Skript zur Vorlesung

Optimierung

Sommersemester 1996

Prof. Seidel

Inhaltsverzeichnis

Kapitel 1

Lineares Programmieren

Vorlesung vom 15.4.
Tobias Müller, Carsten Ullrich

1.1 Geometrische Grundlagen der Linearen Programmierung

1.1.1 Niedrigste-Punkte-Problem (NPP)

Definition 1 *Gegeben ist eine endliche Menge von Halbebenen und eine Richtung c. Im* Niedrigste-Punkte-Problem *wird derjenige zulässige Punkt gesucht, der am wenigsten in Richtung c liegt.*

Jede Gerade teilt die Ebene, in der sie liegt, in zwei Halbebenen auf (siehe Abb. 1.1).

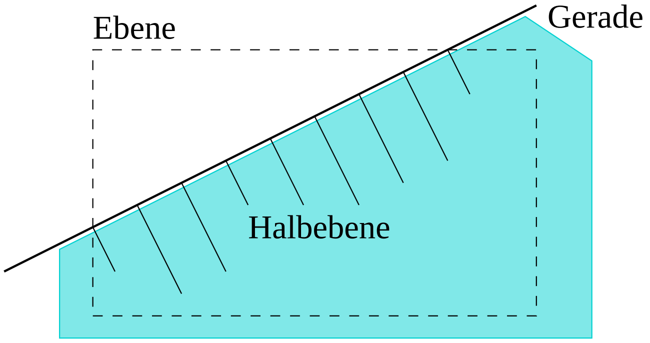


Abbildung 1.1: eine Halbebene.

Definition 2 *Ein Punkt ist ein* zulässiger Punkt *bezüglich einer Menge von Halbebenen, wenn er in jeder Halbebene enthalten ist.*

Im folgenden werden wir ohne Einschränkung annehmen, daß die Richtung c vertikal nach oben zeigt (durch Drehen der Ebene kann man jeden anderen Fall auf diesen zurückführen). Der im Niedrigste-Punkte-Problem gesuchte Punkt ist dann der zulässige Punkt, der am weitesten unten liegt.

Beobachtung 1 Betrachtet man das Niedrigste-Punkte-Problem, so gibt es drei Möglichkeiten:

1. Es gibt keine Lösung, da es keine zulässigen Punkte gibt (siehe Abb. 1.2).
2. Es gibt keine Lösung, da der Bereich der zulässigen Punkte nach unten nicht beschränkt ist (siehe Abb. 1.3).
3. Es gibt eine Lösung. Diese muß nicht notwendigerweise eindeutig sein (siehe Abb. 1.4).

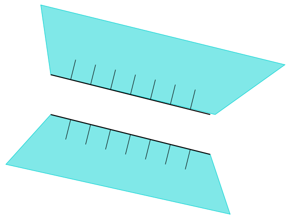


Abbildung 1.2: Zwei parallele Geraden bilden zwei sich nicht schneidene Halbebenen.

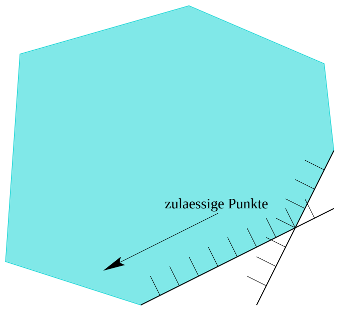


Abbildung 1.3: Der Bereich der zulässigen Punkte ist nach unten unbegrenzt.

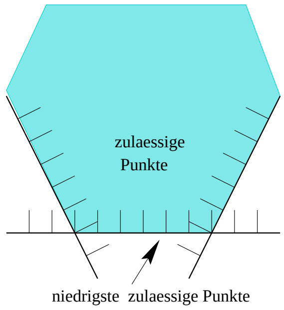


Abbildung 1.4: Es gibt unendlich viele niedrigste zulässige Punkte.

Beobachtung 2 Wenn das Niedrigste-Punkte-Problem eine Lösung hat, dann gibt es eine Lösung, die durch den Schnitt zweier Randgeraden bestimmt wird (siehe Abb. 1.5).
Dies gilt nur im trivialen Fall nicht, d.h. falls alle Randgeraden horizontal sind (siehe Abb 1.6).

1.1.2 Problem der Höchsten V-Kombination

Im folgenden werden wir wieder ohne Einschränkung annehmen, daß die Richtung c vertikal nach oben zeigt.

Definition 3 *Eine* V-Kombination *bezüglich einer Richtung c*

1. *ist eine Halbebene, die von unten durch eine horizontale Gerade begrenzt wird oder*
2. *sind zwei Halbebenen, deren Schnitt einen Keil bildet, der eine untere horizontale Tangente zuläßt (siehe Abb. 1.7).*

Definition 4 *Als* Höhe der V-Kombination *bezeichnet man die Höhe der horizontalen Geraden bzw. die Höhe des Schnittpunktes der beiden Randgeraden.*

Definition 5 *Gegeben ist eine endliche Menge von Halbebenen und eine Richtung c. Im* Problem der Höchsten V-Kombination *wird die am weitesten in Richtung c liegende, d.h. ohne Einschränkung die höchste V-Kombination gesucht (siehe Abb. 1.8).*

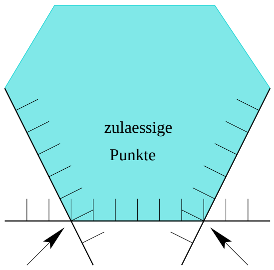


Abbildung 1.5: Niedrigste zulässige Punkte im Schnittpunkt zweier Randgeraden.

