Vorlesung „Optimierung“

René Beier & Bodo Manthey

Universität des Saarlandes
Sommersemester 2006

1 Einleitung

Definition 1.1. *Ein* Optimierungsproblem Π *ist ein 4-Tupel* $(I, \mathrm{sol}, m, \mathrm{type})$ *mit folgenden Eigenschaften:*

1. I *ist die Menge der* Instanzen.
2. *Für eine Instanz* $X \in I$ *ist* $\mathrm{sol}(X)$ *die Menge der* zulässigen Lösungen.
3. *Für* $X \in I$ *und* $Y \in \mathrm{sol}(X)$ *ist* $m(X, Y)$ *der* Wert *der Lösung* Y*. Die Funktion* m *heißt* Zielfunktion.
4. $\mathrm{type} \in \{\min, \max\}$ *gibt an, ob* Π *ein Minimierungs- oder Maximierungsproblem ist.*

2 Lineare Optimierung

Definition 2.1 (LP). *Ein* lineares Optimierungsproblem (LP) *ist folgendes: Gegeben sind* $A = (a_{i,j})_{i\in[m],j\in[n]} \in \mathbb{Z}^{m\times n}$, $b = (b_1, \ldots, b_m)^{\mathrm{T}} \in \mathbb{Z}^m$ *und* $c = (c_1, \ldots, c_n)^{\mathrm{T}} \in \mathbb{Z}^n$ *sowie* $N \subseteq [n]$ *und* $M \subseteq [m]$.

Gesucht ist ein $x = (x_1, \ldots, x_n)^{\mathrm{T}} \in \mathbb{R}^n$, *das* $c^{\mathrm{T}}x$ *minimiert und*

$$\begin{aligned} \sum_{j=1}^{n} a_{i,j}x_j &= b_i \quad \text{für } i \in M, \\ \sum_{j=1}^{n} a_{i,j}x_j &\geq b_i \quad \text{für } i \in [m] \setminus M \text{ und} \\ x_j &\geq 0 \quad \text{für } j \in N \end{aligned} \tag{2.1}$$

erfüllt, falls so ein x *existiert.*

Die Menge F ist die Menge aller zulässigen Lösungen eines LP, d.h.

$$F = \{x \in \mathbb{R}^n \mid x \text{ erfüllt alle Bedingungen}\}.$$

Definition 2.2. *Ein LP heißt*

- unlösbar, *falls* $F = \emptyset$ *ist,*
- lösbar, *falls es ein* $x^\star \in F$ *mit* $c^{\mathrm{T}}x^\star = \inf\{c^{\mathrm{T}}x \mid x \in F\}$ *gibt (*$x^\star$ *heißt* optimale Lösung*) und*
- unbeschränkt, *falls* $F \neq \emptyset$ *ist und* $\inf\{c^{\mathrm{T}}x \mid x \in F\}$ *nicht existiert.*

Fakt 2.3. *Jedes LP ist entweder unlösbar oder lösbar oder unbeschränkt.*
Optimale Lösungen sind nicht unbedingt eindeutig bestimmt.

Definition 2.4 (ILP). *Ein* ganzzahliges LP (ILP) *ist definiert wie ein LP, außer dass nun ein* $x \in \mathbb{Z}^n$ *gesucht ist.*

Fakt 2.5. *ILP ist* NP*-vollständig. LP kann gelöst werden mit Laufzeit polynomiell in* m, n *und der Länge* L *der Bitrepräsentation der Koeffizienten von* A, b *und* c.

2.1 Anwendungen

Maximaler Fluss. Gegeben ist ein gerichteter Graph $G = (V, E)$, zwei Knoten $s, t \in V$ und eine Kantenkapazitätsfunktion $c : E \to \mathbb{N}$.

Das Ziel ist folgendes:

$$\begin{array}{lll} \text{Maximiere} & \displaystyle\sum_{v\in V:e=(s,v)\in E} f(e) & \\ \text{unter den Bedingungen} & f(e) \geq 0 & \text{und} \\ & f(e) \leq c(e) & \text{für alle Kanten } e \in E \text{ und} \\ & \displaystyle\sum_{u\in V:(u,v)\in E} f(u,v) = \sum_{w\in V:(v,w)\in E} f(v,w) & \text{für alle Knoten } v \in V \setminus \{s,t\}. \end{array}$$

Minimaler Schnitt. Die Instanzen sind wie beim Maximal-Fluss-Problem. Das Ziel ist es, eine Menge C von Kanten zu finden, so dass auf jedem Weg von s nach t mindestens eine Kante aus C liegt.

Wir führen Variablen y_e ($e \in E$) ein, wobei wir $y_e = 1$ als $e \in C$ und $y_e = 0$ als $e \notin C$ interpretieren. Das Problem kann folgendermaßen als ILP formuliert werden:

$$\begin{array}{lll} \text{Minimiere} & \sum_{e \in E} y_e c(e) & \\ \text{unter den Bedingungen} & \sum_{e \in p} y_e \geq 1 & \text{für jeden } s\text{-}t\text{-Pfad } p \text{ von } G \text{ und} \\ & y_e \in \{0,1\} & \text{für alle } e \in E. \end{array}$$

Die Anzahl der Bedingungen in diesem ILP ist gleich der Anzahl der s-t-Pfade in G. Dies sind im Allgemeinen unendlich viele. Wir können uns zwar auf einfache Pfade beschränken, d.h. Pfade, die keine Kante mehrfach besuchen, aber selbst dann bleiben im Allgemeinen noch exponentiell viele Bedingungen übrig.

Fakt 2.6. *Selbst manche LPs und ILPs mit einer exponentiellen oder unendlichen Anzahl an Bedingungen können effizient gelöst werden.*

Eine alternative Formulierung des Schnittproblems kommt mit einer linearen Anzahl an Bedingungen aus:

$$\begin{array}{lll} \text{Minimiere} & \sum_{e \in E} y_e c(e) & \\ \text{unter den Bedingungen} & y_u + y_e - y_v \geq 0 & \text{für alle Kanten } e = (u,v) \in E, \\ & y_e, y_v \in \{0,1\} & \text{für alle } e \in E \text{ und } v \in V, \\ & y_s = 0 & \text{und} \\ & y_t = 1. & \end{array}$$

Fakt 2.7. *Die Bedingungen $y_e \in \{0,1\}$ und $y_v \in \{0,1\}$ können durch die schwächeren Bedingungen $0 \leq y_e \leq 1$ und $0 \leq y_v \leq 1$ ersetzt werden. Das so entstandene LP besitzt eine ganzzahlige optimale Lösung.*

Fakt 2.8. *Der Wert des minimalen Schnitts eines Netzwerks ist gleich dem Wert eines maximalen Flusses auf dem gleichen Netzwerk.*

Matching in bipartiten Graphen. Gegeben ist ein bipartiter Graph $G = (A \cup B, E)$ mit Gewichten $w : E \to \mathbb{N}$. Das Ziel ist es, ein Matching maximalen Gewichts zu berechnen. Wir können das Problem wie folgt als ILP formulieren:

$$\begin{array}{lll} \text{Maximiere} & \sum_{e \in E} x_e w(e) & \\ \text{unter den Bedingungen} & \sum_{b \in B : e = \{a,b\} \in E} x_e \leq 1 & \text{für alle } a \in A, \\ & \sum_{a \in A : e = \{a,b\} \in E} x_e \leq 1 & \text{für alle } b \in B, \\ & x_e \in \{0,1\} & \text{für alle } e \in E. \end{array}$$

Fakt 2.9. *Die Bedingungen $x_e \in \{0,1\}$ können durch $0 \leq x_e \leq 1$ ersetzt werden, ohne den Wert einer optimalen Lösung zu verändern.*

2.2 Formen von linearen Optimierungsproblemen

Definition 2.10. *Ein LP ist in* Standardform*, falls es von der Form*

$$\begin{array}{ll} \textit{minimiere} & c^{\mathrm{T}}x \\ \textit{unter den Bedingungen} & Ax = b \quad \textit{und} \\ & x \geq 0 \end{array} \tag{2.2}$$

ist. Ein LP ist in kanonischer Form*, falls es von der Form*

$$\begin{array}{ll} \textit{minimiere} & c^{\mathrm{T}}x \\ \textit{unter der Bedingung} & Ax \geq b \end{array} \tag{2.3}$$

ist. Ein LP in der Form (2.1) ist in allgemeiner Form*.*

Mit folgenden Rechenregeln lassen sich LPs in verschiedenen Formen ineinander überführen:

R1. Maximiere $c^{\mathrm{T}}x \Leftrightarrow$ minimiere $(-c)^{\mathrm{T}}x$.

R2. $a_i x \leq b_i \Leftrightarrow -a_i x \geq -b_i$.

R3. $a_i x = b_i \Leftrightarrow a_i x \leq b_i$ und $a_i x \geq b_i$.

R4. $a_i x \leq b_i \Leftrightarrow a_i x + s_i = b_i$ und $s_i \geq 0$. Eine solche Variable s_i heißt *Schlupfvariable (slack variable).*

R5. $x \Leftrightarrow (x^+ - x^-)$ und $x^+ \geq 0$ und $x^- \geq 0$.

2.3 Basislösungen

In diesem Abschnitt betrachten wir LPs in Standardform (2.2). Es sei A eine $(m \times n)$-Matrix und $m < n$. Im Folgenden gehen wir von folgender Annahme aus.

Annahme 2.11. *A hat Rang m.*

Mit A_i ($i \in [n]$) bezeichnen wir die i-te Spalte der Matrix A.

Definition 2.12. *Sei $B = \langle k_1, \ldots, k_m \rangle$ mit paarweise verschiedenen $k_1, \ldots, k_m \in [n]$. B heißt* Basis*, falls die Matrix $A_B = (A_{k_1}, \ldots, A_{k_m})$ regulär ist.*

Wir betrachten B sowohl als Vektor der Länge m als auch als m-elementige Menge.

Definition 2.13. *Sei $B = \langle k_1, \ldots, k_m \rangle$ eine Basis. Die zugehörige* Basislösung (basic solution) *$x_B \in \mathbb{R}^n$ ist wie folgt definiert: Sei $\hat{x}_B = A_B^{-1} b \in \mathbb{R}^m$. Dann ist*

$$(x_B)_j = \begin{cases} (\hat{x}_B)_\ell & \textit{falls } j = k_\ell \in B \textit{ ist und} \\ 0 & \textit{falls } j \notin B \textit{ ist.} \end{cases}$$

Die Variablen $x_{k_1}, \ldots, x_{k_m}$ heißen Basisvariablen*.*

Es gilt $Ax_B = A_B \hat{x}_B = b$.

Definition 2.14. *Ist x_B eine Basislösung und $x_B \geq 0$, dann heißt x_B* zulässige Basislösung (basic feasible solution, BFS)*.*

Zulässige Basislösungen sind von zentraler Bedeutung für das Lösen von linearen Optimierungsproblemen. Einen ersten Hinweis darauf liefert das folgende Lemma.

Lemma 2.15. *Sei x_B eine BFS zur Basis B von $Ax = b$, $x \geq 0$. Dann gibt es ein $c \in \mathbb{Z}^n$, so dass x_B die eindeutige optimale Lösung des LP (2.2) ist.*

Annahme 2.16. $F = \{x \in \mathbb{R}^n \mid Ax = b, x \geq 0\} \neq \emptyset$.

Satz 2.17. *Unter den Annahmen 2.11 und 2.16 gibt es mindestens eine BFS.*

Lemma 2.18. *Sei x_B eine Basislösung. Dann ist $|(x_B)_j| \leq m!\alpha^{m-1}\beta$ für alle $j \in [n]$, wobei $\alpha = \max_{i\in[m],j\in[n]}\{a_{i,j}\}$ und $\beta = \max_{i\in[m]}\{|b_i|\}$ ist.*

Annahme 2.19. *Die Menge $G = \{c^{\mathrm{T}}x \mid x \in F\}$ ist nach unten beschränkt.*

Satz 2.20. *Unter den Annahmen 2.11, 2.16 und 2.19 ändert die Hinzunahme der Bedingung $x \leq M$ nichts an der optimalen Lösung des LP. Hierbei ist $M = (m+1)!\alpha^m\beta$, $\alpha = \max_{i,j}\{|a_{i,j}|, |c_j|\}$, $\beta = \max_i\{|b_i|, |z|\}$ und $z = \inf(G)$.*

Mit Satz 2.20 können wir im Folgenden immer annehmen, dass F beschränkt ist.

2.4 Geometrie linearer Optimierungsprobleme

Unterräume S des Vektorraums $\mathbb{R}^n$ können als *Kern* einer linearen Abbildung dargestellt werden: $S = \{x \in \mathbb{R}^n \mid Ax = 0\}$. Sei $\operatorname{rank}(A)$ der Rang von A. Dann ist $\dim(S) = n - \operatorname{rank}(A)$ die *Dimension* von S. *Affine Unterräume* S sind Verschiebungen von Unterräumen und können durch $S = \{x \in \mathbb{R}^n \mid Ax = b\}$ beschrieben werden. Es ist wieder $\dim(S) = n - \operatorname{rank}(A)$.

Die Dimension einer beliebigen Teilmenge $S \subseteq \mathbb{R}^n$ ist die kleinste Dimension eines affinen Unterraums von $\mathbb{R}^n$, der S enthält. Die Dimension eines Punktes ist 0, ein Streckenabschnitt hat Dimension 1. $k+1$ Punkte mit $k+1 \leq n$ haben höchstens die Dimension k. Die Dimension von F ist höchstens $n - m$.

Eine *Hyperebene* im $\mathbb{R}^n$ ist ein affiner Unterraum der Dimension $n-1$. Alternativ kann eine Hyperebene H auch in der Form $H = \{x \mid a^{\mathrm{T}}x = b\}$ für $a \in \mathbb{R}^n$, $a \neq 0$, und $b \in \mathbb{R}$ dargestellt werden. Jede Hyperebene definiert zwei *Halbräume* $\{x \mid a^{\mathrm{T}}x \geq b\}$ und $\{x \mid a^{\mathrm{T}}x \leq b\}$.

Seien $x_1, \ldots, x_k \in \mathbb{R}^n$. Eine *konvexe Kombination* dieser Punkte ist ein Punkt $x = \sum_{i=1}^k \lambda_i x_i$ mit $\sum_{i=1}^k \lambda_i = 1$ und $\lambda_i \geq 0$ für alle $i \in [k]$.

Eine Menge $S \subseteq \mathbb{R}^n$ heißt *konvex*, falls alle konvexen Kombinationen von Punkten aus S ebenfalls in S liegen.

Ist der Schnitt von endlich vielen Halbräumen beschränkt, dann heißt er *(konvexes) Polytop*. Wir beschränken uns im Folgenden auf Polytope, die eine Teilmenge von $\{x \in \mathbb{R}^d \mid x \geq 0\}$ sind.

Sei P ein Polytop und S ein Halbraum, der durch die Hyperebene H begrenzt wird. Sei $F = P \cap S$. Die Menge F heißt *Seitenfläche (face)* von P, falls $F \subseteq H$ ist. Eine nulldimensionale Seitenfläche ist ein *Knoten (vertex)*. Eine eindimensionale Seitenfläche heißt *Kante (edge)*. Eine $(d-1)$-dimensionale Seitenfläche heißt *Facette (facet)*. Abbildung 1 zeigt ein Beispiel.

