{ij} y_i$	$=$	c_j

Tabelle 1.1: Konstruktion von DLP und PLP und umgekehrt

Für ein allgemeines (primales) LP, mit Zielfunktion $c^T x$, wird das duale wie folgt konstruiert (vgl Tabelle 1.1.3): Zu jeder Restriktion, die eine Ungleichung (Gleichung) ist, wird eine nicht–negative (freie) duale Variable assoziert. Die Multiplikation der Restriktionen mit diesen Variablen ergibt eine

Restriktion $g(y)^T x \leq d(y)$ (vgl. (1.16)), wobei x der Vektor der primalen und y der Vektor der dualen Variablen ist. Damit $c^T x \leq g(y)^T x$ für alle primal zulässigen Lösungen x gilt, wird zusätzlich jeder primalen Variablen x_i eine duale Restriktion zugeordnet und zwar eine Ungleichung (Gleichung) für jede nicht–negative (freie) primale Variable (vgl. (1.19) – (1.22)). Für das duale Problem soll dann die Funktion $d(y)$ minimiert werden.

Eine *primal zulässige (optimale) Lösung* ist eine zulässige (optimale) Lösung des primalen LP's, eine *dual zulässige (optimale) Lösung* ist eine zulässige (optimale) Lösung des dualen LP's. Man bemerke, dass das duale LP wieder ein LP ist und dass das duale des dualen LP's gleich dem primalen LP ist.

Gemäss Konstruktion des dualen LP's gilt folgender Satz:

Satz 1.6 (Schwacher Dualitätssatz) *Für jede primal zulässige Lösung x und jede dual zulässige Lösung y gilt:*

$$\sum_{j=1}^{n} c_j x_j \equiv c^T x \leq \sum_{i=1}^{m} b_i y_i$$

Falls eine Heuristik eine primal und dual zulässige Lösung x und y liefert, haben wir für den Optimalwert z^* des primalen LP's eine untere und obere Schranke, denn $c^T x \leq z^* \leq y^T b$. Gilt weiter $c^T x = y^T b$, so wissen wir, dass x primal optimal und y dual optimal ist. Das duale LP liefert uns also ein Optimalitätskriterium.

Im obigen Beispiel ist der Vektor $x = (0, \frac{130}{9}, -\frac{1}{9}, \frac{46}{9})$ eine zulässige Lösung von (PLP) und $y = (\frac{78}{9}, \frac{1}{9}, \frac{43}{9})$ eine zulässige Lösung von (DLP). Da $f(x) = 4x_1 + x_2 + 6x_3 + 3x_4 = \frac{262}{9} = y_1 + 55y_2 + 3y_3$, ist x primal optimal und y dual optimal. Effektiv haben immer beide LP's die gleichen Optimalwerte (falls sie existieren):

Satz 1.7 (Starker Dualitätssatz) *Falls sowohl das primale wie das duale LP eine zulässige Lösung besitzen, so existieren eine primal zulässige Lösung x und eine dual zulässige Lösung y, so dass $c^T x = y^T b$: Die Lösungen x und y sind dann optimal.*

Zum Schluss geben wir noch ein weiteres Optimalitätskriterium an:

	Existenz von dual optimaler Lösung	Es existiert keine dual zulässige Lösung	Das duale LP ist unbeschränkt
es existiert eine primal optimale Lösung	möglich	unmöglich	unmöglich
es existiert keine primal zulässige Lösung	unmöglich	möglich	möglich
das primale LP ist unbeschränkt	unmöglich	möglich	unmöglich

Tabelle 1.2: Mögliche und unmögliche Situationen für PLP und DLP

Satz 1.8 (Komplementärschlupfbedingungen) *Eine primal zulässige Lösung x und eine dual zulässige Lösung y sind optimal, genau dann, wenn (i) und (ii) gelten:*

$$\textit{(i)}\ y_i > 0, i \in I_1 \quad \Rightarrow \quad \sum_{j=1}^{n} a_{ij}x_j = b_i$$

$$\textit{(ii)}\ x_j > 0, j \in J_1 \quad \Rightarrow \quad \sum_{i=1}^{m} a_{ij}y_i = c_j$$

(i) und (ii) heissen Komplementärschlupfbedingungen. Für ein primales und zugehöriges duales LP können folgende Situationen auftreten:

1.1.4 Idee des Simplexalgorithmus

Sei $A \in \mathbb{R}^{m\times n}$ eine Matrix mit Rang $m, P := \{x \in \mathbb{R}^n \mid Ax = b, x \geq 0\} \neq \emptyset$ und betrachte ein LP der Form

$$z_{LP} = \max\ cx, \quad \text{sodass } Ax = b,\ x \geq 0 \qquad \text{(PLP)}$$

das eine optimale Lösung besitzt.
Um eine optimale Lösung von (PLP) zu finden, genügt es, die Eckpunkte von P zu betrachten, da gemäss Satz 1.5(ii) immer eine optimale Lösung existiert, die ein Eckpunkt von P ist. Der Simplexalgorithmus nützt diese Tatsache aus und funktioniert, grob gesagt, wie folgt: Er startet mit einem Eckpunkt von P. In jedem Schritt versucht er, zu einem Nachbareckpunkt

mit besserem Zielfunktionswert zu gelangen. Dies wiederholt er so lange, bis er eine optimale Lösung gefunden hat.
Der Simplexalgortihmus sucht also systematisch Eckpunkte von P ab. Genauer gesagt, sucht er sogenannte zulässige *Basislösungen* ab, die wie folgt definiert sind:

Definition 1.9 (Basis) *Sei $A = (a_1, \ldots, a_n)$, wobei $a_j, j = 1, \ldots, n$ die Kolonnen der Matrix A sind. Eine Basis $B = \{\beta_1, \ldots, \beta_m\} \subseteq \{1, \ldots, n\}$ ist eine maximale Auswahl von linear unabhängigen Kolonnen von A. Mit A^B wird die Teilmatrix von A bezeichnet, die aus den Kolonnen $(a_{\beta_1}, \ldots, a_{\beta_m})$ besteht und mit A^N die Teilmatrix von A, die aus den restlichen Kolonnen $N := \{1, \ldots, n\} \setminus B$ besteht. Nach Permutation der Kolonnen von A ist also $A = [A^B, A^N]$ und $Ax = b$ kann als $A^B x^B + A^N x^N = b$ geschrieben werden, wobei $x = \begin{bmatrix} x^B \\ x^N \end{bmatrix}$ ist. Die Variablen $x_i, i \in B$, werden Basisvariable, die Variablen $x_i, i \in N$, Nichtbasisvariable genannt.*

Die zu B gehörige *Basislösung* $x' = \begin{bmatrix} x'^B \\ x'^N \end{bmatrix}$ ist die eindeutige Lösung des Systems

$$Ax = b, x^N = 0, \text{ d.h. } \begin{bmatrix} x'^B \\ x'^N \end{bmatrix} = \begin{bmatrix} (A^B)^{-1} b \\ 0 \end{bmatrix}.$$

Ist $x' = \begin{bmatrix} x'^B \\ x'^N \end{bmatrix}$ zulässig, so heisst x' *zulässige Basislösung* und B heisst dann *zulässige Basis*.
Wir sagen, dass zwei Basen B und B' *Nachbarbasen* sind, falls $B' = B \cup \{k\} \setminus \{s\}$ für $s \in B, k \notin B$.

Man bemerke, dass jede zulässige Basislösung x' ein Eckpunkt von P ist (Satz 1.3 (ii)) und dass zu verschiedenen Basen die gleiche Basislösung (Eckpunkt) gehören kann (vgl. Beispiel weiter unten).

Betrachte nun das zum (PLP) duale LP

$$\min\ y^T b, \quad \text{sodass } y^T A \geq c,\ y \text{ frei} \tag{DLP}$$

Sei B eine Basis, $A = [A^B, A^N], c^T = [c^B, c^N]$, so dass die zugehörige Basislösung x', mit $x'^B = (A^B)^{-1} b, x'^N = 0$ zulässig ist. Zu B kann auch eine duale, im allgemeinen nicht zulässige Lösung zugeordnet werden, nämlich

$y := (c^B)^T(A^B)^{-1}$. Für dieses y und der Basislösung x'gilt $c^T x' = (c^B)^T x'^B = (c^B)^T(A^B)^{-1}b = y^T b$. Falls also y zulässig ist, so ist x' primal optimal und y dual optimal.

Der Simplexalgorithmus startet mit einer primal zulässigen Basis B und überprüft, ob die zugehörige duale Lösung $y := (c^B)^T(A^B)^{-1}$ dual zulässig ist. Ist dies der Fall, ist die zu B gehörige Basislösung primal optimal und y dual optimal. Sonst wird entweder eine Nachbarbasis erzeugt mit Zielfunktionswert grösser gleich dem momentanen Wert oder es wird festgestellt, dass das primale LP kein endliches Optimum besitzt.

Es bleibt zu präzisieren, wie die Nachbarbasis bestimmt wird und wie man gegebenenfalls merkt, dass das LP kein endliches Optimum besitzt.
Sei B eine zulässige Basis, x^1 die zugehörige Basislösung und

$$A = [A^B, A^N], c^T = [c^B, c^N], x = \begin{bmatrix} x^B \\ x^N \end{bmatrix}.$$

Man bemerke, dass $Ax = b$ als $x^B + (A^B)^{-1}A^N x^N = (A^B)^{-1}b$ geschrieben werden kann und somit ist $x^B = (A^B)^{-1}b - (A^B)^{-1}A^N x^N$. Der Zielfunktionswert $c^T x$ ist dann

$$\begin{aligned} c^T x &= (c^B)^T x^B + (c^N)^T x^N \\ &= (c^B)^T[(A^B)^{-1}b - (A^B)^{-1}A^N x^N] + (c^N)^T x^N \\ &= (c^B)^T(A^B)^{-1}b + ((c^N)^T - (c^B)^T(A^B)^{-1}A^N)x^N \end{aligned}$$

Zusammengefasst kann das (PLP) wie folgt geschrieben werden:

(PLP') $$\begin{aligned} z_{LP} &= (c^B)^T(A^B)^{-1}b + \max(c^N - (c^B)^T(A^B)^{-1}A^N)x^N \\ x^B + (A^B)^{-1}A^N x^N &= (A^B)^{-1}b \\ x^B, x^N &\geq 0. \end{aligned}$$

Sei $A'^N := (A^B)^{-1}A^N, b' := (A^B)^{-1}b, c'^N := c^N - c^B(A^B)^{-1}A^N$, dann ist

(PLP') $$\begin{aligned} z_{LP} &= (c^B)^T b' + \max(c'^N)^T x^N \qquad & z_{LP} &= (c^B)^T b' + \max \sum_{j \in N} c'_j x_j \\ x^B + A'^N x^N &= b' \qquad \equiv & x_i + \sum_{j \in N} a'_{ij} x_j &= b'_i, \quad i \in \end{aligned}$$

B

$$x^B, x^N \qquad \geq 0 \qquad\qquad x_i \geq 0, i \in B, x_j \geq 0, \quad j \in N,$$

wobei $a'_{ij}, i \in B, j \in N$ die Elemente der Matrix A'^N sind. Der Vektor c'^N wird der Vektor der reduzierten Kosten genannt. Er ist genau dann nicht positiv, wenn die zu B zugehörige duale Lösung $y := (c^B)^T(A^B)^{-1}$ dual zulässig ist: Die duale Restriktion $y^T A \geq c$ ist gleich $y^T[A^B, A^N] \geq [c^B, c^N]$, d. h. $y^T A^B \geq c^B$ und $y^T A^N \geq c^N$. Für $y := (c^B)^T(A^B)^{-1}$ ist die erste Restriktion trivialerweise erfüllt und die zweite lautet $(c^B)^T(A^B)^{-1}A^N \geq c^N$, d. h. $c'^N \leq 0$.

Eine Nachbarbasis $B' = B \cup \{k\} \setminus \{s\}$ von B ist durch die neu in die Basis eintretende Kolonne $k \in N$ und die neu aus der Basis austretende Kolonne $s \in B$ bestimmt. Die zu B' gehörige Basislösung x^2 ist dann einzige Lösung von $Ax = b$ und $x_j = 0$ für $j \in N \cup \{s\} \setminus \{k\}$, oder äquivalenterweise Lösung von

$$x_i + \sum_{j \in N} a'_{ij} x_j = b'_i \text{ für alle } i \in B \text{ und } x_j = 0 \text{ für alle } j \in N \cup \{s\} \setminus \{k\}. \tag{1.23}$$

Man bemerke, dass $a'_{sk} \neq 0$ sein muss, damit das obige System eine eindeutige Lösung besitzt und somit B' eine Basis ist. Die zu B' gehörige Basislösung lautet dann:

$$\begin{aligned} x^2_k = \frac{b'_s}{a'_{sk}};\ x^2_i = b'_i - a'_{ik} x^2_k \text{ für alle } i \in B \setminus \{s\} \text{ und} \\ x^2_i = 0 \text{ für alle } i \in N \cup \{s\} \setminus \{k\}. \end{aligned} \tag{1.24}$$

Für die zu B gehörige Basislösung x^1 ist der Zielfunktionswert $z^1 = (c^B)^T b'$ (da $x^1_j = 0, j \in N$), für die zu B' gehörige Basislösung x^2 ist er gleich $z^2 = (c^B)^T b' + (c'^N)^T x^N = (c^B)^T b' + c'_k x^2_k$ (da $x^2_j = 0, j \in N \setminus \{k\}$). Damit z^2 grösser z^1 werden kann, wählen wir eine eintretende Kolonne $k \in N$ so, dass $c'_k > 0$ ist (ein solches k existiert sicher, falls $y := (c^B)^T(A^B)^{-1}$ dual unzulässig ist) und wählen dann x^2_k möglichst gross.

Bemerke, dass x^2 zulässig sein muss und deshalb folgende Restriktionen erfüllen muss:

$$x^2_k \geq 0 \text{ und } x^2_i = b'_i - a'_{ik} x^2_k \geq 0 \text{ für alle } i \in B. \tag{1.25}$$

Falls $a'_{ik} \leq 0$ für alle $i \in B$, so kann x_k^2 beliebig gross gewählt werden und somit z^2 beliebig gross werden. (PLP) besitzt dann kein endliches Optimum. Sonst ist x_k^2 wegen (1.25) beschränkt durch

$$\min \left\{ \frac{b'_i}{a'_{ik}} > 0, i \in B \right\} = \frac{b'_s}{a'_{sk}} \text{ für ein } s \in B.$$

Wählt man dieses s als die austretende Kolonne, so ist $B' := B \cup \{k\} \setminus \{s\}$ eine zulässige Nachbarbasis und die durch (1.24) definierte zugehörige Basislösung x^2 ist zulässig. Falls $x_k^2 > 0$, so hat der Zielfunktionswert um $z^2 - z^1 = c'_k x_k^2 > 0$ zugenommen, sonst ist $x^1 = x^2$.

Zusammengefasst lautet der Simplexalgorithmus wie folgt:

Algorithmus 1.1 Simplex Algorithmus

begin (* Simplexalgorithmus *)
  (* Initialisieren *)
  Finde zulässige Basis B
  Stopkriterium : = **false**
  repeat
    Sei $A = [A^B, A^N], c = [c^B, c^N]$ und x' die zu B gehörige Basislösung
    (* Optimalitätstest *)
    if $y = (c^B)^T (A^B)^{-1}$ dual zulässig **then**
      x' ist primal optimal, y ist dual optimal
      Stopkriterium : = **true**
    else
      (* Wahl in Basis eintretende Kolonne *)
      Wähle $k \in N$ mit $c'_k > 0$
      (* Test für unbeschränktes Optimum *)
      if $a'_{ik} \leq 0$ für alle $i \in B$ **then**
        (PLP) besitzt kein endliches Optimum
        Stopkriterium : = **true**
      else
        Sei $s \in B$ mit $\frac{b'_s}{a'_{sk}} = \min \left\{ \frac{b'_i}{a'_{ik}} \mid a'_{ik} > 0, i \in B \right\}$
        $B := B \cup \{k\} \setminus \{s\}$
      end
    end
  until Stopkriterium
end (* Simplexalgorithmus *)

Man bemerke, dass eine Änderung der Basislösungen bei einem Basis–Austauschschritt immer eine strikte Zunahme der Zielfunktion impliziert. Im allgemeinen können aber auch Basisaustausche vorkommen, bei denen die Basislösungen nicht ändern ($b'_s = 0$). In diesen Fällen muss darauf geachtet werden, dass der Algorithmus nicht zwischen verschiedenen Basen mit gleichen Basislösungen zykliert. Es gibt einfache Regeln, die dieses Zyklieren verhindern, darauf soll aber hier nicht eingegangen werden.

Der hier aufgeführte Simplexalgorithmus ist nicht operationell. Mit einer geeigneten Tableau–Darstellung lassen sich die jeweils benötigten Grössen

$A'^N := (A^B)^{-1}A^N, b' := (A^B)^{-1}b$ und $c'^N := c^N - (c^B)^T(A^B)^{-1}A^N$ iterativ berechnen. Wir werden in der Vorlesung nur das Konzept des Simplexalgorithmus verwenden, deshalb gehen wir hier nicht auf diese Darstellung ein.

Beispiel 1.10 Betrachte das LP

$$\max\ x_1 + 2x_2,\ \text{so dass}\ x_2 \leq 1; x_1 + x_2 \leq 2; -x_1 + x_2 \leq 1; x_1, x_2 \geq 0. \tag{PLP0}$$

Zuerst werden sogenannte *Schlupfvariable* $x_3, x_4, x_5 \geq 0$ eingeführt, um die Ungleichungsrestriktionen auf Gleichungsrestriktionen zurückzuführen.

$$\begin{aligned}
(PLP) \qquad \max \quad & x_1 + 2x_2 & & & & & & \\
& & x_2 + x_3 & & & = 1 & & (1.26)\\
& x_1 + & x_2 & + x_4 & & = 2 & & (1.27)\\
& -x_1 + & x_2 & & + x_5 & = 1 & & (1.28)\\
& x_1, & x_2, & x_3, & x_4, & x_5 \geq 0 & & (1.29)
\end{aligned}$$

Das zu (PLP) gehörige duale Problem lautet

$$\min \quad y_1 + 2y_2 + y_3,\ \text{so dass}\ y_2 - y_3 \geq 1; y_1 + y_2 + y_3 \geq 2; y_1, y_2, y_3 \geq 0 \tag{DLP}$$

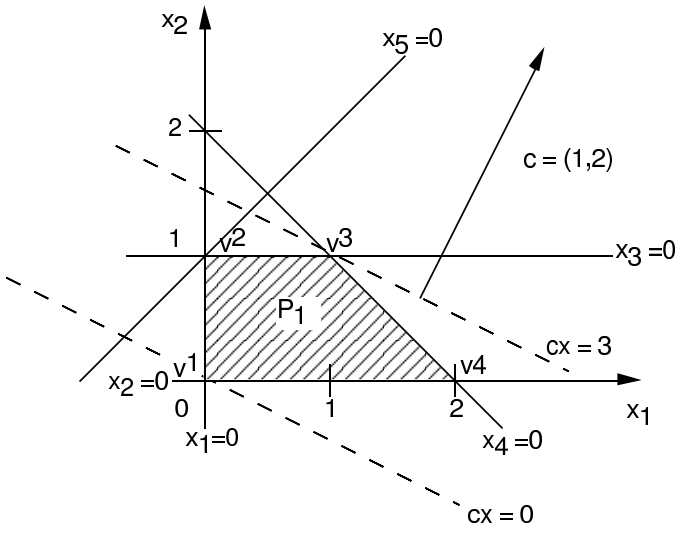


Abbildung 1.6: Das Problem (PLP0)

Wir wollen den Simplexalgorithmus anhand der Abbildung 1.6 erläutern, in der die Komponenten (x_1, x_2) dargestellt sind (x_3, x_4, x_5 sind durch (1.26) - (1.28) bestimmt). Man bemerke, dass falls eine Basis B gegeben ist, die zugehörige Basislösung in Abbildung 1.6 abgelesen werden kann: Sei zum Beispiel $B := \{1, 3, 5\}, N = \{2, 4\}$, so entspricht die zugehörige Basislösung dem Schnittpunkt der Geraden $\{x_2 = 0\}$ (x_1–Achse) und $\{x_4 = 0\}$, also dem Eckpunkt v^4.
Der Simplexalgorithmus startet mit einer zulässigen Basis. Wir können hier folgende Startbasis wählen:

Startbasis $B = \{3, 4, 5\}$ also $x^B = (x_3, x_4, x_5), x^N = (x_1, x_2)$ zugehörige Basislösung $x' = (0, 0, 1, 2, 1)$ (Lösung von $A^B x^B = b$), entspricht Eckpunkt v^1, zugehörige duale Lösung $y' = (0, 0, 0)$ (Lösung von $y^T A^B = c^B$), $c'^N = (c'_1, c'_2) = c^N - y'^T A^N = (1, 2)$, also ist y' nicht zulässig.

Basisaustausch: Wähle eintretende Kolonne = 2, da $c'_2 > 0$ (Kolonne 1 wäre auch möglich gewesen). Für die nächste Basislösung soll weiterhin $x_1 = 0$ (x_1 bleibt ausserhalb der Basis) gelten und x_2 möglichst gross sein. Es tritt diejenige Kolonne s aus der Basis aus, dessen zugehörige Restriktion $x_s \geq 0$ verhindert, dass x_2 nicht grösser gewählt werden kann. In unserem Beispiel (siehe Abbildung 1.6) kann x_2 wegen $x_5 \geq 0$ und wegen $x_3 \geq 0$ nicht grösser als 1 gewählt werden. Als austretende Kolonne wählen wir die Kolonne 5 (Kolonne 3 hätte auch gewählt werden können).

Basis $B = \{2, 3, 4\}$ also $x^B = (x_2, x_3, x_4), x^N = (x_1, x_5)$ zugehörige Basislösung $x' = (0, 1, 0, 1, 0)$ entspricht Eckpunkt v^2 zugehörige duale Lösung $y' = (0, 0, 2), c'^N = (c'_1, c'_5) = (3, -2)$, also ist y' nicht zulässig.

Basisaustausch: Wähle eintretende Kolonne = 1, da $c'_1 > 0$. Für die nächste Basislösung soll weiterhin $x_5 = 0$ gelten und $x_1 \geq 0$ möglichst gross werden. x_1 kann aber nicht grösser 0 gewählt werden, da sonst $x_3 \geq 0$ verletzt wäre. Die austretende Kolonne ist also die Kolonne 3.

Basis $B = \{1, 2, 4\}$ also $x^B = (x_1, x_2, x_4), x^N = (x_3, x_5)$ zugehörige Basislösung $x' = (0, 1, 0, 1, 0)$ (Basislösung hat sich nicht verändert) entspricht Eckpunkt v^2 zugehörige duale Lösung $y' = (3, 0, -1)$ (die duale Lösung hat sich verändert). $c'^N = (c'_3, c'_5) = (-3, 1)$, also ist y' nicht zulässig.

Basisaustausch: Wähle eintretende Kolonne = 5, da $c'_5 > 0$. Für die nächste Basislösung soll weiterhin $x_3 = 0$ gelten und x_5 möglichst gross werden. x_5 kann wegen $x_4 \geq 0$ nicht zu gross gewählt werden. Die austretende Kolonne ist also die Kolonne 4.

Basis $B = \{1, 2, 5\}$ also $x^B = (x_1, x_2, x_5), x^N = (x_3, x_4)$ zugehörige Basislösung $x' = (1, 1, 0, 0, 1)$ entspricht Eckpunkt v_3 zugehörige duale Lösung $y' = (1, 1, 0)$. ${c'}^N = (c'_3, c'_4) = (-1, -1) \leq 0$, also ist y' zulässig.
Es folgt, dass x' und y' optimal sind. Tatsächlich gilt für die Zielfunktionswerte $y'_1 + 2y'_2 + y'_3 = 3 = x'_1 + 2x'_2$. ◇

1.2 Einführung in die polyedrische Kombinatorik

Polyedrische Kombinatorik befasst sich mit dem Anwenden der Theorie der Linearen Systeme und Linearen Algebra auf diskrete (kombinatorische) Probleme. Um den Zusammenhang herzustellen zwischen der diskreten Optimierung, bei der, wie der Name sagt, die Variablen nur diskrete Werte annehmen, und der linearen Programmierung, wollen wir ein typisches Problem der diskreten Optimierung betrachten:

1.2.1 Das LOCO–Problem

Das LOCO–Problem ist eine Abkürzung für Linear Objective Combinatorial Optimization Problem und ist wie folgt definiert:

Gegeben: Eine Menge E von Elementen und für jedes Element $e \in E$ ein Wert c_e. Weiter ist eine Familie $\mathbf{F}$ von Teilmengen von E gegeben, die durch eine Eigenschaft Π charakterisiert ist, also $\mathbf{F} := \{F \subseteq E \mid F \text{ besitzt Eigenschaft } \Pi\}$. Der Wert eines Objektes $F \in \mathbf{F}$ ist durch die Summe der Werte seiner Elemente gegeben, d. h.

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

Gesucht: Das bezüglich dieser Bewertung beste Objekt F^0 von $\mathbf{F}$, also

$$c(F^0) = \max\{c(F) \mid F \in \mathbf{F}\}.$$

Zuerst wollen wir zwei Beispiele von LOCO–Problemen betrachten:

Beispiel 1.11 (Das Rucksackproblem) Gegeben ist ein Rucksack mit vorgegebenem Volumen V und eine Menge von Elementen E, die man gerne für die Ferien in den Rucksack packen möchte. Jedes Element $e \in E$ besitzt ein gewisses Volumen v_e und einen gewissen Wert c_e, der den Nutzen von e ausdrückt, falls e in die Ferien mitgenommen wird. Da das Rucksackvolumen beschränkt ist, muss man nun auswählen, welche Elemente man mitnimmt, so dass diese Elemente im Rucksack Platz haben und der Gesamtnutzen (= Summe der Nutzen der eingepackten Elemente) möglichst gross ist.
Dieses Problem kann als LOCO–Problem formuliert werden: Sei

(i) E: Elemente, die man einpacken möchte,

(ii) c_e: Nutzen des Elementes $e \in E$, falls es eingepackt wird,

(iii) $\mathbf{F}$: $\{F \subseteq E \mid$ Gesamtvol. von F ist nicht grösser $V\} = \left\{F \subseteq E \mid \sum_{e \in F} v_e \leq F\right\}$.

(iv) $F \in \mathbf{F}$ ist eine Menge von Elementen, die gleichzeitig im Rucksack Platz haben.

Das beste Objekt F^0 ist eine Menge von Elementen, die gleichzeitig im Rucksack Platz hat und einen grösstmöglichen Nutzen hat. ◇

Beispiel 1.12 (Das Zuordnungsproblem) Zur Bearbeitung der Jobs $J_1, \ldots, J_n$ stehen die Maschinen $M_1, \ldots, M_n$ zur Verfügung, aber nicht alle Maschinen sind zur Bearbeitung eines gegebenen Jobs gleich gut geeignet. Die Bearbeitungszeit von Job j auf Maschine i beträgt d_{ji} Zeiteinheiten. Die Jobs sollen den Maschinen so zugeordnet werden, dass jede Maschine genau einen Job bearbeitet und dass die Gesamtbearbeitungszeit minimal ist.
Auch dieses Problem kann als LOCO–Problem formuliert werden: Sei

(i) $E := \{(J_j, M_i) \mid i, j = 1, \ldots, n\}$ die Menge aller Möglichkeiten, einen Job J_j auf einer Maschine M_i auszuführen.

(ii) $c_e = c_{(J_j, M_i)} = -d_{ji}$: Bearbeitungszeit des Jobs J_j auf Maschine M_i.

(iii) $\mathbf{F} := \{F = \{(J_1, M_{b(1)}), \ldots, (J_n, M_{b(n)})\} \mid \{b(1), \ldots, b(n)\} = \{1, \ldots, n\}\}$.
F ist eine Zuordnung, die jedem Job J_j eine Maschine $b_{(j)}$ zuordnet, so dass keine zwei Jobs derselben Maschine zugeordnet sind.

Das beste Objekt F^0 ist eine Zuordnung mit maximalem Gesamtwert, d. h. mit minimaler Gesamtbearbeitungszeit. ◇

1.2.2 LOCO–Problem und Lineare Programmierung

Sei nun ein LOCO–Problem gegeben, also eine Menge E, Gewichte $c_e, e \in E$ und eine Familie $\mathbf{F} \subseteq 2^E$. Wir wollen zeigen, dass dieses Problem auf ein Problem der linearen Programmierung zurückgeführt werden kann.
Dazu ordnen wir jedem Objekt $F \in \mathbf{F}$ einen *Inzidenzvektor* $x^F \in \mathbb{R}^E$ zu, der für jedes Element $e \in E$ eine Komponente besitzt, die 1 ist, falls e zu F gehört, und sonst 0 ist:

$$x_e^F = \begin{cases} 1, & e \in F \\ 0, & sonst \end{cases}$$

Zur Familie $\mathbf{F}$ gehört dementsprechend eine Familie von Inzidenzvektoren $X^F := \{x^F \mid x^F \text{ Inzidenzvektor von } F \in \mathbf{F}\}$.
Der Wert von $F \in \mathbf{F}$ ist gleich

$$c(F) = \sum_{e \in F} c_e \underset{\substack{x_e^F = 1 \text{ für} \\ e \in F}}{=} \sum_{e \in F} c_e x_e^F \underset{\substack{x_e^F = 0 \text{ für} \\ e \notin F}}{=} \sum_{e \in E} c_e x_e^F = c^T x^F,$$

falls $c \in \mathbb{R}^E$ der Vektor mit Komponenten $c_e, e \in E$ ist.
Das LOCO–Problem:

(P1)	*Gegeben:*	Grundmenge E, Werte $c_e, e \in E$ und eine Menge $\mathbf{F} \subseteq 2^E$.
	Gesucht:	$F^0 \in \mathbf{F}$ maximalen Werts, also $c(F^0) := \sum_{e \in F^0} c_e = \max\{c(F) \mid F \in \mathbf{F}\}$.

ist somit äquivalent zu

(P2)	*Gegeben:*	Grundmenge E, Wert $c \in \mathbb{R}^E$ und $X^{\mathbf{F}} := \{x^F \mid x^F \text{ Inzidenzvektor von } F \in \mathbf{F}\}$.
	Gesucht:	$x^0 \in X^{\mathbf{F}}$ maximal bezüglich $c^T x$, also $c^T x^0 := \sum_{e \in E} c_e x_e^0 = \max\{c^T x \mid x \in X^{\mathbf{F}}\}$.

Die Menge $X^{\mathbf{F}}$ ist eine Menge von Punkten in $\mathbb{R}^E$. Sei $P := CONV(X^{\mathbf{F}})$, die konvexe Hülle dieser Punkte. P ist ein Polyeder (Satz 1.2), dessen Eckpunkte gerade die Vektoren aus $X^{\mathbf{F}}$, also die Inzidenzvektoren von Mengen $F \in \mathbf{F}$ sind. Das Problem

(P3)	*Gegeben:*	Grundmenge $E, c \in \mathbb{R}^E, X^{\mathbf{F}}$ und $P := CONV(X^{\mathbf{F}})$.
	Gesucht:	maximiere $c^T x := \sum_{e \in E} c_e x_e$, so dass $x \in P$.

ist in dem Sinne äquivalent zu (P2), dass (P3) immer eine optimale Lösung besitzt, die ein Eckpunkt von P, also ein Inzidenzvektor $x^F \in X^{\mathbf{F}}$, ist.
Da P ein Polyeder ist, existiert eine Matrix A und ein Vektor b, so dass $P := \{x \in \mathbb{R}^E \mid Ax \leq b\}$. Das Problem (P3) lässt sich also wie folgt schreiben:

(P4)	*Gegeben:*	Grundmenge $E, c \in \mathbb{R}^E, CONV(X^{\mathbf{F}}) = \{x \in \mathbb{R}^E \mid Ax \leq b\}$.
	Gesucht:	maximiere $c^T x := \sum_{e \in E} c_e x_e$, so dass $Ax \leq b$.

Somit haben wir das diskrete Problem (P1) auf ein Problem der linearen Programmierung (P4) zurückgeführt. Die Beschreibung $Ax \leq b$ von $P := CONV(X^{\mathbf{F}})$ wird dann *polyedrische Beschreibung* von $\mathbf{F}$ genannt.
Für jede Familie $\mathbf{F} \subseteq 2^E$existiert eine polyedrische Beschreibung $Ax \leq b$, sie ist aber häufig sehr schwer zu finden. Auch kann es vorkommen, dass zur Beschreibung von $\mathbf{F}$ sehr viele Restriktionen notwendig sind.
Falls es aber gelingt, die Beschreibung komplett oder teilweise zu finden, kann sie zur *Entwicklung von Algorithmen* sehr nützlich sein (siehe z. B. Ungarischer Algorithmus für das Matching–Problem, Schnittebenenverfahren).
Ein weiterer Vorteil einer LP–Formulierung ist durch die Dualitätstheorie gegeben: Jede duale Lösung liefert eine *obere Schranke* für den Optimalwert des LOCO–Problems. Auch wenn nur ein Teil der Beschreibung $A'x \leq b'$ bekannt ist, also $\{x \in \mathbb{R}^E \mid Ax \leq b\} \subseteq \{x \in \mathbb{R}^E \mid A'x \leq b'\}$, so liefert jede duale Lösung y' von

$$\min y^T b', \text{ so dass } y^T A' = c, y \geq 0$$

eine obere Schranke für den Optimalwert des LOCO–Problems, da

$$\max\{c^T x \mid Ax \leq b\} \leq \max\{c^T x \mid A'x \leq b'\} = \min\{y^T b \mid y^T A' = c, y \geq 0\} \leq y'b.$$

Obere Schranken sind insbesondere zur Beurteilung von Heuristischen Verfahren und in Branch and Bound–Verfahren sehr nützlich.
Ist eine polyedrische Beschreibung des Problems bekannt, so liefern der starke Dualitätssatz (Satz 1.7) und die Komplementärschlupfbedingungen (Satz 1.8) *Optimalitätskriterien*. Diese sind als Stopkriterien für Algorithmen sehr hilfreich.
Der starke Dualitätssatz besagt, dass

$$\max\{c^T x \mid Ax \leq b\} = \min\{y^T b \mid y^T A = c, y \geq 0\}.$$

Die linke Seite besitzt eine kombinatorische Interpretation, sie ist das maximale Gewicht eines Elementes $F \in \mathbf{F}$. Falls auch die rechte Seite eine kombinatorische Interpretation besitzt, so liefert obige Gleichung einen kombinatorischen min $-$ max Satz (siehe z.B. max Flow - min Cut (Satz 7.9) oder Satz von König–Egervary (Satz 4.7)).

1.3 Einige Begriffe und Definitionen der Graphentheorie

1.3.1 Ungerichtete Graphen

Ein *endlicher ungerichteter Graph* $G = (V, E, \Psi)$ ist definiert durch

(i) eine endliche Menge V von Elementen v, genannt Knoten (vertices);

(ii) eine endliche Menge E von Elementen e, genannt Kanten (edges);

(iii) eine *Inzidenz–Abbildung* $\Psi : E \to V \times V$, die jeder Kante $e \in E$ ein *ungeordnetes* Knotenpaar (v, w) zuordnet ($(v, w) \equiv (w, v)$).

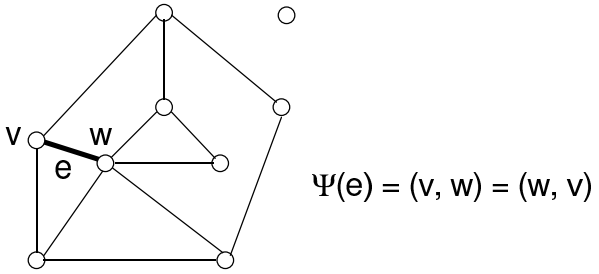


Abbildung 1.7: Ungerichteter Graph

Oft wird der Graph nur mit G oder mit $G = (V, E)$ bezeichnet und die Abbildung Ψ nicht explizite erwähnt.
Falls $e \in E$ und $v, w \in V$, so dass $\Psi(e) = (v, w)$, sind v und w die *Endknoten* oder *Extremitäten* von e, e ist *mit* v (und mit w) *inzident*, v und w sind durch e *miteinander verbunden*, v und w sind *benachbart*. Falls die beiden Endknoten von e identisch sind, heisst e eine *Schlinge*. Zwei Kanten e und e' mit $\Psi(e) = \Psi(e') = (v, w)$ heissen *Parallelkanten*. Ein *schlichter* Graph $G = (V, E)$ ist ein Graph ohne Schlingen und Parallelkanten. Oft wird ein Graph als schlicht angenommen und für die Kante e mit $\Psi(e) = (v, w)$ kurzerhand auch $e = (v, w)$ geschrieben, da jede Kante eines schlichten Graphen durch ihre Endknoten eindeutig identifiziert ist. Ein Graph, der Parallelkanten besitzt, heisst auch *Multigraph*.

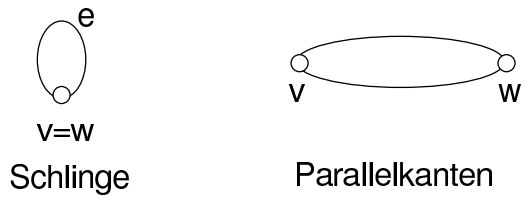


Abbildung 1.8: Schlinge und Parallelkanten

Die Anzahl Kanten, die mit einem Knoten v inzident sind (mit Schlingen doppelt gezählt), ist der *Grad* $d(v)$. Ein Knoten v ist *isoliert*, falls $d(v) = 0$, *hängend*, falls $d(v) = 1$.

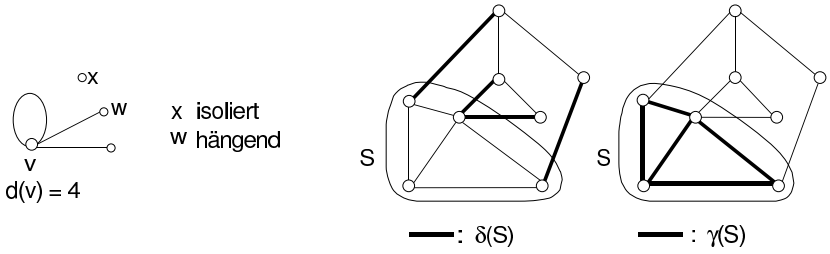


Abbildung 1.9: Grad, $\delta(S), \gamma(S)$

Für eine Knotenmenge $S \subseteq V$ sind $\delta(S) \subseteq E$, die Menge der aus S herausgehenden Kanten, und $\gamma(S) \subseteq E$, die Menge der Kanten in S, wie folgt definiert:

$$\begin{aligned} \delta(S) &:= \{e \in E \mid \Psi(e) = (v, w) \text{ und } v \in S, w \notin S\} \\ \gamma(S) &:= \{e \in E \mid \Psi(e) = (v, w) \text{ und } v \in S, w \in S\}. \end{aligned} \tag{1.30}$$

Ist $S = \{v\}, v \in V$, so wird oft statt $\delta(\{v\})$ auch $\delta(v)$ gechrieben.
Ein *Untergraph* $G' = (V', E', \Psi')$ von $G = (V, E, \Psi)$ ist ein Graph, so dass: $V' \subseteq V, E' \subseteq E, \Psi'(e) = \Psi(e)$ für alle $e \in E'$ und, falls $e \in E'$ und $\Psi(e) = (v, w)$, dann $v, w \in V'$. Ein von der Knotenteilmenge V' *induzierter Untergraph* von G ist der Untergraph $G' = (V', E', \Psi')$ mit $E' = \{e \in E \mid$ beide Extremitäten von e sind in $V'\}$.

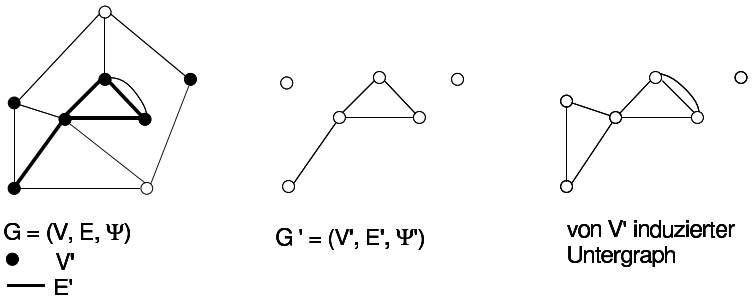


Abbildung 1.10: Untergraphen

Eine *Kantenfolge* ist eine Folge $(e_1, e_2, \ldots, e_n)$ von (nicht notwendigerweise verschiedenen) Kanten, so dass eine Folge von $n+1$ (nicht notwendigerweise verschiedenen) Knoten $(v_0, v_1, \ldots, v_n)$ existiert mit $\Psi(e_i) = (v_{i-1}, v_i), i = 1, 2, \ldots, n$. Die Kantenfolge ist *geschlossen*, falls $v_0 = v_n$, und *offen*, sonst. Ein *Weg* ist eine Kantenfolge, in der alle Knoten $v_1, \ldots, v_n$ und alle Kanten verschieden sind. Ein *Kreis* ist ein Weg mit $v_0 = v_n$. Ist G schlicht, so wird ein Weg auch häufig durch seine Knotenfolge $(v_0, v_1, \ldots, v_n)$ beschrieben.

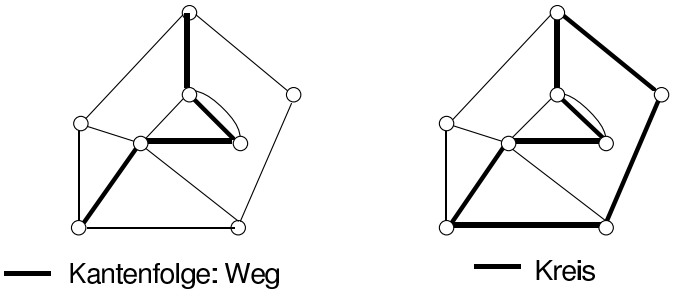


Abbildung 1.11: Weg und Kreis

Ein Graph heisst zusammenhängend, falls jedes Knotenpaar durch eine Kantenfolge (ein Weg) verbunden ist. Eine *Komponente* eines Graphen G ist ein (Knoten–) mengentheoretisch maximaler zusammenhängender induzierter Untergraph von G.

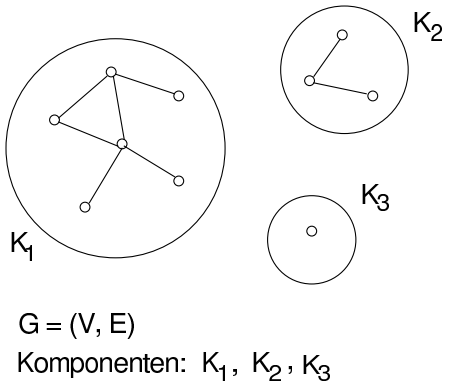


Abbildung 1.12: Komponenten

Ein *Baum* ist ein zusammenhängender Graph ohne Kreise. Ein *Wald* ist ein Graph ohne Kreise (dessen Komponenten Bäume sind). Ein *Gerüst* eines Graphen $G = (V, E)$ ist ein Untergraph $T = (V, E')$ von G, der ein Baum ist (beachte: jeder Knoten von G gehört zu T).

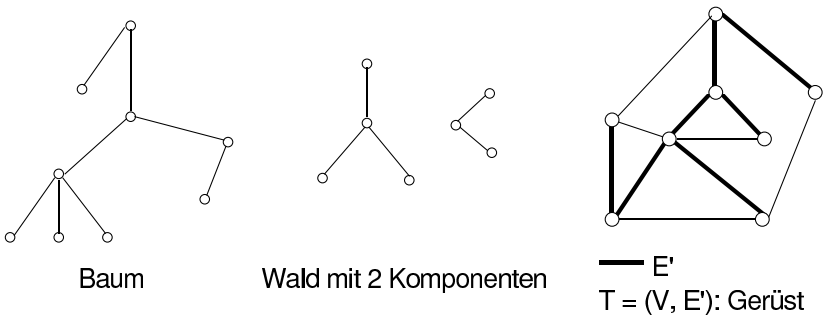


Abbildung 1.13: Baum, Wald, Gerüst

Ein *bipartiter* Graph $G = (S \cup T, E)$ ist ein Graph, dessen Knotenmenge sich in zwei disjunkte Mengen S und T aufteilen lässt, so dass jede Kante von E jeweils einen Endknoten in S und den anderen in T hat.

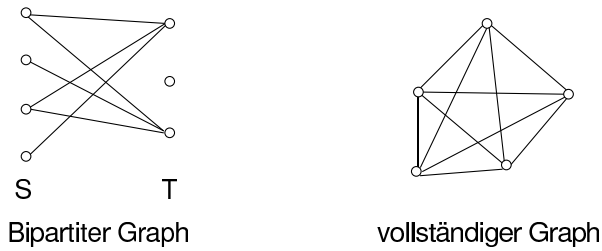


Abbildung 1.14: Bipartiter und vollständiger Graph

Ein *vollständiger* Graph ist ein schlichter Graph mit je einer Kante zwischen *jedem* Knotenpaar.
Eine *Clique* $G' = (V', E', \Psi')$ mit $V' \subseteq V$ ist ein von V' induzierter Untergraph von G, der vollständig ist. Eine *unabhängige Menge* $G' = (V', E', \Psi')$ ist ein von V' induzierter Untergraph von G, dessen Kantenmenge $E' = \emptyset$.

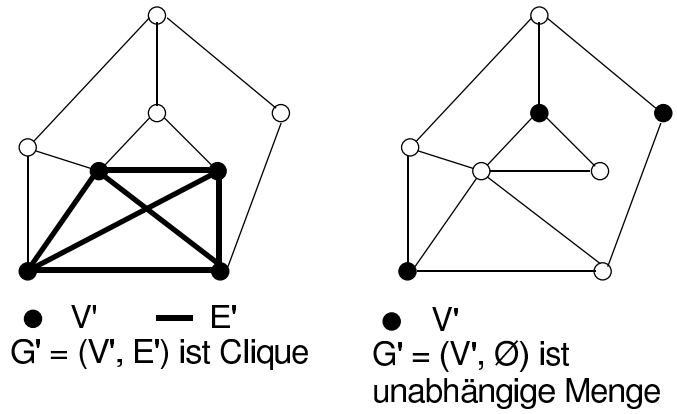


Abbildung 1.15: Clique, unabhängige Menge

1.3.2 Gerichtete Graphen

Ein *endlicher gerichteter* Graph $G = (V, E, \Psi)$ ist definiert durch eine endliche Menge V von Knoten, eine endliche Menge E von *Bögen* und eine *Inzidenz–Abbildung* $\Psi : E \to V \times V$, die jedem Bogen $e \in E$ ein *geordnetes* Knotenpaar (v, w) zuordnet.

Oft wird anstatt $G = (V, E, \Psi)$ nur die Bezeichnung G oder $G = (V, E)$ gebraucht.

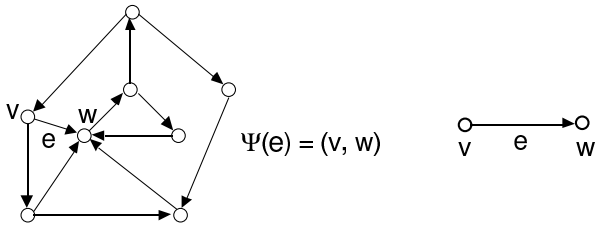


Abbildung 1.16: Gerichteter Graph

Falls $e \in E$ und $v, w \in V$, so dass $\Psi(e) = (v, w)$, sind v und w die *Extremitäten*, v der *Startknoten* oder der *tail* $t(e)$ und w der *Endknoten* oder der *head* $h(e)$. Der Bogen e ist *mit v und w inzident, e geht von v nach w, e geht aus v heraus, e geht in w hinein, e ist von v nach w gerichtet.* v ist ein *Vorgänger* von w, w ein *Nachfolger* von v. Falls beide Extremitäten von e identisch sind, ist e eine *Schlinge*. Zwei Bögen e und e' mit gleichen Start– und Endknoten heissen *Parallelbögen*. Ein *schlichter* (gerichteter) Graph $G = (V, E)$ ist ein Graph ohne Schlingen und Parallelbögen, ein *Multigraph* ein Graph mit Parallelbögen.

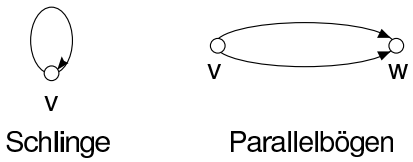


Abbildung 1.17: Schlinge und Parallelbögen

Jedem ungerichteten Graphen kann auch ein gerichteter Graph zugeordnet werden, indem jede Kante durch zwei Bögen entgegengesetzter Richtung ersetzt wird.

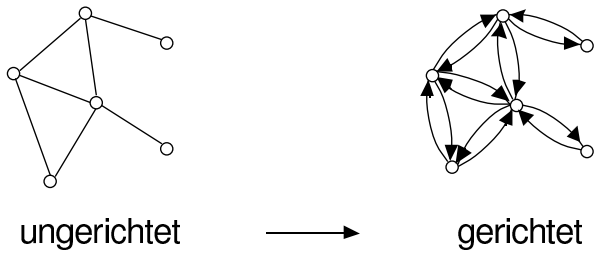


Abbildung 1.18: Gerichteter Graph eines ungerichteten Graphen

Die Anzahl Bögen mit Startknoten v ist der *Ausgangsgrad* $d^+(v)$ von v, die Anzahl Bögen mit Endknoten v der *Eingangsgrad* $d^-(v)$. $d(v) = d^+(v) + d^-(v)$ ist der Grad von v. Die Definitionen von *isolierten* und *hängenden* Knoten und von *Untergraphen* und *induzierten Untergraphen* sind mit den für ungerichtete Graphen gegebenen Definitionen identisch.

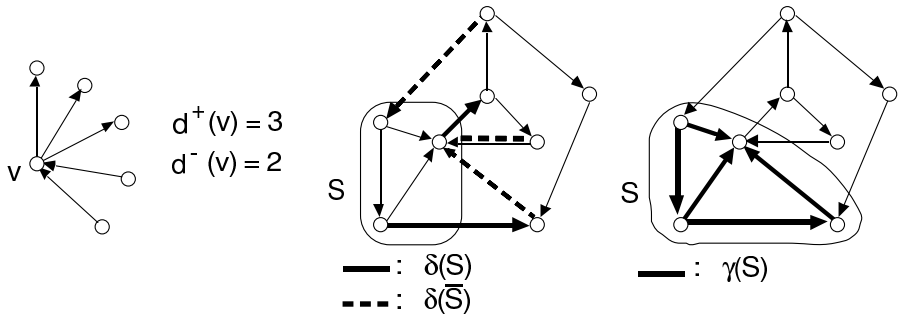


Abbildung 1.19: Grad, $\delta(S), \gamma(S)$

Für eine Knotenmenge $S \subseteq V$ sind $\delta(S) \subseteq E$, die Menge der aus S herausgehenden Bögen, und $\gamma(S) \subseteq E$, die Menge der Bögen in S, wie folgt definiert:

$$\begin{aligned} \delta(S) &:= \{e \in E \mid \Psi(e) = (v,w) \text{ und } v \in S, w \notin S\} \\ \gamma(S) &:= \{e \in E \mid \Psi(e) = (v,w) \text{ und } v \in S, w \in S\}. \end{aligned} \tag{1.31}$$

Da $\overline{S} := V \setminus S$, ist dann

$$\delta(\overline{S}) := \{e \in E \mid \Psi(e) = (v,w) \text{ und } v \notin S, w \in S\}$$

die Menge der nach S hineingehenden Bögen. Ist $S := \{v\}, v \in V$, so wird oft $\delta(v)$ für $\delta(\{v\})$ und $\delta(\overline{v})$ für $\delta(\overline{\{v\}})$ geschrieben.
Für die Definitionen von *Bogenfolge, geschlossene* und *offene* Bogenfolge, gerichteter Weg und *Zyklus*, ersetze „Kante“ durch „Bogen“ in den Definitionen für Kantenfolge, geschlossene und offene Kantenfolge, Weg und Kreis.

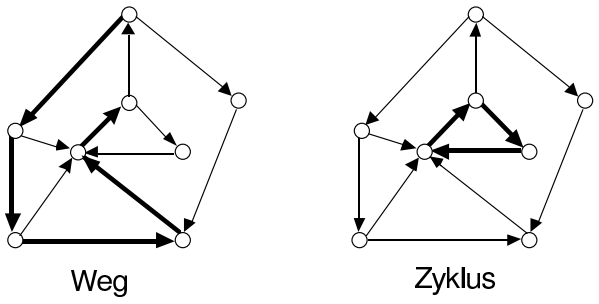


Abbildung 1.20: Weg und Zyklus

Ein Graph ohne Zyklen heisst *azyklisch*. Ein Graph heisst stark *zusammenhängend*, falls jedes (gerichtete) Knotenpaar durch eine Bogenfolge (einen Weg) verbunden ist, und *schwach zusammenhängend*, falls jedes Knotenpaar durch eine Kantenfolge verbunden ist, wenn man von den Richtungen auf den Bögen absieht.
Eine *stark (bzw. schwach) zusammenhängende Komponente* eines Graphen G ist ein (Knoten–)mengentheoretisch maximaler stark (schwach) zusammenhängender induzierter Untergraph von G.

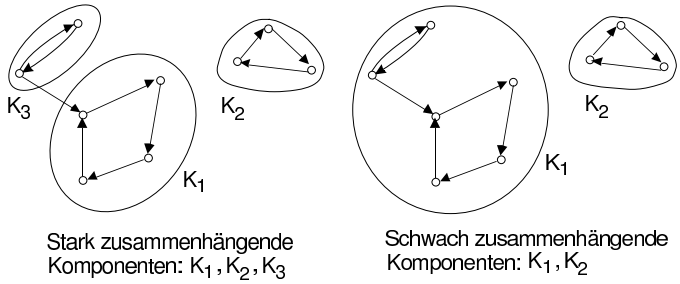


Abbildung 1.21: Komponenten

Ein *Euler'scher Zyklus* eines Graphen G ist eine geschlossene Bogenfolge, die durch jeden Bogen von G genau einmal verläuft.

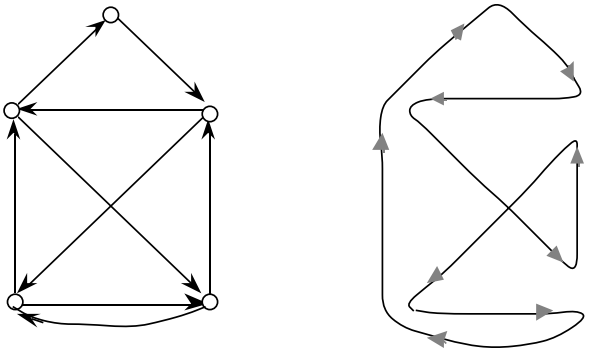


Abbildung 1.22: Euler'scher Zyklus

Eine *Arboreszenz* (oder *Wurzelbaum*) ist ein zusammenhängender azyklischer Graph, so dass in jeden Knoten höchstens ein Bogen hineingeht. Der (einzige) Knoten der Arboreszenz, in den kein Bogen hineingeht, heisst *Wurzel*. Ein *Branching* ist ein azyklischer Graph, so dass in jeden Knoten höchstens ein Bogen hineingeht. Eine *aufspannende Arboreszenz* eines Graphen $G = (V, E)$ ist ein Untergraph $B = (V, E')$ von G, der eine Arboreszenz ist.

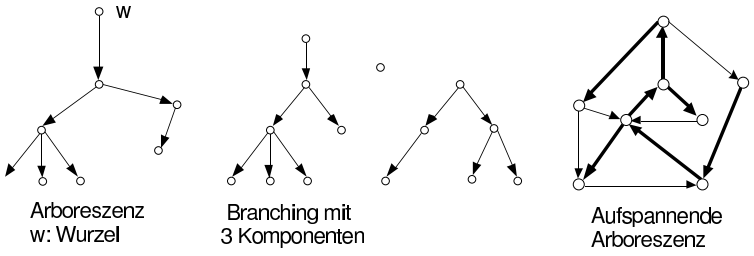


Abbildung 1.23: Arboreszenz und Branching

Ein vollständiger Graph ist ein schlichter Graph mit je einem Bogen zwischen jedem (geordneten) Knotenpaar.

1.4 Topologisches Sortieren

Definition 1.13 *Sei $G = (V, E)$ ein gerichteter, azyklischer Graph. Eine Numerierung der Knoten $n(v) \in \mathbf{N}, v \in V$ heisst topologische Sortierung von G, falls gilt: $n(v) \leq \mid V \mid$ für alle $v \in V, n(v) \neq n(w)$ für $v \neq w$ und $e = (v, w) \in E$ impliziert $n(v) < n(w)$.*

Man bemerke, dass für einen Graphen mit Zyklen keine topologische Sortierung existiert. Betrachte

Problem der Topologischen Sortierung:

Gegeben: Gerichteter Graph $G = (V, E)$.

Gesucht: Bestimme, ob G azyklisch ist und falls ja, finde eine topologische Sortierung.

Ein Knoten $v \in V$ heisst *Quelle* von $G = (V, E)$, falls kein Bogen nach v zeigt, d. h. $d^-(v) = 0$. Um eine topologische Sortierung zu finden, sucht man jeweils eine Quelle von G und gibt ihr die kleinste noch nicht vergebene Nummer. Dann entfernt man diesen Knoten aus G und beginnt von vorne:
Algorithmus zum Finden einer topologischen Sortierung:

Satz 1.14 (Gültigkeit von TopologSort1) *Der Algorithmus* TOPOLOGSORT1 *berechnet eine topologische Sortierung oder einen Zyklus eines gerichteten Graphen G.*

Beweis. Wir zeigen zuerst, dass jeder azyklische Graph $G' = (V', E')$ eine Quelle besitzt:
Sei v_1 irgend ein Knoten von V'. Entweder ist v_1 eine Quelle oder es existiert v_2 mit $(v_2, v_1) \in E'$. Falls v_2 immer noch keine Quelle ist, so existiert $(v_3, v_2) \in E'$. Auf diese Weise erhalten wir eine Folge von Knoten. Da G azyklisch ist, sind die Knoten dieser Folge alle verschieden und somit ist die Folge endlich. Der letzte Knoten dieser Folge ist dann eine Quelle.

Falls am Schluss von Topsort1 $G' \neq (\emptyset, \emptyset)$, so besitzt G' keine Quelle und somit mindestens einen Zyklus. Da G' ein Untergraph von G ist, besitzt in diesem Fall auch G mindestens einen Zyklus.
Es bleibt zu zeigen, dass falls am Schluss von Topsort1 $G' = (\emptyset, \emptyset)$, so ist G azyklisch und $n(v), v \in V$ ist eine topologische Sortierung:

Algorithmus 1.2 TopologSort 1

(*Sei $G = (V, E)$ ein gerichteter Graph.*)
(* Der Algorithmus bestimmt, ob G azyklisch ist und falls ja, findet er eine topologische Sortierung.*)
begin (*TopologSort1*)
  $i := 0; G' = (V', E') := G = (V, E)$
  while (G' besitzt Quelle) **do**
    finde Quelle $v \in V'$
    $i := i + 1$
    $n(v) := i$
    $V'' := V' \setminus \{v\}$
    Sei $G' = (V', E')$ der von V'' induzierte Untergraph
  end (* while *)
  if $G' = (\emptyset, \emptyset)$ **then**
    $n(v), v \in V$, ist eine topologische Sortierung von G
  else
    G besitzt Zyklen
end (*TopologSort1*)

Sei $e = (v, w) \in E$. Als w seine Nummer bekam galt: $G' = (V', E')$ mit $w \in V'$ und $v \notin V'$, da $d^-(w) = 0$ in G'. Da $v \notin V'$, hatte zu diesem Zeitpunkt v bereits eine Nummer erhalten, also $n(v) < n(w)$. Somit ist $n(v), v \in V$ eine topologische Sortierung und G azyklisch. ♣

Will man den obigen Algorithmus implementieren, so muss G' nicht immer explizit konstruiert werden. Es genügt jeweils, den Eingangsgrad jedes Knotens von G' und alle Quellen von G' zu kennen. Der Algorithmus lautet dann:

Algorithmus zum Finden einer topologischen Sortierung:

Algorithmus 1.3 TopologSort 2

```
(* Sei G = (V, E) ein gerichteter Graph. *)
(* Der Algorithmus bestimmt, ob G azyklisch ist und falls ja, findet er eine
topologische Sortierung. *)
begin (* TopologSort2 *)
    berechne d⁻(v) für alle v ∈ V
    quellen := {v ∈ V | d⁻(v) = 0}
    i := 0;
    while quellen ≠ ∅ do
        wähle v ∈ quellen
        quellen := quellen \{v}
        i := i + 1
        n(v) := i
        for all w ∈ {w' | (v, w') ∈ E} do
            d⁻(w) := d⁻(w) − 1
            if d⁻(w) = 0 then quellen := quellen ∪{w}
        end (* for *)
    end (* while *)
    if i =| V| then
        n(v), v ∈ V, ist eine topologische Sortierung von G
    else
        G besitzt Zyklen
end (* TopologSort2 *)
```

Der Aufwand für TopologSort2 ist $O(E)$.

1.5 Kürzeste Wege

In diesem Abschnitt wird angenommen, dass $G = (V, E)$ ein schlichter, gerichteter Graph ist und dass für jeden Bogen $e = (i, j)$ eine Bogenlänge, die mit $c_e \equiv c_{ij}$ bezeichnet wird, gegeben ist. Die Länge $L(P)$ eines Weges P mit Bogenfolge $P = (e_1, \ldots, e_k)$ ist die Summe der Längen seiner Bögen, also

$$L(P) := \sum_{i=1}^{k} c_{e_i}$$

Für ein gegebenes Knotenpaar a, b werden wir uns für einen sogenannten *kürzesten Wege* von a nach b interessieren, also für einen Weg P mit Bogenfolge $P = (e_1, \ldots, e_k)$ und $t(e_1) = a, h(e_k) = b$, für den

$$L(P) := \sum_{i=1}^{k} c_{e_i} = \min\{L(P') \mid P' \text{ Weg von } a \text{ nach } b\}.$$

Die Länge eines solchen kürzesten Weges werden wir mit u_{ab} bezeichnen. Wir treffen die Konvention, dass $u_{ab} = \infty$, falls kein Weg von a nach b führt und dass für $v \in V$ der Knoten $\{v\}$ auch als Weg bezeichnet wird (Weg mit leerer Bogenfolge), dessen Länge $L(\{v\}) := 0$ ist. Somit ist $u_{vv} \leq 0$. Falls $u_{vv} < 0$, so besitzt der Graph einen Zyklus negativer Länge (ein Weg von v nach v, der mindestens einen Bogen besitzt, ist ein Zyklus).

1.5.1 Kürzeste Wege in einem azyklischen Graphen

Gegeben: $G = (V, E)$ gerichtet, Bogenlängen c_e für $e \in E$, eine Quelle $s \in V$.

Gesucht: Bestimme, ob G azyklisch ist und falls ja, bestimme für alle Knoten $v \in V$ einen kürzesten Weg von s nach v und dessen Länge (falls einer existiert).

Um die kürzesten Wege in einem azyklischen Graphen zu berechnen, werden wir folgende Beziehung benutzen:

Lemma 1.15 *Falls $G = (V, E)$ keine Zyklen negativer Länge besitzt, so gilt für $a, b \in V, a \neq b$*

$$u_{ab} = \min\{u_{aw} + c_{wb} \mid w \in V \text{ mit } (w, b) \in E\} \tag{1.32}$$

Beweis. Wir zeigen zuerst, dass $u_{ab} \geq \min\{u_{aw}+c_{wb} \mid w \in V \text{ mit } (w,b) \in E\}$: Sei $u_{ab} < \infty$ (sonst ist nichts zu beweisen) und sei $P = (a = v_1, \ldots, v_k = b)$ die Knotenfolge eines kürzesten Weges von a nach b. Mit der Konvention $\sum_{i=1}^{0} c_{v_i v_{i+1}} := 0$ gilt:

$$\begin{aligned} u_{ab} &= \sum_{i=1}^{k-2} c_{v_i v_{i+1}} + c_{v_{k-1} v_k} \\ &\geq u_{a v_{k-1}} + c_{v_{k-1} v_k} \\ &\geq \min\{u_{aw} + c_{wb} \mid w \in V \text{ mit } (w,b) \in E\} \end{aligned}$$

da $P' := (a = v_1, \ldots, v_{k-1})$ ein Weg von a nach v_{k-1} ist.
Es bleibt zu zeigen, dass $u_{ab} \leq \min\{u_{aw} + c_{wb} \mid w \in V \text{ mit } (w,b) \in E\} =: u_{av} + c_{vb}$, falls $u_{av} + c_{vb} < \infty$: Sei $P = (a = v_1, \ldots, v_k = v)$ die Knotenfolge eines kürzesten Weges von a nach v. Falls $b \neq v_i, i = 2, \ldots, k-1$, dann ist $P' := (a = v_1, \ldots, v_k = v, b)$ ein Weg. Folglich ist $u_{av} + c_{vb} \geq u_{ab}$. Sonst ist $v_i = b$ für genau ein i. $P'' := (a = v_1, \ldots, v_i = b)$ ist ein Weg und $Q := (b = v_i, \ldots, v_k = v, b)$ ein Zyklus und es gilt (da $L(Q) \geq 0$):

$$u_{ab} \leq L(P'') \leq L(P'') + L(Q) = u_{av} + c_{vb} \tag{1.33}$$

♣

Falls u_{sw} für alle Vorgänger $\{w \in V \mid (w,v) \in E\}$ eines Knotens $v \in V$ bekannt ist, kann u_{sv} mittels (1.32) berechnet werden. Falls G azyklisch ist, existiert eine topologische Sortierung $n(v), v \in V$. Berechnet man die u_{sv} in dieser Reihenfolge, so ist jeweils garantiert, dass die kürzesten Wege für die Vorgänger bereits berechnet wurden und somit (1.32) verwendet werden kann. Der Algorithmus leitet sich dann vom Algorithmus zur Berechnung einer topologischen Sortierung ab:
Man bemerke, dass die kürzeste Wege mit Hilfe der z–Werte bestimmt werden können, denn gemäss Beweis von Lemma 1.15 kann ein kürzester Weg von s nach v zusammengesetzt werden aus dem kürzesten Weg von s nach z_{sv} und dem Bogen (z_{sv}, v) (da G azyklisch ist, ist dies immer ein Weg).
Der Aufwand von diesem Algorithmus ist $O(E)$.

1.5.2 Kürzeste Wege bei nicht negativen Bogenlängen

Gegeben: $G = (V, E)$ gerichtet, Bogenlängen $c_{ij} \geq 0$ für $(i,j) \in E$, eine Quelle $s \in V$.