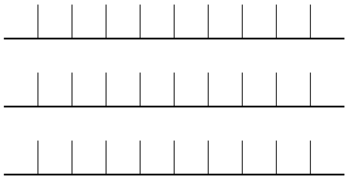


Abbildung 1.6: Trivialer Fall.

Beobachtung 3 Die Höhe jeder V-Kombination ist kleiner oder gleich der Höhe jedes zulässigen Punktes.

Beobachtung 4 Ist der zulässige Bereich nach unten unbeschränkt, dann gibt es keine V-Kombination.

Beobachtung 5 Ist x eine optimale Lösung des Niedrigste-Punkte-Problems und V eine Lösung des Problems der Höchsten V-Kombination, dann haben sie dieselbe Höhe. Sie sind nicht notwendigerweise gleich (siehe Abb. 1.9).

Beobachtung 6 Ist V eine Lösung des Problems der Höchsten V-Kombination und liegt x auf allen Randgeraden der an V teilnehmenden Halbebenen, dann ist x eine optimale Lösung des Niedrigste-Punkte-Problems (siehe Abb. 1.10).

Diese Beobachtungen stellen die geometrischen Interpretationen der Hauptsätze der linearen Programmierung dar. Insbesondere entspricht

Bemerkung 1 der Frage, ob es eine, keine oder mehrere Lösungen gibt (Existenzsatz),
Bemerkung 2 dem Basislösungsatz,
Bemerkung 3 dem schwachen Dualitätssatz,
Bemerkung 5 dem starker Dualitätssatz,
Bemerkung 6 dem Satz des komplementären Schlupfes.

Es ist möglich, durch das Betrachten von Halbräumen anstelle von Ebenen das Niedrigste Punkte Problem und das Problem der Höchsten V-Kombination in drei Dimensionen zu übertragen. Die Bemerkungen gelten entsprechend.

Vorlesung vom 18.4.
Udo Adamy, Torsten Bücheler, Oliver Pabst

1.2 Algebraische Grundlagen der Linearen Programmierung

Nachdem wir die geometrischen Ansätze des linearen Programmierens gesehen haben, definieren wir das Problem jetzt algebraisch.

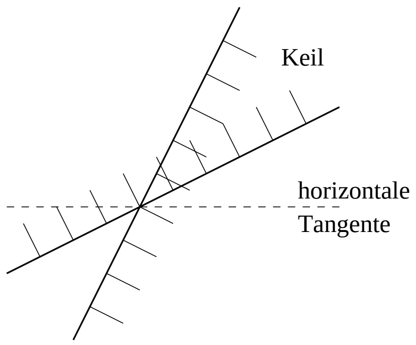


Abbildung 1.7: eine V-Kombination.

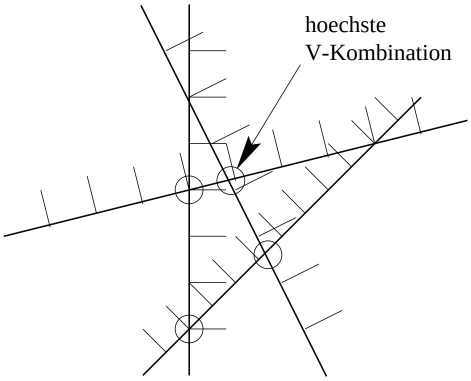


Abbildung 1.8: Problem der höchsten V-Kombination.

1.2.1 Das primale Problem

Definition 6 *Gegeben seien* $c_1, \ldots, c_d, b_1, \ldots, b_m, a_{11}, \ldots, a_{md} \in \mathbb{R}$. *Minimiere* $c_1x_1 + c_2x_2 + \cdots + c_dx_d$*, so daß*

$$\begin{array}{rcl} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1d}x_d & \geq & b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2d}x_d & \geq & b_2 \\ & \vdots & \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{md}x_d & \geq & b_m \end{array}$$

Oder in Vektorschreibweise: Seien $c \in \mathbb{R}^d, b \in \mathbb{R}^m$ *und* $A \in \mathbb{R}^{m \times d}$ *gegeben. Minimiere* c^Tx*, so daß* $Ax \geq b$*. Dabei gilt für Vektoren* $u, v \in \mathbb{R}^n$

$$u \leq v \quad :\Leftrightarrow \quad u_i \leq v_i \qquad \forall i \in \{1, \ldots, n\}$$

Beachte: Aus $u \not\leq v$ folgt nicht $u > v$. $u \not\leq v$ bedeutet, daß die Bedingung $u_i \leq v_i$ für mindestens ein $i \in \{1, \ldots, n\}$ verletzt ist. Das impliziert nicht, daß $u_i \leq v_i$ für alle $i \in \{1, \ldots, n\}$ verletzt ist, wie es für $u > v$ der Fall sein muß.

In manchen Büchern wird das Problem anders formuliert. Die verschiedenen Standardformen sind äquivalent. Eine allgemeine Formulierung lautet: *Optimiere* c^Tx mit $a_i^Tx\heartsuit_ib_i$, wobei *Optimiere maximiere* oder *minimiere* bedeutet und $\heartsuit_i$ aus der Menge $\{\leq, =, \geq\}$ gewählt ist.