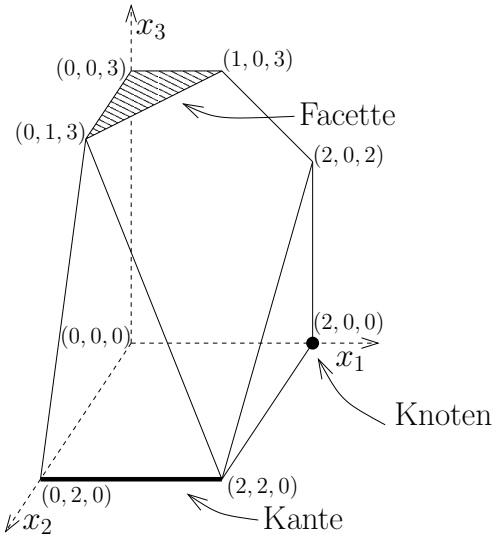

Abbildung 1: Ein 3-dimensionales Polytop zu den Ungleichungen $x_1 + x_2 + x_3 \leq 4$, $x_1 \leq 2$, $x_3 \leq 3$, $3x_2 + x_3 \leq 6$ und $x_1, x_2, x_3 \geq 0$.

Satz 2.21. *Jedes Polytop ist die konvexe Hülle seiner Knoten.*

Ist $V \subseteq \mathbb{R}^d$ eine endliche Menge, dann ist die konvexe Hülle von V ein Polytop P. Die Menge der Knoten von P ist eine Teilmenge von V.

Nach Satz 2.21 kann man Polytope unter verschiedenen Aspekten betrachten:

1. Als konvexe Hülle einer endlichen Punktemenge.

2. Als Schnitt endlich vieler Halbräume, falls dieser beschränkt ist.

3. Ein Polytop $P \subseteq \mathbb{R}^d$ kann durch $Ax = b$, $x \geq 0$ beschrieben werden (so dass die Annahmen 2.11, 2.16 und 2.19 erfüllt sind und $d = n - m$ gilt):

   Die Matrix A hat Rang m. Es gibt also m Spalten, die linear unabhängig sind. Wir nehmen an, dass die letzten m Spalten von A linear unabhängig sind. (Ansonsten müssten wir die Spalten permutieren, was die folgende Rechnung wesentlich komplizierter machen würde.) Wir multiplizieren A und b von links mit $A^{-1}_{\langle n-m+1,\ldots,n\rangle}$, und erhalten ein äquivalentes Gleichungssystem $A^{-1}_{\langle n-m+1,\ldots,n\rangle}Ax = A^{-1}_{\langle n-m+1,\ldots,n\rangle}b$. Die neue Matrix und den neuen Vektor nennen wir der Einfachheit halber wieder A bzw. b. Nun ist $A_{\langle n-m+1,\ldots,n\rangle}$ die $(m \times m)$-Einheitsmatrix. Damit ist $x_k = b_k - \sum_{j=1}^{n-m} a_{k,j}x_j$ für $k \in \{n-m+1, \ldots, n\}$. Also ist $Ax = b$, $x \geq 0$ äquivalent zu

$$\begin{aligned} b_k - \sum_{j=1}^{n-m} a_{k,j}x_j \geq 0 \quad &\text{für } k \in \{n-m+1, \ldots, n\} \text{ und} \\ x_j \geq 0 \quad &\text{für } j \in [n-m]. \end{aligned} \tag{2.4}$$

   Nun beschreibt (2.4) den Schnitt von n Halbräumen, der nach Satz 2.20 beschränkt ist.

Umgekehrt sei $P \subseteq \mathbb{R}^d$ ein d-dimensionales Polytop, das durch n Halbräume beschrieben ist. Wir können annehmen, dass d Halbräume von der Form $x_j \geq 0$ für $j \in [d]$ sind. (Ansonsten fügen wir diese Bedingungen hinzu. Das ändert nichts an P, da $P \subseteq \{x \in \mathbb{R}^d \mid x \geq 0\}$ vorausgesetzt ist.) Die anderen $n - d = m$ Halbräume sind von der Form

$$\sum_{j=1}^{n-m} h_{i,j} x_j + g_i \geq 0 \quad \text{für } i \in [m].$$

Durch Einführen von Schlupfvariablen $x_{n-m+1}, \ldots, x_n$ können wir nun die verbleibenden m Ungleichungen in der Form

$$\sum_{j=1}^{n-m} h_{i,j} x_j + g_i - x_{n-m+i} = 0 \quad \text{für } i \in [m].$$

schreiben. Wir setzen nun $H' = (h_{i,j})_{i \in [m], j \in [n-m]}$ und $g' = (g_1, \ldots, g_m)^{\mathrm{T}}$. Sei I_m die $(m \times m)$-Einheitsmatrix. Dann muss $(H', -I_m)x + g' = 0$ und $x \geq 0$ gelten. Mit $A = (H', -I_m)$ und $b = -g'$ ergeben sich die Bedingungen $Ax = b$ und $x \geq 0$.

Seien $g \in \mathbb{R}^n$ und $H \in \mathbb{R}^{n \times (n-m)}$ mit

$$g = (\underbrace{0, \ldots, 0}_{n-m}, g_1, \ldots, g_m)^{\mathrm{T}} \text{ und } H = \begin{pmatrix} I_{n-m} \\ H' \end{pmatrix}.$$

Dann entspricht ein Punkt $\hat{x} = (x_1, \ldots, x_{n-m})^{\mathrm{T}} \in P$ des Polytops einem Punkt $x = (x_1, \ldots, x_n)^{\mathrm{T}} \in F$ der Lösungsmenge mit

$$x = H\hat{x} + g.$$

Umgekehrt erhalten wir den zu $x \in F$ gehörenden Punkt $\hat{x} \in P$ durch Weglassen der letzten m Einträge von x. Die Abbildung $x \mapsto \hat{x}$ bildet F bijektiv auf P ab.

Wir können nun auch Knoten Kosten zuteilen: Die Kosten von $\hat{x}$ sind $c^{\mathrm{T}}x = c^{\mathrm{T}}(H\hat{x} + g) = d^{\mathrm{T}}\hat{x} + e$ mit $d^{\mathrm{T}} = c^{\mathrm{T}}H$ und $e = c^{\mathrm{T}}g$.

Satz 2.22. *Sei $P \subseteq \mathbb{R}^{n-m}$ ein Polytop und $F = \{x \mid Ax = b, x \geq 0\}$ die Menge der zulässige Lösungen des zugehörigen LP. Sei $\hat{x} \in P$. Dann sind die folgenden Aussagen äquivalent:*

1. *Der Punkt $\hat{x}$ ist ein Knoten von P.*
2. *Ist $\hat{x} = \lambda\hat{x}' + (1 - \lambda)\hat{x}''$ mit $\lambda \in (0, 1)$ und $\hat{x}', \hat{x}'' \in P$. Dann ist $\hat{x}' = \hat{x}'' = \hat{x}$.*
3. *Der zu $\hat{x}$ gehörende Vektor $x \in F$ ist eine BFS.*

Definition 2.23. *Eine BFS x_B heißt* degeneriert*, falls x_B mehr als $n - m$ Nullen enthält.*

Lemma 2.24. *Gehört eine BFS zu zwei verschiedenen Basen, dann ist sie degeneriert.*

Satz 2.25. *Unter den Annahmen 2.11, 2.16 und 2.19 gibt es eine BFS, die eine optimale Lösung des LP ist.*

Sind die Vektoren $x_1, \ldots, x_q$ BFS und optimale Lösungen, dann sind auch konvexe Kombinationen von $x_1, \ldots, x_q$ optimale Lösungen.

Wir können uns mit Satz 2.25 bei der Lösung von linearen Optimierungsproblemen auf BFS beschränken. Damit haben wir den unendlich großen Suchraum auf einen endlichen Suchraum eingeschränkt. Wir könnten nun alle höchstens $\binom{n}{m}$ BFS ausrechnen und ihre Kosten berechnen. Damit würden wir eine optimale BFS finden. Dieses Verfahren würde aber bereits für kleine Werte von m und n viel zu lange dauern.

3 Simplex-Algorithmus

Die Idee des Simplex-Algorithmus ist es, ausgehend von einer BFS eine neue BFS mit geringeren Kosten zu suchen. Dies entspricht dem Wandern auf Kanten des Polytops. Im Folgenden betrachten wir lineare Optimierungsprobleme in Standardform.

3.1 Finden einer besseren BFS

Sei $B = \langle k_1, \ldots, k_m \rangle$ eine Basis und $x_B \in \mathbb{R}^n$ die zugehörige BFS. Damit ist $b = A_B \hat{x}_B = A x_B$ und $x_B \geq 0$. Jede Spalte A_j ($j \in [n]$) kann als Linearkombination der Spalten von A_B geschrieben werden: $A_j = \sum_{\ell=1}^{m} A_{k_\ell} y_{\ell,j}$. Wir schreiben die Werte $y_{\ell,j}$ als Matrix $Y = (y_{i,j})_{i\in[m],j\in[n]} \in \mathbb{R}^{m\times n}$. Sei Y_j die j-te Spalte von Y. Dann ist $A_j = A_B Y_j$. Für $\theta \in \mathbb{R}$ ist

$$A_B(\hat{x}_B - \theta Y_j) + \theta A_j = b. \tag{3.1}$$

Wir wollen eine Spalte $p \in [n] \setminus B$ in die Basis bringen. Sei dazu $x^\theta \in \mathbb{R}^n$ mit

$$x^\theta_j = \begin{cases} (x_B)_{k_\ell} - \theta y_{\ell,p} & \text{für } j = k_\ell \in B, \\ \theta & \text{für } j = p \text{ und} \\ 0 & \text{sonst.} \end{cases}$$

Der Vektor x^θ ist eine zulässige Lösung, falls $\theta \geq 0$ und $(x_B)_{k_\ell} - \theta y_{\ell,p} \geq 0$ für alle $\ell \in [m]$ ist. (Wegen (3.1) ist $Ax^\theta = b$ für alle $\theta \in \mathbb{R}$ erfüllt.) Sei daher

$$\theta = \min\left\{ \frac{(x_B)_{k_\ell}}{y_{\ell,p}} \mid \ell \in [m], y_{\ell,p} > 0 \right\}.$$

Ist x_B degeneriert, $(x_B)_{k_q} = 0$ und $y_{q,p} > 0$ für ein $q \in [m]$, dann ist $\theta = 0$. Wir bleiben damit am gleichen Knoten, haben aber eine neue Basis $(B \setminus \{k_q\}) \cup \{p\} = \langle k_1, \ldots, k_{q-1}, p, k_{q+1}, \ldots, k_m \rangle$.

Ist $y_{\ell,p} \leq 0$ für alle $\ell \in [m]$, dann kann θ beliebig groß gewählt werden. Dann wäre der F unbeschränkt. Wir nehmen im Folgenden an, dass F beschränkt ist. Der Simplex-Algorithmus berücksichtigt auch den Fall, dass F und G unbeschränkt sind.

Satz 3.1. *Sei $p \in [n] \setminus B$, $\theta = \min_{\ell\in[m],y_{\ell,p}>0}\{\frac{(x_B)_{k_\ell}}{y_{\ell,p}}\}$ und $q \in [m]$, so dass $\theta = \frac{(x_B)_{k_q}}{y_{q,p}}$ ist. Dann ist der Vektor x^θ eine BFS zur Basis $(B \setminus \{k_q\}) \cup \{p\}$.*

Gibt es mehrere q mit $y_{q,p} > 0$ und $\frac{(x_B)_{k_q}}{y_{q,p}} = \theta$, dann ist x^θ degeniert.

Unser Ziel ist es, ausgehend von x_B eine BFS mit geringeren Kosten zu finden und uns so der optimalen Lösung zu nähern. Sei $c_B = (c_{k_1}, \ldots, c_{k_m})^{\mathrm{T}}$. Die Kosten von x_B sind $z_0 = c^{\mathrm{T}} x_B = c_B^{\mathrm{T}} \hat{x}_B$ und die Kosten von x^θ sind

$$c^{\mathrm{T}} x^\theta = z_0 + \theta \overbrace{\Big(c_p - \underbrace{c_B^{\mathrm{T}} Y_p}_{z_p}\Big)}^{\overline{c}_p}. \tag{3.2}$$

Spalte p in die Basis zu bringen, liefert damit eine bessere Lösung, falls $\overline{c}_p < 0$ ist. Wir setzen $z^{\mathrm{T}} = (z_1, \ldots, z_n)$. Damit ist $z^{\mathrm{T}} = c_B^{\mathrm{T}} A_B^{-1} A = c_B^{\mathrm{T}} Y$. Insbesondere ist $z_{k_\ell} = c_{k_\ell}$ und damit $\overline{c}_{k_\ell} = 0$ für alle $\ell \in [m]$.

Satz 3.2. *Ist $\overline{c}_j \geq 0$ für alle $j \in [n]$, dann ist x_B eine optimale Lösung.*

Algorithmus 3.1 Pivotieren.

T = pivot (T, z, s)

$T = (T[i,j])_{0 \le i \le m, 0 \le j \le n}$, $z \in [m]$, $s \in [n]$

1: **for** $i \leftarrow 0$ **to** m **do**
2: **if** $i \neq z$ **then**
3: $T[i,\cdot] \leftarrow T[i,\cdot] - \frac{T[i,s]}{T[z,s]} \cdot T[z,\cdot]$
4: $T[z,\cdot] \leftarrow \frac{1}{T[z,s]} \cdot T[z,\cdot]$

3.2 Tableaus

Um ausgehend von einer BFS eine bessere finden zu können, benötigen wir die Werte $y_{\ell,p}$ und $\overline{c}_p$. Wir schreiben zunächst die Kostenfunktion als neue Gleichung: $0 = c^{\mathrm{T}}x - d$. Damit sieht das LP jetzt so aus: Minimiere d unter den Bedingungen $Ax = b$, $c^{\mathrm{T}}x - d = 0$ und $x \geq 0$. Wir schreiben die Gleichungen kompakt als *Tableau*, wobei wir die Spalte für d weglassen:

$$T = \left[\begin{array}{c|c} 0 & c^{\mathrm{T}} \\ \hline b & A \end{array}\right].$$

Hierbei ist $T[0,0] = 0$, $T[i,j] = a_{i,j}$, $T[i,0] = b_i$ und $T[0,j] = c_j$ für $i \in [m]$ und $j \in [n]$.

Durch *Pivotieren* (Algorithmus 3.1) auf einen Eintrag (i,j) mit $T[i,j] \neq 0$ wird Spalte j zum i-ten Einheitsvektor. Eine Basislösung, die aber nicht unbedingt zulässig ist, kann mit *Gaußelimination* gefunden werden: Für $\ell = 1, \ldots, m$ suchen wir eine Position k_ℓ in Zeile ℓ mit $T[\ell, k_\ell] \neq 0$ und pivotieren auf (ℓ, k_ℓ). Gibt es keinen solchen Eintrag, dann ist $T[\ell,1] = \ldots = T[\ell,n] = 0$ und $\operatorname{rank}(A) < m$. Ist $T[\ell,0] = 0$, dann können wir Zeile ℓ streichen. Ist $T[\ell,0] \neq 0$, dann ist $F = \emptyset$, das LP ist unlösbar.

Auf diese Weise steht in Spalte k_ℓ der ℓ-te Einheitsvektor. Das Tableau ist

$$T = \left[\begin{array}{c|c} \cdot & \cdots \\ \hline \hat{x}_B & Y \end{array}\right].$$

(Auf die Punkte im Tableau gehen wir in Kürze ein.) Nach Konstruktion ist Y_B für $B = \langle k_1, \ldots, k_m \rangle$ die $(m \times m)$-Einheitsmatrix. Da Pivotieren dem Multiplizieren von links mit einer $(m \times m)$-Matrix entspricht, ist $Y_B = A_B^{-1} A_B$. Also ist $Y = A_B^{-1} A$. Damit enthält das Tableau die gewünschten y-Werte. Außerdem steht in der ersten Spalten $A_B^{-1} b = \hat{x}_B$.

Nehmen wir an, dass wir eine Basislösung zur Basis B und das dazugehörige Tableau T haben. Wir wollen nun eine Spalte $p \notin B$ in die Basis bringen, und Spalte k_q soll im Gegenzug die Basis verlassen. Durch Pivotieren auf Eintrag (q,p) des Tableaus (unter der Voraussetzung $T[q,p] \neq 0$) erhalten wir das Tableau zu der neuen Basis.