Algorithmus 1.4 Kürzeste Wege – azyklisch

(*Sei $G = (V, E)$ ein gerichteter Graph, $s \in V$ eine Quelle von G. Der Algorithmus bestimmt, ob G azyklisch ist und falls ja, berechnet er für alle $v \in V$ die Länge u_{sv} eines kürzesten Weges von s nach v, sowie den letzten von diesem Weg benützten Bogen (z_{sv}, v). $z_{sv} = \emptyset$ bedeutet, dass der letzte Bogen noch nicht bestimmt ist oder kein Weg von s nach v existiert*)

begin (* Kürzeste Wege - azyklisch *)
  berechne $d^-(v)$ für alle $v \in V$
  quellen := $\{v \in V \mid d^-(v) = 0\}$
  $i := 0$, $u_{ss} := 0, z_{ss} := \emptyset$ und $u_{sv} := \infty; z_{sv} := \emptyset$ für alle $v \in V \setminus \{s\}$
  while quellen $\neq \emptyset$ **do**
    wähle $v \in$ quellen und lösche v aus quellen
    $i := i + 1$ und $n(v) := i$;
    if $\{w \in V \mid (w, v) \in E\} \neq \emptyset$ **then**
      sei $x \in V$ mit $u_{sx} + c_{xv} = \min\{u_{sw} + c_{wv} \mid (w, v) \in E\}$
      $u_{sv} := u_{sx} + c_{xv}$ und $z_{sv} := x$
    end (* if *)
    for all $w \in \{w' \mid (v, w') \in E\}$ **do**
      $d^-(w) := d^-(w) - 1$
      if $d^-(w) = 0$ **then** quellen := quellen $\cup\{w\}$
    end (* for *)
  end (* while *)
  if $i =\mid V \mid$ **then**
    - G ist azyklisch und $n(v), v \in V$, ist eine topologische Sortierung von G
    - für alle $v \in V$ gilt: Falls $u_{sv} < \infty$, so existiert ein kürzester Weg von s nach v
      der Länge u_{sv}, dessen letzter Bogen (z_{sv}, v) ist. Sonst existiert kein Weg
      von s nach v.
  else
    G besitzt Zyklen
end (* Kürzeste Wege - azyklisch *)

Gesucht: Für alle Knoten $v \in V$ einen kürzesten Weg von s nach v und dessen Länge (falls einer existiert).

Um dieses Problem zu lösen, wollen wir den sogenannten Dijkstra–Algorithmus betrachten.
Die algorithmische Idee kann wie folgt zusammengefasst werden: Die Knotenmenge V ist jeweils in eine Knotenmenge $T \subseteq V$ und $V \setminus T$ partitioniert, wobei $s \in T$ ist. Jeder Knoten $v \in V$ besitzt zwei Marken u_{sv} und z_{sv}, die folgende Bedeutung haben: u_{sv} ist die Länge eines kürzesten Weges von s nach v, gegeben dass nur Zwischenknoten in T benützt werden dürfen und (z_{sv}, v) ist der letzte Bogen eines solchen Weges. Wiederum gilt die Konvention, dass falls kein solcher Weg existiert $u_{sv} := \infty$ und $z_{sv} := \emptyset$ ist. Weiter gilt für alle Knoten $v \in T$, dass ein kürzester Weg von s nach v in G existiert, der nur Knoten aus T benützt, d. h. für ein solches v ist u_{sv} gerade die Länge eines kürzesten Weges von s nach v.

Der Algorithmus startet mit $T := \{s\}$. In jedem Schritt wird ein neuer Knoten zu T hinzugenommen. Für die Knoten ausserhalb T wird dann geschaut, ob dieser neu erlaubte Zwischenknoten den entsprechenden u–Wert verkleinert.

Satz 1.16 (Korrektheit des Dijkstra–Algorithmus) *Falls wir den Dijkstra–Algorithmus auf einem nicht–negativ gewichteten, gerichteten Graphen $G = (V, E)$ mit Quelle $s \in V$ ausführen, dann gilt am Ende des Algorithmus, dass*

(i) Falls $u_{sv} < \infty$, so existiert ein kürzester Weg von s nach v der Länge u_{sv}, dessen letzter Bogen (z_{sv}, v) ist.

(ii) Falls $u_{sv} = \infty$, dann existiert kein Weg von s nach v.

Beweis. Zu Beginn des Algorithmus, wenn $T = \{s\}$ ist, gelten (a) und (b):

(a) für $w \in V$ gilt: Falls $u_{sw} < \infty$ ist u_{sw} die Länge eines kürzesten Weges von s nach w, der nur Zwischenknoten aus T benützt, und (z_{sw}, w) ist der letzte Bogen eines solchen Weges. Falls $u_{sw} = \infty$ existiert kein solcher Weg.

(b) für $w \in T$ gilt: u_{sw} ist die Länge eines kürzesten Weges von s nach w.

Algorithmus 1.5 Kürzeste Wege – Dijkstra

(* Sei $G = (V, E)$ ein gerichteter Graph, $s \in V$ eine Quelle und $c_{ij} \geq 0$ für alle $(i, j) \in E$. Der Algorithmus bestimmt für alle $v \in V$ die Länge u_{sv} eines kürzesten Weges von s nach v, sowie den letzten von diesem Weg benützten Bogen (z_{sv}, v). $z_{sv} = \emptyset$ bedeutet, dass kein Weg von s nach v existiert. Im Verlaufe des Algorithmus ist u_{sv} die Länge eines kürzesten Weges von s nach v der nur Zwischenknoten aus einer Menge T benützt und (z_{sv}, v) der letzte Bogen eines solchen Weges *)

begin (* Dijkstra *)
  $T := \{s\}$
  $u_{ss} := 0, z_{ss} := \emptyset$
  $u_{sv} := c_{sv}, z_{sv} := s$ für alle $v \in V \setminus \{s\}$, so dass $(s, v) \in E$
  $u_{sv} := \infty; z_{sv} := \emptyset$ für alle $v \in V \setminus \{s\}$, so dass $(s, v) \notin E$
  $v := s$ (* letzter in T aufgenommener Knoten *)
  while $(T \neq V)$ **and** $(u_{sv} \neq \infty)$ **do**
    Sei $v \notin T$ mit $u_{sv} = \min\{u_{sw} \mid w \notin T\}$
    if $u_{sv} < \infty$ **then**
      $T := T \cup \{v\}$
      (* Marken neu berechnen *)
      for all $w \in \{w' \notin T \mid (v, w') \in E\}$ **do**
        if $u_{sv} + c_{vw} < u_{sw}$ **then**
          (*kürzester s–w–Weg mit Zwischenknoten aus T geht über v*)
          $u_{sw} := u_{sv} + c_{vw}; \quad z_{sw} := v$
        end (* if *)
      end (* for *)
    end (* if *)
  end (* while *)
    für alle $v \in V$ gilt:
      Falls $u_{sv} < \infty$, so existiert ein kürzester Weg von s nach v der Länge u_{sv},
      dessen letzter Bogen (z_{sv}, v) ist.
      Sonst existiert kein Weg von s nach v.
end (*Dijkstra*)

Wir zeigen, dass falls (a) und (b) zu Beginn der **while**–Schleife für T und die momentanen Marken u_{sw} und $z_{sw}, w \in V$ gelten, so gelten sie auch am Ende der **while**–Schleife. Sei $v \notin T, u_{sv} < \infty$, der nächste zu T hinzugenommene Knoten, $T' := T \cup \{v\}$ und $u'_{sw}, z'_{sw}, w \in V$ die neu berechneten Marken.

Für $w \in T$ gilt (a) und (b) sicher. Wir zeigen nun (a) und (b) für v: Ein kürzester Weg P von s nach v besitzt einen ersten Knoten w, der nicht zu T gehört. Die Länge von P ist sicher grösser gleich der Länge des Teilweges P_1 von P, der von s nach w geht. Somit ist $L(P) \geq L(P_1) \geq u_{sw} \geq u_{sv}$, was (a) und (b) impliziert.

Sei nun $w \notin T', G'$ der von $T' \cup \{w\}$ knoteninduzierte Untergraph und u''_{sw} die Länge eines kürzesten Weges von s nach w in G'. Mit der Konvention $c_{vw} := \infty$, falls $(v, w) \notin E$, gilt gemäss Lemma 1.15:

$$\begin{aligned} u''_{sw} &= \min\{u''_{sz} + c_{zw} \mid z \in T'\} \\ &= \min\{\min\{u''_{sz} + c_{zw} \mid z \in T\}, u''_{sv} + c_{vw}\} \\ &= \min\{\min\{u_{sz} + c_{zw} \mid z \in T\}, u_{sv} + c_{vw}\} \\ &= \min\{u_{sw}, u_{sv} + c_{vw}\} = u'_{sw}, \end{aligned}$$

Dies impliziert nun (a).

Falls nun am Ende des Algorithmus $T \neq V$ ist, so ist $u_{sv} = \infty$ für alle $v \notin T$. Für einen solchen Knoten v gibt es keinen Weg von s nach v: Falls es einen Weg P von s nach v gäbe, hätte P einen ersten Knoten w, der nicht zu T gehört. Dann wäre aber gemäss (a) $u_{sw} \leq L(P) < \infty$, was im Widerspruch zu $u_{sw} = \infty$ steht. ♣

Man bemerke, dass die kürzesten Wege mit Hilfe der z–Marken zurückkonstruiert werden können: Sei $n(v), v \in T$ die Reihenfolge, in der die Knoten zu T hinzugenommen wurden. Dann ist $n(z_{sv}) < n(v)$. Deshalb kommt in der Knotenfolge $P = (s = v_n, \ldots, v_1 = v)$, mit $v_{i+1} := z_{sv_i}, i = 1, \ldots, n-1$, jeder Knoten von V höchstens einmal vor und P ist somit ein Weg, ein kürzester Weg von s nach v.

Aufwand des Dijkstra–Algorithmus: Die **while**–Schleife wird höchstens für jeden Knoten von V einmal durchgeführt. Innerhalb der Schleife werden das kleinste $u_{sw}, w \notin T$ bestimmt und die Marken u_{sw} und z_{sw} für die Knoten ausserhalb T neu berechnet, was einem Aufwand $O(V)$ entspricht. Der Gesamtaufwand ist also $O(V^2)$.

1.5.3 Kürzeste Wege für beliebige Bogenlängen

Gegeben: $G = (V, E)$ gerichtet, $V = \{1, \ldots, n\}$, Bogenlängen c_{ij} für $(i, j) \in E$.

Gesucht: Bestimme, ob G einen Zyklus negativer Länge besitzt und falls nicht, bestimme für alle Knotenpaare (v, w) $v, w \in V$ einen kürzesten Weg von v nach w und dessen Länge u_{vw} (falls einer existiert).

Definition 1.17 *Für $v, w \in V := \{1, \ldots, n\}$ und $k \in \{2, \ldots, n+1\}$ wird mit u^k_{vw} die Länge eines kürzesten Weges von v nach w bezeichnet, gegeben dass nur Zwischenknoten aus $\{1, \ldots, k-1\}$ benützt werden dürfen. Zusätzlich sei*

$$u^1_{vw} := \begin{cases} 0 & \text{für } v = w \\ c_{vw} & \text{für } (v, w) \in E \\ \infty & \text{für } v \neq w, (v, w) \notin E \end{cases}$$

Lemma 1.18 *Falls $G = (V, E)$ keinen Zyklus negativer Länge besitzt gilt für $v, w \in V, k = 1, \ldots, n$:*

$$u^{k+1}_{vw} = \min\left\{u^k_{vw}, u^k_{vk} + u^k_{kw}\right\} \tag{1.34}$$

Falls zusätzlich $u^k_{vw} > u^k_{vk} + u^k_{kw}$ und $P'(P'')$ ein kürzester Weg von v nach k (k nach w) ist, gegeben dass nur Zwischenknoten aus $\{1, \ldots, k-1\}$ benützt werden, so ist die aus P' und P'' zusammengesetzte Bogenfolge ein kürzester Weg von v nach w, gegeben dass nur Zwischenknoten aus $\{1, \ldots, k\}$ benützt werden.

Beweis. Sei $v, w \in V$ und $k \in \{1, \ldots, n\}$. Wir zeigen zuerst, dass gilt:

$$u^{k+1}_{vw} \geq \min\left\{u^k_{vw}, u^k_{vk} + u^k_{kw}\right\} \tag{1.35}$$

Falls $u^{k+1}_{vw} = \infty$ ist nichts zu beweisen. Sonst folgt die Ungleichung aus der Tatsache, dass ein kürzester Weg P (mit Zwischenknoten aus $\{1, \ldots, k\}$) entweder k nicht benützt, also $u^{k+1}_{vw} = u^k_{vw}$ ist, oder k benützt. Im zweiten Fall gelten für die Teilwege P_1 von v nach k und P_2 von k nach w: $u^{k+1}_{vw} = L(P) = L(P_1) + L(P_2) \geq u^k_{vk} + u^k_{kw}$.

Wir zeigen nun, dass $u^{k+1}_{vw} \leq \min\left\{u^k_{vw}, u^k_{vk} + u^k_{kw}\right\}$, also dass:

$$u^{k+1}_{vw} \leq u^k_{vw} \text{ und} \tag{1.36}$$

$$u^{k+1}_{vw} \leq u^k_{vk} + u^k_{kw} \tag{1.37}$$

(1.36) gilt gemäss Definition und (1.37) gilt sicher falls $u^k_{vk} = \infty$ oder $u^k_{kw} = \infty$.

Es bleibt (1.37) für den Fall $u^k_{vk} < \infty, u^k_{kw} < \infty$ zu zeigen: Sei $(v = v_1, \ldots, v_a = k)$ ein kürzester Weg von v nach k, der nur Zwischenknoten aus $\{1, \ldots, k-1\}$ benützt, und $(k = v_a, \ldots, v_b = w)$, $a \leq b$, einer von k nach w. Die Bogenfolge $P_1 = ((v_1, v_2), \ldots, (v_{b-1}, v_b))$ besitzt die Länge $L(P_1) = u^k_{vk} + u^k_{kw}$. Falls P_1 ein Weg ist, gilt $u^{k+1}_{vw} \leq L(P_1) = u^k_{vk} + u^k_{kw}$, also (1.37). Sonst existieren Knoten $v_i = v_j$ mit $i < a < j$ und alle Knoten $v_i, \ldots, v_{j-1}$ verschieden. Sei P_2 die Bogenfolge $((v_1, v_2), \ldots, (v_{i-1}, v_i = v_j), (v_j, v_{j+1}), \ldots, (v_{b-1}, v_b))$ und Q der Zyklus $((v_i, v_{i+1}), \ldots, (v_{j-1}, v_j))$.

Es gilt $L(P_2) \leq L(P_2) + L(Q) = L(P_1)$. Falls P_2 immer noch kein Weg ist, kann wiederum ein Zyklus abgespalten werden. Dies wird so lange wiederholt, bis ein Weg P' (von v nach w, mit Zwischenknoten aus $\{1, \ldots, k-1\}$) erhalten wird. Für P' gilt dann: $u^{k+1}_{vw} \leq u^k_{vw} \leq L(P') \leq L(P_1) = u^k_{vk} + u^k_{kw}$, also gilt (1.37) und somit (1.34). Man bemerke, dass in diesem Fall (P_1 ist kein Weg) $u^k_{vw} \leq L(P') \leq L(P_1) = u^k_{vk} + u^k_{kw}$, da P' nicht durch k geht, und somit impliziert $u^k_{vw} > u^k_{vk} + u^k_{kw}$, dass P_1 ein Weg ist, der wegen (1.34) ein kürzester Weg von v nach w ist, gegeben dass nur Zwischenknoten aus $\{1, \ldots, k\}$ benützt werden. ♣

Man bemerke, dass falls G Zyklen negativer Länge besitzt, (1.34) nicht immer gilt.

In dem in der folgenden Abbildung dargestellten Graphen gilt:

$$u^5_{34} = 10 \qquad u^5_{35} = 1 \qquad u^5_{54} = 4,$$

$$u^6_{34} = 10 > u^5_{35} + u^5_{54} = 5$$

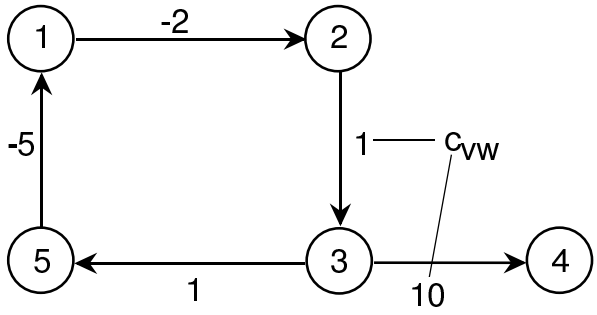


Abbildung 1.24: Graph mit Zyklus negativer Länge

Der Algorithmus zur Bestimmung von kürzesten Wegen zwischen beliebigen Knotenpaaren leitet sich direkt aus dem Lemma 1.18 ab:

Bemerkung 1.19

(i) Falls ein Zyklus negativer Länge entdeckt wird ($u_{ii} < 0$), kann er mit Hilfe eines Weges von i nach z_{ii} und eines Weges von z_{ii} nach i konstruiert werden. Diese Wege können ebenfalls über die z–Werte konstruiert werden.

(ii) Gültigkeit des Floyd-Warshall-Algorithmus:

Beweis. Falls keine Zyklen negativer Länge existieren, garantiert Lemma 1.18, dass u_{ij} die Länge eines kürzesten Weges von i nach j ist und dass $u_{ii} = 0$ ist. Die über die z_{ij} rekonstruierbaren Bogenfolgen sind dann Wege (Lemma 1.18). Falls ein Zyklus negativer Länge existiert und i ein Knoten dieses Zyklus ist, muss spätestens, wenn alle Knoten des Zyklus als Zwischenknoten benützt werden dürfen, $u_{ii} < 0$ sein. ♣

(iii) Der Aufwand vom Floyd–Warshall–Algorithmus ist $O(V^3)$.

Algorithmus 1.6 Floyd–Warshall Algorithmus

(* Sei $G = (V, E)$ ein gerichteter Graph, $V = \{1, \ldots, n\}$. Falls keine Zyklen negativer Länge existieren, bestimmt der Algorithmus für alle $i, j \in V$ die Länge u_{ij} eines kürzesten Weges von i nach j, sowie einen Zwischenknoten z_{ij}. $z_{ij} = \emptyset$ bedeutet, dass kein Zwischenknoten verwendet wird. *)

begin (* Floyd–Warshall *)
  $u_{ii} := 0$ für alle $i \in V$
  $u_{ij} := c_{ij}$ für alle $i, j \in V$, so dass $(i, j) \in E$
  $u_{ij} := \infty$ für alle $i, j \in V$, so dass $(i, j) \notin E$ und $i \neq j$
  $z_{ij} := \emptyset$ für alle $i, j \in V$
  for $k := 1$ **to** n **do**
    for $i := 1$ **to** $n, i \neq$k **do**
      for $j := 1$ **to** $n, j \neq k$ **do**
        if $u_{ij} > u_{ik} + u_{kj}$ **then**
          $u_{ij} := u_{ik} + u_{kj}$
          $z_{ij} := k$
          if $(i = j)$ **and** $(u_{ii} < 0)$ **then**
            es existiert Zyklus negativer Länge
            stop
          end (* if *)
        end (* if *)
      end (* for *)
    end (* for *)
  end (* for *)
  für alle $i, j \in V$ gilt:
    Falls $u_{ij} < \infty$, so existiert ein kürzester Weg von i nach j der Länge u_{ij},
      der sich aus einem kürzesten Weg von i nach z_{ij} und einem kürzesten Weg
      von z_{ij} nach j zusammensetzt.
    Sonst existiert kein Weg von i nach j.
end (* Floyd-Warshall *)

Kapitel 2

Optimale Gerüste und Arboreszenzen

2.1 Optimale Gerüste

Beispiel 2.1 $N = 9$ Orte sind untereinander zu verbinden. Die Orte sind als Knoten und die möglichen Verbindungen mit ihren Längen als Kanten des untenstehenden Graphen $G = (V, E)$ gegeben. Welches ist ein Verbindungsnetz minimaler Länge?

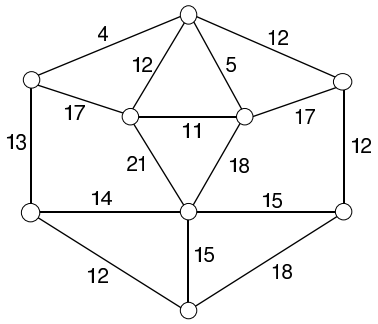


Abbildung 2.1: Mögliche Verbindungen

Offensichtlich müssen die gewählten Verbindungen insgesamt die Kantenmenge E' eines zusammenhängenden Untergraphen $T = (V, E')$ von G bilden. Da zudem die Längen nicht–negativ sind, darf ein Verbindungsnetz minimaler Länge als ein Gerüst minimaler Länge angenommen werden. (Allfällige

Kreise können durch Entfernen von Kanten aufgebrochen werden, wobei der Zusammenhang erhalten bleibt und die Gesamtlänge des Verbindungsnetzes sicher nicht grösser wird.) ◇

Deshalb gilt es, folgende Aufgabe zu lösen:

Problem des Gerüstes minimalen Gewichts:

Gegeben: Ein zusammenhängender Graph $G = (V, E)$. Gewichte $c_e \in \mathbb{R}, e \in E$.

Gesucht: Ein Gerüst $T^* = (V, E^*)$ von G minimalen Gewichts: $c(E^*) \equiv \sum_{e \in E^*} c_e = \min\{c(E') \mid T' = (V, E') \text{ Gerüst von } G\}$.

Bemerke, dass es in einem Graphen viele Gerüste gibt. So kann man zeigen, dass ein vollständiger Graph mit n Knoten n^{n-2} Gerüste hat. Dennoch ist das Problem des optimalen Gerüstes eines der einfachsten der kombinatorischen Optimierung.

Definition 2.2 (Aufspannender Wald) *Ein aufspannender Wald W von $G = (V, E)$ mit Komponenten $(V_1, E_1), \ldots, (V_k, E_k)$ ist ein Wald von G, so dass $\bigcup_{i=1}^{k} V_i = V$.*

Bemerke, dass gewisse E_i leer sein können, und dass jeder kreislosen Kantenmenge $E' \subseteq E$ ein aufspannender Wald zugeordnet ist.

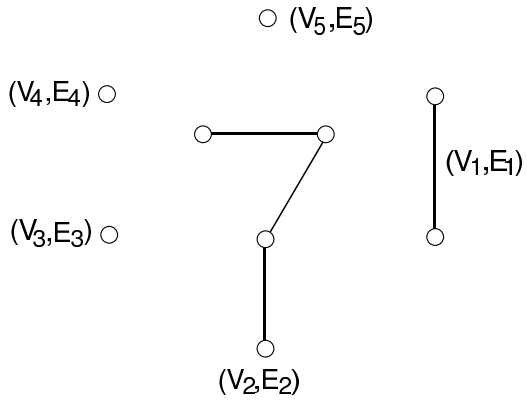


Abbildung 2.2: Aufspannender Wald mit 5 Komponenten für den Graphen von Abbildung 2.1. Die Komponenten $(V_i, E_i), i = 3, 4, 5$, bestehen nur aus einem Knoten, d.h. $E_i = \emptyset$.

Definition 2.3 *Für jedes $S \subseteq V$, sei $\delta(S)$ die Menge aller Kanten von G mit einem Endknoten in S und dem anderen in $V \setminus S$. Für jedes $A \subseteq E$, sei $c(A) \equiv \sum_{e \in A} c_e$.*

Lemma 2.4 *Sei W ein aufspannender Wald eines Graphen G mit k Komponenten $(V_1, E_1), \ldots, (V_k, E_k), k > 1, E_W \equiv \bigcup_{i=1}^{k} E_i$ und $e^* \in \delta(V_1)$ mit $c_{e*} = \min\{c_e \mid e \in \delta(V_1)\}$. Dann existiert ein Gerüst $T^* = (V, E^*)$ von G, so dass $e^* \cup E_W \subseteq E^*$ und $c(E^*) = \min\{c(E') \mid T' = (V, E')$ Gerüst von G mit $E_W \subseteq E'\}$.*

Beweis. Sei T^* ein Gerüst minimalen Gewichts unter allen Gerüsten $T'' = (V, E'')$ mit $E_W \subseteq E''$ und $e^* \in E''$, und nehme an, es gebe ein Gerüst $T' = (V, E')$ mit $E_W \subseteq E'$ und $c(E') < c(E^*)$. $e^* \cup E'$ enthält genau einen Kreis K, und es existiert eine Kante $e' \in K \cap T'$ mit $e' \in \delta(V_1)$. Dann ist $c_{e*} \leq c_{e'}$ und $T' - e' \cup e^*$ ein Gerüst, das e^* und E_W in seiner Kantenmenge enthält. Aber $c(E' - e' \cup e^*) \leq c(E') < c(E^*)$, ein Widerspruch zur Definition von T^*. ♣

Mehrere Algorithmen für das Problem des minimalen Gerüstes beruhen auf diesem Lemma (siehe auch Übungen). Der folgende Algorithmus ist nicht der effizienteste, jedoch ein Spezialfall eines grundlegenden Algorithmus der kombinatorischen Optimierung.

Algorithmus 2.1 Kruskal-Algorithmus

(*Findet die Kantenmenge eines minimalen Gerüstes im zusammenhängenden Graphen $G = (V, E)$*)

begin (* Kruskal *)
  Ordne Kanten von E, so dass $c_{e_1} \leq \cdots \leq c_{e_m}$, $m := \mid E \mid$
  $E^* := \emptyset$
  for $j := 1$ **to** m **do**
    if ($E^* \cup e_j$ enthält keinen Kreis) **then**
      $E^* := E^* \cup e_j$
    end (* if *)
  end (* for *)
  E^* ist Kantenmenge eines Gerüstes minimalen Gewichts
end (* Kruskal *)

Bemerkung 2.5 Offensichtlich brauchen die Kantengewichte nicht im voraus geordnet zu werden. Zudem kann, falls $\mid E^* \mid = \mid V \mid -1$, die for–Schleife verlassen werden.

Satz 2.6 (Gültigkeit des Kruskal–Algorithmus) *Der Kruskal–Algorithmus findet ein Gerüst minimalen Gewichts.*

Beweis. Verwende Induktion: Sei E^*_{j-1} die Kantenmenge E^* am Anfang des j–ten Durchlaufs der for–Schleife. Annahmegemäss existiert ein optimales Gerüst, dessen Kantenmenge E^*_{j-1} enthält (richtig für $E^*_0 = \emptyset$). Falls E^*_{j-1} zu $E^*_j = E^*_{j-1} \cup e_j$ erweitert wird, ist E^*_{j-1} die Kantenmenge eines aufspannenden Waldes W mit Komponenten $(V_1, E_1), \ldots, (V_k, E_k), k \geq 2$. O.E.d.A. ist $e_j \in \delta(V_1)$: nach Lemma 2.4 existiert ein optimales Gerüst, das E^*_j enthält. ♣

Es ist leicht einzusehen, dass ein Gerüst *maximalen* Gewichts bezüglich Gewichten $c_e, e \in E$, mit folgendem Greedy–Algorithmus gelöst werden kann:

In dieser Form besagt der Algorithmus: Betrachte jeweils die gewichts–(„gewinn–“) trächtigste Kante unter den noch nicht betrachteten Kanten: Falls möglich (d.h. falls die Kante keinen Kreis mit schon genommenen Kanten bildet), nehme sie in E^* auf, sonst verwerfe sie. Deshalb der Name „greedy“ (= gierig).

Algorithmus 2.2 Greedy–Algorithmus

(*Findet Kantenmenge eines maximalen Gerüstes im zusammenhängenden Graphen $G = (V, E)$ *) **begin** (* Greedy *)
  Ordne Kanten von E, so dass $c_{e_1} \geq \cdots \geq c_{e_m}$, $m := \mid E \mid$
  $E^* := \emptyset$
  for $j := 1$ **to** m **do**
    if ($E^* \cup e_j$ enthält keinen Kreis) **then**
      $E^* := E^* \cup e_j$
    end (* if *)
  end (* for *)
  E^* ist Kantenmenge eines Gerüstes maximalen Gewichts
end (* Greedy *)

Der Greedy–Algorithmus kann auch zum Auffinden eines Waldes maximalen Gewichts verwendet werden.

Problem des Waldes maximalen Gewichts:

Gegeben: Ein Graph $G = (V, E)$. Gewichte $c_e \in \mathbb{R}, e \in E$.

Gesucht: Ein Wald $W = (V, E^*)$ von G maximalen Gewichts:
$c(E^*) = \max\{c(E') \mid W = (V, E') \text{ Wald von } G\}$.

Falls gewisse Gewichte c_e negativ sind, werden natürlich nur Kanten mit nicht–negativen c_e in E^* aufgenommen (m ist als die Anzahl Kanten mit positivem Gewicht zu wählen). Falls die Gewichte positiv sind, ist ein Wald maximalen Gewichts auch immer maximaler Kardinalität, d.h. – bei zusammenhängendem Graphen G – ein Gerüst von G.

2.2 Optimale Arboreszenzen

2.2.1 Das Problem der optimalen Arboreszenz

Sei $G = (V, E)$ ein *gerichteter* Graph. Für jeden Bogen $e \in E$, bezeichne mit

$h(e)$ den Endknoten von e (head)
$t(e)$ den Anfangsknoten von e (tail)

Soll der zugrunde liegende Graph G spezifiziert werden, wird für $h(e)$ $(t(e))$ auch $h_G(e)(t_G(e))$ geschrieben. Im selben Sinn wird die Menge der aus $S \subseteq V$ herausgehenden Bögen mit $\delta_G(S) := \{e \in E \mid h(e) \notin S, t(e) \in S\}$ bezeichnet.

Definition 2.7 (Arboreszenz) *Eine Arboreszenz $A = (V', E')$ ist ein Graph , so dass*

(i) A ein Baum ist, wenn man von den Richtungen auf den Bogen absieht,

(ii) in jeden Knoten höchstens ein Bogen von A hineingeht: Falls $e, e' \in E'$ und $e \neq e'$, so ist $h(e) \neq h(e')$.

Definition 2.8 (Spannende Arboreszenz) *Eine spannende Arboreszenz $A = (V, E')$ in $G = (V, E)$ ist ein Untergraph von G, der eine Arboreszenz ist und der alle Knoten von G aufspannt.*

Definition 2.9 (Branching) *Ein Branching $B = (V', E')$ in G ist ein Untergraph von G, dessen (schwach zusammenhängende) Komponenten Arboreszenzen sind.*

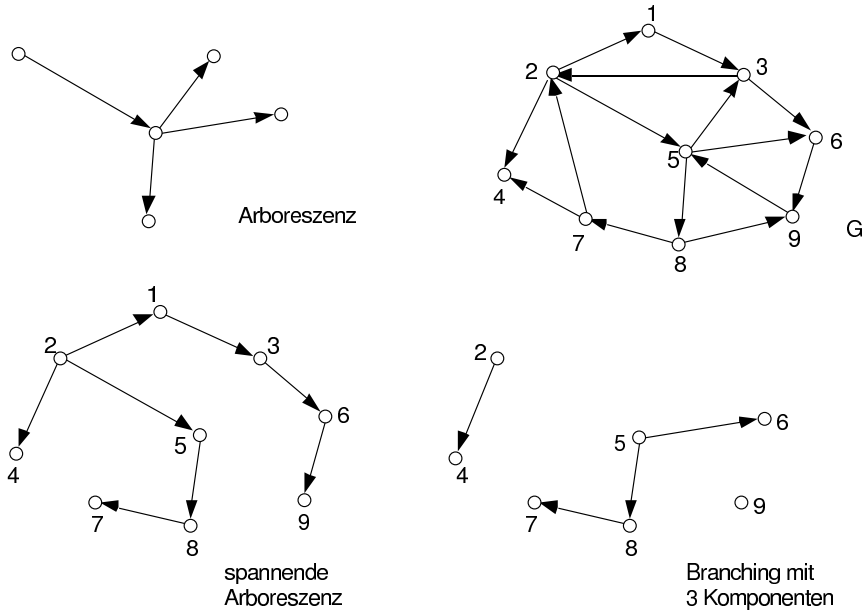


Abbildung 2.3: Spannende Arboreszenz und Branching

Man bemerke, dass die in einem gerichteten Graphen definierten Begriffe Arboreszenz/spannende Arboreszenz/Branching den in einem ungerichteten Graphen definierten Begriffen Baum/Gerüst/Wald entsprechen.
Das Analogon zum Problem des Gerüstes minimalen Gewichts von Kapitel 2.1 ist das Problem der spannenden Arboreszenz minimalen Gewichts:

Problem der spannenden Arboreszenz minimalen Gewichts:

Gegeben: Ein gerichteter Graph $G = (V, E)$. Gewichte $c_e \in \mathbb{R}, e \in E$.

Gesucht: Eine spannende Arboreszenz $A^* = (V, E^*)$ von G minimalen Gewichts:
$c(E^*) = \min\{c(E') \mid A' = (V, E') \text{ spannende Arboreszenz von } G\}$.

Bemerkung 2.10

(i) Wiederum gilt die Bezeichnung $c(E^*) = \sum\{c_e \mid e \in E^*\}$.

(ii) Nicht jeder gerichtete Graph besitzt eine spannende Arboreszenz: Der Graph von Abbildung 2.4 besitzt keine solche.

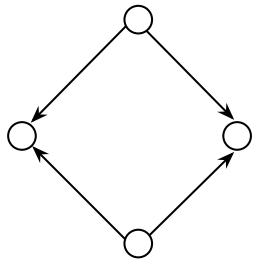


Abbildung 2.4: Gerichteter Graph ohne spannende Arboreszenz

(iii) Andererseits kann das Problem in einem erweiterten Graphen G' betrachtet werden, in dem sicher eine zulässige Lösung, d. h. eine spannende Arboreszenz existiert: Füge einfach zusätzliche Bögen mit hohem Gewicht M in G hinein, um G stark zusammenhängend zu machen. Entweder ist in G' eine spannende Arboreszenz minimalen Gewichts auch in G, oder sie macht von Bögen mit Gewicht M Gebrauch, in welchem Falle es keine spannende Arboreszenz in G gibt.

Dem Problem des Waldes maximalen Gewichts von Kapitel 2.1 entspricht das Problem des Branching maximalen Gewichts.

Problem des Branching maximalen Gewichts:

Gegeben: $G = (V, E)$ gerichtet. Gewichte $c_e \in \mathbb{R}, e \in E$.

Gesucht: Ein Branching $B^* = (V^*, E^*)$ von G maximalen Gewichts:
$c(E^*) = \max\{c(E') \mid B' = (V', E') \text{ Branching von } G\}$.

Hier gibt es offensichtlich immer eine zulässige Lösung, nämlich ein Branching mit $E^* = \emptyset$. Zudem ist es einfach, das Problem der spannenden Arboreszenz minimalen Gewichts in das Problem des Branching maximalen Gewichts zurückzuführen, indem eine einfache Gewichtstransformation vorgenommen wird (siehe Übung). Im nächsten Kapitel wird deshalb das Branching–Problem diskutiert und gelöst.

2.2.2 Branching–Algorithmus

Für die Beschreibung des Algorithmus von Edmonds zur Lösung des Branching–Problems benötigen wir das Konzept der *Schrumpfung* (shrinking, contracting).

Definition 2.11 (Schrumpfen) *Sei $G_0 = (V_0, E_0, \Psi_0)$ ein zusammenhängender Untergraph von $G = (V, E, \Psi)$. Das Schrumpfen von G_0 zu einem Pseudoknoten q besteht im Übergang von G zu einem Graphen $G' = (V', E', \Psi')$, der wie folgt definiert ist:*

$$V' = V - V_0 \cup q \tag{2.1}$$
$$E' = E - \{e \mid \Psi(e) = (v, w) \text{ mit } v, w \in V_0\} \tag{2.2}$$

und die Inzidenz–Abbildung $\Psi' : E' \to V' \times V'$ ist definiert durch:

$$\Psi'(e) = \begin{cases} \Psi(e) = (v, w), & \text{falls } v, w \notin V_0 \\ (v, q), & \text{falls } \Psi(e) = (v, w) \text{ und } v \notin V_0, w \in V_0 \\ (q, w), & \text{falls } \Psi(e) = (v, w) \text{ und } w \notin V_0, v \in V_0 \end{cases} \tag{2.3}$$

Beispiel 2.12 In G von Abbildung 2.3 sei G_0 der von der Knotenmenge $\{3, 5, 6, 9\}$ induzierte Untergraph. Nach Schrumpfung von G_0 in G ist G':

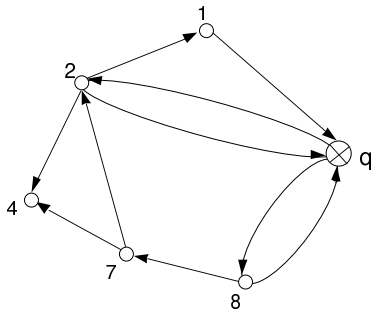


Abbildung 2.5: Graph G' nach der Schrumpfung

$\diamond$

Die *Expansion* des Pseudoknotens q von G' ist die *Umkehroperation* der Schrumpfung und liefert wieder G.

Bemerkung 2.13 Offensichtlich kann man sich im Problem des Branching maximalen Gewichts auf solche Branchings B beschränken, die keine isolierten Knoten haben. Deshalb identifizieren wir im weiteren der Einfachheit halber ein Branching B mit seiner Bogenmenge, die wir auch mit B bezeichnen wollen.

Im folgenden Algorithmus wird vom Schrumfen eines Zyklus Q gesprochen. Damit wird gemeint, dass der ganze von den Knoten von Q induzierte Untergraph zu einem Pseudoknoten q geschrumpft wird (auch ein Bogen ausserhalb Q, dessen Extremitäten Knoten von Q sind, verschwindet in der Schrumpfung).

Algorithmus 2.3 Branching–Algorithmus

(* Sei $G_0 = (V_0, E_0)$ gegeben mit Gewichten $c_e, e \in E_0$. Der Algorithmus bestimmt eine Bogenmenge B eines Branchings maximalen Gewichts *)
begin (* Branching *)
$V := V_0, E := E_0, G := G_0$
$P := V$
$B := \emptyset$
$j := 0$
(* Greedy–Schritte *)
while $P \neq \emptyset$ **do**
wähle $v \in P, P := P \setminus v$
if $\delta_G(\overline{v}) \cap \{e \mid c_e > 0\} \neq \emptyset$ **then**
Sei $e \in E$ mit $c_e = \max\{c_e \mid e \in \delta_G(\overline{v}) \cap \{e \mid c_e > 0\}\}$
$B := B \cup e$
if (B enthält Zyklus) **then**
(* Schrumpfung *)
$j := j + 1$
$Q_j :=$ Zyklus von B
$c_{\min} := c_{e_j} := \min\{c_e \mid e$ Bogen von $Q_j\}$
Bilde $G' = (V', E')$ durch Schrumpfen von Q_j zum Pseudoknoten q_j
$c'_e := c_e$ for all $e \in E' \setminus \delta_{G'}(\overline{q}_j)$
for all $e \in \delta_{G'}(\overline{q}_j)$ **do**
$c'_e := c_e - (c_{\overline{e}} - c_{\min})$, wobei $\overline{e}$ derjenige Bogen von Q_j ist mit $h_G(\overline{e}) = h_G(e)$
end (* for *)
$B := B \cap E'$
$P := P \cup \{q_j\}$
$c := c'$
$G := G'; V := V'; E := E'$
end (* if *)
end (* if *)
end (* while *)
(* Expansion *)
for $i := j$ **downto** 1 **do**
Expandiere Pseudoknoten q_i, erhalte Graph $G' = (V', E')$
if $B \cap \delta_G(\overline{q}_i) \neq \emptyset$ **then**
$B := B \cup Q_i \setminus e'$, wobei e' Bogen von Q_i mit $h_{G'}(e') = h_{G'}(\overline{e})$ und $\overline{e} \in B \cap \delta_G(\overline{q}_i)$
end (* if *)
else
$B := B \cup Q_i \setminus e_i$ (e_i Bogen mit minimalem Gewicht in Q_i)
end (* else *)
$G := G', V := V', E := E'$
end (* for *)
B ist Bogenmenge eines Branchings maximalen Gewichts
end (* Branching *)

Beispiel 2.14 Den Branching–Algorithmus erklären wir hier an einem Beispielgraphen. Der Graph ist eine Gerichtete Variante des Beispiels aus Abbildung 2.1.

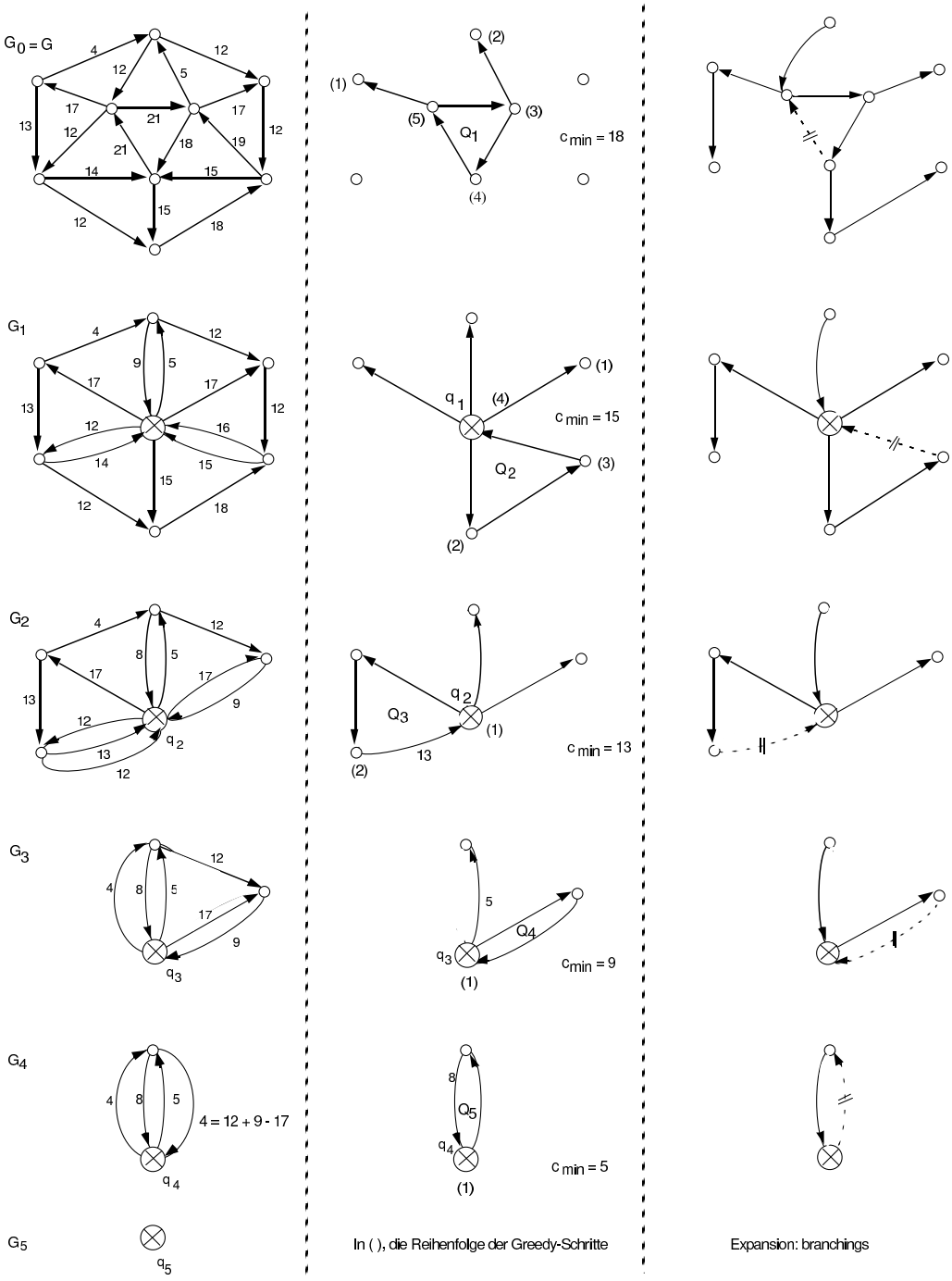


Abbildung 2.6: Branching–Algorithmus anhand eines Beispiels

$\diamond$

2.2.3 Beweis der Gültigkeit des Algorithmus

Beweis. Seien $G_0, G_1, \ldots, G_m$ die in der Schrumpfungsphase gebildeten Graphen, wobei der Graph G_i durch Schrumpfung eines in Greedy–Schritten in G_{i-1} bestimmten Zyklus Q_i entstand. Seien weiter $c^0 = c, c^1, \ldots, c^m$ die entsprechenden Gewichtsvektoren und $B^m, \ldots, B_0$ die in der Expansionsphase konstruierten Branchings in $G_m, \ldots, G_0$. Bezeichne ein Branching maximalen Gewichts in G_i bezüglich Gewichte c_i als c_i–*optimal*.

Der Beweis der Gültigkeit des Algorithmus erfolgt induktiv, indem gezeigt wird:

$$\begin{array}{l} B^i \quad c^i\text{–optimales} \\ \text{Branching in } G_i \end{array} \quad \Rightarrow \quad \begin{array}{l} B^{i-1} \quad c^{i-1}\text{–optimales} \\ \text{Branching in } G_{i-1} \end{array} \qquad \text{für } i = m, \ldots, 1 \tag{2.4}$$

Zur Verankerung der Induktion, bemerke, dass B^m in G_m offensichtlich c^m–optimal ist, da es durch Greedy–Schritte erhalten wurde (oder das triviale Branching $\emptyset$ in einem Graphen G_m mit leerer Bogenmenge ist).

Um (2.4) zu beweisen, sei zunächst die einfachere Schreibweise des Algorithmus eingeführt, d. h.

$$\begin{array}{llll} G, B, c & \text{anstatt} & G_{i-1}, B^{i-1}, c^{i-1} & \text{und} \\ G', B', c' & \text{anstatt} & G_i, B^i, c^i. & \end{array}$$

Der Beweis beruht auf folgenden zwei Sätzen:

Satz 2.15 *Sei $Q = (V_Q, E_Q)$ ein Zyklus von G, der im Algorithmus erhalten wurde (Greedy–Schritte). Dann existiert ein c–optimales Branching B von G, so dass*

$$|E_Q - B| = 1, \ \textit{und} \tag{2.5}$$

$$E_Q - B = \{e_{\min}\}, \textit{falls kein } e \in B \textit{existiert mit } h(e) \in V_Q \textit{und } t(e) \notin V_Q. \tag{2.6}$$

Dabei bezeichnet $e_{\min}$ einen Bogen minimalen Gewichts $c_{\min}$ in E_Q.

Sei **B** die Familie aller Branchings von G, die (2.5) und (2.6) erfüllen. Satz 2.15 besagt, dass man sich in der Suche nach einem c–optimalen Branching auf die Familie **B** beschränken kann.

Satz 2.16 *Sei Q wie in Satz 2.15, und sei G' der aus G durch Schrumpfung von Q zum Pseudoknoten q erhaltene Graph. Folgendes gilt:*

(i) Es existiert eine ein–eindeutige Abbildung zwischen der Familie **B** *und der Familie aller Branchings von G'.*

(ii) Zwischen $B \in \mathbf{B}$ und seinem Bild $B' = B \cap E'$ in $G' = (V', E')$ besteht die Beziehung:

$$c(B) = c'(B') + c(E_Q) - c_{\min}.$$

Bevor die Sätze 2.15 und 2.16 bewiesen werden, sei bemerkt, dass damit (2.4) unmittelbar folgt, d.h. es gilt:

Korollar 2.17 *Ein c'–optimales Branching B' in G' liefert ein c–optimales Branching B in G, wenn q wieder zu G expandiert wird.*

Beweis. $c(E_Q) - c_{\min}$ in (ii) von Satz 2.16 stellt eine Konstante dar: B ist genau dann c–optimal über alle Branchings von **B** wenn das Bild B' in G' c'–optimal ist. Zudem sichert Satz 2.15, dass ein B, das c–optimal über **B** auch c–optimal in G schlechthin ist (über alle Branchings von G). ♣

Damit ist die Gültigkeit des Algorithmus bewiesen. ♣

Es bleibt die Sätze 2.15 und 2.16 zu beweisen.

Beweis Satz 2.15. Unter allen optimalen Branchings von G wähle eines, etwa B, mit maximaler Anzahl Bögen aus Q. Da $Q \not\subseteq B$, ist $|E_Q - B| = k \geq 1$. Wir zeigen (2.5), d. h. dass $|E_Q - B| = 1$ ist: Sei $E_Q - B = \{e_1, \ldots, e_k\}$, wobei für $i = 1, \ldots, k$, mit $e_0 := e_k$ gilt: $h(e_{i-1}) = t(e_i)$ oder es existiert ein gerichteter Weg in $E_Q \cap B$ von $h(e_{i-1})$ nach $t(e_i)$.

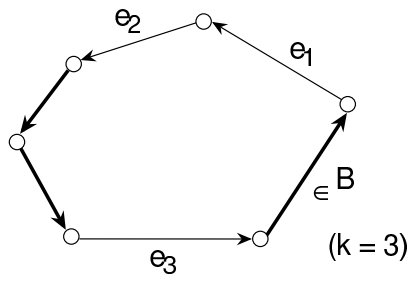


Abbildung 2.7: Numerierung der Bögen von $E_Q - B$

Sei $i \in \{1, \ldots, k\}$, wir zeigen:

$$\text{In } B \text{ existiert ein gerichteter Weg von } h(e_i) \text{ nach } h(e_{i-1}) \tag{2.7}$$

Sei f derjenige Bogen von B mit $h(f) = h(e_i)$ ($f := \emptyset$, falls kein solcher Bogen existiert). Man bemerke, dass $B \cup e_i - f$ kein Branching ist, da sonst wegen $c_{e_i} \geq c_f$ $B \cup e_i - f$ ein optimales Branching von G wäre, mit mehr Bögen aus E_Q als B.

Somit enthält $B \cup e_i - f$ einen Zyklus und somit einen gerichteten Weg in B von $h(e_i)$ nach $t(e_i)$. Da in jedem Knoten höchstens ein Bogen von B einmündet, geht dieser Weg durch $h(e_{i-1})$, womit (2.7) bewiesen ist.

Falls nun $|E_Q - B| = k \geq 2$ ist, dann enthalten die k gerichteten Wege in B von $h(e_i)$ nach $h(e_{i-1}), i = 1, \ldots, k$ einen Zyklus Z, ein Widerspruch, dass B ein Branching ist. Somit ist $k = 1$ und (2.5) bewiesen.

Zum Beweis von (2.6): Falls $E_Q - B = e \neq e_{\min}$, ist $e_{\min} \in B$ und $B' = B \cup e - e_{\min}$ auch ein optimales Branching mit $|E_Q - B'| = 1$ und $E_Q - B = \{e_{\min}\}$. ♣

Beweis des Satzes 2.16. Wir spalten den Beweis in 3 Teile auf:

(a) Falls $B \in \mathbf{B}$, dann ist $B' = B \cap E'$ ein Branching von G':

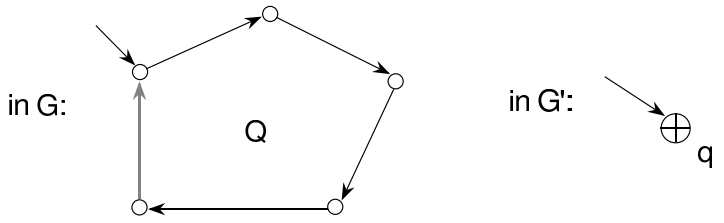


Abbildung 2.8: $B' = B \cap E'$ ein Branching von G'

Da $|E_Q - B| = 1$, gibt es höchstens einen Bogen $e \in B'$ mit $h(e) \in V_Q$, $t(e) \notin V_Q$, und somit höchstens einen Bogen $e \in B'$ mit $h'(e) = q$. Dass höchstens ein Bogen $e \in B'$ existiert mit $h'(e) = v$, für alle $v \in V' - q$, ist offensichtlich.

Würde zudem B' einen Zyklus Q' enthalten, so würde B einen Zyklus in $Q \cup Q'$ enthalten, ein Widerspruch. Also ist B' ein Branching.

(b) Falls B' ein Branching von G' ist, gibt es genau ein Branching $B \in \mathbf{B}$, so dass $B' = B \cap E'$:Da $B \in \mathbf{B}$, ist der Bogen $e \in E_Q - B$, der entfernt wird, genau festgelegt ($|E_Q - B| = 1$!).

(c) Zu zeigen: $c(B) = c'(B') + c(E_Q) - c_{min}$.

Es gilt:

$$\begin{aligned} c(B) &= c(B') + c(B \cap E_Q), \text{ d.h.} \\ c(B) &= c(B') + c(E_Q) - c(E_Q - B) \end{aligned} \tag{2.8}$$

Zerlege E' nach

$$E' = E_0 \cup E_q \quad \text{mit} \quad \begin{cases} E_0 &= \{e \in E' \mid h'(e) \in V' - q\} \\ E_q &= \{e \in E' \mid h'(e) = q\} \end{cases} \tag{2.9}$$

Somit gilt nun

$$c(B') = c(B' \cap E_0) + c(B' \cap E_q) = c'(B' \cap E_0) + c(B \cap E_q) \tag{2.10}$$

(Wir haben $B' = B \cap E'$ und die Gewichtsbeziehungen

$$c'_e = \begin{cases} c_e - (c_{\overline{e}} - c_{min}), & \text{falls } h'(e) = q \\ c_e, & \text{sonst} \end{cases}$$

benützt.)

Wir zeigen jetzt

$$c(B \cap E_q) - c(E_Q - B) = c'(B \cap E_q) - c_{min}. \tag{2.11}$$

(i) $B \cap E_q = \{e\}, E_Q - B = \{\overline{e}\}$ und damit

$$c(B \cap E_q) - c(E_Q - B) = c_e - c_{\overline{e}} = c'_e - c_{\min} = c'(B \cap E_q) - c_{\min} \tag{2.12}$$

(ii) $B \cap E_q = \emptyset, E_Q - B = \{e_{\min}\}$ und damit

$$c(B \cap E_q) - c(E_Q - B) = -c_{\min} = c'(B \cap E_q) - c_{\min} \tag{2.13}$$

Somit gilt unter Berücksichtigung von (2.8), (2.10) und (2.11):

$$\begin{aligned} c(B) &= c'(B' \cap E_0) + c(B \cap E_q) + c(E_Q) - c(E_Q - B) && (2.14)\\ &= c'(B' \cap E_0) + c(E_Q) + c'(B \cap E_q) - c_{\min} && (2.15)\\ &= c'(B') + c(E_Q) - c_{\min}. && (2.16) \end{aligned}$$

Kapitel 3

Matroide

3.1 Motivation und Definition

Allgemein spielt der Greedy–Algorithmus in der Matroidentheorie eine zentrale Rolle.

Definition 3.1 (Matroid) *Ein Matroid $M = (E, \mathcal{F})$ ist gegeben durch:*

(i) eine endliche Grundmenge E;

(ii) eine Familie $\mathcal{F}(\neq \emptyset)$ von Teilmengen von E, den sogenannten unabhängigen Mengen, die zwei Axiome erfüllen:

(a) Falls $I \in \mathcal{F}$ und $J \subseteq I$, dann $J \in \mathcal{F}$ (insb. $\emptyset \in \mathcal{F}$);

(b) Falls $I \in \mathcal{F}$, $J \in \mathcal{F}$ und $|I| < |J|$, dann existiert $e \in J - I$, so dass $I \cup e \in \mathcal{F}$.

Eine äquivalente Definition eines Matroids erhält man, wenn man (iib) ersetzt durch

(b') Für beliebiges $A \subseteq E$;

$$\left.\begin{array}{l} J \in \mathcal{F} \text{ maximal in } A \\ J' \in \mathcal{F} \text{ maximal in } A \end{array}\right\} \Rightarrow |J| = |J'| =: r(A) \qquad (3.1)$$

Dabei wird unter „$J \in \mathcal{F}$ maximal in A“ verstanden: $J \in \mathcal{F}$, $J \subseteq A$, sodass kein $I \in \mathcal{F}$ mit $J \subset I$ exisiert. $r(A)$ *heisst* Rang *von A. Alle maximalen unabhängigen Mengen in A haben dieselbe Kardinalität* $r(A)$ *und heissen* Basen *von A (siehe auch Definition 3.9).*

Bemerkung 3.2 Übung: Zeige, dass die Axiome (iib) und (iib') äquivalent sind.

Betrachten wir einige Beispiele von Matroiden:

Beispiel 3.3 (Graphisches Matroid) Sei $G = (V, E)$ ein ungerichteter Graph und $M_G = (E, \mathcal{F})$ sei definiert durch:

$$\mathcal{F} := \{I \subseteq E \mid I \text{ ist Kantenmenge eines Waldes von } G\}. \tag{3.2}$$

Beachte, dass das Problem der unabhängigen Menge maximalen Gewichts in M_G genau das Problem des Waldes maximalen Gewichts in G ist. ◇

Beispiel 3.4 (Matrix–Matroid) Seien $A_1, \ldots, A_k$ Vektoren in $\mathbb{R}^n$ und $M_A = (E, \mathcal{F})$ wobei

$$E := \{1, \ldots, k\} \text{ und}$$
$$\mathcal{F} := \{I \subseteq E \mid A_i, i \in I, \text{ sind linear unabhängig}\}.$$

M_A ist ein sog. Matrix–Matroid. Hier fällt der Begriff der Unabhängigkeit im Matroid mit dem Begriff der gewöhnlichen linearen Unabhängigkeit zusammen. ◇

Beispiel 3.5 (Partitionsmatroid) Seien $E_1, \ldots, E_k$ eine beliebige Zerlegung einer Menge E, d. h. $E = \bigcup_{i=1}^{k} E_i$ und $E_i \cap E_j = \emptyset$, falls $i \neq j$, und $d_1, \ldots, d_k$ nichtnegative ganze Zahlen. $M_p = (E, \mathcal{F})$, wobei

$$\mathcal{F} := \{I \subseteq E \mid \; \mid I \cap E_i\mid \leq d_i \text{ für alle } i = 1, \ldots, k\} \tag{3.3}$$

ist ein Partitionsmatroid. Falls alle $d_i = 1$ sind, heisst M_p ein *unitäres* Partitionsmatroid. ◇

Beispiel 3.6 (Transversalmatroid) Sei E eine beleibige, endliche Menge und $L = \{S_1, \ldots, S_n\}$ eine Familie von (nicht notwendigerweise verschiedenen) Teilmengen $S_i \subseteq E$. Ein *Transversal* $T \subseteq E$ von L ist eine Menge von n distinkten Elementen $e_{j(1)}, \ldots, e_{j(n)}$, so dass $e_{j(i)}, \in S_i, i = 1, \ldots, n$. Ein *partielles Transversal* $I \subseteq E$ von L ist eine Menge von k verschiedenen Elementen $e_{j(1)}, \ldots, e_{j(k)}$, $k \leq n$, so dass k verschiedene Mengen $S_{j(1)}, \ldots, S_{j(k)}$ aus L existieren mit $e_{j(m)} \in S_{j(m)}, m = 1, \ldots, k$. (Bemerke, dass ein Transversal ein partielles Transversal mit n Elementen ist.)
Das Matroid $M_T = (E, \mathcal{F})$, wobei

$$\mathcal{F} := \{I \subseteq E \mid I \text{ ist ein partielles Transversal von } L\}, \tag{3.4}$$

ist ein sog. Transversalmatroid. ◇

3.2 Einige Grundlagen

Definition 3.7 (Inzidenzvektor) *Gegeben sei ein Matroid $M = (E, \mathcal{F})$, $E = \{e_1, e_2, e_3, \ldots, e_n\}$. Der Inzidenzvektor von $A \subset E$ ist definiert durch $I_A \in \{0,1\}^E$ mit $(I_A)_i = 1 \iff e_i \in A$.*

Beispiel 3.8 $M = (E, \mathcal{F})$ mit $E = \{e_1, e_2, e_3, e_4\}$.

$$\begin{aligned} A &= \{e_1, e_2, e_3\} \qquad I_A^T = (1,1,1,0) \\ B &= \{e_1, e_2, e_4\} \qquad I_B^T = (1,1,0,1) \end{aligned} \tag{3.5}$$

Beachte, dass $I_{AB} = I_A^T \cdot I_B = |A \cap B|$ $\diamond$

Definition 3.9 (Rang) *Der Rang von A (eine beliebige Teilmenge von E) ist definiert durch $r(A) = \max\{|F| \mid F \subset A,\ F \in \mathcal{F}\}$*

Bemerkung 3.10 Daraus folgt unmittelbar: $\forall\, F \in \mathcal{F}$ gilt: $|\underbrace{F \cap A}_{\in \mathcal{F}}| \leq r(A)$

Bemerkung 3.11 Sei x der Inzidenzvektor einer unabhängigen Menge von E eines Matroids, dann gilt

$$\sum_{e \in A} x_e = I_A^T x \leq r(A) \quad \forall A \subset E \tag{3.6}$$

Definition 3.12 (Matroid-Polyeder) *Sei $M \in \{0,1\}^{\mathcal{P}(E) \times E}$ eine Matrix mit den Zeilen I_A^T für alle $A \subset E$. Sei $r \in \mathbb{N}^{\mathcal{P}(E)}$ mit $r_A := r(A)$. Der Matroid-Polyeder P von M ist definiert durch:*

$$\begin{gathered} P := \{x \in \mathbb{R}^{|E|} \mid 0 \leq x_e \leq 1 \quad \forall e \in E,\ Mx \leq r\} \\ \iff \\ P := \{x \in \mathbb{R}^{|E|} \mid 0 \leq x_e \leq 1 \quad \forall e \in E,\ \textstyle\sum_{e \in A} x_e \leq r(A) \quad \forall A \subset E\} \end{gathered}$$

Lemma 3.13

(i) Der Rang ist submodular, d. h. für alle $A, B \subset E$ mit $A \cap B \neq \emptyset$ gilt:

$$r(A) + r(B) \geq r(A \cup B) + r(A \cap B) \tag{3.7}$$

(ii) Sei $x \in P$ und $A, B \subset E$ und $A \cap B \neq \emptyset$. Dann gilt:

$$\begin{array}{l} I_A^T x = r(A) \\ I_B^T x = r(B) \end{array} \iff \begin{array}{c} I_{A\cap B}^T x = r(A \cap B) \\ I_{A\cup B}^T x = r(A \cup B) \end{array} \tag{3.8}$$

(iii) Sei $x \in P$ und $A, B \subset E$ und $A \cap B = \emptyset$. Dann gilt:

$$\begin{array}{l} I_A^T x = r(A) \\ I_B^T x = r(B) \end{array} \implies \begin{array}{c} I_A^T x = r(A) \\ I_{A\cup B}^T x = r(A \cup B) \end{array} \tag{3.9}$$

Beweis.

(i) Sei $L \subset A \cap B$, $L \in \mathcal{F}$ mit $|L| = r(A \cap B)$
und sei $F \subset A \cup B$, $L \subset F$, $F \in \mathcal{F}$ mit $|F| = r(A \cup B)$
Definiere $H := F \cap (A \setminus B)$, $H' := F \cap (B \setminus A)$
Dann gilt $H \cup L \subset A$, $H \cup L \in \mathcal{F}$, $|H| + |L| = |H \cup L| \leq r(A)$
$H' \cup L \subset B$, $H' \cup L \in \mathcal{F}$, $|H'| + |L| = |H' \cup L| \leq r(B)$
und also

$$r(A \cup B) + r(A \cap B) = |H| + |L| + |H'| + |L| \leq r(A) + r(B)$$

(ii) Seien $A, B \subset E$ mit $\sum_{e\in A} x_e = r(A)$ bzw. $\sum_{e\in B} x_e = r(B)$. Dann:

$$\begin{aligned} \sum_{e\in A\cup B} x_e + \sum_{e\in A\cap B} x_e &\leq r(A \cup B) + r(A \cap B) \\ &\leq r(A) + r(B) \\ &= \sum_{e\in A} x_e + \sum_{e\in B} x_e \end{aligned}$$

Da die Summen links und rechts offensichtlich identisch sind, gilt "=" überall.

$$\begin{aligned} &\Rightarrow \sum_{e\in A\cup B} x_e + \sum_{e\in A\cap B} x_e = r(A \cup B) + r(A \cap B) \\ &\Rightarrow \qquad\qquad \sum_{e\in A\cup B} x_e = r(A \cup B)) \end{aligned}$$

und weil für alle $U \subset E$ gilt, dass $\sum_{e \in U} x_e \leq r(U)$, ist auch

$$\sum_{e \in A \cap B} x_e = r(A \cap B)$$

(iii) Übung (Beachte: Die Implikation gilt nur in einer Richtung)

♣

Zum Abschluss führen wir noch die Definition von *Span* $sp(F)$ und die Definition eines Circuits ein. $sp(F)$ beinhaltet alle Elemente aus der Grundmenge E, so dass die um eines dieser Elemente vergrösserte Menge F nicht mehr zu $\mathcal{F}$ gehört:

Definition 3.14 (Span) *Sei $F \in \mathcal{F}$. $sp(F)$ ist definiert durch $sp(F) = F \cup \{e \in E | F \cup e \notin \mathcal{F}\}$*

Lemma 3.15

(i) $r(sp(F)) = |F|$

(ii) $F \subset F'$ *und* $x \in sp(F) \Longrightarrow x \in sp(F')$

Beweis. Übung. ♣

Anstatt maximale unabhängige Mengen, betrachten wir nun minimale abhängige Mengen. Gegeben sei ein Matroid $M(E, \mathcal{F})$.

Definition 3.16 (Circuit) *Wir nennen eine minimale abhängige Menge $C \subset E$ einen Circuit, d. h. C ist ein Circuit, falls:*

$$C \notin \mathcal{F} \text{ und für alle } C' \subset C \text{ gilt: } C' \in \mathcal{F} \tag{3.10}$$

Lemma 3.17 *Es gilt:*

$$F \in \mathcal{F}, \quad F \cup e \notin \mathcal{F} \Rightarrow \exists! \text{ Circuit } C \subset F \cup e \tag{3.11}$$

Beweis. Sei die Aussage falsch. Dann existieren mindestens 2 Circuits $c_1 \neq C_2$ in $F \cup e$.
Sei $\bar{F} \subset F$ minimal für diese Eigenschaft. Dann enthält $\bar{F} \cup e$ 2 Circuits C_1 und C_2. Daraus folgt, dass $\bar{F} \cup e = C_1 \cup C_2$.

$C_1 \not\subset C_2, \quad C_2 \not\subset C_1 \quad \Rightarrow \quad \exists a \in C_1 \setminus C_2, \quad \exists b \in C_2 \setminus C_1$
Behauptung: $C_1 \cup C_2 \setminus \{a, b\} \in \mathcal{F}$. Dies gilt, da sonst ein Circuit C existiert, für den gilt: $C \subset C_1 \cup C_2 \setminus \{a, b\} \subset \bar{F} \cup e \setminus a = (\bar{F} \setminus a) \cup e \supset C_2$. Es folgt $|\bar{F} \setminus a| < |\bar{F}|$ enthält 2 Circuits, was ein Widerspruch zu $\bar{F}$ minimal ist!

Wir haben dann $\bar{F} \in \mathcal{F}$ maximal $\subset \bar{F} \cup e$
$C_1 \cup C_2 \setminus \{a, b\} \in \mathcal{F}$ maximal $\subset \bar{F} \cup e$
$|C_1 \cup C_2 \setminus \{a, b\}| < |\bar{F}|$; ein Widerspruch, da zwei maximale unabhängige Mengen immer die gleiche Kardinalität haben müssen (siehe Definition 3.1)
♣

Theorem 3.18 *Eine Menge $\mathcal{C}$ von Untermengen von E ist die Menge der Circuits eines Matroids genau dann, wenn*

(i) $\emptyset \notin C$.

(ii) Falls $C_1, C_2 \in \mathcal{C}$ *und* $C_1 \neq C_2$, *dann* $C_1 \not\subset C_2$.

(iii) Falls $C_1, C_2 \in \mathcal{C}$, $C_1 \neq C_2$ *und* $e \in C_1 \cup C_2$, *dann existiert* $C \in \mathcal{C}$, *so dass* $C \subseteq (C_1 \cup C_2) \setminus \{e\}$.