Häufig verwendete Standardformen sind:

1. minimiere a^Tx, so daß $Ax \geq d$
2. minimiere b^Tx, so daß $Bx \geq e$ und $x \geq 0$, d.h. alle Variablen sind nicht negativ
3. minimiere c^Tx, so daß $Cx = f$ und $x \geq 0$, d.h. alle Variablen sind nicht negativ

Übung: Es ist eine gute Übung die veschiedenen Standardformen ineinander zu überführen.

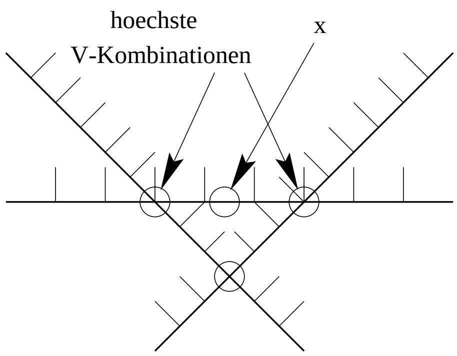


Abbildung 1.9: x und V haben dieselbe Höhe, sind aber nicht identisch.

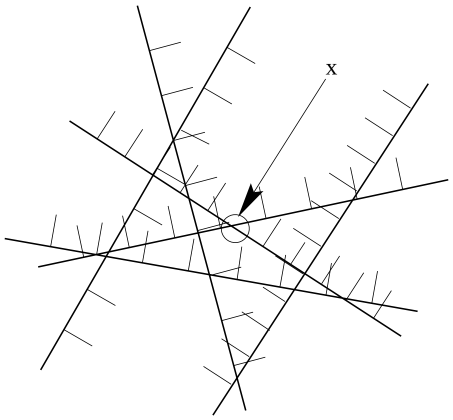


Abbildung 1.10: x liegt auf den an V teilnehmenden Halbebenen.

Dazu sind die folgenden Äquivalenzen („Rechenregeln") hilfreich:

1. maximiere $c^T x \quad \Leftrightarrow \quad$ minimiere $(-c)^T x$
2. $a_i^T x \leq b_i \quad \Leftrightarrow \quad (-a_i)^T x \geq -b_i$
3. $a_i^T x = b_i \quad \Leftrightarrow \quad a_i^T x \geq b_i$ und $(-a_i)^T x \geq -b_i$
4. $a_i^T x \geq b_i \quad \Leftrightarrow \quad a_i^T x + s_i = b_i$ und $s_i \geq 0$ $\quad s_i$ heißt *Schlupfvariable* (engl. slack variable).

Beispiel Ein typisches Beispiel für lineares Programmieren ist das *Diätproblem* (engl. dietproblem).
Ziel ist es, einen billigsten „Speiseplan" zu finden, der den Tagesmindest- und -höchstbedarf für alle Nährstoffe sicherstellt.
In der mathematischen Formulierung stehen d Nahrungsmittel (Rindfleisch, Käse, ...) zur Verfügung, die m Nährstoffe (Eiweiß, Zucker, ...) enthalten. Es seien:

c_j Preis von Nahrungsmittel j

a_{ij} Gehalt von Nährstoff i in Nahrungsmittel j

b_i^M Tagesmindestbedarf an Nährstoff i

b_i^H Tageshöchstbedarf an Nährstoff i

x_j Menge an Nahrungsmittel j pro Tag

Die Aufgabe besteht darin, $c_1x_1 + c_2x_2 + \cdots + c_dx_d$ zu minimieren, so daß für alle i die Ungleichungen $b_i^M \leq a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{id}x_d \leq b_i^H$ gelten. Außerdem muß $x_j \geq 0$ für alle j gelten, da die Nahrungsmittelmenge nicht negativ sein darf.

Definition 7 *Gegeben sei das Lineare Problem: Minimiere $c^T x, Ax \geq b$.*
Die Abbildung $C : \mathbb{R}^d \to \mathbb{R}, x \to c^T x$ heißt „Zielfunktion“. *Ein $x \in \mathbb{R}^d$ ist* zulässig *(engl. feasible), wenn $Ax \geq b$ gilt. Die Menge $P = \{x \in \mathbb{R}^d \mid Ax \geq b\}$ heißt der* Zulässigkeitsbereich.

Für den Zulässigkeitsbereich eines Linearen Problems gibt es drei Möglichkeiten. Diese entsprechen den drei möglichen Fällen, die bei der geometrischen Interpretation (NPP) auftauchen können.