Was steht nun in Zeile 0 des Tableaus? Wir können uns vorstellen, dass Zeile 0 wie folgt entsteht: Zunächst ignorieren wir Zeile 0 des Tableaus beim Pivotieren, bis alle ℓ Pivotierungen vorgenommen sind. Dann subtrahieren wir das c_{k_ℓ}-fache der

Algorithmus 3.2 Verbessern einer zulässigen Basislösung.

improveBFS
$T = (T[i,j])_{0\le i\le m, 0\le j\le n}$, $k_1, \ldots, k_m$
1: choose any $p \in [n] \setminus \{k_1, \ldots, k_m\}$ with $T[0,p] < 0$
2: **if** no such p exists **then**
3: $\quad R \leftarrow$ solved; **return**
4: $\theta \leftarrow \min_{\ell\in[m]} \left\{ \frac{T[\ell,0]}{T[\ell,p]} \mid T[\ell,p] > 0 \right\}$
5: **if** $\theta = \infty$ **then**
6: $\quad R \leftarrow$ unbounded; **return**
7: choose q with $\frac{T[k_q,0]}{T[k_q,p]} = \theta$
8: $T \leftarrow$ pivot (T, q, p)
9: $k_q \leftarrow p$

Zeile ℓ von Zeile 0. Wegen $c_{k_\ell} = (c_B)_\ell$ erhalten wir damit für $j \in [n]$

$$T[0,j] = c_j - \sum_{\ell=1}^{m} (c_B)_\ell y_{\ell,j} = c_j - c_B^{\mathrm{T}} Y_j = c_j - z_j = \overline{c}_j.$$

Außerdem ist

$$T[0,0] = 0 - \sum_{\ell=1}^{m} (c_B)_\ell (\hat{x}_B)_\ell = -c_B^{\mathrm{T}} \hat{x}_B = -z_0.$$

Das Tableau T sieht damit wie folgt aus:

$$T = \left[\begin{array}{c|c} -z_0 & \overline{c}^{\mathrm{T}} \\ \hline \hat{x}_B & Y \end{array}\right].$$

Damit enthält das Tableau alle Informationen, die wir benötigen.

3.3 Simplex-Algorithmus (Phase 2)

Algorithmus 3.2 berechnet ausgehend von einer BFS eine neue BFS oder stellt fest, dass bereits das Optimum vorliegt oder das LP unbeschränkt ist. Wir haben zwar bislang unbeschränkte LPs ausgeschlossen, es ist aber keine Schwierigkeit, herauszufinden, ob ein LP unbeschränkt ist: Es genügt, zu testen, ob θ beliebig groß gewählt werden kann.

Das Verbessern der Basislösung, bis entweder eine optimale Lösung gefunden wurde oder gezeigt wurde, dass das LP unbeschränkt ist, bildet die Phase 2 des Simplex-Algorithmus (Algorithmus 3.3). (Phase 1 ist das Suchen einer ersten BFS.) Wie bei improveBFS gehen wir davon aus, dass $\langle k_1, \ldots, k_m \rangle$ eine Basis, die zugehörige Basislösung zulässig und T das zugehörige Tableau ist.

Damit haben wir fast alles zusammen, um lineare Optimierungsprobleme lösen zu können. Es bleiben lediglich noch zwei Frage offen:

1. Wie finden wir eine erste zulässige Basislösung? Diese Frage wird in Abschnitt 3.5 beantwortet.

Algorithmus 3.3 Lösen von linearen Optimierungsproblemen (Phase 2).

Phase2

$T = (T[i,j])_{0 \le i \le m, 0 \le j \le n}$, $k_1, \ldots, k_m$

1: **repeat**

2: improveBFS

3: **until** $R =$ solved **or** $R =$ unbounded

2. Terminiert der Algorithmus immer? Wie ist die Laufzeit? Damit in Zusammenhang steht die Frage: Wie sollen in den Zeilen 1 und 7 von improveBFS p und q gewählt werden?

3.4 Degeneriertheit und Zyklen

Es ist etwas Nichtdeterminismus in improveBFS: Welche Spalte p sollen wir in die Basis bringen, und wie lösen wir Konflikte, wenn $\frac{T[k_q,0]}{T[k_q,p]} = \theta$ von mehreren q erfüllt wird? Besitzt ein LP keine degenerierten BFS, dann terminiert Phase 2 immer: Mit jedem Aufruf von improveBFS finden wir eine BFS mit geringeren Kosten und es gibt nur endlich viele BFS. Wenn es degenerierte BFS gibt und wir p und q ungünstig wählen, dann kann es passieren, dass Phase 2 nicht terminiert. Da es nur endliche viele BFS gibt, kann Phase 2 nur dann nicht terminieren, wenn wir in einen Zyklus geraten. Eine Möglichkeit, Zyklen zu vermeiden und damit Terminierung zu garantieren, ist *Blands Regel*:

1. Wähle $p = \min\{j \mid \overline{c}_j < 0\}$ und
2. q, so dass $k_q = \min\{k_\ell \mid T[\ell,p] > 0 \text{ und } \frac{T[\ell,0]}{T[\ell,p]} = \theta\}$

Satz 3.3. *Werden p und q nach Blands Regel gewählt, dann terminiert Phase 2 nach einer endlichen Anzahl von Pivotierungen.*

3.5 Finden einer ersten BFS (Phase 1)

In vielen Fällen ergibt sich eine erste BFS direkt aus der Problemformulierung. Können wir nicht einfach eine erste BFS finden, dann gehen wir wie folgt vor: Ohne Einschränkung nehmen wir $b \ge 0$ an. Dann führen wir neue Variablen $s = (s_1, \ldots, s_m)^{\mathrm{T}}$ ein und lösen zunächst folgendes LP:

$$\begin{array}{lll} \text{minimiere} & \sigma = s_1 + s_2 + \ldots + s_m & \\ \text{unter den Bedingungen} & Ax + s = b \quad \text{und} & \\ & x, s \ge 0. & \end{array} \tag{3.3}$$

Lemma 3.4. *LP (3.3) ist lösbar und besitzt eine Lösung mit $\sigma = 0$ genau dann, wenn LP (2.2) eine Lösung besitzt.*

Eine BFS dieses LP ist $x = 0$ und $s = b$. Das Anfangstableau ist

$$T = \left[\begin{array}{c|c|c} 0 & \cdots 0 \cdots & \cdots 1 \cdots \\ \hline 0 & c^{\mathrm{T}} & \cdots 0 \cdots \\ \hline b & A & I_m \end{array}\right].$$

Algorithmus 3.4 Finden einer ersten BFS (Phase 1).

Phase1

$A \in \mathbb{Z}^{m \times n}$, $b \in \mathbb{Z}^m$, $c \in \mathbb{Z}^n$

build the phase 1 tableau $T \in \mathbb{Z}^{(m+2)\times(n+m+1)}$
for $\ell \leftarrow 1$ **to** m **do**
    $T \leftarrow$ pivot $(T, \ell, n+\ell)$
    $k_\ell \leftarrow n+\ell$
repeat
    improveBFS ▷ caution: line -1 contains cost information
until $R =$ solved
if $\sigma > 0$ **then**
    $R \leftarrow$ infeasible; **return**
$R \leftarrow \perp$
while $\{k_1, \ldots, k_m\} \cap \{n+1, \ldots, n+m\} \neq \emptyset$ **do**
    choose any ℓ with $k_\ell > n$
    choose any $p \in [n]$ with $T[\ell, p] \neq 0$
    if no such p exists **then**
        remove line ℓ
        $(k_\ell, \ldots, k_{m-1}) \leftarrow (k_{\ell+1}, \ldots, k_m)$
        $m \leftarrow m-1$
    else
        $T \leftarrow$ pivot (T, ℓ, p)
        $k_\ell \leftarrow p$
remove line -1 and columns $n+1, \ldots, n+m$ from T

Die oberste Zeile sei Zeile -1. Die zweitoberste Zeile (Zeile 0) ist eigentlich überflüssig, und wir brauchen sie zur Lösung des LP (3.3) nicht. Es ist aber nützlich, sie mitzuführen, da wir dann nach Lösen des Phase-1-LP bereits das Tableau für Phase 2 vorliegen haben.

Wir unterscheiden nun drei Fälle:

1. Wir haben eine Lösung mit $\sigma = 0$ gefunden, und keine der Hilfsvariablen $s_1, \ldots, s_m$ ist Teil der gefundenen optimalen BFS. Damit haben wir eine BFS für LP (2.2) gefunden, und wir können mit Phase 2 beginnen.

2. Die gefundene optimale Lösung hat Kosten $\sigma > 0$. Dann ist das LP (2.2) unlösbar.

3. Wir haben eine Lösung mit $\sigma = 0$ gefunden, aber einige Hilfsvariablen sind noch in der Basis. Nehmen wir an, dass s_i die ℓ-te Basisvariable ist. Wir suchen eine Position $j \in [n]$ mit $T[\ell, j] \neq 0$ und Pivotieren auf (ℓ, j). Es kann sein, dass $T[\ell, j] < 0$ oder $T[0, j] > 0$ ist, aber das ist hier egal.

   Ist $T[\ell, j] = 0$ ist für alle $j \in [n]$, dann haben wir eine Nullzeile in der ursprünglichen Matrix A erzeugt, was Annahme 2.11 widerspricht. Wir können Zeile ℓ einfach löschen.

Algorithmus 3.5 Der Simplex-Algorithmus.

Simplex
$A \in \mathbb{Z}^{m \times n}$, $b \in \mathbb{Z}^m$, $c \in \mathbb{Z}^n$
1: $R \leftarrow \perp$
2: Phase1
3: **if** $R =$ infeasible **then**
4: **return** infeasible
5: Phase2
6: **if** $R =$ unbounded **then**
7: **return** unbounded
8: construct an optimum BFS x from T
9: **return** x

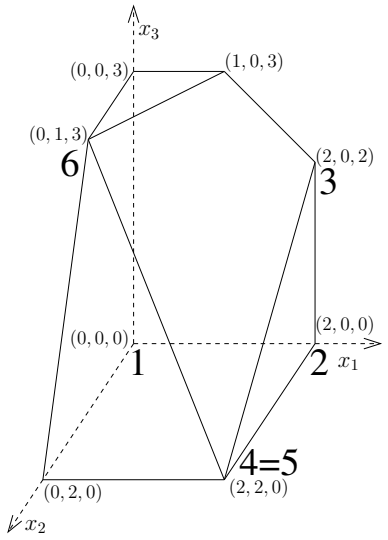

Abbildung 2: Der Weg zur optimalen Lösung verläuft entlang der Kanten.

Wir wiederholen den Vorgang solange, bis keine Hilfsvariable mehr in der Basis ist.

Phase 1 ist als Algorithmus 3.4 zusammengefasst.

3.6 Simplex-Algorithmus im Überblick

Der gesamte Simplex-Algorithmus ist als Algorithmus 3.5 zusammengefasst. Ist das zu lösende lineare Optimierungsproblem unlösbar (infeasible), dann wird dies in Phase 1 herausgefunden. Ob es unbeschränkt ist, findet Phase 2 heraus. Ansonsten wird eine optimale BFS berechnet.

3.7 Geometrische Betrachtung des Pivotierens

Der Weg von BFS zu BFS im Verlauf des Simplex-Algorithmus bildet ein Pfad entlang der Kanten des Polytops. Abbildung 2 zeigt einen möglichen Weg durch das

Polytop aus Abbildung 1. Das zugrunde liegende LP ist

$$
\begin{array}{lrcl}
\text{minimiere} & 2x_2 + x_4 + 5x_7 & & \\
\text{unter den Bedingungen} & x_1 + x_2 \;+ x_3 + x_4 & \leq & 4, \\
& x_1 + x_5 & \leq & 2, \\
& x_3 + x_6 & \leq & 3, \\
& 3x_2 + x_3 + x_7 & \leq & 6 \text{ und} \\
& x_1, x_2, x_3, x_4, x_5, x_6, x_7 & \geq & 0.
\end{array}
$$

Definition 3.5. *Zwei Knoten $\hat{x}$ und $\hat{x}'$ eines Polytops P heißen* adjazent*, falls die Strecke $[\hat{x}, \hat{x}']$ von $\hat{x}$ nach $\hat{x}'$ eine Kante von P ist.*

Zwei BFS x und x' eines LP heißen adjazent*, falls es zwei Basen B und B' mit $x = x_B$ und $x' = x_{B'}$ gibt, so dass $B' = (B \setminus \{k\}) \cup \{k'\}$ ist.*

Satz 3.6. *Sei $F = \{x \mid Ax = b, x \geq 0\} \subseteq \mathbb{R}^n$, $P \subseteq \mathbb{R}^{n-m}$ das dazugehörige Polytop und $\hat{x}, \hat{x}' \in P$ zwei verschiedene Knoten von P. Dann sind die folgenden Aussagen äquivalent:*

1. *$\hat{x}$ und $\hat{x}'$ sind adjazent.*
2. *Für jedes $y \in [\hat{x}, \hat{x}']$ mit $y = \lambda y' + (1 - \lambda)y''$ für $\lambda \in (0, 1)$ und $y', y'' \in P$ sind $y', y'' \in [\hat{x}, \hat{x}']$.*
3. *Die zu $\hat{x}$ und $\hat{x}'$ korrespondierenden Vektoren $x, x' \in F$ sind adjazente BFS.*

4 Dualität

Definition 4.1. *Das* duale Problem *zu einem LP in allgemeiner Form (siehe Definition 2.1),* primales Problem *genannt, ist folgendes LP:*

Primales Problem	*Duales Problem*	
$\min c^{\mathrm{T}}x$	$\max \pi^{\mathrm{T}}b$	
$a_i x = b_i$	π_i *unbeschränkt*	$i \in M$
$a_i x \geq b_i$	$\pi_i \geq 0$	$i \in [m] \setminus M$
$x_j \geq 0$	$\pi^{\mathrm{T}} A_j \leq c_j$	$j \in N$
x_j *unbeschränkt*	$\pi^{\mathrm{T}} A_j = c_j$	$j \in [n] \setminus N.$

(Hierbei ist a_i die i-te Zeile und A_j die j-te Spalte von A.)

Das duale Problem zu einem LP in Standardform ist (fast) in kanonischer Form:

$$\begin{array}{ll} \text{Maximiere} & \pi^{\mathrm{T}}b \\ \text{unter der Bedingung} & \pi^{\mathrm{T}}A \leq c^{\mathrm{T}}. \end{array}$$

Satz 4.2. *Das duale des dualen Problems ist das primale Problem.*

4.1 Dualitätssatz

Satz 4.3. *Ist das primale Problem lösbar, dann ist auch das duale Problem lösbar. Die Kosten der optimalen Lösungen sind gleich.*

Man sieht leicht, dass der Wert jeder zulässigen Lösung π für das duale Problem kleiner oder gleich dem Wert jeder zulässigen Lösung für das primale Problem ist (schwache Dualität). Wir beschränken uns hier auf den Fall, dass das primale Problem in Standardform vorliegt: $\pi^{\mathrm{T}}b = \pi^{\mathrm{T}}Ax \leq c^{\mathrm{T}}x$, wobei die Ungleichung aus $x \geq 0$ folgt.

Satz 4.4. *Gegeben sei ein Paar aus primalem und dualem LP. Dann trifft immer einer der folgenden Fälle zu:*

1. *Beide Probleme besitzen eine optimale Lösung.*
2. *Das primale Problem ist unbeschränkt, das duale unlösbar.*
3. *Das primale Problem ist unlösbar, das duale unbeschränkt.*
4. *Beide Probleme sind unlösbar.*

4.2 Komplementärer Schlupf

Je stärker eine Bedingung im primalen Problem ist, desto schwächer ist ihr Gegenstück im dualen Problem, und umgekehrt.