Beweis. (Wir zeigen nur : $\mathcal{C}$-Axiome $\Rightarrow$ $\mathcal{F}$-Axiome)
Gegeben $\mathcal{C} \subset \mathcal{P}(E)$ und sei $\mathcal{F} := \{F \subset E \mid C \not\subseteq F \quad \forall C \in \mathcal{C}\}$. Zeige: $\mathcal{F}$ erfüllt Axiome (i), (ii) der unabhängigen Menge eines Matroiden.

(i) OK !

(ii) Annahme: Sei $A \subset E$ und $J_1, J_2 \subset A$, $J_1, J_2 \in \mathcal{F}$ maximal mit $|J_1| > |J_2|$. Wähle J_1, J_2 mit $|J_1 \cap J_2|$ maximal. Dann

$$\exists\, e \in J_1 \setminus J_2 \tag{3.12}$$
$$J_2 \cup e \supset C \in \mathcal{C} \tag{3.13}$$
$$C \not\subseteq J_1 \Rightarrow \exists f \in C \setminus J_1 \tag{3.14}$$
$$J_3 = (J_2 \cup e) \setminus f \in \mathcal{F} \quad (C \text{ einziger Circuit im } J_2 \cup e) \tag{3.15}$$
$$|J_3| = |J_2| < |J_1| \tag{3.16}$$
$$|J_3 \cap J_1| > |J_2 \cap J_1| \text{ Widerspruch!} \tag{3.17}$$

♣

3.3 Problem der unabhängigen Menge maximalen Gewichts

Ein grundlegendes und einfaches Optimierungsproblem in der Matroidentheorie ist das Problem der unabhängigen Menge max. Gewichts in einem Matroid

Problem der unabhängigen Menge max. Gewichts in einem Matroid:

Gegeben: Ein Matroid $M = (E, \mathcal{F})$. Gewichte $c_e \in \mathbb{R}, e \in E$.

Gesucht: Eine unabhängige Menge $I^* \in \mathcal{F}$ maximalen Gewichts:
$c(I^*) \equiv \sum_{e \in I^*} c_e = \max\{c(I) \mid I \in \mathcal{F}\}$.

3.3.1 Ansatz: Greedy–Algorithmus

Obige Aufgabe wird mit folgendem Greedy–Algorithmus gelöst. Ein Variante davon haben wir schon in Algorithmus 2.2 im Kapitel 2.1 kennengelernt:

Algorithmus 3.1 Greedy–Algorithmus (allgemein)

```
(* Findet unabhängige Menge maximalen Gewichts *)
begin(* Greedy *)
   Ordne Elemente von E, so dass c_{e_1} ≥ ··· ≥ c_{e_m} > 0 ≥ c_{e_{m+1}} ≥ ··· ≥ c_{e_{|E|}}
   I* := ∅
   for j := 1 to m do
      if (I* ∪ e_j ∈ F) then
         I* := I* ∪ e_j
      end (* if *)
   end (* for *)
   I* ist unabhängige Menge maximalen Gewichts
end (* Greedy *)
```

Beweis der Gültigkeit. Bemerke zunächst, dass folgende Eigenschaft des Algorithmus gilt: Nach der j–ten Ausführung der **for**–Schleife ist das gegenwärtige I^* eine Basis der Menge $\{e_1, \ldots, e_j\}$. Insbesondere, falls $I^* = \{e_{i(1)}, \ldots, e_{i(p)}\}$

und $c_{e_{i(k-1)}} > c_{e_{i(k)}}$ für $1 < k \leq p$, dann ist $\{e_{i(1)}, \ldots, e_{i(k-1)}\}$ eine Basis von $\{e \mid c_e > c_{e_{i(k)}}\}$.
Sei nun $I^* = \{e_{i(1)}, \ldots, e_{i(p)}\}$ die erhaltene Lösung, wobei

$$c_{e_{i(1)}} \geq \cdots \geq c_{e_{i(p)}} > 0$$

Nehme an, I^* sei nicht optimal, d. h. es existiert ein $J = \{e_{j(1)}, \ldots, e_{j(q)}\} \in F$ mit $c(I^*) < c(J)$. Wir zeigen einen Widerspruch. Es darf angenommen werden, dass

$$c_{e_{j(1)}} \geq \cdots \geq c_{e_{j(q)}} > 0 \tag{3.18}$$

Da I^* eine Basis von $\{e_1, \ldots, e_m\}$ ist, muss gelten, dass $q = |J| \leq |I^*| = p$. Da $c(I^*) < c(J)$, muss deshalb $1 \leq k \leq q$ existieren, so dass

$$c_{e_{j(s)}} \leq c_{e_{i(s)}}, s \in \{1, \ldots, k-1\} \text{ (evtl. } \emptyset), \text{ und} \tag{3.19}$$

$$c_{e_{j(k)}} > c_{e_{i(k)}} \tag{3.20}$$

Dann ist aber $I^* = \{e_{i(1)}, \ldots, e_{i(k-1)}\}$ keine Basis von $\{e \mid c_e > c_{e_{i(k)}}\}$, denn letztere Menge enthält $J = \{e_{j(1)}, \ldots, e_{j(k)}\} \in F$, ein Widerspruch. ♣

3.3.2 Ansatz: Lineare Programmierung

Anstatt das Problem der unabhängigen Menge maximalen Gewichts mit dem Greedy–Algorithmus zu lösen, versuchen wir hier dasselbe Problem mittels Linearer Programmierung ebenfalls zu lösen:

$$\max \quad c^T x \tag{3.21}$$

$$x \in P \tag{3.22}$$

Ein LP-Algorithmus (z. B. der Simplex Algorithmus) liefert optimale Lösungen. Wir interressieren uns für optimale Lösungen, die Eckpunkte von P sind (Simplex liefert einen Eckpunkt).

Lemma 3.19

1. $\forall F \in \mathcal{F}$ *gilt* $I_F \in P$
2. $\forall x \in \{0,1\}^E, \quad x \in P \Rightarrow x$ *ist Inzidenzvektor einer* $F \in \mathcal{F}$

Beweis. Übung ♣

Als nächstes wenden wir uns der Frage der Ganzzahligkeit zu:

1. Sind alle Eckpunkte von P 0-1 Vektoren ?
2. Ist P ganzzahlig ?

Die Antwort auf beide Fragen ist **Ja**, wie wir im nächsten Theorem beweisen werden:

Theorem 3.20 *Der Polyeder $P = \{x \in \mathbb{R}_+^E | Mx \leq r\}$ ist ganzzahlig.*

Bemerkung 3.21 Theorem 3.20 begründet diesen zweiten Ansatz.

Beweis. Sei $\overline{x}$ ein Eckpunkt von P. Dann $\overline{x}$ ist einzige Lösung eines Untersystems von Gleichungen $\overline{M}\overline{x} = \overline{r}$ induziert durch eine Teilmenge von Zeilen in M.

$$\text{Sei } I := [I_{A_1}, I_{A_2}, \ldots, I_{A_n}] := \begin{cases} I_{A_1}^T x & = r(A_1) \\ \vdots & \\ I_{A_n}^T x & = r(A_n) \end{cases} \tag{3.23}$$

Dann ist I ein minimales Untersystem von $\overline{M}\overline{x} = \overline{r}$ mit $\overline{x}$ als einziger Lösung. (Allgemein existieren mehrere solcher Systeme.) Für den Beweis benötigen wir das sogenannte Ketten–Lemma:

Lemma 3.22 (Ketten-Lemma) *Im obigen Kontext gilt: Es exisitiert ein System $I = [I_{A_1}, I_{A_2}, \ldots, I_{A_n}]$ mit $A_1 \subsetneq A_2 \subsetneq A_3 \ldots \subsetneq A_n$.*

Mit Hilfe dieses Lemmas können wir den Beweis zu Ende führen. Da die Inklusionen im Ketten–Lemma strikt sind und $n = |E|$, gilt:

$$|A_i| = i \quad 1 \leq i \leq n \tag{3.24}$$

Folgendes Gleichungssystem $\bar{I}$ ist äquivalent zu I:

$$\overline{I} \begin{cases} I_{A_1}^T x & = r(A_1) \\ (I_{A_2} - I_{A_1})^T x & = r(A_2) - r(A_1) \\ \vdots & \vdots \\ (I_{A_n} - I_{A_{n-1}})^T x & = r(A_n) - r(A_{n-1}) \end{cases} \tag{3.25}$$

Dieses Gleichungssystem hat auf der linken Seite eine Permutationsmatrix und einen ganzzahligen Vektor auf der rechten Seite, d. h. es sieht so aus:

$$\begin{bmatrix} \text{Permutation} \\ \\ \text{Matrix} \end{bmatrix} \begin{bmatrix} \\ x \\ \\ \end{bmatrix} = \begin{bmatrix} \\ \\ \end{bmatrix} \leftarrow \text{ganz} \tag{3.26}$$

Offensichtlich ist die einzige Lösung $\bar{x}$ somit ganzzahlig. ♣

Beweis des Ketten–Lemma 3.22. Wir beweisen das Lemma durch Widerspruch. Nehme an das Lemma sei falsch.

Für ein System $I = [A_1, A_2, \ldots, A_{|E|}]$ sei $k(I)$ die Kardinalität der längsten Kette $A_1 \subset A_2 \subset \cdots \subset A_{k(I)-1} \subset A_{k(I)} \not\subseteq A_{k(I)+1}, \ldots, A_{|E|}$ (eventuell nach Umnumerierung). Für $r > k(I)$ sei

$$j(A_r) := \max\{j \mid A_r \not\subseteq A_j\} \qquad \text{(Beachte, dass diese Menge nicht leer ist)} \tag{3.27}$$

$$j(I) := \min\{j(A(r)) \mid r > k(I)\} \tag{3.28}$$

Sei nun I mit $k(I)$ maximal und zweitens (falls es mehrere I's mit gleichem $k(I)$ gibt) sei $j(I)$ minimal. Zur Vereinfachung verwenden wir folgende Notation: $k := k(I),\ j := j(I)$ und $r > k$ mit $j(A(r)) = j$.

Es gilt $A_r \cap A_j \neq \emptyset$, sonst wäre das System

$$\begin{aligned} I' :&= I \setminus A_r \cup (A_j \cup A_r) \\ &= [A_1, \ldots, A_j, A_j \cup A_r, A_{j+1}, \ldots, A_{r-1}, A_{r+1}, \ldots, A_{|E|}] \end{aligned} \tag{3.29}$$

ein äquivalentes System mit $k(I') = k(I) + 1$, was im Widerspruch zu $k(I)$ maximal steht. Zur Verdeutlichung sei auf die Figur 3.1 verwiesen.

Definiere also $I'' := [I \setminus \{A_j, A_r\}] \cup \{A_j \cap A_r, A_j \cup A_r\}$. Dann ist $k(I'') \geq k(I)$, da $I'' \supset$ Kette $A_1 \subset \cdots \subset A_{j-1} \subset A_j \cup A_r \subset A_{j+1} \subset \ldots A_k$ ist. Wir unterscheiden nun zwei Fälle:

(1) $A_j \cap A_r$ passt in die Kette (siehe Figur b)
Daraus folgt, dass $k(I'') > k(I)$, was im Widerspruch zu $k(I)$ maximal steht.

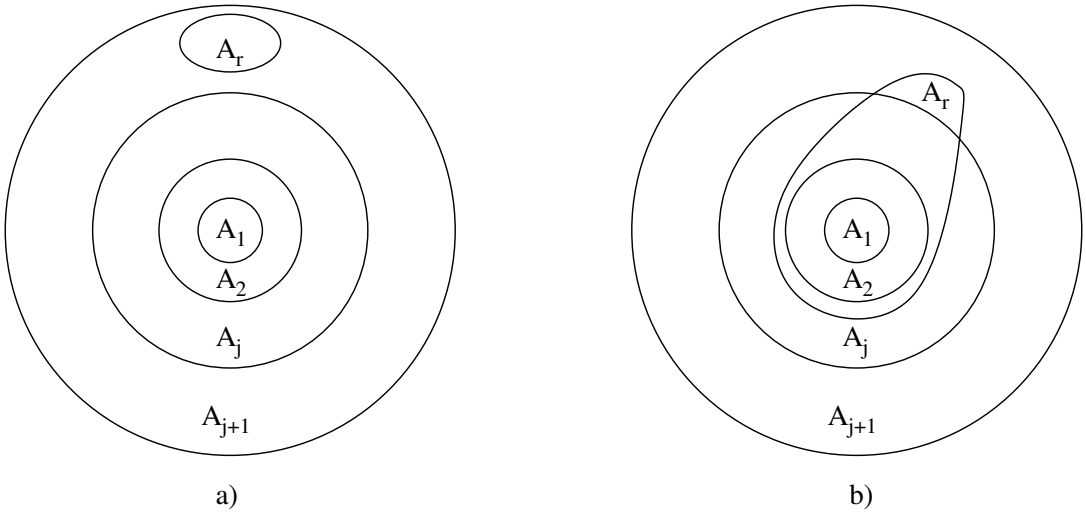


Abbildung 3.1: a)$A_r \cap A_j = \emptyset$ b)$A_r \cap A_j \neq \emptyset$

(2) $A_j \cap A_r$ passt nicht in die Kette.
Es gilt dann $k(I'') = k(I)$ und $j(I'') < j(I)$. Wegen $j(A_j \cap A_r) < j(I) = j$ haben wir einen Widerspruch zur Minimalität von $j(I)$.

♣

Wir führen nun einen zweiten Beweis zu Theorem 3.20. Hier ziegen wir, dass die LP-Lösung der durch den Greedy–Algorithmus gefundenen Lösung entspricht.

Zweiter Beweis von Theorem 3.20. Wir benutzen folgendes Lemma:

Lemma 3.23 *Gegeben sei ein Polytop $P = \{x \in \mathbb{R}^n | Ax \leq b,\ x \geq 0\}$. Falls für alle $c \in \mathbb{Z}^n$ der Zielfunktionswert $max_{x \in P}\ c^T x \in \mathbb{Z}$ (also ganzzahlig) ist, dann ist auch P ganzzahlig.*

Beweis. Sei x' ein Extremalpunkt, x' nicht integral, d. h. mindestens eine Komponente $x'_j \notin \mathbb{Z}$. Aus der LP-Theorie ist bekannt, dass

(i) $\exists\, \tilde{c} \in \mathbb{Z}^n$, so dass x' einzige Lösung der Funktion $max_{x \in P}\ \tilde{c}^T x$

(ii) Sei $c' = \tilde{c} + \frac{1}{q}e_j$. Für ein genügend grosses $q \in \mathbb{N}$ ist x' auch die einzige Lösung zu $max_{x\in P}\, c'^T x$.

Folglich ist x' einzige Lösung von $max_{x\in P}\, qc'^T x$ und $max_{x\in P}\, q\tilde{c}^T x$, wobei sowohl $qc' \in \mathbb{Z}^n$, als auch $q\tilde{c} \in \mathbb{Z}^n$. Schriebe nun die Gleichung in (ii) als $qc' = q\tilde{c} + e_j$. Dann ist

$$qc'^T x' - q\tilde{c}^T x' = (qc' - q\tilde{c})^T x' = e_j^T x' = x'_j \notin \mathbb{Z}$$

Daraus folgt, dass mindestens einer der beiden Ausdrücke $qc'^T x'$ oder $q\tilde{c}^T x'$ fraktional sein muss. Wir haben also gezeigt, dass

$$P \text{ nicht ganzzahlig} \quad \Rightarrow \quad \exists\, c \in \mathbb{Z}^n \text{ mit } max\{c^T x \mid x \in P\} \notin \mathbb{Z}$$

♣

Um dieses Lemma für das Matroiden-Polytop $P = \{x \in \mathbb{R}^n_+ | Mx \leq r\}$ zu benutzen, zeigen wir, dass für alle $c \in \mathbb{Z}^n$ die Greedy Lösung optimal für $max_{x\in P}\, c^T x$ ist. Um das zu beweisen, findet man eine duale Lösung mit gleichem Zielfunktionswert:

$$\begin{array}{lll|lll} \max & c^T x & & \min & y^T r & \\ & Mx \leq r & & & y^T M \geq c^T & \\ & x \geq 0 & & & y \geq 0 & \end{array} \tag{3.30}$$

Unser Ziel ist es jetzt, mit gegebener Greedy Lösung eine entsprechende duale Lösung zu erzeugen. Dazu verwenden wir den Span $sp(F)$ (siehe Definition 3.14). Der Greedy-Algorithmus erzeugt eine Folge $e_{i_1}\, e_{i_2}\, \ldots$ und unabängige Mengen $F_k := \{e_{i_1}\, \ldots\, e_{i_k}\}$ mit den Elementen $e_1, e_2, e_3, \ldots, e_m, \ldots$, so dass $c_{e_1} \geq c_{e_2} \geq c_{e_3} \geq \cdots \geq c_{e_m} \geq 0 \geq \ldots$

Der Algorithmus, der diese Folge berechnet, lautet:
Initialisiere $F = \emptyset$ und $j := 0$
for $i = 1$ **to** m **do**
  if $F \cup e_i \in \mathcal{F}$ **then**
    $F := F \cup e_i$
    $j := j + 1$
    $e_{i_j} := e_i$
    $F_j := F$

Bemerkung 3.24 Falls $F = F_j$ am Anfang der Schleife i ist und e_i nicht genommen wird, d. h. $F_j \cup e_i \notin \mathcal{F}$, dann ist $e_i \in sp(F_j)$, $j(i) := j$ (j wird nicht erhöht) und somit $j < i$. Da $F_1 \subset F_2 \subset \cdots \subset F_j \subset F_{j+1} \subset \ldots$, folgt weiter, dass $e_i \in sp(F_k)$ für alle $k \geq j$

Sei $\{e_{i_1}, e_{i_2}, \ldots, e_{i_s}\}$ die Greedy-Lösung des Problems mit $F_k := \{e_{i_1}, e_{i_2}, \ldots, e_{i_k}\}$ und $k = 1, \ldots, s$ und $e_{i_j} \notin F_l \quad \forall l < j$ und $e_{i_j} \in F_l \quad \forall j \leq l \leq s$. Somit folgt aus Bemerkung 3.24, dass falls $e_i \notin F$, dann ist $e_i \in sp(F_l) \quad \forall j(i) \leq l \leq s$.

Wir definieren folgende duale Lösung y und zeigen, dass die beiden Werte max $c^T x$ und min $y^T r$ in (3.30) übereinstimmen. Sei y also:

$$y_{sp(F_k)} := c_{e_{i_k}} - c_{e_{i_{k+1}}} \quad \text{für } k = 1, \ldots, s-1 \tag{3.31}$$

$$y_{sp(F_s)} := c_{e_{i_s}} \tag{3.32}$$

$$y_a := 0 \quad sonst \tag{3.33}$$

y ist zulässig, da

1. Falls, $e_i = e_{i_j} \in F$ gilt:

$$y^T \cdot M^{(i)} = \sum_{k=j}^{s} y_{sp(F_k)} = c_{e_j} = c_{e_i} \tag{3.34}$$

2. Falls $e_i \notin F$ dann ist $e_i \in sp(F_{j(i)})$ und $y^T \cdot M^{(i)} \geq y^T \cdot M^{j(i)} = c_{j(i)} \geq c_i$, weil $j(i) < i$

Schlussendlich ist

$$y^T r = (c_{e_{i_1}} - c_{e_{i_2}}) \cdot 1 + (c_{e_{i_2}} - c_{e_{i_3}}) \cdot 2 + \cdots + (c_{e_{i_{s-1}}} - c_{e_{i_s}}) \cdot (s-1) + c_{e_{i_s}} \cdot s \tag{3.35}$$

$$= c_{e_{i_1}} + c_{e_{i_2}} + \cdots + c_{e_{i_k}} \quad \text{weil } r(sp(F_k)) = |F_k| = k \tag{3.36}$$

♣

Beispiel 3.25 Betrachten wir den Graphen $G = (V, E)$, wie er in Figur 3.2 zu sehen ist. Die Kantenmenge ist $E = \{e_1, e_2, \ldots, e_7\}$ mit den entsprechenden Gewichten $c_{e_1} \geq \cdots \geq c_{e_7}$
Die Greedy Lösung für das Problem eines aufspannenden Baumes ist $\{e_1, e_2, e_4, e_5, e_7\}$

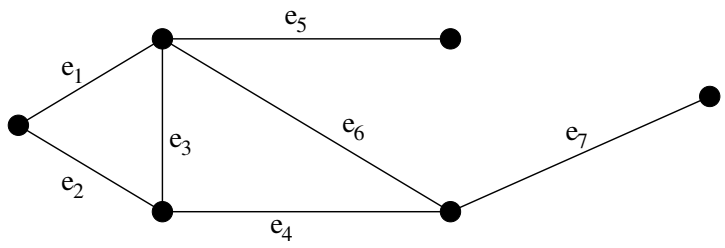


Abbildung 3.2: Beispiel 3.25

$$\begin{array}{llll} Max & c^T x & Min & y^T r \\ & Mx \leq r & & y^T M \geq c^T \\ & x \geq 0 & & y \geq 0 \end{array}$$

$$\left[\begin{array}{c} \mathrm{M} \\ \\ \\ \end{array}\right] \left[\begin{array}{c} x \end{array}\right] \leq \left[\begin{array}{c} r \\ \\ \\ \end{array}\right]$$

	e_1	e_2	e_3	e_4	e_5	e_6	e_7	r	y
I sp(F_1)	1							1	$c_{e_1} - c_{e_2}$
I sp(F_2)	1	1	**1**					2	$c_{e_2} - c_{e_4}$
I sp(F_3)	1	1	**1**	1		**1**		3	$c_{e_4} - c_{e_5}$
I sp(F_4)	1	1	**1**	1	1	**1**		4	$c_{e_5} - c_{e_7}$
I sp(F_5)	1	1	**1**	1	1	**1**	1	5	c_{e_7}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

$y \equiv 0$ sonst
$\Rightarrow y^T \cdot k_i \quad \geq \quad c_{e_i} \quad \forall$ Kolonnen k_i
$\Rightarrow y^T \cdot r = c_{e_1} + c_{e_2} + c_{e_4} + c_{e_5} + c_{e_7}$ $\diamond$

3.4 Unabhängigkeitssysteme – Exkurs

Definition 3.26 *Ein Unabhängigkeitssystem $N = (E, F)$ ist gegeben durch*

(i) Eine endliche Grundmenge E.

(ii) Eine Familie $F(\neq \emptyset)$ von Teilmengen von E, wiederum unabhängige Mengen genannt, so dass

$$\textit{Falls } I \in F \textit{ und } J \subseteq I, \textit{ dann } J \in F \textit{ (insb. } \emptyset \in F) \tag{3.37}$$

Bemerke, dass Unabhängigkeitssysteme allgemeiner als Matroide sind, da Axiom (iib) (oder (ii)) nicht verlangt wird. Allerdings verlieren solche Systeme die angenehmen Eigenschaften der Matroide. So lässt sich z. B. das Problem der unabhängigen Menge maximalen Gewichts im allgemeinen *nicht* mit dem Greedy–Algorithmus lösen. Es gilt sogar:

Satz 3.27 *Sei $N = (E, F)$ ein Unabhängigkeitssystem. Der Greedy–Algorithmus löst das Problem einer unabhängigen Menge maximalen Gewichts in $N = (E, F)$ für jede Gewichtung c dann und nur dann, wenn $N = (E, F)$ ein Matroid ist.*

Beweis. Es bleibt zu beweisen, dass falls $N = (E, F)$ *kein* Matroid ist, dann eine Gewichtung c existiert, für die der Greedy–Algorithmus keine unabhängige Menge maximalen Gewichts findet. Sei $N = (E, F)$ kein Matroid, dann existiert $I, J \in F$ mit $|I| < |J|$ und $I \cup j \notin F$ für alle $j \in J \setminus I$. Sei $\varepsilon < 1/|E|$ und $c_e := 1$ für $e \in I, c_e := 1 - \varepsilon$ für $e \in J \setminus I$ und $c_e := -1$ sonst. Der Greedy–Algorithmus findet I als Lösung mit $c(I) = |I|$. Es gilt aber $c(J) = |J| - |J \setminus I| \cdot \varepsilon > |J| - 1 \geq |I|$. I ist also nicht die unabhängige Menge maximalen Gewichts. ♣

Ein Beispiel eines Unabhängigkeitssystems ist durch die Familie aller unabhängigen *Knotenmengen* in einem Graphen gegeben (Grundmenge = Knoten). In einem ungerichteten Graphen $G = (V, U)$ mit Knotenmenge V und Kantenmenge U ist eine Knoten(teil)menge $I \subseteq V$ *unabhängig*, falls keine zwei Knoten von I mit einer Kante verbunden, d. h. benachbart sind.

Beispiel 3.28 Abbildung 3.3 zeigt einen Graphen und zwei unabhängige Mengen I und J in diesem Graphen. Offensichtlich gilt Axiom (iib) nicht: I kann nicht vergrössert werden. Zudem funktioniert auch der Greedy–Algorithmus nicht, um eine unabhängige Knotenmenge maximalen Gewichts zu bestimmen: Mit den in der Abbildung 3.3 angegebenen Gewichten ist I die Greedy–Lösung (Gewicht 3) und J die Lösung maximalen Gewichts (Gewicht 2 + 2 = 4).

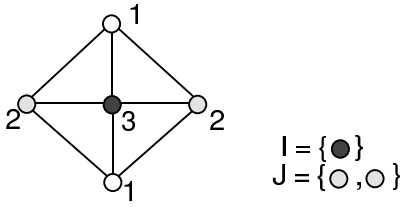


Abbildung 3.3: Unabhängige Mengen

◇

3.5 Durchschnitt von zwei Matroiden

Wie das Problem des Waldes maximalen Gewichts ein Spezialfall des Problems der unabhängigen Menge maximalen Gewichts in einem Matroid darstellt, ist das Problem des Branching maximalen Gewichts ein Spezialfall des Problems des Durchschnitts maximalen Gewichts zweier Matroiden. Wir führen dieses letzte Problem ein und begründen die eben gemachte Aussage.

Definition 3.29 (Durchschnitt) *Seien $M_a = (E, \mathcal{F}_a)$ und $M_b = (E, \mathcal{F}_b)$ zwei Matroide. Ein Durchschnitt $I \subseteq E$ ist eine Menge, die unabhängig in beiden Matroiden ist, d. h. $I \in \mathcal{F}_a \cap \mathcal{F}_b$.*

Problem des Durchschnitts maximalen Gewichts zweier Matroiden:

Gegeben: Zwei Matroide $M_a = (E, \mathcal{F}_a)$ und $M_b = (E, \mathcal{F}_b)$. Gewichte $c_e \in \mathbb{R}, e \in E$.

Gesucht: Ein Durchschnitt $I^* \in \mathcal{F}_a \cap \mathcal{F}_b$ maximalen Gewichts:
$c(I^*) \equiv \sum_{e \in I^*} c_e = \max\{c(I) \mid I \in \mathcal{F}_a \cap \mathcal{F}_b\}$.

Bemerkung 3.30 Offensichtlich ist das Problem der unabhängigen Menge maximalen Gewichts in einem Matroid M ein Spezialfall des obigen Problems: Wähle $M_a = M, M_b = (E, \mathcal{F}_b)$, wobei $\mathcal{F}_b = \{I \subseteq E\}$ die Potenzmenge von E ist (M_b ist das sog. freie oder triviale Matroid).

Satz 3.31 *Das Problem des Branchings maximalen Gewichts in $G = (V, E)$ ist das Problem des Durchschnitts maximalen Gewichts der zwei Matroide:*

$$M_a = (E, \mathcal{F}_a): \text{ graphisches Matroid von } G, \tag{3.38}$$
$$M_b = (E, \mathcal{F}_b): \text{ unitäres Partitionsmatroid,} \tag{3.39}$$

bzgl. der Partition $E = \cup\{E_v \mid v \in V\}$, wobei $E_v := \{e \in E \mid h(e) = v\}$ für alle v.

Beweis. Folgt unmittelbar aus der Definition eines Branchings und der graphischen, bzw. unitären Partitionsmatroiden (vgl. Kapitel 3.1). ♣

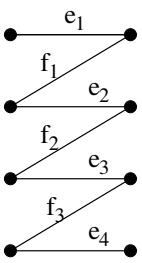
Es gibt effiziente Algorithmen zur Lösung des Problems des Durchschnitts maximalen Gewichts zweier (beliebiger) Matroiden. Wir verweisen an dieser Stelle auf Kapitel 3.5.2. Wir wenden uns zuerst dem ungewichteten Problem zu und versuchen einen Durchschnitt maximaler Kardinalität zu finden.

3.5.1 Unabhängige Menge maximaler Kardinalität zweier Matroide

Seien $M_1(E, \mathcal{F}_1)$ und $M_2(E, \mathcal{F}_2)$ zwei Matroide. Wir suchen nun $J \subset E$, $J \in \mathcal{F}_1 \cap \mathcal{F}_2$ mit $|J|$ maximal.

Das Problem des Matchings maximaler Kardinalität in einem bipartiten Graphen G(S∪T,E) ist eine Instanz dieses Problems ist. Das Konzept der vergrössernden Wege spielt eine zentrale Rolle in der Lösung des Matching–Problems. Unser Ziel ist es, in diesem Abschnitt das Problem für zwei beliebige Matroiden beruhend auf einem ähnlichen Ansatz zu lösen.

Bemerkung 3.32 Zur Erinnerung: Ein vergrössernder Weg bzgl. eines Matchings J in einem bipartiten Graphen ist eine Kantenfolge $W = (e_1, f_1, e_2, f_2, \ldots, e_m, f_m, e_{m+$ die folgende Bedingungen, die sogenannten VW–Bedingungen erfüllt (siehe auch Figur 3.4):



$$\begin{aligned} e_i &\notin J \\ f_i &\in J \\ J \cup e_1 &\in \mathcal{F}_2 \\ J \cup e_{m+1} &\in \mathcal{F}_1 \\ (J \cup e_i) \setminus f_i &\in \mathcal{F}_1 \\ (J \cup e_{i+1}) \setminus f_i &\in \mathcal{F}_2 \end{aligned} \tag{3.40}$$

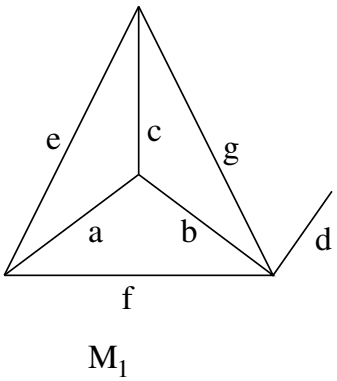
Abbildung 3.4: Vergrössernder Weg: Links steht $\mathcal{F}_2$, rechts $\mathcal{F}_1$. Die Knoten links entsprechen den S–Knoten, diejenigen rechts den T–Knoten

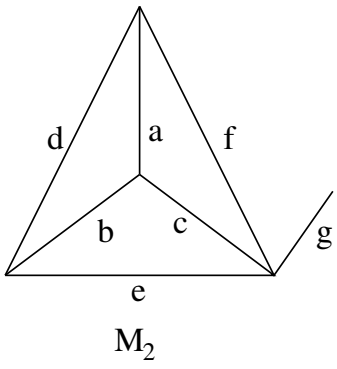
wobei $\mathcal{F}_1$ die Famile der Kantenmengen ist, die in T disjunkt sind ($\mathcal{F}_2$ in S). Der vergrössernde Weg J' is dann

$$J' := J \triangle W \in \mathcal{F}_1 \cap \mathcal{F}_2 \qquad \text{(symmetrische Differenz)} \tag{3.41}$$

Beachte, dass ein solcher „vergrössernder Weg“ J' die VW–Bedingungen (3.40) erfüllt und für ein beliebiges Paar von Matroiden definiert werden könnte.

Beispiel 3.33 Betrachte die beiden folgenden graphischen Matroide mit gemeinsamer Grundmenge $E = \{a, b, c, d, e, f, g\}$. Unabhängige Mengen entsprechen Kanten, die keinen Kreis bilden. Beachte, dass $J = \{a, b, c\}$, in beiden Matroiden unabhängig ist. Ist J maximaler Kardinalität mit dieser Eigenschaft? Unser Ziel ist es ein J zu finden, das sowohl in M_1 als auch in M_2 unabhängig ist.





◇

Um einen vergrössernden Weg bezüglich J zu finden, erzeugen wir folgenden gerichteteten Hilfsgraphen $H(M_1, M_2, J)$ mit erweiterter Knotenmenge $E \cup r \cup s$ und Kantenmenge $\mathcal{A}$).

(i) $es \in \mathcal{A} \quad \forall e \in E \setminus J \quad$ mit $J \cup e \in \mathcal{F}_1$

(ii) $re \in \mathcal{A} \quad \forall e \in E \setminus J \quad$ mit $J \cup e \in \mathcal{F}_2$

(iii) $ef \in \mathcal{A} \quad \forall e \in E \setminus J, f \in J \quad$ mit $J \cup e \notin \mathcal{F}_1 \quad$ und $(J \cup e) \setminus f \in \mathcal{F}_1$

(iv) $fe \in \mathcal{A} \quad \forall e \in E \setminus J, f \in J \quad$ mit $J \cup e \notin \mathcal{F}_2 \quad$ und $(J \cup e) \setminus f \in \mathcal{F}_2$

Beachte, dass ein $r - s$ gerichter Weg einem vergrössernden Weg im obigen Sinn entspricht. Dies sieht man deutlich in Abbildung 3.5.

Beispiel 3.34

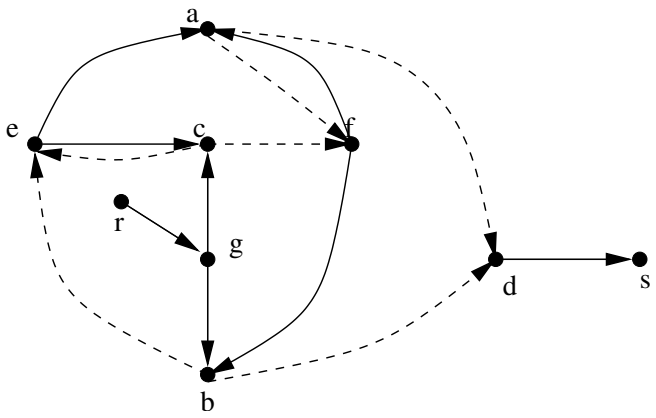


Abbildung 3.5: Hilfsgraph $H(M_1, M_2, J)$

Ein vergrössender Weg ist zum Beispiel $W = (r, g, c, f, b, e, a, d, s)$.

Beachte, dass $J \triangle (W \setminus \{r, s\})$ einen Kreis enthält und zwar in beiden Graphen! W ist aber nicht minimal und wir werden sehen, dass die VW-Bedingunegn zu schwach sind. Es sind zusätzliche Restriktionen nötig. Wie wir im Δ-Lemma sehen werden, genügt es für unser Beispiel *knotenminimale* $r - s$ *Wege* zu betrachten; wie $r - g - b - d - s$ oder $r - g - c - f - a - d - s$. $\diamond$

Theorem 3.35 (Vergrössender-Weg Theorem)

1. *Falls kein $r-s$ gerichteter Weg in $H(M_1, M_2, J)$ existiert, dann ist J maximal. Wenn $A \subseteq E$ und $\delta^+(A \cup \{r\}) = \emptyset$, dann $|J| = r_1(A) + r_2(\bar{A})$.*

2. *Falls ein $r-s$ gerichteter Weg in $H(M_1, M_2, J)$ existiert, dann ist J nicht maximal. Wenn $r, e_1, f_1, \ldots, e_m, f_m, e_{m+1}, s$ die Knotensequenz eines minimalen $r-s$ gerichteten Weges ist, dann ist $J \triangle \{e_1, f_1, \ldots, e_m, f_m, e_{m+1}\} \in \mathcal{F}_1 \cap \mathcal{F}_2$.*

Bemerkung 3.36 Die Bedeutung des zweiten Teils in der ersten Aussage des Vergrössernden Wege Theorems folgt aus folgenden Ungleichungen: Für alle $A \subset E$, $J \in \mathcal{F}_1 \cap \mathcal{F}_2$ gilt:

$$J \cap A \subset J \in \mathcal{F}_1 \text{ und } J \cap A \subset A \quad \Rightarrow \quad |J \cap A| \leq r_1(A) \tag{3.42}$$

$$J \cap \bar{A} \subset J \in \mathcal{F}_2 \text{ und } J \cap \bar{A} \subset \bar{A} \quad \Longleftrightarrow \quad |J \cap \bar{A}| \leq r_2(\bar{A}) \tag{3.43}$$

Somit ist $|J| = |J \cap A| + |J \cap \bar{A}| \leq r_1(A) + r_2(\bar{A})$. Gibt es für ein J ein bestimmtes A, so dass diese letzte Ungleichung mit Gleichheit gilt, dann ist J maximaler Kardinalität in $\mathcal{F}_1 \cap \mathcal{F}_2$ und A liefert einen Beweis dafür!

Beweis von Theorem 3.35. Wir beweisen die beiden Teile getrennt

Teil 1: Sei A die Menge der Knoten v in $H(M_1, M_2, J)$, die von r aus erreicht werden kann (d. h. es existiert mindestens ein Weg von r nach v).

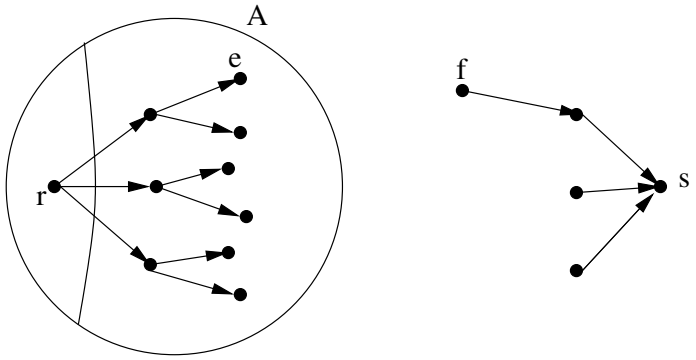


Abbildung 3.6: Erreichbare Knoten

(a) Da $e \in A \setminus J$ und $es \notin \mathcal{A}$ folgt, dass $J \cup e \notin \mathcal{F}_1$ und weiter folgt, dass $J \cup \{e\} \supset C \in \mathcal{C}_1$ (auf Grund von Def 3.16 und Lemma 3.17).

(b) Für alle f, die nicht von r erreichbar sind, gilt: $\forall f \in E \backslash A \quad ef \notin \mathcal{A}$
$J \cup e \notin \mathcal{F}_1$ analog zu a)
$\Rightarrow \quad (J \cup e) \setminus f \notin \mathcal{F}_1$ sonst ef $\in \mathcal{A} \quad \Rightarrow \quad (J \cup e) \setminus f \supset$ Circuit
$\Rightarrow \quad (J \cup e) \setminus f \supset C$ weil $J \cup e \supset$ einzigem Circuit
$\Rightarrow \quad f \notin C, \forall f \in E \setminus A$
$\Rightarrow C \subset A$

Aus (a) und (b) folgt, dass $C \subset (A \cap J) \cup e$, und somit

$$\forall e \in A \setminus J, \ (A \cap J) \cup e \notin \mathcal{F}_1 \text{ aber } (A \cap J) \in \mathcal{F}_1 \tag{3.44}$$
$$\Rightarrow \quad A \cap J \text{ unabhängig in } \mathcal{F}_1 \text{ und maximal in} A \tag{3.45}$$
$$\Rightarrow \quad r_1(A) = |A \cap J| \tag{3.46}$$

Symmetrisch behandelt man $r_2(\bar{A}) = |\bar{A} \cap J|$

Alles in allem hat man nun $|J| = |A \cap J| + |\bar{A} \cap J| = r_1(A) + r_2(\bar{A})$, womit Teil 1 bewiesen wäre.

Teil 2: Wir zeigen hier, dass Teil 2 aus dem $\triangle$-Lemma 3.37 folgt und beweisen das $\triangle$-Lemma später. Wir überprüfen also die Voraussetzungen des Δ-Lemma:

(i) Hypothesen $\triangle$-Lemma gelten für $[J, \{e_1, f_1, \ldots, e_m, f_m\}]$ und $\mathcal{F}_1$

$\Rightarrow \quad J' \in \mathcal{F}_1 \quad sp(J') = sp(J)$
$e_{m+1}s \in \mathcal{A} \quad \Rightarrow \quad J \cup e_{m+1} \in \mathcal{F}_1$
$\Rightarrow \quad e_{m+1} \notin sp_1(J) = sp_1(J')$
$\Rightarrow \quad e_{m+1} \cup J' \in \mathcal{F}_1$

Idem für $\mathcal{F}_2$ durch Symmetrie

(ii) Hypothesen des $\triangle$-Lemma : Weg Knoten-minimal (Verifikation) daraus ergibt sich die 3. Bedingung im Δ-Lemma:

$$e_i s \notin \mathcal{A} \quad \Rightarrow \quad J \cup \{e_i\} \notin \mathcal{F}_1 \text{ und } (J \cup \{e_i\}) \setminus \{f_i\} \in \mathcal{F}_1 \qquad 1 \leq i \leq k \tag{3.47}$$
$$e_i f_j \notin \mathcal{A} \quad i < j \quad \Rightarrow \quad (J \cup \{e_i\}) \setminus \{f_j\} \notin \mathcal{F}_1 \qquad 1 \leq i < j \leq k \tag{3.48}$$

♣

Lemma 3.37 (△- Lemma für [$J, \{p_1, q_1, \ldots, p_k, q_k\}$]) *Gegeben ist ein Matroid $M = (E, \mathcal{F})$. Sei $J \in \mathcal{F}$ und $p_1, q_1, \ldots, p_k, q_k$ eine Sequenz von disjunkten Elementen von E, so dass*

(i) $p_i \notin J$, $q_i \in J$ *für* $1 \leq i \leq k$

(ii) $J \cup \{p_i\} \notin \mathcal{F}$, $(J \cup \{p_i\}) \setminus \{q_i\} \in \mathcal{F}$ *für* $1 \leq i \leq k$

(iii) $(J \cup \{p_i\}) \setminus \{q_j\} \notin \mathcal{F}$ *für* $1 \leq i < j \leq k$

Dann ist $J' := J \triangle \{p_1, q_1, \ldots, p_k, q_k\} \in \mathcal{F}$ *und* $sp(J') = sp(J)$.

Beweis. Wir beweisen dieses Lemma mittels Induktion über k:

$k = 1$: Wir müssen beweisen, dass

$$(J \cup \{p_i\}) \setminus \{q_i\} \in \mathcal{F} \quad 1 \leq i \leq k \quad \Longleftrightarrow \quad J' = J \cup p_1 \setminus q_1 \in \mathcal{F} \tag{3.49}$$

Dies folgt aber direkt aus Hypothese (ii).

Wir nehmen an, dass ein $g \in sp(J) \quad g \notin sp(J')$ existiert:

$$\begin{array}{l} J \in \mathcal{F} \text{ max in } J \cup \{p_1, g\} \text{ wegen (ii) und Definition von Span} \\ J' \cup g \in \mathcal{F} \quad J' \cup g = J \cup \{p_1, g\} \setminus q_1 \\ |J| < |J' \cup g| \qquad \text{Widerspruch zu } J \text{ maximal!} \end{array} \tag{3.50}$$

Nehmen wir andererseits an, dass ein $g \notin sp(J) \quad g \in sp(J')$ existiert, so erhalten wir ebenfalls einen Widerspruch:

$$\begin{array}{l} J' \cup q_1 = J \cup p_1 \notin \mathcal{F} \quad J' \cup g \in \mathcal{F} \\ J' \in \mathcal{F} \text{ max in } J' \cup \{q_1, g\} \\ J \cup g \in \mathcal{F} \text{ max in } J' \cup \{q_1, g\} \\ |J'| < |J \cup g| \qquad \text{Widerspruch!} \end{array} \tag{3.51}$$

So ein g kann also nicht existieren und somit gilt: $sp(J') \equiv sp(J)$.

$k \geq 2$: Nehme an, das △-Lemma gilt für alle Werte kleiner k, d. h. das △-Lemma gilt für $[J, \{p_1, q_1, \ldots, p_{k-1}, q_{k-1}\}]$

1. $J \triangle \{p_1, q_1, \ldots, p_{k-1}, q_{k-1}\} = (J' \cup q_k) \setminus p_k$
   $\Rightarrow \quad \underbrace{(J' \cup q_k) \setminus p_k \in \mathcal{F}}_{\blacktriangle} \qquad \underbrace{sp((J' \cup q_k) \setminus p_k) = sp(J)}_{\bigstar}$
2. $\triangle$-Lemma für $[(J \setminus q_k),\ \{p_1, q_1, \ldots, p_{k-1}, q_{k-1}\}]$
   $(J \setminus q_k) \triangle \{p_1, q_1, \ldots, p_{k-1}, q_{k-1}\} = J' \setminus p_k$
   $\Rightarrow \quad J' \setminus p_k \in \mathcal{F} \qquad sp(J' \setminus p_k) = sp(J \setminus q_k)$

$J' \in \mathcal{F}$: $p_k \notin sp(J \setminus q_k)$ wegen $(J \cup \{p_i\}) \setminus \{q_i\} \in \mathcal{F} \quad 1 \leq i \leq k$
$\Rightarrow \quad p_k \notin sp(J' \setminus p_k) \quad \Rightarrow \underbrace{(J' \setminus p_k) \cup p_k}_{J'} \in \mathcal{F}$

$sp(J') = sp(J)$: Es bleibt zu zeigen, dass $sp(J') = sp(J) =_{\blacktriangle} sp((J' \cup q_k) \setminus p_k)$:
Benutze Δ-Lemma für k=1 und $J',\ \{q_k, p_k\}$

Hypothesen	$J' \cup q_k \notin \mathcal{F}$	$\blacktriangledown$
	$sp((J' \cup q_k) \setminus p_k)$	$\blacktriangle$

$J \cup p_k \notin \mathcal{F}$ bei $J \cup \{p_i\} \notin \mathcal{F} \quad 1 \leq i \leq k \quad \Rightarrow \quad p_k \in sp(J)$
$\Rightarrow \quad p_k \in sp((J' \cup q_k) \setminus p_k)$
$\Rightarrow \quad ((J' \cup q_k) \setminus p_k) \cup p_k \notin \mathcal{F}$
$\Rightarrow \quad J' \cup q_k \notin \mathcal{F}$

$\clubsuit$

3.5.2 Durchschnitt maximalen Gewichts zweier Matroide

Gegeben seien die Matroide $M_1(E, \mathcal{F})$ und $M_2(E, \mathcal{F})$ mit Rang r_1 bzw. r_2. Bevor wir das Problem des Durchschnitts maximalen Gewichts zweier Matroide anpacken, beweisen wir, dass der Durchschnitt zweier Matroid-Polyeder ganzzahlig ist.

Theorem 3.38 *Seien $P^1 = \{x \in \mathbb{R}^+ \mid M_1 x \leq r_1\}$ und $P^2 = \{x \in \mathbb{R}^+ \mid M_2 x \leq r_2\}$ zwei Matroid-Polyeder. Sei weiter $P^{12} = \{x \in \mathbb{R}^+ \mid M_1 x \leq r_1,\ M_2 x \leq r_2\} = P^1 \cap P^2$ der Durchschnitt dieser zwei Polyeder. Dann ist P^{12} ist ganzzahlig.*

Beweis. Sei $\overline{x}$ Extremalpunkt von P^{12}. Dann ist $\overline{x}$ die einzige Lösung eines Untersystem I von $M_1 x \leq r_1,\ M_2 x \leq r_2,\ x \geq 0$, in welchem alle Ungleichungen mit Gleichheit gelten. Dieses Untersystem I hat einen Teil I^1 im System $M_1 x \leq r_1$ und den anderen Teil I^2 im System $M_2 x \leq r_2$.

Auf Grund des Ketten-Lemmas 3.22 gibt es ein System $\overline{I}^1$ äquivalent mit I^1 und ein zu I^2 äquivalentes System $\overline{I}^2$, deren Gleichungen disjunkten Trägern entsprichen.

Die 0-1 Matrix $\overline{I} = \begin{bmatrix} \overline{I}^1 \\ \overline{I}^2 \end{bmatrix}$ hat höchstens 2 Einsen in jeder Spalte, das erste in $\overline{I}^1$ und das zweite in $\overline{I}^2$. Somit ist $\overline{I}$ total unimodular und es folgt, dass P^{12} ganzzahlig ist. ♣

Formulieren wir nochmals das Problem des Durchschnitts maximalen Gewichts. Beachte die Ähnlichkeit zu einem LP:

Problem des Durchschnitts maximalen Gewichts zweier Matroiden:

Gegeben: Zwei Matroide $M_1 = (E, \mathcal{F}_1)$ und $M_2 = (E, \mathcal{F}_2)$, sowie eine Gewichtsfunktion $c : E \to \mathbb{R}$.

Gesucht: Ein Durchschnitt $F \in \mathcal{F}_1 \cap \mathcal{F}_2$ mit $c(F) = \max_{J \in \mathcal{F}_1 \cap \mathcal{F}_2}\{c(J)\}$

Formulieren wir dieses Problem als lineares Programm:

$$\max\{c(J) | J \in \mathcal{F}_1 \cap \mathcal{F}_2\} = \max c^T x \qquad \left\{ \begin{array}{ll} I_A^T x \leq r_1(A) & \forall A \subset E \\ I_A^T x \leq r_2(A) & \forall A \subset E \\ x_e \geq 0 & \forall e \in E \end{array} \right. \tag{P}$$

Das duale Programm D von P ist dann

$$\begin{aligned} \min & \sum_{A \subset E} (r_1(A) y_A^1 + r_2(A) y_A^2) \\ & \sum_{A \subset E} (y_A^1 + y_A^2) \cdot I_A^T \geq c^T \\ & y_A^1, y_A^2 \geq 0 \qquad \forall A \subset E \end{aligned} \tag{D}$$

Sei $(\bar{y}^1, \bar{y}^2)$ eine optimale duale Lösung von (D). Definiere $c^1 := \sum_{A \subset E} y_A^1 I_A^T$ und $c^2 := c - c^1$ und formuliere folgende Probleme für $i = 1, 2$:

$$\begin{aligned} \min \sum_{A \subset E} & r_i(A) y_A^i \\ \sum_{A \subset E} & y_A^i I_A^T \geq c^i \qquad \text{(Probleme } D_i\text{)} \\ & y_A^i \geq 0 \qquad \forall A \subset E \end{aligned}$$

Lemma 3.39 *$(\bar{y}^1, \bar{y}^2)$ optimal für D $\iff$ optimal für D_i mit i=1 und 2.*

Beweis. Übung ♣

Definition 3.40 (Gewichtssplittung) *Gegeben c^1, c^2 mit $c = c^1 + c^2$. (c^1, c^2) wird eine Gewichtssplitterung genannt.*

Um das Problem des Durchschnitts maximalen Gewichts zweier Matroiden zu lösen führen wir Wege und die Länge von Wegen ein: Gegeben sei eine Gewichtssplitterung $c = c^1 + c^2$ $c \in \mathbb{R}^E$, zwei Matroide $M_1(E, \mathcal{F}_1)$ und $M_2(E, \mathcal{F}_2)$ sowie eine unabhängige Menge $J \in \mathcal{F}_1 \cap \mathcal{F}_2$.
Die Kantenmenge $\mathcal{A}$ von H wird gewichtet: $H(M_1, M_2, J, c^1, c^2)$
Sei $c_o^i := max\{c_e^i | e \notin J, J \cup e \in \mathcal{F}_i\} \quad i = 1, 2$

Wir definieren folgende Kantenlängen:

$$p_{es} = c_o^1 - c_e^1\text{:} \quad M_1\text{-Kante } es, e \notin J \; J \cup e \in \mathcal{F}_1 \tag{3.52}$$

$$p_{re} = c_o^2 - c_e^2\text{:} \quad M_2\text{-Kante } re, e \notin J \; J \cup e \in \mathcal{F}_2 \tag{3.53}$$

$$p_{ef} = -c_e^1 + c_f^1\text{: } M_1\text{-Kante } ef, e \notin J,\; f \in J \; J \cup e \notin \mathcal{F}_1,\; J \cup e \setminus f \in \mathcal{F}_1 \tag{3.54}$$

$$p_{fe} = -c_e^2 + c_f^2\text{: } M_2\text{-Kante } fe, e \notin J,\; f \in J \; J \cup e \notin \mathcal{F}_2,\; J \cup e \setminus f \in \mathcal{F}_2 \tag{3.55}$$

Gegeben ein Weg $\mathbb{W} = \{e_1, f_1, e_2, f_2, e_3\}$ in H, beachte dass

Allgemein gilt für jeden VW Weg $\mathbb{W}$
Länge $(\mathbb{W}) = c_0^2 - c_{e_1} + c_{f_1} - c_{e_2} + c_{f_2} - c_{e_3} + c_o^1$
und $J' := J \triangle \mathbb{W} \quad c(J') = c(J) -$ Länge$(\mathbb{W}) + c_o^1 + c_o^2$
Die Strategie um einen Durchschnitt maximalen Gewichts zweier Matroiden zu bilden sieht folgendermassen aus: Gegeben sei ein J; wir suchen dann einen

$r - s$ Weg $\mathbb{W}$ minimaler Länge und bilden die symmetrische Differenz $J' = J \triangle \mathbb{W}$. Es gilt folgende Aussage, die wir anschliessend (Sätze 3.42 bis 3.48) beweisen werden:

J maximales Gewicht und $\mathbb{W}$ minimale Länge $\iff$ J' maximales Gewicht (3.56)

Algorithmus 3.2 Durchschnitt maximalen Gewichts zweier Matroiden

---

Gegeben $M_1(E, \mathcal{F}_1), M_2(E, \mathcal{F}_2), \quad c : E \to \mathbb{R}$
$k := 0, J_k := \varnothing, c^1 := c, c^2 := \underline{0}$, STOP := False
While STOP = False **Do**
Construct $H(M_1, M_2, J_k, c^1, c^2)$ $\forall v \in E \cup s$, find r-v-path minimal length $:= d_v$
$\sigma_v := min\{d_v, d_s\}$
$c_v^1 := c_v^1 - \sigma_v$
$c_v^2 := c_v^2 + \sigma_v$
Construct $H(M_1, M_2, J_k, c^1, c^2)$ (i.e. update edge-length)
If $\exists$ r-s path **then**
Find r-s path P minimal length with a minimal number of arcs
$J_{k+1} := J_k \triangle P$
$k := k + 1$
Else
$J := J_p$ with $c(J_p) \geq c(J_i) \quad 1 \leq i \leq k$
STOP := True

Definition 3.41 (Zertifikat) *Eine Gewichtssplitterung (c^1, c^2) ist ein Zertifikat für $J \in \mathcal{F}_1 \cap \mathcal{F}_2$ falls für $i = 1$ und 2 gilt:*

$$c^i(J) = max\{c^i(F) | F \in \mathcal{F}_i, |F| = |J|\} \tag{3.57}$$

Theorem 3.42 *Es gilt:*

$$J_k \in \mathcal{F}_1 \cap \mathcal{F}_2 \qquad c(J_k) = \max\{c(F) | F \in \mathcal{F}_1 \cap \mathcal{F}_2, |F| = k\} \tag{3.58}$$

$$J \in \mathcal{F}_1 \cap \mathcal{F}_2 \qquad c(J) = \max\{c(F) | F \in \mathcal{F}_1 \cap \mathcal{F}_2\} \tag{3.59}$$

Lemma 3.43 (Zertifikatslemma)

$$\begin{gathered}(c^1, c^2) \textit{ ist ein Zertifikat für } J \\ \Longleftrightarrow \\ \textit{alle Kantenlängen in } H(M_1, M_2, J, c^1, c^2) \textit{ sind } \geq 0 \textit{ und} \\ \textit{für } i = 1, 2 \textit{ und } \forall\, f \in J \textit{ gilt: } c_o^i \leq c_f^i\end{gathered} \tag{3.60}$$

Beweis. Bemerke, dass erstens:

$$\begin{gathered}J \text{ optimal für seine Kardinalität} \\ \Longleftrightarrow \\ \forall \quad e \notin J \text{ und } f \in J \text{ mit } (J \cup e) \setminus f \in \mathcal{F} \Rightarrow c_e^i \leq c_f^i\end{gathered} \tag{3.61}$$

- Sei e, f mit $J \cup e \notin \mathcal{F}_i$, $(J \cup e) \setminus f \in \mathcal{F}_i$. Falls $i = 1$ ist, gilt: ef ist eine M_1 - Kante und $p_{ef} = -c_e^1 + c_f^1 \geq 0$. Falls $i = 2$ ist, gilt: fe ist eine M_2 - Kante und $p_{fe} = -c_e^2 + c_f^2 \geq 0$.
- Sei e mit $J \cup e \in \mathcal{F}_i$. Dann gilt für alle $f \in J$, dass $(J \cup e) \setminus f \in \mathcal{F}$; und wegen 3.5.2 gilt weiter, dass $c_e^i \leq c_f^i$. Es folgt, dass $c_o^i = max\{c_e^i | e \notin J, J \cup e \in \mathcal{F}_i\} \leq c_f^i$.

♣

Lemma 3.44 (Stabilitätslemma) (c^1, c^2) *ist ein Zertifikat für* $J_k \Rightarrow$ (c_{neu}^1, c_{neu}^2) *ist ein Zertifikat für* J_k

Beweis. Wir beweisen die Aussage für M_1-Kanten: Seien c^1, c^2 die Länge p und c_{neu}^1, c_{neu}^2 die Länge p'

1. Für jede Kante $uv, p'_{uv} \geq 0$
   M_1-Kante ef
   $p'_{ef} = -(c_e^1 - \sigma_e) + (c_f^1 - \sigma_f) = p_{ef} + \sigma_e - \sigma_f$
   Allgemein gilt: $d_f \leq d_e + p_{ef}$

   - $\sigma_e = d_s \Rightarrow \sigma_e - \sigma_f \geq 0 \Rightarrow p'_{ef} \geq p_{ef} \geq 0$
   - $\sigma_e = d_e \Rightarrow p'_{ef} = p_{ef} + d_e - \sigma_f \geq d_f - \sigma_f \geq 0$

2. Lemma falsch $\Rightarrow \exists f \in J_k$ mit $c^1_{o_{neu}} > c^1_{f_{neu}}$
   $c^1_o = max\{c^1_e | e \in \delta^-(s)\}$
   $\exists f \in J_k, e \notin J_k, J_k \cup e \in \mathcal{F}_1$

$$d_e = \sigma_e \Leftarrow \sigma_e < \sigma_f \begin{cases} c^1_e - \sigma_e > c^1_f - \sigma_f \\ c^1_e \leq c^1_f \end{cases}$$

   $c^1_e - \sigma_e = c^1_e - d_e > c^1_f - \sigma_f \geq c^1_f - d_s \geq$
   $[d_s \leq d_e + p_{es} = d_e + c^1_o - c^1_e]$
   $\geq c^1_f - [d_e + c^1_o - c^1_e] = c^1_e - d_e + c^1_f - c^1_o \Rightarrow c^1_o > c^1_f$

♣

Lemma 3.45 (Null-Weg-Lemma) *Für jede Kante $uv \in P$ gilt $p'_{uv} = 0$*

Beweis. Wiederum beweisen wir die Aussage für M_1-Kanten: Sei $\bar{P}$ ein $r-s$ Weg minimaler Länge für p. Wir zeigen unten, dass $p'_{uv} = 0 \ \forall uv$ in $\bar{P}$. Es folgt, dass p'-Länge$(P) \leq p'$-Länge$(\bar{P}) = 0$. Und da $p'_{uv} \geq 0 \ \forall uv$, gilt $p'_{uv} = 0 \ \forall uv \in P$. Weiter ist für alle Kanten $uv \in \bar{P}$, $d_v = d_u + p_{uv}$ und somit $d_v \leq d_s \ \forall v \in \bar{P}$.

1.

$$M_1\text{-Kante ef} \begin{cases} J_k \cup e \notin \mathcal{F}_1 \\ J_k \cup e \setminus f \in \mathcal{F}_1 \end{cases}$$
$$p'_{ef} = -c^1_{e_{neu}} + c^1_{f_{neu}}$$
$$p'_{ef} = -[c^1_e - \sigma_e] + [c^1_f - \sigma_f]$$
$$p'_{ef} = -c^1_e + c^1_f + d_e - d_f$$
$$p'_{ef} = p_{ef} + (-p_{ef}) = 0$$

2. M_1-Kante $e's$ (e' vorletzter Knoten im $\bar{P}$)
   $\delta^-(s) = \{e \notin J_k, J \cup e \in \mathcal{F}_1\}$
   $c^1_o = max\{c^1_e | e \in \delta^-(s)\}$
   $\sigma_e := min\{d_e, d_s\}$ mit $d_e =$ minimaler Länge eines r-e-Weges
   $e \in \delta^-(s) \quad \Rightarrow \quad d_s \leq d_e + p_{es} \quad \Rightarrow \quad -d_e \leq -d_s + p_{es}$ ▲
   Alte Länge $p : p_{es} = c^1_o - c^1_e$
   Neue Länge $p' : p'_{es} = c^1_{o_{neu}} - c^1_{e_{neu}}$

Behauptung $\forall e \in \delta^-(s), c^1_{e_{neu}} = c^1_e - \sigma_e \leq c^1_o - d_s$

- Falls $e \in \delta^-(s), \sigma_e = d_s$ dann
  $c^1_{e_{neu}} = c^1_e - \sigma_e = c^1_e - d_s \leq c^1_o - d_s$

- Falls $e \in \delta^-(s), \sigma_e = d_e$ dann
  $c^1_{e_{neu}} = c^1_e - \sigma_e = c^1_e - d_e \leq_{\blacktriangle} c^1_e - d_s + p_{es} = c^1_e - d_s + c^1_o - c^1_e = c^1_o - d_s$

Bemerkung: e' vorletzter Knoten in $\bar{P}$ (Fall 2)
$d_s = d_{e'} + p_{e's} \quad \Rightarrow \quad$ in $\blacktriangle \quad \Rightarrow \quad c^1_{e'_{neu}} = c^1_o - d_s$
$\Rightarrow \quad c^1_{o_{neu}} = max\{c^1_{e_{neu}} | e \in \delta^-(s)\} = c^1_{e'_{neu}}$
$\Rightarrow \quad p'_{e's} = c^1_{o_{neu}} - c^1_{e'_{neu}} = 0$

♣

Lemma 3.46 (gewichtetes $\triangle$-Lemma) *Sei $M(E, \mathcal{F})$ ein gegebenes Matroid und $c : E \to \mathbb{R}$ eine Gewichtsfunktion und $J \in \mathcal{F}$, sodass J $c-$optimal ist. Weiter seien $\{p_1, q_1, p_2, q_2, \ldots p_k, q_k\}$ disjunkt und $p_i \notin J$ und $q_i \in J$ für alle i, sodass folgende Bedingungen gelten:*

(a) $J \cup p_i \notin \mathcal{F},\ (J \cup p_i) \setminus q_i \in \mathcal{F} \qquad 1 \leq i \leq k$

(b) $c_{p_i} = c_{q_i} \qquad 1 \leq i \leq k$

(c) $c_{p_i} = c_{q_l}$ *und* $i < l \quad \Rightarrow \quad (J \cup p_i) \setminus q_l \notin \mathcal{F}$

Dann gilt: $J' = J \triangle \{p_1, q_1, \ldots p_k, q_k\} \in \mathcal{F}$ *und* J' *ist c-optimal.*

Beweis. Renummerierung, so dass $c_{p_1} \geq c_{p_2} \geq \cdots \geq c_{p_k}$ ohne Reihenfolgeänderung falls Gleichheit gilt.
$\Rightarrow$ a), b) und c) gelten immer noch:

Nach Renumerierung gilt

$$(J \cup p_i) \setminus q_l \notin \mathcal{F}\ \forall i < l \tag{3.62}$$

sonst existieren i_o und l_o mit $\bar{J} := (J \cup p_{i_o}) \setminus q_{l_o} \in \mathcal{F}$, $i_o < l_o$ und $c_{p_{i_o}} > c_{q_{l_o}}$ wegen c) und es folgt

$$c(\bar{J}) > c(J) \quad \text{Widerspruch mit } J\, c-\text{optimal.} \tag{3.63}$$

(3.62) ist 3. im $\triangle$-Lemma $\Rightarrow J' \in \mathcal{F},\ sp(J') = sp(J)$ und $c(J') = c(J)$ bei b). ♣

Korollar 3.47 *Sei $M(E, \mathcal{F})$ ein Matroid und $c : E \to \mathbb{R}$ eine Gewichtsfunktion. Sei $J \in \mathcal{F}$ $c-$optimal. Weiter seien $\{p_1, q_1, p_2, q_2, \ldots p_k, q_k\}$ disjunkt und $p_i \notin J$ und $q_i \in J$ für alle i, sodass folgende Bedingungen gelten:*

(i) $J \cup p_i \notin \mathcal{F},\ (J \cup p_i) \setminus q_i \in \mathcal{F} \qquad 1 \le i \le k$

(ii) $c_{p_i} = c_{q_i} \qquad 1 \le i \le k$

(iii) $c_{p_i} = c_{q_l}$ *und* $i < l \quad \Rightarrow \quad (J \cup p_i) \setminus q_l \notin \mathcal{F}$

(iv) $\exists\ p_{k+1}$ *mit* $J \cup p_{k+1} \in \mathcal{F}$

Dann gilt: $J' \cup p_{k+1} \in \mathcal{F}$.

Beweis. Aus (d) folgt unmittelbar, dass $p_{k+1} \notin sp(J) = sp(J')$ ♣

Lemma 3.48 (Optimalitäts-Lemma) *$J_{k+1} := J_k \triangle P$ hat maximales Gewichts für Kardinalität $k+1$ mit Zertifikat $(c^1,\ c^2)$.*

Beweis. Wir zeigen J_{k+1} M_1-optimal für c_1^1.
$P = \{r, e_1, f_1, e_2, \ldots, e_m, f_m, e_{m+1}, s\}$
$J_k \cup e_{m+1} \in \mathcal{F}_1$
$\begin{cases} c^1_{e_{m+1}} = c^1_o = & \max\{c^1_e \mid J_k \cup e \in \mathcal{F}_1\} \\ c^1_o \le c^i_f & \forall\, f \in J_k \end{cases}$
$\Rightarrow \quad J_k \cup e_{m+1}$ c-optimal für Kardinalität $k+1$ (Greedy Algorithm würde e_{m+1} wählen)
$\Rightarrow \quad c^1(J_k \triangle P) = c^1(J'_k \cup e_{m+1}) = c^1(J'_k) + c^1_{e_{m+1}} = c^1(J'_k) + c^1_{e_{m+1}} = c^1(J_k \cup e_{m+1})$
$\begin{cases} J_{k+1} \quad M_2\text{-optimal für } c^2 \\ \text{Beweis: idem} \end{cases}$

♣

3.6 Allgemeine Fakten

3.6.1 Linearisierbarkeit von Matroiden

Matrix-Matroide sind wichtige Beispiele von Matroiden. Daher stellt sich die Frage, ob alle Matroide auch von diesem Typ sind. Wir wollen diese Frage nun beantworten.
Mathematisch ausgedrückt lautet die Frage: Existiert ein Vektorraum $\mathbb{W}$ und eine Abbildung $f : E \to \mathbb{W}$, so dass

$$\forall\ A \subset E,\ A \in \mathcal{F} \iff \{f(e)|e \in A\} \text{ linear unabhängig} \tag{3.64}$$

Bemerkung 3.49 $\forall$ A $\subset$ E, r(A)=dim(f(A))

Beispiel 3.50 (Vámos Cube) Das folgende Beispiel (der sog. "Vámos Cube") zeigt, dass die Antwort auf obige Frage **negativ** ist !

$$\begin{aligned}
E &= \{1,2,3,\ldots,8\} \\
\mathcal{B} &= \binom{8}{4} \setminus \{V_1,\ldots,V_5\} \qquad (\mathcal{B} = \text{ Menge der maximalen } F \in \mathcal{F}) \\
V_1 &= \{1,2,3,4\} \\
V_2 &= \{1,2,5,6\} \\
V_3 &= \{1,2,7,8\} \\
V_4 &= \{3,4,5,6\} \\
V_5 &= \{3,4,7,8\} \\
r(V_i) &= 3 \qquad \forall 1 \leq i \leq 5
\end{aligned} \tag{3.65}$$

Übung 3.51 *Man verifiziere die Matroiden-Axiome.*

Es existiert ein $W \ni f_1,\ldots,f_8,\ f_i = f(i) \quad 1 \leq i \leq 8$, so dass $\alpha_i, \beta_i, \gamma_i \quad 1 \leq i \leq 3$ existieren mit

$$V_1 \Rightarrow f_1 + \alpha_1 f_2 + \beta_1 f_3 + \gamma_1 f_4 = 0 \tag{3.66}$$
$$V_2 \Rightarrow f_1 + \alpha_2 f_2 + \beta_2 f_5 + \gamma_2 f_6 = 0 \tag{3.67}$$
$$V_3 \Rightarrow f_1 + \alpha_3 f_2 + \beta_3 f_7 + \gamma_3 f_8 = 0 \tag{3.68}$$

Es folgt, dass

(i) $\alpha_1 = \alpha_2$: sonst:(3.66) und (3.67) $\Rightarrow f_2 \in$ Lin. Span $\{f_3, f_4, f_5, f_6\}$ und folglich

$$4 = dim(f_2, f_3, f_4, f_5, f_6) = dim(f_3, f_4, f_5, f_6) = 3 \tag{3.69}$$

(ii) $\alpha_1 = \alpha_3$ idem mit (3.66) und (3.68)

Aus diesen Überlegungen folgt, dass $\alpha_2 = \alpha_3$ sein muss. Bildet man die Differenz von (3.67) und (3.68) so folgt, dass $\{f_5, f_6, f_7, f_8\}$ abhängig sind — ein Widerspruch!