1. $P = \emptyset \quad \Rightarrow$ LP ist unlösbar
2. $P \neq \emptyset$, aber das $\inf\{c^T x \mid x \in P\}$ existiert nicht (z.B. $0 \cdot x \geq -1$)
   $\Rightarrow$ LP ist „lösbar“, aber es gibt keine optimale Lösung.
3. $P \neq \emptyset$ und das $\min\{c^T x \mid x \in P\}$ existiert
   $\Rightarrow$ LP ist „lösbar“ und hat eine endliche Lösung x^*
   $c^T x^* = \min\{c^T x \mid x \in P\}$

Für ein Lineares Problem stellt sich nun die Frage: Welcher der drei Fälle trifft zu? Die Beantwortung dieser Frage führt zu weiterer Theorie und dem *Dualen Problem.*

1.2.2 Das duale Problem

Für ein Primales Problem ist es vorteilhaft, untere Schranken für die Zielfunktion angeben zu können.
Betrachten wir das Primale Problem: Minimiere $c^T x$, so daß

$$\begin{array}{rcl} a_1^T x & \geq & b_1 \\ a_2^T x & \geq & b_2 \\ & \vdots & \\ a_m^T x & \geq & b_m \end{array}$$

gilt. Wie kann man untere Schranken für den optimalen Wert der Zielfunktion herleiten?
Mathematische Betrachtung: Seien $\lambda, \lambda_1, \lambda_2 \in \mathbb{R}$

1. Skalarmultiplikation: $a^T x \geq b,\ \lambda \geq 0 \quad \Rightarrow \quad (\lambda a)^T x \geq \lambda b$
2. Linearkombination: $a_1^T x \geq b_1,\ a_2^T x \geq b_2,\ \lambda_1 \geq 0,\ \lambda_2 \geq 0 \quad \Rightarrow \quad (\lambda_1 a_1 + \lambda_2 a_2)^T x \geq \lambda_1 b_1 + \lambda_2 b_2$

Stelle c als positive Linearkombination der a_i dar, d.h. $c = \sum_{i=1}^m y_i a_i$ mit $y_i \geq 0$. Es folgt für die Zielfunktion $c^T x$:

$$c^T x = (\sum_i y_i a_i)^T x = \sum_i y_i \cdot (a_i^T x) \geq \sum_i y_i b_i$$

Wenn wir c als positive Linearkombination darstellen können, so erhalten wir auf diese Weise eine untere Schranke für die Zielfunktion $c^T x$.

Beispiel Minimiere $4x_1 + 3x_2$ unter den Nebenbedingungen:

I. $x_1 + 2x_2 \geq 7$

II. $2x_1 - x_2 \geq 5$

III. $3x_1 + x_2 \geq -2$

Wenn wir die Ungleichung I mit 2 multiplizieren und zur Ungleichung II addieren, erhalten wir:

$$4x_1 + 3x_2 \geq 19$$

19 ist also eine untere Schranke für die Zielfunktion. Addieren von Ungleichung I zu Ungleichung III führt zu:

$$4x_1 + 3x_2 \geq 5$$

Also ist 5 auch eine untere Schranke, die allerdings schwächer ist.

Die Frage ist jetzt: Wie bekommen wir die beste untere Schranke?

Definition 8 *Gegeben sei das Lineare Problem: minimiere $c^T x$, so daß $Ax \geq b$ gilt. Das Finden der besten unteren Schranke für dieses LP ist wiederum ein Lineares Problem. Es heißt* duales Problem *und ist definiert als:*
Maximiere $y^T b$, so daß $y^T A = c^T$ und $y \geq 0$ gelten.

Satz 1 *(Schwache Dualität) Sei $\overline{x}$ eine zulässige Lösung für das primale Problem:*

$$\textit{Minimiere } c^T x \textit{ unter der Bedingung } Ax \geq b$$

und $\overline{y}$ eine zulässige Lösung für das duale Problem:

$$\textit{Maximiere } y^T b, \textit{ wobei } y^T A = c^T \textit{ und } y \geq 0.$$

Dann ist die Zielfunktion des dualen Problems immer kleiner oder gleich der Zielfunktion des primalen Problems. D.h es gilt:

$$\overline{y}^T b \;\leq\; c^T \overline{x}.$$

Beweis Eigentlich ist es einsichtig, daß der Wert der besten unteren Schranke nicht größer als der Wert der Zielfunktion sein kann.
Es gilt: $A\overline{x} \geq b$. Da $\overline{y} \geq 0$ ist, können wir die Ungleichung $A\overline{x} \geq b$ mit $\overline{y}^T$ multiplizieren ohne daß sich das Ungleichheitszeichen ändert. Es folgt:

$$\begin{array}{lrcl} & \overline{y}^T \cdot (A\overline{x}) & \geq & \overline{y}^T b \\ \Leftrightarrow & (\overline{y}^T A)\overline{x} & \geq & \overline{y}^T b \\ \Leftrightarrow & c^T \overline{x} & \geq & \overline{y}^T b \end{array}$$

□

Korollar 1 *Ist das primale Problem unbeschränkt, dann ist das duale Problem unlösbar.*

Beweis Zulässige Lösungen für das duale Problem wären untere Schranken für das primale Problem. Dies ist ein Widerspruch zur Unbeschränktheit des primalen Problems. □

1.2.3 Bezug zur geometrischen Interpretation ($d = 2$)

Im $\mathbb{R}^2$ beschreibt eine Gleichung der Form $\alpha_1 x_1 + \alpha_2 x_2 = \beta$ eine Gerade. β ist ein Maß für die Entfernung der Geraden vom Ursprung des Koordinatensystems.