Satz 4.5 (Satz vom komplementären Schlupf). *Seien $x \in \mathbb{R}^n, \pi \in \mathbb{R}^m$ zulässige Lösungen des primalen bzw. dualen Problems. Die Vektoren x und π sind beide genau dann optimale Lösungen des primalen bzw. dualen Problems, wenn*

$$\begin{array}{lll} u_i = \pi_i(a_i x - b_i) & = 0 & \textit{für alle } i \in [m] \textit{ und} \\ v_j = (c_j - \pi^{\mathrm{T}} A_j)x_j & = 0 & \textit{für alle } j \in [n] \textit{ gilt.} \end{array} \tag{4.1}$$

4.3 Anwendungen

Kürzeste Wege. Gegeben sei ein gerichteter Graph $G = (V, E)$ mit Kantengewichten. Das Ziel ist es, einen kürzesten Weg von $s \in V$ nach $t \in V$ zu berechnen. Sei $V = \{v_1, \ldots, v_m\}$, $E = \{e_1, \ldots, e_n\}$, $s = v_1$ und $t = v_m$. Das Gewicht von Kante e_j sei $w_j \in \mathbb{N}$, und wir setzen $w = (w_1, \ldots, w_n)^{\mathrm{T}}$. Die zugehörige Inzidenzmatrix ist $A = (a_{i,j})_{i \in [m], j \in [n]}$ mit

$$a_{i,j} = \begin{cases} +1 & \text{falls } e_j \text{ eine ausgehende Kante von } v_i \text{ ist,} \\ -1 & \text{falls } e_j \text{ eine eingehende Kante von } v_i \text{ ist und} \\ 0 & \text{sonst.} \end{cases}$$

Die i-te Zeile von A bezeichnen wir wieder mit a_i, die j-te Spalte mit A_j.

Um das Kürzeste-Wege-Problem als LP zu formulieren, stellen wir uns einen Einheitsfluss $f = (f_1, \ldots, f_n)^{\mathrm{T}}$ vor, der durch das Netzwerk fließt.

An jedem Knoten $v_i \in V \setminus \{s, t\}$ muss $a_i f = 0$ gelten (Flusserhaltung). Ein Pfad von s nach t entspricht einem Fluss f mit $Af = (+1, 0, 0, \ldots, 0, -1)^{\mathrm{T}}$. Der Vektor f muss nicht unbedingt ganzzahlig sein, um die Gleichung zu erfüllen.

Wir betrachten folgendes LP:

$$\begin{array}{lll} \text{Minimiere} & w^{\mathrm{T}} f & \\ \text{unter den Bedingungen} & Af = (+1, 0, 0, \ldots, 0, -1)^{\mathrm{T}} \quad \text{und} & \\ & f \geq 0. & \end{array} \tag{4.2}$$

Fakt 4.6. *LP (4.2) besitzt eine ganzzahlige optimale Lösung, die einen kürzesten Pfad von s nach t repräsentiert.*

Das zu LP (4.2) duale Problem hat eine Variable π_i für jeden Knoten $v_i \in V$:

$$\begin{array}{ll} \text{Maximiere} & \pi_1 - \pi_m \\ \text{unter den Bedingungen} & \pi^{\mathrm{T}} A \leq w. \end{array}$$

(Die Variablen π_1 und π_m repräsentieren s bzw. t.) Die Ungleichungen können einfach als

$$\pi_i - \pi_{i'} \leq w_j \text{ für jede Kante } e_j = (v_i, v_{i'}) \in E$$

geschrieben werden.

Sei f ein Pfad von s nach t in G und π eine Lösung des dualen Problems. Dann sind f und π nach Satz 4.5 genau dann optimale Lösungen der beiden LPs, wenn $v_j = (w_j - (\pi_i - \pi_{i'}))f_j = 0$ für alle Kanten $e_j = (v_i, v_{i'})$ erfüllt ist. Also muss für jede Kante e_j des Weges $\pi_i - \pi_{i'} = w_j$ sein.

Maximaler Fluss und minimaler Schnitt. Wir modifizieren das LP des Flussproblems aus Abschnitt 2.1 etwas. Der Einfachheit halber gehen wir davon aus, dass s keine eingehenden und t keine ausgehenden Kanten besitzt.

$$\begin{array}{lrcll} \text{Minimiere} & \displaystyle\sum_{u \in V : e = (u,t) \in E} -f_e & & & \\ \text{unter den Bedingungen} & -f_e & \geq & -c(e) & \text{für alle Kanten } e \in E, \\ \displaystyle\sum_{u \in V : (u,v) \in E} f_{(u,v)} - \sum_{w \in V : (v,w) \in E} f_{(v,w)} & & = & 0 & \text{für alle Knoten } v \in V \setminus \{s, t\} \text{ und} \\ & f_e & \geq & 0 & \text{für alle Kanten } e \in E. \end{array}$$

Für jede Bedingung $-f_e \geq -c(e)$ führen wir eine Variable π_e ein, und für jede Bedingung $\sum_{u\in V:(u,v)\in E} f_{(u,v)} - \sum_{w\in V:(v,w)\in E} f_{(v,w)}$ eine Variable π_v ($v \in V \setminus \{s,t\}$). Es ergibt sich folgendes duale Problem:

$$\begin{array}{lll}
\text{Maximiere} & \sum_{e\in E} -c(e)\pi_e & \\
\text{unter den Bedingungen} & -\pi_e + \pi_v - \pi_u \leq & 0 \text{ für } e=(u,v)\in E \text{ mit } u\neq s,\ v\neq t,\\
& -\pi_e + \pi_v \leq & 0 \text{ für } e=(s,v)\in E,\\
& -\pi_e - \pi_u \leq & -1 \text{ für } e=(u,t)\in E \text{ und}\\
& \pi_e \geq & 0 \text{ für } e\in E.
\end{array}$$

Wir führen zwei künstliche Variablen π_s und π_t ein und erhalten folgendes LP:

$$\begin{array}{ll}
\text{Minimiere} & \sum_{e\in E} c(e)\pi_e\\
\text{unter den Bedingungen} & \pi_u + \pi_e - \pi_v \geq 0 \text{ für } e=(u,v)\in E,\\
& \pi_s = 0,\\
& \pi_t = 1 \text{ und}\\
& \pi_e \geq 0 \text{ für } e\in E.
\end{array}$$

Das Minimal-Schnitt-Problem ist also dual zum Maximal-Fluss-Problem (Fakt 2.8, Satz von Ford und Fulkerson). (Genau genommen müssen wir noch zeigen, dass das konstruierte LP optimale ganzzahlige Lösungen mit $0 \leq \pi_v \leq 1$ und $0 \leq \pi_e \leq 1$ für alle $v \in E$ und $e \in E$ besitzt.)

Matching in bipartiten Graphen. Bei der Suche nach Matchings maximalen Gewichts können wir uns auf perfekte Matchings beschränken. Des Weiteren gehen wir der Einfachheit halber davon aus, dass $G = (A \cup B, E)$ ein vollständiger bipartiter Graph ist. Das Gewicht der Kante $\{a,b\}$ ($a \in A$, $b \in B$) bezeichnen wir mit $w_{a,b}$. Wir ersetzen $w_{a,b}$ durch $\max_{a'\in A, b'\in B} w_{a',b'} - w_{a,b}$ und suchen nach einem perfekten Matching minimalen Gewichts. Es ergibt sich folgendes LP:

$$\begin{array}{ll}
\text{Minimiere} & \sum_{a\in A, b\in B} x_{a,b} w_{a,b}\\
\text{unter den Bedingungen} & \sum_{b\in B} x_{a,b} = 1 \quad \text{für alle } a\in A,\\
& \sum_{a\in A} x_{a,b} = 1 \quad \text{für alle } b\in B \text{ und}\\
& x_{a,b} \geq 0 \quad \text{für alle } a\in A, b\in B.
\end{array} \tag{4.3}$$

Das duale Problem zum Matching in bipartiten Graphen sieht wie folgt aus:

$$\begin{array}{ll}
\text{Maximiere} & \sum_{v\in A\cup B} \pi_v\\
\text{unter den Bedingungen} & \pi_a + \pi_b \leq w_{a,b} \quad \text{für } a\in A \text{ und } b\in B.
\end{array} \tag{4.4}$$

Wir weisen also jedem Knoten a einen Wert π_a zu, so dass $\pi_a + \pi_b \leq w_{a,b}$ ist. Sei $M \subseteq E$ ein beliebiges Matching. Dann ist

$$\sum_{v\in A\cup B} \pi_v = \sum_{e=\{a,b\}\in M} \pi_a + \pi_b \leq \sum_{\{a,b\}\in M} w_{a,b}, \tag{4.5}$$

was bereits aus dem Dualitätssatz folgt.

Sei nun $M \subseteq E$ ein Matching maximalen Gewichts und $(\pi_v)_{v \in A \cup B}$ eine optimale Lösung des dualen Problems. (Voraussetzung ist, dass es optimale ganzzahlige Lösungen des primalen Problems gibt.) Der Dualitätssatz besagt

$$\sum_{v \in A \cup B} \pi_v = \sum_{\{a,b\} \in M} w_{a,b}. \tag{4.6}$$

Nach dem Satz des komplementären Schlupfes ist ein Lösungspaar $(x_{a,b})_{a \in A, b \in B}$ und $(\pi_v)_{v \in A \cup B}$ genau dann optimal, wenn $0 = x_{a,b}(\pi_a + \pi_b - w_{a,b})$ für alle $a \in A$ und $b \in B$ ist. Das heißt, dass für alle Kanten $\{a, b\}$ eines optimalen Matchings $\pi_a + \pi_b = w_{a,b}$ ist, woraus ebenfalls Gleichung (4.6) folgt.

4.4 Informationen über das Duale im Tableau

Das Tableau zur optimalen Lösung enthält auch Informationen über eine optimale Lösung des dualen Problems.

Sei x_B eine optimale Lösung. Dann ist $\overline{c} \geq 0$, also $z \leq c$. Damit ist $c_B^{\mathrm{T}} Y = c_B^{\mathrm{T}} A_B^{-1} A \leq c^{\mathrm{T}}$. Also ist $\pi^{\mathrm{T}} = c_B^{\mathrm{T}} A_B^{-1}$ eine zulässige Lösung des dualen Problems. Die Kosten von π sind $\pi^{\mathrm{T}} b = c_B^{\mathrm{T}} A_B^{-1} b = c_B^{\mathrm{T}} \hat{x}_B = c^{\mathrm{T}} x$. Folglich ist π eine optimale Lösung des dualen LPs.

Nehmen wir an, dass $A_{\langle 1,\ldots,m \rangle}$ die Einheitsmatrix ist. (A enthält beispielsweise immer dann die Einheitsmatrix, wenn wir die Standardform durch Einführen von Schlupfvariablen gebildet haben.) Dann ist $Y_{\langle 1,\ldots,m \rangle} = A_B^{-1} A_{\langle 1,\ldots,m \rangle} = A_B^{-1}$.

In den Zeilen $1, \ldots, m$ ist A_j der j-te Einheitsvektor. Damit ist $z_j = c_B^{\mathrm{T}} A_B^{-1} A_j = \pi^{\mathrm{T}} A_j = \pi_j$ für $j \in [m]$. Also ist $\pi_j = c_j - \overline{c}_j$ für $j \in [m]$.

Allgemein können wir die zu einer Basis B gehörende duale Lösung definieren als $\pi_B^{\mathrm{T}} = c_B^{\mathrm{T}} A_B^{-1}$. Der Simplex-Algorithmus geht nun von einem Paar x_B und π_B aus, wobei x_B eine zulässige Lösung ist, und verbessert x_B solange, bis auch π_B zulässig ist. Der duale Simplex-Algorithmus geht genau anders herum vor: Er startet mit einer zulässigen Lösung des dualen Problems und verbessert diese solange, bis die zugehörige Lösung des primalen Problems zulässig ist.

Die Kosten von π_B sind $\pi_B^{\mathrm{T}} b = c_B^{\mathrm{T}} A_B^{-1} b = c^{\mathrm{T}} x_B$, also gleich den Kosten von x_B. Außerdem erfüllt das Paar x_B, π_B den Satz vom komplementären Schlupf: Wegen $A x_B = b$ ist $(\pi_B)_i (a_i x - b) = 0$. Ist j nicht in der Basis, dann ist $x_j = 0$ und damit $(c_j - \pi_B^{\mathrm{T}} A_j) x_j = 0$. Ist j in der Basis, dann ist $\overline{c}_j = 0$ und damit $c_j = z_j = c_B^{\mathrm{T}} A_B^{-1} A_j = \pi_B^{\mathrm{T}} A_j$.

5 Primal-Dual-Algorithmus

Wir wollen in diesem Abschnitt ein weiteres Verfahren zur Lösung von linearen Optimierungsproblemen untersuchen. Sei dazu wieder ein LP in Standardform gegeben, das wir im Folgenden P nennen:

$$\begin{array}{ll} \text{Minimiere} & c^{\mathrm{T}}x \\ \text{unter den Bedingungen} & Ax = b \geq 0 \quad \text{und} \\ & x \geq 0. \end{array} \tag{P}$$

Das duale Problem dazu nennen wir D:

$$\begin{array}{ll} \text{Maximiere} & \pi^{\mathrm{T}}b \\ \text{unter der Bedingung} & \pi^{\mathrm{T}}A \leq c. \end{array} \tag{D}$$

Da P in Standardform ist, ist x, π genau dann ein Paar optimaler Lösungen, wenn $(c_j - \pi^{\mathrm{T}}A_j)x_j = 0$ für alle $j \in [n]$ ist.

Sei π irgendeine zulässige Lösung von D. Dann müssen wir nur noch ein x dazu finden, so dass

$$x_j = 0 \text{ für alle } j \text{ mit } \pi^{\mathrm{T}}A_j < c_j \tag{5.1}$$

ist. Finden wir so ein x, dann haben wir optimale Lösungen von P und D gefunden. Finden wir so ein x nicht, dann verändern wir π und suchen wieder ein entsprechendes x.

Sei $J = \{j \in [n] \mid \pi^{\mathrm{T}}A = c_j\}$ die Menge der *zulässigen Spalten*. Dann suchen wir ein x, das die Bedingungen

$$\begin{array}{ll} Ax = b, & \\ x_j \geq 0 & \text{für } j \in J \text{ und} \\ x_j = 0 & \text{für } j \notin J \end{array}$$

erfüllt. Wir suchen so ein x mit Hilfe des sogenannten *eingeschränkten primalen Problems (restricted primal)*. Dazu führen wir Hilfsvariablen $x_1^a, \ldots, x_m^a$ ein:

$$\begin{array}{lll} \text{Minimiere} & \xi = \sum_{i=1}^{m} x_i^a & \\ \text{unter den Bedingungen} & \sum_{j \in J} a_{i,j}x_j + x_i^a = b_i & \text{für } i \in [m], \\ & x_j \geq 0 & \text{für } j \in J \text{ und} \\ & x_i^a \geq 0 & \text{für } i \in [m]. \end{array} \tag{RP}$$

Finden wir eine Lösung von RP mit $\xi = 0$, dann haben wir ein x gefunden, das (5.1) erfüllt. Somit haben wir optimale Lösungen von P und D gefunden.

Interessanter ist der Fall, dass die optimale Lösung von RP einen Wert $\xi > 0$ hat. Dazu untersuchen wir das zu RP duale Problem

$$\begin{array}{lll} \text{maximiere} & \pi^{\mathrm{T}}b & \\ \text{unter den Bedingungen} & \pi^{\mathrm{T}}A_j \leq 0 & \text{für } j \in J \text{ und} \\ & \pi_i \leq 1 & \text{für } i \in [m]. \end{array} \tag{DRP}$$

Sei $\overline{\pi}$ eine optimale Lösung von DRP mit Kosten $\overline{\pi}^{\mathrm{T}}b = \xi > 0$. Wir versuchen nun, aus der Lösung π von D und der optimalen Lösung $\overline{\pi}$ von DRP eine neue, bessere

Algorithmus 5.1 Primal-Dual-Algorithmus.