3.6.2 Matroiden-Axiome

Sei $M(E, \mathcal{F})$ mit $\mathcal{F} = \{\text{unabhängige Mengen}\} \subset \mathcal{P}(E)$. Wir formulieren folgende Axiome:

$\mathcal{F}$-Axiome Für alle I, J und $\mathcal{F} \neq \emptyset$ gilt

(f_1) $I \subset J \in \mathcal{F} \quad \Rightarrow \quad I \in \mathcal{F}$

(f_2) $I, J \in \mathcal{F} \quad |I| < |J| \quad \Rightarrow \quad \exists e \in J \setminus I \quad I \cup e \in \mathcal{F}$

$\mathcal{B}$-Axiome Eine Basis ist eine maximale unabhängige Menge. Sei $\mathcal{B} = \{\text{Basen}\} \subset \mathcal{P}(E)$. Für alle $B_1, B_2 \in \mathcal{B}$ $B_1 \neq B_2$ gilt:

(b_1) $B_1 \not\subset B_2$

(b_2) Für alle $z \in B_1 \setminus B_2$ existiert ein $t \in B_2 \setminus B_1$ mit $(B_1 \cup t) \setminus z \in \mathcal{B}$

$\mathcal{C}$-Axiome Ein Circuit ist eine minimale abhängige Menge. Sei $\mathcal{C} = \{\text{Cuircuits}\} \subset \mathcal{P}(E)$. Für alle $C_1, C_2 \in \mathcal{C}$ $C_1 \neq C_2$ gilt:

(c_1) $C_1 \not\subset C_2$ und $\emptyset \notin \mathcal{C}$

(c_2) Für alle $e \in E$ existiert ein $C_3 \in \mathcal{C}$ mit $C_3 \subset (C_1 \cup C_2) \setminus e$

$\mathcal{H}$-Axiome Für alle $H_1, H_2 \in \mathcal{H} \quad H_1 \neq H_2$ gilt:

(h_1) $H_1 \not\subset H_2 \quad E \notin \mathcal{H}$

(h_2) $x \notin H_1 \cup H_2 \Rightarrow \exists H_3 \in \mathcal{H}, H_3 \supseteq (H_1 \cap H_2) \cup x$

Definition 3.52 (Gute Familie) *Sei $E = \{1, 2, \ldots, n\}$. $\mathcal{G} \subset \mathcal{P}(E)$ ist eine gute Familie, falls*

(i) $\forall G \in \mathcal{G}: \quad |G| = \lfloor \frac{n}{2} \rfloor$

(ii) $\forall G_1, G_2 \in \mathcal{G}: \quad |G_1 \cap G_2| \leq \lfloor \frac{n}{2} \rfloor - 2$

Lemma 3.53 *Gegeben sei eine gute Familie $\mathcal{G}$. Sei weiter*

$$\tilde{\mathcal{G}} = \{H \subset E \mid |H| = \lfloor \frac{n}{2} \rfloor - 1,\ H \not\subset G\, \forall G \in \mathcal{G}\}. \tag{3.70}$$

Dann gilt: $\mathcal{H} = \mathcal{G} \cup \tilde{\mathcal{G}}$ erfüllt die $\mathcal{H}$-Axiome.

Korollar 3.54 *Jede gute Familie liefert einen Matroiden.*

Bemerkung 3.55 Jede Unterfamilie $\mathcal{G}'$ einer guten Familie ist eine gute Familie.

Korollar 3.56 *Eine gute Familie $\mathcal{G}$ liefert $2^{|\mathcal{G}|}$ Matroiden.*

Theorem 3.57 *Es existieren gute Familien $\mathcal{G}$ mit $|\mathcal{G}| \geq \frac{\binom{n}{\lfloor \frac{n}{2} \rfloor}}{2n}$*

Beweis. $k := [\log_2 n] + 1$
Sei A die $n \times k$ Matrix mit $A_i =$ Zeile $i = bin(i) =$ binäre Darstellung von i

$$A = \begin{pmatrix} .. \\ bin(i) \\ .. \end{pmatrix}$$

$U_j := \{v \in \{0,1\}^n | v^T \cdot \underline{1} = \lfloor \frac{n}{2} \rfloor \quad v^T \cdot A = bin(j)\}$
$U_i \cap U_j = \emptyset \quad i \neq j$

1. $\sum |U_j| = \binom{n}{\lfloor \frac{n}{2} \rfloor}$
2. 2^kmögliche Werte für j

1) & 2) $\Rightarrow \exists\ j_0$ mit $|U_{j_0}| \geq \frac{\binom{n}{\lfloor \frac{n}{2} \rfloor}}{2^k} \geq \frac{\binom{n}{\lfloor \frac{n}{2} \rfloor}}{2n}$
Definiere $\mathcal{G} := U_{j_0}$ ♣

Bemerkung 3.58 $\forall j \quad U_j$ ist eine gute Familie $\forall x, y \in U_j, \delta(x,y) > 2$
$x^T A = y^T A \Rightarrow (x+y)^T A = \underline{0}$.
Falls $\delta(x,y) = 2$
$y = x + e_i + e_j$
$(x+y)^T A = (e_i + e_j)^T A = A_i + A_j \neq 0$

Kapitel 4

Matching maximaler Kardinalität

4.1 Matching und vergrössernde Wege

Definition 4.1 (Matching) *Sei $G = (V, E)$ ein ungerichteter Graph. Ein Matching von G ist eine Kantenmenge M, so dass keine zwei Kanten von M einen gemeinsamen Endknoten besitzen.*

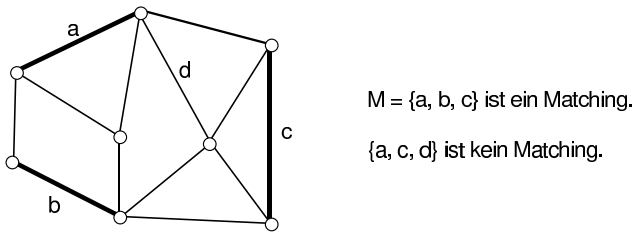


Abbildung 4.1: Beispiel eines Matchings

In diesem Kapitel wird folgendes Problem betrachtet:

Problem des Matching maximaler Kardinalität:

Gegeben: $G = (V, E)$ ungerichtet.

Gesucht: Ein Matching M von G maximaler Kardinalität $|M|$.

In diesem Zusammenhang wird das Konzept der vergrössernden Wege benutzt:

Definition 4.2 (Vergrössernde Wege) *Sei M ein Matching von $G = (V, E)$. Ein Knoten heisst* exponiert *bezüglich M, falls keine Kante von M mit ihm inzidiert, und von M* bedeckt *andernfalls. Ein* alternierender Weg *bezüglich M ist ein Weg von G, dessen Kanten alternierend in M und ausserhalb von M sind. Ein vergrössernder Weg bezüglich M ist ein alternierender Weg bezüglich M, dessen Anfangs- und Endknoten beide exponiert sind.*

Falls W ein vergrössernder Weg bezüglich M ist (genauer gesagt, die Kantenmenge eines vergrössernden Weges), so kann ein Matching M' mit $|M'| = |M| + 1$ konstruiert werden:

$$M' = M \oplus W \equiv (M - W) \cup (W - M) \tag{4.1}$$

Man sagt, M sei mit Hilfe von W *vergrössert* worden. In Abbildung 4.2 ist $\{b, f, a, d\}$ ein alternierender Weg und $W = \{e, a, d\}$ ein vergrössernder Weg bezüglich M. M kann mit Hilfe von W vergrössert werden:

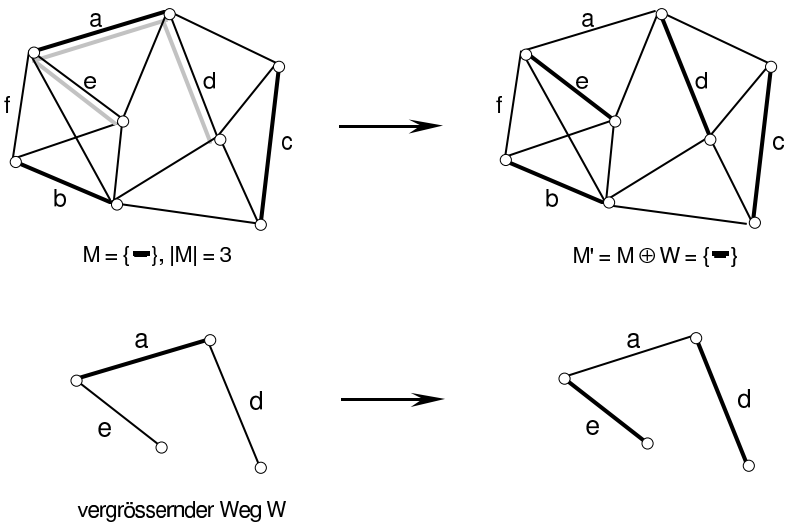


Abbildung 4.2: Vergrösserung des Matchings mittels eines vergrössernden Weges

Satz 4.3 *(i) und (ii) sind äquivalent:*

(i) Ein Matching M ist maximaler Kardinalität.

(ii) Es existiert kein vergrössernder Weg bezüglich M.

Beweis. (i) $\Rightarrow$ (ii) klar. (ii) $\Rightarrow$ (i): Nehme an, M sei nicht maximaler Kardinalität. Dann existiert ein Matching M' mit $|M'| > |M|$. Betrachte den Untergraphen $G' = (V(M \oplus M'), (M \oplus M'))$ von G [1]. In G' hat jeder Knoten einen Grad kleiner gleich 2. Deshalb sind seine Komponenten von einem der folgenden vier Typen:

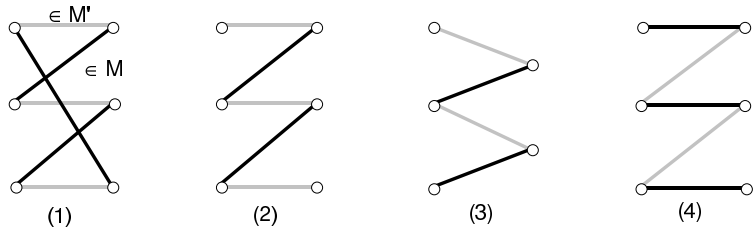


Abbildung 4.3: Vier Typen der Komponenten

(1): Kreis, (2), (3) und (4): Weg W mit $|W \cap M| = |W \cap M'| + k, k = -1, 0, +1$. Da $|M'| > |M|$, muss mindestens eine Komponente vom Typus (2) sein. (2) ist aber ein vergrössernder Weg bezüglich M. ♣

Ausgehend von einem Matching M (evtl. $= \emptyset$) wird ein Matching maximaler Kardinalität konstruiert, indem M sukzessive mit Hilfe von vergrössernden Wegen erweitert wird. Vergrössernde Wege sind sehr einfach zu finden, wenn der Graph $G = (S \cup T, E)$ bipartit ist:

Definition 4.4 (Bipartiter Graph) *Ein Graph $G = (V, E)$ heisst bipartit, falls die Knotenmenge V in zwei Mengen S und T zerlegt werden kann, so dass jede Kante $e \in E$ den einen Endknoten in S und den anderen in T hat.*

[1] In $G = (V, E)$ bezeichnet $V(A)$ die Menge der Endknoten der Kanten von A für beliebiges $A \subseteq E$.

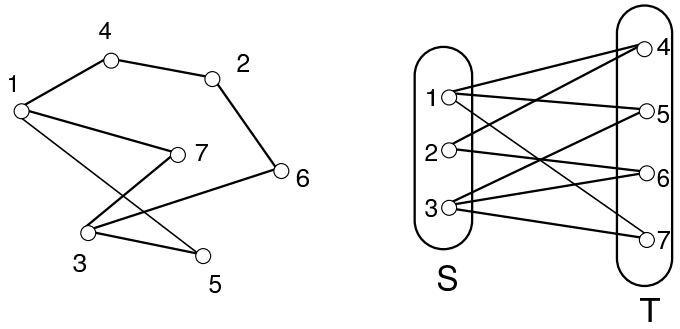


Abbildung 4.4: Beispiel eines bipartiten Graphen

Wir betrachten zuerst das Problem des Matchings maximaler Kardinalität für bipartite Graphen.

4.2 Matching maximaler Kardinalität für bipartite Graphen

Wir betrachten zuerst ein weiteres Optimalitätskriterium für ein Matching maximaler Kardinalität in einem bipartiten Graphen:

Definition 4.5 (Überdeckung) *Eine Überdeckung der Kanten von $G = (S \cup T, E)$ mit Knoten ist eine Knotenmenge $U \subseteq S \cup T$, so dass jede Kante von G mit mindestens einem Knoten von U inzidiert.*

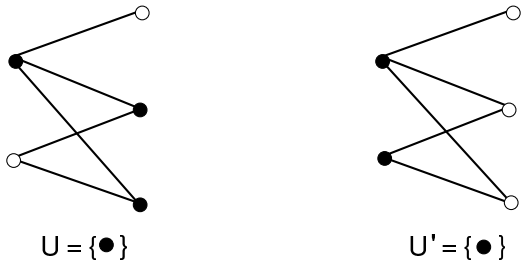


Abbildung 4.5: Zwei Überdeckungen U, U', wobei U' minimaler Kardinalität ist.

Lemma 4.6 *Für jedes Matching M und jede Überdeckung U gilt $|M| \leq |U|$.*

Beweis. Es sind $|M|$ Knoten notwendig, um allein die Kanten von M zu überdecken. ♣

Satz 4.7 (König–Egervary) *Die maximale Kardinalität eines Matchings in einem bipartiten Graphen ist gleich der minimalen Kardinalität einer Überdeckung.*

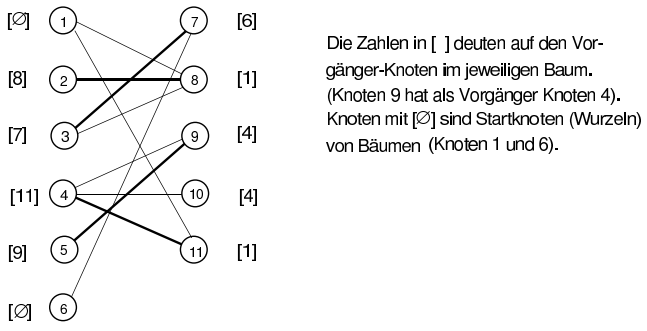
Beweis. Mit Hilfe des nachfolgenden Algorithmus 4.1. ♣

Ausgehend von einem Matching M (evtl. $= \emptyset$) vergrössert der Algorithmus wiederholt M, indem jeweils ein vergrössernder Weg gesucht wird.
Findet der Algorithmus einmal keinen vergrössernden Weg bezüglich M, so weist er eine Überdeckung U mit $|U| = |M|$ auf und bricht ab. Auf Grund des Lemmas 4.6 ist $|M|$ maximal (und $|U|$ minimal).
Die Suche nach einem vergrössernden Weg bezüglich M erfolgt mit einem Markierungsprozess, der Bäume sukzessiv aufbaut. Diese Bäume sind (knoten–)disjunkt und jeder Baum hat folgende Eigenschaften:

(i) Der Startknoten r im Aufbau des Baumes ist ein exponierter S–Knoten bezüglich M;

(ii) Der (eindeutige) Weg im Baum von r zu jedem anderen Knoten des Baumes ist ein alternierender Weg bezüglich M.

Beispiel 4.8



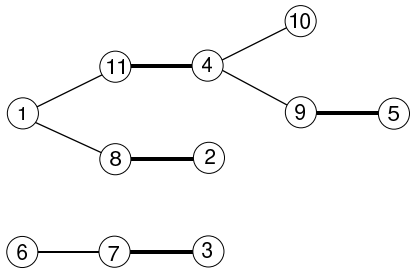


Abbildung 4.6: Suchen von vergrössernden Wegen

Der erste Baum enthält einen vergrössernden Weg bezüglich M mit Knoten 1, 11, 4, 10. ◇

Algorithmus 4.1 Matching maximaler Kardinalität (bipartit)

(* Sei $G = (S \cup T, E)$ ein bipartiter Graph. Der Algorithmus findet eine Kantenmenge M eines Matchings maximaler Kardinalität *)
begin (* Matching maximaler Kardinalität *)
Sei M ein Matching von G (z.B. $M = \emptyset$)
repeat
Vergrösserung := false (*Vergrösserung gefunden *)
inspKnoten := $\emptyset$ (* zu inspizierende Knoten *)
marke $[v] := -1 \, \forall v \in S \cup T$ (* $\sim$ nicht markiert *)
for all $v \in S, v$ exponiert **do**
marke $[v] := \emptyset$
inspKnoten := inspKnoten $\cup v$
end (* for *)
while inspKnoten $\neq \emptyset$ und Vergrösserung = false **do**
wähle $v \in$ inspKnoten, inspKnoten := inspKnoten $\setminus v$
if $v \in S$ **then**
for all $j \in \{j \in T \mid (v, j) \in E \setminus M,$ marke $[j] = -1\}$ **do**
marke $[j] := v$
inspKnoten := inspKnoten $\cup j$
end (* for *)
end (* if *)
if $v \in T$ und v nicht exponiert **then**
Sei $i \in S$ mit $(i, v) \in M$
marke $[i] := v$
inspKnoten := inspKnoten $\cup i$
end (* if *)
if $v \in T$, v exponiert **then**
Vergrösserung := true
end (* if *)
end (* while *)
if Vergrösserung **then**
identifiziere rückwärtsschreitend den vergrössernden Weg
mit Endknoten v,vorletzten Knoten marke $[v] \ldots$, Anfangsknoten i
mit marke $[i] = \emptyset$
vergrössere M
end (* if *)
until (Vergrösserung = false)
$L := \{v \in S \cup T \mid$ marke $[v] \neq -1\}$
M ist Matching maximaler Kardinalität
$U := (S \setminus L) \cup (T \cap L)$ ist Überdeckung mit $|M| = |U|$
end (* Matching maximaler Kardinalität *)

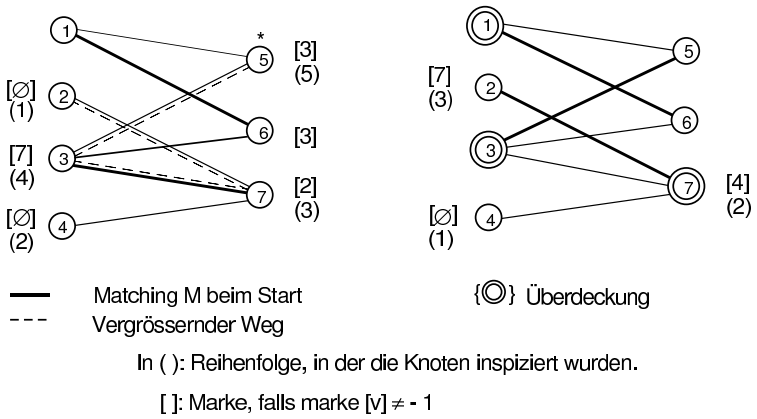


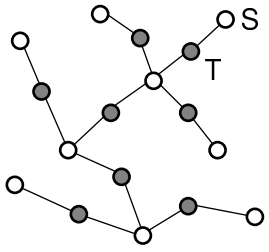
Abbildung 4.7: Beispiel zu Algorithmus 4.1

Gültigkeit des Algorithmus. Offensichtlich ist, falls Vergrösserung = true, der gefundene Weg vergrössernd bezüglich dem jeweilig vorhandenen Matching M. Es bleibt zu beweisen, dass U am Schluss eine Überdeckung ist, und dass $|U| = |M|$ (Übung). ♣

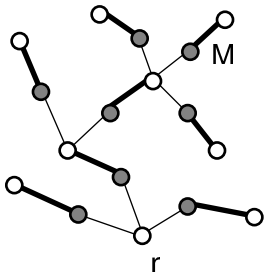
4.3 Matching maximaler Kardinalität für allgemeine Graphen

Das Auffinden eines vergrössernden Weges bezüglich eines gegebenen Matchings M ist komplizierter als im Fall eines bipartiten Graphen. Den Arbeiten von Edmonds folgend, seien folgende Definitionen eingeführt. M bezeichne stets ein Matching in G.

Ein *alternierender Baum* $J = (V_J, E_J)$ ist ein Baum von G, dessen Knoten in zwei Klassen, die S–Knoten und die T–Knoten, eingeteilt sind, so dass jede Kante von J einen S– mit einem T–Knoten verbindet und in jedem T–Knoten genau zwei Kanten inzidieren.

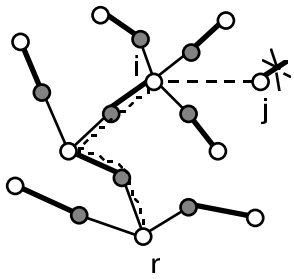


Ein *gepflanzter Baum* $J = (V_J, E_J)$ bezüglich M ist ein alternierender Baum, so dass $M^J \equiv M \cap E_J$ ein Matching maximaler Kardinalität für J ist, und der Knoten r, der bezüglich M_J exponiert ist, es auch bezüglich M ist. r heisst *Wurzel* des gepflanzten Baumes.

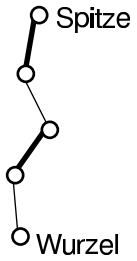


Vereinbarung: Ein exponierter Knoten bezüglich M ist für sich ein gepflanzter Baum (ohne Kanten).

Ein *vergrössernder Baum* J' bezüglich M ist ein gepflanzter Baum $J = (V_J, E_J)$ zusammen mit einer Kante $(i, j) \in E$, wobei $i \in V_J$ ein S–Knoten und $j \notin V_J$ ein exponierter Knoten bezüglich M ist. Bemerke, dass der (eindeutige) Weg in J' von j zur Wurzel r von J ein vergrössernder Weg ist.

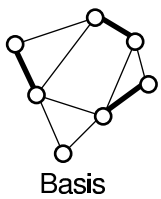


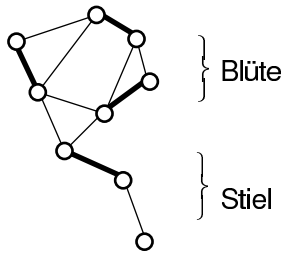
Ein *Stiel* bezüglich M ist ein exponierter Knoten bezüglich M oder ein alternierender Weg mit einem exponierten Knoten bezüglich M an einem Ende und einer Matching–Kante am anderen Ende. Der exponierte Knoten heisst *Wurzel des Stiels*, der Knoten am anderen Ende *Spitze des Stiels*.



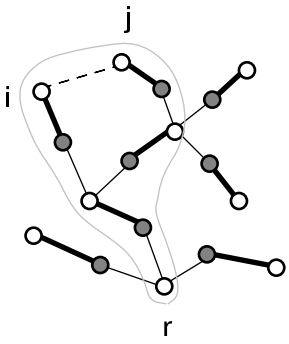
Bezeichne für eine beliebige Knotenmenge $U \subseteq V$ mit $\gamma(U)$ die Menge aller Kanten $(i, j) \in E$ mit i und $j \in U$.

Eine *Blüte* $B = (V_B, \gamma(V_B))$ bezüglich M ist ein Teilgraph von G, so dass (i): $|V_B|$ ungerade ist, (ii): $|M \cap \gamma(V_B)| = (|V_B| - 1)/2$ (d.h. M ist maximaler Kardinalität in B), und (iii): für jeden Knoten $v \in V_B$ ein Matching $M' \subseteq \gamma(V_B)$ mit $|M'| = (|V_B| - 1)/2$ existiert, das v in B exponiert lässt. Der in B bezüglich M exponierte Knoten heisst *Basis* der Blüte.





Eine *Blume* F bezüglich M besteht aus einem Stiel und einer Blüte bezüglich M, die als einzigen gemeinsamen Knoten die Spitze des Stiels gleich der Basis der Blüte haben.



Ein *beblumter Baum* J' bezüglich M ist ein gepflanzter Baum J bezüglich M zusammen mit einer Kante (i, j), die zwei S–Knoten i, j von J verbindet. Die Vereinigung der Wege von i und von j zur Wurzel r von J, zusammen mit (i, j), bilden eine Blume.

Ein *ungarischer Baum* J' bezüglich M ist ein gepflanzter Baum J bezüglich M, so dass für jede Kante $(i, j) \in E$ mit i S–Knoten von J, der Knoten j ein T–Knoten von J ist.

Satz 4.9 *Sei M ein Matching von G (evtl. $\emptyset$). Dann kann jeder gepflanzte Baum J entweder zu einem vergrössernden Baum oder einem beblumten Baum oder einem ungarischen Baum erweitert werden.*

Beweis. Jeder gepflanzte Baum $J = (V_J, E_J)$ kann mit folgendem Markierungsprozess aufgebaut werden:

Initialisierung: Markiere einen bezüglich M exponierten Knoten r mit $S : \emptyset, V_J := \{r\}, E_J := \emptyset$, und wiederhole folgenden Schritt:

Erweiterungsschritt: Sei $i \in V_J$ ein S–Knoten und $(i, j) \in E$ mit $j \notin V_J$ und von M bedeckt: Bestimme $(j, k) \in M$ und markiere j mit $T : i$ und k mit $S : j$. Setze $V_J := V_J \cup \{j, k\}, E_J := E_J \cup \{(i, j), (j, k)\}$.

Es gibt weniger als $|V|/2$ Erweiterungsschritte, da in jedem Schritt zwei neue Kanten zum Baum J kommen, und $|E_J| \leq |V| - 1$. Nehme an, man kann keinen Erweiterungsschritt durchführen. Dann tritt einer der Fälle (a), (b1) oder (b2) ein.

(a) Alle Kanten $(i, j) \in E$ mit i S–Knoten von J sind so, dass j ein T–Knoten von J ist. Der Baum J ist ungarisch.

(b) Es existiert mindestens eine Kante $(i, j) \in E$ mit i S–Knoten von J, so dass entweder

(b1) $j \notin V_J$ und exponiert bezüglich M oder

(b2) $j \in V_J$ und j S–Knoten von J.

Im Falle (b1) ist J zusammen mit (i, j) ein vergrössernder Baum, im Falle (b2) ein beblumter Baum. ♣

Satz 4.9 und sein Beweis liefern bereits einen Algorithmus zur Bestimmung eines Matchings maximaler Kardinalität, denn es gelten folgende Sätze:

Satz 4.10 *Seien $J = (V_J, E_J)$ ein ungarischer Baum bezüglich $M, M^J \equiv M \cap E_J$ und $G - J \equiv (V - V_J, \gamma(V - V_J))$ und M' ein Matching in $G - J$. M' ist maximaler Kardinalität in $G - J$, falls und nur falls $M' \cup M^J$ maximaler Kardinalität in G ist.*

Beweis.

$M' \cup M^J$ **max. Kard. in** $G \Rightarrow M'$ **max. Kard. in** $G - J$**.**
Falls M' nicht maximaler Kardinalität in $G - J$ wäre, existiere ein M'' in $G - J$ mit $|M''| > |M'|$ und $M'' \cup M^J$ wäre ein Matching von G grösserer Kardinalität als $M' \cup M^J$, ein Widerspruch.

M' **max. Kard. in** $G - J \Rightarrow M' \cup M^J$ **max. Kard. in** G**.**
Sei M_1 ein beliebiges Matching von G. Es lässt sich wie folgt in zwei disjunkte Mengen partitionieren

$$M_1 = M_1' \cup M_1'', \tag{4.2}$$

wobei $M_1' = M_1 \cap \gamma(V - V_J)$, und $M_1'' = M_1 - M_1'$ aus Kanten besteht, die mindestens einen Endknoten in V_J haben. Es gilt

$$|M_1'| \leq |M'|, \tag{4.3}$$

da M' maximaler Kardinalität in $G - J$ ist. Man bemerke, dass jede Kante von M_1'' mit mindestens einem T–Knoten von J inzident ist. Sei $|T|$ die Anzahl T–Knoten von J, dann gilt (da M_1'' ein Matching ist)

$$|M_1''| \leq |T|, \tag{4.4}$$

Aus den Gleichungen (4.2) bis (4.4) folgt, dass

$$|M_1| \leq |M_1'| + |M_1''| \leq |M'| + |T| = |M' \cup M^J|. \tag{4.5}$$

♣

Satz 4.10 besagt: Falls J ein ungarischer Baum bezüglich M ist, genügt es, J von G „abzuspalten“, ein maximales Matching M' im reduzierten Graphen $G - J$ zu suchen und es nachher mit M^J zusammenzusetzen.

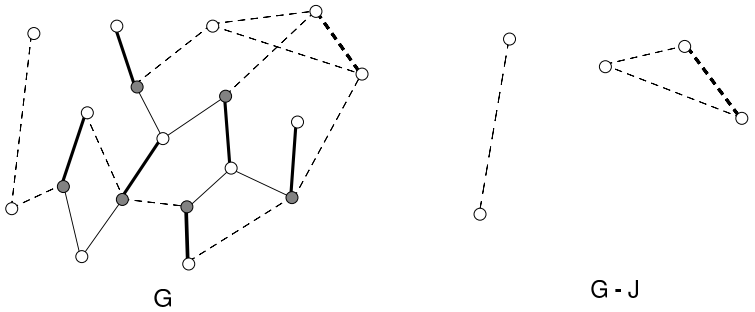


Abbildung 4.8: Ungarischer Baum J abspalten

Satz 4.11 *Seien $B = (V_B, \gamma(V_B))$ eine Blüte einer Blume bezüglich M und G/B der Graph, der aus G durch Schrumpfen von B entsteht. Es existiert ein vergrössernder Weg bezüglich $M - \gamma(V_B)$ in G/B, falls und nur falls ein vergrössernder Weg bezüglich M in G existiert. (Für die Definition von „Schrumpfen“ siehe Kapitel 2.2.2.)*

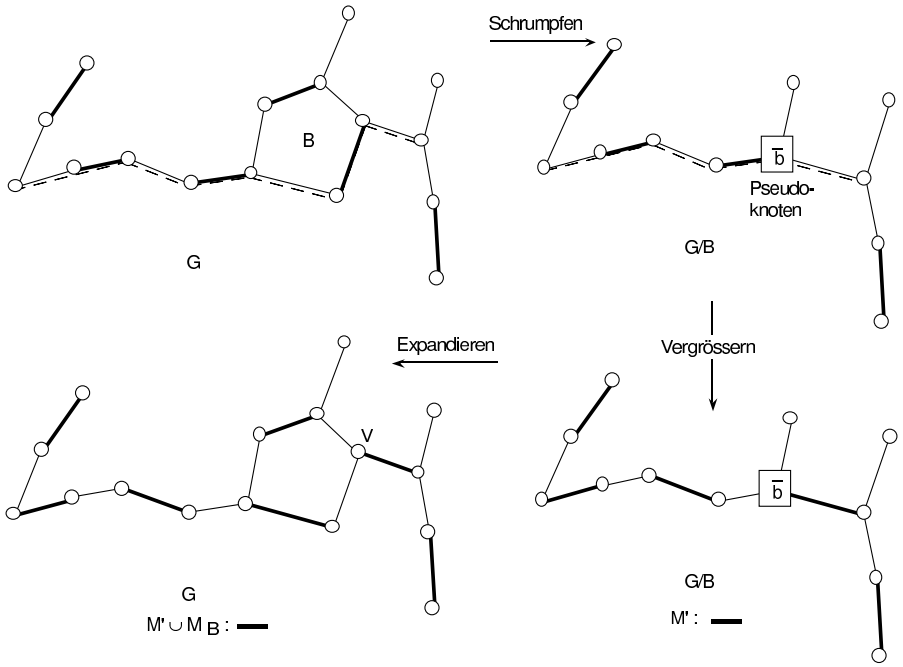


Abbildung 4.9: Matching in G/B vergrössern

Bemerkung 4.12 Beachte, wie die Kanten des Matchings in der Blüte durch die Vergrösserung verändert wurden.

Beweis.

(i) $\exists$ *ein vergrössernder Weg bezüglich* $M - \gamma(V_B)$ *in* $G/B \Rightarrow \exists$*ein vergrössernder Weg bezüglich* M *in* G*:*

Bei einer Blüte B, wie sie im Satz 4.9 zustande kommt (ungerader Zyklus), ist es leicht zu sehen, dass, falls P ein vergrössernder Weg bezüglich $M - \gamma(V_B)$ ist, P' ein vergrössernder Weg bezüglich M ist, wobei P' wie folgt definiert ist:

Fall 1: P geht nicht durch $\overline{b}$, den B darstellenden Pseudoknoten in G/B. Dann ist $P' = P$.

Fall 2: P geht durch $\overline{b}$, $\overline{b}$ ist nicht Endknoten vonn P, d.h. $\overline{b}$ ist von $M \cap P$ bedeckt. Sei $P = P_1 \cup P_2$, wobei P_1 der Teilweg von P bis

zu $\overline{b}$ mit gerader Anzahl Kanten ist, P_2 der restliche Teilweg von $\overline{b}$ an ist (siehe Abbildung 4.10). Dann ist $P' = P_1 \cup W \cup P_2$, wobei W der alternierende Weg gerader Anzahl Kanten im Kreis ist, der von der Basis b von B zum „Eingangsknoten" v von P_2 in die Blüte führt.

Fall 3: $\overline{b}$ ist Endknoten von P, d.h. bezüglich $M \cap P$ exponiert. Dann ist $P' = P \cup W$, wobei W der alternierende Weg gerader Anzahl Kanten vom „Eingangsknoten" v von P in B zur Basis b von B ist (siehe Abbildung 4.10, wobei $W = \emptyset$, falls v die Basis von B ist).

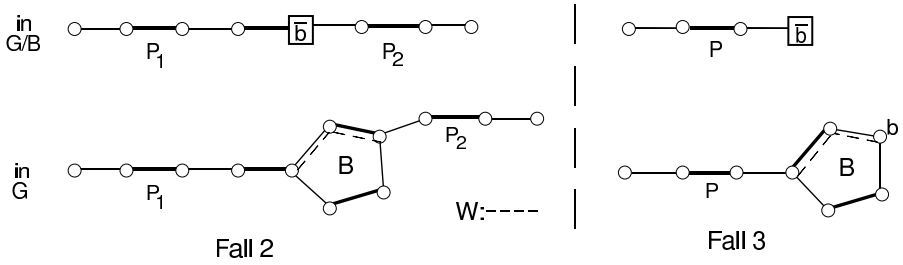


Abbildung 4.10: Vegrössernder Weg im Fall 2 und Fall 3

Ein kürzerer, indirekter Beweis von (i) unter Benutzung von Satz 4.11 ist es, zu zeigen, dass, falls $M - \gamma(V_B)$ nicht maximaler Kardinalität in G/B ist, es auch M nicht in G ist: Dies folgt direkt aus der Tatsache, dass jedes Matching M' von G/B, in G zu einem Matching $M' \cup M_B$ von G mit $|M_B| = (|V_B| - 1)/2$ erweitert werden kann: Falls $|M - \gamma(V_B)| < |M'|$, ist $|M| < |M' \cup M_B|$.

(ii) $\exists$ *ein vergrössernder Weg bezüglich* M *in* $G \Rightarrow \exists$ *ein vergrössernder Weg bezüglich* $M - \gamma(V_B)$ *in* G/B*:*

Bemerke zunächst, dass, falls B nicht Blüte einer Blume von G ist (kein Stiel), (ii) nicht richtig ist:

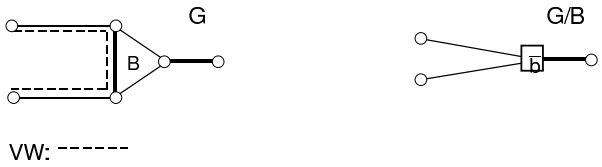


Abbildung 4.11: B ist nicht Blüte einer Blume

Es ist möglich, (ii) direkt zu beweisen, indem ein vergrössernder Weg in G/B ausgewiesen wird. Kürzer ist, wiederum Satz 4.11 zu benützen und zu zeigen, dass, falls M nicht maximal in G ist, $|M - \gamma(V_B)|$ es in G/B auch nicht ist. Sei S der Stiel der Blume in G. Bilde $M' = M \oplus S$. M' und $M' - \gamma(V_B)$ sind wieder Matchings von G und G/B mit $|M'| = |M|$ und $|M' - \gamma(V_B)| = |M - \gamma(V_B)|$ und $\overline{b}$ ist *exponiert* bezüglich $M' - \gamma(V_B)$ in G/B. Falls $|M| = |M'|$ nicht maximal ist in G, gibt es einen vergrössernden Weg P bezüglich M' in G.

Fall 1: P enthält keinen Knoten von V_B. Dann ist P auch ein vergrössernder Weg bezüglich $M' - \gamma(V_B)$ in G/B und $|M' - \gamma(V_B)|$ ist nicht maximal.

Fall 2: P enthält Knoten von V_B.Definitionsgemäss hat P seine beiden Endknoten exponiert bezüglich M'. Andererseits ist genau ein Knoten von V_B (die Spitze des Stiels) exponiert bezüglich M'. Somit ist einer der Endknoten von P ausserhalb V_B, etwa v genannt. Sei P' der Teilweg von P von v bis zum ersten Knoten von V_B, etwa v'. Wenn B geschrumpft wird, geht v' in $\overline{b}$ über, das bezüglich $M' - \gamma(V_B)$ exponiert ist, d. h. P' ist ein vergrössernder Weg bezüglich $M' - \gamma(V_B)$ in G/B.

♣

Algorithmus 4.2 Matching maximaler Kardinalität (allgemein)

begin (* Matching maximaler Kardinalität eines ungerichteten Graphen G*)
Sei M ein Matching (z.B. $M = \emptyset$) und $G' = (V', E') := G$
$j := 0$; (* Zähler für ungarische Bäume *)
while $\exists$ exponierter Knoten in V' **do**
Vergrösserung := false
marke $[w] := [\emptyset, -1]$ für alle $w \in V'$
Sei v ein exponierter Knoten von V'; setze marke $[v] := [S, \emptyset]$; inspKnoten $:= v$
$k := 0$ (* Zähler für Blüten *)
while inspKnoten $\neq \emptyset$ und Vergrösserung = false **do**
Sei $v \in$ inspKnoten, inspKnoten := inspKnoten $\setminus v$
if $\exists w \in V'$ mit w exponiert, $(v, w) \in E'$ und marke $[w] = [\emptyset, -1]$ **then**
Vergrösserung := true
else if $\exists\, w \in V'$ mit $(v, w) \in E'$, marke$[w] = [S, .]$ **then** (*Blüte gefunden*)
$k := k + 1$
Sei B_k die gefundene Blüte und p_k seine Basis
Sei $G' = (V', E')$ der durch Schrumpfen von B_k zum Pseudoknoten b_k
resultierende Graph
$M := M \cap E'$
inspKnoten := inspKnoten $\cup\, b_k \cap V'$ und marke $[b_k]$:= marke $[p_k]$
else
for all $w \in V'$ mit $(v, w) \in E'$ und marke $[w] = [\emptyset, -1]$ **do**
marke $[w] := [T, v]$
Sei $(w, x) \in M$; marke $[x] := [S, w]$; inspKnoten := inspKnoten $\cup\, x$
end (* for *)
end (* else *)
end (* else *)
end (* while *)
if Vergrösserung **then**
finde vergrössernden Weg rückwärtsschreitend von w aus
Expandiere alle Blüten $B_k, \ldots, B_1$ und führe jeweils Matching M nach
Sei $G' = (V', E')$ der expandierte Graph
else (* Ungarischer Baum abspalten *)
$j := j + 1$
Expandiere alle Blüten $B_k, \ldots, B_1$ und führe Matching M nach
Sei $G' = (V', E')$ der expandierte Graph
Sei $V_j := \{v \in V' \mid$ marke $[v] \neq [\emptyset, -1]\}$,
$G_j = (V_j, E_j)$ der von V_j induzierte Untergraph und $M_j := M \cap E_j$
Sei $G' = (V', E')$ der nach Abspalten von G_j verbleibende Graph
$M := M \cap E'$
end (* else *)
end (* while *)
$M := M \cup M_1 \cup \cdots \cup M_j$ ist ein Matching maximaler Kardinalität in G
end (* Matching maximaler Kardinalität *)

marke $[w] = [S, .]$ bedeutet, dass die erste Komponente von marke$[w]$ "S" sein muss, die zweite Komponente beliebig sein darf.

Beispiel 4.13 Das folgende Beispiel soll dieses Verfahren verdeutlichen:

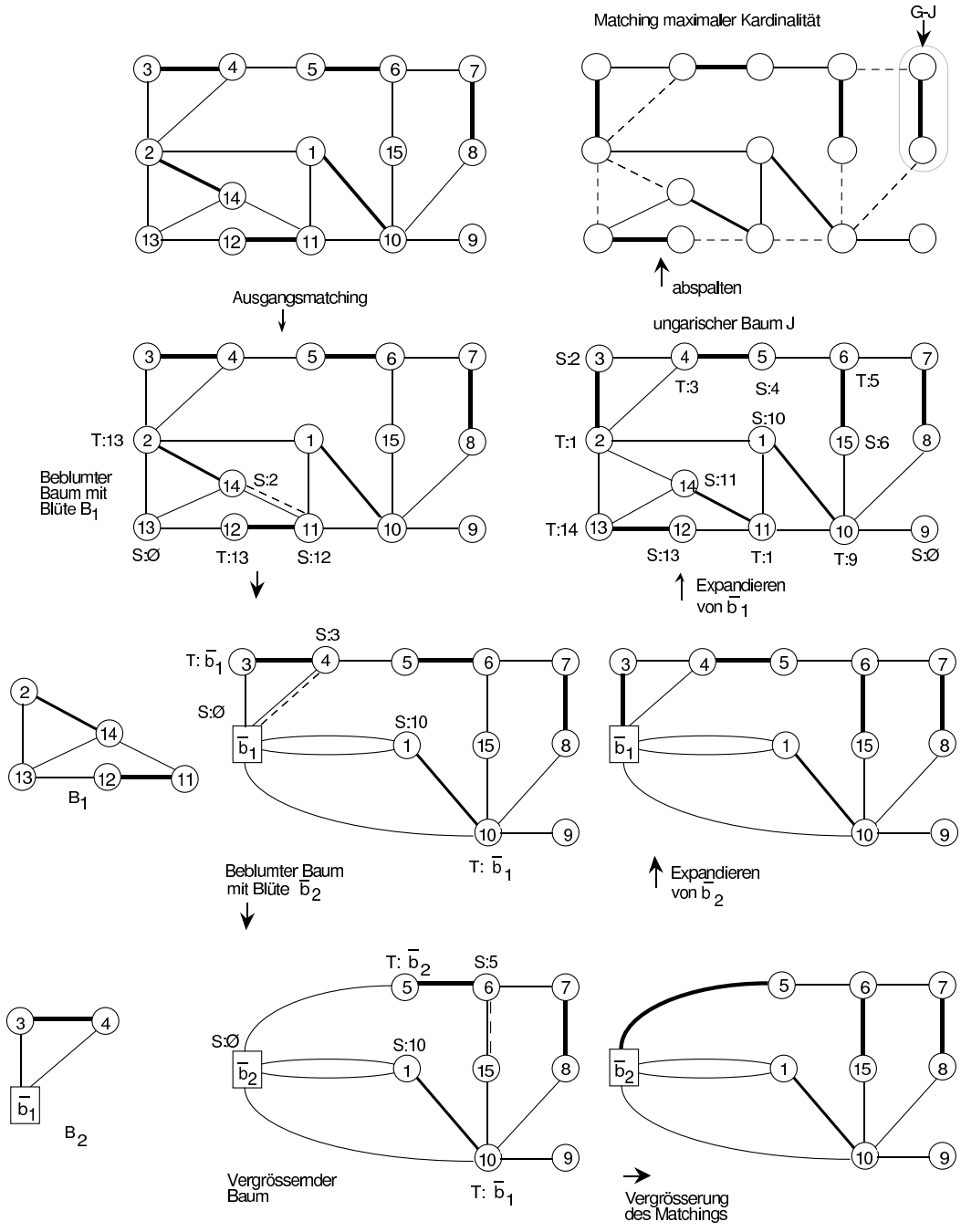


Abbildung 4.12: Algorithmus für das Matching maximaler Kardinalität

◇

Bemerkung 4.14 Um diesen Algorithmus zu implementieren, brauchen die Blüten nicht wirklich geschrumpft zu werden, es müssen dann aber zusätzliche Markierungen eingeführt werden.

Kapitel 5

Matching maximalen Gewichts

5.1 Problemformulierung

Sei $G = (V, E)$ ein ungerichteter Graph. Wir betrachten das

Problem des Matching maximalen Gewichts:

Gegeben: $G = (V, E)$ ungerichtet, Gewichte $c_e \in \mathbb{R}$, $e \in E$.

Gesucht: Ein Matching M^* von G maximalen Gewichts, d. h. ein Matching M^* mit $c(M^*) \equiv \sum_{e \in M^*} c_e = \max\{c(M) \mid M \text{ Matching von } G\}$.

Falls $G = (S \cup T, E)$ ein bipartiter Graph ist, gleich viele Knoten in S und T hat und vollständig ist ($E = \{(i, j) \mid i \in S, j \in T\}$), stellt sich das verwandte

Zuordnungsproblem:

Gegeben: Ein vollständiger, bipartiter Graph $G = (S \cup T, E)$ mit $n \equiv |S| = |T|$, und Gewichte $c_e \in \mathbb{R}, e \in E$.

Gesucht: Ein Matching M^* von G mit $|M^*| = n$ minimalen Gewichts: $c(M^*) = \min\{c(M) \mid M \text{ Matching von } G \text{ mit } |M| = n\}$.

Ein Matching heisst in diesem Fall auch Zuordnung, da es jedem Knoten von S einen Knoten von T, und umgekehrt, zuordnet.

Beispiel 5.1 In einer Werkstatt sind zur Ausführung von 5 Aufträgen 5 Pressen verfügbar, die verschieden schnell sind und sich für jeden Auftrag verschieden gut eignen.
So wurden aus den Auftragsgrössen (Anzahl Stücke) und den geschätzten Bearbeitungszeiten pro Stück für die verschiedenen Pressen folgende Bearbeitungszeiten pro Auftrag in Minuten ausgerechnet:

Auftrag / Presse	1	2	3	4	5
1	800	1100	400	600	460
2	480	600	240	360	240
3	480	700	240	360	300
4	$\infty^{*)}$	∞	160	240	220
5	∞	∞	160	240	220

*) Presse für den Auftrag nicht geeignet

Gesucht ist eine Zuordnung der Aufträge zu den Pressen, so dass die Gesamtbearbeitungsdauer minimal ist.

Das Problem ist ein Zuordnungsproblem auf nebenstehendem Graphen und Gewichtung, wie in Tabelle angegeben. In ausgezogenem Strich ist eine mögliche Zuordnung angegeben.

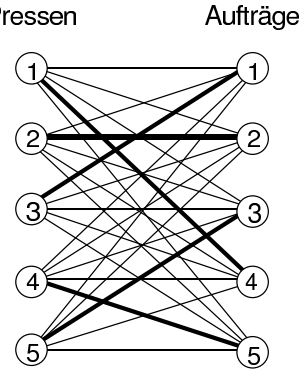


Abbildung 5.1: Zuordnung

◇

Es ist leicht, das Zuordnungsproblem in ein Problem des Matching maximalen Gewichts in einem bipartiten Graphen zu überführen und umgekehrt.
Bezeichne $\delta(v)$ für ein $v \in V$ die Menge aller Kanten von E, die mit v inzidieren. Das Problem des Matchings maximalen Gewichts ist folgendem Problem der ganzzahligen Programmierung äquivalent:

Ganzzahlige Programmierung (Matching maximalen Gewichts):

Gegeben:

$$\max \qquad c^T x = \sum_{e \in E} c_e x_e \tag{5.1}$$

$$\text{so dass} \sum_{e \in \delta(v)} x_e \leq 1 \text{ für alle } v \in V \tag{5.2}$$

$$x_e \geq 0 \text{ für alle } e \in E \tag{5.3}$$

Gesucht:

$$x_e \text{ ganzzahlig für alle } e \in E \tag{5.4}$$

Dies sieht man wie folgt: Falls $x \in \mathbb{R}^E$ zulässig ist, ist x ein Vektor mit Komponenten 0 oder 1. (5.2) besagt, dass höchstens eine Kante $e \in \delta(v)$ existiert mit $x_e = 1$. x ist deshalb Inzidenzvektor eines Matchings.
(5.1) – (5.4) ist ein ganzzahliges lineares Programm und deshalb a priori schwer zu lösen. Falls $G = (S \cup T, E)$ aber bipartit ist, darf man die *Ganzzahligkeitsforderung (5.4) fallen lassen*, da das *lineare Programm* (5.1) - (5.3) die Eigenschaft hat, dass jede *Basislösung ganzzahlig* ist. Man spricht von *natürlicher Ganzzahligkeit*.

Beweis dieser Eigenschaft. Sei A die $(m+n) \times |E|$–Matrix ($m := |S|, n := |T|$) mit Elementen

$$a_{ve} = \begin{cases} 1 & \text{falls } v \text{ ein Endnkoten von } e \text{ ist} \\ 0 & \text{sonst} \end{cases}, v \in S \cup T, e \in E. \tag{5.5}$$

A ist die (Knoten–Kanten–) *Inzidenzmatrix* von $G = (S \cup T, E)$. Das lineare Programm (LP) (5.1) – (5.3) schreibt sich auch für $x \in \mathbb{R}^E$:

$$\max \quad c^T x \tag{5.6}$$

$$\text{sodass} \quad Ax \leq \underline{1} \tag{5.7}$$

$$x \geq 0 \tag{5.8}$$

wobei $\underline{1}$ den Vektor mit allen Komponenten gleich 1 bezeichnet.

Satz 5.2 *Die Determinante jeder quadratischen Teilmatrix B von A ist* 0, $+1$ *oder* -1, *d. h. die Matrix A ist total unimodular.*

Beweis. Die Matrix A (und somit jede quadratische Teilmatrix B von A) setzt sich aus zwei Teilmatrizen A_S und A_T (B_S und B_T) zusammen, welche beide die Eigenschaft haben, dass jede ihrer Spalten höchstens eine Eins, sonst lauter Nullen enthaltet. Betrachte also eine $k \times k$–Teilmatrix B und treffe die Induktionshypothese, dass $\det(B') = 0, \pm 1$ für jede $\ell \times \ell$–Teilmatrix B' von A mit $1 \leq \ell < k$. Falls jede Spalte von B zwei Einsen enthält, ist $\det(B) = 0$ (Zeilen von B_S – Zeilen von $B_T = \underline{0}$). Falls eine Spalte nur Nullen enthält, ist $\det B = 0$, falls sie eine 1 enthält, entwickelt man $\det(B)$ nach dieser Spalte: $\det(B) = \pm 1 \cdot \det(B')$, wobei B' eine $(k-1) \times (k-1)$–Matrix ist. Da $\det(B') \in \{0, \pm 1\}$ ist $\det(B) \in \{0, \pm 1\}$. ♣

Korollar 5.3 *Jede Basislösung $\overline{x}$ von (5.7) ist ganzzahlig.*

Beweis. Nehme an, $\overline{x}_{ij} = 0$ für $(i,j) \in E_0 \subseteq E$. x' mit Komponenten $x'_{ij} = \overline{x}_{ij}, (i,j) \in E - E_0$ ist die einzige Lösung eines Systems $Bz = \underline{1}$, wobei B eine reguläre $|E - E_0| \times |E - E_0|$–Teilmatrix von A mit Spalten in $E - E_0$ ist. Dann ist $x' = B^{-1} \cdot \underline{1} = (B^+ \cdot \underline{1}) / \det(B)$ ganzzahlig, da $\det(B) = \pm 1$. (B^+ ist die adjungierte Matrix von B). ♣

Folglich wird das Problem des Matching maximalen Gewichts mit der Lösung des LP (5.1)–(5.4) gelöst (solange das benutzte LP–Verfahren Eckpunkte liefert). ♣

Die grosse Anzahl Variablen $|S| \cdot |T| = m \times n$ macht die Anwendung des Simplex–Algorithmus jedoch umständlich. Wir werden im nächsten Abschnitt einen effizienteren Algorithmus, das sogenannte ungarische Verfahren, betrachten.
Falls G kein bipartiter Graph ist, darf man die Ganzzahligkeitsforderungen (5.4) nicht einfach fallen lassen, da dabei nicht–ganzzahlige Basislösungen auftreten können, wie folgendes Beispiel zeigt:

Beispiel 5.4

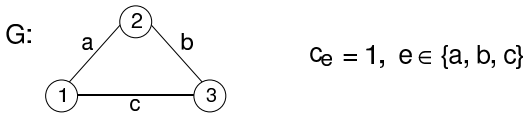


Abbildung 5.2: Graph G

Das LP:

$$\begin{array}{lllll} \text{maximiere} & x_a + & x_b + & x_c & \\ \text{so dass} & x_a + & & x_c & \leq 1 \quad (v=1) \\ & x_a + & x_b & & \leq 1 \quad (v=2) \\ & & x_b + & x_c & \leq 1 \quad (v=3) \\ & x_a, & x_b, & x_c & \geq 0 \end{array} \tag{5.9}$$

hat als optimale Basislösung $x_a = x_b = x_c = \frac{1}{2}$. ◇

Betrachte die Ungleichungen

$$\sum\{x_e \mid e \in \gamma(S)\} \leq \frac{|S|-1}{2} \tag{5.10}$$

für alle Knotenmengen $S \subseteq V$ mit $|S| \geq 3, |S|$ ungerade. (5.10) sind die sog. *Blütenungleichungen* und werden offenbar vom Inzidenzvektor jedes Matchings erfüllt.

Satz 5.5 *Die Basislösungen des linearen Programms (5.1)–(5.3) und (5.10) sind die Inzidenzvektoren der Matchings von G.*

Beweis. siehe Algorithmus. ♣

Satz 5.5 besagt, dass das LP (5.1)–(5.3) und (5.10) dem Problem des Matchings maximalen Gewichts „äquivalent“ ist. Beachte jedoch, dass dieses LP sehr viele Restriktionen besitzt (i.a. exponentiell viele Blütenungleichungen in der Anzahl Knoten), so dass lediglich schon die Aufnahme der Restriktionen in Hinsicht auf die Anwendung eines Standard–Verfahrens der Linearen Programmierung (z. B. Simplex) Schwierigkeiten bieten würde! Dennoch wird (5.1)–(5.3) und (5.10) mit einem speziellen primal–dual–Verfahren effizient ($0(|V|^3)$ in einer geschickten Implementation) gelöst (siehe Abschnitt 5.3).

Wir wollen zuerst das Problem des Matchings maximalen Gewichts im bipartiten Graphen lösen.

5.2 Matching maximalen Gewichts in bipartiten Graphen

Wir werden dieses Problem mit dem sogenannten Ungarischen Verfahren, einem primal–dual Verfahren, lösen.

Das LP (5.1) - (5.3) und (5.10) kann für $S := \{1, \ldots, m\} und T := \{1, \ldots, n\}$ sowie $c_{ij} := c_{(i,j)}$ für $e = (i, j) \in E$ wie folgt geschrieben werden:

$$\begin{array}{lll} \text{maximiere} & \displaystyle\sum_{(i,j)\in E} c_{ij}x_{ij} & \\ \text{so dass} & \displaystyle\sum_{j:(i,j)\in E} x_{ij} \quad \leq 1, & i = 1, \ldots, m, \\ & \displaystyle\sum_{i:(i,j)\in E} x_{ij} \quad \leq 1, & j = 1, \ldots, n, \\ & x_{ij} \quad \geq 0, & (i, j) \in E \end{array} \tag{5.11}$$

Das duale LP zum LP (5.11) lautet

$$\begin{array}{lll} \text{minimiere} & \displaystyle\sum_{i=1}^{m} u_i + \sum_{j=1}^{n} v_j, & \\ \text{so dass} & u_i + v_j \geq c_{ij} & (i, j) \in E \\ & u_i \geq 0 & i = 1, \ldots, m \\ & v_j \geq 0 & j = 1, \ldots, n \end{array} \tag{5.12}$$

Die Komplementärschlupfbedingungen (KSB) lauten:

$$x_{ij} > 0 \Rightarrow u_i + v_j = c_{ij} \tag{5.13}$$

$$u_i > 0 \Rightarrow \sum_{j:(i,j)\in E} x_{ij} = 1 \tag{5.14}$$

$$v_j > 0 \Rightarrow \sum_{i:(i,j)\in E} x_{ij} = 1 \tag{5.15}$$

Falls x der Inzidenzvektor eines Matchings M ist, besagen (5.13) bis (5.15):

$$(i, j) \in M \Rightarrow u_i + v_j = c_{ij} \tag{5.13'}$$

$$u_i > 0 \Rightarrow \text{ Knoten } i \text{ ist von } M \text{ bedeckt} \tag{5.14'}$$

$$v_j > 0 \Rightarrow \text{ Knoten } j \text{ ist von } M \text{ bedeckt} \tag{5.15'}$$

Gemäss dem Gesetz des Komplementärschlupfs der linearen Programmierung sind ein Matching M maximalen Gewichts und eine zulässige Duallösung

$u \in \mathbb{R}^m, v \in \mathbb{R}^n$ optimal, falls (und nur falls) die Komplementärschlupfbedingungen (5.13')–(5.15') erfüllt sind.
Der Ungarische Algorithmus 5.1 liefert ein Matching M und eine zulässige Lösung u, v, die (5.13')–(5.15') erfüllen. Sei $G = (S \cup T, E)$ ein bipartiter Graph mit Gewichten $c_e \in \mathbb{R}$, $e \in E$. Der Algorithmus findet ein Matching M maximalen Gewichts und eine optimale duale Lösung $(u_i, i \in S; v_j, j \in T)$ von 5.12.

Algorithmus 5.1 Matching maximalen Gewichts (bipartit)

begin (* Matching maximalen Gewichts *)
$M := \emptyset$ und $maxc := \max\{0, \max\{c_e \mid e \in E\}\}$
$u_i := maxc \, \forall i \in S$; $v_j := 0 \, \forall j \in T$
stopkrit := false
repeat
if $M = \emptyset$ oder Vergrösserung **then**
Vergrösserung := false; inspKnoten := $\emptyset$ (* zu inspizierende Knoten *)
marke $[v] := -1$ für alle $v \in S \cup T$ (* $\sim$ nicht markiert *)
$\pi_j := \infty$ für alle $j \in T$
end (* if *)
for all $i \in S$ mit i exponiert **do** (* Markierung *)
marke $[i] := \emptyset$, inspKnoten := inspKnoten $\cup \, i$
end (* for *)
while inspKnoten $\neq \emptyset$ und Vergrösserung = false **do**
wähle $v \in$ inspKnoten; inspKnoten := inspKnoten $\setminus v$
(* Inspektion von v *)
case $v \in S$:
for all $(v, j) \in E \setminus M$ **do**
if $u_v + v_j - c_{vj} < \pi_j$ **then**
marke $[j] := v; \pi_j := u_v + v_j - c_{vj}$
if $\pi_j = 0$ **then** inspKnoten := inspKnoten $\cup \, j$
end (* if *)
end (* for *)
case $v \in T, v$ nicht exponiert:
sei $(i, v) \in M$
marke $[i] := v$, inspKnoten := inspKnoten $\cup \, i$
case $v \in T, v$ exponiert:Vergrösserung := true
end (* while *)
if Vergrösserung **then**
ein vergrössernder Weg mit Endknoten v ist gefunden identifiziere ihn
rückwärtsschreitend entlang Marken und vergrössere das Matching
else (* Dualvariablenveränderung *)
$\delta_1 := \min\{u_i \mid i \in S\}, \delta_2 := \min\{\pi_j \mid \pi_j > 0, j \in T\}$
$\delta := \min\{\delta_1, \delta_2\}$
$u_i := u_i - \delta$ für alle $i \in S$ mit marke $[i] \neq -1$
$v_j := v_j + \delta$ für alle $j \in T$ mit $\pi_j = 0$
for all $j \in T$ mit marke $[j] \neq -1$ und $\pi_j > 0$ **do**
$\pi_j := \pi_j - \delta$
if $\pi_j = 0$ **then** inspKnoten := inspKnoten $\cup \, j$
end (* for *)
if $\delta = \delta_1$ **then** stopkrit := true
end (* else *)
until stopkrit = true
M ist ein Matching maximalen Gewichts
$(u_i, i \in S$ und $v_j, j \in T)$ ist eine optimale duale Lösung von 5.12
end (* Matching maximalen Gewichts *)

Die Haupteigenschaften des Ungarischen Algorithms sind:

(i) In *jedem* Schritt liegen ein Matching M (primär zulässig) und eine zulässige duale Lösung u, v vor, die die Komplementärschlupfbedingungen (KSB) (5.13') und (5.15') erfüllen.

(ii) Im Verlaufe des Algorithmus nimmt die Anzahl verletzter KSB (5.14') stetig ab, bis sie bei Abbruch gleich Null ist.

(iii) Es werden *Vergrösserungsschritte* und *Dualvariablenänderungen* vorgenommen. In einem Vergrösserungsschritt wird das Matching mit einem zu demjenigen von Kapitel 4.2 analogen Verfahren vergrössert. Um jedoch die Einhaltung von (5.13') weiter zu gewährleisten, wird nur im Untergraphen $(V(E_0), E_0)$ mit $E_0 = \{(i,j) \in E \mid u_i + v_j = c_{ij}\}$ vergrössert. Falls das jeweilige Matching in diesem Graphen nicht vergrössert werden kann, werden die Dualvariablen verändert.

Beweis der Gültigkeit des Algorithmus.

(a) *Zu Beginn der repeat–Schleife liegt jeweils eine primal zulässige Lösung und eine dual zulässige Lösung vor.*

Die primale Zulässigkeit folgt aus der Tatsache, dass M immer ein Matching ist. Die duale Zulässigkeit überprüft man wie folgt: Offensichtlich erfüllt die Initiallösung u, v die Bedingungen im dualen LP 5.12. Seien u, v und u', v' die Variablen vor und nach einer Dualvariablenänderung. u', v' erfüllen offensichtlich die Nichtnegativitätsbedingungen in 5.12 (Wahl von δ). Bei der Überprüfung einer bestimmten Ungleichung $u'_i + v'_j \geq c_{ij}$ ist offenbar nur der Fall $u'_i = u_i - \delta$ näher zu betrachten. In diesem Fall ist marke $[i] \neq -1$. Falls $(i,j) \in M$, so ist marke $[i] = j \neq -1$ und $\pi_j = 0$. Deshalb ist $u'_i = u_i - \delta, v'_j = v_j + \delta$ und $u'_i + v'_j = u_i + v_j = c_{ij}$. Falls $(i,j) \notin M$ und $\pi_j > 0$, gilt (da i einmal inspiziert wurde und wegen der Wahl von δ)

$$\delta \leq \pi_j \leq u_i + v_j - c_{ij} \text{ und deshalb } c_{ij} \leq u_i - \delta + v_j = u'_i + v'_j. \quad (5.16)$$

(b) *Zu Beginn der repeat–Schleife sind jeweils die KSB (5.13') und (5.15') erfüllt.*

Dies gilt trivialerweise für die Initiallösung $M = \emptyset$ und $v = 0$. Betrachte einen Vergrösserungsschritt, und seien M und M' das Matching vor und

nach Vergrösserung. Da der vergrössernde Weg nur Kanten (i, j) mit $u_i + v_j = c_{ij}$ benutzt, bleibt (5.13') erhalten. Da alle von M bedeckten Knoten auch von M' bedeckt sind, bleibt (5.15') erhalten. Betrachte eine Dualvariablenänderung. Falls $(i, j) \in M$ und marke $[i] = j$, dann ist $\pi_j = 0$. Folglich gilt: $u'_i + v'_j = (u_i - \delta) + (v_j + \delta) = u_i + v_j - c_{ij}$. Falls $(i, j) \in M$ und marke $[i] \neq j$, so ist marke $[i] = -1$ und $\pi_j \neq 0$. Also ist $u'_i + v'_j = u_i + v_j = c_{ij}$ und (5.13') bleibt erhalten. Für (5.15') ist der Fall $\pi_j = 0$ speziell zu überprüfen: j wurde inspiziert und ist somit von M überdeckt (sonst wäre ein vergrössernder Weg gefunden worden).

(c) *Falls eine KSB (5.14') einmal erfüllt ist, bleibt sie es im weiteren Verlauf des Algorithmus.*

Dies folgt aus der Tatsache, dass bei einer Vergrösserung die von M bedeckten Knoten auch von M' bedeckt sind, und dass bei einer Dualvariablenänderung $u'_i \leq u_i, i = 1, \ldots, m$, gilt.

(d) *Optimalität und Aufwand.*

Zu jeder Zeit im Verlauf des Algorithmus sind die Werte u_i der exponierten Knoten $i \in S$ bezüglich des jeweiligen Matching alle gleich $\min\{u_i \mid i \in S\}$. Bei Abbruch sind deshalb die u_i der exponierten Knoten $i \in S$ alle Null und alle KSB (5.14') erfüllt.

Zwischen zwei aufeinander folgenden Vergrösserungen ist der Aufwand für Markieren und Inspizieren der Knoten, inklusive eventueller Dualvariablenänderungen, $O(mn)$. Zudem gibt es höchstens m Vergrösserungsschritte. Der Gesamtaufwand ist demnach $O(m^2 n)$.

♣

Bemerkung 5.6 Unter Ungarischem Algorithmus wird auch oft ein Verfahren für das Zuordnungsproblem verstanden, das auf denselben Prinzipien beruht, wie das erläuterte Verfahren.