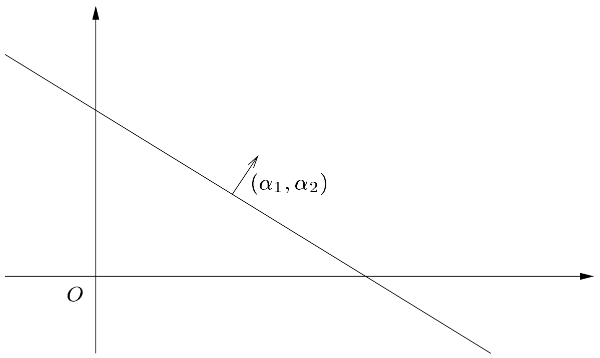


Abbildung 1.11: Gerade mit Normalenvektor $\alpha = (\alpha_1, \alpha_2)$

Ersetzen wir die Gleichung durch eine Ungleichung, so erhalten wir eine Halbebene. Die Gleichung $\alpha_1 x_1 + \alpha_2 x_2 \geq \beta$ beschreibt also die schraffierte Oberfläche der Abbildung 1.12.

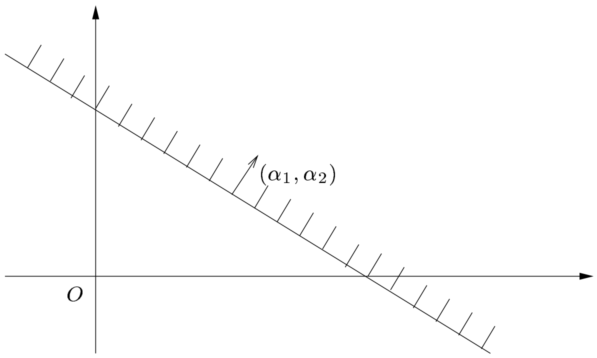


Abbildung 1.12: Halbebene mit Normalenvektor $\alpha = (\alpha_1, \alpha_2)$

In einem primalen linearen Problem der Form: Minimiere $c^T x, Ax \geq b$ mit $c \in \mathbb{R}^2, b \in \mathbb{R}^m, A \in \mathbb{R}^{m\times 2}$ beschreiben die Ungleichungen $Ax \geq b$, oder ausführlich:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 &\geq b_1 \\ &\vdots \\ a_{m1}x_1 + a_{m2}x_2 &\geq b_m, \end{aligned}$$

eine Menge von m Halbebenen. Der Zulässigkeitsbereich entspricht dem Schnitt dieser Halbebenen. Der Vektor c gibt die Optimierungsrichtung an. Die Geradenschar

$$c_1x_1 + c_2x_2 = \gamma$$

ist orthogonal zum Vektor c. Eine Änderung von γ bewirkt ein Verschieben der Gerade in Richtung des Vektors c. Das duale Problem zum primalen Problem besteht genau darin, das kleinste γ zu finden, so daß die Gerade den Schnitt der Halbebenen noch berührt (siehe Abbildung 1.13).

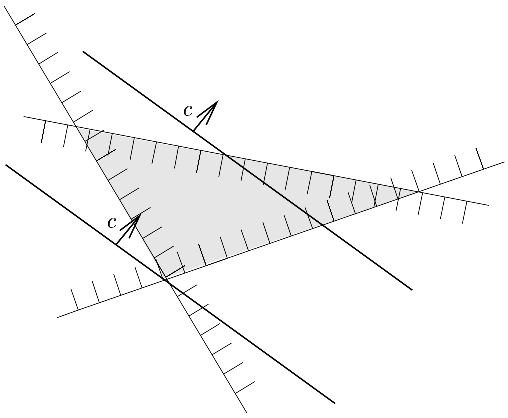


Abbildung 1.13: Verschieben der Gerade zu einer unteren Tangente

Diese Tangente existiert natürlich nur dann, wenn das primale Problem beschränkt ist (siehe Satz 1). Sie löst das duale Problem, d.h. sie stellt die beste untere Schranke für das primale Problem dar. Es ist einsichtig, daß der Schnitt der Tangente mit dem Schnitt der Halbebenen (Zulässigkeitsbereich) sowohl das duale (HV-Komb.) als auch das primale Problem (NPP) optimal löst. Dies besagt der Satz der *starken Dualität*.

Beispiel Minimiere $4x + y$ unter den Nebenbedingungen:

$H_1 \quad 2x - 3y \geq 5$

$H_2 \quad x + 2y \geq -1$

Der Vektor c ist folglich $(4, 1)$. Wenn wir die Ungleichung H_2 mit 2 multiplizieren und zur Ungleichung H_1 addieren, erhalten wir:

$$H_3 = H_1 + 2 \cdot H_2 \qquad 4x + y \geq 3.$$

Dies stellt eine Linearkombination für c dar, d.h. $c = (4, 1) = (2, -3) + 2 \cdot (1, 2)$. Somit ist 3 eine untere Schranke für das primale Problem. Es ist sogleich die optimale Lösung wie die Abbildung 1.14 veranschaulicht.

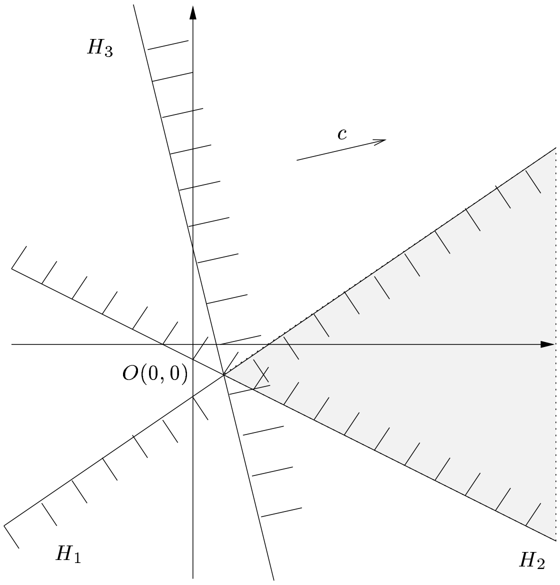


Abbildung 1.14: H_3 bildet eine Tangente an den Zulässigkeitsbereich, die orthogonal zu c ist.