PrimalDual

1: $R \leftarrow \perp$
2: $\pi \leftarrow$ feasible solution of D
3: **while** $R = \perp$ **do**
4: $\quad J \leftarrow= \{j \mid \pi^{\mathrm{T}} A_j = c_j\}$
5: $\quad$ solve RP
6: $\quad$ **if** $\xi = 0$ **then**
7: $\quad\quad R \leftarrow$ solved
8: $\quad$ **else if** $\overline{\pi}^{\mathrm{T}} A_j \leq 0$ for all $j \notin J$ **then**
9: $\quad\quad R \leftarrow$ unbounded
10: $\quad$ **else**
11: $\quad\quad \pi \leftarrow \pi + \theta\overline{\pi}$

Lösung für D zu konstruieren: $\tilde{\pi} = \pi + \theta\overline{\pi}$. Wir müssen θ so wählen, dass $\tilde{\pi}$ eine zulässige Lösung von D mit $\tilde{\pi}^{\mathrm{T}} b > \pi^{\mathrm{T}} b$ ist. Es ist

$$\tilde{\pi}^{\mathrm{T}} b = \pi^{\mathrm{T}} b + \theta \underbrace{\overline{\pi}^{\mathrm{T}} b}_{=\xi > 0}.$$

Wir müssen daher $\theta > 0$ wählen, um eine bessere Lösung für D zu erhalten. Der Vektor $\tilde{\pi}$ ist eine zulässige Lösung, falls

$$\tilde{\pi}^{\mathrm{T}} A_j = \pi^{\mathrm{T}} A_j + \theta\overline{\pi}^{\mathrm{T}} A_j \leq c_j$$

ist für alle $j \in [n]$. Ist $\overline{\pi}^{\mathrm{T}} A_j \leq 0$, dann ist dies auf jeden Fall erfüllt. Ist $\overline{\pi}^{\mathrm{T}} A_j \leq 0$ für alle $j \in [n]$, dann kann θ beliebig groß gewählt werden. D ist dann unbeschränkt und P unlösbar. Für alle $j \in J$ ist $\overline{\pi}^{\mathrm{T}} A_j \leq 0$, da $\overline{\pi}$ eine zulässige Lösung von DRP ist. Damit ist $\tilde{\pi}$ zulässig, falls

$$\tilde{\pi}^{\mathrm{T}} A_j = \pi^{\mathrm{T}} A_j + \theta\overline{\pi}^{\mathrm{T}} A_j \leq c_j \text{ für } j \notin J \text{ und } \overline{\pi}^{\mathrm{T}} A_j > 0$$

ist. Da $\pi^{\mathrm{T}} A_j < c_j$ ist für alle $j \notin J$, ist

$$\theta = \min_{j \notin J, \overline{\pi}^{\mathrm{T}} A_j > 0} \left\{ \frac{c_j - \pi^{\mathrm{T}} A_j}{\overline{\pi}^{\mathrm{T}} A_j} \right\} > 0.$$

Wir haben also eine neue Lösung $\tilde{\pi}$ von D gefunden, die besser ist als π. Es ergibt sich Algorithmus 5.1.

Satz 5.1. *Der Primal-Dual-Algorithmus (mit einer geeigneten Regel zur Vermeidung von Zyklen) terminiert und löst P.*

5.1 Kürzeste Wege: Algorithmus von Dijkstra

Wir lassen im LP (4.2) die Zeile für den Zielknoten $t = v_m$ weg, da sie redundant ist. Es ergibt sich das folgende LP, wobei A entsprechend eingeschränkt ist:

$$\begin{array}{ll} \text{Minimiere} & w^{\mathrm{T}} f \\ \text{unter den Bedingungen} & Af = (+1, 0, 0, \ldots, 0)^{\mathrm{T}} \quad \text{und} \\ & f \geq 0. \end{array}$$

Das duale Problem ist

$$\begin{array}{ll} \text{maximiere} & \pi_1 \\ \text{unter den Bedingungen} & \pi_i - \pi_{i'} \leq w_j \quad \text{für jede Kante } e_j = (v_i, v_{i'}) \in E \text{ und} \\ & \pi_m = 0. \end{array}$$

(Wir müssen $\pi_m = 0$ festlegen, da wir die zu $v_m = t$ gehörende Zeile entfernt haben.) Die Menge der zulässigen Kanten ist $J = \{e_j = (v_i, v_{i'}) \in E \mid \pi_i - \pi_{i'} = w_j\}$. Damit ist RP

$$\begin{array}{ll} \text{Minimiere} & \xi = \sum_{i=1}^{m-1} x_i^a \\ \text{unter den Bedingungen} & Af + x^a = (+1, 0, 0, \ldots, 0)^{\mathrm{T}}, \\ & f \geq 0, \\ & f_j = 0 \quad \text{für alle } e_j \notin J \text{ und} \\ & x^a \geq 0. \end{array}$$

Gibt es einen Pfad $e_{j_1}, \ldots, e_{j_\ell}$ von s nach t in J, dann ist $f_{j_i} = 1$ für alle $i \in [\ell]$ und $f_j = 0$ sonst eine Lösung von RP mit Kosten 0. Gibt es keinen Pfad von s nach t in J, dann sei $e_{j_1}, \ldots, e_{j_\ell}$ ein beliebiger Pfad von s nach $v_i \in V \setminus \{t\}$. Wir setzen $x_i^a = 1$ und erhalten eine Lösung mit Kosten 1.

DRP ist

$$\begin{array}{ll} \text{maximiere} & \pi_1 \\ \text{unter den Bedingungen} & \pi_i - \pi_{i'} \leq 0 \quad \text{für alle } e_j = (v_i, v_{i'}) \in J, \\ & \pi \leq 1 \quad \text{und} \\ & \pi_m = 0. \end{array}$$

DRP ist einfach zu lösen: Gibt es einen Pfad von s nach t in (V, J), dann ist $\pi = 0$ eine optimale Lösung von DRP mit Kosten $\pi_1 = 0 = \xi$. Jeder Pfad von s nach t, der nur Kanten aus J verwendet, ist ein kürzester Weg.

Gibt es keinen Pfad von s nach t in (V, J), dann ist $\overline{\pi}$ mit

$$\overline{\pi}_i = \begin{cases} 1 & \text{falls } v_i \text{ in } (V, J) \text{ von } s \text{ aus erreichbar ist,} \\ 0 & \text{falls } t \text{ in } (V, J) \text{ von } v_i \text{ aus erreichbar ist und} \\ 1 & \text{sonst} \end{cases} \tag{5.2}$$

eine optimale Lösung von DRP mit Kosten 1. Wir setzen

$$\theta = \min_{e_j = (v_i, v_{i'}) \in E \setminus J, \overline{\pi}_i > \overline{\pi}_{i'}} \{w_j - \pi_i + \pi_{i'}\}$$

und aktualisieren π.

Der Primal-Dual-Algorithmus reduziert das Kürzeste-Wege-Problem auf ein einfaches Erreichbarkeitsproblem in Graphen. Er entspricht Dijkstras Algorithmus.

5.2 Maximaler Fluss: Algorithmus von Ford und Fulkerson

Wir schreiben das Maximal-Fluss-LP wieder etwas anders: Sei

$$d_v = \begin{cases} +1 & \text{falls } v = s, \\ -1 & \text{falls } v = t \text{ und} \\ 0 & \text{sonst.} \end{cases}$$

Dann modelliert das LP

$$\begin{array}{lll} \text{Maximiere} & w & \\ \text{unter den Bedingungen} & Af + dw \leq 0, & \\ & f \leq b & \text{und} \\ & -f \leq 0 & \end{array}$$

das Maximal-Fluss-Problem, wobei $b = (b_e)_{e \in E}$ der Kapazitätsvektor ist. Dieses LP ist D, es ist das duale eines LP in Standardform. (Das primale Problem wird hier nicht benötigt.) DRP ist folgendes LP:

$$\begin{array}{lll} \text{Maximiere} & w & \\ \text{unter den Bedingungen} & Af + dw \leq 0 & \text{für alle Zeilen,} \\ & f_e \leq 0 & \text{falls in D } f_e = b_e \text{ ist,} \\ & -f_e \leq 0 & \text{falls in D } f_e = 0 \text{ ist,} \\ & f \leq 1 & \text{und} \\ & v \leq 1. & \end{array}$$

Die Interpretation von DRP ist wie folgt: Finde einen Pfad von s nach t, der die Kanten von G folgendermaßen verwenden darf: Kanten ohne Fluss (mit $f_e = 0$) nur vorwärts, gesättigte Kanten (mit $f_e = b_e$) nur rückwärts und alle anderen Kanten in beide Richtungen. Wir haben damit auch das Flussproblem auf ein Erreichbarkeitsproblem (finde einen zunehmenden Weg) reduziert.

Gibt es keinen solchen Pfad, dann ist $v = 0$, und wir haben einen maximalen Fluss gefunden. Gibt es einen Pfad von s nach t, dann entspricht der nächste Schritt des Primal-Dual-Algorithmus dem erhöhen des Flusses entlang des Pfades. Der Fluss wird dabei maximal vergrößert, so dass keine Kapazitätsbedingung ($f_e \leq b_e$ und $f_e \geq 0$) verletzt wird.

Der Primal-Dual-Algorithmus angewendet auf das Maximal-Fluss-Problem ist entspricht also dem Algorithmus von Ford und Fulkerson.

5.3 Bemerkungen

Durch den Schritt vom primalen zum eingeschränkten primalen Problem vereinfachen wir die Kostenfunktion: Ein beliebiger Kostenvektor wird durch Einheitskosten ersetzt. Beim Kürzeste-Wege-Problem enthält RP keine Kantengewichte mehr.

Der Schritt vom dualen Problem zu dualen des eingeschränkten primalen Problems vereinfacht die Randbedingungen. Beim Flussproblem wird die Kapazitätsbedingung vereinfacht.

Die Hauptanwendung für den Primal-Dual-Algorithmus ist, dass RP und DRP für bestimmte Probleme einfach zu lösen sind, wodurch sich oft effiziente Algorithmen ergeben.

6 Matching in bipartiten Graphen

6.1 Ungewichtetes Matching

Satz 6.1 (Satz von Hall, Heiratssatz). *Sei $G = (A \cup B, E)$ ein bipartiter Graph mit $|A| = |B|$. Für eine Teilmenge $A' \subseteq A$ sei $\Gamma(A') \subseteq B$ die Menge der Knoten, die zu Knoten aus A' adjazent sind.*

Der Graph G besitzt genau dann ein perfektes Matching, wenn $|A'| \leq |\Gamma(A')|$ für alle Teilmenge $A' \subseteq A$ gilt.

6.2 Primal-Dual-Algorithmus für das Matching-Problem

Das primale Problem D und das duale Problem kennen wir bereits als (4.3) bzw. (4.4). Es sei $|A| = |B| = n$. Sei π eine Lösung von D. Wir setzen

$$J = \{\{a, b\} \mid a \in A, b \in B, \pi_a + \pi_b = w_{a,b}\}.$$

Um RP bilden zu können, führen wir Hilfsvariablen y_v ($v \in A \cup B$) ein:

$$\begin{aligned}
&\text{Minimiere} && \xi = \sum_{v \in A \cup B} y_v \\
&\text{unter den Bedingungen} && \sum_{b \in B} x_{a,b} + y_a = 1 \quad \text{für alle } a \in A, \\
& && \sum_{a \in A} x_{a,b} + y_b = 1 \quad \text{für alle } b \in B, \\
& && x_{a,b} = 0 \quad \text{für } \{a, b\} \notin J \text{ und} \\
& && x_{a,b} \geq 0 \quad \text{für alle } a \in A \text{ und } b \in B.
\end{aligned}$$

Es ist $y_a = 1 - \sum_{b \in B} x_{a,b}$ für $a \in A$ und $y_b = 1 - \sum_{a \in A} x_{a,b}$ für $b \in B$, also

$$\sum_{v \in A \cup B} y_v = 2n - 2 \cdot \sum_{a \in A, b \in B} x_{a,b}.$$

Da $x_{a,b} = 0$ ist für $\{a, b\} \notin J$, können wir RP wie folgt umschreiben:

$$\begin{aligned}
&\text{Maximiere} && \sum_{\{a,b\} \in J} x_{a,b} \\
&\text{unter den Bedingungen} && \sum_{b \in B : \{a,b\} \in J} x_{a,b} \leq 1 \quad \text{für alle } a \in A, \\
& && \sum_{a \in A : \{a,b\} \in J} x_{a,b} \leq 1 \quad \text{für alle } b \in B \text{ und} \\
& && x_{a,b} \geq 0 \quad \text{für alle } a \in A \text{ und } b \in B.
\end{aligned}$$

Eine Lösung mit $\xi = 0$ entspricht dann einer Lösung des modifizierten RP mit $\sum_{\{a,b\} \in J} x_{a,b} = n$. Wir haben damit ein Matching minimalen Gewichts genau dann gefunden, wenn der bipartite Graph $H = (A \cup B, J)$ ein perfektes Matching besitzt.

Der Graph $H = (A \cup B, J)$ heißt *Gleichheitsgraph* zu π. RP entspricht dem ungewichteten Matching-Problem im bipartiten Graphen H. Wir haben also das gewichtete Matching-Problem auf das ungewichtete Matching-Problem zurückgeführt.

Algorithmus 6.1 Vergrößern des Matchings.

mate = augment (mate, p, b)

1: **repeat**
2: $a \leftarrow p(b)$
3: $\text{mate}(b) \leftarrow a$
4: $b' \leftarrow \text{mate}(a)$
5: $\text{mate}(a) \leftarrow b$
6: $b \leftarrow b'$
7: **until** $b = \perp$

Lemma 6.2. *Sei μ das Gewicht eines perfekten Matchings minimalen Gewichts von G. Der Gleichheitsgraph $H = (A \cup B, J)$ enthält genau dann ein perfektes Matching, wenn $\sum_{v \in A \cup B} \pi_v = \mu$ ist. Jedes perfekte Matching $M \subseteq J$ von H ist ein Matching minimalen Gewichts von G.*

Lemma 6.2 beweist Fakt 2.9. Besitzt H kein perfektes Matching, dann müssen wir π anpassen. DRP ist

$$\begin{array}{lll} \text{maximiere} & \displaystyle\sum_{v \in A \cup B} \pi_v & \\ \text{unter den Bedingungen} & \pi_a + \pi_b \leq 0 & \text{für } \{a, b\} \in J \text{ und} \\ & \pi_v \leq 1 & \text{für alle } v \in A \cup B. \end{array}$$

Besitzt H ein perfektes Matching, dann sind wir fertig. Ansonsten gibt es eine Teilmenge $A' \subseteq A$, so dass $B' = \Gamma(A') \subseteq B$ mit $|B'| < |A'|$ ist (Heiratssatz). Dann ist $\pi_v = 0$ für $v \notin A' \cup B'$, $\pi_a = 1$ für $a \in A'$ und $\pi_b = -1$ für $b \in B'$ eine zulässige Lösung von DRP mit Kosten $|A'| - |B'| > 0$.

6.3 Der Ungarische Algorithmus

Der Ungarische Algorithmus für das gewichtete Matching-Problem in bipartiten Graphen ist im Wesentlichen eine effiziente Implementierung des Primal-Dual-Algorithmus. Der Algorithmus versucht, ein bestehendes Matching von H zu vergrößern. Gelingt dies nicht, dann werden die Werte π verändert.