5.3 Matching maximalen Gewichts in allgemeinen Graphen

Um dieses Problem zu lösen, werden wir ein Verfahren betrachten, das von Edmonds (1965) entwickelt wurde und das auf der oben eingeführten LP–

Formulierung (5.1)–(5.3) und (5.10) basiert. Es ist ein primal–duales Verfahren, das zum Algorithmus für den bipartiten Fall ähnlich ist, die im allgemeinen Fall zu berücksichtigenden Blütenungleichungen (5.10) komplizieren den Algorithmus aber wesentlich.
Das duale LP zu (5.1)–(5.3) und (5.10) ist:

$$\min \qquad \sum_{v\in V} u_v + \sum\{r_S z_S \mid S \subseteq V, |S| \geq 3, \text{ ungerade}\} \tag{5.17}$$

$$\text{so dass} \quad u_v + u_w + \sum\{z_S \mid S : e \in \gamma(S)\} \geq c_e \qquad e = (v,w) \in E, \tag{5.18}$$

$$u_v \geq 0, \qquad v \in V, \tag{5.19}$$

$$z_S \geq 0, \qquad S \subseteq V, |S| \geq 3, \text{ ungerade} \tag{5.20}$$

wobei $r_s := (|S| - 1)/2$ die rechte Seite von (5.10) ist. (Jedem Knoten v ist eine (duale) Variable u_v, jeder Knotenmenge S ungerader Knotenzahl $|S| \geq 3$ eine Variable z_S zugeordnet.)
Damit eine primal zulässige Lösung x (die (5.2), (5.3) und (5.10) erfüllt) und eine dual zulässige Lösung (u, z) (, die (5.18) – (5.20) erfüllt) beide optimal sind, genügt es, dass die Komplementärschlupfbedingungen *(KSB)* erfüllt sind:

$$x_e > 0 \Rightarrow d_e \equiv \underset{\text{(wobei } e=(v,w))}{u_v + u_w} + \sum\{z_S \mid S : e \in \gamma(S)\} - c_e = 0 \tag{5.21}$$

$$u_v > 0 \Rightarrow \sum_{e\in\delta(v)} x_e = 1 \tag{5.22}$$

$$z_S > 0 \Rightarrow \sum_{e\in\gamma(S)} x_e = (|S| - 1)/2 \tag{5.23}$$

Falls x der Inzidenzvektor eines Matchings M ist, besagen (5.21) und (5.22):

$$e \in M \Rightarrow de = 0 \tag{5.21'}$$

$$u_v > 0 \Rightarrow v \text{ von } M \text{ bedeckt} \tag{5.22'}$$

Der primal–dual–Algorithmus beginnt mit dem primal zulässigen Matching $M = \emptyset$ (Inzidenzvektor $x = 0$), und der dual zulässigen Lösung $u_v =$

$\max\{0, \max_{e \in E}\{c_e\}\}$ für alle $v \in V$, $z_S = 0$ für alle $S \subseteq V$, $|S| \geq 3$, ungerade. Dieses Lösungspaar erfüllt (5.21') und (5.23), jedoch i.a. nicht (5.22').
Im Verlaufe des Algorithmus wird das Matching, also x, in *Vergrösserungsschritten*, und die dualen Variablen in *Dualvariablenänderungen* so geändert, dass in jedem Schritt

(i) die primale Zulässigkeit (indem x Inzidenzvektor eines Matchings M ist),

(ii) die duale Zulässigkeit,

(iii) die KS (5.21'),

(iv) die KS (5.23) (indem $(S, \gamma(S))$ mit $z_S > 0$ im Graph $(V, E' := \{e \in E \mid de = 0\})$ eine Blüte bezüglich M ist),

erhalten bleiben, und

(v) die Bedingungen (5.22'), die verletzt sind,

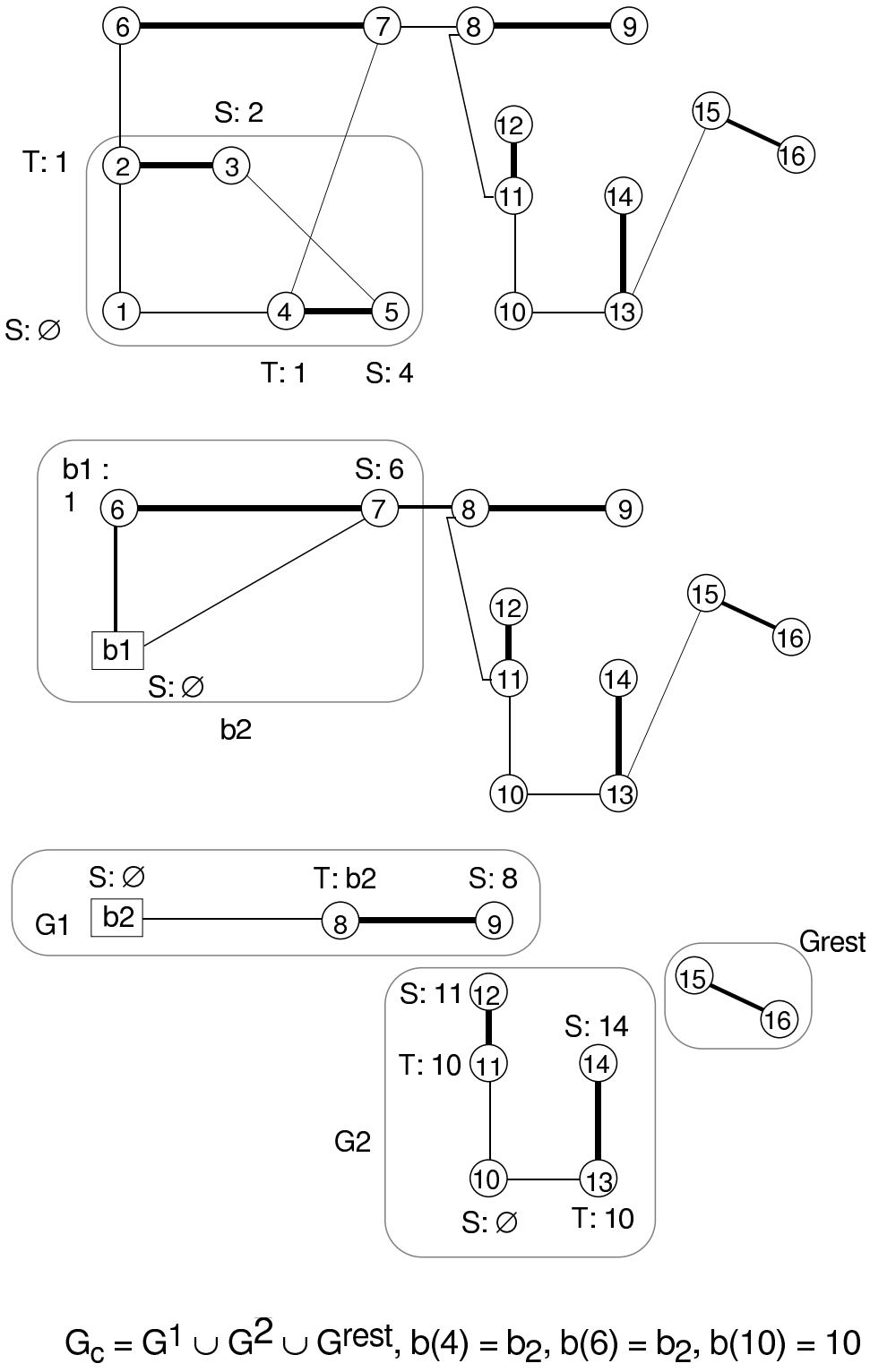
nach und nach in Ordnung gebracht werden.
In einem Vergrösserungsschritt wird versucht, das jeweilige Matching M zu vergrössern: es wird ein Matching maximaler Kardinalität gesucht, jedoch nicht in G, sondern in einem abgeleiteten Graphen G', in dem

(i) alle Kanten e mit $d_e > 0$ gelöscht sind, und

(ii) alle Blüten $B = (S, \gamma(S))$ mit $z_S > 0$ geschrumpft sind.

(i) bzw. (ii) stellen sicher, dass (5.21') bzw. (5.23) nach Vergrösserung weiter gelten. Zudem, falls für ein $v \in V$ (5.22') vor Vergrösserung erfüllt war, bleibt es nach Vergrösserung erfüllt. Falls keine Vergrösserung möglich ist, werden die Dualvariablen verändert.
Um den Algorithmus zu beschreiben, werden folgende Notationen verwendet (vgl. Abbildung 5.3):
Sei $G = (V, E)$ ein Graph und M ein Matching maximaler Kardinalität. Seien $G^1, \ldots, G^j$ die ungarischen Bäume, die der Algorithmus für ein Matching maximaler Kardinalität aufbaut, wobei jeweils die Blüten **nicht** expandiert werden, und G^{rest} der nach Abspalten der ungarischen Bäume verbleibende Graph. Mit $G_c = (V_c, E_c)$ bezeichnen wir den Graphen $G_c := G^1 \cup \cdots \cup G^j \cup G^{rest}$. Jeder Knoten $v \in V_c$ ist entweder ein Pseudoknoten,

dessen Expansion ein Untergraph $(S_v, \gamma(S_v))$ von G ergibt, oder ein Knoten $v \in V$, also $S_v := \{v\}$. Für jeden Knoten $i \in V$ bezeichnen wir mit $b(i)$ den äussersten Pseudoknoten, zu dem i gehört, also denjenigen Knoten $b(i)$ von V_c für den $i \in S_{b(i)}$.



$$G_c = G^1 \cup G^2 \cup G^{rest},\ b(4) = b_2,\ b(6) = b_2,\ b(10) = 10$$

Abbildung 5.3: verwendete Notationen beim Matching-Algorithmus

Sei $G = (V, E)$ ein ungerichteter Graph mit Gewichten $c_e \in \mathbb{R}$, $e \in E$. Der Algorithmus 5.2 findet ein Matching maximalen Gewichts:

Algorithmus 5.2 Matching maximalen Gewichts (allgemein)

begin (* Matching maximalen Gewichts *)
$M := \emptyset$
$\max c := \max\{0, \max\{c_e \mid e \in E\}\}$
$u_i := 1/2 \max c$
$z_s := 0$ für alle $S \subseteq V$, $|S|$ ungerade, $|S| \geq 3$
repeat
Sei $E' := \{(i,j) \in E \mid u_i + u_j + \sum\{z_s \mid (i,j) \in \gamma(S)\} = c_{ij}\}$
und $\mathbf{S} := \{S \mid z_s > 0\}$
Sei G' der von G abgeleitete Graph, indem alle Kanten $e \notin E'$ gelöscht
wurden und alle Blüten $(S, \gamma(S))$, $S \in S$ geschrumpft wurden.
(* Matching in G' vergrössern *)
Finde in G' ein Matching M' maximaler Kardinalität und sei M das
entsprechende Matching in G (mit Kanten aus E')
Sei $G'_c = (V'_c, E'_c) = G^1 \cup \cdots \cup G^j \cup G^{rest}$, wobei $G^1, \ldots, G^j$ ungarische
Bäume sind.
Sei $\mathbf{S}_T := \{S_v \mid v \in V'_c, v$ T–Knoten und v Pseudoknoten, $|S_v| \geq 3\}$
und $\mathbf{S}_S := \{S_v \mid v \in V'_c, v$ S–Knoten und v Pseudoknoten, $|S_v| \geq 3\}$
(* Dualvariablenveränderung *)
$\delta_1 := \min\{u_i \mid i \in V\}$
$\delta_2 := \frac{1}{2} \cdot \min\{z_s \mid S \in \mathbf{S}_T\}$
$\delta_3 := \frac{1}{2} \cdot \min\{u_i + u_j + \sum\{z_s \mid (i,j) \in \gamma(S)\} - c_{ij} \mid (i,j) \in E \setminus E',$
$b(i) \neq b(j),\ b(i)$ und $b(j) S$ –Knoten$\}$
$\delta_4 := \min\{u_i + u_j + \sum\{z_s \mid (i,j) \in \gamma(S)\} - c_{ij} \mid (i,j) \in E \setminus E',$
$b(i)\, S$ –Knoten und $b(j)$ Knoten von $G^{rest}\}$
$\delta := \min\{\delta_1, \delta_2, \delta_3, \delta_4\}$
$u_i := u_i - \delta$ für alle $i \in V$ mit $b(i)$ S–Knoten
$u_i := u_i + \delta$ für alle $i \in V$ mit $b(i)$ T–Knoten
$z_S := z_S + 2\delta$ für alle $S \in \mathbf{S}_S$
$z_S := z_S - 2\delta$ für alle $S \in \mathbf{S}_T$
until $\delta = \delta_1$
M ist ein Matching maximalen Gewichts
(u, z) ist eine optimale duale Lösung von (5.17) – (5.20)
end (* Matching maximalen Gewichts *)

Bemerkung 5.7

(i) Nach einer Dualvariablenveränderung kann das Markierungsverfahren ausgehend von G'_c weitergeführt werden, wobei die Kanten $(i,j) \in E \setminus E'$, für die neu

$$ui_i + u_j + \sum\{z_S \mid (i,j) \in \gamma(S)\} - c_{ij} = 0 \tag{5.24}$$

gilt, zu E' hinzugefügt werden und die Blüten $S \in \mathbf{S}_T$, für die neu $z_s = 0$ wird, expandiert werden. Die einzige Änderung gegenüber dem Algorithmus vom Matching maximaler Kardinalität des Kapitels 4.3 ist, dass nun mehrere gepflanzte Bäume gleichzeitig aufgebaut werden (die ungarischen Bäume $G^1, \ldots, G_j$, die nach der Dualvariablenveränderung nicht mehr ungarische Bäume sind). Beim Markierungsverfahren kann deshalb neu die Situation auftreten, dass eine Kante $(i,j) \in E$ existiert mit $b(i)$ und $b(j)$ S–Knoten, wobei $b(i)$ und $b(j)$ zu verschiedenen gepflanzten Bäumen gehört (die Kante $(b(i), b(j))$ gehört nicht zu einer Blüte). In diesem Fall ist aber ein vergrössernder Weg gefunden worden, nämlich der Weg von der Wurzel des gepflanzten Baumes, zu dem $b(i)$ gehört, nach $b(i)$ zusammen mit (i,j) und dem Weg von $b(j)$ zur Wurzel des anderen gepflanzten Baumes.

(ii) Dieser Algorithmus kann in $O(V^3)$ implementiert werden.

(iii) Der Algorithmus liefert für jedes c einen Inzidenzvektor eines Matchings, der eine optimale Lösung des LP's (5.1)–(5.3) und (5.10) ist. Somit ist jede Basislösung vom LP (5.1)–(5.3) und (5.3) ein Inzidenzvektor eines Matchings, welches den Satz 5.5 beweist wie wir nun sehen werden.

Beweis der Gültigkeit des Algorithmus.

(a) *Es liegt jeweils eine primal zulässige Lösung und eine dual zulässige Lösung vor.*

Die primale Zulässigkeit folgt aus der Tatsache, dass M immer ein Matching ist. Die duale Zulässigkeit überprüft man wie folgt: Offensichtlich erfüllt die Initiallösung (u,z) die Bedingungen (5.18) – (5.20). Seien

(u, z) und (u', z') die Variablen vor und nach einer Dualvariablenveränderung. (5.19) und (5.20) sind wegen $\delta \leq \min\{\delta_1, \delta_2\}$ sicher erfüllt. Sei nun $(i, j) \in E$. Falls $b(i) = b(j)$, dann ist

$$\begin{aligned} u'_i + u'_j + \sum\{z'_S \mid (i,j) \in \gamma(S)\} & \\ &= (u_i + \Delta)(u_j + \Delta) + \sum\{z_S \mid (i,j) \in \gamma(S), S \neq S_{b(i)}\} \\ &\quad + (z_{S_{b(i)}} - 2\Delta) \\ &= u_i + u_j + \sum\{z'_S \mid (i,j) \in \gamma(S)\} \\ &\geq c_{ij} \end{aligned} \tag{5.25}$$

wobei $\Delta = -\delta$, falls $b(i)$ ein S–Knoten, $\Delta = \delta$, falls $b(i)$ ein T–Knoten und $\Delta = 0$ sonst ist.

Falls $b(i) \neq b(j)$ so ist

$$\sum\{z'_S \mid (i,j) \in \gamma(S)\} = \sum\{z_S \mid (i,j) \in \gamma(S)\}. \tag{5.26}$$

In diesem Fall ist nur $u'_i = u_i - \delta, u'_j \neq u_j + \delta$ näher zu betrachten, also $b(i)$ ist S–Knoten und $b(j)$ kein T–Knoten.

Falls $b(j)$ ein S–Knoten ist, so ist $(i, j) \notin E'$ und

$$\begin{aligned} u'_i + u'_j + \sum\{z'_S \mid (i,j) \in \gamma(S)\} - c_{ij} & \\ &= u_i + u_j + \sum\{z_S \mid (i,j) \in \gamma(S)\} - c_{ij} - 2\delta \\ &\geq u_i + u_j + \sum\{z_S \mid (i,j) \in \gamma(S)\} - c_{ij} - 2\delta_3 \\ &\geq 0 \end{aligned} \tag{5.27}$$

Falls $b(j)$ kein S–Knoten ist, so ist $b(j)$ ein Knoten von G^{rest}. Analog zum Fall $b(j)$ S–Knoten, ist (5.18) wegen der Wahl von δ_4 auch in diesem Fall erfüllt.

(b) *Die KSB (5.21') und (5.23) sind immer erfüllt und für alle $S \subseteq V$ mit $z_S > 0$ ist $(S, \gamma(S))$ im Graphen $(V, E' := \{e \in E \mid d_e = 0\})$ eine Blüte bezüglich M.*

Die Initiallösung erfüllt die Bedingungen. Nehmen wir an, dass die Bedingungen zu Beginn der repeat–Schleife gelten.

Bei einer Vergrösserung des Matchings werden für M' nur Kanten aus E' verwendet, also gilt (5.21') für M'. Da in G' alle $(S, \gamma(S))$ mit $z_S > 0$ geschrumpft worden sind und nach Annahme ein solches $(S, \gamma(S))$ eine Blüte im Graphen $(V, E' := \{e \in E \mid d_e = 0\})$ bezüglich dem alten Matching war, kann M' zu einem Matching von G so erweitert werden, dass jedes $(S, \gamma(S))$ mit $z_S > 0$ eine Blüte im Graphen $(V, E' := \{e \in E \mid d_e = 0\})$ bezüglich dem neuen Matching ist. Somit benützt auch M nur Kanten aus E', also gilt (5.21'), und da jedes $(S, \gamma(S))$ mit $z_S > 0$ eine Blüte bezüglich M ist, gilt (5.23).

Bei Dualvariablenveränderungen ändert d_e für Matchingkanten nicht: Sei $(i, j) \in M$. Dann ist entweder $b(i) = b(j)$ oder $b(i)$ ist S–Knoten und $b(j)$ ist T–Knoten oder $b(i)$ und $b(j)$ sind beide Knoten aus G^{rest}. In allen drei Fällen ändert d_e nicht (siehe (a), also bleibt (5.21') erhalten. Falls für ein $S \subseteq V$ neu $z_S > 0$ wird, so ist $(S, \gamma(S))$ eine Blüte in G' bezüglich M', also ist auch $(S, \gamma(S))$ eine Blüte in $(V, E' := \{e \in E \mid d_e = 0\})$ bezüglich M und (5.23) gilt auch nach der Dualvariablenveränderung.

Somit gelten die Bedingungen auch am Ende der repeat–Schleife.

(c) *Endlichkeit*

Zwischen zwei aufeinander folgenden Matching Vergrösserungen können nur endlich viele Dualvariablenvergrösserungen stattfinden:

$\delta = \delta_4$: Ein Knoten aus G^{rest} kommt neu zu einem gepflanzten Baum. Dies kann höchstens $|V|$ mal geschehen.

$\delta = \delta_3$: Zwei S–Knoten werden neu durch eine Kante mit $d_e = 0$ verbunden. Falls die beiden S–Knoten zu verschiedenen gepflanzten Bäumen gehören, ist ein vergrössernder Weg, sonst eine Blüte gefunden worden. Im zweiten Fall wird die Blüte zu einem S–Knoten geschrumpft und bis zur nächsten Vergrösserung nicht mehr expandiert. Dadurch nimmt die Anzahl Knoten in G'_c um mindestens 2 ab. Somit kann $\delta = \delta_3$ höchstens $|V|/2$ mal auftreten.

$\delta = \delta_2$: Sei $\mathbf{S}' := \{S \subseteq V \mid z_S > 0$ unmittelbar nach der letzten Vergrösserung$\}$. Es gilt dann $\mathbf{S}_T \subseteq \mathbf{S}'$, da alle während des Aufbaus der gepflanzten Bäume neu geschrumpften Blüten S-Knoten sind. Ist $\delta = \delta_2$, so wird ein Pseudonoten v mit $S_v \in \mathbf{S}_T \subseteq \mathbf{S}'$ expandiert. Dies kann nur $|S'| \leq |V|/2$ mal geschehen.

Da das Matching höchstens $|V|/2$ mal vergrössert wird, tritt der Fall $\delta = \delta_1$ nach endlich vielen Schritten auf und der Algorithmus stoppt.

(d) Optimalität

Alle exponierten Knoten sind immer S–Knoten von G'_c. Somit wurde der u–Wert eines exponierten Knotens bei jeder Dualvariablenveränderung reduziert und ist deshalb gleich $\min\{u_v \mid v \in V\}$. Ist $\delta = \delta_1$ so werden bei dieser Dualvariablenveränderung alle u–Werte der exponierten Knoten auf 0 gesetzt und damit gilt die KS (5.22') für alle Knoten. Deshalb ist (u, z) eine optimale Lösung vom dualen LP (5.17) - (5.19) und der Inzidenzvektor von M eine optimale Lösung vom LP (5.1)–(5.10). Also ist M ein Matching maximalen Gewichts.

Beispiel 5.8

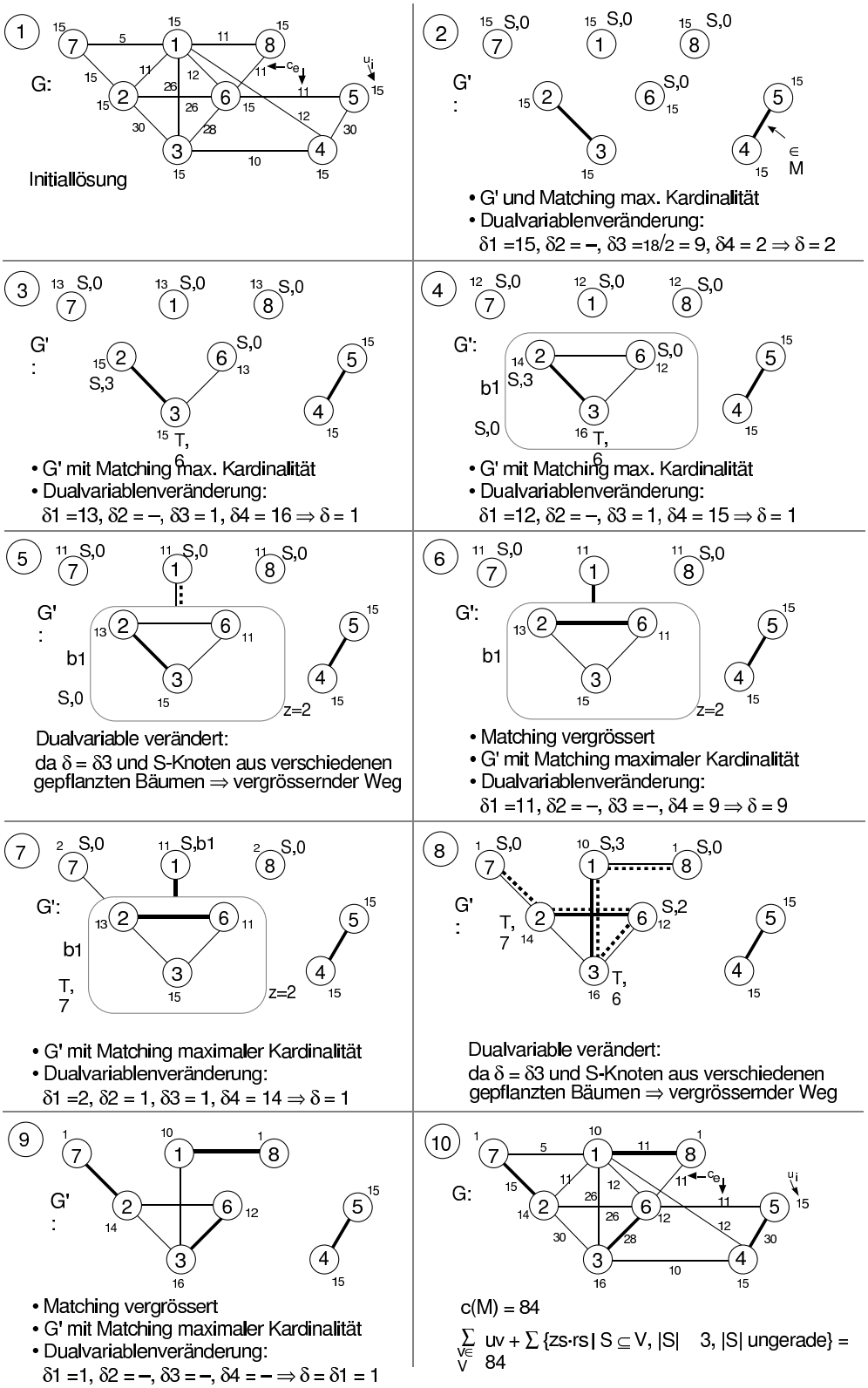


Abbildung 5.4: Matching maximalen Gewichts

$\diamond$

Kapitel 6

Das Chinese Postman Problem

In diesem Abschnitt können (und werden auch oft) die erwähnten Graphen Multigraphen (mit Parallelkanten bzw. Parallelbögen) sein. Wir beginnen mit der Definition eines *Euler-Kreises*.

Definition 6.1 (Euler'scher Kreis, Euler'scher Zyklus) *Sei $G = (V, E)$ ein ungerichteter Graph. Ein* Euler'scher Kreis *oder eine* Euler'sche Linie *ist eine geschlossene Kantenfolge von lauter verschiedenen Kanten, die sämtliche Kanten von G enthält. Falls G ein gerichteter Graph ist, heisst eine geschlossene Bogenfolge lauter verschiedener Bögen, die sämtliche Bögen von G enthält,* Euler'scher Zyklus.

Beispiel 6.2

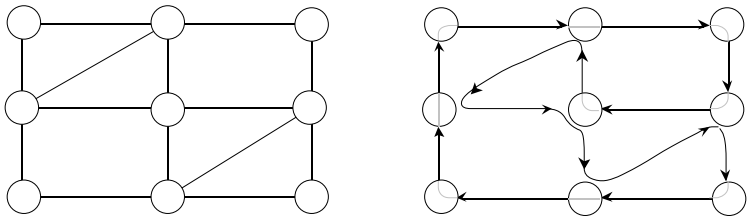


Abbildung 6.1: Euler'scher Kreis

◇

Beispiel 6.3

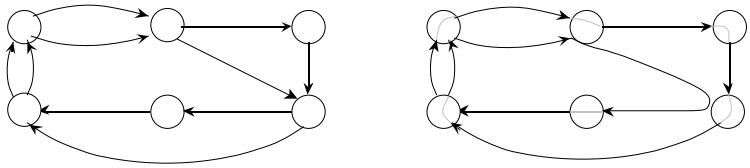


Abbildung 6.2: Euler'scher Zyklus

◇

Nicht jeder Graph besitzt einen Euler'schen Kreis (Zyklus). Falls ein Graph einen solchen besitzt, heisst er *Euler–Graph*. In einem ungerichteten Graphen $G = (V, E)$ sei $d(v)$ der Grad von v, $v \in V$ (Anzahl in v inzidierender Kanten). Es gilt:

Satz 6.4 *Ein zusammenhängender ungerichteter Graph $G = (V, E)$ ist ein Euler–Graph, falls und nur falls für alle Knoten v $d(v)$ eine gerade Zahl ist.*

In einem gerichteten Graphen sei d_v^+der Ausgangsgrad von v und d_v^- der Eingangsgrad von v für $v \in V$ (Anzahl aus v herausgehender bzw. hineingehender Bögen). Es gilt:

Satz 6.5 *Ein zusammenhängender gerichteter Graph $G = (V, E)$ ist ein Euler–Graph, falls und nur falls für alle Knoten $v \in V, d_v^+ = d_v^-$.*

Beweis von Satz 6.4. Dass in einem Euler–Graphen $d(v)$ für alle v gerade sein muss, folgt daraus, dass ein Euler–Kreis einen Knoten v gleich oft angeht wie verlässt. Umgekehrt werde induktiv über die Anzahl Kanten argumentiert. Falls $G = (V, E)$ zusammenhängend und $d_v = 2$ für alle $v \in V$ ist, ist G ein gewöhnlicher Kreis (Verankerung). Sei $G = (V, E)$ mit d_v gerade, $v \in V$. Von einem beliebigen Knoten v startend, schreite längs einer Kantenfolge vor, bis ein Kreis K gebildet wird. Entferne K aus E. In $G' = (V, E - K)$ sind alle Knotengrade gerade, und aus Induktionsannahme sind alle zusammenhängenden Komponenten G'_i von G' Euler–Graphen mit Euler–Kreis etwa EK_i. Durch richtiges Zusammenketten der EK_i's und K erhält man einen Euler–Kreis für G.

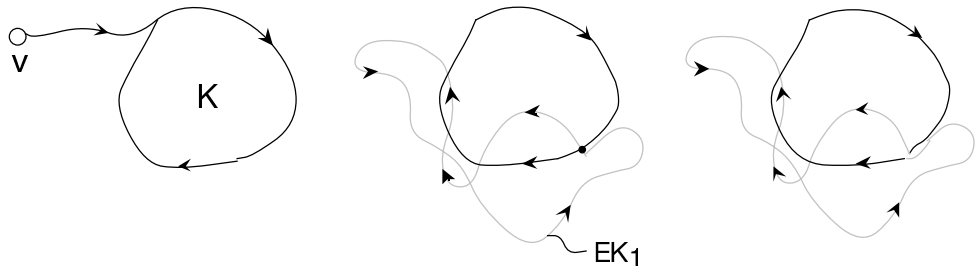


Abbildung 6.3: Zusammenketten von K und EK_1. Zusammenketten von K und EK_1. Kette analog $(K \cup EK_1)$ mit EK_2 zusammen, usw.

♣

Bemerkung 6.6 Der Beweis von Satz 6.5 führt man analog zum Beweis von Satz 6.4 (Übung).

Algorithmus 6.1 Euler-Kreis (Fleury-Regel)

---

begin (* Euler-Kreis *)

Starte von einem beliebigen Knoten aus und lösche eine Kante, nachdem sie durchschritten wurde. Durchschreite aber nie eine Kante mit Endknoten v, w von v nach w, falls durch ihr Löschen zwei neue Komponenten K_v und K_w entstehen, mit $v \in K_v \neq \{v\}$ und $w \in K_w$.

end (* Euler-Kreis *)

Man betrachte nun das Chinese Postman Problem:

Chinese Postman Problem (im ungerichteten Graphen):

Gegeben: $G = (V, E)$ ungerichtet, $c_e \geq 0, e \in E$ (Kantenlängen).

Gesucht: Eine geschlossene Kantenfolge F, die jede Kante $e \in E$ mindestens einmal enthält, und so, dass die Gesamtlänge minimal ist. (Dabei bezeichnet $\sum_{e \in E} n_e c_e$ die Anzahl der Durchschreitungen von e in F.)

Falls G ein Euler–Graph ist, ist obiges Problem leicht, und dessen Lösung ist einen Euler–Kreis der Länge $\sum_{e \in E} c_e$.

Andernfalls müssen in G bestimmte Kanten durch eine „Chinese Postman–Tour“ F mehrmals durchlaufen werden, was man durch einen Multigraphen G_F mit Parallelkanten darstellen kann:

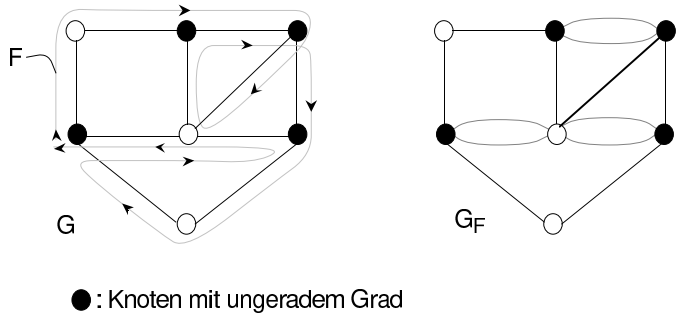


Abbildung 6.4: Graph G und Multigraph G_F mit Parallelkanten

Sei U die Menge der zusätzlich eingeführten Kanten in G_F (gestrichelt in obiger Abbildung). Es ist leicht zu zeigen, dass U als die Vereinigung der Kantenmengen von kantendisjunkten Wegen, die paarweise Knoten mit ungeradem Grad verbinden, angenommen werden darf. Man benützt dabei die Eigenschaft, dass in einem beliebigen Graphen $H = (W, D)$ mit Menge von Knoten mit geradem (ungeradem) Grad $W_g(W_u)$, gilt:

$$2|D| = \sum\{d_w \mid w \in W_g\} + \sum\{d_w \mid w \in W_u\} \tag{6.1}$$

Daher sind $\sum\{d_w \mid w \in W_u\}$ und somit $|W_u|$ gerade. Man darf annehmen, dass die genannten Wege keine Parallelkanten haben: Die Wege P_{vx} und

P_{wy} in der Abbildung 6.5 sind bei nicht negativen Kantenlängen zusammen mindestens so kurz wie P_{vw} und P_{xy} zusammen.

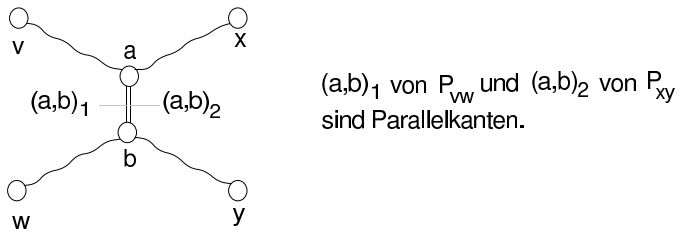


Abbildung 6.5: Keine Parallelkanten auf den Wegen

Dies bedeutet, dass immer eine optimale Tour existiert, bei der der Briefträger (Postman) höchstens zweimal einen Strassenzug durchschreiten muss.

Das Lösungsvorgehen sollte nun klar sein: Sei $G = (V, E)$ ein ungerichteter Graph mit Gewichten $c_e, e \in E$. Gesucht ist eine geschlossene Kantenfolge minimaler Länge, die jede Kante mindestens einmal enthält.

Algorithmus 6.2 Chinese Postman Problem

begin (* Chinese Postman *)
bestimme $V_u := \{v \in V \mid d(v)$ ungerade$\}$
if $V_u \neq \emptyset$ **then**
bestimme den kürzesten Weg d_{vw} für alle Paare $\{v, w\}$ mit $v, w \in V_u, v \neq w$
Sei $G_u = (V_u, E_u)$ vollständig, also $E_u := \{(v, w) \mid v \neq w, v, w \in V_u\}$
Bestimme in G_u mit Gewichten $d_{vw}, (v, w) \in E_u$ ein perfektes Matching M^*
minimalen Gewichts (siehe Bemerkung 6.7)
Verdopple in $G = (V, E)$ alle Kanten, welche auf den kürzesten Wegen vorkommen, die M^* entsprechen
Sei $G' = (V', E')$ der resultierende Graph (* G' ist Euler Graph *)
else $G' := G$
bestimme in G' einen Euler–Kreis mit Hilfe der Fleury–Regel (Algorithmus 6.1)
end (* Chinese Postman *)

Bemerkung 6.7 Ein perfektes Matching M im Graphen $G = (V, E)$ ist ein Matching, das alle Knoten bedeckt, also hier $|M| = |V|/2$.

Anwendungen des Chinese Postman–Problems sind bei Tourenproblemen (Post– oder Zeitungslieferungen, Inspektion von Netzen, Einsammeltouren, Schneeräumung und Salzen, usw.) zu finden. Es gibt auch eine gerichtete Version des Problems (siehe Übungen) sowie eine gemischte (gerichtet/ungerichtet) Version.

Kapitel 7

Flüsse in Netzwerken

7.1 Das Problem des maximalen Flusses

Definition 7.1 (Netzwerk) *Ein gerichteter Graph $G = (V, E)$, zusammen mit vorgegebenen nicht–negativen unteren und oberen Schranken $\ell_e$und k_e, $\ell_e \leq k_e$, für jeden Bogen $e \in E$ ist ein* Netzwerk.
Die Schranke ℓ_e heisst untere Kapazitätsschranke, k_e heisst obere Kapazitätsschranke oder kurzerhand Kapazität *des Bogens e, $e \in E$.*

Nicht selten sind die unteren Schranken $\ell_e = 0$ für alle $e \in E$. Es vereinfachen sich dann Formulierungen und Resultate. Auf den Fall $\ell_e = 0$ für alle $e \in E$ wird jeweils in Bemerkungen eingegangen.

Definition 7.2 (Fluss) *Oft sind zwei Knoten s, t der Knotenmenge V ausgezeichnet: s ist die* Quelle, *t die* Senke. *Man definiert dann als* Fluss *von s nach t oder s–t–*Fluss *einen Vektor $x \in \mathbb{R}^E$, der*

$$\ell_e \leq x_e \leq k_e, e \in E \tag{7.1}$$

und

$$\sum_{e \in \delta(v)} x_e - \sum_{e \in \delta(\overline{v})} x_e = \begin{cases} +\sigma, & \textit{falls } v = s \\ 0, & \textit{falls } v \in V - \{s, t\} \\ -\sigma, & \textit{falls } v = t \end{cases} \tag{7.2}$$

erfüllt. Dabei gelten die Bezeichnungen für $S \subseteq V$ beliebig:

$$\overline{S} \equiv V \setminus S,\ \delta(S) \equiv \{e \in E \mid t(e) \in S, h(e) \in \overline{S}\}, \tag{7.3}$$

und für die einelementige Menge $\{v\}$ wird kurzerhand v geschrieben. $\delta(v)$ ist also die Menge der aus v herausgehenden Bögen, $\delta(\overline{v})$ die Menge der in v hineingehenden Bögen. σ in (7.2) kann zunächst als ein Parameter aufgefasst werden und ist die Stärke des Flusses x.

Beispiel 7.3

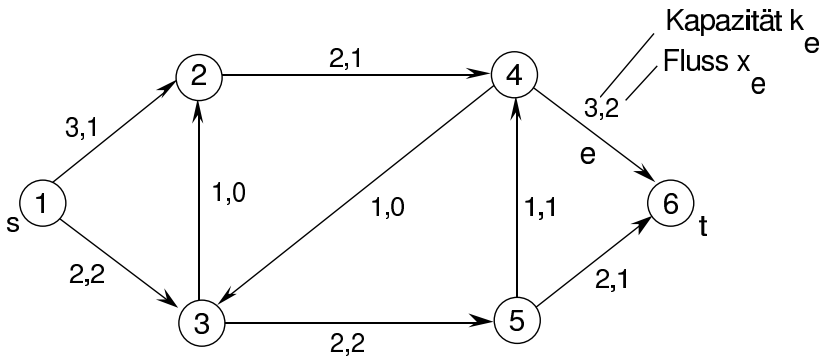


Abbildung 7.1: Beispiel: alle ℓ_e sind hier $= 0$

◇

Bemerke, dass bei gegebenem σ nicht immer ein s–t–Fluss der Stärke σ existiert. Es kann sogar sein, dass überhaupt kein (zulässiger) s–t–Fluss, gleichgültig welcher Stärke, existiert.
Hier werden wir zunächst annehmen, dass ein zulässiger s–t–Fluss existiert und vorliegt, und stellen uns das Problem des maximalen Flusses. Bemerke, dass falls $\ell_e = 0$ für alle $e \in E$, ein s–t–Fluss (der Stärke 0) existiert , nämlich $x = 0$.

Problem des maximalen s–t–Flusses:

Gegeben: Ein Netzwerk $G = (V, E), 0 \leq \ell_e \leq k_e$, $e \in E$, $s, t \in V$, mit zulässigem Initial– s–t–Fluss x.

Gesucht: Ein s–t–Fluss maximaler Stärke σ.

Offensichtlich ist dieses Problem ein Problem der linearen Programmierung, mit Variablen $x_e, e \in E$ und σ, nämlich *maximiere σ, so dass (7.1) und (7.2)* eingehalten werden. Allerdings wird es – effizienter als mit dem Simplex–Verfahren – mit einem rein kombinatorischen Algorithmus gelöst.

Definition 7.4 (s-t-Schnitt) *Man definiert einen* s–t–Schnitt *als eine Knotenmenge $S \subseteq V$ mit $s \in S$ und $t \notin S$. Die* Kapazität *des s–t–Schnittes S*

ist definiert als

$$k(S) - \ell(\overline{S}) \equiv \sum_{e \in \delta(S)} k_e - \sum_{e \in \delta(\overline{S})} \ell_e. \tag{7.4}$$

Wählt man im Beispiel $S = \{1, 3, 4\}$, so sind $k(S) = 9, \ell(\overline{S}) = 0$.

Lemma 7.5 *Für jeden s–t–Fluss $x \in \mathbb{R}_+^E$ der Stärke σ und jeden s–t–Schnitt S gilt $\sigma \leq k(S) - \ell(\overline{S})$.*

Beweis. Addiere alle Gleichungen (7.2) für $v \in S$:

$$\sigma = \sum_{v \in S} \left\{ \sum_{e \in \delta(v)} x_e - \sum_{e \in \delta(\overline{v})} x_e \right\} \tag{7.5}$$

$$= \sum_{v \in S} \left\{ \sum_{\substack{t(e)=v \\ h(e) \in S}} x_e + \sum_{\substack{t(e)=v \\ h(e) \in \overline{S}}} x_e - \sum_{\substack{h(e)=v \\ t(e) \in S}} x_e - \sum_{\substack{h(e)=v \\ t(e) \in \overline{S}}} \right\} \tag{7.6}$$

$$= \underbrace{\sum_{v \in S} \left\{ \sum_{\substack{t(e)=v \\ h(e) \in S}} x_e - \sum_{\substack{h(e)=v \\ t(e) \in S}} x_e \right\}}_{0} + \underbrace{\sum_{v \in S} \left\{ \sum_{\substack{t(e)=v \\ h(e) \in \overline{S}}} x_e - \sum_{\substack{h(e)=v \\ t(e) \in (\overline{S})}} x_e \right\}}_{\leq \sum_{e \in \delta(S)} x_e - \sum_{e \in \delta(\overline{S})} x_e} \tag{7.7}$$

und es folgt

$$\sigma = \sum_{e \in \delta(S)} x_e - \sum_{e \in \delta(\overline{S})} x_e \leq k(S) - \ell(\overline{S}). \tag{7.8}$$

♣

Siehe für einen Augenblick von den Richtungen auf den Bögen von G ab und betrachte einen (ungerichteten) Weg W von s nach t. In G wiederum gerichtet, ist $e \in W$ ein *Vorwärtsbogen (Rückwärtsbogen)* von W, falls die Richtung von e gleich (entgegengesetzt) der Richtung von W ist, wenn man W von s nach t abschreitet.

Definition 7.6 (Flussvergrössernder Weg) *Sei x ein s–t–Fluss. Ein* flussvergrössernder Weg *FW bezüglich x ist ein ungerichteter Weg von s nach t im obigen Sinn, so dass*

$$x_e < k_e \text{ für alle Vorwärtsbögen von } FW, \tag{7.9}$$
$$\ell_e < x_e \text{ für alle Rückwärtsbögen von } FW. \tag{7.10}$$

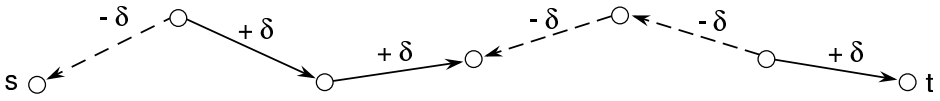


Abbildung 7.2: Flussvergrössender Fluss bzgl.s–t–Fluss

Falls FW ein flussvergrössernder Weg bezüglich x der Stärke σ ist, ist x' — erhalten durch

$$x'_e := x_e + \delta, \; e \text{ Vorwärtsbogen (von FW)}, \tag{7.11}$$
$$x'_e := x_e - \delta, \; e \text{ Rückwärtsbogen}, \tag{7.12}$$
$$x'_e := x_e, \quad e \notin FW \tag{7.13}$$

mit

$$\delta = \min\{\{k_e - x_e \mid e \text{ Vorwärtsbogen }\}; \{x_e - \ell_e \mid e \text{ Rückwärtsbogen}\}\} \tag{7.14}$$

wieder ein s–t–Fluss mit Stärke $\sigma + \delta$. Man sagt, x sei mit Hilfe von FW *vergrössert* worden.

Beispiel 7.7

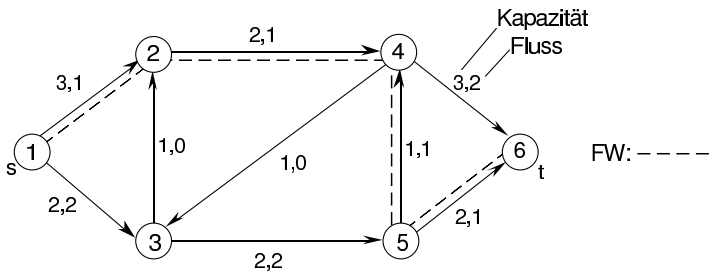


Abbildung 7.3: Flussvergrössender Fluss anhand eines Beispiels

◇

Satz 7.8 *Ein s–t–Fluss x ist maximal, falls und nur falls kein flussvergrössernder Weg bezüglich x existiert.*

Beweis. Falls x maximal ist, gibt es offensichtlich kein FW bezüglich x. Umgekehrt nehme an, es gebe kein FW bezüglich x und sei S die Menge aller Knoten $v \in V$, für die ein FW von s nach v bezüglich x existiert. Natürlich ist $t \notin S$. Dann gilt für alle $e \in \delta(S)\, x_e = k_e$, ansonsten $h(e) \in S$, und für alle $e \in \delta(\overline{S})\, x_e = \ell_e$, ansonsten $t(e) \in S$. Es folgt aus (7.8), dass $\sigma = k(S) - \ell(\overline{S})$ die Stärke σ maximal und S ein Schnitt minimaler Kapazität $k(S) - \ell(\overline{S})$ sind. ♣

Satz 7.9 (Max Flow – Min Cut Satz von Ford und Fulkerson) *Die maximale Stärke eines s–t–Flusses ist gleich der minimalen Kapazität eines s–t–Schnittes.*

Beweis. Dieser Satz folgt direkt aus dem Beweis von Satz 7.8. ♣

7.2 Ein Algorithmus für einen maximalen *s–t–*Fluss

Ausgehend von einem s–t–Fluss vergrössert der Algorithmus von Ford/Fulkerson wiederholt den jeweiligen Fluss x, indem ein FW bezüglich x mit einem Markierungsverfahren gesucht wird. Findet der Algorithmus keinen FW bezüglich x der Stärke σ, so weist er einen s–t–Schnitt S aus, mit $k(S) - \ell(\overline{S}) = \sigma$. Auf Grund des Lemmas 7.5 ist σ maximal. (Beachte Analogie mit dem Algorithmus für ein Matching maximaler Kapazität im bipartiten Graphen.)
O.E.d.A. darf G schlicht angenommen werden (keine Schlingen und Parallelbögen). Ein Bogen e wird mit $(i, j), i = t(e), j = h(e)$, und x_e, ℓ_e, k_e mit x_{ij}, ℓ_{ij} und k_{ij} bezeichnet.

Algorithmus 7.1 Maximaler Fluss

(* Gegeben ist ein Netzwerk $G = (V, E)$ mit Schranken $0 \leq \ell_e, k_e, e \in E; s, t \in V$, und ein zulässiger Initialfluss x. Der Algorithmus findet einen s–t–Fluss maximaler Stärke σ und einen s–t–Schnitt minimaler Kapazität *)

begin (* maximaler Fluss *)

Sei x ein zulässiger Initialfluss der Stärke σ'

$\sigma := \sigma'$

repeat

(* flussvergrössernden Weg suchen *)

marke$[v] := [-1, \infty]\ \forall v \in V \setminus s$ (* 1. Komp = woher kommt flussvergr. Weg*)

marke$[s] := [\emptyset, \infty]$ (* 2. Komp = um wieviel kann Fluss vergrössert werden*)

inspKnoten $:= \{s\}$

repeat

sei $i \in$ inspKnoten; inspKnoten := inspKnoten $\setminus i$

(* Vorwärtsbögen *)

for all $(i, j) \in E$, mit marke$[j] = [-1, \infty]$ und $x_{ij} < k_{ij}$ **do**

marke$[j] := [i^+, \delta_j]$ mit $\delta_j := \min\{\delta_i, k_{ij} - x_{ij}\}$

inspKnoten := inspKnoten $\cup j$

end (* for *)

(* Rückwärtsbögen *)

for all $(j, i) \in E$ mit marke$[j] = [-1, \infty]$ und $x_{ji} > \ell_{ji}$ **do**

marke$[j] := [i^-, \delta_j]$ mit $\delta_j := \min\{\delta_i, x_{ji} - \ell_{ji}\}$

inspKnoten := inspKnoten $\cup j$

end (* for *)

until (marke$[t] \neq [-1, \infty]$) or (inspKnoten $= \emptyset$)

if marke$[t] \neq [-1, \infty]$ **then** (* Fluss vergrössern *)

Identifiziere flussvergrössernder Weg FW von t an mit Hilfe der Markierung

(marke$[j] = [i^+, \delta_j] \Rightarrow (i, j)$ ist Vorwärtsborgen,

marke$[j] = [i^-, \delta_j] \Rightarrow (j, i)$ ist Rückwärtsbogen)

Vergrössere den Fluss um δ_t auf Vorwärtsbögen und verringere ihn um δ_t auf

Rückwärtsbögen

$\sigma := \sigma + \delta_t$

end (* if *)

until marke$[t] = [-1, \infty]$

der Fluss x ist maximaler Stärke σ

$S := \{v \in V \mid \text{marke}[v] \neq [-1, \infty]\}$

S ist s–t–Schnitt minimaler Kapazität und $\sigma = k(S) - \ell(\overline{S})$

end (* maximaler Fluss *)

Beispiel 7.10

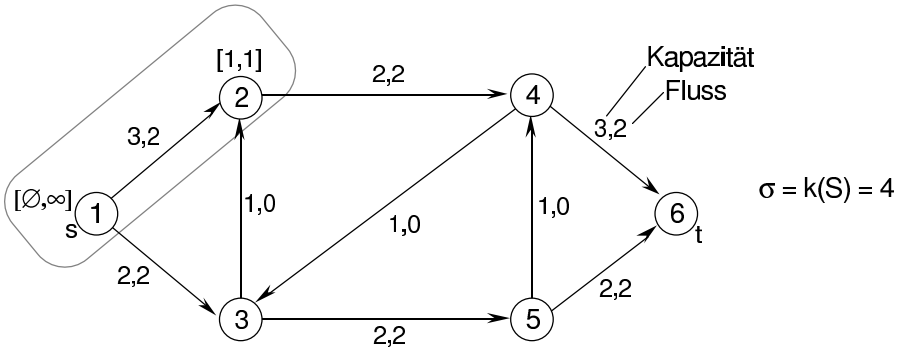


Abbildung 7.4: Beispiel nach Benutzung des FW von Abbildung 7.3

◇

Endlichkeit und Aufwand. Falls die Kapazitäten rationale Zahlen sind, ist der Algorithmus endlich, jedoch kann sein Aufwand gross sein:

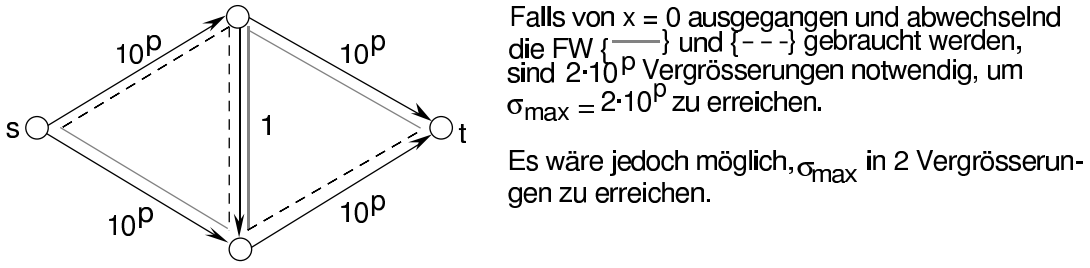


Abbildung 7.5: unterschiedlicher Aufwand des Algorithmus

Es sind aber höchstens $|E| \cdot |V|/2$ Vergrösserungen notwendig, falls jeweils flussvergrössernde Wege mit einer minimalen Anzahl Bögen benützt werden. Dies ist im gegebenen Algorithmus sehr einfach zu bewerkstelligen, nämlich: Inspiziere die Knoten in einer Reihenfolge entsprechend der Regel „Wer zuerst markiert, wird zuerst inspiziert“. Da pro Vergrösserung höchstens $2|E|$ Bogeninspektionen erfolgen, ist der Gesamtaufwand $0(|E|^2 \cdot |V|)$. ♣

7.3 Inkrementgraph und vergrössernde Zyklen

Betrachte im Netzwerk $G = (V, E)$, $0 \leq \ell_e \leq k_e$, $e \in E$, $s, t \in V$, einen s–t–Fluss x und bezeichne für einen beliebigen Bogen e mit $\overline{e}$ den Bogen mit denselben Extremitäten und entgegengesetzter Richtung.

Definition 7.11 (Inkrementgraph) *Der* Inkrementgraph $G^x = (V, E^x)$ *ist definiert durch*

$$E^x = E^x_+ \cup E^x_- \tag{7.15}$$

wobei

$$E^x_+ = \{e \mid e \in E \text{ und } x_e < k_e\} \tag{7.16}$$
$$E^x_- = \{e \mid \overline{e} \in E \text{ und } \ell_{\overline{e}} < x_{\overline{e}}\}. \tag{7.17}$$

Definition 7.12 (Kapazität) *Die* Kapazität q_e *von* $e \in E^x$ *ist*

$$q_e = k_e - x_e : \qquad e \in E^x_+, \tag{7.18}$$
$$q_e = x_{\overline{e}} - \ell_{\overline{e}} : \qquad e \in E^x_-. \tag{7.19}$$

Beispiel 7.13

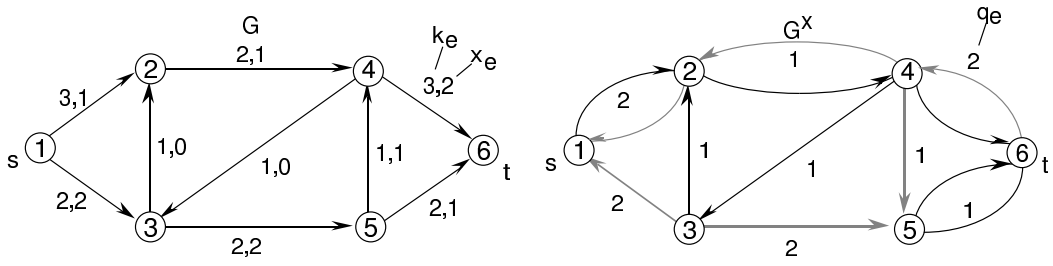


Abbildung 7.6: Konstruktion eines Inkrementgraphen (alle ℓ_e sind hier $= 0$)

◇

Offensichtlich entspricht jedem gerichteten Weg W von s nach t in G^x ein flussvergrössernder Weg FW bezüglich x, und umgekehrt.

Definition 7.14 (Flussvergrössernder Zyklus) *Ein* flussvergrössernder Zyklus *FZ bezüglich x in G ist wie folgt definiert:*

(i) FZ ist ein Kreis, wenn von den Richtungen auf den Bögen von G abgesehen wird;

(ii) Bei einem gewählten Orientierungssinn $\mathbf{O}$ *für FZ gilt:*

$$x_e < k_e, \text{ falls } e \text{ ein Vorwärtsbogen (bezüglich } \mathbf{O}\text{) von FZ ist,} \quad (7.20)$$

$$\ell_e < x_e, \text{ falls } e \text{ ein Rückwärtsbogen (bezüglich } \mathbf{O}\text{) von FZ ist.} \quad (7.21)$$

Bemerkung 7.15 Bemerke, dass ein Kreis ein FZ bezüglich einem Orientierungssinn sein kann, und nicht bezüglich dem entgegengesetzten Orientierungssinn. Diese Schwierigkeit wird durch die Benutzung des Inkrementgraphen G^x aufgehoben, denn jedem Zyklus Z in G^x entspricht ein flussvergrössernder Zyklus bezüglich x, und umgekehrt.
In Abbildung 7.6 ist $\{4 \to 5 \to 6 \to 4\}$ ein FZ, jedoch nicht $\{4 \to 6 \to 5 \to 4\}$.

Sei FW (FZ) ein flussvergrössernder Weg (Zyklus) bezüglich x, und $W(Z)$ der entsprechende Weg (Zyklus) in G^x. Man sagt, dass x mit Hilfe von FW (FZ) vergrössert wird, falls x zu x' geändert wird, wobei

$$x'_e = \begin{cases} x_e + \delta, & \text{falls } e \in W \cap E^x_+ (\in Z \cap E^x_+) \\ x_e - \delta, & \text{falls } \overline{e} \in W \cap E^x_- (\in Z \cap E^x_-) \\ x_e \quad , & \text{sonst} \end{cases} \quad (7.22)$$

und

$$0 < \delta \leq \min\{qe \mid e \in W\} (\in Z). \quad (7.23)$$

Bemerkung 7.16 Min $\{q_e \mid e \in W\} (\in Z)$ wird gelegentlich *Kapazität* von FW (FZ) genannt.

Ein gesamter Flussvergrösserungsschritt (Markierung und Flussvergrösserung) im Algorithmus 7.1 kann daher folgendermassen beschrieben werden:

Gegeben den jeweiligen Fluss x, bilde den entsprechenden Inkrementgraphen G^x, und suche darin einen gewöhnlichen Weg W von s nach t, und vergrössere x indem x zu x' entsprechend den eben besprochenen Regeln geändert wird.
Vergrösserungen mit Hilfe flussvergrössernden Zyklen werden später im Kapitel 8 für die Ermittlung kostenminimaler Flüsse gebraucht.

7.4 Preflow–push Algorithmus

Definition 7.17 (Preflow, Exzess) *Ein* Vorfluss (Preflow) x *ist ein Vektor* $x \in \mathbb{R}^E$*, welcher der Bedingung (7.1) sowie folgender Abschwächung von (7.2) genügt:*

$$\sum_{e \in \delta(\overline{v})} x_e - \sum_{e \in \delta(v)} x_e \geq 0, v \in V \setminus \{s,t\}. \tag{7.24}$$

Der Exzess $\varepsilon(v)$ *eines Knotens* v *wird definiert als*

$$\varepsilon(v) = \sum_{e \in \delta(\overline{v})} x_e - \sum_{e \in \delta(v)} x_e, v \in V \setminus \{s,t\}. \tag{7.25}$$

Offensichtlich ist jeder s–t–Fluss ein Vorfluss, und ein Vorfluss ist ein s–t–Fluss, falls der Exzess in jedem von s und t verschiedenen Knoten gleich Null ist.
Für einen beliebigen Vorfluss x lässt sich der Inkrementgraph $G^x = (V, E^x)$ wie für einen Fluss definieren (vgl. Abschnitt 7.3).

Definition 7.18 (gültige Distanzfunktion) *Sei* x *ein Vorfluss und* $G^x = (V, E^x)$ *der entsprechende Inkrementgraph. Eine Funktion* $d : V \to \mathbf{Z}^+$ *in* G^x *heisst* gültige Distanzfunktion*, falls folgende Bedingungen erfüllt sind:*

$$d(t) = 0 \tag{7.26}$$

$$d(t(e)) \leq d(h(e)) + 1 \qquad \textit{für alle Bögen } e \in E^x. \tag{7.27}$$

Definiere in G^x die *Länge* eines Weges von v nach w in $G^x, v, w \in V$, als die Anzahl seiner Bogen.

Lemma 7.19 *$d(v)$ ist eine untere Schranke für die minimale Länge eines Weges („Länge eines kürzesten Weges“) von v nach t, für beliebiges $v \in V$.*

Beweis. Seien $v = v_0, v_1, v_2, v_3, \ldots, v_k = t$ die (Folge der) Knoten eines Weges von v nach t der Länge k. Es gilt $d(v) = d(v_0) \leq d(v_1) + 1, d(v_1) \leq d(v_2) + 1, \ldots, d(v_{k-1}) \leq d(t) + 1 = 1$, daher also $d(v) \leq k$. ♣

Der nun dargestellte *Preflow–push Algorithmus* (Algorithmus 7.2) zur Bestimmung eines maximalen s–t–Flusses versucht, ausgehend von einem Vorfluss x, jeweils diesen Vorfluss so zu ändern, dass positive Flussexzesse näher

an die Senke t gebracht werden. Der Algorithmus arbeitet stets im Inkrementgraphen G^x, der natürlich (der Änderung von x entsprechend) nachgeführt wird.
Ein Knoten mit positivem Exzess in G^x heisst *aktiv*, mit der Konvention, dass s und t nie als aktiv gelten.
In G^x wird auch eine gültige Distanzfunktion d mit– und nachgeführt. Ein Bogen $e \in E^x$ heisst *zulässig* falls $d(t(e)) = d(h(e)) + 1$.
Der Algorithmus wählt nun in jeder Iteration einen aktiven Knoten und versucht seinen Exzess über einen zulässigen Bogen näher zur Senke t zu schicken. Ist dies nicht möglich, so wird die Distanz des betrachteten Knotens neu definiert, und zwar so erhöht, dass mindestens ein weiterer Bogen zulässig wird.

Wie in Kapitel 7.2 wird im folgenden G und entsprechend G^x als schlicht angenommen und für einen Bogen e mit $i = t(e), j = h(e)$, die Bezeichnung (i, j) verwendet. Gegeben sei nun ein Netzwerk $G = (V, E)$ mit Schranken $0 \leq \ell_e \leq k_e, e \in E$; $s, t \in V$, und ein zulässiger Initialfluss x. Der Preflow–Push–Algorithmus 7.2 findet einen s–t–Fluss maximaler Stärke.

Algorithmus 7.2 Preflow–Push

begin (* Preflow-Push *)
  (* Initialisierung *)
  sei x ein zulässiger s–t–Fluss
  bilde G^x mit Kapazitäten $q_{ij}, (i,j) \in E^x$
  berechne Distanzen $d(i), i \in V$ mittels BFS (breadth first search)
    ausgehend von $t(d(i) := n$, falls kein Weg von i nach t existiert
  if $d(s) \neq n$ **then**
    $x_{sj} := x_{sj} + q_{sj}$ für alle $(s,j) \in E^x_+$
    $x_{js} := x_{js} - q_{sj}$ für alle $(s,j) \in E^x_-$
    führe G^x nach
    $d(s) := n$
    while $\exists$ aktiver Knoten **do**
      sei i aktiver Knoten
      if $\exists$ zulässiger Bogen $(i,j) \in E^x$ **then**
        (* Flussstoss auf (i,j) *)
        if $(i,j) \in E^x_+$ **then** $x_{ij} := x_{ij} + \min\{\varepsilon_i, q_{ij}\}$
        else $x_{ji} := x_{ji} - \min\{\varepsilon_i, q_{ij}\}$
        führe G^x nach
      else (* Neumarkierung von i *)
        $d(i) := \min\{d(j) + 1 \mid (i,j) \in E^x\}$
      end (* else *)
    end (* while *)
  end (* if *)
  x ist ein maximaler s–t–Fluss
end (* Preflow–Push *)

Man bemerke, dass, falls in der Initialisierung s mittels BFS nicht markiert wurde, kein s–t–Weg in G^x existiert. Also existiert kein flussvergrössernder Weg bezüglich x und x ist maximal (Satz 7.8).

Lemma 7.20 *Falls Knoten i in G^x aktiv ist, existiert ein Weg von i nach s oder nach t in G^x.*

Beweis. Wir zeigen zuerst, dass für alle $S \subseteq V$ mit $s \notin S$ und $t \notin S$,

$k(S) \geq \ell(\overline{S})$ ist: Da ein zulässiger Fluss z existiert, gilt:

$$0 = \sum_{v \in S} \left(\sum_{e \in \delta(\overline{v})} z_e - \sum_{e \in \delta(v)} z_e \right) \tag{7.28}$$

$$= \sum_{e \in \delta(S)} z_e - \sum_{e \in \delta(S)} z_e \tag{7.29}$$

$$\geq \sum_{e \in \delta(S)} \ell_e - \sum_{e \in \delta(S)} k_e \tag{7.30}$$

$$= \ell(\overline{S}) - k(S) \tag{7.31}$$

also

$$k(S) \geq \ell(\overline{S}). \tag{7.32}$$

Sei nun i ein Knoten, für den kein Weg nach s oder t in G^x existiert. Wir zeigen, dass i nicht aktiv sein kann: Sei S die Menge der Knoten, die von i aus in G^x erreichbar sind. Da $s \notin S$ und $t \notin S$, gilt:

$$\begin{aligned} 0 \geq \ell(\overline{S}) - k(S) &= \sum_{e \in \delta(\overline{S})} \ell_e - \sum_{e \in \delta(S)} k_e = \sum_{e \in \delta(\overline{S})} x_e - \sum_{e \in \delta(S)} x_e \\ &= \sum_{v \in S} \left(\sum_{e \in \delta(\overline{v})} x_e - \sum_{e \in \delta(v)} x_e \right) = \sum_{v \in S} \varepsilon(v) \geq \varepsilon(i) \geq 0, \end{aligned} \tag{7.33}$$

denn $e \in \delta(S)$ $(\delta(\overline{S}))$ impliziert $e \notin E^x (\overline{e} \notin E^x)$ und somit $x_e = k_e (x_e = \ell_e)$. Also ist $\varepsilon(i) = 0$ und deshalb i nicht aktiv. ♣

Lemma 7.20 sichert, dass die Neumarkierung eines Knotens i (ersetzen von $d(i)$ durch $\min\{d(j) + 1 \mid (i,j) \in E^x\}$) wohldefiniert ist: es wird nicht über die leere Menge minimiert.

Lemma 7.21 *Die Distanzmarken $d(i)$ bilden zu jedem Zeitpunkt eine gültige Distanzfunktion, d.h. $d(i) \leq d(j) + 1$ für alle Bögen $(i,j) \in E^x$.*

Beweis. Nach der Initialisierung sind die $d(i)$'s gültig (für $d(s) = n$ trifft dies zu, weil kein Bogen mehr in G^x von s wegführt). In einem allgemeinen Schritt:

Fall 1: *Flussstoss auf (i, j) wird ausgeführt:*

Die Distanzmarken bleiben unverändert, aber im Inkrementgraphen kann ein neuer Bogen (j, i) hinzukommen, sowie (i, j) verschwinden. Letzteres ist bedeutungslos, da das Verschwinden eines Bogens die Gültigkeit einer Marke $d(i)$ nicht beeinträchtigen kann. Falls (j, i) ein neuer Bogen ist, so gilt (7.27) auch für ihn, denn, da der Stoss auf zulässigen Bogen erfolgt, ist $d(i) = d(j) + 1$, und somit $d(j) \leq d(i) + 1$.

Fall 2: *Knoten i wird neu markiert:*

Vor der Markierung gilt für alle $j \in \delta(i) d_{vor}(i) < d(j) + 1$. Nach Markierung von i ist $d_{vor}(i) < d_{nach}(i) \leq d(j)+1$ für alle $(i, j) \in E^x$. Wegen $d_{vor}(i) < d_{nach}(i)$ ist auch $d(k) \leq d_{nach}(i)$ für alle $(k, i) \in E^x$.