Vorlesung vom 22.4.
Christian Lennerz, Harald Müller, Thorsten Oelgart

1.2.4 Starke Dualität

Wir verwenden nun die folgenden Bezeichnungen.

$$A = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_m \end{pmatrix} \in \mathbb{R}^{m \times d}, \qquad M = \{1, \ldots, m\}, \qquad B \subset M, \qquad |B| = d$$

Die Vektoren $a_1, \ldots, a_m$ sind also die *Zeilen* der Matrix A.

Definition 9 *Für eine Teilmenge*

$$B = \{i_1, \ldots, i_d\}$$

von M definieren wir

$$A_B = \begin{pmatrix} a_{i_1} \\ a_{i_2} \\ \vdots \\ a_{i_d} \end{pmatrix} \in \mathbb{R}^{d \times d} \quad \textit{und} \quad b_B = \begin{pmatrix} b_{i_1} \\ \vdots \\ b_{i_d} \end{pmatrix} \in \mathbb{R}^d.$$

B heißt Basis, *wenn A_B invertierbar ist. Ein Vektor x aus $\mathbb{R}^d$, der den Gleichungen $A_B x = b_B$ genügt, heißt* Basislösung *(„Eckenlösung"); d ist die Dimension des betrachteten Vektorraumes.*

Satz 2 *Gegeben ist ein lineares Programm*

$$\min\ c^T x, \qquad Ax \geq b.$$

Wir nehmen weiterhin an, daß der Rang von A gleich d ist.
Gibt es eine zulässige Lösung x, so existiert auch eine Basis B und eine zugehörige Basislösung $\bar{x}$, die ebenfalls zulässig ist. D.h., es gelten

$$A\bar{x} \geq b, \quad \textit{und} \quad A_B\bar{x} = b_B.$$

Um diese Aussage zu beweisen, wollen wir zeigen, daß man aus einem beliebigen zulässigen Vektor eine Basislösung konstruieren kann. Für $d = 3$ kann man sich das Verfahren folgendermaßen veranschaulichen.

1. Haben wir einen zulässigen Punkt x im Raum, dann kann dieser Punkt entlang eines Strahls r verschoben werden, bis man an eine Begrenzung des Zulässigkeitsbereiches (bei $d = 3$ eine Ebene) „stößt".
2. Auf dieser Begrenzungsebene kann der Punkt entlang eines anderen Vektors r' bis zu der Schnittgeraden zweier Begrenzungsebenen bewegt werden.
3. Auf dieser Geraden kann der Punkt wiederum entlang eines neuen Vektors r'' bis zum Schnittpunkt der Geraden mit einer weiteren Begrenzungsebene, also dem Schnittpunkt dreier Begrenzungsebenen verschoben werden.

Der Schnittpunkt, den man auf diese Weise erhält, ist die gesuchte Basislösung.

Beweis
Sei x eine zulässige Lösung für das lineare Programm, aus der wir eine Basislösung $\hat{x}$ konstruieren wollen. Wir führen den Beweis durch Induktion über die Anzahl k der Ungleichungen, die $\hat{x}$ mit Gleichheit erfüllt. Dabei nehmen wir ohne Einschränkung an, daß diese den ersten k Zeilenvektoren von A entsprechen.
Man kann leicht nachrechnen, daß der Vektor

$$x \;-\; \left(\frac{a_1 x}{a_1 a_1^T}\right) \cdot a_1^T$$

die erste Gleichung erfüllt. Wir nehmen nun an, daß x bereits $(k-1) < d$ Ungleichungen mit Gleichheit erfüllt; d.h.

$$\begin{array}{rcll} a_i x & = & b_i \qquad 1 \leq i < k \leq d & \qquad a_i \text{ linear unabhängige Zeilenvektoren} \\ Ax & \geq & b. & \end{array}$$

Nun wollen wir x in Richtung eines Vektors r verschieben, der die folgenden Eigenschaften erfüllt:

- $r \neq 0$, da sonst keine Verschiebung des Punktes möglich ist, und
- $a_i r = 0,\ 1 \leq i < k$, da die bereits erzeugten Gleichungen erhalten bleiben müssen.