Wir fangen mit einer zulässigen Lösung für D an: $\pi_a = 0$ für $a \in A$ und $\pi_b = \min_{a \in A} w_{a,b}$. Für jeden Knoten $v \in A \cup B$ bezeichnet $\text{mate}(v)$ den Partner im aktuellen Matching von H. Besitzt v keinen Partner, dann ist $\text{mate}(v) = \perp$.

Der Primal-Dual-Algorithmus für das Matching-Problem in bipartiten Graphen entspricht dem sogenannten Ungarischen Algorithmus (Algorithmus 6.2). Ähnlich wie beim ungewichteten Matching-Problem benötigen wir eine Prozedur augment (Algorithmus 6.1), die ein bestehendes Matching vergrößert.

Satz 6.3. *Der Ungarische Algorithmus (Algorithmus 6.2) berechnet ein perfektes Matching minimalen Gewichts in Zeit $O(n^3)$.*

Der Ungarische Algorithmus findet gerade eine ganzzahlige Lösung $(x_{a,b})_{a \in A, b \in B}$ und Zahlen $(\pi_v)_{v \in A \cup B}$ mit $\sum_{a \in A, b \in B} x_{a,b} w_{a,b} = \sum_{v \in A \cup B} \pi_v$.

Algorithmus 6.2 Ungarischer Algorithmus.

HungarianAlgorithm

vertex sets A and B with $|A| = |B| = n$; edge weights $w_{a,b}$ for $a \in A$, $b \in B$

1: **for all** $a \in A$ **do**
2: $\quad \pi_a \leftarrow 0$; $\text{mate}(a) = \perp$
3: **for all** $b \in B$ **do**
4: $\quad \pi_b \leftarrow \min_{a \in A}(w_{a,b})$; $\text{mate}(b) = \perp$
5: $N \leftarrow n$
6: **while** $N > 0$ **do**
7: $\quad A' \leftarrow \emptyset$
8: $\quad$ **for all** $b \in B$ **do**
9: $\quad\quad \delta(b) \leftarrow \infty$; $p(b) \leftarrow 0$
10: $\quad Q \leftarrow \{a \in A \mid \text{mate}(a) = \perp\}$
11: $\quad$ **while** $Q \neq \emptyset$ **do**
12: $\quad\quad$ remove a from Q
13: $\quad\quad A' \leftarrow A' \cup \{a\}$
14: $\quad\quad$ **for all** $b \in B$ **do**
15: $\quad\quad\quad$ **if** $\text{mate}(a) \neq b$ **then**
16: $\quad\quad\quad\quad$ **if** $w_{a,b} - (\pi_a + \pi_b) < \delta_b$ **then**
17: $\quad\quad\quad\quad\quad \delta_b \leftarrow w_{a,b} - (\pi_a + \pi_b)$
18: $\quad\quad\quad\quad\quad p(b) \leftarrow a$
19: $\quad\quad\quad\quad\quad$ **if** $\delta_b = 0$ **then**
20: $\quad\quad\quad\quad\quad\quad$ **if** $\text{mate}(b) = \perp$ **then**
21: $\quad\quad\quad\quad\quad\quad\quad \text{mate} \leftarrow \text{augment}(\text{mate}, p, b)$
22: $\quad\quad\quad\quad\quad\quad\quad N \leftarrow N - 1$; **goto** 6
23: $\quad\quad\quad\quad\quad\quad$ **else**
24: $\quad\quad\quad\quad\quad\quad\quad Q \leftarrow Q \cup \{\text{mate}(b)\}$
25: $\quad B' \leftarrow \{b \in B \mid \delta_b = 0\}$
26: $\quad \delta \leftarrow \min\{\delta_b \mid b \in B \setminus B'\}$
27: $\quad$ **for all** $a \in A'$ **do**
28: $\quad\quad \pi_a \leftarrow \pi_a + \delta$
29: $\quad$ **for all** $b \in B'$ **do**
30: $\quad\quad \pi_b \leftarrow \pi_b - \delta$
31: $\quad$ **for all** $b \in B \setminus B'$ **do**
32: $\quad\quad \delta_b \leftarrow \delta_b - \delta$
33: $\quad B^\star \leftarrow \{b \in B \setminus B' \mid \delta_b = 0\}$
34: $\quad$ **if** $\text{mate}(b) \neq \perp$ for all $b \in B^\star$ **then**
35: $\quad\quad Q \leftarrow Q \cup \{\text{mate}(b) \mid b \in B^\star\}$; **goto** 11
36: $\quad$ **else**
37: $\quad\quad$ choose $b \in B^\star$ with $\text{mate}(b) = \perp$
38: $\quad\quad \text{mate} \leftarrow \text{augment}(\text{mate}, p, b)$
39: $\quad\quad N \leftarrow N - 1$; **goto** 6

7 Wie schnell ist der Simplexalgorithmus

Die Laufzeit setzt sich aus zwei Faktoren zusammen:

- der Laufzeit für einen Simplexschritt (siehe Übung) und
- der Anzahl der Simplexschritte.

Empirisch hat sich gezeigt, dass man normalerweise weniger als $3n$ Iterationen benötigt. Sei im Folgenden

$B(A,b)$ die Anzahl der Basen von $Ax \leq b$ und
$BL(A,b)$ die Anzahl der Basislösungen (BFS) von $Ax \leq b$ (Ecken des Polyeders)

Basis und Basislösung für ein System $Ax \leq b$ definiert man analog zu den Definitionen 2.12 und 2.13. $B(A,b)$ und $BL(A,b)$ sind obere Schranken für die Anzahl der Simplexschritte, falls keine Degeneriertheiten auftreten.

Satz 7.1 (Upper Bound Theorem (McMullen, 1970)). *Sei $A \in \mathbb{R}^{m \times d}$. Dann gilt*

$$\begin{aligned} B(A,b) &\leq \binom{m}{d} \\ BL(A,b) &\leq \begin{cases} \frac{m}{m-s}\binom{m-s}{s}, & \text{falls } d = 2s, \\ 2\binom{m-s-1}{s}, & \text{falls } d = 2s+1. \end{cases} \end{aligned}$$

Man kann Beispiele konstruieren, bei denen diese Schranken auch angenommen werden. Auf diese Weise erhält man nur exponentiell große obere Schranken für die Anzahl der Iterationen.

Durchmesser von Polyedern

Seien x, y Knoten des Polyeders P. Dann ist $d(x,y)$ die minimale Anzahl von Kanten von P, die man entlang laufen muss, um von x nach y zu gelangen. Der Durchmesser für ein gegebendes Polyeder P ist

$$D(P) = \max_{x,y} d(x,y).$$

Weiterhin definieren wir $\Delta(d,m)$ als das Maximum von $D(P)$ über alle endlichen Polyeder im $\mathbb{R}^d$, die durch m Halbräume definiert werden können. Analog definieren wir $\Delta_u(d,m)$ für unbeschränkte Polyeder. $\Delta(d,m)$ ist also eine untere Schranken für die Anzahl der Simplexschritte im schlechtesten Fall. Die bisher nicht bewiesene *Hirsch Vermutung* (1957) besagt, dass

$$\Delta(d,m) \leq m - d \quad \textit{gilt.}$$

Man weiß, dass $\Delta_u(d,m) \geq m - d + \lfloor d/5 \rfloor$. Man weiß jedoch nicht einmal, ob $\Delta(d,m)$ polynomiell beschränkt ist.

Der Pfad, um von der Start-BFS zur optimalen BFS zu gelangen, wird durch die Pivotregel bestimmt. Für viele in der Praxis erfolgreich eingesetzten Pivotregeln

kann man eine Familie von LPs definieren, so dass der Simplex-Algorithmus eine exponentielle Anzahl von Iteration benötigt. Für die Regel von Bland ist der Klee-Minty-Würfel eine solche Familie von schlechten Instanzen. Der d-dimensionale Klee-Minty-Würfel ist wie folgt definiert:

$$\begin{aligned} 0 \leq \quad & x_1 \quad \leq 1 \\ \epsilon x_{i-1} \leq \quad & x_i \quad \leq 1 - \epsilon x_{i-1} \qquad \text{für } j = 2, \ldots, d \text{ und } \epsilon \in (0, 1/2) \end{aligned}$$

Eine der großen offenen Problem in der Optimierung ist die Frage, ob es eine Pivotregel gibt, für die der Simplexalgorithmus auch im schlechtesten Fall nur eine polynomielle Anzahl von Iterationen benötigt. Es ist nicht einmal klar, ob es eine solche geben kann (siehe Durchmesser $\Delta(d, m)$).

Randomisierte Pivotregeln Wählt man unter den möglichen Kandidaten für in die Basis ein- und austretenden Variablen eine zufällig aus, so kann man eine subexponentielle obere Schranke von $O(m^{\sqrt{d}})$ für die erwartete Anzahl der Iterationen beweisen.

Average-Case Analyse Bei der Average-Case-Analyse untersucht man die erwartete Laufzeit für eine zufällig gewählte Instanz vorgegebener Größe. Dabei wählt man beispielsweise die Zeilenvektoren der Matrix A zufällig mit einer d-dimensionalen Normalverteilung. Bei Anwendung der sogenannten Schatteneckenpivotregel kann man eine polynomielle obere Schranke für die erwartete Anzahl der Simplexschritte beweisen.

Implementierung des Simplexalgorithmuses Der in der Vorlesung vorgestellte Algorithmus zum Pivotieren ist ohne weitere Optimierung relativ einfach zu implementieren. Allerdings stellen sich in der Praxis folgende zwei Probleme:

1. **Dünn besetzte Matrizen:** Die in der Praxis auftretenden Instanzen von LPs entsprechen häufig Matrizen mit nur sehr wenigen Einträgen ungleich Null, d.h. fast alle Einträge von A sind Null. Will man eine konkurenzfähige Implementierung (Speicherplatz, Laufzeit), muss man diese Eigenschaft ausnutzen.

2. **Numerische Stabilität:** Aus Gründen der Effizienz benutzt man bei der Implementierung üblicherweise Floating-Point-Arithmetik, da diese direkt von der Hardware unterstützt wird. Dadurch ergeben sich jedoch Rundungsfehler bei den Zwischenergebnissen, d.h. die berechneten Zahlen sind nur Approximationen der eigentlichen Werte. Bei einer naiven Implementierung können sich diese Rundungsfehler schnell akkumulieren und zu falschen Ergebnissen führen (beispielsweise unzulässige und suboptimale Lösungen).

8 Komplexität der Linearen Programmierung

Wir sind interessiert an der Frage, ob man LPs in Polynomialzeit lösen kann. Bisher kann man eine definitive Antwort nur für ein bestimmtes Rechenmodell geben. Wir betrachten hier die folgenden zwei Rechenmodelle:

Algebraischen Rechenmodell: Arithmetische Grundoperationen können in konstanter Zeit durchgeführt werden, also unabhängig von der Größe der Operanden. Als Eingabegröße benutzt man die Anzahl der Zahlen in der Eingabe.

Bitkomplexität: Die Zeit für eine arithmetische Operation hängt von der Länge der Operanden ab. Man nimmt an, dass die Operanden aus $\mathbb{Z}$ oder $\mathbb{Q}$ kommen. Als Eingabegröße benutzt man die Länge der Eingabe in Bits.

Definition 8.1. *Sei $A \in \mathbb{Z}^{m\times n}$. Als Größe von A definieren wir die Länge der binären Kodierung von A:*

$$L(A) = \sum_{i,j} (\log_2 |a_i j| + 2).$$

Definition 8.2. *Sei $A \in \mathbb{Z}^{m\times n}, b \in \mathbb{Z}^m$. Die Frage, ob ein x existiert mit $Ax \leq b$, heißt* LP-Entscheidungsproblem.

Satz 8.3. *Das LP-Entscheidungsproblem kann in Zeit polynomiell in $L(Ab), m$ und n gelöst werden.*

Dieser Satz beweist Fakt 2.5.

Bemerkung 8.4. *Für das algebraische Rechenmodel ist eine solche Tatsache bisher unbekannt. In Übung 6, Aufgabe 3 haben wir bereits gesehen, wie man mit Hilfe eines Algorithmus, der das LP-Entscheidungproblem löst, auch das LP-Optimierungsproblem lösen kann.*

8.1 Die Ellipsoidmethode

Die Ellipsoidmethode löst das LP-Entscheidungsproblem und liefert, falls existent, ein x mit $Ax \leq b$. Die Ellipsoidmethode kann man als Verallgemeinerung der binären Suche verstehen. Wir machen zunächst folgende Annahmen, die später gerechtfertigt werden. Falls $P := \{x \in \mathbb{R}^n \mid Ax \leq b\} \neq \emptyset$, dann hat P ein Volumen der Größe mindestens v (v ist abhängig von L(Ab), also der Länge der binären Kodierung der Eingabe). Wir nehmen weiterhin an, dass dieses Mindestvolumen von P innerhalb eines n-dimensionalen Würfels

$$W_n = \{x \in \mathbb{R}^n \ : \ |x_i| \leq U, \text{ für alle } i = 1, \ldots, n\}$$

liegt, wobei U von $L(Ab)$ abhängig ist.

Zum besseren Verständnis betrachten wir zunächst den eindimensionalen Fall, also $n = 1$.

Ziel: Finde $x \in P$ oder stelle fest, dass $P = \emptyset$ ist.
Entweder ist P leer oder $P \cap [-U, U]$ ist ein Intervall mit Länge mindestens v. Für ein gegebenes $x \in \mathbb{R}$ können wir entscheiden, ob $x \in P$. Falls $x \notin P$, dann ist P entweder vollständig links von x oder vollständig rechts von x, da P ein Intervall ist. Bei der folgenden binären Suche verwenden wir ein Suchintervall I mit der Invariante $(P \cap [-U, U]) \subset I$. Das Länge des Suchintervall wird in jeder Iteration halbiert. Wenn das Suchintervall kleiner als v ist, muss wegen unserer Invariante $P = \emptyset$ gelten.

Binäre Suche:

1. Stare mit Suchintervall $I = [-U, U]$
2. Falls Länge von I kleiner als $v \Rightarrow$ Output $P = \emptyset$ (FERTIG).
3. Falls Mittelpunkt z von I zulässig $\Rightarrow$ Output z (FERTIG).
4. Ersetze I durch linke bzw. rechte Hälfte von I, entsprechend der von z nicht erfüllten Bedingung. Gehe zu Punkt 2.

Wie viele Iterationen benötigen wir? Wir stoppen spätestens, wenn das Suchintervall kleiner als v ist, wenn also $2^{-k}2U < v$ gilt. Es werden somit maximal $2 + \log \frac{U}{v}$ Iterationen ausgeführt.

Wir kann man diese Binärsuche auf höhere Dimensionen verallgemeinern? Wir betrachten als nächstes den 2-dimensionalen Fall, also $n = 2$.

Ziel: Finde $x \in P$ oder stelle fest, dass $P = \emptyset$ ist.
Entweder ist P leer oder $P \cap W_2$ ist ein Polygon mit Flächeninhalt mindestens v. Für ein gegebenes $x \in \mathbb{R}$ können wir entscheiden, ob $x \in P$ ist. Falls $x \notin P$, dann können wir eine Halbebene mit x auf dem Rand finden, die keinen Punkt aus P enthält. Statt Suchintervalle (wie im 1-dimensionalen Fall) benutzen wir Ellipsen. Die Invariante bleibt gleich, nämlich dass die Ellipsen P enthalten. Wir konstruieren also eine Folge $\mathcal{E}_0, \ldots, \mathcal{E}_k$ von immer kleiner werdenden Ellipsen, die jeweils den Teil von P enthalten, der auch im Quadrat mit Seitenlänge U enthalten ist. Wenn P beschränkt ist, dann liegt P vollständig in den konstruierten Ellipsen.