♣

Lemma 7.22 *Falls der Algorithmus endet, so findet er einen maximalen Fluss.*

Beweis. Bei Ende des Algorithmus gibt es keinen Knoten $i \neq s$ oder t, mit positivem Exzess $\varepsilon(i)$, d.h. der Vorfluss ist ein s–t–Fluss. $d(s) = n$ blieb konstant und ist eine gültige Distanzmarke, daher eine untere Schranke für den kürzesten Weg von s nach t. Da jeder s–t–Weg eine Länge $\leq (n - 1)$ hat, folgt dass im letzten Inkrementgraphen kein Weg von s nach t führt. Der Fluss ist deshalb ein maximaler Fluss (Satz 7.8). ♣

Um die Laufzeit genauer analysieren zu können, müssen wir uns die **while**–Schleife im Algorithmus 7.2 näher anschauen. Bei jedem Durchgang der **while**–Schleife tritt einer der folgenden drei Fälle ein:

(a) Kein Fluss kann näher an t gestossen werden, und die Distanzmarke des betrachteten Knotens wird erhöht *(Neumarkierung).*

(b) Ein Stoss, welcher die ganze Kapazität eines Bogens (i, j) beansprucht wird ausgeführt *(sättigender Stoss).*

(c) Ein *nicht–sättigender* Stoss wird ausgeführt.

Bemerke, dass bei einer Neumarkierung von i die Marke $d(i)$ immer erhöht wird. Seien $n := |V|$ und $m := |E|$.

Lemma 7.23

(i) Zu jedem Zeitpunkt gilt $d(i) < 2n$ für alle Knoten $i \in V$.

(ii) Es werden höchstens $2n^2$ Neumarkierungen ausgeführt.

Beweis.

(i) Die Behauptung gilt, wenn i nicht neumarkiert wurde. Sonst hatte i bei der letzten Neumarkierung einen positiven Exzess, und war deshalb mit s oder mit t über einen Weg der Länge $k \leq (n-1)$ verbunden. $d(s) = n$ und $d(t) = 0$ bleiben konstant und für einen Weg mit Knoten $i = i_0, i_1, i_2, i_3, \ldots, i_k = s$ oder t gilt:

$$d(i_{k-1}) \leq d(i_k) + 1, \tag{7.34}$$
$$d(i_{k-2}) \leq d(i_{k-1}) + 1 \leq d(i_k) + 2, \tag{7.35}$$
$$\vdots$$
$$d(i_0) \leq d(i_k) + k < 2n \tag{7.36}$$

(ii) Es gibt n Knoten in G^x, bei jeder Neumarkierung eines Knotens nimmt $d(i)$ zu und $d(i) < 2n$.

♣

Lemma 7.24

(i) Zwischen zwei sättigenden Stössen längs des gleichen Bogens (i, j) erhöht sich $d(i)$ um mindestens 2.

(ii) Die Anzahl sättigender Stösse ist höchstens $2nm$.

Beweis. Beim ersten Stoss gilt $d_0(i) = d_0(j) + 1$. Bevor auf (i, j) ein erneuter Stoss ausgeführt werden kann, muss ein Stoss längs (j, i) geschehen. Zu diesem Zeitpunkt gilt $d_1(j) = d_1(i) + 1 \geq d_0(i) + 1$. Beim zweiten Stoss längs (i, j) gilt $d_2(i) = d_2(j) + 1 \geq d_1(j) + 1 \geq d_0(i) + 2$. (ii) folgt unmittelbar aus (i) und Lemma 7.23. ♣

Lemma 7.25 *Die Anzahl nicht–sättigender Stösse ist höchstens $3n^2 + 4n^2m$.*

Beweis. Man betrachte folgende, sogenannte Potentialfunktion

$$P := \sum_{i \text{ aktiv}} d(i) \tag{7.37}$$

Die Auswirkungen der obigen Fälle (a), (b) und (c) auf den Wert von P sind wie folgt:

(a) P wird erhöht. Die Summe dieser Erhöhungen beträgt jedoch maximal $2n^2$ wegen Lemma 7.23.

(b) Ein sättigender Stoss kann einen neuen Knoten aktivieren, und daher P erhöhen, und zwar um maximal $2n$ (Lemma 7.23). Die Summe dieser Erhöhungen über alle sättigenden Stösse beträgt höchstens $4n^2m$ (Lemma 7.24).

(c) Ein nicht–sättigender Stoss längs (i, j) baut den Exzess in i voll ab und i wird inaktiv; deshalb verringert sich P um $d(i)$. Falls jedoch dieser Stoss den Knoten j aktiviert, so erhöht dies P gleichzeitig um $d(j)$. Da Stösse nur längs zulässigen Bogen gemacht werden (d.h. $d(j) = d(i) - 1$), verringert sich der Wert von P für einen nicht–sättigenden Stoss insgesamt mindestens um 1.

Der Startwert von P beträgt höchstens n^2 ($d(i) \leq n$ für alle $i \in V$), die Zunahme von P höchstens $2n^2 + 4n^2m$. Zudem ist $P \geq 0$. Wegen (c) gibt es höchstens $n^2 + 2n^2 + 4n^2m$ nicht–sättigende Stösse. ♣

Insgesamt führt also der Algorithmus höchstens: nm sättigende Stösse, $3n^2 + 4n^2m$ nicht–sättigende Stösse und $2n^2$ Neumarkierungen aus. Daraus ergibt sich folgendes Korollar:

Korollar 7.26 *Der Aufwand des Preflow–Push Algorithmus ist $O(n^2m)$.*

Bemerkung 7.27 Für die Implementierung siehe Übungenen.

Beispiel 7.28

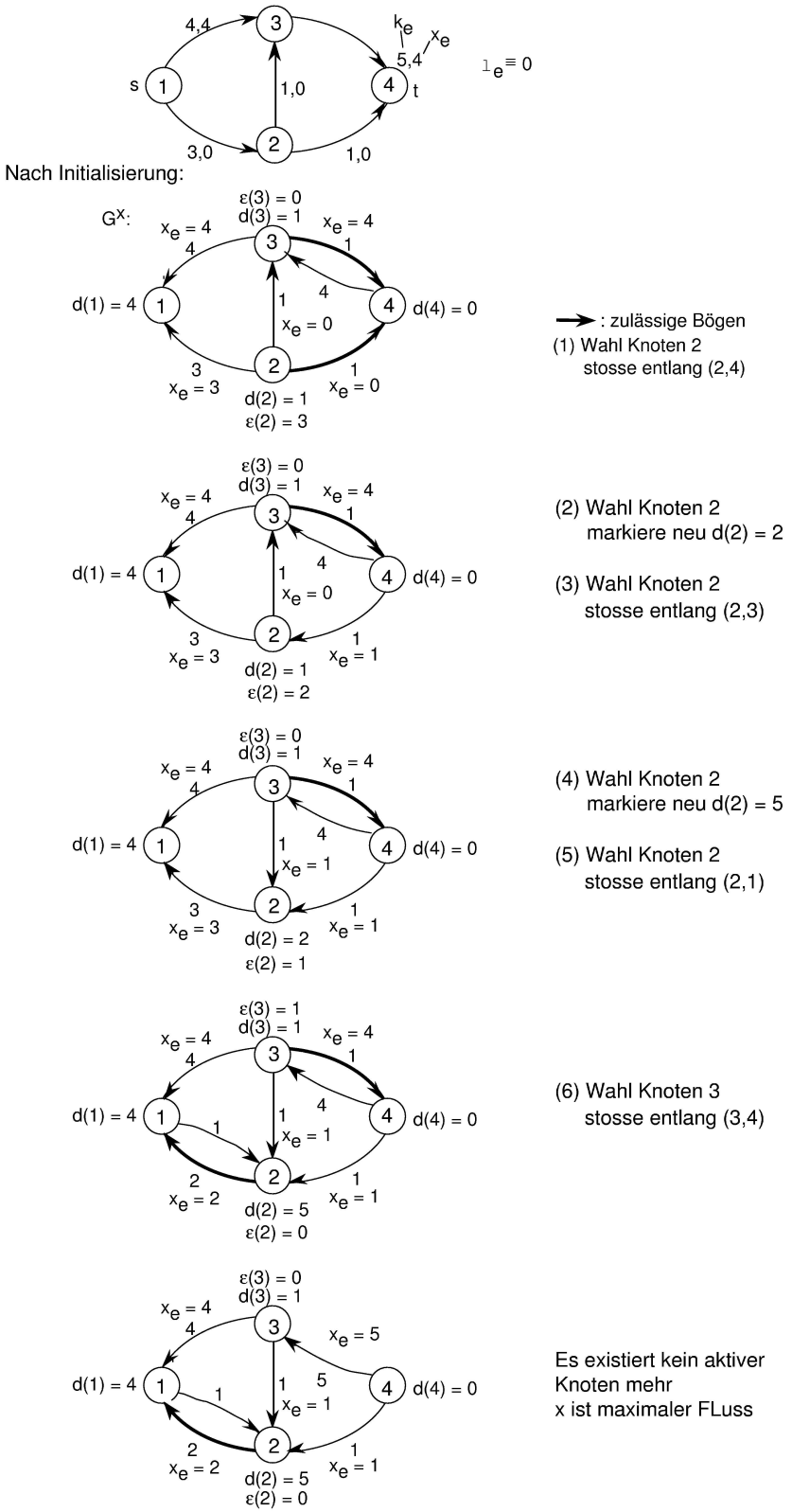


Abbildung 7.7: Preflow–Push Algorithmus

$\diamond$

7.5 Max Flow – min Cut und lineare Programmierung

Wie erwähnt, lässt sich das Problem des maximalen Flusses als LP formulieren. Der Max Flow – min Cut Satz folgt direkt aus dem LP–Dualitätssatz und das duale LP hat eine Interpretation. Dies sei im folgenden für den Fall $\ell_e = 0$ für alle $e \in E$ gezeigt. Das Problem des maximalen s–t–Flusses ist folgendem LP äquivalent:

$$\begin{aligned}
&\text{maximiere} && \sigma && && (7.38)\\
&\text{so dass} && \sum_{e\in\delta(\overline{v})} x_e - \sum_{e\in\delta(v)} x_e + \sigma = 0 && v = s && (7.39)\\
& && \sum_{e\in\delta(\overline{v})} x_e - \sum_{e\in\delta(v)} x_e = 0 && v \in V - \{s,t\} && (7.40)\\
& && \sum_{e\in\delta(\overline{v})} x_e - \sum_{e\in\delta(v)} x_e - \sigma = 0 && v = t && (7.41)\\
& && 0 \le x_e \le k_e && e \in E && (7.42)
\end{aligned}$$

Das duale LP ist demnach

$$\begin{aligned}
&\text{minimiere} && \sum_{e\in E} k_e w_e && && (7.43)\\
&\text{so dass} && u_{h(e)} - u_{t(e)} + w_e \ge 0, && e \in E && (7.44)\\
& && u_s - u_t \ge 1 && && (7.45)\\
& && w_e \ge 0 && e \in E && (7.46)\\
& && u_v \quad \text{frei} && v \in V && (7.47)
\end{aligned}$$

Jedem s–t–Schnitt S kann $\hat{u}, \hat{w}$ mit Komponenten $\hat{u}_v, \hat{w}_e$:

$$\hat{u}_v = \begin{cases} 1, & \text{für } v \in S \\ 0, & \text{sonst} \end{cases}, \qquad \hat{w}_e = \begin{cases} 1, & \text{für } e \in \delta(S) \\ 0, & \text{sonst} \end{cases}, \tag{7.48}$$

zugeordnet werden. $\hat{u}, \hat{w}$ ist eine zulässige duale Lösung und $k(S) = \sum k_e \hat{w}_e$ (prüfe nach!). Umgekehrt gibt es immer eine optimale duale Lösung, so dass

auf obiger Weise ein Schnitt zugeordnet werden kann (beweise!). Deshalb ist der auf die obigen LP's angewandte Dualitätssatz der linearen Programmierung genau der Max Flow – min Cut Satz 7.9. Zudem haben die uv's einer optimalen dualen Lösung u, w die Bedeutung von Potentialen. Die Komplementärschlupfbedingungen lauten

$$x_e > 0 \Rightarrow \quad u_{h(e)} - u_{t(e)} + w_e = 0 \tag{7.49}$$

$$w_e > 0 \Rightarrow \quad x_e = k_e \tag{7.50}$$

Betrachte einen beliebigen Bogen e:

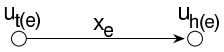


Abbildung 7.8: Bogen e

(i) Falls $u_{t(e)} < u_{h(e)}$, muss $x_e = 0$ gelten wegen (7.49).

(ii) Falls $u_{t(e)} = u_{h(e)}$, kann, aber muss nicht, $x_e > 0$ fliessen.

(iii) Falls $u_{t(e)} > u_{h(e)}$, muss wegen (7.44) $w_e > 0$, und deshalb wegen (7.50) $x_e = k_e$ sein: der Fluss ist auf e auf der oberen Kapazität.

7.6 Zirkulationen

Im Problem des maximalen s–t–Flusses wird die Existenz und Vorgabe eines (zulässigen) s–t–Flusses vorausgesetzt. Um abzuklären, ob ein solcher existiert und um ihn zu bestimmen, gilt es, das Problem des zulässigen s–t–Flusses in einem Netzwerk, oder das verwandte, sog. Problem der zulässigen Zirkulation zu lösen.

Definition 7.29 (Zirkulation) *Gegeben sei ein Netzwerk $G = (V, E)$, $0 \leq \ell_e \leq k_e, e \in E$. Eine* Zirkulation *ist ein Vektor $x \in \mathbb{R}^E$, so dass*

$$\ell_e \leq x_e \leq k_e \qquad e \in E \tag{7.51}$$

$$\sum_{e \in \delta v} x_e - \sum_{e \in \delta(\overline{v})} x_e = 0 \qquad v \in V. \tag{7.52}$$

Beachte, dass für eine Zirkulation keine Knoten s, t ausgezeichnet sind und für alle Knoten die Flusserhaltungsgleichungen (7.52) gelten.

Problem der (zulässigen) Zirkulation:

Gegeben: $G = (V, E), \ell_e, k_e, e \in E (0 \leq \ell_e \leq k_e)$.

Gesucht: Eine Zirkulation $x \in \mathbb{R}^E$.

Bemerkung 7.30 Im Netzwerk $G = (V, E), \ell_e, k_e, e \in E, s, t \in V$, existiert ein s–t–Fluss $x \in \mathbb{R}^E$ nicht–negativer Stärke genau dann, wenn in dem um einen Bogen erweiterten Netzwerk $G' = (V, E \cup (t, s))$ mit $\ell_{ts} = 0, k_{ts} = \infty$, eine Zirkulation existiert. (Die Stärke ist gerade der Fluss auf (t, s).)

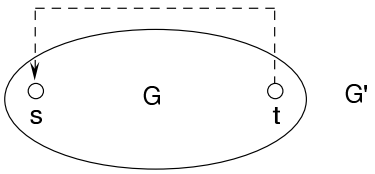


Abbildung 7.9: s–t–Flusses in G

Das Auffinden eines zulässigen s–t–Flusses in G, ist also dem Zirkulationsproblem in G' äquivalent. (Falls ein zulässiger Fluss beliebiger Stärke gesucht wird, ist oben $\ell_{ts} = -\infty$ zu setzen.)

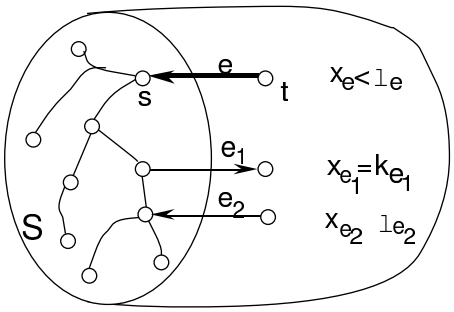
Lemma 7.31 *Falls im Netzwerk $G = (V, E), \ell_e, k_e, e \in E$, eine Zirkulation existiert, gilt $k(S) \geq \ell(\overline{S})$ für alle $S \subseteq V$, $\emptyset \neq S \neq V$.*

Beweis. Sei $S \subseteq V, \emptyset \neq S \neq V$. Durch Addieren der Gleichungen (7.52) für alle $v \in S$ ergibt sich $\sum_{e \in \delta(S)} x_e - \sum_{e \in \delta(\overline{S})} x_e = 0$. (Vgl. Beweis von Lemma 7.5). Zusammen mit (7.51) folgt $k(S) \geq \ell(\overline{S})$. ♣

Verfahren für das Problem der (zulässigen) Zirkulation:

Starte mit $x = 0$. Falls alle $\ell_e = 0$, ist x eine Zirkulation. Sonst gibt es (mindestens) einen Bogen e mit $x_e < \ell_e$. x wird mit Hilfe des Algorithmus für einen maximalen Fluss (flussvergrössernder Wege) solange modifiziert, bis $\ell_e \leq x_e \leq k_e$, wobei die anderen Bedingungen $\ell_d \leq x_d \leq k_d, d \in E, d \neq e$, falls vorher erfüllt, weiterhin erhalten bleiben. Wenn dieses Verfahren nicht gelingt, wird ein $S \subseteq V, \emptyset \neq S \neq V$, ausgewiesen, für welches $k(S) < \ell(\overline{S})$. Gemäss Lemma 7.31 existiert dann keine Zirkulation.

Sei also e mit $t \equiv t(e)$ und $s \equiv h(e)$, und $x_e < \ell_e$. Man entferne gedanklich e und vergrössere sukzessive den s–t–Fluss x. (Der s–t–Fluss muss hier nicht unbedingt alle Bedingungen (7.1) erfüllen!) x (der Stärke x_e) wird mittels flussvergrössernder Wege solange vergrössert, bis die Stärke σ des Flusses $\ell_e \leq \sigma \leq k_e$ erfüllt (dabei wird σ bei der letzten Vergrösserung gegebenenfalls kleiner als in 7.2 gewählt, damit $\sigma \leq k_e$). Der s–t–Fluss der Stärke σ, zusammen mit $x_e = \sigma$, ist eine – eventuell noch unzulässige – Zirkulation, die jedoch eine Bedingung (7.51) mehr erfüllt.



Durch Wiederholen der obigen Prozedur werden alle Bedingungen (7.51) „in Ordnung“ gebracht und man erhält eine Zirkulation, oder aber für ein bestimmtes e mit $x_e < \ell_e$ ist die maximale Stärke σdes s–t–Flusses, wie oben definiert, kleiner als ℓ_e. Sei S die Menge der Knoten, die von s mit einem FW erreichbar sind. Es gilt $s \in S \not\ni t$ (vgl. Beweis Satz 7.8) und

$$\sigma = \sum_{e' \in \delta(S)} x_{e'} - \sum_{\substack{e' \in \delta(\overline{S}) \\ e' \neq e}} x_{e'} \geq \sum_{e' \in \delta(S)} k_{e'} - \sum_{\substack{e' \in \delta(\overline{S}) \\ e' \neq e}} \ell_{e'}. \tag{7.53}$$

Da $e \in \delta(\overline{S})$ und $\sigma < \ell_e$, gilt demnach

$$\ell_e > \sum_{e' \in \delta(S)} k_{e'} - \sum_{e' \in \delta(S)} \ell_{e'} + \ell_e \tag{7.54}$$

und somit $k(S) < \ell(\overline{S})$.

Satz 7.32 (Zirkulationssatz von Hoffman) *Im Netzwerk $G = (V, E)$ mit den Kapazitäten $\ell_e, k_e, e \in E$, existiert eine Zirkulation, falls und nur falls $k(S) \geq \ell(\overline{S})$ für alle $S \subseteq V$, $\emptyset \neq S \neq V$.*

Beweis. Durch obiges Verfahren. ♣

Bemerkung 7.33 Das für die Bestimmung einer Zirkulation erläuterte Verfahren ist eine prinzipielle Methode. Es gibt effizientere Verfahren, die auf einer (und nicht mehrerer wie oben) Flussmaximierung beruhen (vgl. Übungen).

Bei gegebenem Netzwerk $G = (V, E), \ell_e, k_e, e \in E, s, t \in V$, interessiert man sich manchmal für einen s–t–Fluss minimaler Stärke. Dieser kann mit Hilfe flussvermindernder Wege bestimmt werden. Es gilt

Satz 7.34 (Min Flow – Max Cut Satz) *Falls im Netzwerk $G = (V, E)$ mit den Kapazitäten $\ell_e, k_e, e \in E, s, t \in V$, ein s–t–Fluss existiert, ist die minimale Stärke eines s–t–Flusses*

$$\sigma = \max_{s \in S \not\ni t} \{\ell(S) - k(\overline{S})\}. \tag{7.55}$$

7.7 Problemvariationen

(a) Mehrere Quellen und Senken: Einführen zweier neuer Knoten, einer *Superquelle* s_0 bzw. *Supersenke* t_0, die man mit den gegebenen Quellen bzw. Senken verbindet:

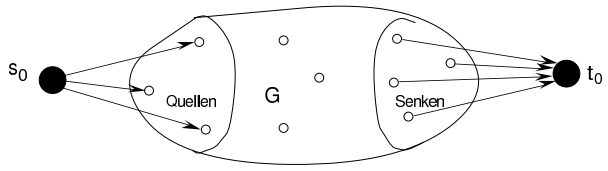


Abbildung 7.10: Superquellen und Supersenken

(b) Untere und obere Schranken auf den Knoten: Man verdoppelt den Knoten und führt Kapazitäten auf dem verbindenden Bogen ein:

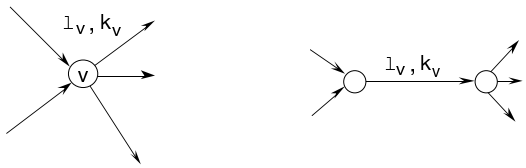


Abbildung 7.11: Ersetzen des Knotens durch einen Bogen

(c) Angebot a_v und Bedarf b_v im Knoten v : Verbinde v mit der Quelle bzw. der Senke: Einführen des Bogens (s_0, v) mit $\ell_{s_0 v} = 0$ und $k_{s_0 v} = a_v$, bzw. (v, t_0) mit $\ell_{v t_0} = b_v$ und $k_{vt} = \infty$.

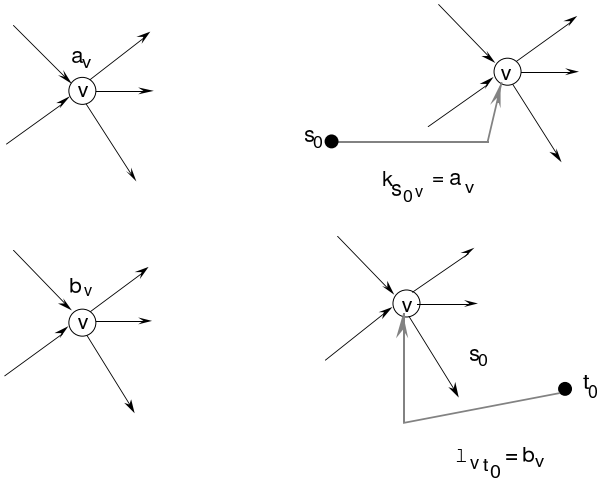


Abbildung 7.12: Angebot und Bedarf

(d) Beliebige (auch negative) Schranken k_e und ℓ_e : Für eine beliebige, reelle Zahl p, sei $[p]_+ \equiv \max\{0, p\}$. Ersetze den Bogen e durch ein Bogenpaar $e, \overline{e}$ (gleiche Endknoten wie e, aber entgegengesetzte Richtung) mit $\ell_e := [\ell_e]_+, k_e := [k_e]_+$ und $\ell_{\overline{e}} := [-k_e]_+, k_{\overline{e}} := [-\ell_e]_+$:

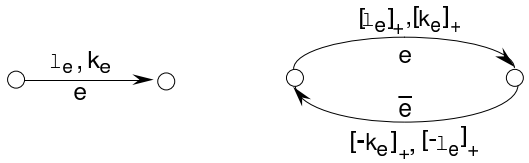


Abbildung 7.13: Beliebige Schranken

Betrachte ein Netzwerk $G = (V, E)$ mit (beliebigen) Schranken auf den Bögen und den Knoten $\ell_e, k_e, e \in E, \ell_v, u_v, v \in V$ (möglicherweise $\pm\infty$), und Angebots– bzw. Bedarfszahlen $d_v, v \in V$ ($d_v \geq 0$: Angebot in $v, d_v < 0$: Bedarf in v). Eine (allgemeiner definierte) Zirkulation in diesem Netzwerk ist ein Vektor $x \in \mathbb{R}^E$, so dass

$$\ell_v \leq \sum_{e \in \delta(v)} x_e \leq k_v, \qquad v \in V,\ \ell_e \leq x_e \leq k_e,\ e \in E \tag{7.56}$$

$$\sum_{e \in \delta(v)} x_e - \sum_{e \in \delta(\overline{v})} x_e = d_v, \qquad v \in V \tag{7.57}$$

Es folgt aus den Transformationen (a) bis (d), dass einer solchen Zirkulation eine Zirkulation x', wie durch (7.51), (7.52) definiert, in einem abgeleiteten Netzwerk zugeordnet ist, und umgekehrt.

Kapitel 8

Kostenminimale Flüsse

8.1 Eine Problemstellung

Im Netzwerk $G = (V, E),\ 0 \leq \ell_e \leq k_e,\ e \in E,\ s, t \in V$, werden reelle Kosten c_e jedem Bogen $e \in E$ zugeordnet. Es stellt sich das Problem des kostenminimalen s–t–Flusses gegebener Stärke:

Problem des kostenminimalen s–t–Flusses gegebener Stärke:

Gegeben: Ein Netzwerk $G = (V, E),\ 0 \leq \ell_e \leq k_e,\ e \in E,\ s, t \in V,\ c_e,\ e \in E,\ \sigma$.

Gesucht: Ein s–t–Fluss der Stärke σ, d. h.. $x \in \mathbb{R}^E$, so dass

$$\ell_e \leq x_e \leq k_e,\ e \in E, \tag{8.1}$$

$$\sum_{e \in \delta(v)} x_e - \sum_{e \in \delta(\overline{v})} x_e = \begin{cases} +\sigma, & v = S \\ 0, & v \in V - \{s, t\} \\ -\sigma, & v = t \end{cases}, \tag{8.2}$$

der die Kosten $\sum\limits_{e \in E} c_e x_e$ minimiert.

Hierbei handelt es sich um ein Problem der linearen Programmierung, mit Variablen $x_e, e \in E$, und Zielfunktion $c^T x \equiv \sum\limits_{e \in E} c_e x_e$. Für seine Lösung wird zunächst ein kombinatorischer Algorithmus erläutert, der auf den Arbeiten von Klein, Busacker und Gowen beruht und mit vergrössernden Wegen und Zyklen arbeitet.

8.2 Kosten als Bogenlängen im Inkrementgraphen

Sei x ein s–t–Fluss im Netzwerk $G = (V, E),\ 0 \leq \ell_e \leq k_e,\ e \in E,\ s, t \in V$, und sei $G^x = (V, E^x)$ der Inkrementgraph mit Kapazitäten $q_e,\ e \in E^x$. Zur Erinnerung sei erwähnt, dass ein flussvergrössernder Weg FW (Zyklus FZ) bezüglich x in G ein Weg W (Zyklus Z) in G^x ist, und dass x mit Hilfe von FW (FZ) vergrössert wird, falls x zu x' geändert wird, wobei

$$x'_e = \begin{cases} x_e + \delta, & \text{falls } e \in W \cap E^x_+ (\in Z \cap E^x_+) \\ x_e - \delta, & \text{falls } \overline{e} \in W \cap E^x_- (\in Z \cap E^x_-) \\ x_e, & \text{sonst} \end{cases} \tag{8.3}$$

und

$$0 < \delta \leq \min\{q_e \mid e \in W\} (\in Z). \tag{8.4}$$

Im Falle eines FW wird offensichtlich die Stärke σ von x um δ vergrössert. Im Falle eines FZ liefert die *Vergrösserung* von x einen Fluss x' *derselben Stärke* wie x. Solche Vergrösserungen, uninteressant im Problem des maximalen Flusses, sind jedoch hier von Interesse, falls x' kleinere Kosten als x aufweist.

Definition 8.1 (Kosten eines flussvergrössernden Weges) *Die* Kosten eines flussvergrössernden Weges *sind definiert durch*

$$c(FW) \equiv \sum\{c_e \mid e \ \textit{Vorwärtsbogen}\ \} - \sum\{c_e \mid e \ \textit{Rückwärtsbogen}\}. \tag{8.5}$$

Die Kosten $c(FZ)$ eines flussvergrössernden Zyklus *FZ sind analog definiert.*

Bemerke, dass, falls x' aus der Vergrösserung von x mit Hilfe von FW erhalten wird,

$$c^T x' = c^T x + c(FW) \cdot \delta \tag{8.6}$$

und analoges gilt bei Ersetzung von FW durch FZ. Definiere im Inkrementgraphen G^x die *Bogenlängen*

$$d_e = c_e, \qquad \text{falls } e \in E^x_+, \tag{8.7}$$

$$d_e = -c_{\overline{e}}, \qquad \text{falls } e \in E^x_-, \tag{8.8}$$

und sei W der Weg in G^x, der dem flussvergrössernden Weg FW entspricht. Offensichtlich ist

$$d(W) \equiv \sum_{e \in W} d_e = c(FW), \tag{8.9}$$

d. h. die Kosten eines flussvergrössernden Weges FW sind genau die Länge des entsprechenden Weges in G^x bei Bewertung d. Analoges gilt für einen flussvergrössernden Zyklus FZ. Folgende Fragestellungen sind deshalb äquivalent:

in G	in G^x
Bestimme einen kostenminimalen flussvergrössernden Weg FW bezüglich x.	Bestimme einen kürzesten Weg W von s nach t.
Bestimme einen flussvergrössernden Zyklus FZ bezüglich x negativer Kosten.	Bestimme einen Zyklus Z negativer Länge.

Tabelle 8.1: Fragestellungen in G und G^x

Beide Aufgaben in G^x lassen sich mit einem Algorithmus für kürzeste Wege (bei beliebigen Bogenlängen), z.B. mit dem Floyd–Warshall–Algorithmus, lösen.

8.3 Hauptsätze

Satz 8.2 *Ein s–t–Fluss x der Stärke σ ist ein kostenminimaler s–t–Fluss der Stärke σ, genau dann wenn kein flussvergrössernder Zyklus bezüglich x negativer Kosten existiert.*

Auf Grund dieses Satzes bietet sich folgendes *Verfahren für das Problem des kostenminimalen s–t–Flusses gegebener Stärke σ an:*

Algorithmus 8.1 Kostenminimaler s–t–Fluss gegebener Stärke σ

Phase 1: Finde einen s–t–Fluss x der Stärke σ, falls ein solcher existiert.

Phase 2: Vergrössere sukzessive x mit Hilfe flussvergrössernder Zyklen negativer Kosten, solange dies möglich ist.

Für den Beweis von Satz 8.2 benötigen wir eine Dekompositionseigenschaft von Zirkulationen.

Lemma 8.3 *In einem Graphen $G = (V, E)$ sei $x \in \mathbb{R}^E$, so dass*

$$\begin{aligned} 0 \leq x_e, \qquad & e \in E, \\ \sum_{e \in \delta(v)} x_e - \sum_{e \in \delta(\overline{v})} x_e = 0, \qquad & v \in V \end{aligned} \tag{8.10}$$

Dann lässt sich x immer darstellen als $x = \sum_{i \in I} \alpha_i z^i$, wobei $\alpha_i \geq 0, i \in I$, und $z^i, i \in I$, Inzidenzvektoren von Zyklen von G sind.

Beweis. Falls $x = 0$, nehme $\alpha_i = 0, i \in I$. Sei n ganz positiv, nehme an, das Lemma sei richtig für alle x', die (8.10) erfüllen und weniger als n Nichtnull–Komponenten $x'e, e \in E$, haben, und sei x, das (8.10) genügt und n Nichtnull–Komponenten zählt. Streiche aus G alle Bögen $e \in E$ mit $x_e = 0$, und erhalte den Graphen G'. Es existiert ein Zyklus Z_0 in G', ansonsten gäbe es einen Knoten v in G', in welchen kein Bogen hineingeht, d.h. für welchen die Bilanzgleichung (8.10) verletzt wäre. Sei $\alpha_0 = \min\{x_e \mid e \in Z_0\} (> 0)$. $x' \in \mathbb{R}^E$, definiert durch $x'_e = x_e - \alpha_0$ für $e \in Z_0, x'_e = x_e$ sonst, d.h. $x' = x' - \alpha_0 z^0$, erfüllt (8.10) und zählt weniger als n Nichtnull–Komponenten. Deshalb ist x' darstellbar als $x' = \sum_{i \in I'} \alpha_i z^i$, und $x = \sum_{i \in I} \alpha_i z^i$ mit $I = I' \cup \{0\}$. ♣

Beispiel 8.4 Illustration zu Lemma 8.3

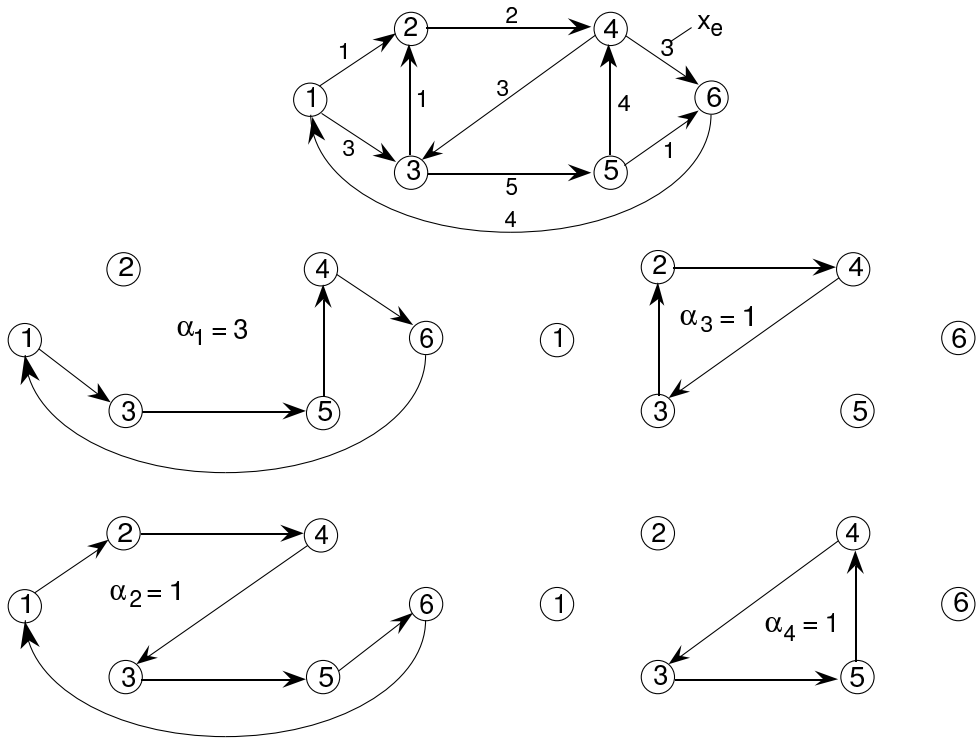


Abbildung 8.1: Aufteilen in verschiedene Zyklen

◇

Beweis von Satz 8.2.

(i) (notwendig): Falls ein FZ negativer Kosten existiert, kann x mit Hilfe dieses FZ vergrössert und die Kostenverminderung $c(FZ) \cdot \delta$ bei Beibehaltung der Stärke σ erzielt werden.

(ii) (hinreichend): Man nehme an, x und x' seien zwei s–t–Flüsse der Stärke σ mit $c^T x' < c^T x$ und zeige, dass bezüglich x ein FZ negativer Kosten existiert. Sei $y = x' - x$. Es gilt $y \neq 0$ und $c^T y < 0$, und, da x, x' s–t–Flüsse der gleichen Stärke sind,

$$\sum_{e \in \delta(v)} y_e - \sum_{e \in \delta(\overline{v})} y_e = 0 \text{ für alle } v \in V. \tag{8.11}$$

Gewisse y_e mögen jedoch negativ sein, deshalb wird der Graph $G' = (V, E')$ wie folgt definiert, und dessen Bögen bewertet:

$$e \in E' \text{ und } d_e = \ c_e, \text{ falls } e \in E \text{ und } y_e > 0, \tag{8.12}$$
$$e \in E' \text{ und } d_e = -c_{\overline{e}}, \text{ falls } \overline{e} \in E \text{ und } y_{\overline{e}} > 0. \tag{8.13}$$

$y' \in \mathbb{R}^{E'}$, definiert durch

$$y'_e = y_e, \text{ falls } e \in E \text{ und } y_e > 0, \tag{8.14}$$
$$y'_e = y_{\overline{e}}, \text{ falls } \overline{e} \in E \text{ und } y_{\overline{e}} > 0, \tag{8.15}$$

ist eine Zirkulation in $G' = (V, E')$, $\ell'_e = 0$, $k'_e = \infty$, $e \in E'$, und $dy' = cy < 0$, $y' \neq 0$. Gemäss Lemma 8.3 existieren Zyklen $Z'_i, i \in I$, so dass

$$y' = \sum_{i \in I} \alpha_i z'^i, \ \alpha_i > 0, \ i \in I, \tag{8.16}$$

wobei z'_i die Inzidenzvektoren von $Z'_i, i \in I$, sind. Da $dy' < 0$, existiert ein Zyklus Z'_{i_0} mit $dz'^{i_0} < 0$.

Nun entspricht Z'_{i_0} einem Kreis Z_{i_0} in G. Orientiere ihn übereinstimmend mit der Richtung von Z'_{i_0}. Es gilt:

$$y_e = x'_e - x_e > 0, \text{ falls } e \text{ Vorwärtsbogen von } Z_{i_0}, \tag{8.17}$$
$$y_e = x'_e - x_e < 0, \text{ falls } e \text{ Rückwärtsbogen von } Z_{i_0}, \tag{8.18}$$

d. h. $x_e < x'_e \leq k_e$, falls e Vorwärtbogen und $\ell_e \leq x'_e < x_e$, falls e Rückwärtsbogen: Z_{i_0} ist ein flussvergrössernder Zyklus FZ_{i_0} bezüglich x. Zudem ist $c(FZ_{i_0}) = dz'^{i_0} < 0$.

♣

Satz 8.5 *Sei x ein kostenminimaler s–t–Fluss der Stärke σ und FW ein kostenminimaler flussvergrössernder Weg bezüglich x der Kapazität δ_0. Dann liefert die Vergrösserung von x um δ, $0 \leq \delta \leq \delta_0$, mit Hilfe von FW einen kostenminimalen s–t–Fluss x' der Stärke $\sigma + \delta$.*

Beweis. Seien x, x' und FW wie im Satz definiert und sei x' nicht kostenminimal. Dann existiert wegen Satz 8.2 ein flussvergrössernder Zyklus FZ

bezüglich x' negativer Kosten. Sei W der zu FW entsprechende Weg in G^x und Z der zu FZ entsprechende Zyklus in $G^{x'}$ und betrachte den Graphen $G' = (V, E')$ mit Bögen $E' := W \cup Z \setminus \{(i,j), (j,i) \mid (i,j) \in W \text{ und } (j,i) \in Z\}$, wobei ein Bogen, der in W und Z in gleicher Richtung vorkommt, als Parallelbogen in G' auftritt. Es gilt $d^+_{E'}(v) = d^-_{E'}(v)$ für $v \in V \setminus \{s,t\}, d^+_{E'}(s) = d^-_{E'}(s) + 1$ und $d^+_{E'}(t) = d^-_{E'}(t) - 1$. Deshalb lassen sich die Bögen von E' in einen s–t–Weg W_0 und Zyklen $Z_i, i = 1, \ldots, k$ zerlegen. Man bemerke, dass alle Bögen von E' Bögen von G^x sind, somit entspricht $W_0(Z_i)$ einem flussvergrössernden Weg FW_0 (Zyklus FZ_i) bezüglich x. Da x kostenminimal ist, gilt $d(Z_i) \geq 0, i = 1, \ldots, k$ (d_e sind die Bogenlängen in G^x) und

$$\begin{aligned} c(FW) &> c(FW) + c(FZ) && (8.19)\\ &= d(E') && (8.20)\\ &= d(W_0) + \sum\{d(Z_i) \mid i = 1, \ldots, k\} && (8.21)\\ &\geq d(W_0) && (8.22)\\ &= c(FW_0), && (8.23) \end{aligned}$$

ein Widerspruch zu FW kostenminimaler flussvergrössernder Weg. ♣

Satz 8.5 suggeriert ein zweites Verfahren für die Bestimmung eines kostenminimalen s–t–Flusses gegebener Stärke σ, nämlich: gehe von einem kostenminimalen s–t–Fluss der Stärke $\sigma' < \sigma$ aus, und vergrössere den Fluss mit Hilfe kostenminimaler flussvergrössernder Wege, bis die Stärke gleich σ ist.

Folgender Algorithmus kombiniert beide Verfahren. Dabei ist ein Netzwerk $G = (V, E)$ mit Schranken $0 \leq \ell_e \leq k_e$, $e \in E$ und Kosten c_e, $e \in E$ gegeben. Weiter sind $s, t \in V$, eine Stärke σ und ein zulässiger Initialfluss der Stärke $\sigma' \leq \sigma$ gegeben. Der Algorithmus 8.2 findet einen kostenminimalen s–t–Fluss der Stärke σ.

Algorithmus 8.2 Kostenminimaler s–t–Fluss

begin (* kostenminimaler Fluss *)
Sei x ein zulässiger Fluss der Stärke $\sigma' \leq \sigma$
(* Flussvergrössernde Zyklen negativer Kosten *)
repeat
suche in G^x mit Bewertung δ einen Zyklus Z negativer Länge
if Z gefunden **then**
vergrössere x mit Hilfe des Z entsprechenden flussvergrössernden Zyklus mit
$\delta := \min\{q_e \mid e \in Z\}$
end (* if *)
until Z nicht gefunden
(* Flussvergrössernde Wege minimaler Kosten *)
if $\sigma' < \sigma$ **then**
repeat
suche in G^x mit Bewertung d einen kürzesten s–t–Weg W
if W gefunden **then**
vergrössere x mit Hilfe dem W entsprechenden flussvergrössernden
Weg mit:
$\delta := \min\{\sigma - \sigma', \min\{q_e \mid e \in W\}\}$
$\sigma' := \sigma' + \delta$
end (* if *)
until (W nicht gefunden) or ($\sigma = \sigma'$)
end (* if *)
if $\sigma = \sigma'$ **then**
x ist kostenminimaler Fluss der Stärke σ
else
x ist kostenmin. Fluss der Stärke σ' und es existiert kein Fluss der Stärke σ
end (* else *)
end (* kostenminimaler Fluss *)

8.4 Verwandte Probleme

Problem des kostenminimalen s–t–Flusses beliebiger Stärke

Falls die Stärke des s–t–Flusses nicht vorgegeben ist, ändert sich die Problemstellung von 8.1 entsprechend:

Problem des kostenminimalen s–t–Flusses beliebiger Stärke:

Gegeben: Ein Netzwerk $G = (V, E),\ 0 \leq \ell_e \leq k_e,\ e \in E,\ s, t \in V,\ c_e,\ e \in E$.

Gesucht: Ein s–t–Fluss $x \in \mathbb{R}^E$ und σ, so dass

$$\ell_e \leq xe \leq k_e, \quad e \in E, \tag{8.24}$$

$$\sum_{e \in \delta(v)} x_e - \sum_{e \in \delta(\overline{v})} x_e = \begin{cases} +\sigma, & v = s \\ 0, & v \in V - \{s, t\} \\ -\sigma, & v = t \end{cases}, \tag{8.25}$$

und die $\sum_{e \in E} c_e x_e$ minimieren.

Seien $f(\sigma)$ die Kosten eines kostenminimalen s–t–Flusses der Stärke σ (falls ein solcher existiert). Es ist leicht zu zeigen (siehe Übungen), dass $f(\sigma)$ eine *konvexe* Funktion von σ ist.

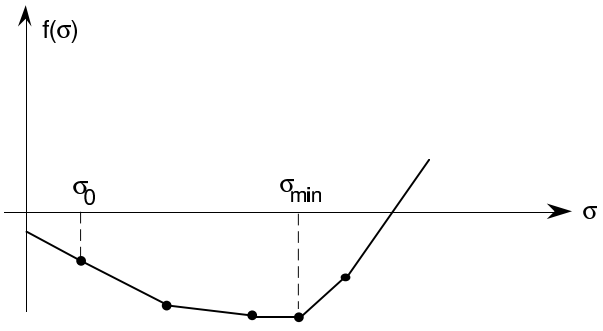


Abbildung 8.2: Kosten $f(\sigma)$ eines kostenminimalen s–t–Flusses der Stärke σ

Um obiges Problem zu lösen, genügt es deshalb, von einem kostenminimalen Fluss der Stärke $\sigma_0 \leq \sigma_{\min}$ auszugehen und mit flussvergrössernden Wegen minimaler Kosten den Fluss solange zu vergrössern, bis seine Kosten wieder ansteigen (d.h. bis die minimalen Kosten eines flussvergrössernden Weges positiv sind).

Beliebige Schranken k_e und ℓ_e

Das Problem des kostenminimalen s–t–Flusses gegebener oder beliebiger Stärke bei beliebigen, auch negativen Schranken k_e und ℓ_e, kann durch Einführen von Bogenpaaren (vgl. Kap. 7) auf den Fall nicht–negativer Schranken zurückgeführt werden.

Transshipment–Problem ohne [mit] Kapazitäten

Ein klassisches Netzwerkflussproblem ist das sogenannte

Transshipment–Problem ohne [mit] Kapazitäten:

Gegeben: Ein Netzwerk $G = (V, E)$ mit $[0 \leq k_e,\ e \in E]$, $d_v,\ v \in V$, $c_e,\ e \in E$.

Gesucht: Ein Fluss $x \in \mathbb{R}_+^E$, so dass

$$0 \leq x_e [\leq k_e], \qquad e \in E \tag{8.26}$$

$$\sum_{e \in \delta(v)} x_e - \sum_{e \in \delta(\overline{v})} x_e \leq d_v, \qquad v \in V \tag{8.27}$$

der

$$\sum_{e \in E} c_e x_e \ \text{ minimiert.} \tag{8.28}$$

Es kann als *Verteilproblem* eines bestimmten Gutes gedeutet werden (daher der Name): In jedem Knoten v mit $d_v > 0$ sind d_v Einheiten des Gutes verfügbar: d_v ist das Angebot im Produktionsknoten v (Fabrik, Zentrallager, usw.). Das Gut soll über die von den Bögen dargestellten Verbindungen zu

den Knoten v mit $d_v < 0$ transportiert werden, wobei der Bedarf $-d_v$ jeder dieser Verbrauchsknoten (Kunde, Aussenlager, usw.) zu decken ist. Für jedes $e \in E$ stellen x_e die Anzahl Einheiten des über den Bogen e transportierten Gutes und c_e die Transportkosten pro Einheit des Gutes dar. Die Gesamttransportkosten (8.28) sind zu minimieren.

Beispiel 8.6 Die Zahlen neben den Knoten sind die d_v. Der Übersicht halber sind keine Kosten c_e oder Kapazitäten $k_e, e \in E$, eingezeichnet.

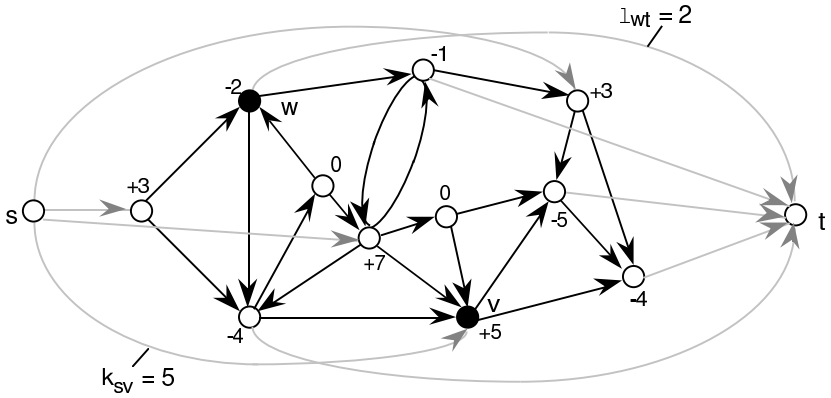


Abbildung 8.3: Transshipment–Problem

◇

Das Transshipment–Problem lässt sich leicht auf das Problem des kostenminimalen s–t–Flusses beliebiger Stärke zurückführen. Führe eine Quelle s und eine Senke t ein, sowie die Bögen (s, v) mit oberer Schranke $k_{(s,v)} = d_v$ und Kosten $c_{(s,v)} = 0$ für jeden Produktionsknoten v, und die Bögen (v, t) mit unterer Schranke $\ell_{(v,t)} = -d_v$ und Kosten $c_{(v,t)} = 0$ für jeden Verbrauchsknoten v (siehe Abbildung 8.3).

Bemerkung 8.7

(a) Eine notwendige Bedingung für die Existenz eines zulässigen Flusses ist:

$$\sum_{v:v_v<0} -d_v \leq \sum_{v:v_v>0} d_v, \tag{8.29}$$

d. h. Gesamtangebot muss den Gesamtbedarf übersteigen, denn das Addieren aller Ungleichungen (8.27) liefert $0 \leq \sum_{v \in V} d_v$, d. h. (8.29).

(b) *Die Standard–Form des Transshipment–Problems* erhält man, indem man in (8.27) $\leq$ durch $=$ ersetzt. Falls kein gerichteter Weg W mit Kosten $\sum\{c_e \mid e \in W\} < 0$ von einem Produktionsknoten zu einem Verbrauchsknoten führt (dies ist sicher der Fall, wenn alle $c_e \geq 0$), lässt sich aufgrund folgender Überlegungen ein Transshipment–Problem mit (8.27) in Ungleichungsform immer in die Standardform überführen:

(i) Es existiert immer ein optimaler Fluss x, der (8.27) für die Verbrauchsknoten mit Gleichheit erfüllt, d. h. die Bedarfe genau deckt.

(ii) Durch Hinzufügen eines fiktiven Verbrauchsknotens v_0 mit $d_{v_0)} = -$ (Gesamtangebot $-$ Gesamtbedarf) und der Bögen (v, v_0) mit $k_{(v,v_0)} = +\infty$ und $c_{(v,v_0} = 0$, für alle Produktionsknoten v müssen dann alle Ungleichungen (8.27) für jeden zulässigen Fluss mit Gleichheit erfüllt werden.

Transportproblem ohne [mit] Kapazitäten

Das *Transportproblem* ohne [mit] Kapazitäten ist ein Spezialfall des Transshipment–Problems, bei welchem G ein bipartiter Graph $(S \cup T, E)$ ist, die Knoten von S alle Produktionsknoten, diejenigen von T alle Verbrauchsknoten, alle Bögen von S nach T gerichtet und die Kosten $c_e, e \in E$, nicht–negativ sind. Üblicherweise wird es wie folgt formuliert:

Transportproblem ohne [mit] Kapazitäten

Gegeben: Angebote a_i, $i = 1, \ldots, m$, Bedarfe b_j, $j = 1, \ldots, n$, und für alle $i = 1, \ldots, m,\ j = 1, \ldots, n$, Transportkosten pro Mengeneinheit $c_{ij} \geq 0$ [und Kapazitäten $k_{ij} \geq 0$].

Gesucht: Ein Transportplan $x_{ij},\ i = 1, \ldots, m,\ j = 1, \ldots, n$, so dass

$$0 \leq x_{ij} [\leq k_{ij}], \qquad i = 1, \ldots, m,\ j = 1, \ldots, n, \tag{8.26'}$$

$$\begin{aligned} \sum_{j=1}^{n} x_{ij} &\leq a_i, \qquad i = 1, \ldots, m \\ \sum_{i=1}^{m} x_{ij} &\geq b_i, \qquad i = 1, \ldots, m \end{aligned} \tag{8.27'}$$

und

$$\sum_{i,j} c_{ij} x_{ij} \quad \text{minimiert.} \tag{8.28'}$$

(Falls die Verbindung von i nach j nicht zugelassen ist, wird $c_{ij} = +\infty$ gesetzt.)

Beispiel 8.8 Beispiel eines Transportproblems

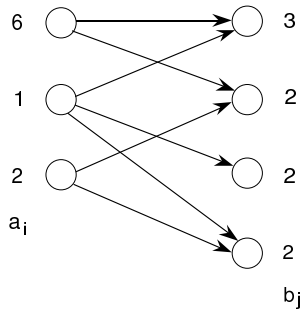


Abbildung 8.4: Beispiel eines Transportproblems

◇

Falls $\sum a_i \geq \sum b_j$, kann das Problem immer in Standardform formuliert werden, bei welcher $\sum a_i = \sum b_j$ und alle Ungleichungen 8.27' Gleichungen sind (siehe Bemerkungen a) und b), Abschnitt „Transshipment–Problem“). Das Zuordnungproblem (vgl. Kapitel 5) ist ein Transportproblem ohne Kapazitäten in Standardform, mit $a_i = b_j = 1$ für alle $i \in S, j \in T$.

Problem der kostenminimalen Zirkulation

Ein weiteres klassisches Problem ist das

Problem der kostenminimalen Zirkulation:

Gegeben: Ein Netzwerk $G = (V, E)$, $0 \leq \ell_e \leq k_e$, $e \in E$, c_e, $e \in E$.

Gesucht: Eine Zirkulation, d. h. $x \in \mathbb{R}_+^E$, so dass

$$\ell_e \leq x_e \leq k_e, \qquad e \in E \tag{8.30}$$

$$\sum_{e \in \delta(v)} x_e - \sum_{e \in \delta(\overline{v})} x_e = 0, \qquad v \in V. \tag{8.31}$$

die

$$\sum_{e \in E} c_e x_e \text{ minimiert.} \tag{8.32}$$

Alle bisher gesehenen Flussprobleme von Kapitel 7 und 8 lassen sich als Problem der kostenminimalen Zirkulation formulieren.

Kapitel 9

Schnittebenenverfahren

Häufig lassen sich diskrete Optimierungsprobleme in der Form schreiben

$$\begin{aligned} \max \quad & c^T x && (9.1)\\ s.d. \quad & Ax \leq b && (9.2)\\ & 0 \leq x \leq d && (9.3)\\ & x \text{ ganzzahlig} && (9.4) \end{aligned}$$

wobei A, b, c und d ganzzahlig sind.

In den bis anhin betrachteten Beispielen (z. B. Gerüst-, Arboreszenz-, oder Matchingproblem) konnte jeweils die Ganzzahligkeitsbedingung (9.4) weggelassen werden, da das LP (9.1) – (9.3) nur ganzzahlige Basislösungen besass. In diesen Fällen konnte das Problem (9.1) – (9.4) mit einem LP–Verfahren gelöst werden, solange dieses immer Eckpunkte liefert. (Für die oben erwähnten Probleme haben wir aber jeweils effizientere Algorithmen betrachtet.) Im Allgemeinen kann jedoch (9.4) nicht fallengelassen werden. Deshalb betrachten wir nun

$$\mathbf{S} := \{x \in Z^n \mid x \text{ erfüllt (9.2) und(9.3)}\}$$

Sei $P := CONV(S)$, dann ist P ein Polyeder (siehe Kapitel 1) und es existieren eine Matrix A' und ein Vektor b', so dass sich P schreiben lässt als

$$P = \{x \in \mathbb{R}^n \mid A'x \leq b', x \geq 0\}.$$

Somit ist das Problem (9.1) – (9.4) äquivalent zum LP

$$\max c^T x \tag{9.5}$$
$$s.d.\ A'x \leq b' \tag{9.6}$$
$$x \leq 0 \tag{9.7}$$

(im Sinne dass das LP (9.5) – (9.7) immer eine optimale Basislösung, also eine optimale Lösung aus $\mathbf{S}$ besitzt).

Wie wir bereits im Kapitel 1 (LOCO–Problem und Lineare Programmierung) erwähnt haben, ist die Beschreibung $A'x \leq b'$, $x \geq 0$ oft schwer zu finden und es können exponentiell viele Restriktionen für diese Beschreibung notwendig sein.

In diesem Kapitel werden wir ein Verfahren betrachten, das von der Problemstellung (9.1) – (9.4) aus startet. Falls das jeweils zugehörige LP keine ganzzahlige Optimallösung besitzt, werden sukzessive zusätzliche Restriktionen zum LP hinzugefügt. Dieses Verfahren wird fortgesetzt, bis das erweiterte LP eine ganzzahlige Optimallösung besitzt.

9.1 Schnittebenenverfahren und Gomory–Schnitt

Um das Problem (9.1) – (9.4) zu lösen, versuchen wir zuerst einmal, die Ganzzahligkeitsrestriktion fallen zu lassen, d. h. wir lösen das LP

$$\max c^T x, \text{ so dass } x \in P^0 := \{x \in \mathbb{R}^n \mid x \text{ erfüllt (9.2) und (9.3) }\}. \tag{9.8}$$

Besitzt dieses LP eine ganzzahlige Optimallösung x^*, so ist x^* sicher auch eine Optimallösung von unserem ursprünglichen Problem. Im Allgemeinen wird x^* aber nicht ganzzahlig sein. Man bemerke, dass man in diesem Fall x^* nicht einfach runden kann, um eine Lösung von (9.1) – (9.4) zu erhalten (siehe Abbildung 9.1).

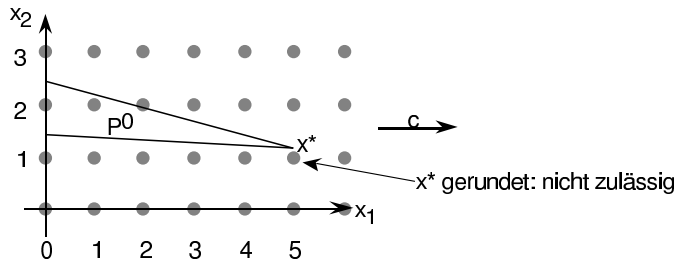


Abbildung 9.1: Einfaches Runden ist nicht unbedingt zulässig

Falls x^* nicht ganzzahlig ist, versuchen wir, eine Restriktion $a^T x \leq a_0$ zum LP hinzuzufügen, die x^* abschneidet, also

$$x^* \notin P^1 := P^0 \cap \{x \in \mathbb{R}^n \mid a^T x \leq a_0\}. \tag{9.9}$$

Durch diese Restriktion soll aber kein ganzzahliger Vektor von P^0 abgeschnitten werden: Es soll weiterhin $P \subseteq P^1$ gelten. Als nächstes lösen wir das LP

$$\max c^T x, \text{ so dass } x \in P^1. \tag{9.10}$$

Entweder besitzt dieses LP eine ganzzahlige Optimallösung, die wegen $P \subseteq P^1$ eine Optimallösung des ursprünglichen Problems ist, oder es wird erneut eine Restriktion hinzugefügt. Dieses Verfahren wird so lange fortgesetzt, bis das LP eine ganzzahlige Optimallösung besitzt. Dieses Verfahren ist in Abbildung 9.2 dargestellt.

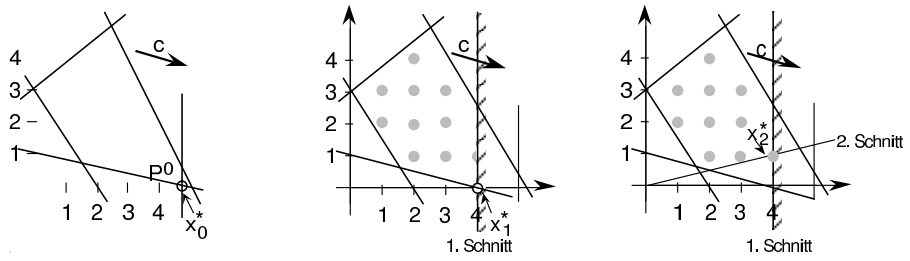


Abbildung 9.2: Schnittebenenverfahren

Definition 9.1 (Schnitt, Schnittebene) *Sei* $P^0 := \{x \in \mathbb{R}^n \mid Ax \leq b\}$ *ein Polyeder und* $S := \{x \in \mathbf{Z} \mid x \in P^0\}$. *Eine Restriktion* $a^T x \leq a_0$

heisst Schnittebene *oder* Schnitt*, falls sie keinen ganzzahligen Vektor von* P^0 *abschneidet, also* $S \subseteq P^0 \cap \{x \in \mathbb{R}^n \mid a^T x \leq a_0\}$.

Das obige Verfahren wird Schnittebenenverfahren genannt, da in jeder Iteration eine neue Schnittebene zum LP hinzugeführt wird. Ausgehend vom ILP-Problem max cx, so dass $x \in \mathbf{S} := \{x \in \mathbf{Z}^n \mid Ax \leq b, 0 \leq x \leq d\}$, lautet es zusammengefasst:

Algorithmus 9.1 Schnittebenenverfahren

begin (* Schnittebenenverfahren *)
  Sei $P := CONV(S)$ und $P^0 := \{x \in \mathbb{R}^n \mid Ax \leq b, 0 \leq x \leq d\}$
  $t := 0$; keineLösung := false;
  repeat
    löse $(LP)_t$: max cx, so dass $x \in P^t$, erhalte Lösung x^*, falls eine existiert
    if $(LP)_t$ besitzt Lösung **then**
      if x^* nicht ganzzahlig **then**
        finde Schnitt $(a^t)^T x \leq a_0^t$ mit $(a^t)^T x^* > a_0^t$
        $P^{t+1} := P^t \cap \{x \in \mathbb{R}^n \mid (a^t)^T x \leq a_0^t\}$ (* $P \subseteq P^{t+1} \subset P^t$ *)
        $t := t + 1$
      end (* if *)
    else keineLösung := true
  until (x^*ganzzahlig) or (keineLösung = true)
  if keineLösung = true **then**
    ILP besitzt keine Lösung
  else
    x^* ist optimale Lösung von ILP
end (* Schnittebenenverfahren *)

Um diesen Algorithmus anwenden zu können, müssen wir solche Schnitte finden können. Wir beschreiben nun eine algebraische Methode, um Schnitte zu finden. Dazu gehen wir davon aus, dass das Problem (9.1) – (9.4) in folgender Form geschrieben ist:

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

dass A vollen Zeilenrang besitzt und dass das zugehörige LP

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

eine optimale Lösung besitzt. Wir bezeichnen wiederum mit

$$P^0 := \{x \in \mathbb{R}^n \mid Ax = b, x \geq 0\} \text{ und mit} \tag{9.11}$$
$$\mathbf{S} := \{x \in \mathbf{Z}^n \mid x \in P^0\}. \tag{9.12}$$

Sei nun x^* eine optimale Basislösung von (LP) und seien x_i, $i \in B$ die Basisvariablen; x_j, $j \in N$ die Nichtbasisvariablen. Dann lässt sich $Ax = b$ auch schreiben als

$$x_i + \sum_{j \in N} \overline{a}_{ij} x_j = \overline{b}_i, \qquad \forall i \in B. \tag{9.13}$$

Bemerkung 9.2 Man bemerke, dass sich diese Form beim Simplexalgorithmus direkt aus dem Endtableau herauslesen lässt.

Wir wollen nun eine Schnittebene konstruieren. Sei dazu $i \in B$ fest. Es gilt natürlich

$$x_i + \sum_{j \in N} \overline{a}_{ij} x_j = \overline{b}_i, \qquad \forall x \in \mathbf{S} \tag{9.14}$$

Da $\mathbf{S} \subseteq \mathbb{R}^n_+$, gilt

$$x_i + \sum_{j \in N} \lfloor \overline{a}_{ij} \rfloor x_j \leq \overline{b}_i, \qquad \forall x \in \mathbf{S} \tag{9.15}$$

wobei $\lfloor a \rfloor$ die grösste ganze Zahl kleiner gleich a bedeutet. Da weiter $S \subseteq \mathbf{Z}^n$, gilt auch

$$x_i + \sum_{j \in N} \lfloor \overline{a}_{ij} \rfloor x_j \leq \lfloor \overline{b}_i \rfloor, \qquad \forall x \in \mathbf{S}. \tag{9.16}$$

Zählt man von der Gleichung (9.14) die Ungleichung (9.16) ab, erhält man

$$\sum_{j\in N}(\overline{a}_{ij} - \lfloor \overline{a}_{ij} \rfloor)x_j \geq \overline{b}_i - \lfloor \overline{b}_i \rfloor, \quad \forall x \in \mathbf{S}. \tag{9.17}$$

Die Restriktion (9.17) ist also eine Schnittebene. Weiter gilt $x_j^* = 0$ für alle $j \in N$. Deshalb schneidet die Restriktion (9.17) x^* ab, falls $x_i^* = \overline{b}_i$ nicht ganzzahlig ist. In diesem Fall ist nämlich

$$\sum_{j\in N}(\overline{a}_{ij} - \lfloor \overline{a}_{ij} \rfloor)x_j^* = 0 < \overline{b}_i - \lfloor \overline{b}_i \rfloor. \tag{9.18}$$

Für das Schnittebenenverfahren kann deshalb, falls die Basislösung x^* nicht ganzzahlig ist, ein möglicher Schnitt wie folgt gewählt werden:

Definition 9.3 (Gomory–Cut) *Wähle einen Index i der Basis, für den x_i^* nicht ganzzahlig ist und wähle als Schnitt die Restriktion (9.17) für diesen Index. Eine Schnittebene der Form (9.17) wird* Gomory–Cut *genannt.*

Beispiel 9.4 Betrachte das ILP (vgl. Abbildung 9.3):

$$\begin{array}{rr} \max & x_2 \\ \text{so dass} & 3x_1 + 2x_2 \leq 6 \\ & -3x_1 + 2x_2 \leq 0 \\ & x_1 \text{ ganz} \\ & x_2 \text{ ganz} \end{array} \tag{ILP}$$

Das dazugehörige relaxierte LP schreiben wir in der Form

$$\begin{array}{rr} \max & z \\ \text{so dass} & z - x_2 = 0 \\ & 3x_1 + 2x_2 + y_1 = 6 \\ & -3x_1 + 2x_2 + y_2 = 0 \\ & z, x_1, x_2, y_1, y_2 \geq 0 \end{array} \tag{LP1}$$

Löst man (LP1) mit dem Simplex-Algorithmus, erhält man als Optimallösung

$$(z, x_1, x_2, y_1, y_2)_1^* = (\frac{3}{2}, 1, \frac{3}{2}, 0, 0) \tag{9.19}$$

und aus dem Endtableau liest man

$$z + \frac{1}{4}\, y_1 + \frac{1}{4}\, y_2 = \frac{3}{2} \tag{9.20}$$

$$x_1 + \frac{1}{6}\, y_1 - \frac{1}{6}\, y_2 = 1 \tag{9.21}$$

$$x_2 + \frac{1}{4}\, y_1 + \frac{1}{4}\, y_2 = \frac{3}{2}. \tag{9.22}$$

$(z, x_1, x_2, y_1, y_2)_1^*$ ist nicht ganzzahlig, also muss ein Schnitt generiert werden. Der zur Basisvariablen z gehörige Gomory–Cut lautet:

$$\frac{1}{4}y_1 + \frac{1}{4}y_2 \geq \frac{1}{2} \text{ oder in den Variablen } x_1, x_2 : x_2 \leq 1. \tag{9.23}$$

Durch Hinzufügen dieser Restriktion zu (LP1) erhält man

$$\begin{array}{rr} \max & z \\ \text{so dass} & z - x_2 = 0 \\ & 3x_1 + 2x_2 + y_1 = 6 \\ & -3x_1 + 2x_2 + y_2 = 0 \\ & x_2 + y_3 = 1 \\ & z, x_1, x_2 \geq 0 \\ & y_1, y_2, y_3 \geq 0 \end{array} \tag{LP2}$$

Eine Optimallösung von (LP2) lautet $(z, x_1, x_2, y_1, y_2, y_3)_2^* = (1, \frac{2}{3}, 1, 2, 0, 0)$ und aus dem Endtableau liest man

$$\begin{array}{cccccccc} z & & & & + & & y_3 & = & 1 \\ x_1 & - & \frac{1}{3} & y_2 & + & \frac{2}{3} & y_3 & = & \frac{2}{3} \\ x_2 & & & & & + & y_3 & = & 1 \\ y_1 & + & & y_2 & - & 4 & y_3 & = & 2 \end{array}$$

Da $(z, x_1, x_2, y_1, y_2, y_3)_2^*$ nicht ganzzahlig ist, wird wieder ein Schnitt generiert, diesmal der zur Basisvariablen x_1 gehörige Gomory–Cut:

$$\frac{2}{3}y_2 + \frac{2}{3}y_3 \geq \frac{2}{3} \text{ oder } x_1 - x_2 \geq 0. \tag{9.24}$$

Durch Hinzufügen dieser Restriktion zu (LP2) erhält man:

$$\begin{array}{rr} \max & z \\ \text{so dass} & z - x_2 = 0 \\ & 3x_1 + 2x_2 + y_1 = 6 \\ & -3x_1 + 2x_2 + y_2 = 0 \\ & x_2 + y_3 = 1 \\ & x_1 - x_2 - y_4 = 0 \\ & z, x_1, x_2, y_1, y_2, y_3, y_4 \geq 0 \end{array} \tag{LP3}$$

Die Optimallösung von (LP3) lautet:

$$(z, x_1, x_2, y_1, y_2, y_3, y_4)_3^* = (1, 1, 1, 1, 1, 0, 0). \tag{9.25}$$

Sie ist ganzzahlig und $(x_1, x_2)^* := (x_1, x_2)_3^* = (1, 1)$ ist Optimallösung von (ILP).