Um einen Vektor r mit diesen Eigenschaften zu bestimmen muß also ein Gleichungssystem mit $(k-1) < d$ linear unabhängigen Gleichungen in d Variablen gelöst werden. Eine solche Lösung existiert immer.
Für ein $\lambda \in \mathbb{R}_{\geq 0}$ definieren wir dann den Punkt

$$\hat{x}(\lambda) \quad = \quad x + \lambda r.$$

Für $\hat{x}(\lambda)$ können wir nachrechnen, daß

$$a_i\hat{x}(\lambda) \quad = \quad \underbrace{a_i x}_{=b_i} + \underbrace{\lambda a_i r}_{=0} \quad = \quad b_i$$

für jedes i aus $\{1, \ldots, k-1\}$ gilt. Dieser Punkt erfüllt also noch immer die ersten $(k-1)$ Gleichungen. Wir müssen durch die Wahl von λ allerdings auch gewährleisten, daß $x(\lambda)$ im zulässigen Bereich liegt. Das bedeutet, daß

$$a_j\hat{x}(\lambda) \geq b_j$$

für jedes $j \in \{k, \ldots, m\}$ erfüllt sein muß. Da x ein zulässiger Punkt ist, erhalten wir daraus die Einschränkungen

$$\lambda a_j r \quad \geq \quad \underbrace{b_j - a_j x}_{\leq 0}$$

für λ. Ist bereits $a_j r = 0$, so kann λ beliebig groß gewählt werden; sonst müssen wir zwei Fälle unterscheiden.

1. $a_j r > 0$
$$\lambda \geq \underbrace{\frac{b_j - a_j x}{a_j r}}_{\leq 0} \qquad \Rightarrow \qquad \lambda \in \left[\frac{b_j - a_j x}{a_j r}; \infty\right[$$

2. $a_j r < 0$
$$\lambda \leq \underbrace{\frac{b_j - a_j x}{a_j r}}_{\geq 0} \qquad \Rightarrow \qquad \lambda \in \left]-\infty; \frac{b_j - a_j x}{a_j r}\right]$$

Für jedes $j \in \{k, \ldots, m\}$ mit $a_j r < 0$ erhalten wir somit eine nicht negative ober Schranke für λ. Wählen wir $\lambda_0 = (b_{j_0} - a_{j_0}x)/a_{j_0}r$ als das Minimum dieser Schranken, so ist $\hat{x}(\lambda_0)$ ein zulässiger Punkt, der zusätzlich auch die Ungleichung j_0 mit Gleichheit erfüllt. (Gilt für alle $j \in \{k, \ldots, m\}$, daß $a_j r \geq 0$, so können wir r auch durch $(-r)$ ersetzen und erhalten dann eine obere Schranke für λ.)
Da das Skalarprodukt $a_{j_0}r$ echt kleiner als 0 ist, muß (nach Auswahl von r) der Zeilenvektor a_{j_0} linear unabhängig sein von $\{a_1, \ldots, a_{k-1}\}$. Nach entsprechendem Umnumerieren der Zeilen von A (d.h., Vertauschen von a_{j_0} und a_k) erhalten wir also:

$$\begin{array}{rcll} a_i\hat{x} & = & b_i & 1 \leq i < k \\ a_k\hat{x} & = & b_k & \\ \forall a_l : a_l\hat{x} & \geq & b_l & k < l \leq m, \end{array}$$

und dabei sind die Vektoren $\{a_1, \ldots, a_k\}$ linear unabhängig.

$\square$

Man kann sich überlegen, daß man die Konstruktion des vorigen Beweises so modifizieren kann, daß man zu jeder *optimalen* Lösung eine optimale Basislösung finden kann. Diese Aussage werden wir im folgenden verwenden.

Satz 3 *(Starke Dualität)*
Sind x^* und y^* zulässige primale bzw. duale Lösungen, dann gilt:

$$c^T x^* = {y^*}^T b \Leftrightarrow x^* \textit{ und } y^* \textit{ sind beide optimal}$$

Beweis
Aus dem Satz über die *schwache Dualität* können wir sofort folgern, daß die Lösungen x^* und y^* nicht mehr verbessert werden können, wenn bereits $c^T x^* = {y^*}^T b$ gilt. Also müssen x^* und y^* in diesem Falle optimal sein.
Es bleibt also nur noch zu zeigen, daß die durch das duale Problem gefundene beste untere Schranke für die Zielfunktion des primalen Problems auch stets erreicht werden kann.
Seien also x^* und y^* jeweils optimale Lösungen. Ohne Einschränkung können wir annehmen, daß x^* sogar eine optimale *Basislösung* ist. Es existiert also eine Basis B, für die gilt:

$$\begin{array}{rcll} A_B x^* & = & b_B & \text{und} \\ A x^* & \geq & b & \\ \Rightarrow x^* & = & {A_B}^{-1} b_B & \end{array}$$

Da A_B invertierbar ist, erhalten wir insbesondere eine eindeutige Lösung für

$$c^T = {y_B}^T A_B,$$

nämlich

$$y_B{}^T = c^T A_B{}^{-1}.$$

Erweitert man den d-Vektor y_B zu einem m-Vektor y durch Auffüllen mit Nullen, so ergibt sich $c^T = y^T A$. Wir untersuchen nun, ob der so berechnete Vektor y eine zulässige Lösung des dualen Problems ist.

1. Fall:
$y_B \geq 0 \quad \Rightarrow \quad y \geq 0$ (d.h. y ist zulässige Lösung)
Da y^* optimal gewählt war, gilt

$$y^{*T} b \geq y^T b = y_B{}^T b_B = \underbrace{(c^T A_B{}^{-1})}_{y_B{}^T} b_B = c^T \underbrace{(A_B{}^{-1} b_B)}_{x^*} = c^T x^*$$

Der Satz über die schwache Dualität liefert

$$y^{*T} b \leq c^T x^*,$$

und daraus folgt die Aussage des Satzes.