Algorithus:

1. Die Anfangsellipse $\mathcal{E}_0$ ist ein Kreis mit Radius $\sqrt{2}U$ und Zentrum $(0, 0)$.
2. Falls die Fläche von $\mathcal{E}_i$ kleiner als v ist $\Rightarrow$ Output $P = \emptyset$ (FERTIG)
3. Falls Zentrum z von $\mathcal{E}_i$ zulässig ist $\Rightarrow$ Output z (FERTIG).
4. Finde eine Halbebene H mit z auf dem Rand, die $P \cap W_2$ enthält. Definiere $\frac{1}{2}\mathcal{E}_i$ als den Schnitt von H und $\mathcal{E}_i$. Also gilt $(P \cap W_2) \subset \frac{1}{2}\mathcal{E}_i$. ($\frac{1}{2}\mathcal{E}_i$ hat den halben Flächeninhalt von $\mathcal{E}_i$, jedoch ist $\frac{1}{2}\mathcal{E}_i$ keine Ellipse.) Setze $\mathcal{E}_{i+1}$ zu kleinster Ellipse, die $\frac{1}{2}\mathcal{E}_i$ vollständig enthält. Gehe zu Punkt 2.

Definition 8.5. *Eine symmetrische Matrix $D \in \mathbb{R}^{n\times n}$ heißt* positiv definit, *falls $x^T Dx > 0$ für alle $x \in \mathbb{R}^n \backslash \{0\}$.*

Bemerkung 8.6. *D hat nur reale positive Eigenwerte und ist invertierbar.*

Definition 8.7. *Sei D eine $n \times n$ symmetrische und positiv definite Matrix und $z \in \mathbb{R}^n$. Eine Menge E von Vektoren aus $\mathbb{R}^n$ von der Form*

$$E = E(z, D) = \{x \in \mathbb{R}^n \mid (x - z)^T D^{-1}(x - z) \leq 1\}$$

heißt Ellipsoid mit Zentrum z.

Satz 8.8. *Sei $E = E(z, D)$ ein Ellipsoid in $\mathbb{R}^n$ und sei $a \in \mathbb{R}^n \backslash \{0\}$. Sei $H = \{x \in \mathbb{R}^n \mid a^T x \geq a^T z\}$ und sei*

$$\bar{z} = z + \frac{1}{n+1} \frac{Da}{\sqrt{a^T Da}} \quad \text{und}$$

$$\bar{D} = \frac{n^2}{n^2 - 1} \left(D - \frac{2}{n+2} \frac{Daa^T D}{a^T Da} \right).$$

Die Matrix $\bar{D}$ ist symmetrisch und positiv definit und $E' = E(\bar{z}, \bar{D})$ somit ein Ellipsoid. Weiterhin gilt

- *$E \cap H \subset E'$ und*
- *$Vol(E') < e^{-\frac{1}{2(n+1)}} Vol(E)$.*

Wie groß muss das Anfangsellipsoid sein?

Lemma 8.9. *Sei $A \in \mathbb{Z}^{n\times n}$ invertierbar und $b \in \mathbb{Z}^m$, $L = L(A|b), L' = L + n \log n$. Dann kann jede Komponente von $x = A^{-1}b$ durch*

$$x_j = \frac{D_j}{D}$$

dargestellt werden, wobei $|D_j| \leq 2^{L'}$ und $|D| \leq 2^{L'}$ gilt.

Für den Fall, dass P nicht leer, kann man zeigen, dass es ein $x \in P$ gibt, das auch im n-dimensionalen Würfel W_n mit Seitenlänge $2^{L'+1}$ liegt. Für das Anfangsellipsoid $\mathcal{E}_0$ wählt man die kleinste n-dimensionale Kugel, die W_n vollständig enthält (Radius: $\sqrt{n}2^{L'}$).

Wie kann man sicherstellen, dass P ein Mindestvolumen v hat?

Hier besteht die Idee darin, strikte Ungleichungen zu benutzen, wir betrachten also $Ax < b$ statt $Ax \leq b$.

Lemma 8.10 (Localization Lemma). *Falls $\{x \in \mathbb{R}^n \mid Ax < b\} \neq \emptyset$, dann ist das Volumen von $\{x \in \mathbb{R}^n \mid Ax < b, |x_j| \leq 2^{L'}\}$ mindestens $2^{-(d+1)L'}$.*

Natürlich können wir nicht einfach $Ax < b$ statt $Ax \leq b$ benutzen, da hierdurch Lösungen verloren gehen könnten. Wir können jedoch gleichzeitig die rechte Seite ein wenig vergrößern, so dass die Lösbarkeit der Systeme korreliert ist.

Lemma 8.11 (Perturbation-Lemma). *Sei $A \in \mathbb{Z}^{m\times n}, b \in \mathbb{Z}^m$. Dann ist*

$$Ax \leq b \quad \text{genau dann lösbar, wenn } Ax < b + \epsilon(1, \ldots, 1)^T \quad \text{lösbar ist,}$$

wobei $\epsilon = 2^{-L'}$ ist.

Wie viele Iterationen werden maximal ausgeführt?

Das Volumen des Ellipsoids schrumpft mindestens um den Faktor $f = e^{-\frac{1}{2(n+1)}} < 1$. Sei V das Volumen von $\mathcal{E}_0$ und v die untere Schranke für das Volumen von P, falls P nichtleer ist. Dann gilt für die maximale Anzahl von Iterationen ℓ:

$$\begin{aligned} f^\ell V &< v \\ \Leftrightarrow f^\ell &< \frac{v}{V} \\ \Leftrightarrow \ell \ln f &< \ln \frac{v}{V} \\ \Leftrightarrow \ell \frac{-1}{2(n+1)} &< -\ln \frac{V}{v} \quad (\text{da } \ln(f) = -1/(2(n+1))) \\ \Leftrightarrow \ell &> 2(n+1) \ln \frac{V}{v} \end{aligned}$$

Durch Einsetzen entsprechender Schranken für V und v erhält man $\ell \in O(n^2 L' + n^3)$ als obere Schranke für die Anzahl von Iterationen.

Aufwand pro Iteration Die Anzahl der arithmetischen Operationen pro Iteration ist $O((n+m)n)$. Allerdings untersuchen wir die Komplexiät unter dem Modell der Bitkomplexität. Durch die Multiplikationen können die Operanden sehr groß werden. Man kann jedoch folgendes Lemma beweisen:

Lemma 8.12. *Es genügt, Operationen auf $O(n^4 L')$ Nachkommastellen genau auszuführen.*

8.2 LPs mit exponentiell vielen Nebenbedingungen

Betrachte das LP

$$\max\{c^T x \mid Ax \leq b, A \in \mathbb{Z}^{m \times n}, b \in \mathbb{Z}^n\}$$

Durch eine leicht abgewandelte Analyse kann man zeigen, dass die Laufzeit polynomiell beschränkt ist in n und $\log U$, wobei U der größte Absolutbetrag über alle Einträge von A und b ist. Die Schranke ist also unabhängig von m, der Anzahl der Nebenbedingungen. Die Ellipsoidmethode kann solche LPs in polynomieller Zeit lösen, falls das folgende Separationsproblem in polynomieller Zeit gelöst werden kann.

Definition 8.13 (Separationsproblem). *Sei $P \subset \mathbb{R}^n$ ein Polyeder und $x \in \mathbb{R}^n$. Das Separationsproblem besteht darin entweder*

- *festzustellen, dass $x \in P$ oder*
- *einen Vektor d zu finden, so dass $d^{\mathrm{T}} x < d^{\mathrm{T}} y$ für alle $y \in P$ gilt.*

9 Ganzzahlige Lineare Programmierung

ILPs (Integer Linear Programming) sind LPs, bei denen man für den Lösungsvektor x zusätzlich fordert, dass er nur ganzzahlige Werte annehmen darf, also $x \in \mathbb{Z}^n$.

Bei *gemischt linearen Programmen* fordert man die Ganzzahligkeit nur für einen Teil der Variablen.

Beispiel 9.1.

$$\begin{aligned} \min c^T + d^T y & \\ Ax + by &= b \\ x, y &\geq 0 \\ x \text{ ganzzahlig} & \end{aligned}$$

Sei $A \in \mathbb{Z}^{m,n}, b \in \mathbb{Z}^m$. Das ILP-Entscheidungsproblem besteht darin zu entscheiden, ob ein $x \in \mathbb{Z}^n$ existiert, so dass $Ax \leq b$ gilt.

Satz 9.2. *Das ILP-Entscheidungsproblem ist* NP*-vollständig.*

Der Beweis benutzt eine Reduktion von SAT. Sei $\phi = C_1 \wedge \ldots \wedge C_m$ eine SAT Formel in konjunktiver Normalform über die Variablen $v_1, \ldots, v_n$. Für jede boolsche Variable v_i führt man eine binäre Variable x_i ein, die nur die Werte 0 und 1 annehmen darf. Dabei soll TRUE durch 1 und FALSE durch 0 modelliert werden.

Für jede Klausel wird eine Nebenbedingung konstruiert. Variablen v_i werden durch x_i, negierte Variablen $\neg v_i$ durch $(1 - x_i)$ und die Disjunktion $\vee$ durch $+$ ersetzt. Der so konstruierte Ausdruck für eine Klausel "zählt" die Anzahl der Literale, die zu TRUE evaluiert werden. Beschränkt man diese Audrücke nach unten durch 1, dann ist leicht einzusehen, dass eine SAT-Formel genau dann erfüllbar ist, wenn das zugehörige ILP zulässig ist. Um zu zeigen, dass das ILP-Entscheidungsproblem in NP ist, muss man noch zeigen, dass es einen Lösungsvektor x mit höchstens polynomieller Länge gibt. Diesen kann man dann in Polynomialzeit verifizieren.

9.1 Modellierung diskreter Optimierungsproblemen als ILP

Man spricht von *diskreten Optimierungsproblemen*, wenn der Lösungsraum nur endlich viele Elemente enthält.

Beispiel 9.3 (Das Rucksackproblem). *Gegeben sind n Objekte mit Gewichten w_i und Profiten p_i für $i \in [n]$. Weiterhin ist eine Gewichtsschranke B gegeben. Ziel ist es, eine Teilmenge K der Objekte zu finden, die das Gesamtgewicht K nicht übersteigt und gleichzeitig den Profit der Objekte in K maximiert. Das Rucksackproblem kann als ILP formuliert werden, indem man für jedes Objekt eine binäre Variable einführt, die genau dann den Wert* 1 *annimmt, wenn das Objekt zur Menge K gehört.*

$$\begin{aligned} &\max \sum_{i=1}^{n} p_i x_i \\ u.d.B. &\sum_{i=1}^{n} w_i x_i \quad \leq B \\ &x_i \in \{0,1\} \qquad\qquad \text{für } i \in [n]. \end{aligned}$$

Die Bedingung $x_i \in \{0,1\}$ kann man durch $0 \leq x_i \leq 1$ und $x_i \in \mathbb{Z}$ ausdrücken.

Definition 9.4. *Sei*

$$\begin{aligned} \text{optimiere } & c^T + d^T y \\ u.d.B. \ & Ax + by \quad = b \\ & x, y \quad \geq 0 \\ & x \text{ ganzzahlig} \end{aligned}$$

ein gemischt lineares Programm P. Als Relaxation von P bezeichnet man folgendes LP:

$$\begin{aligned} \text{optimiere } & c^T + d^T y \\ u.d.B. \ & Ax + by \quad = b \\ & x, y \quad \geq 0 \end{aligned}$$

Beispiel 9.5 (Relaxation des Rucksackproblems).

$$\begin{aligned} &\max \sum_{i=1}^{n} p_i x_i \\ \text{mit } &\sum_{i=1}^{n} w_i x_i \quad \leq B \\ &0 \leq x_i \leq 1 \qquad\qquad \text{für } i \in [n] \end{aligned}$$

Während das Rucksackproblem NP-vollständig ist, kann das relaxierte Problem in Linearzeit gelöst werden.

Allgemein möchte man die Tatsache ausmutzen, das man das relaxierte effizient lösen kann. Sei x^* eine optimale Lösung der Relaxation eines ILPs. Falls x^* zufälligerweise ganzzahlig ist, dann ist x^* auch eine optimale Lösung des ILPs. Ansonsten stellen sich folgende Fragen:

1. $c^T x$ ist eine untere (Maximierung) bzw. obere (Minimierung) Schranke für das Optimum des ILPs. Wie gut ist diese Schranke? Hier betrachtet man beispielsweise das *Integrality-Gap*, das definiert ist als die Differenz (manchmal auch der Quotient) zwischen den optimalen Kosten des ILP und der Relaxation des ILPs.

2. Kann man sich x^* zunutze machen, um das ganzzahlige Optimum zu ermitteln?

3. Gibt es Ungleichungen, die man zu $Ax = b$ hinzufügen kann, und die die Bedingung der Ganzzahligkeit ersetzen?

Beispiel 9.6. *Betrachten wir noch mal das Rucksackproblem. Wir definieren eine Instanz mit nur zwei Objekten und $w_1 = w_2 = p_1 = p_2 = c \in \mathbb{N}$ und $B = 2c - 1$. Die Kosten* OPT*(ILP) der optimalen Lösung des ILPs sind c, während die Kosten* OPT*(LP) der optimalen Lösung des relaxierten Problems $2c - 1$ sind.* OPT*(LP) ist also fast doppelt so groß wie* OPT*(ILP).*

Sei $T = \{x^1, \ldots, x^k\}$ die Menge der zulässigen Lösungen eines ILPs. Wir nehmen an, dass die Menge T endlich ist. Die *konvexe Hülle* von T ist

$$CH(T) = \left\{ \sum_{i=1}^{k} \lambda_i x^i, \lambda \geq 0; x^i \in T \right\}.$$

$CH(T)$ ist ein Polytop mit ganzzahligen Extrempunkten. Der Zulässigkeitsbereich P der Relaxation des zugehörigen ILPs erfüllt $CH(T) \subset P$. LPs können wir effizient lösen. Daher ist es wünschenswert eine ILP Formulierung zu finden, die nur $CH(T)$ als Zulässigkeitbereich umfasst.

Beispiel 9.7 (Perfektes Matching in allgemeinen Graphen). *Das Problem, in allgemeinen Graphen ein perfektes Matching mit minimalen Kosten zu berechnen, kann wie folgt als ILP modelliert werden:*

$$\begin{aligned} \min & \sum_{e \in E} c_e x_e & & \\ u.d.B. & \sum_{e: v \in e} x_e & = 1 & \quad \text{für alle } v \in V \\ x_e \in \{0,1\} & & & \quad \text{für alle } e \in E \end{aligned}$$

Sei P_{GRAD} das Polyeder, welches durch die Relaxation des obigen ILPs beschrieben wird. Sei M die Menge der Vektoren x, die perfekten Matchings entsprechen. Im Allgemeinen gilt nicht, dass $P_{\mathrm{GRAD}} = CH(M)$ (siehe Beispiel). Die folgenden Bedingungen können die Ganzzahligkeitsbedingung des ILPs ersetzen:

$$\forall S \subset V, |S| \text{ ungerade: } \sum_{e \in E(S)} x_e \leq \left\lfloor \frac{|S|}{2} \right\rfloor,$$

wobei $E(S) = \{\{u, v\} \in E \mid u, v \in S\}$. Die Anzahl der zusätzlichen Nebenbedingungen ist exponentiell in $|E|$. Man kann leicht sehen, dass jedes perfekte Matching alle Nebenbedingungen erfüllt. Sei P_M das Polyeder, das durch die Relaxation des ILP und die zusätzlichen Nebenbedingungen beschrieben wird. Es gilt $P_M = CH(M)$.

9.1.1 Minimaler Spannbaum

Gegeben ist ein Graph $G = (V, E)$ mit $n = |V|$ Knoten und einer Gewichtsfunktion auf den Kanten $c : E \to \mathbb{R}$. Gesucht ist ein Spannbaum mit minimalen Kosten. Einen Spannbaum in G kann man auf (mindestens) zwei Arten charakterisieren:

1. Ein Spannbaum ist ein zyklenfreier Teilgraph von G mit $n - 1$ Kanten.
2. Ein Spannbaum ist ein zusammenhängender Teilgraph von G mit n Knoten und $n - 1$ Kanten.