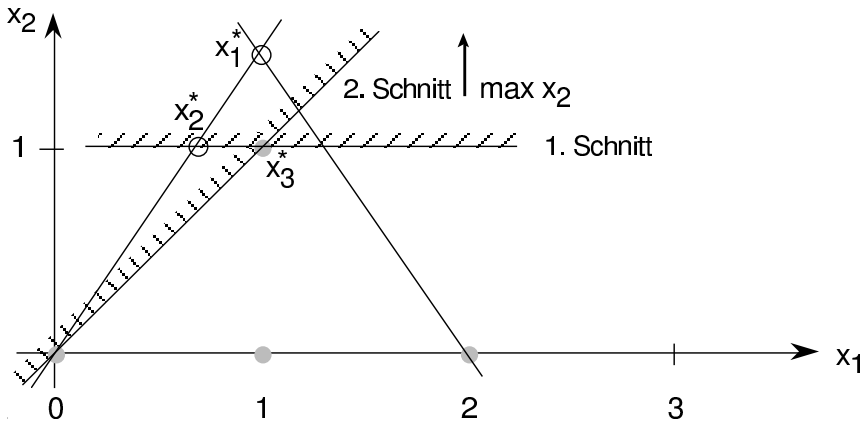


Abbildung 9.3: Optimallösungen von (LP1), (LP2), (LP3) und (ILP)

In Abbildung 9.3 ist $x_1^* = (1, \frac{3}{2})$ Optimallösung von (LP1), $x_2^* = (\frac{2}{3}, 1)$ Optimallösung von (LP2) und $x_3^* = (1, 1)$ Optimallösung von (LP3) und (ILP). $\diamond$

Wir haben hier eine Methode gesehen, die es erlaubt, für das Schnittebenenverfahren Schnitte zu generieren. Falls nur solche Gomory–Cuts verwendet werden, gilt sogar:

Theorem 9.5 *Betrachte das ILP*

$$\max\ c^T x \ \ \textit{so dass}\ Ax \leq b,\ \textit{und}\ 0 \leq x \leq d,\ \textit{sowie}\ x\ \textit{ganz}. \tag{9.26}$$

Es existiert eine Implemenation des Schnittebenenverfahrens, das nur Gomory–Cuts verwendet und das nach endlich vielen Iterationen entweder eine optimale Lösung des ILP findet, oder beweist, dass das ILP keine ganzzahlige Lösung besitzt.

Um die Endlichkeit sicherzustellen, wird das ILP zuerst auf die Form

$$\max\ z\ \text{ so dass } A'(x,z) = b',\ \text{ und } x, z \geq 0,\ \text{ sowie } x \text{ ganz} \tag{9.27}$$

zurückgeführt (vgl. dazu auch Beispiel 9.4). Als Gomory–Cut wird jeweils diejenige nicht–ganzzahlige Basisvariable gewählt, die den kleinsten Index besitzt. Das nach Zufügen dieser Restriktion resultierende LP wird dann so gelöst, dass es eine lexikographisch maximale optimale Lösung liefert (z.B. mit dem lexikographischen dualen Simplex–Algorithmus). Für den Beweis dieses Theorems wird auf die Literatur verwiesen, z.B. *G.L. Nemhauser, L.A. Wolsey: Integer and Combinatorial Optimization, Wiley 1988.*

Wir schliessen die theoretischen Betrachtungen zu den Schnittebenenverfahren und den Gomory-Cuts mit folgenden Bemerkungen ab:

Bemerkung 9.6

(1) Da in jeder Iteration $\mathbf{S} \subseteq P^t$ gilt, liefert die optimale Lösung x^* von $(LP)_t$ eine obere Schranke $OS_t := c^T x^*$ für den Optimalwert ω von (ILP). Weiter gilt $OS_0 \geq OS_1 \geq \cdots \geq OS_t \geq \omega$.

(2) Der Simplex–Algorithmus startet mit einer zulässigen Basislösung. Sei x^* eine optimale Basislösung von $(LP)_t$ und y^* eine optimale Basislösung des zugehörigen dualen Problems. Nach Zufügen eines Schnittes ist x^* keine zulässige Lösung vom neuen Problem $(LP)_{t+1}$ mehr. Hingegen lässt sich y^* einfach zu einer zulässigen Basislösung des zu $(LP)_{t+1}$ dualen Problems erweitern, nämlich zu $(y^*, 0)$, wobei die Variable, die zur neuen Restriktion assoziiert ist, den Wert 0 besitzt. Deshalb ist es

sinnvoll, $(LP)_{t+1}$ mit einem dualen Simplex–Algorithmus zu lösen. (Man bemerke, dass ein Gomory–Cut der Form (9.17) unmittelbar, ohne weitere Umformung, ins Endtableau eingefügt werden kann.)

(3) In jeder Iteration t des Schnittebenenverfahrens wird eine neue Restriktion eingefügt. Damit die Anzahl Restriktionen nicht zu gross wird, werden Schnitte $(a^{t'})^T x \leq a_{0'}^{t'}$, $t' \leq t$, die nicht zum ursprünglichen Problem gehören und die zur Beschreibung der optimalen Lösung x^* von $(LP)_t$ nicht verwendet werden, also $(a^{t'})^T x^* < a_{0'}^{t'}$, wieder gestrichen. Man bemerke, dass die Beziehung $OS_0 \geq OS_1 \geq \cdots \geq OS_t \geq \omega$ von Bemerkung (1) mit dieser Modifikation weiterhin gilt.

(4) Obwohl das Schnittebenenverfahren bei Verwendung von Gomory–Cuts endlich ist, ist es in der Praxis nicht effizient. Im allgemeinen werden sehr viele Schnitte benötigt, bis eine ganzzahlige Lösung gefunden wird. Deshalb werden für spezielle Probleme jeweils auch spezielle Schnitte verwendet, die die Problemstruktur berücksichtigen (vgl. Kapitel 9.2: Schnittebenenverfahren für das Traveling Salesman Problem). Auch werden Schnittebenenverfahren oft mit Branch and Bound–Verfahren (vgl. Kapitel 11) gekoppelt.

(5) Ein Nachteil des hier beschriebenen Schnittebenenverfahrens ist, dass die während des Algorithmus berechneten Lösungen x^* der $(LP)_t$'s für das (ILP) bis zum Ende des Algorithmus unzulässig sind. Möchte man den Algorithmus nach einer gewissen Anzahl Iterationen abbrechen (z.B. aus Rechenzeitgründen) hat man zwar eine obere Schranke für das (ILP) (siehe Bemerkung (1)), aber keine Lösung von (ILP), nicht einmal eine schlechte.

Es gibt Schnittebenenverfahren, primale Schnittebenenverfahren genannt, die nur im Zulässigkeitsbereich des (ILP) arbeiten. Ein solches Verfahren, angewendet auf das Traveling Salesman Problem, ist im Kapitel 9.2 beschrieben. Im allgemeinen benötigen solche Verfahren aber enorm viele Schnitte.

9.2 Schnittebenenverfahren für das Traveling Salesman Problem

9.2.1 Problemstellung

Das Traveling Salesman Problem (TSP) kann wie folgt formuliert werden: Gegeben sind n Städte, zusammen mit der $n \times n$–Matrix der Distanzen c_{ij} zwischen jedem Städtepaar i, j. Gesucht ist eine (geschlossene) Tour minimaler Länge, die jede Stadt genau einmal besucht (daher der Name *Problem des Handelsreisenden*). Falls die Distanzmatrix c_{ij} symmetrisch ist, d. h. $c_{ij} = c_{ji}$, spricht man vom symmetrischen, sonst vom asymmetrischen TSP.

In einem beliebigen ungerichteten (gerichteten) Graphen $G = (V, E)$ ist ein *Hamilton'–scher Kreis* (Zyklus) ein Kreis (Zyklus) von G, der sämtliche Knoten von G durchläuft. Das symmetrische (asymmetrische) TSP ist deshalb auch folgendes Optimierungsproblem in Graphen (Bemerke: Kanten–, bzw. Bogenlängen dürfen auch negativ sein):

Symmetrisches TSP:

Gegeben: $G = (V, E)$ ungerichtet und vollständig, $c_e \in \mathbb{R}$, $e \in E$.

Gesucht: Ein Hamilton'scher Kreis T^* minimaler Länge, d. h. so, dass $c(T^*) \equiv \sum_{e \in T^*} c_e = \min\{c(T) \mid T \text{ Hamilton'scher Kreis von } G\}$.

Asymmetrisches TSP:

Gegeben: $G = (V, E)$ gerichtet und vollständig, $c_e \in \mathbb{R}$, $e \in E$.

Gesucht: Ein Hamilton'scher Zyklus T^* minimaler Länge.

Bemerkung 9.7 Nicht jeder ungerichtete (gerichtete) Graph G besitzt einen Hamilton'schen Kreis (Zyklus). Wenn G einen solchen besitzt, heisst er *Hamilton'scher Graph*. Es gibt jedoch keine schöne Charakterisierung der Hamilton–Graphen, wie dies für Euler–Graphen der Fall ist (siehe Kapitel 6). Das Problem herauszufinden, ob ein Graph $G' = (V, E')$ ein Hamilton'scher Graph ist, ist ein TSP: ergänze G' zu einem vollständigen Graphen $G = (V, E)$ durch Hinzufügen aller fehlenden Kanten (Bögen), und setze $c_e = 0$ für $e \in E'$ und $c_e = K > 0$ für $e \in E \setminus E'$. G' ist ein Hamilton'scher Graph genau dann, wenn $c(T^*) = 0$ im TSP in G.

Das TSP (und das Problem der Abklärung, ob ein Graph ein Hamilton–Graph ist), ist ein schwieriges Problem, für welches kein Algorithmus bekannt ist, dessen Aufwand mit einem Polynom in der Anzahl Knoten begrenzt ist. Im folgenden wird ein Schnittebenenverfahren für das symmetrische TSP erläutert. Dazu werden zuerst gültige Ungleichungen für das TSP hergeleitet, die dann im Schnittebenenverfahren verwendet werden.

9.2.2 Gültige Ungleichungen für das TSP

Sei $G = (V, E)$ ungerichtet und vollständig mit $n \equiv |V|$ und $m \equiv |E| = \frac{n(n-1)}{2}$, und definiere für beliebige $S, T \subseteq V, D \subseteq E$ und $x \in \mathbb{R}^m$:

$$\gamma(S) = \{(i,j) \in E \mid i \text{ und } j \in S\} \tag{9.28}$$
$$\delta(S) = \{(i,j) \in E \mid i \in S, j \in V \setminus S\} \tag{9.29}$$
$$\delta(S,T) = \{(i,j) \in E \mid i \in S, j \in T\} \tag{9.30}$$
$$V(D) = \{i \in V \mid (i,j) \in D\} \tag{9.31}$$
$$x(D) = \sum\{x_{ij} \mid (i,j) \in D\} \tag{9.32}$$
$$x(S) = x(\gamma(S)) \tag{9.33}$$

Sei $A \in \mathbb{R}^{m \times n}$ die Knoten–Kanten Inzidenzmatrix von G, d. h. $A_{ik} = 1$, falls Knoten i Endknoten der Kante k ist, und 0 sonst, $i = 1, \ldots, n$, $k = 1, \ldots, m$. Betrachte das TSP auf G mit Kantenlängen c_{ij}, $1 \leq i < j \leq n$, und folgendes lineare Programm (LP)

$$\min_{x \in \mathbb{R}^m} \quad c^T x \tag{9.34}$$
$$\text{so dass} \quad Ax = \underline{2} \tag{9.35}$$
$$0 \leq x \leq \underline{1} \tag{9.36}$$

($\underline{2}$ und $\underline{1}$ sind Vektoren mit allen Komponenten gleich 2, respektive 1. Der Strich unter 2 und 1 sei im weiteren weggelassen, wenn aus dem Zusammenhang ersichtlich ist, ob es sich um den Vektor oder die Zahl handelt).
Das LP (9.34) - (9.36) hat ganzzahlige und nicht–ganzzahlige Basislösungen. Das Polyeder

$$Q_A^n = \{x \in \mathbb{R}^m \mid Ax = 2,\ 0 \leq x \leq 1\} \tag{9.37}$$

hat nämlich drei Typen von Eckpunkten:

(i) nicht ganzzahlige Eckpunkte (mit gewissen Komponenten $= \frac{1}{2}$),

(ii) ganzzahlige Eckpunkte, die Teiltouren entsprechen,

(iii) ganzzahlige Eckpunkte, die Touren entsprechen.

Ein Eckpunkt x von Q_A^n des Typs (ii) ist der Inzidenzvektor einer Familie von mehreren knotendisjunkten Kreisen, sog. Teiltouren, die insgesamt alle Knoten überdecken.

Beispiel 9.8

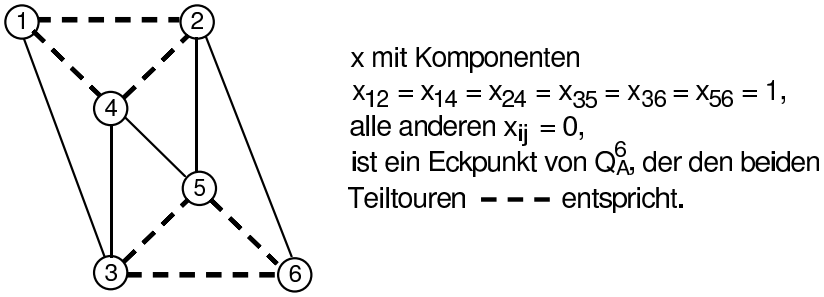


Abbildung 9.4: Ganzzahlige Eckpunkte, die Teiltouren entsprechen

$\diamond$

Um die Eckpunkte des Typs (ii) von Q_A^n abzuschneiden, führt man folgende *Schnittebenen*, die sog. *Teiltoureliminationsungleichungen* (von Dantzig) ein:

$$x(S) \leq |S| - 1, \qquad S \subset V, \quad 2 \leq |S| \leq \lfloor \frac{n}{2} \rfloor. \tag{9.38}$$

Sei nun $Q_S^n = \{x \in Q_A^n \mid x \text{ erfüllt } (9.38)\}$. Man bemerke, dass das TSP als ganzzahliges LP formuliert werden kann, nämlich:

$$\begin{array}{rl} \min & c^T x \\ \text{so dass} & x \in Q_S^n \\ & x \text{ ganz} \end{array} \tag{ILP}$$

Die Ganzzahligkeitsrestriktion kann aber nicht weggelassen werden, denn Q_S^n hat nicht ganzzahlige Eckpunkte, gewisse davon Eckpunkte von Q_A^n, andere durch Hinzunahme von (9.38) neu erzeugte.

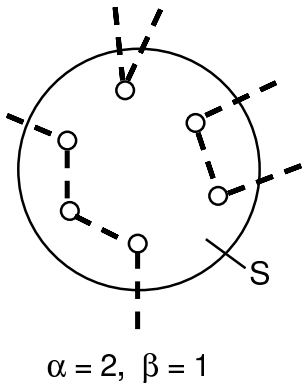
Die nicht–ganzzahligen Eckpunkte von Q_A^n werden von den sogenannten *2–Matching–Ungleichungen* (von Edmonds) abgeschnitten:

$$x(S) + x(M) \leq |S| + \lfloor \frac{|M|}{2} \rfloor, \tag{9.39}$$

wobei $S \subseteq V$ mit $3 \leq |S| \leq \lfloor \frac{n}{2} \rfloor$, und $M \subseteq \delta(S)$ und die Eigenschaft hat, dass die Endpunkte *in* S der Kanten von M alle *verschieden* sind.

Lemma 9.9 *Der Inzidenzvektor x jeder Tour T erfüllt (9.39).*

Beweis.



Nehme an, (S, T∩γ(S)) bestehe aus α nicht kantenleeren Komponenten und β Ein-Knoten-Komponenten. Es gilt

x(S) = |T∩γ(S)| = |S| - (α + β).

Sei ein M ⊆ δ(S) mit genannter Eigenschaft. Es gilt

x(M) = |T∩M| min {|M|, 2α + β}.

Abbildung 9.5: 2–Matching-Ungleichungen erfüllen 9.39

Falls $2\alpha + \beta \geq |M|$, ist $\alpha + \beta \geq \lceil \frac{|M|}{2} \rceil$ [1] und $x(M) \leq |M|$, und falls $2\alpha + \beta < |M|$, ist $\alpha \leq \lfloor \frac{|M|}{2} \rfloor$ und $x(M) \leq 2\alpha + \beta$. In beiden Fällen folgt (9.39). ♣

Auch $\{x \in Q_A^n \mid x$ erfüllt (9.38), (9.39)$\}$ hat nicht–ganzzahlige Eckpunkte. Gewisse von ihnen werden von den sog. Comb–Ungleichungen (Grötschel und Padberg) abgeschnitten.

[1] für beliebiges reelles η ist $\lceil \eta \rceil$ die kleinste Zahl $\geq \eta$: Aufrunden von η.

Definition 9.10 (Comb) *Ein Comb (Kamm auf Deutsch) C besteht aus $k+1$ Knotenmengen W_0 und W_i, $i = 1, \ldots, k$, k ungerade, die sich wie folgt schneiden:*

$$|W_0 \cap W_i| \geq 1, \qquad i = 1, \ldots, k \tag{9.40}$$
$$|W_i \setminus W_0| \geq 1, \qquad i = 1, \ldots, k \tag{9.41}$$
$$|W_i \cap W_j| = 0, \qquad 1 \leq i < j \leq k \tag{9.42}$$

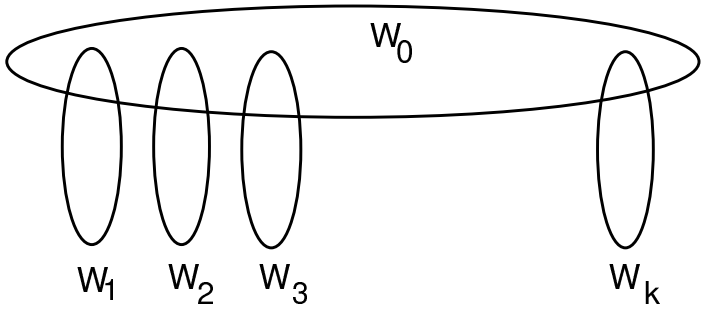


Abbildung 9.6: Comb

Theorem 9.11 (Comb (Kamm) Ungleichungen) *Sei $T_1, \ldots, T_s \subset V$, s ungerade und $T_i \cap T_j = \emptyset$ für alle $1 \leq i < j \leq s$. Sei $H \subset V$ mit $T_i \cap H \neq \emptyset$ und $T_i \setminus H \neq \emptyset$ für alle $1 \leq i \leq s$. Sei weiter $x \in \mathbb{R}^E$ ein Inzidenzvektor einer Tour, dann gilt:*

$$\sum_{e \in \delta(H)} x_e + \sum_{i=1}^{s} \sum_{e \in \delta(T_i)} x_e \geq 3s + 1 \tag{9.43}$$

Beweis. Für jede Teilmenge $A \not\subseteq S$ und x, dem Inzidenzvektor einer Tour, gilt:

$$\alpha(A) := \sum_{e \in \delta(A)} x_e \geq 2 \tag{9.44}$$

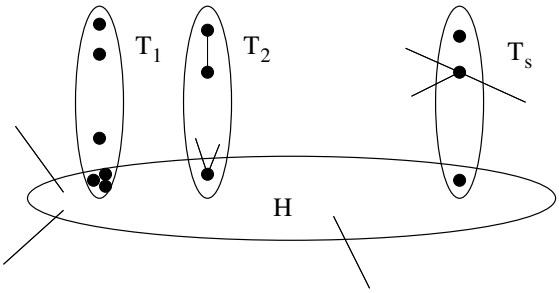
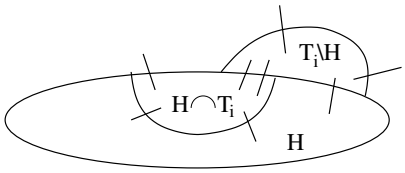


Abbildung 9.7: Comb Ungleichungen



Für alle i definiere σ_i durch

$$\begin{aligned}\sigma_i :&= \sum_{e\in\delta(T_i)} x_e + \sum_{e\in\delta(H),\, e\subset T_i} x_e && (9.45)\\ &= \frac{1}{2}(\alpha(T_i) + \alpha(T_i\cap H) + \alpha(T_i \setminus H)) && (9.46)\\ &\geq 3 && (9.47)\end{aligned}$$

Summieren über alle σ_i $i = 1 \ldots s$ ergibt

$$\alpha(H) + \sum_{i=1}^{s} \alpha(T_i) \geq \sum_{i=1}^{s} \sigma_i \geq 3s \tag{9.48}$$

Da $\alpha(A)$ gerade ist, ist die linke Seite gerade und folglich, grösser gleich 3s+1. ♣

Bemerkung 9.12 Für alle $A \subset S$ und x Inzidenzvektor einer Tour gilt offensichtlich

$$\begin{aligned}2|A| &= \sum_{v\in A} \sum_{e\in\delta(v)} x_e\\ &= \sum_{e\in\delta(A)} x_e + 2\sum_{e\in\gamma(A)} x_e\end{aligned} \tag{9.49}$$

mit $\gamma(A) = \{e \mid e \subset A\}$.

Die zum Comb C gehörende Comb–Ungleichung lautet:

$$(a^C x \equiv) \sum_{i=0}^{k} x(W_i) \leq |W_0| + \sum_{i=1}^{k} (|W_i| - 1) - \lceil \frac{k}{2} \rceil. \tag{9.50}$$

Lemma 9.13 *Der Inzidenzvektor x jeder Tour erfüllt (9.50).*

Beweis. Es gelten folgende Beziehungen:

$$2a^C x = \sum_{i=0}^{k} 2x(W_i) = 2x(W_0) + \sum_{i=1}^{k} (x(W_i) + x(W_i)) \tag{9.51}$$

$$2x(W_0) = \sum_{v \in W_0} x(\delta(v)) - x(\delta(W_0)) \tag{9.52}$$

$$x(W_i) = x(W_i \setminus W_0) + x(W_i \cap W_0) + x(\delta(W_i \setminus W_0), W_i \cap W_0)) \tag{9.53}$$

$$x(\delta(W_0)) \geq \sum_{i=1}^{k} x(\delta(W_i \setminus W_0, W_i \cap W_0)) \tag{9.54}$$

Deshalb ist

$$2a^C x = \sum_{v \in W_0} x(\delta(v)) + \sum_{i=1}^{k} \{x(W_i + x(W_i \setminus W_0) + x(W_i \cap W_0)\} \tag{9.55}$$

und, da T eine Tour ist, folgt

$$2a^C x \leq 2|W_0| + \sum_{i=1}^{k} \{|W_i| - 1 + |W_i \setminus W_0| - 1 + |W_i \cap W_0| - 1\}, \tag{9.56}$$

d. h.

$$2a^C x \leq 2\{W_0| + \sum_{i=1}^{k} (|W_i| - 1)\} - k. \tag{9.57}$$

Dividieren durch 2 und Aufrunden von $\frac{k}{2}$ gibt (9.50). ♣

Beweis. Zweiter Beweis durch Herleitung aus Theorem 9.11:

$$\begin{aligned} a^c x \equiv \sum_{i=0}^{k} x(W_i) &= \sum_{i=0}^{k} (|W_i| - \frac{1}{2}\alpha(W_i)) \\ &= |W_0| - \frac{1}{2}\alpha(W_0) + \sum_{i=1}^{k} (|W_i| - \frac{1}{2}\alpha(W_i)) \\ &= |W_0| + \sum_{i=1}^{k} |W_i| - \frac{1}{2}(\alpha(W_0) + \sum_{i=1}^{k} \alpha(W_i)) \\ &\leq |W_0| + \sum_{i=1}^{k} |W_i| - \frac{3k+1}{2} \\ &\leq |W_0| + \sum_{i=1}^{k} (|W_i| - 1) - \lceil \frac{k}{2} \rceil \end{aligned}$$

♣

Bemerkung 9.14 Eine Comb–Ungleichung mit $k = 1$ und $|W_0| = 1$ ist eine Teiltoureleminationsungleichung (9.38), und eine solche, die (9.40) und (9.41) mit Gleichheit erfüllt, eine 2–Matching–Ungleichung (9.39).

9.2.3 Ein Schnittebenenverfahren

In diesem Abschnitt wird ein spezielles Schnittebenenverfahren für das symmetrische TSP betrachtet, das die Strukturen des Problems gut ausnutzt. Als Schnittebenen werden nicht Gomory–Cuts verwendet, sondern nur eine spezielle Klasse **K** von Schnitten (Teiltoureliminations–, 2–Matching–, Combungleichungen, sowie weitere spezielle Schnitte).

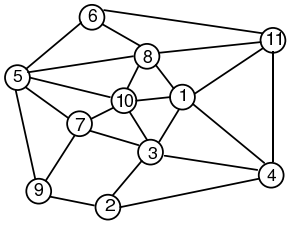
Algorithmus 9.2 Schnittebenenverfahren für das TSP

begin (* TSP *)
  Sei $P^0 := \{x \in \mathbb{R}^E \mid Ax = 2, 0 \leq x \leq 1\}, t := 0$
  keinSchnitt := false
  repeat
    löse $(LP)_t : \min cx$, so dass $x \in P^t$, erhalte Lösung x^*
    if x^* nicht Inzidenzvektor von Tour **then**
      finde von x^* verletzte Ungleichungen $a^i x \leq a^i_o, i \in I^t$, der Klasse **K**
      if $I^t \neq \emptyset$ **then**
        (* Behalte nur bindende Restriktionen *)
        $P^{t+1} := P^0 \cap \{x \in \mathbb{R}^E \mid a^i x \leq a^I_0, i \in \{i' \mid i' \in I^{t'}, t' < t$ und $a^{i'} x^* = a^{i'}_o\}$
        (* neue, verletzte Restriktionen zuführen *)
        $P^{t+1} := P^{t+1} \cap \{x \in \mathbb{R}^E \mid a^i x \leq a^i_0, i \in I^t\}, t := t + 1$
      else keinSchnitt := true
    end (* if *)
  until (x^* Inzidenzvektor von Tour) oder (keinSchnitt = true)
  if x^* Inzidenzvektor von Tour **then**
    x^* entspricht optimaler Tour
  else cx^* ist obere Schranke für die Länge einer optimalen Tour
end (* TSP *)

Man bemerke, dass bei diesem Verfahren jeweils inaktiv gewordene Schnittebenen entfernt werden, damit die in jeder Iteration zu lösenden LP's nicht zuviele Restriktionen besitzen. Auch können in einem Schritt mehrere Schnittebenen gleichzeitig neu eingefügt werden. Da nur eine spezielle Klasse von Schnitten verwendet werden, ist es möglich, dass das Verfahren keine zulässige Lösung, also keine Tour findet. Im allgemeinen ist aber die gefundene obere Schranke sehr gut. Um eine optimale Tour zu finden, wird das Schnittebenenverfahren in ein Branch and Bound–Verfahren eingebettet (vgl. Kapitel 11).

Beispiel 9.15 Im folgenden TSP-Beispiel ist dieses Schnittebenenverfahren veranschaulicht und numerisch durchgerechnet:

	1	2	3	4	5	6	7	8	9	10	11
1		33	10	38	33	30	18	15	38	8	32
2			23	45	50	61	38	46	20	31	65
3				37	33	38	15	23	35	8	42
4					71	68	52	53	65	45	44
5						25	20	25	30	25	59
6							40	15	55	30	40
7								25	20	10	50
8									45	15	34
9										30	70
10											40
11											

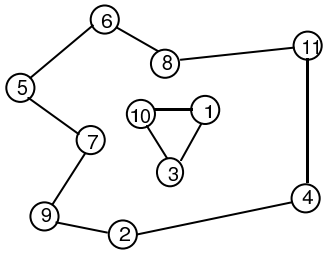


Schritt 1:

cx = min!
Ax = 2
$0 \le x \le 1$
L = 249,0

Im Graphen ablesbar:
- eingeführte Schnitte
- optimale Lösung , mit Optimalwert L

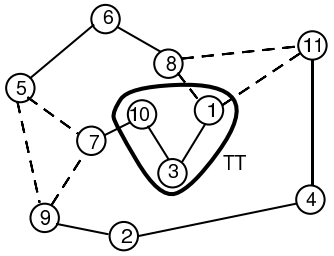
——: $x_{ij} = 1$, -----: $x_{ij} = 1/2$



Schritt 2:

zusätzlich Teiltoureliminationsungleichung:
TT: $x_{1,3}+x_{1,10}+x_{3,10} \le 2$

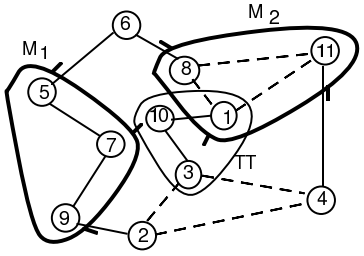
L = 252,5



Schritt 3:

zusätzlich 2-Matching-Ungleichungen:
M_1: $(x_{5,7}+x_{5,9}+x_{7,9}) + (x_{5,6}+x_{7,10}+x_{2,9}) \le 4$
M_2: $(x_{1,8}+x_{1,11}+x_{8,11})+(x_{6,8}+x_{1,3}+x_{4,11}) \le 4$

L = 253,0



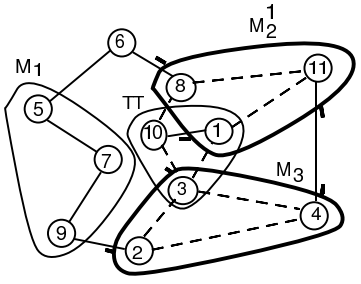
Schritt 4:

M_2 nicht aktiv, weglassen, zusätzlich:

M_2^1 : $(x_{1,8}+x_{1,11}+x_{8,11})+(x_{6,8}+x_{1,10}+x_{4,11}) \le 4$

M_3 : $(x_{2,3}+x_{2,4}+x_{3,4}) + (x_{2,9}+x_{3,10}+x_{4,11}) \le 4$

L = 254,0



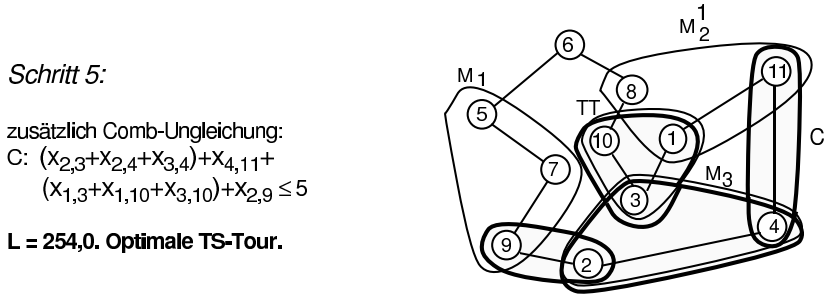


Abbildung 9.8: Numerisches Beispiel zu TSP und Schnittebenenverfahren

◇

Bemerkung 9.16 Das skizzierte Schnittebenenverfahren wurde mit verschiedenen Massnahmen (zusätzliche Klassen von Schnitten, effiziente Heuristiken zur Suche von Schnitten) weiter verfeinert und es werden TSP mit bis zu 1'000 Städten exakt gelöst.

9.2.4 Ein primales Schnittebenenverfahren

In diesem Abschnitt wird ein primales Schnittebenenverfahren, d. h. ein Verfahren, das im Zulässigkeitsbereich arbeitet, für das TSP beschrieben.
Ausgehend vom LP

$$\min \quad c^T x \tag{9.58}$$

$$\text{so dass} \quad Ax = 2 \tag{9.59}$$

$$0 \leq x \leq 1 \tag{9.60}$$

und einer heuristisch ermittelten Tour T, wird zu x^T eine Ausgangsbasis zugeordnet und jeweils ein Simplex–Austauschschritt vorgenommen[2], solange dieser entweder stationär ist oder zu einem benachbarten Eckpunkt führt, der auch Inzidenzvektor einer *Tour* T' ist (ersetze dann T durch T').
Sobald jedoch ein Austauschschritt zu einem benachbarten Eckpunkt x' führen würde, der keiner Tour entspricht, wird er nicht ausgeführt, sondern es wird eine Ungleichung des Typs (9.38), (9.39) oder (9.50) zum jeweiligen LP *zugefügt*, die

(i) von x^T mit Gleichheit erfüllt ist,

[2]Falls keine optimale Basis vorliegt.

(ii) x' abschneidet (von x' verletzt wird).

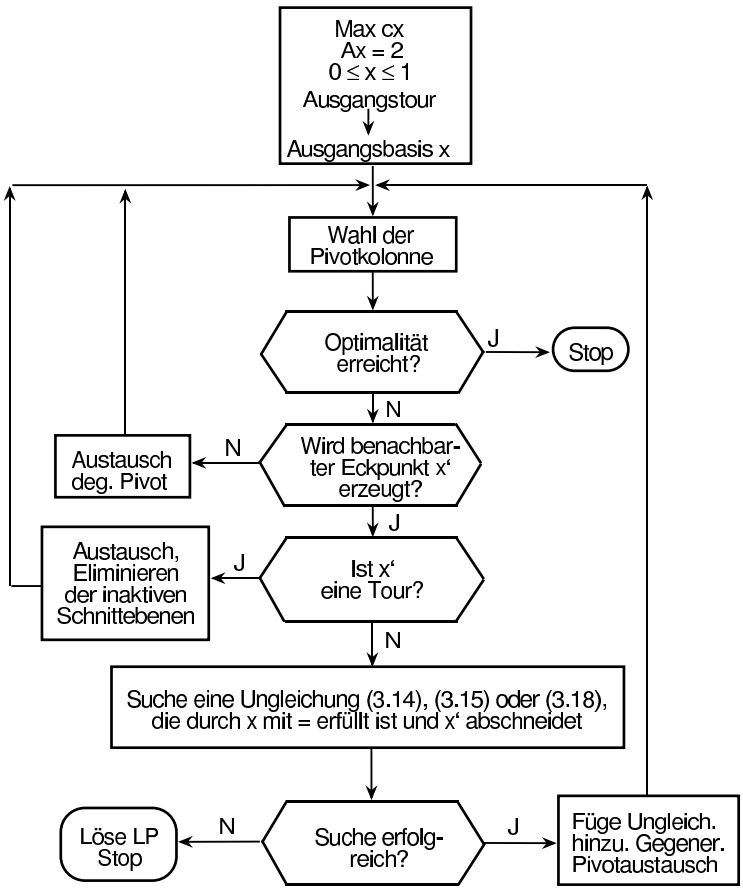


Abbildung 9.9: LP-basiertes Schnittebenenverfahren für TSP

Im erweiterten LP wird dann weiter unter obigen Regeln ausgetauscht. Vorheriges Flussdiagramm (siehe Abbildung 9.9) veranschaulicht das Verfahren. Um die Anzahl zugefügter Ungleichungen (Schnittebenen) in Grenzen zu halten, werden darin (eventuell unter Einbusse der Effizienz) die jeweils inaktiv gewordenen Schnittebenen wieder entfernt. Für das Problem der Suche einer geeigneten Ungleichung (9.38), (9.39) oder (9.50) werden i.a. Heuristiken verwendet. Man bemerke, dass das Verfahren gemäss Flussdiagramm das Auffinden einer optimalen Tour nicht *garantiert* (falls die Lösung des LP im

unteren Ausgang eine nicht ganzzahlige optimale Lösung liefert), es liefert aber immer eine Tour, die meistens sehr gut ist.

Bemerkung 9.17 Mit diesem Verfahren wurden TSP's mit über 300 Knoten exakt gelöst, wobei das Verfahren noch in ein Branch & Bound–Verfahren eingebettet wurde.

Kapitel 10

Lagrangerelaxation und Subgradientenmethode

In diesem Kapitel wollen wir ein Problem (P) der Form

$$w_p := \min \ f(x) \tag{10.1}$$
$$\text{so dass} \ \ g(x) = b \tag{10.2}$$
$$x \in \mathbf{C}_1 \tag{10.3}$$

betrachten, wobei $x \in \mathbb{R}^n$, $\mathbf{C}_1 \subseteq \mathbb{R}^n$, $f(x) : \mathbb{R}^n \to \mathbb{R}$, $g(x) : \mathbb{R}^n \to \mathbb{R}^m$, und $b \in Q^m$ ist. Wir gehen davon aus, dass $x \in \mathbf{C}_1$ „einfache“ Restriktionen und $g(x) = b$ „komplizierte“ Restriktionen sind, in dem Sinne, dass das relaxierte Problem

$$\min \ f(x) \tag{10.4}$$
$$\text{so dass} \quad x \in \mathbf{C}_1 \tag{10.5}$$

einfach lösbar ist und (P) schwer zu lösen ist. Wir wollen dazu ein Beispiel betrachten:

Beispiel 10.1 n Jobs müssen n Arbeitern so zugewiesen werden, dass jeder Job genau einem Arbeiter und keine zwei Jobs demselben Arbeiter zugewiesen werden. Seien c_{ij} die Bearbeitungszeit des Jobs j und t_{ij} die Ausbildungskosten, falls der Arbeiter i den Job j bearbeitet. Die Zuordnung soll so erfolgen, dass die Gesamtbearbeitungszeit der Jobs minimal ist und dass die Ausbildungskosten ein gewisses Budget nicht überschreitet.

Mit der Konvention

$$x_{ij} = \begin{cases} 1, & \text{Job } j\text{wird Arbeiter } i \text{ zugeordnet} \\ 0, & \text{sonst} \end{cases} \tag{10.6}$$

lässt sich dieses Problem wie folgt formulieren:

$$\begin{aligned} \min \quad & \sum_{i,j} c_{ij} x_{ij} \\ \text{so dass} \quad & y + \sum_{i,j} t_{ij} x_{ij} = b \\ & \sum_{j} x_{ij} = 1 \qquad \forall \text{ Arbeiter } i \\ & \sum_{j} x_{ij} = 1 \qquad \forall \text{ Job } i \\ & x_{ij} \in \{0,1\} \qquad \forall\, i,j \\ & y \geq 0 \end{aligned} \tag{P_1}$$

Lässt man in diesem Beispiel die Budget–Restriktion $\sum_{i,j} t_{ij} x_{ij} \leq b$ fallen, so erhält man ein Zuordnungsproblem (vgl. Kapitel 4 bzw. 5), das gut lösbar ist. Hingegen ist (P_1) ein schwieriges Problem. In diesem Beispiel könnte man

$$g(x) = b \quad \text{als} \quad \sum_{i,j} t_{ij} x_{ij} = b \tag{10.7}$$

und

$$\mathbf{C}_1 \quad \text{als} \quad \left\{ (x,y) \mid x_{ij} \in \{0,1\}\, \forall i,j; \sum_j x_{ij} = 1\, \forall i; \sum_i x_{ij} = 1\, \forall j, y \geq 0 \right\} \tag{10.8}$$

definieren. ◇

In diesem Kapitel wollen wir eine Methode betrachten, die das schwierige Problem (P) durch Lösen einer Folge von einfacheren Problemen der Form

$$\min\ f'(x),\ \text{so dass } x \in \mathbf{C}_1, \tag{10.9}$$

approximativ löst, wobei die veränderte Zielfunktion $f'(x)$ das Einhalten der Restriktion (10.2) begünstigt.

10.1 Lagrangerelaxation

Definition 10.2 (Lagrangeproblem) *Betrachte das Problem (P) und sei $u \in \mathbb{R}^m$. Das Problem $(LR(u))$, definiert durch*

$$w_{LR}(u) := \min \; f(x) + u^T(g(x) - b), \tag{10.10}$$

$$\textit{so dass} \quad x \in \mathbf{C}_1 \tag{10.11}$$

heisst das bezüglich $g(x) = b$ relaxierte Lagrangeproblem von (P).

$LR(u)$ enthält die „komplizierten Restriktionen“ nicht als Restriktionen. Stattdessen werden sie in der Zielfunktion berücksichtigt, indem eine nicht eingehaltene Restriktion $g(x)_i - b_i \neq 0$ mit „Strafkosten“ $u_i(g(x)_i - b_i)$ bestraft wird.
Zwischen dem Optimalwert w_P von Problem (P) und $w_{LR}(u)$ gilt folgende Beziehung:

Satz 10.3 *$w_P \geq w_{LR}(u)\, \forall u \in \mathbb{R}^m$, falls (P) eine zulässige Lösung hat.*

Beweis. Sei $u \in \mathbb{R}^m$.
Jede zulässige Lösung x' von (P) ist auch zulässig in $(LR(u))$ und

$$f(x') = f(x') + u(g(x') - b) \geq w_{LR}(u), \tag{10.12}$$

da $g(x') - b = 0$. Besitzt (P) keine beschränkte Lösung, so besitzt auch $(LR(u))$ wegen (10.12) keine beschränkte Lösung. Besitzt (P) eine Optimallösung x^*, so sagt (10.12), dass $f(x^*) = w_P \geq w_{LR}(u)$. ♣

Definition 10.4 (Lagrange–Dualproblem) *Das Lagrange-Dualproblem ist definiert durch*

$$w_{LD} := \max_{u \in \mathbb{R}^m} w_{LR}(u) \tag{LD}$$

Die beste untere Schranke für w_P erhält man, indem man Lagrange–Dualproblem löst.

Korollar 10.5 *Falls (P) eine zulässige Lösung besitzt, gilt $w_P \geq w_{LD}$.*

Beweis. Übung ♣

Bemerkung 10.6 Man bemerke, dass im allgemeinen $w_P > w_{LD}$ sein kann (siehe auch folgendes Beispiel)

Beispiel 10.7

$$\min \quad -x_1 - x_2 \tag{10.13}$$

$$\text{so dass} \quad x_1 + \frac{2}{3}x_2 = 1 \tag{10.14}$$

$$0 \leq x_1, x_2 \leq 1 \tag{10.15}$$

$$x_1, x_2 \text{ ganz} \tag{10.16}$$

Der Zulässigkeitsbereich von (P) ist nur der Vektor $x = (1, 0)$. Betrachte die Restriktion $x_1 + \frac{2}{3}x_2 = 1$ als komplizierte Restriktion. Das relaxierte Lagrange–Problem $(LR(u))$ lautet dann für $u \in \mathbb{R}$:

$$\min \quad -x_1 - x_2 + u(x_1 + \frac{2}{3}x_2 - 1) \tag{10.17}$$

$$\text{so dass} \quad 0 \leq x_1, x_2 \leq 1 \tag{10.18}$$

$$x_1, x_2 \text{ ganz} \tag{10.19}$$

oder, da $\{0 \leq x_1, x_2 \leq 1; x_1, x_2 \text{ ganz }\}$ gleich $\{(0,0); (1,0); (0,1); (1,1)\}$ ist, kann man $(LR(u))$ auch schreiben als:

$$w_{LR}(u) := \min \quad (u-1)x_1 + (23u-1)x_2 - u \tag{10.20}$$

$$\text{so dass} \qquad x \in \{(0,0); (1,0); (0,1); (1,1)\} \tag{10.21}$$

Der Optimalwert ist $w_{LR}(u) = \min\{-u, -1, -1 - \frac{1}{3}u, -2 + \frac{2}{3}u\}$.

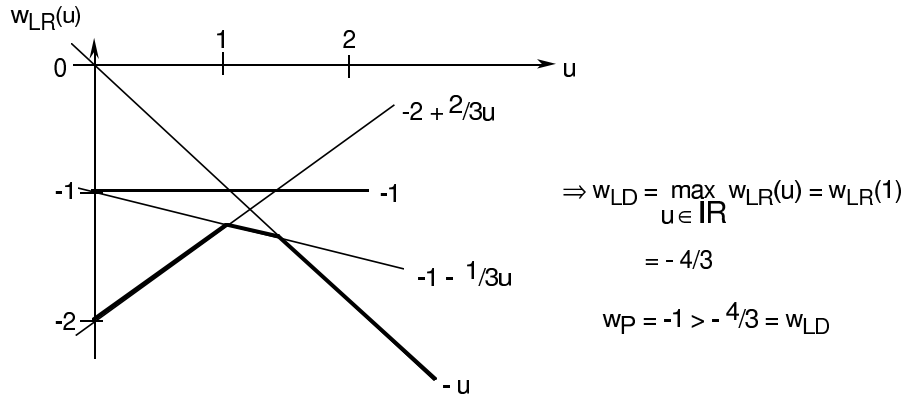


Abbildung 10.1: Optimalwert $w_{LR}(u)$

◇

Der Fall $w_P = w_{LD}$ kann durch folgende Bedingung charakterisiert werden:

Satz 10.8 *$w_P = w_{LD}$ genau dann, wenn ein u^* und eine für (P) zulässige Lösung x^* existiert, die eine Optimallösung von $LR(u^*)$ ist.*

Beweis.

$\Leftarrow$: $w_P \overset{(1)}{\leq} f(x^*) \overset{(1)}{=} f(x^*) + u^*(g(x^)) - b) \overset{(2)}{=} w_{LR}(u^*) \leq w_{LD} \overset{(3)}{\leq} w_P$, weil:

(1) x^* zulässig für (P)

(2) x^* optimal für $LR(u^*)$

(3) Korollar 10.5

$\Rightarrow w_{LD} = w_P$ und x^* ist Optimallösung von (P).

$\Rightarrow$: Sei $w_P = w_{LD}$ und x^* Optimallösung von (P), u^* Optimallösung von (LD). Wir zeigen, dass x^* Optimallösung von $LR(u^*)$ ist:

$$\begin{aligned} w_{LD} = w_{LR}(u^*) &= f(x^*) + w_{LR}(u^*) - f(x^*) \\ &= w_P + (w_{LR}(u^*) - f(x^*)) \\ &= w_{LD} + (w_{LR}(u^*) - f(x^*)) \end{aligned}$$

$\Rightarrow w_{LR}(u^*) = f(x^*) = f(x^*) + u^*(g(x^*) - b)$ und somit ist x^* eine Optimallösung von $LR(u^*)$.

♣

Insbesondere haben wir im ersten Teil des Beweises auch folgendes gezeigt:

Korollar 10.9 *Sei $u^* \in \mathbb{R}^m$ und x^* eine Optimallösung von $LR(u^*)$, die für (P) zulässig ist, dann ist x^* Optimallösung von (P).*

Das heisst, dass falls beim Lösen eines Problems $LR(u)$ eine Lösung x^* gefunden wird, die für (P) zulässig ist, dass dann auch (P) gelöst wurde.

Im allgemeinen kann $w_P > w_{LD}$ auftreten. Wir wollen nun einen Spezialfall, in dem $w_P = w_{LD}$ gilt, betrachten, nämlich den Fall, in dem (P) folgende

Form hat:

$$\begin{aligned} w_{P'} := \min \quad & c^T x \\ \text{so dass} \quad & A^1 x = b \\ & A^2 x \geq d \\ & x \geq 0 \end{aligned} \tag{10.22}$$

Seien $A^1 x = b^1$ die komplizierten Restriktionen, so lautet das relaxierte Lagrange–Problem für $u \in \mathbb{R}^m$:

$$\begin{aligned} w_{LR'}(u) := \min \quad & c^T x + u^T (A^1 x - b) \\ \text{so dass} \quad & A^2 x \geq d \\ & x \geq 0 \end{aligned} \tag{LR'(u))}$$

und das Lagrange–Dualproblem

$$w_{LD'} := \max \quad w_{LR'}(u) \tag{LD'}$$

Satz 10.10 *$w_{LD'} = w_{P'}$, falls (P') eine Optimallösung besitzt.*

Beweis. Betrachte das zu (P') duale Problem (DLP'):

$$\begin{aligned} w_{P'} = \max \quad & v^T b + y^T d \\ \text{so dass} \quad & v^T A^1 + y^T A^2 \leq c \qquad \text{(DLP')} \\ & y \geq 0 \qquad (10.23) \end{aligned}$$

und sei (v^*, y^*) eine optimale Lösung von (DLP'). Für $u := -v^*$ gilt:

$$w_{LD'} \geq w_{LR'}(u) = \min\,(c + u^T A^1)x - u^T b \overset{Dualität}{=} \max\, y^T d - u^T b \tag{10.24}$$

$$\overset{y^* zulässig}{\geq} y^{*T} d - u^T b = y^{*T} d + v^{*T} b = w_{P'} \tag{10.25}$$

(unter den Voraussetzungen, dass $A^2 x \geq d$, $x \geq 0$, bzw. $y^T A^2 \leq c + u^T A^1$, $y \geq 0$).

Also gilt $w_{LD'} \geq w_{P'}$ und da $w_{P'} \geq w_{LD'}$ (Korollar 10.5), folgt $w_{P'} = w_{LD'}$. ♣

Wir wollen nun sehen, wie man das Lagrange–Dualproblem approximativ lösen kann. Dazu gehen wir von der Annahme aus, dass das relaxierte Lagrange–Problem so definiert wurde, dass es gut lösbar ist.

10.2 Subgradientenmethode

In diesem Kapitel wollen wir eine iterative Methode betrachten, um ein Problem der Form

$$w^* = \max_{u \in \mathbb{R}^m} w(u) \tag{MK}$$

approximativ zu lösen, falls die Funktion $w : \mathbb{R}^m \to \mathbb{R}$ konkav ist.

Definition 10.11 (Konkave Funktion) *Die Funktion $w : \mathbb{R}^m \to \mathbb{R}^m$ ist* konkav*, falls für alle $u_1, u_2 \in \mathbb{R}^m$ und $0 \leq \alpha \leq 1$ gilt:*

$$w(\alpha u_1 + (1 - \alpha u_2) \geq \alpha w(u_1) + (1 - \alpha) w(u_2). \tag{10.26}$$

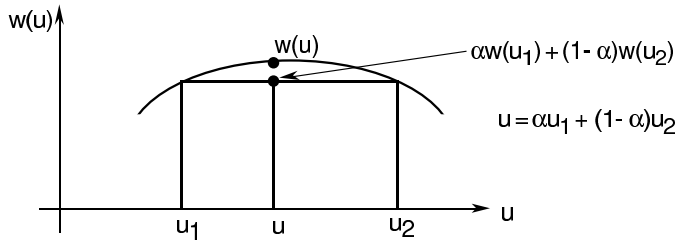


Abbildung 10.2: Konkave Funktion

Die Definition besagt, dass die Funktion $w(u)$ durch eine lineare Interpolation *unterschätzt* wird. Man bemerke, dass das Lagrange–Dualproblem

$$w_{LD} := \max_{u \in \mathbb{R}^m} w_{LR}(u) \tag{LD}$$

ein Problem der Form (MK) ist, denn es gilt folgender Satz:

Satz 10.12 *Die Funktion $w_{LR} : \mathbb{R}^m \to \mathbb{R}$ ist konkav (falls $\mathbf{C}_1 \neq \emptyset$).*

Beweis. Seien $u_1, u_2 \in \mathbb{R}^m$, so dass $LR(u_i), i = 1, 2$ ein Optimum besitzt und seien x_1 resp. x_2 Optimallösungen von $LR(u_1)$ resp. $LR(u_2)$. Sei weiter $0 \leq \alpha \leq 1$, $u := \alpha u_1 + (1 - \alpha) u_2$ und x eine zulässige Lösung von $LR(u)$.

$$\begin{aligned} f(x) + u(g(x) - b) &= f(x) + (\alpha u_1 + (1 - \alpha) u_2)(g(x) - b) \\ &= \alpha(f(x) + u_1(g(x) - b)) + (1 - \alpha)(f(x) + u_2(g(x) - b)) \\ &\geq \alpha(f(x_1) + u_1(g(x_1) - b)) + (1 - \alpha)(f(x_2) + u_2(g(x_2) - b)) \\ &= \alpha w_{LR}(u_1) + (1 - \alpha) w_{LR}(u_2). \end{aligned}$$

Also gilt für jede zulässige Lösung von $LR(u)$:

$f(x) + u(g(x) - b) \geq \alpha w_{LR}(u_1) + (1 - \alpha) w_{LR}(u_2)$

und damit auch

$$w_{LR}(u) \geq \alpha w_{LR}(u_1) + (1 - \alpha) w_{LR}(u_2)$$

♣

Man bemerke, dass die Funktion w_{LR} konkav und somit auch stetig ist, dass sie im allgemeinen aber nicht differenzierbar ist (vgl. Beispiel 10.7 und Abbildung 10.1). Um w^* des Problems (MK) zu finden, werden wir eine iterative Methode verwenden, d. h. wir werden eine Folge u_t, $t = 1, 2, \ldots$ konstruieren, die gegen u^* mit $w^* = w(u^*)$ konvergiert.

Falls $w(u)$ differenzierbar wäre, würde man die Folge

$$u_{t+1} := u_t + \delta_t \nabla_w(u_t), \quad t = 2, 3, \ldots \tag{10.27}$$

betrachten, wobei $\nabla w(u_t)$ der Gradient von w an der Stelle u_t ist, da $\nabla w(u_t)$ in die Richtung des grössten Zuwachses von $w(u)$ zeigt. Unter gewissen Annahmen für die Schrittweite δ_t konvergiert diese Folge gegen einen optimalen Punkt u^* (= Methode des steilsten Zuwachses).
Da in unserem Fall $w(u)$ nicht differenzierbar ist, betrachten wir an Stelle des Gradienten, der nicht definiert ist, einen sogenannten *Subgradienten*:

Definition 10.13 (Subgradient) *Sei $w : \mathbb{R}^m \to \mathbb{R}$ konkav. $h \in \mathbb{R}^m$ ist ein* Subgradient *von w an der Stelle u^*, falls*

$$w(v) \leq w(u^*) + h(v - u^*) \qquad v \in \mathbb{R}^m \tag{10.28}$$

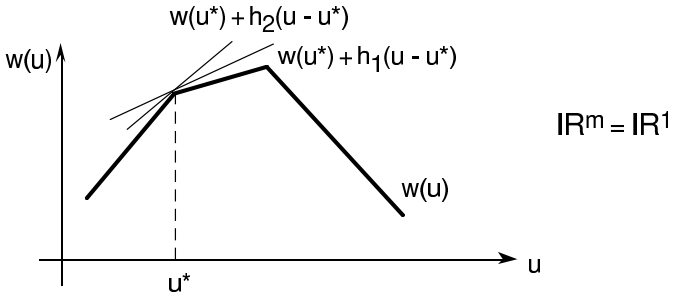


Abbildung 10.3: h_1 und h_2 sind Subgradienten von w an der Stelle u^*.

Ein Subgradient in u^* ist die Steigung einer Geraden, die durch den Punkt $(u^*, w(u^*))$ und oberhalb des Graphen der Funktion w verläuft.

Bemerkung 10.14

(i) Falls w konkav und differenzierbar ist, so gibt es in einem Punkt u^* genau einen Subgradienten, nämlich $\nabla w(u)$.

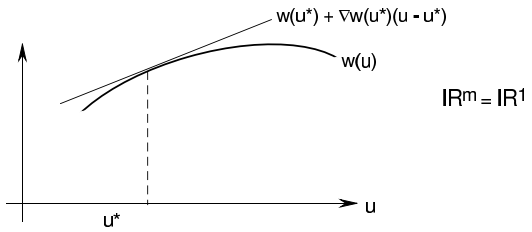


Abbildung 10.4: Subgradient bei konkavem und differenzierbarem w

(ii) Falls w konkav ist, können in einem Punkt u^* mehrere Subgradienten existieren (vgl. Abbildung 10.3).

(iii) Es ist möglich, dass für eine konkave Funktion w und einen Subgradienten h im Punkt u^* die Funktion w in Richtung des Subgradienten nicht zunimmt, d. h. es ist möglich, dass

$$w(u^*) > w(u^* + \delta \cdot h) \tag{10.29}$$

(iv) Falls $h = 0$ ein Subgradient in u^* ist, so ist u^* eine Optimallösung von (MK).

Der folgende Algorithmus verwendet Subgradienten, um w^* zu bestimmen:

Algorithmus 10.1 Subgradientenmethode

begin (* Subgradientenmethode *)
  $t := 0; u_0 := 0; w_{\max} := -\infty$
  repeat
    berechne $w(u_t)$
    if $w(u_t) > w_{\max}$ **then**
      $w_{\max} := w(u_t) u_{\max} := u_t$
    end; (* if *)
    berechne Subgradient h_t an der Stelle u_t
    $u_{t+1} := u_t + \delta_t h_t; t := t + 1$
  until ($h_t = 0$) oder (Abbruchkriterium erfüllt)
  if $h_t = 0$ **then** $w^* = w(u_{t-1})$
  else $w_{\max} = w(u_{\max})$ ist beste gefundene untere Schranke für w^*
end (* Subgradientenmethode *)

F ür die Schrittlänge δ_t werden in der Literatur folgende Möglichkeiten vorgeschlagen:

$\delta_t \to 0$ für $t \to \infty$ und $\sum_{t=1}^{\infty} \delta_t = \infty$: Falls die Schrittlänge so gewählt wird, konvergiert die Folge $(u_t), t = 0, 1, \ldots$ gegen eine Optimallösung u^*. In der Praxis ist diese Konvergenz aber sehr langsam.

$\delta_t := \delta_0 \cdot \rho_t$ mit $0 < \rho < 1$ oder

$\delta_t := \frac{(\overline{w} - w(u_t))\rho^t}{||h_t||^2}$: wobei $\overline{w}$ eine obere Schranke für w^* ist, und $0 < \rho < 1$. Unter gewissen Bedingungen auf (δ_0, ρ) oder $(\overline{w}, \rho)$ kann in beiden Fällen die Konvergenz $u_t \to u^*$, Optimallösung von (MK), bewiesen werden. In der Praxis werden sie empirisch ermittelt.

Idealerweise stoppt die Subgradientenmethode mit $h_t = 0$. In der Praxis ist dies nicht immer der Fall. Typische Abbruchkriterien sind durch eine maximale Anzahl Iterationen, oder durch eine maximale Anzahl Iterationen, während denen $w_{\max}$ nicht zugenommen hat, gegeben. Will man diese Subgradientenmethode auf unser Lagrange–Dualproblem anwenden, muss jeweils an der Stelle u_t ein Subgradient h_t gefunden werden. Dies ist sehr einfach zu finden, wie aus folgendem Satz hervorgeht:

Satz 10.15 (Auffinden eines Subgradienten) *Sei x^* eine Optimallösung von $LR(u_0)$, d. h. $w_{LR}(u_0) = f(x^*) + u_0(g(x^*) - b)$, dann ist*

$$h := g(x^*) - b \tag{10.30}$$

ein Subgradient der Funktion $w_{LR}(u)$ an der Stelle u_0.

Beweis.

$$w_{LR}(u_0) + (v - u_0) \cdot h = f(x^*) + u_0(g(x^*) - b) + (v - u_0)(g(x^*) - b) \tag{10.31}$$

$$= f(x^*) + v(g(x^*) - b) \tag{10.32}$$

$$\geq w_{LR}(v). \tag{10.33}$$

♣

10.3 Die 1–Baum–Relaxation für das Traveling Salesman Problem

Wir wollen nun eine Lagrangerelaxation und Subgradientenmethode auf das symmetrische Traveling Salesman Problem (TSP) anwenden (Problembeschreibung in Kapitel 9.2).
Dazu bezeichnen wir im vollständigen Graphen $G = (V, E)$ mit $n := |V|$ kurzerhand die Knoten mit i, $i = 1, \ldots, n$, die Kanten mit (i, j), $1 \leq i < j \leq n$, und (aus später ersichtlichen Gründen) die Kantenlängen c_{ij} mit c^0_{ij}, $1 \leq i < j \leq n$. Zudem sei ein Hamilton'scher Kreis minimaler Länge bei Kantenlängen $c = \{c_{ij} \mid 1 \leq i < j \leq n\}$ eine c–optimale Tour genannt.

Für das TSP gehen wir von der Formulierung

$$\begin{array}{ll} \min & c^T x \\ \text{so dass} & x \text{ erfüllt die Teiltoureliminationsungleichungen} \\ & Ax = 2 \\ & x \text{ ganz} \end{array} \tag{TSP}$$

aus (vgl. Kapitel 9.2.2). Für die Lagrangerelaxation schreiben wir (TSP) in der folgenden Form (wobei die zu $(i, j) \in E$ gehörige Komponente von x mit

$x_{ij} \equiv x_{ji}$ bezeichnet wird):

$$\begin{aligned}
\min \quad & \sum_{i<j} c_{ij} x_{ij} && && (10.34)\\
\text{so dass} \quad & \sum_{j} x_{ij} = 2 && i = 2, \ldots, n && (10.35)\\
& \sum_{j} x_{1j} = 2 && && (10.36)\\
& \sum_{i<j} x_{ij} = n && && (10.37)\\
& \sum_{(i,j)\in\gamma(S), i<j x_{ij}} \leq |S-1| && \forall\, S \subseteq V \setminus \{1\},\ |S| \geq 3 && (10.38)\\
& 0 \leq x_{ij} \leq 1 && i,j = 1,\ldots,n,\ i \neq j && (10.39)\\
& x_{ij} \text{ ganz} && i,j = 1,\ldots,n,\ i \neq j && (10.40)
\end{aligned}$$

Wir betrachten die Restriktionen (10.35) als komplizierte Restriktionen. Die zugehörige Lagrangerelaxation lautet dann für $u \in \mathbb{R}^{n-1}$:

$$\begin{aligned}
w(u) = \min \quad & \sum_{i<j} c_{ij} x_{ij} + \sum_{i=2}^{n} u_i \left(\sum_{j=1, j\neq i} x_{ij} - 2 \right) \\
\text{so dass} \quad & x \text{ erfüllt (10.36) - (10.40)}
\end{aligned} \tag{LR'}$$

oder, falls $u \in \mathbb{R}^n$ und $u_1 := 0$ definiert wird

$$\begin{aligned}
w(u) = \min \quad & \sum_{i<j} (c_{ij} + u_i + u_j) x_{ij} - \sum_{i=1}^{n} u_i \\
\text{so dass} \quad & x \text{ erfüllt (10.36) - (10.40)}
\end{aligned} \tag{LR"}$$

Wir definieren für $u \in \mathbb{R}^n$ den Vektor $c^u \in \mathbb{R}^n$ als

$$c^u_{ij} := c^0_{ij} + u_i + u_j, \quad 1 \leq i < j \leq n. \tag{10.41}$$

Das relaxierte Lagrangeproblem lautet nun für $u \in \mathbb{R}^n$ mit $u_1 = 0$

$$\begin{aligned}
w(u) = \quad & -2 \min \sum_{i=1}^{n} u_i + \min \sum_{i<j} c^u_{ij} x_{ij} \\
\text{so dass} \quad & x \text{ erfüllt (10.36) - (10.40)}
\end{aligned} \tag{LR}$$

und das Lagrange Dualproblem lautet

$$w^* = \max_{u \in \mathbb{R}^n,\, u_1 = 0} w(u) \tag{LD}$$

Wir müssen uns nun vergewissern, dass das relaxierte Lagrangeproblem (LR) wirklich einfacher lösbar ist als das ursprüngliche TSP. Betrachten wir die zulässigen Vektoren von (LR), also

$$X^B := \{x \mid x \text{ erfüllt (10.36) - (10.40)}\} \tag{10.42}$$

Ein $x \in X^B$ ist ein Inzidenzvektor einer Kantenmenge E_B von G und der Untergraph $B = (V, E_B)$ erfüllt folgende Bedingungen:

(a) der Knoten 1 ist mit genau zwei Kanten $(1, i)$ und $(1, j)$ von E_B inzident (wegen (10.36), (10.39) und (10.40)).

(b) $B \setminus \{1\} := (V \setminus \{1\}, E_B \setminus \{(1, i), (1, j)\})$ ist ein Baum von G (wegen (10.37) - (10.40), siehe Übungen über Gerüste). Anders ausgedrückt ist $B \setminus \{1\}$ ein Gerüst auf $G \setminus \{1\}$, den Graphen, der aus G durch Löschen des Knotens 1 und der mit ihm inzidenten Kanten entsteht.

Definition 10.16 (1–Baum) *Ein Teilgraph $B = (V, E_B)$ von G, der (a) und (b) erfüllt wird ein* 1–Baum *genannt.*

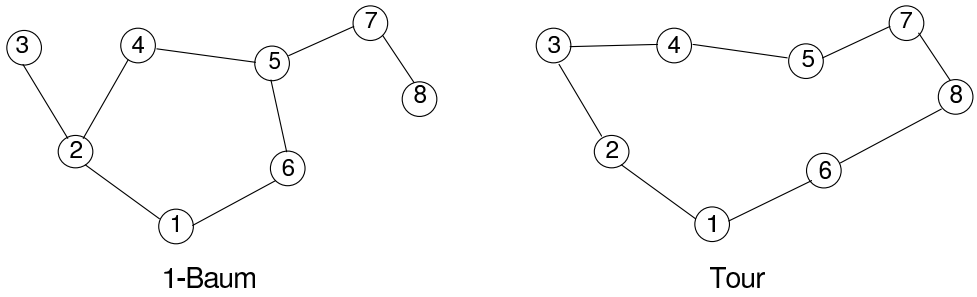


Abbildung 10.5: Zusammenhang Tour und 1–Baum

Bezeichne mit **B** die Familie aller 1–Bäume von G. Die Menge X^B entspricht also den Inzidenzvektoren der 1–Bäume und deshalb gilt:

Satz 10.17 *Bei gegebenem $u \in \mathbb{R}^n$, mit $u_1 = 0$ ist*

$$w(u) = \min\{c^u(B) \mid B \in \mathbf{B}\} - 2\sum_{i=1}^{n} u_i. \tag{10.43}$$

Beweis. Übung. ♣

Bevor wir darauf eingehen, wie man einen c^u–optimalen 1–Baum finden kann, wollen wir einige Beziehungen, die zwischen 1-Bäumen und Touren bestehen, besprechen. Man bemerke, dass jede Tour ein 1–Baum ist, bei welchem jeder Knoten den Grad 2 (im 1–Baum) besitzt. Es gilt also:

Lemma 10.18 *Jede Tour ist ein 1–Baum. Für beliebiges c^u, c^u–optimaler Tour T^u und c^u–optimalem 1–Baum B^u gilt:*

$$c^u(T^u) \geq c^u(B^u) := \min\{c^u(B) \mid B \in \mathbf{B}\} \tag{10.44}$$

Beweis. Übung. ♣

Weiter gilt:

Satz 10.19 *Sei $u \in \mathbb{R}^n$, mit $u_1 = 0$, T^u eine c^u–optimale Tour und weiter T^0 eine c^0–optimale Tour. Es gilt:*

$$c^0(T^0) \geq w(u) \tag{10.45}$$

Falls zusätzlich T^u ein c^u–optimaler 1–Baum ist, so ist T^u eine c^0–optimale Tour, also

$$c^0(T^u) = c^0(T^0) = w(u) \tag{10.46}$$

Beweis. Obwohl (10.45) direkt aus Satz 10.3 folgt, wollen wir es hier, speziell auf Touren und 1–Bäume bezogen, nochmals beweisen.
Seien T eine beliebige Tour und B ein beliebiger 1–Baum. Es gilt

$$\begin{aligned} c^u(T) &:= \sum_{(i,j)\in T} c^u_{ij} = \sum_{(i,j)\in T} c^0_{ij} + 2\sum_{i=1}^{n} u_i \\ &= c^0(T) + 2\sum_{i=1}^{n} u_i \end{aligned} \tag{10.47}$$

und

$$\begin{aligned} c^u(B) \equiv \sum_{i,j\in B} c^u_{ij} &= \sum_{i,j\in B} c^0_{ij} + \sum_{i=1}^{n} u_i d_i(B) \\ &= c^0(B) + \sum_{i=1}^{n} u_i d_i(B), \end{aligned} \tag{10.48}$$

wobei $d_i(B)$ den Grad des Knotens i in B bezeichnet, $i = 1, \ldots, n$. (10.47), zusammen mit der Tatsache, dass jede Tour ein 1–Baum ist, gibt

$$\begin{aligned} c^0(T^0) &= c^u(T^0) - 2\sum_{i=1}^{n} u_i \\ &\geq c^u(B^u) - 2\sum_{i=1}^{n} u_i \\ &= w(u). \end{aligned}$$

Die Behauptung (10.46) folgt direkt aus (10.45), denn

$$\begin{aligned} c^0(T^u) &= c^u(T^u) - 2\sum_{i=1}^{n} u_i \\ &= c^u(B^u) - 2\sum i = 1^n u_i \\ &= w(u) \leq c^0(T^0) \end{aligned} \tag{10.49}$$

♣

Wir kommen nun zur wichtigen Frage, wie sich für $u \in \mathbb{R}^n$, mit $u_1 = 0$

$$w(u) = \min\{c^u(B) \mid B \in \mathbf{B}\} - 2\sum_{i=1}^{n} u_i \tag{10.50}$$

berechnen lässt. Da ein 1–Baum die Vereinigung eines Gerüsts auf $G \setminus \{1\}$ mit zwei zum Knoten 1 inzidenten Kanten ist, lässt sich $w(u)$ mit folgendem Algorithmus berechnen:

Algorithmus 10.2 Berechnung von $w(u)$

begin (* $w(u)$ berechnen *)
(* Berechnung eines c_u–optimalen 1-Baumes *)
Wähle zwei Kanten $(1,i)$ und $(1,j)$ mit kleinster und nächst kleinster Länge
c^u_{1i}, c^u_{1j}n $\{(1,v) \mid v \in V \setminus (1)\}$
$E_B := \{(1,i),(1,j)\}$
Bestimme ein Gerüst $(V\setminus\{1\}, E')$ minimalen Gewichts bezüglich c^u in $G \setminus \{1\}$
(vgl. Kapitel 2)
$E_B := E_B \cup E'$
$B^u := (V, E_B)$
(* Berechne $w(u)$ *)
$w(u) := c^u(B^u) - 2\sum u_i$
end (* $w(u)$ berechnen *)

Das Lagrange–Dualproblem

$$w^* = \max_{u\in\mathbb{R}^n,\, u_1=0} w(u) \tag{LD}$$

kann mit einer Subgradientenmethode approximativ gelöst werden. Man bemerke, dass $w(u)$ gerade die Form hat

$$\begin{aligned} w(u) &= \min\{c^u(B) - 2\sum_{i=1}^n u_i \mid B \in \mathbf{B}\} \\ &= \min\{c^0(B) - 2\sum_{i=1}^n u_i d_i(B) - 2\sum_{i=1}^n u_i \mid B \in \mathbf{B}\} \\ &= \min\{c^0(B) - 2\sum_{i=1}^n (d_i(B)-2) u_i \mid B \in \mathbf{B}\} \end{aligned}$$

Somit gilt wegen Satz 10.15:

Satz 10.20 *$w(u)$ ist eine stückweise lineare, konkave Funktion mit Subgradientenvektor*

$$h(u) := (d_1(B^u) - 2, d_2(B^u) - 2, \ldots, d_n(B^u) - 2) \tag{10.51}$$

im Punkt u, wobei B^u ein c^u–optimaler 1–Baum ist.

Satz 10.20 besagt, dass bei der Berechnung von $w(u)$ an der Stelle u gemäss obigem Algorithmus, sozusagen gratis, ein Subgradient bestimmt wird, nämlich der Vektor $h(u)$, dessen Komponenten die Abweichungen vom (gewünschten) Grad 2 der Grade $d_i(B^u)$ der Knoten im c^u–optimalen 1–Baum B^u sind.

Die Subgradientenmethode des Kapitels 10.2, angewendet auf die Maximierung der Funktion $w(u)$ für das TSP, lautet nun

Algorithmus 10.3 1–Baum Relaxation für das TSP

begin (* TSP *)
  $u(0) := 0; m := 0; wmax := -\infty$
  stop := false
  repeat
    (* Bestimmung von $w(u)$ *)
    Bestimme einen $c^{u(m)}$–optimalen 1–Baum $B^{u(m)}$
    $w(u(m)) := c^{u(m)}(B^{u(m)}) - 2\sum u_i(m)$
    if $B^{u(m)}$ keine Tour **then**
      if $w(u(m)) > wmax$ **then**
        $wmax := w(u(m))$; $B^{best} := B^{u(m)}$
      end (*if *)
      (* Subgradient *)
      $h(u(m)) := (d_1(B^{u(m)}) - 2, d_2(B^{u(m)}) - 2, \ldots, d_n(B^{u(m)}) - 2)$
      $u(m+1) := u(m) + \delta_m h(u(m))$
      if Abruchkriterium erfüllt **then** stop := true
    else stop := true
  until stop = true
  if $B^{u(m)}$ Tour **then** $B^{u(m)}$ ist c^0–optimale Tour der Länge $w(u(m))$
  else $wmax$ ist untere Schranke für die Länge einer minimalen Tour
end (* TSP *)

Bemerkung 10.21 Auf eine mögliche Wahl der Schrittlänge und des Abbruchkriteriums wird im Kapitel 11 eingegangen.

Zusammenfassend wurde in diesem Abschnitt das TSP auf G mit Kantenbewertung c durch eine Folge von einfacheren Problemen, nämlich die Bestimmung eines $c^{u(m)}$–optimalen 1–Baums in G bei Kantenbewertungen

$c^{u(0)} = c, c^{u(1)}, \ldots, c^{u(m)}, \ldots$ ersetzt. Dieses Verfahren liefert eine gute untere Schranke für die minimale Länge einer Tour. Zudem, falls in der Folge ein $c^{u(m)}$–optimaler Baum eine Tour ist, ist diese Tour natürlich c-optimal. Das Eintreffen dieses günstigen Falls ist nicht garantiert, deshalb wird die erläuterte Methode in einem Branch and Bound–Verfahren eingebettet (siehe Kapitel 11). Dennoch bemerke man, dass die Art, wie die Kantenlängen $c^{u(0)} = c, c^{u(1)}, \ldots, c^{u(m)}$ in der Folge geändert werden, das Bilden eines optimalen 1–Baums mit Grad 2 in jedem Knoten, d.h. einer Tour, begünstigt.

Beispiel 10.22 Die Zahlen auf den Kanten sind die Längen $c^{u(m)}, m = 0, 1, 2, 3$. Ein $c^{u(m)}$–optimaler 1–Baum ist jeweils mit gestrichelten Linien angegeben.

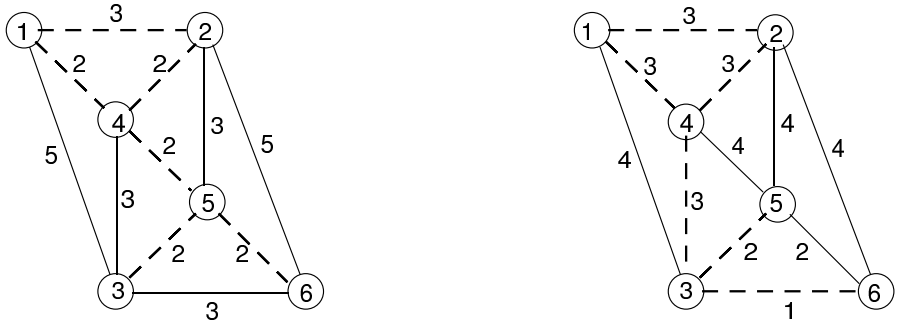


Abbildung 10.6: Schritte 0 und 1

In der linken Abbildung sehen wir die Ausgangslage für $m = 0$. $u(0)$ ist gleich $(0, 0, 0, 0, 0, 0)$, $w(u(0)) = 13$ und der Subgradient $h(u(0)) = (0, 0, -1, 1, 1, -1)$. Die Schrittweite δ_0 ist (und bleibt in diesem Beispiel) 1.

Für $m = 1$ erhalten wir $u(1) = (0, 0, -1, 1, 1, -1)$, $w(u(1)) = 15$, sowie $h(u(1)) = (0, 0, 1, 1, -1, -1)$ und $\delta_1 = 1$.

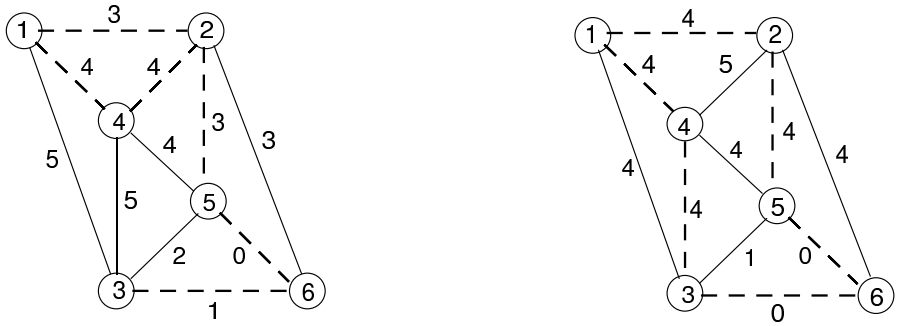


Abbildung 10.7: Schritte 2 und 3

Für $m = 2$ erhalten wir $u(2) = (0, 0, 0, 2, 0, -2)$, $w(u(2)) = 15$, sowie $h(u(2)) = (0, 1, -1, 0, 0, 0)$ und $\delta_2 = 1$.

Schliesslich erhalten wir in der dritten Iteration ($m = 3$) für $u(3) = (0, 1, -1, 2, 0, -2)$, $w(u(3)) = 16$ und $h(u(3)) = (0, 0, 0, 0, 0, 0)$, $\delta_3 = 1$. Der 1–Baum ist eine Tour, also eine optimale Tour. $\diamond$

Kapitel 11

Branch and Bound Algorithmus

Das Branch and Bound (B&B)–Verfahren ist eine Methode, um mittels intelligenter Enumeration ein (schwieriges) kombinatorisches Optimierungsproblem zu lösen. Das Wort „Branch“ (= Verzweigung) bezieht sich auf die Enumeration, das Wort „Bound“ (= Schranke) auf die Intelligenz, die in der Benützung von Schranken besteht, um eine vollständige Enumeration aller zulässigen Lösungen zu vermeiden.

11.1 Branch and Bound für die ganzzahlige Lineare Programmierung

Wir wollen die Idee des B&B am Beispiel der ganzzahligen linearen Programmierung erläutern. Betrachte ein Problem

$$\begin{aligned} z_0 = \min \quad & c^T x \\ \text{so dass} \quad & Ax \leq b \\ & x \geq 0 \\ & x \text{ ganz} \end{aligned} \tag{ILP_0}$$

und seine LP–Relaxation

$$\begin{aligned} US_0 = \min \quad & c^T x \\ \text{so dass} \quad & Ax \leq b \qquad (LP_0) \\ & x \geq 0 \qquad (11.1) \end{aligned}$$

Eine Optimallösung x_0 von (LP_0) wird im allgemeinen nicht ganzzahlig sein, sein Optimalwert $US_0 = c^T x^0$ ist aber eine untere Schranke für den Optimalwert z_0 von (ILP_0). Sei nun x^0 nicht ganzzahlig. Im Schnittebenenverfahren würden wir neue Schnitte zu (LP_0) hinzufügen, die x^0 abschneiden. Hier wollen wir das Problem (ILP_0) in zwei Teilprobleme unterteilen. Sei z.B. $x_i^0 \neq \mathbf{Z}$, so definieren wir 2 Probleme:

$$
\begin{array}{lllll}
(ILP_1) & z_1 = \min & c^T x & (ILP_2) & z_2 = \min \; c^T x \\
& \text{so dass} & Ax \leq b & & \text{so dass} \; Ax \leq b \\
& & x \geq 0 & & x \geq 0 \\
& & x_i \leq \lfloor x_i^0 \rfloor & & x_i \leq \lfloor x_i^0 \rfloor + 1 \\
& & x \text{ ganz} & & x \text{ ganz}
\end{array}
$$

Es ist klar, dass eine Optimallösung x^* von (ILP_0) entweder Lösung von (ILP_1) oder von (ILP_2) ist, da $x_i^* \in \mathbf{Z}$ und deshalb entweder $x_i^* \leq \lfloor x_i^0 \rfloor$ oder $x_i^* \geq \lfloor x_i^0 \rfloor + 1$. Es gilt also

$$z_0 = \min \{z_1, z_2\}. \tag{11.2}$$

Als nächstes wählen wir eines der Unterprobleme, z.B. (ILP_1). Wiederum versuchen wir, das Problem (ILP_1) zu lösen, indem das relaxierte LP, (LP_1) gelöst wird. Falls die Lösung x^1 von (LP_1) nicht ganzzahlig ist, unterteilen wir das Problem (ILP_1) wieder in zwei Unterprobleme (ILP_3) und (ILP_4).

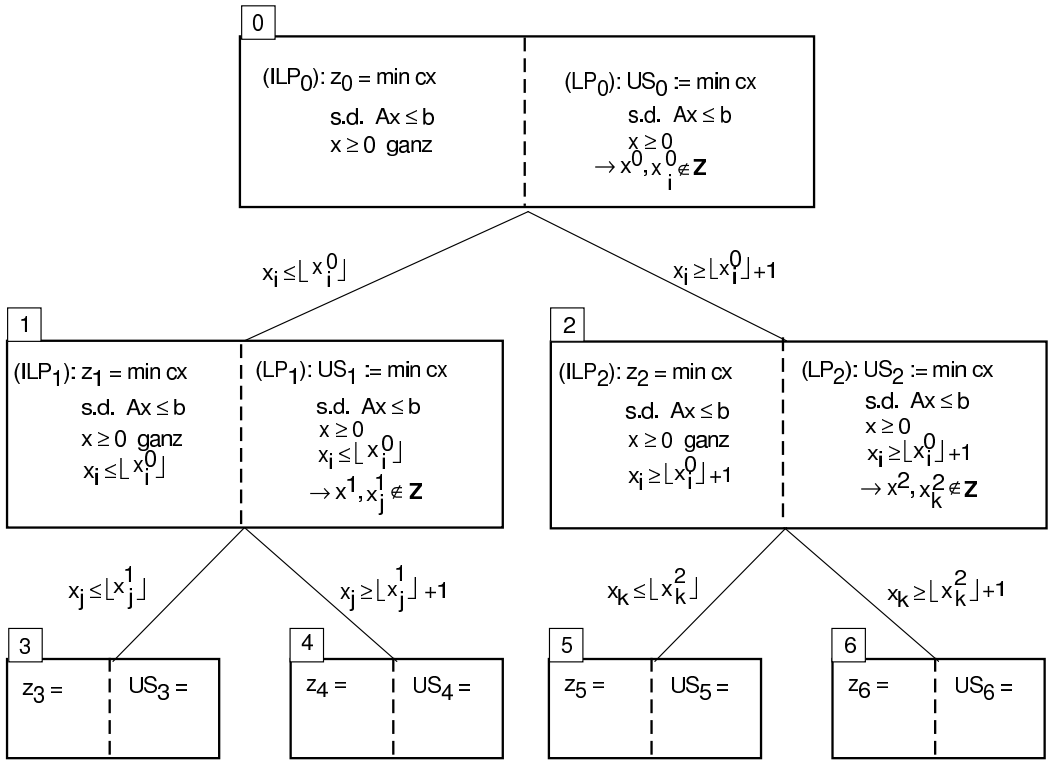


Abbildung 11.1: Problemaufsplittung bei Branch und Bound Methode

Diese Problemaufsplitterung kann in einem Baum (siehe Abbildung 11.1) dargestellt werden. Ein Problem (ILP_v) wird nicht mehr weiter unterteilt, falls das relaxierte LP (LP_v) entweder eine ganzzahlige Lösung x_v oder keine zulässige Lösung besitzt. Im ersten Fall wurde eine Optimallösung von (ILP_v) gefunden, im zweiten Fall wurde gezeigt, dass (ILP_v) keine zulässige Lösung besitzt. Falls wir diesen Verzweigungsprozess weiterführen, bis kein Blatt weiterverzweigt werden kann, so ist die beste Lösung unter den Blättern des Enumerationsbaumes eine Optimallösung des ursprünglichen Problems (ILP_0).

Bis dahin haben wir den Branching–Teil des B&B–Verfahrens beschrieben. Wir müssen nun noch den Bound–Aspekt besprechen. Nehmen wir an, dass wir beim Lösen des (LP_v) eine ganzzahlige Lösung x^v und somit Optimallö-

sung des Problems (ILP_v) gefunden haben. Dann ist

$$z_0 \leq z_v = US_v = c^T x^v, \tag{11.3}$$

und z_v ist eine obere Schranke für den Optimalwert z_0 von (ILP_0). Betrachten wir nun einen Knoten k (Problem ILP_k), für den x^k nicht ganz ist, aus dem noch nicht verzweigt wurde und für den $US_k \geq z_v$ ist. Da US_k eine untere Schranke für den Optimalwert von (ILP_k) ist, gilt für die Optimallösung $\hat{x}^k$ von (ILP_k)

$$c\hat{x}^k \geq US_k \geq z_v = c^T x^v. \tag{11.4}$$

Somit hat $\hat{x}^k$ sicher keinen besseren Zielfunktionswert als die bis dahin beste gefundene Lösung von (ILP_0). Für uns ist es deshalb nicht interessant, die Lösung von (ILP_k) zu kennen, dieses Problem muss nicht weiter untersucht werden, es kann *gestrichen* werden.

Beispiel 11.1 Wir wollen nun dieses B&B–Verfahren an einem kleinen numerischen Beispiel illustrieren (siehe auch Abbildung 11.2):

$$\begin{aligned} (ILP_0) \qquad z_0 = \min \quad & -x_1 - x_2 \\ \text{so dass} \quad & x_2 \geq \frac{1}{2} \\ & 8x_1 + 3x_2 \leq 20 \\ & 5x_1 - 2x_2 \geq 0 \\ & x_1, x_2 \text{ ganz} \end{aligned}$$

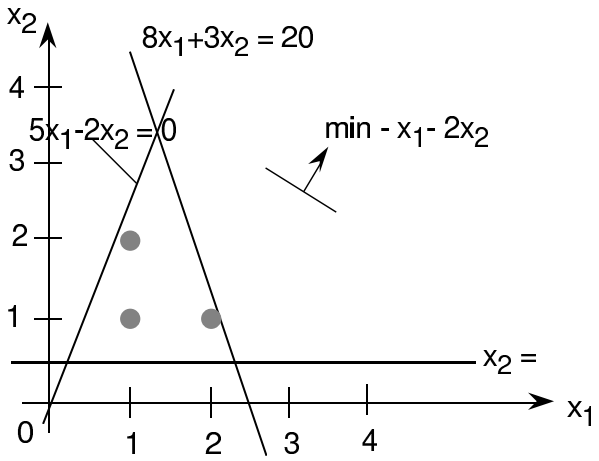


Abbildung 11.2: Branch und Bound Beispiel: Ausgangsproblem

Zuerst versuchen wir, das Problem (ILP_0) zu lösen, indem das zugehörige relaxierte (LP_0) gelöst wird. Als Lösung erhalten wir $x_0 = (\frac{40}{31}, \frac{100}{31})$ und eine untere Schranke $c^T x^0 = -\frac{240}{31}$ für z_0. Da z_0 ganzzahlig ist, gilt auch $z_0 \geq US_0 := \lceil c^T x^0 \rceil = -7$. x^0 ist nicht ganzzahlig, deshalb wird (ILP_0) in zwei Unterprobleme (ILP_1) und (ILP_2) unterteilt, wobei für (ILP_1) zusätzlich $x_1 \leq 1 = \lfloor \frac{40}{31} \rfloor$ und für (ILP_2) zusätzlich $x_2 \geq 2 = \lfloor \frac{40}{31} \rfloor + 1$ zugefügt wird. Die Problemaufteilung ist jeweils in Abbildung 11.3, der zum B&B gehörige Enumerationsbaum in Abbildung 11.4 dargestellt.
Wählt man als nächsten *aktiven* Knoten, also Knoten im Enumerationsbaum, dessen Problem noch nicht behandelt wurde, den Knoten 1, so erhält man als Lösung des zum (ILP_1) gehörigen LP's $x^1 = (1, \frac{5}{2})$ und eine untere Schranke $US_1 = \lceil c^T x^1 \rceil c^T x 1_= - 6$. x^1 ist nicht ganzzahlig, also wird (ILP_1) unterteilt in (ILP_3) und (ILP_4), wobei für (ILP_3) die Restriktion $x_2 \leq 2 = \lfloor \frac{5}{2} \rfloor$ und für (ILP_4) die Restriktion $x_2 \geq 3 = \lfloor \frac{5}{2} \rfloor + 1$ zugefügt wird.
Wählen wir als nächstes den Knoten 3, so erhält man $x_3 = (1, 2)$ mit unterer Schranke $US_3 = c^T x^3 = -5$. x^3 ist ganz und somit Optimallösung von (ILP_3). x^3 ist natürlich auch zuläss ige Lösung von (ILP_0), also ist $OS := c^T x^2 = -5$ eine obere Schranke für z_0.
Betrachten wir als nächstes den Knoten 4, so merken wir, dass der Zulässigkeitsbereich des zu (ILP_4) gehörigen LP's leer ist. Somit besitzt auch (ILP_4) keine zulässige Lösung. Der letzte aktive Knoten ist der Knoten 2. $x^2 = (2, \frac{4}{3})$ und die untere Schranke für (ILP_2) ist $US_2 = \lceil cx^2 \rceil = -4$. Also

besitzt (ILP_2) keine bessere Optimallösung als x^3. Der Knoten 2 muss nicht weiterverfolgt werden, er wird gestrichen.

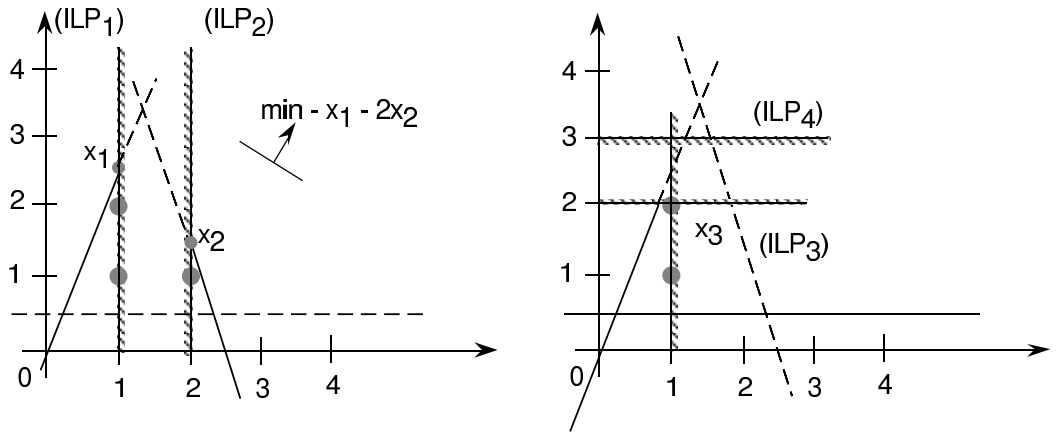


Abbildung 11.3: Branch und Bound Beispiel: Branching-Schritte

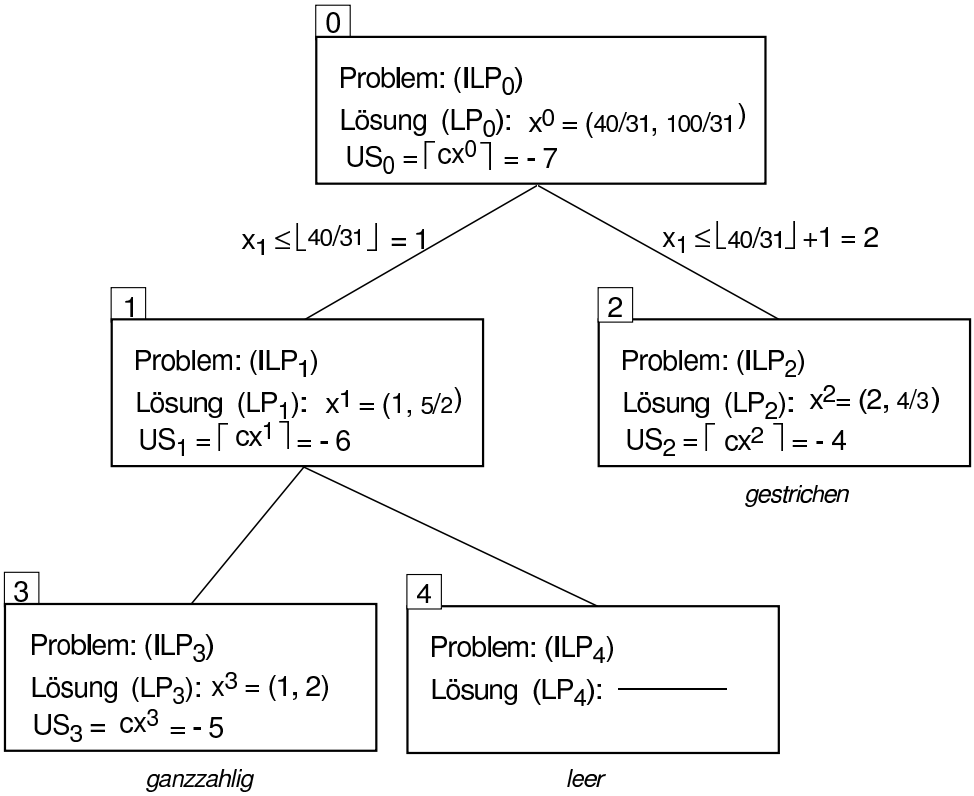


Abbildung 11.4: Branch und Bound Beispiel: Enumerationsbaum

◇

Bemerkung 11.2 Für diesen Algorithmus muss noch spezifiziert werden, welcher aktive Knoten jeweils als nächster Knoten gewählt wird und nach welcher nicht–ganzzahligen Variablen verzweigt wird. Auf diese zwei Punkte wird im nächsten Abschnitt eingegangen.

11.2 Allgemeiner Branch and Bound Algorithmus

Um für ein Minimierungsproblem einen B&B Algorithmus zu entwickeln, müssen zwei Operationen definiert werden, nämlich die

Verzweigung: Ein Problem P muss in zwei oder mehrere Unterprobleme U_i, $i \in I$, der gleichen Art unterteilt werden können, so dass die Vereinigung der zulässigen Lösungen der U_i's mindestens eine Optimallösung von P enthält (vgl. Abbildung 11.5). Anders gesagt, muss die beste Lösung aus den Optimallösungen der U_i's eine Optimallösung von P sein. Dieser Verzweigungsmechanismus muss auch endlich sein, d. h. der Enumerationsbaum, der durch rekursives Durchführen der Verzweigung entsteht, muss endlich sein.

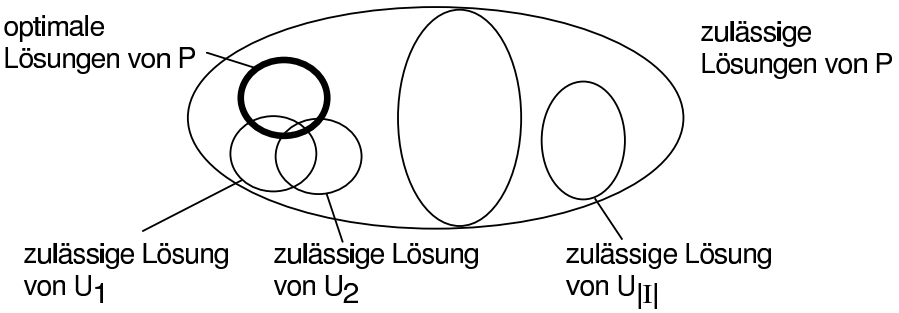


Abbildung 11.5: Vereinigung von zulässigen Lösungen

Schrankenberechnung: Es muss ein Algorithmus bekannt sein, um für ein Problem P eine untere Schranke für den Optimalwert von P zu berechnen.

Bemerkung 11.3 Man bemerke, dass es möglich ist, dass bei der Verzweigung die zulässigen Lösungen der Unterprobleme nicht disjunkt sind. Aus Aufwandsüberlegungen ist es aber wünschenswert, dass sie disjunkt sind.

Die Schrankenberechnung kann unmittelbar bei der Erzeugung von Unterproblemen erfolgen, oder bei der Behandlung eines Knotens (unmittelbar vor der

Verzweigung). Zum Teil werden auch an beiden Stellen Schranken (mit unterschiedlichen Algorithmen) berechnet. Der Vorteil, falls die Schranke direkt bei der Erzeugung der Kinder berechnet wird, liegt darin, dass gewisse Knoten direkt bei der Erzeugung gestrichen werden können und deshalb nicht abgespeichert werden müssen.

Nachfolgend ist ein allgemeiner B&B Algorithmus angegeben, wobei ein Knoten k im Enumerationsbaum durch das Paar (P_k, US_k) definiert ist und P_k das jeweilige Problem, US_k eine untere Schranke für den Optimalwert von P_k bedeutet. Wird eine Lösung für P_k gefunden, wird sie mit x^k bezeichnet. Wird festgestellt, dass P_k keine zulässige Lösung besitzt, wird $US_k := \infty$ gesetzt.

Algorithmus 11.1 B & B Algorithmus für ein Minimierungsproblem P

begin (* B&B *)
  $OS :=$ obere Schranke für P (* $OS = \infty$ oder Wert einer heuristisch gefundenen Lösung von P *)
  $P_0 := P$; berechne untere Schranke US_0 für P_0 (und evtl. Lösung x^0)
  if P_0 gelöst **then**
    $OS := US_0$; $x_{Best} := x^0$
  else
    $aktivL := (P_0, US_0)$ (* Liste der aktiven Knoten *)
    while $aktivL \neq \emptyset$ **do**
      wähle $(P_k, US_k) \in aktivL$; $aktivL := aktivL \setminus (P_k, US_k)$
      if $US_k < OS$ **then** (* sonst Knoten streichen *)
        versuche US_k zu verbessern und evtl. P_k zu lösen
        if $US_k < OS$ **then** (* sonst Knoten streichen *)
          if (P_k gelöst) **then** (* bis jetzt beste Lösung *)
            $OS := US_k$, $x_{Best} := x^k$ (* x^k Lösung von P_k *)
          else (* Verzweigen *)
            Generiere Kinder (P_i, US_i), $i \in I_k$, von P_k (und evtl. Lösung x^i)
            for all $i \in I_k$ **do**
              if $US_i < OS$ **then** (* sonst Knoten streichen *)
                if (P_i gelöst) **then** (* bis jetzt beste Lösung *)
                  $OS := US_i, x_{Best} := x^i$ (x^i Lösung von P_i *)
                else $aktivL := aktivL \cup (P_i, US_i)$
              end (* if *)
            end (*for *)
          end (* else *)
        end (* if *)
      end (* if *)
    end (* while*)
  end (*else *)
  if $OS < \infty$**then** x_{Best} ist Optimallösung von P mit Wert OS
  else P besitzt keine zulässige Lösung
end (* B&B *)

Bemerkung 11.4 Man bemerke, dass beim B&B für ganzzahlige lineare Programmierung beim Generieren der Kinder von P_k keine untere Schranke

für die Unterprobleme P_i berechnet wurde. In diesem Fall kann man $US_i := US_k$ setzen.

Wir wollen nun noch die Wahl des aktiven Knotens und die Wahl der Verzweigungsregel diskutieren.

Wahl des aktiven Knotens: Wir geben hier zwei häufig angewendete Regeln an, es können aber auch andere Regeln verwendet werden.

(i) *Depth First Serch:* Es wird jeweils ein im Enumerationsbaum möglichst tiefer Knoten gewählt. D.h.: Wird aus einem Knoten weiterverzweigt, so wird als nächster aktiver Knoten einer seiner Kinder gewählt. Wird nicht weiterverzweigt, gehen wir entlang dem Weg zur Wurzel zurück bis zu einem Knoten, der ein aktives Kind besitzt. Dieser wird als nächster aktiver Knoten gewählt. Diese Regel ist sehr einfach handhabbar: Als nächster aktiver Knoten wird der zuletzt in die Liste der aktiven Knoten augenommene Knoten gewählt.

(ii) *Beste Schranke:* Bei dieser Regel wird als nächster aktiver Knoten ein aktiver Knoten mit kleinster unterer Schranke gewählt ($\approx$ beste Chance, die optimale Lösung zu finden).

Die Regel (i) hat gegenüber (ii) zwei Vorteile: Bei (i) ist die Anzahl zu speichernder aktiver Knoten relativ klein, bei (ii) kann sie sehr gross sein, was zu Speicherplatzproblemen führen kann. In der Praxis hat sich gezeigt, dass zulässige Lösungen eher tief im Enumerationsbaum gefunden werden. Die Regel (i) führt somit schneller zu einer zulässigen Lösung und zu einer oberen Schranke, die zum Streichen von Knoten verwendet werden kann.

Die Regel (ii) braucht vielleicht lange, bis sie eine zulässige Lösung findet. Falls sie eine zulässige Lösung findet, wird sie vermutlich gut sein, und somit eine gute obere Schranke liefern, die zum Streichen von Knoten verwendet werden kann.

Wahl der Verzweigung: Nehmen wir an, wir befinden uns in einem Knoten k mit unterer Schranke US_k aus dem weiter verzweigt werden soll. Nehmen wir weiter an, dass verschiedene Möglichkeiten $\{M_1, M_2, \ldots, M_m\}$ zur Verfügung stehen. Im Fall der ganzzahligen linearen Programmierung (Kap. 11.1) kann z. B.. nach irgendeiner nicht–ganzzahligen Komponente von x^k (= Optimallösung des relaxierten LP's) verzweigt werden, die Möglichkeiten sind dann die Komponenten $\{j \mid x_j^k \text{ nicht ganz}\}$.

Um zu entscheiden, welche Möglichkeit M_j im Knoten k angewendet werden soll, wird für jede Möglichkeit M_j und jedes bei der Verzweigung M_j erzeugte Unterproblem , $U_i^j, i \in I_k^j$, ein Wert D_i^j definiert, wobei

$$D_i^j = \text{Abschätzung der Zunahme der unteren Schranke } US_i^j - US_k, \text{ wobei } US_i^j \text{ eine untere Schranke für das Problem bedeutet.} \tag{11.5}$$

Betrachten wir wieder das Beispiel der ganzzahligen linearen Programmierung: Für die Möglichkeit nach der nicht–ganzzahligen Komponente x_j^k zu verzweigen, könnte für das Unterproblem U_1^j mit $x_j \leq \lfloor x_j^k \rfloor, D_1^j := x_j^k - \lfloor x_j^k \rfloor$, und für das Unterproblem U_2^j mit $x_j \geq \lfloor x_j^k \rfloor + 1, D_2^j := x_j^k - \lfloor x_j^k \rfloor + 1 - x_j^k$definiert werden, in der Annahme, dass die Zunahme der unteren Schranke proportional zu diesen Werten ist. Eine andere, etwas aufwendigere Art, D_i^j zu definieren, wäre folgende Möglichkeit: Füge zum Problem (ILP_k) gehörigen relaxierten LP die Restriktion $x_j \leq \lfloor x_j^k \rfloor$, für $i = 1$, und $x_j \geq \lfloor x_j^k \rfloor + 1$, für $i = 2$, hinzu und führe, ausgehend von der zu x^k gehörigen Basis, eine Iteration des dualen Simplex aus. Der erhaltene Zielfunktionswert US_i^j ist eine untere Schranke für das Unterproblem U_i^j und deshalb kann $D_i^j := US_i^j - US_k$ definiert werden.

Sind die Werte für jede Möglichkeit M_j und jedes zugehörige Unterproblem U_i^jgegeben, so werden in der Literatur zwei Wahlkriterien für die Wahl der Verzweigungsmöglichkeit gegeben:

(i) *Wähle Möglichkeit M_j mit* $\min_i\{D_i^j\} = \max_{j'}\min_i\{D_i^{j'}\}$*:* Das $\min_i(US_i^j)$ ist eine untere Schranke für das zum Knoten k gehörige Problem. Es wird diejenige Verzweigungsmöglichkeit M_j gewählt, für die diese Schranke möglichst gross ist (= beste Schranke).

(ii) *Wähle Möglichkeit M_j mit* $\max_i\{D_i^j\} = \max_{j'}\max_i\{D_i^{j'}\}$*:* Hier ist die Idee, dass ein Unterproblem wegen der Schranke ziemlich sicher gestrichen werden kann.

Welche Möglichkeit für ein B&B–Verfahren nun angewendet werden soll, hängt stark vom jeweiligen Problem ab und meistens sind viel

Intuition und Experimente notwendig, um einen effizienten Algorithmus dieses Typs zu entwickeln. Dabei muss vor allem abgewogen werden, ob viel Aufwand in die Berechnung einer guten unteren Schranke und/oder einer guten Verzweigungsmöglichkeit investiert werden soll, um den Enumerationsbaum klein zu halten, oder ob dieser Aufwand lieber etwas kleiner gehalten wird und der Enumerationsbaum etwas grösser wird.

Bemerkung 11.5 Man bemerke, dass Schnittebenenverfahren und die Subgradientenmethode (zusammen mit der Lagrange–Relaxation) einfach in ein B&B–Verfahren eingebettet werden können, liefern doch beide Verfahren gute untere Schranken für das jeweilige Problem. Als Beispiel wollen wir die 1-Baum–Relaxation für das Traveling Salesman–Problem in ein B&B einbetten.

11.3 Branch and Bound mit der 1–Baum–Relaxation

Wir wollen nun die im Kapitel 10.3 besprochene 1–Baum–Relaxation in einem B&B–Verfahren einbetten. Es ist ja durchaus möglich, dass die 1–Baum–Relaxation keine Tour findet. Man bemerke, dass selbst wenn $w^* = \max_u w(u)$ erreicht wird, es nicht immer einen c^{u^*}–optimalen 1–Baum gibt, welcher eine Tour ist, d. h. $w^* < c^0(T^0)$ ist möglich.

Im Traveling Salesman Problem (TSP) wird in der Regel das Verzweigen durch *Hineinzwingen* oder *Ausschliessen* bestimmter Kanten erzielt. Offensichtlich sind die resultierenden Unterprobleme auch TSP's. Im folgenden sei das auf Held und Karp zurückzuführende B&B–Verfahren erläutert.

Ein Knoten im Enumerationsbaum wird charakterisiert durch $[X, Y, u, w_{X,Y}(u)]$, wobei

X	die Menge der hineingezwungenen Kanten,	(11.6)
Y	die Menge der ausgeschlossenen Kanten,	(11.7)
u	die Stelle, an welcher die untere Schranke berechnet ist,	(11.8)
$w_{X,Y}(u)$	die untere Schranke für das durch X, Y charakterisierte TSP	(11.9)

bezeichnen. $w_{X,Y}(u)$ berechnet sich analog zu (10.41), Kapitel 10.2:

$$w_{X,Y}(u) = \min\{c^u(B) \mid B \in \mathbf{B}_{X,Y}\} - 2\sum_{i=1}^{n} u_i, \tag{11.10}$$

wobei

$$\mathbf{B}_{X,Y} = \{B \mid B1\text{–Baum von } G \text{ mit } X \subseteq B \text{ und } Y \cap B = \emptyset\}. \tag{11.11}$$

Aus dem Knoten $[X, Y, u, w_{X,Y}(u)]$ wird wie folgt verzweigt: Die Kanten aus $E - (X \cup Y)$, die sog. *freien* Kanten, werden geordnet nach dem Betrag, um welchen die Schranke zunimmt, falls die Kante ausgeschlossen wird. D.h. falls $e(1), e(2), \ldots, e(r)$ die Reihenfolge dieser Kanten ist, gilt:

$$w_{X,Y\cup e(1)}(u) \geq w_{X,Y\cup e(2)}(u) \geq \cdots \geq w_{X,Y\cup e(r)}(u) \tag{11.12}$$

Die Unterprobleme werden dann wie folgt definiert:

$$X_1 = X \qquad Y_1 = Y \cup e(1) \tag{11.13}$$
$$X_2 = X \cup e(1) \qquad Y_2 = Y \cup e(2) \tag{11.14}$$
$$X_3 = X \cup e(1) \cup e(2) \qquad Y_3 = Y \cup e(3) \tag{11.15}$$
$$\ldots \qquad \ldots \tag{11.16}$$
$$X_q = X \cup e(1) \cup \cdots \cup e(q-1) \qquad Y_q = Y \cup R \tag{11.17}$$

q ist dabei der kleinste Index, bei dem durch Hinzufügen von $e(q-1)$ sich der Grad bezüglich X_q für mindestens einen Knoten (evtl. für 2 Knoten) auf 2 erhöht. R ist dann die Menge der mit diesem Knoten (diesen 2 Knoten) inzidenten Kanten aus $E \setminus X_q$.

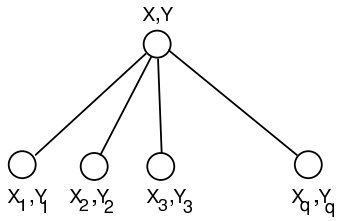


Abbildung 11.6: Unterprobleme

Beispiel 11.6 Das folgende Beispiel veranschaulicht diesen Verzweigungsmechanismus.

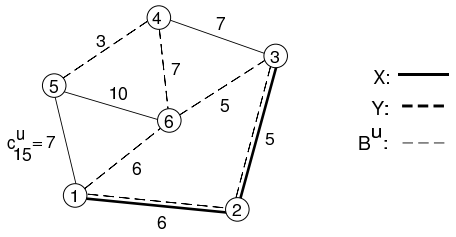


Abbildung 11.7: TSP-Beispiel

$e(i)$	$(4,5)$	$(3,6)$	$(1,6)$	$(4,6)$
$w_{X,Y\cup e(i)}(u)-$	7	2	1	0

Die Verzweigung aus dem Knoten $[X, Y, u, w_{X,Y}(u)]$ ergibt demnach folgende drei Nachfolgerknoten im Enumerationsbaum (bemerke, dass hier $Y = \emptyset$). Die Mengen X_i und $Y_i, i = 1, 2, 3$, sind aus den zugeordneten Graphen ersichtlich (sie sind in der unteren Abbildung dargestellt, wie es X und Y in obiger Abbildung sind). Bemerke, dass bei Hinzufügen der Kante $(6, 3)$ in X_2 der Grad (bezüglich X_3) des Knotens 3 zwei geworden ist und somit ist diese Kante $e(q-1)$ und $R = \{(3, 4)\}$.

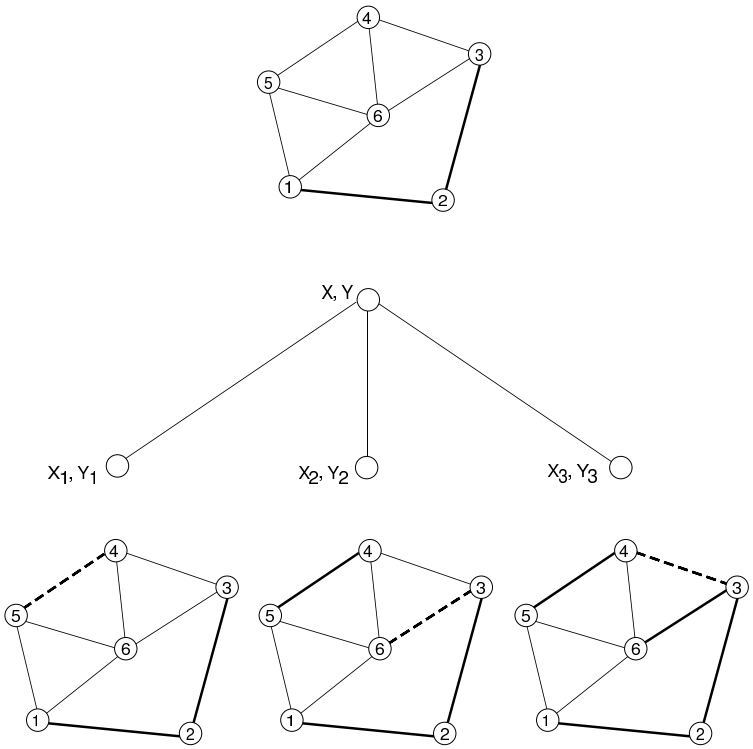


Abbildung 11.8: TSP-Beispiel: Verzweigung

◇

Folgendes Flussdiagramm stellt den Ablauf des B & B dar:

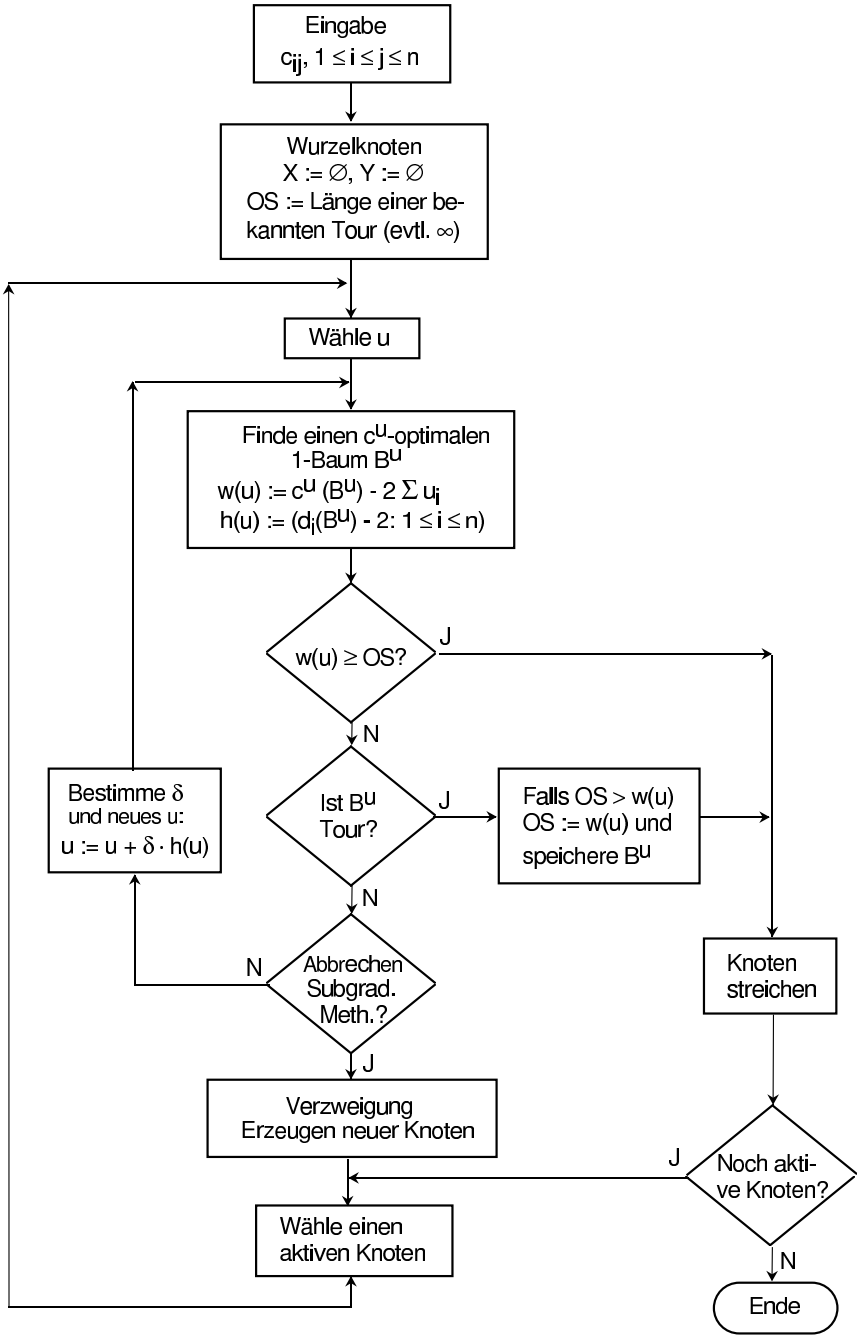


Abbildung 11.9: Flussdiagramm zum Ablauf des B & B

Folgende Punkte sind noch zu präzisieren:

(a) Wahl eines aktiven Knotens, des sog. aktuellen Knotens,

(b) Wahl eines Ausgangswertes u für den aktuellen Knoten,

(c) Abbruchkriterium für die Subgradientenmethode,

(d) Wahl der Schrittlänge δ

Grundsätzlich ist zu bemerken, dass a bis d auf Heuristiken und numerischen Experimenten beruhen, und somit die folgenden Angaben keinen mathematisch zwingenden Charakter haben. b, c und d beziehen sich auf den jeweils aktuellen Knoten. Es ist vorteilhaft, hier zwischen Wurzelknoten und anderen Knoten zu unterscheiden.

Zu a: Es wird jeweils derjenige aktive Knoten aktuell, der die kleinste untere Schranke unter allen aktiven Knoten aufweist.

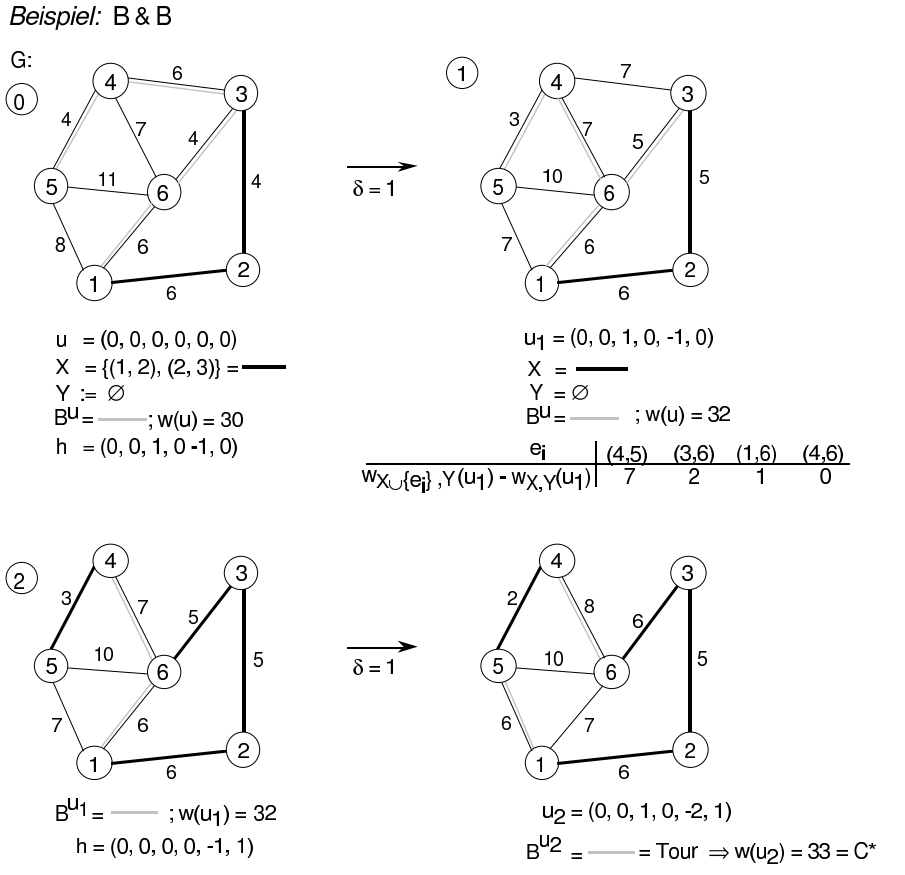
Zu b: Im Wurzelknoten: $u = 0$, sonst die Stelle u, bei welcher die höchste untere Schranke des Vorgängerknotens (aus dem der aktuelle Knoten verzweigt wurde) oder des Wurzelknotens berechnet wurde.

Zu c und d: Eine Anfangsschrittlänge δ_0 und eine minimale Schrittlänge $\underline{\delta}$ werden festgelegt. Falls während p aufeinander folgenden Interationen $w(u)$ nicht zunimmt [1], wird die Schrittlänge halbiert. Falls die Schrittlänge $\underline{\delta}$ unterschreitet, wird abgebrochen.

Wurzelknoten: $p = 4 + \lfloor \frac{n}{10} \rfloor \cdot \delta_0 = 2^i$, wobei $i \in \{0, 1, 2, \dots\}$ so gewählt ist, dass der erste Schritt einen möglichst grossen Zuwachs $w(0 + \delta_0 \cdot h(0)) - w(0)$ bringt. $\delta = \min\{1, \frac{\delta_0}{2^4}\}$.

Anderer Knoten: p wie oben, $\delta_0 = 2$ mal die zuletzt im Wurzelknoten benutzte Schrittlänge, $\underline{\delta} = \delta_0/2$.

[1] In der Subgradientenmethode muss $w(u)$ nicht monoton zunehmen, denn nicht jeder Subgradient zeigt in eine Zuwachsrichtung von $w(u)$.



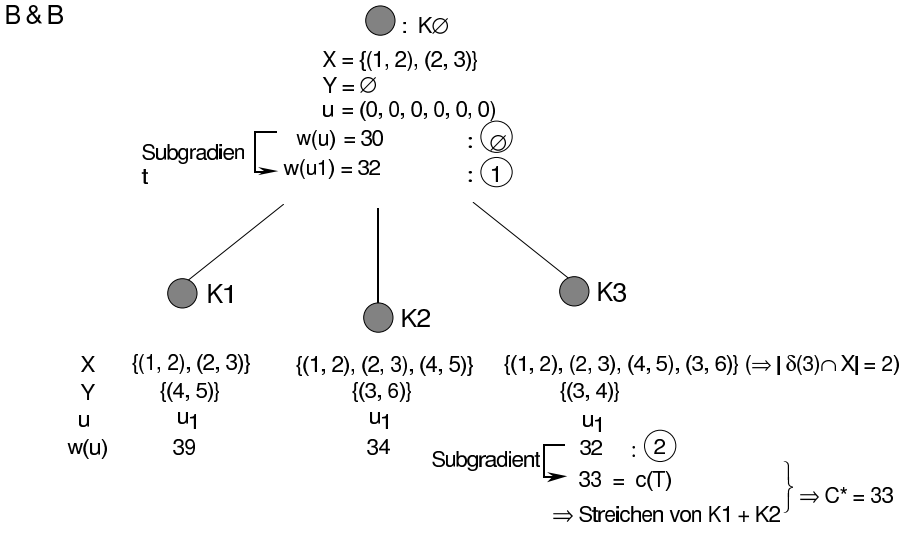


Abbildung 11.10: Beispiel zu B & B mit 1–Baum-Relaxation

Allgemeine Lehrbücher in Operations Research

Gal, Th. — Grundlagen des Operations Research
Springer Verlag, Berlin, 1987

Hillier F.S., Liebermann G.J. — Introduction to Operations Research
(Fourth Edition)
Holden–Day, Inc. Oakland, 1986

Nemhauser G.L. et al., Eds. — Handbooks in Operations Research
and Management Science
Vol. 1: Optimization;
Vol. 2: Stochastic Models
Elsevier Science Publishers
(North–Holland), 1989

Lehrbücher in Diskreter Optimierung

Ahuja R.K., Magnanti T.L., Orlin J.B. — Networks Flows Prentice Hall,
Englewood Cliffs, 1993

Evans J.R., Minieka E. — Optimization Algorithms for Networks
and Graphs
Marcel Dekker Inc., New York, 1992

Jungnickel Dieter — Graphen, Netzwerke und Algorithmen
BI Wissenschaftsverlag,
Mannheim, 1994

Nemhauser G.L., Wolsey L.A. — Integer and Combinatorial Optimization
John Wiley & Sons, New York, 1988

Papadimitriou Ch.H., Steiglitz K. — Combinatorial Optimization Algorithms
and Complexity
Prentice Hall, New Jersey, 1983

Syslo M., Deo N., Kowalik J.S. — Discrete Optimization Algorithms with
Pascal Programs
Prentice Hall, New Jersey, 1983

Ausgewählte vertiefende Bücher

Lawler E.L., Lenstra J.K.,
Rinnooy Kan A.H.G., Schmoys D.B. — The Traveling Salesman Problem
John Wiley and Sons,
New York, 1985

Lovasz L., Plummer M.D. — Matching Theory
North Holland Mathematics Studies 121
Annals of Discrete Mathematics (29)
North–Holland, Amsterdam, 1986

Schrijver A. — Theory of Linear and Integer
Programming
John Wiley & Sons, New York, 1986

Kapitel 12

Fragenkatalog

We will try to help you prepare the exams in discrete optimization (for the sake of effectiveness in writing them, we will use a mixture of German and English). What you should find in here is:

1. a list of what you are not supposed to know

2. a list of questions that you should be able to answer

WARNING! THIS LIST IS CERTAINLY NOT EXHAUSTIVE OTHER QUESTIONS MIGHT OCCUR DURING THE EXAMS, AND IT REMAINS YOUR RESPONSABILITY TO KNOW WHAT HAS BEEN COVERED DURING THE LECTURE AND THE EXERCISES. IN PARTICULAR NOBODY SHOULD IGNORE THAT THERE HAS BEEN SOME DEVIATION FROM THE BASIC SCRIPT.

What you are supposed to know is what is available in written form on our webpage with some exceptions we will indicate now:

Material you are allowed to skip for SS05-Lecture: Kapitel 4.3(Matching maximaler Kardinalität für allgemeine Graphen) Kapitel 5.3 (Matching maximalen Gewichts in allgemeinen Graphen) Kapitel 6(Das Chinese Postman Problem) Kapitel 7(Flüsse in Netzwerken) Kapitel 8(Kostenminimale Flüsse) Kapitel 11 (Branch and Bound Algorithmus).

Facts mentionned in the lecture: Lagenstaffelproblem, Preemptive Open-Shop Problem

Two things you should never say during an exam:

(a) I did not prepare that part

(b) I did not understand that point (... The bad alternative answer to the first question)

The admissible answer is: As I did not understand, I asked (to you, or to the assistants) and I got the following answer...

Comments on Chapter 1:
Sections 1.1 and 1.3:
There should be no questions on section 1.1 and 1.3, unless some deficiencies appear in the context of another question. You are supposed to know what a polyeder is, that a polytope (bounded polyeder) is the set of convex combination of its extreme points and following theorems: Sätze 1.3, 1.6, 1.7, 1.8.
If during the lecture we somewhere happened to consider the dual of an LP, you should be able to reconstruct it from the primal.
You have to understand the Simplex algorithm just as needed to understand the Gomory cuts (section 9.1)

Section 1.2:
You should read section 1.2 now and then, in particular at the end of your preparation (you can skip the *Rucksack Problem*). Finding that it is clear and that you recognize what we have been doing most of the time is an excellent sign about the state of your preparation.

Questions to Chapter 1:

1.1) Give an example of a problem that is not a LOCO problem (Answer: Quadratic assignment)

1.2) Is the *Shortest Path Problem* difficult (A: Yes in the general case, because it is known that the Hamiltonian path problem is difficult and ...)

1.3) What makes it difficult (A:negative cycles)

1.4) with ref. to 3) A:What are then good particular cases and the corresponding algorithms?

Chapter 2: Branchings

2.1) Was ist der Unterschied zwischen Branching und Arboreszenz?

2.2) Give an example of *Branching maximalen Gewichts* where the greedy algorithms find the wrong answer

2.3) Define a Branching as *Independent Set in the intersection of 2 matroids*

2.4) Explain the Branching algorithms

2.5) Justify the use of the *Schrumpfen* operation (i.e, state Sätze 2.15 (with proof), 2.16 and Korollar 2.17)

Chapter 3: Greedy algorithmus und Matroide

3.1) Give 2 examples of *Unabhängigheitsysteme* which are not matroids

3.2) Explain the greedy algorithm for searching an independent set of maximal weight in einem *Unabhängigheitsystem*

3.3) Define a matroid (Axioms ii) and ii'))

3.4) show that Kantenmengen von Wäldern liefern die unabhängige Menge eines Matroids

3.5) Why can we solve *Gerüst MINIMALEN Gewichts* with the greedy algorithm, that is meant for a MAXIMIZATION problem?

3.6) Define Matroids by means of an algorithmic property (Satz 3.27)

3.7) Define the matroid polyeder (includes definition of Rang)

3.8) Properties of the Rang (lemma 3.13)?

3.9) Gives two approaches for solving the *maximal weight independent set problem in a matroid*

3.10) Why is it important to show Theorem 3.20, that is that the matroid polyeder is integral (ie ganzzahlig)?

3.11) Show how to use the Kettenlemma 3.22 for proving Theorem 3.20

3.12) Give a general criteria for proving that a polytpope (ie bounded polyedron) is integral (Lemma 3.23)

3.13) How can we apply Frage 3.12) for the matroid polyeder

3.14) Explain the *maximal weight independent set problem in the intersection of 2 matroid*

3.15) Define the construction of the auxiliary directed graph $H(M_1, M_2, J)$, and state Th.3.35 (Vergrössender-Weg Theorem)

3.16) Explain the relevance of the set A in part 1 of Theorem 3.35

3.17) How does hypothesis 3 of lemma 3.37 (Delta-lemma) appear in the statement of Theorem 3.35

3.18) What are the 2 conclusions of lemma 3.37 (Delta-lemma) + idea of proof

3.19) State and prove Theorem 3.38 (the intersection of 2 matroid polyedra is integral)

3.20) On the basis of Th.3.38, the "maximal weight independent set problem in the intersection of 2 matroid" can be expressed as an LP. Write its dual, explain what a "Gewichtssplitterung" is and explain the relevance of the concept of "Zertifikat" (actually, what does a Zertifikat certifies?).

3.21) define the construction of the auxiliary directed graph $H(M_1, M_2, J, c_1, c_2)$, explain the algorithmus and state Theorem 3.42.

3.22) You are not supposed to reproduce the proofs of Lemmata 3.43-3.48. You are expected to know the statements of these lemmata, and to explain how these six results fit together for proving Th.3.42.

3.23) Are all matroids matrix-matroids?

3.24) Show that there at most 2^{2^n} matroids on a groundset having n elements

3.25) Define a circuit in a matroid , state the circuit axioms, and the relation to the H-axioms (ie, the same expressed for the complements of the circuits)

3.26) How do we derive a lower bound for the number of matroids on a groundset having n elements (ie, explain the construction of "Gute Familien" and their relevance)

3.27) What is the implication of this (superpolynomial) lower bound for discussing the complexity of matroid algorithm?
Answer: we cannot expect to have algorithms solving problems for all matroids and running in time $P(n)$ polynomial in the size n of the matroid groundset. The reason is that there is no way to encode all

matroids of the size n using chains of bits of length bounded by some polynomial $Q(n)$ (In fact there are $2^{Q(n)}$ such chains only, not enough to encode all matroids). Said differently, an encoding of all matroids require chains of length superpolynomial in n, so that already reading the input is not polynomial in n. As you are likely to observe in the litterature, it is assumed that we are given an oracle (ie some black-box which tells us if yes or no a subset of the ground set is independent) and by a polynomial algorithm or matroids one usually means an algorithm using this oracle number of time at most a polynomial in n.

3.28) Ein Matching ist was in Bezug auf Matroiden? A: für bipartite Graphen der Durchschnitt von zwei unitären Partitionsmatroiden

3.29) Zusammenhang Vergrössernder Wege in Matroiden und für Matchings

3.30) Ist die Branchingmenge ein Matroid? Wie beschreibe ich Branchings mit Matroiden?

3.31) Wie kann man Matroide als Polyeder darstellen? A: mit dem Inzidenzvektor jeder teilmenge als Zeile der Matrix A und $rang(A)$ als entsprechendem Eintrag in $\mathbf{b}$.

3.32) Ist dieses Matrixpolyeder ganzzahlig? Beweisen Sie es. A: Wichtigste Punkte der Antwort: Rang, Submodularität, Lemma 3.13, Ketten-Lemma, ...

Chapter 4: Matching max. cardinality

4.1) Relation between max. card. and *vergrössender Weg* (i.e Satz 4.3). Wann ist ein Matching maximaler Kardinalität?

4.2) Give an upper bound for the maximal cardinality, as well as an optimality criterion in the bipartite case (A: Überdeckung ...)

4.3) Explain the corresponding algorithm in the bipartite case

Chapter 5

5.1) Schreibe das ILP für da Problem *Matching maximalen Gewichts in bipartiten Graphen* und zeige die natürliche Ganzzahligkeit dieses LPs sowie die Unimodularität der Matrix A (Satz 5.2 + Kor 5.3).

5.2) Give an application of the Matching maximalen Gewichts in general graphs (A: Chinese postman ... again explain why there is no problem with minimisation/maximisation)

Questions on Chapter 9 (Cutting planes algorithms = Schnittebenenverfahren)

9.1) Explain the difficulties related to ILP's (Integer Linear Programs)((9.1)-(9.4)) and their linear relaxation ((9.5)-(9.7)) + Abbildungen 9.1-9.2

9.2) Definition 9.1 of a cutting plane and corresponding algorithm

9.3) Derivation of Gomory-cuts (s.195)

9.4) Formuliere das TSP als ILP-Problem

9.5) Schreibe und erkläre die Teiltoureliminationsungleichung (Gleichung (9.38))

9.6) Schnittebenen für TSP: Explain die Comb inequality

9.7) Schnittebenen für TSP: Explain die 2-matching inequality ((9.39) und Lemma 9.9)

9.8) Be ready to comment the example 9.8 on page 211ff.

Questions on Chapter 10 (Lagrangerelaxation)

10.1) Describe the general setting ((10.1)-(10.3)) and the concept of hard/easy restrictions

10.2) Define $LR(u)$, $w_{LR}(u)$, w_P, LD, w_{LD} (sorry but you have to memorize this specific notations) which (in)equalities hold or are possible?

10.3) What is a subgradient? the subgradient method? Concavity of w_{LR}

10.4) State corollary 10.9

10.5) State and prove Satz 10.15

10.6) Wann gilt $w_P = w_{LD}$?

10.7) Explain the 1-tree relaxation in the sense of 10.1), what is the "easy" problem, what is the subgradient