Die *Subtour-Formulierung* des MST benutzt die erste Charakterisierung:

$$\begin{aligned} \min & \sum_{e \in E} c_e x_e & & \\ \text{u.d.B.} & \sum_{e \in E} x_e & = & \; n - 1 \\ & \sum_{e \in E(S)} x_e & \leq & \; |S| - 1 \quad \forall S \subset V, S \neq \emptyset, V \\ & x_e \in \{0, 1\} & & \quad \forall e \in E \end{aligned}$$

Die Formulierung hat $2^n - 1$ Nebenbedingungen. Sei P_{SUB} das Polyeder, welches durch die Relaxation des ILPs beschrieben wird. Um das MST-Problem mit Hilfe der zweiten Charakterisierung als ILP zu formulieren, benötigen wir noch folgende Definition.

Definition 9.8. *Sei $G = (V, E)$ ein Graph und $S \subset V$ eine beliebige Teilmenge der Knoten. Dann ist der durch S definierte Schnitt*

$$\delta(S) = \{\{u, v\} \in E \mid u \in S, v \notin S\}.$$

Damit können wir die sogenannte *Cutset*-Formulierung des MST-Problems aufstellen:

$$\begin{aligned} \min & \sum_{e \in E} c_e x_e & & \\ \text{u.d.B.} & \sum_{e \in E} x_e & = & \; n - 1 \\ & \sum_{e \in \delta(S)} x_e & \geq & \; 1 \quad \forall S \subset V, S \neq \emptyset, V \\ & x_e \in \{0, 1\} & & \quad \forall e \in E \end{aligned}$$

Sei entsprechend P_{CUT} das Polyeder, welches durch die Relaxation dieses ILP beschrieben wird. Beide Formulierungen haben dieselbe Anzahl von Nebenbedingungen.

Satz 9.9. *Sei $G = (V, E)$ ein Graph mit $n = |V|$ Knoten und einer Gewichtsfunktions auf den Kanten $c : E \to \mathbb{R}$. Sei weiterhin T die Menge 0/1 Vektoren, die zu spannenden Bäume in G korrespondieren. Es gilt*

$$CH(T) = P_{SUB} \subset P_{CUT}.$$

P_{CUT} kann Extrempunkte haben, die nicht ganzzahlig sind.

9.1.2 Das Problem des Handlungsreisenden

Gegeben ist ein Graph $G = (V, E)$ mit $n = |V|$ Knoten und einer Gewichtsfunktions auf den Kanten $c : E \to \mathbb{R}$. Das Ziel ist es, eine Tour (Hamiltonkreis) mit minimalen Kosten zu finden.

Subtour-Formulierung:

$$\begin{aligned} & \min \sum_{e \in E} c_e x_e & & & \\ \text{u.d.B.} \quad & \sum_{e \in \delta(\{v\})} x_e & = & 2 \quad \forall v \in V \\ & \sum_{e \in E(S)} x_e & \leq & |S| - 1 \quad \forall S \subset V, S \neq \emptyset, V \\ & x_e \in \{0, 1\} & & \forall e \in E \end{aligned}$$

Cutset-Formulierung:

$$\begin{aligned} & \min \sum_{e \in E} c_e x_e & & & \\ \text{u.d.B.} \quad & \sum_{e \in \delta(\{v\})} x_e & = & 2 \quad \forall v \in V \\ & \sum_{e \in \delta(S)} x_e & \geq & 2 \quad \forall S \subset V, S \neq \emptyset, V \\ & x_e \in \{0, 1\} & & \forall e \in E \end{aligned}$$

Seien P_{SUB} und P_{CUT} die Polyeder, die durch die Relaxation des Subtour-ILPs bzw. des Cutset-ILPs beschrieben werden.

Satz 9.10. *$P_{SUB} = P_{CUT}$.*

Beweis. siehe übung □

9.2 Totale Unimodularität

Definition 9.11. *Eine quadratische Matrix A heißt* unimodular*, falls* $\det(A) \in \{-1, 1\}$ *ist.*

Eine $(m \times n)$-Matrix A heißt total unimodular*, falls jede reguläre quadratische Submatrix von A unimodular ist.*

Eine äquivalente Definition ist folgende: Eine Matrix A heißt total unimodular, falls die Determinante jeder quadratische Submatrix von A -1, 0 oder 1 ist.

Satz 9.12. *Sei $A \in \mathbb{Z}^{m \times n}$ eine total unimodulare Matrix und $b \in \mathbb{Z}^m$. Dann ist jede Basislösung von $Ax = b$ ganzzahlig.*

Insbesondere sind also alle BFS des LPs „minimiere $c^{\mathrm{T}}x$ unter den Bedingungen $Ax = b$ und $x \geq 0$“ ganzzahlig, falls b ganzzahlig und A total unimodular ist. Damit gibt es eine ganzzahlige optimale Lösung des LPs, falls das LP lösbar ist.

Satz 9.13. *Sei $G = (V, E)$ ein gerichteter Graph und A die zugehörige Inzidenzmatrix. Dann ist A total unimodular.*

Satz 9.13 ist der Grund dafür, dass das Maximal-Fluss-Problem, das Minimal-Schnitt-Problem und das Kürzeste-Wege-Problem ganzzahlige optimale Lösungen besitzen.

Sei $G = (V, E)$ ein ungerichteter Graph. Die Inzidenzmatrix $A = (a_{v,e})_{v \in V, e \in E}$ von G ist gegeben durch

$$a_{v,e} = \begin{cases} 1 & \text{falls } v \in e \text{ ist und} \\ 0 & \text{sonst.} \end{cases}$$

Satz 9.14. *Sei $G = (V, E)$ ein ungerichteter Graph und A die zugehörige Inzidenzmatrix. A ist genau dann total unimodular, wenn G bipartit ist.*

Satz 9.14 ist der Grund, warum das Matching-Problem in bipartiten Graphen wesentlich einfacher ist als das Matching-Problem in allgemeinen Graphen.

10 Approximationsalgorithmen

Algorithmen für NP-harte Optimierungsprobleme klassifiziert man typischerweise durch folgende drei Kategorien:

Exakte Algorithmen liefern garantiert eine optimale Lösung, die Laufzeit kann aber exponentiell in der Eingabegröße sein.

Approximationsalgorithmen liefern in polynomieller Zeit eine möglicherweise suboptimale Lösung. Jedoch gibt es eine Garantie für die Qualität der Lösung, d.h. eine Bedingung, um wieviel schlechter die Kosten der berechneten Lösung im Vergleich zu den optimalen Kosten sein können.

Heuristiken liefern keinerlei Garantien, weder für die Laufzeit noch für die Qualität der berechneten Lösung.

10.1 PCS - Precedence Constrained Scheduling

Gegeben ist ein gerichteter azyklischer Graph (DAG) $G = (V, E)$ und $m \in \mathbb{N}$. Gesucht ist ein Plan $S : V \to \mathbb{N}$ mit den Eigenschaften

- $(u, v) \in E \Rightarrow S(u) < S(v)$ und
- $|\{v \in V \mid S(v) = k\}| \leq m$ für alle $k \in \mathbb{N}$,

der $(\max\{S(v) \mid v \in V\})$ minimiert.

2-Approximation:

1. Markiere jeden Knoten mit seiner Tiefe im Graphen.
2. Sei d_i die Anzahl der Knoten mit Tiefe i. Verwende die ersten $\lceil d_1/m \rceil$ Zeiteinheiten für Knoten der Tiefe 1, die nächsten $\lceil d_2/m \rceil$ Zeiteinheiten für Knoten der Tiefe 2, usw.

Satz 10.1. *Sei L die Länge des konstruierten Plans und L_{OPT} die Länge eines optimalen Plans. Dann gilt $L \leq 2L_{OPT}$.*

Satz 10.2. *Wenn es einen polynomiellen Approximationsalgorithmus gibt, der stets $L < \frac{4}{3}L_{OPT}$ garantiert, dann ist $\mathsf{P} = \mathsf{NP}$.*

10.2 SIT - Scheduling of Independent Tasks

Gegeben sind n Aufgaben mit den Längen $w_1, \ldots, w_n$ und m Maschinen. Gesucht ist ein Plan, der die Aufgaben auf die Maschinen verteilt, so dass alle Aufgaben möglichst schnell komplett abgearbeitet werden. Gesucht ist also eine Funktion $S : [n] \to [m]$, die

$$\left(\max_{k \in [m]} \sum_{i : S(i) = k} w_i \right) \text{ minimiert.}$$

Algorithmus 1:

1. Sortiere die Aufgaben der Länge nach absteigend. Wir nehmen im Folgenden an, dass $w_1 \geq w_2 \geq \ldots \geq w_n$ gilt.

2. Verteile nacheinander die Aufgaben 1 bis n an die Maschinen, wobei die aktuelle Aufgabe der Maschine zugeordnet wird, die bisher am wenigsten zu tun hat.

Satz 10.3. *Sei L die Länge des durch Algorithmus 1 konstruierten Plans und L_{OPT} die Länge eines optimalen Plans. Es gilt*

$$L \leq \left(\frac{4}{3} - \frac{1}{3m}\right) L_{OPT}.$$

Algorithmus 2:

1. Sortiere die Aufgaben der Länge nach absteigend. Wir nehmen im Folgenden an, dass $w_1 \geq w_2 \geq \ldots \geq w_n$ gilt.

2. Berechne einen optimalen Plan für die Aufgaben $w_1, \ldots, w_k$.

3. Verteile nacheinander die Aufgaben $k+1$ bis n an die Maschinen, wobei die aktuelle Aufgabe der Maschine zugeordnet wird, die bisher am wenigsten zu tun hat.

Satz 10.4. *Sei L die Länge des durch Algorithmus 2 konstruierten Plans und L_{OPT} die Länge eines optimalen Plans. Es gilt*

$$L \leq \left(1 + \frac{m-1}{k+1}\right) L_{OPT}.$$

Für jedes $\epsilon > 0$ kann man k so wählen, dass der Algorithmus eine $(1+\epsilon)$-Approximation in Zeit $O(m^{\frac{m-1}{\epsilon}} + n \log n)$ liefert.

Betrachtet man die Anzahl der Maschinen m nicht als Teil der Eingabe sondern als Konstante, dann lässt sich das SIT-Problem mit m Maschinen (SIT_m) beliebig genau in Polynomialzeit approximieren.

Definition 10.5. *Seien Π ein Minimierungsproblem mit positiven optimalen Kosten, I eine Instanz von Π und L_{OPT} die Kosten einer optimalen Lösung von I.*

1. *Eine Lösung von I mit Kosten $L \leq (1+\epsilon)L_{OPT}$ heißt $(1+\epsilon)$-Approximation.*
2. *Ein $(1+\epsilon)$-Approximationsalgorithmus für Π berechnet für jedes I eine $(1+\epsilon)$-Approximation.*
3. *Ein PTAS (Polynomial Time Approximation Scheme) für Π ist eine Methode, die für jedes $\epsilon > 0$ einen $(1+\epsilon)$-Approximationsalgorithmus für Π mit polynomieller Laufzeit in der Größe von I liefert.*

4. *Ein FPTAS (Fully Polynomial Time Approximation Scheme) ist ein PTAS, bei der die Laufzeit der erzeugten Algorithmen polynomiell in der Größe von I und in $1/\epsilon$ ist.*

Für das PCS-Problem haben wir einen 2-Approximationsalgorithmus kennengelernt. Außerdem haben wir gesehen, dass die Existenz eines PTAS für das PCS-Problem $\mathsf{NP} = \mathsf{P}$ impliziert. Für das SIT_m-Problem wurde ein PTAS vorgestellt. Man kann zeigen, das es keinen FPTAS für das SIT_m-Problem geben kann, es sei denn $\mathsf{NP} = \mathsf{P}$. Im nächsten Abschitt werden wir für das Rucksackproblem ein FPTAS kennenlernen.

10.3 Das Rucksackproblem

Eingabe: Gewichte $w_1, \ldots, w_n \in \mathbb{N}$
Profite $p_1, \ldots, p_n \in \mathbb{N}$
Rucksackkapazität B
Gesucht ist eine Teilmenge $K \subseteq [n]$ mit $\sum_{i \in K} w_i \leq B$, die $\sum_{i \in K} p_i$ maximiert.

Dynamisches Programm Sei $F(p, i)$ das minimale Gewicht einer Rucksackfüllung, die nur die Elemente $1, \ldots, i$ benutzt und genau den Profit p erzielt. ($F(p, i)$ ist ∞, falls keine solche Rucksackfüllung existiert oder falls p negativ ist.) $F(p, i)$ kann folgendermaßen rekursiv bestimmt werden:

$$F(p, i) = \min\{F(p, i-1), F(p - p_i, i-1) + w_i \quad \text{für } i = 2, \ldots, n$$

$$F(p, 1) = \begin{cases} w_i & \text{für } p = p_1 \\ 0 & \text{für } p = 0 \\ \infty & \text{sonst} \end{cases}$$

Für die Kosten L_{OPT} der optimalen Lösung gilt:

$$L_{\text{OPT}} = \max\{p \mid F(p, n) \leq B\}$$

Sei $P = \max_{i \in [n]} p_i$. Der maximal erzielbare Profit ist somit nP. Es reicht also $F(p, n)$ für alle $p \in [nP]$ zu berechnen. Für die Laufzeit T gilt $T = O(n^2 P)$. Die Laufzeit ist pseudopolynomiell, d.h. polynomiell in der Größe der Instanz und der größten in der Eingabe auftretenden Zahl.

10.3.1 Ein FPTAS für das Rucksackproblem

Die Idee besteht darin, die Profite der Objekte zu skalieren und zu runden, so dass sie polynomiell beschränkt sind in n und $1/\epsilon$. Dann können wir den exakten pseudopolynomiellen Algorithmus verwenden, der dann entsprechend eine polynomielle Laufzeit bzgl. n und $1/\epsilon$ aufweist.

Wir skalieren die Profite mit dem Faktor $1/S$ und runden ab,.d.h. die skalierte Probleminstanz hat die Profite $\lfloor p_1/S \rfloor, \ldots, \lfloor p_n/S \rfloor$. Sei K eine optimale Lösung für die Originalinstanz und K_S die optimale Lösung der skalierten Instanz, die durch den pseudopolynomiellen Algorithmus berechnet wird. Dann gilt:

$$
\begin{aligned}
L = \sum_{i \in K_S} p_i &= S \sum_{i \in K_S} p_i / S \\
&\geq S \sum_{i \in K_S} \lfloor p_i / S \rfloor \\
&\geq S \sum_{i \in K} \left\lfloor \frac{p_i}{S} \right\rfloor \\
&\geq \sum_{i \in K} p_i / S - 1 \\
&\geq \left(\sum_{i \in K} p_i \right) - nS = L_{\mathrm{OPT}} - nS.
\end{aligned}
$$

Für $-nS \geq -\epsilon L_{\mathrm{OPT}}$ gilt also $L \geq (1-\epsilon) L_{\mathrm{OPT}}$. Wir suchen ein S mit $S \leq \frac{\epsilon L_{\mathrm{OPT}}}{n}$. Wir kennen jedoch L_{OPT} nicht und so benutzen wir $P \leq L_{\mathrm{OPT}}$ und setzen entsprechend $S = \frac{\epsilon P}{n}$. Für die Laufzeit T gilt

$$
T = O(n^2 P/S) = O\left(n^2 P \frac{n}{\epsilon P}\right) = O\left(\frac{n^3}{\epsilon}\right).
$$

Satz 10.6. *Für das Rucksackproblem gibt es einen FPTAS mit Laufzeit $O(n^3/\epsilon)$.*