2. Fall:
$y_B \not\geq 0$, d.h. $\exists i \in B$ mit $y_i < 0$; y ist keine zulässige duale Lösung. Ohne Einschränkung sei $i = 1$ und somit $y_1 < 0$.
Wir werden nun zeigen, daß dieser Fall nicht auftreten kann, indem wir von x^* ausgehend eine *bessere* Lösung konstruieren. Da x^* als optimal vorausgesetzt war, führt dies zu einem Widerspruch.
Sei dazu a die erste Spalte von $A_B{}^{-1}$. Dann gilt

$$A_B a = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \quad \text{da} \quad A_B \cdot A_B{}^{-1} = I.$$

Verschiebt man nun den Vektor x^* in Richtung a,

$$\hat{x}(\lambda) = x^* + \lambda a \qquad \lambda \in \mathbb{R}_{\geq 0},$$

so erfüllt dieser Punkt alle Ungleichungen $j \in B$, da

$$A_B \hat{x}(\lambda) = \underbrace{A_B x^*}_{b_B} + \underbrace{\lambda}_{\geq 0} \underbrace{A_B a}_{=(1,0,\ldots,0)^T} \geq b_B.$$

Für die Zielfunktion gilt dann

$$c^T \hat{x}(\lambda) = c^T x^* + \lambda \cdot \underbrace{c^T a}_{=y_1<0} \leq c^T x^*. \tag{1.1}$$

Wir können also eine bessere Lösung als x^* angeben, wenn wir λ positiv wählen können und dabei $\hat{x}(\lambda)$ zulässig ist. Dazu müssen wir nur noch die Ungleichungen $j \notin B$ überprüfen.

$$\begin{array}{rrcc} & a_j \hat{x}(\lambda) & \geq & b_j \\ \Leftrightarrow & a_j x^* + \lambda a_j a & \geq & b_j \\ \Leftrightarrow & \lambda a_j a & \geq & \underbrace{b_j - a_j x^*}_{\leq 0} \end{array}$$

Ist $a_j a = 0$, so können wir λ beliebig groß wählen; anderenfalls müssen wir auch hier wieder zwei Fälle untersuchen:

1. $a_j a > 0$

$$\lambda \geq \underbrace{\frac{b_j - a_j \hat{x}}{a_j a}}_{\leq 0} \qquad \Rightarrow \qquad \lambda \in \left[\frac{b_j - a_j \hat{x}}{a_j a}; \infty\right[$$

2. $a_j r < 0$

$$\lambda \leq \underbrace{\frac{b_j - a_j\hat{x}}{a_j a}}_{\geq 0} \qquad \Rightarrow \qquad \lambda \in \left]-\infty; \frac{b_j - a_j\hat{x}}{a_j a}\right]$$

D.h. gilt für alle $j \notin B : a_j x^* > b_j$ dann kann man ein $\lambda > 0$ wählen, so daß $\hat{x}(\lambda)$ zulässig und $c^T\hat{x}(\lambda) < c^T x^*$ ist, woraus folgt, daß x^* nicht optimal war. □

Bisher haben wir den vorigen Satz nur unter der folgenden „Nichtdegeneriertheitsannahme“ bewiesen: Jeder Vektor x aus $\mathbb{R}^d$ erfüllt höchstens d Zeilen von $Ax \geq b$ mit Gleichheit. Im Falle $d = 2$ besagt dies, daß sich nie mehr als zwei Geraden, die die Halbebenen begrenzen, in einem Punkt schneiden.

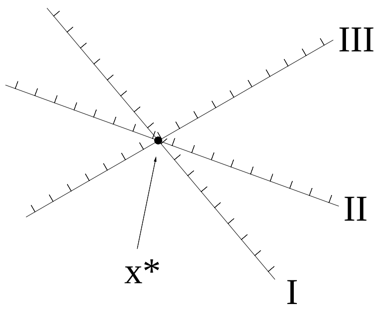


Abbildung 1.15: Degeneriertheitsproblem

Wir werden später sehen, wie wir diese Annahme umgehen können.

Vorlesung vom 25.4.1996
Christoph Endres, Rolf–Armin Puin, Tim Schwartz, Robert Wirth

1.2.5 Der Simplex–Algorithmus

Der Beweis des Satzes zur starken Dualität erlaubt uns folgende Beaobachtung: Zu jeder Basislösung x eines Linearen Programmes können wir eine zugehörige *duale* Lösung y finden. Ist y auch dual *zulässig*, so muß x schon eine optimale Lösung sein; anderenfalls können wir x durch eine Verschiebung in eine andere Basislösung $\hat{x}$ überführen, die einen besseren Wert unter der Zielfunktion ergibt.
Dies liefert uns somit eine Idee für einen Algorithmus zur Betimmung einer optimalen Lösung. Anschaulich können wir uns diesen folgendermaßen vorstellen: Die Nebenbedingungen des Linearen Programms definieren ein Polytop im $\mathbb{R}^d$. Die Basislösungen des Programmes entsprechen genau dessen Eckpunkten. Wir wissen weiter, daß mindestens einer dieser Eckpunkte eine optimale Lösung des Problems ist. Die Idee für einen Algorithmus ist nun, diese ganz bestimmte Ecke dadurch zu finden, daß man eine Ecke des Polytops als Startpunkt wählt und zu einer benachbarten Ecke wechselt, wenn diese einen besseren Wert unter der Zielfunktion liefert. Dies wiederholt man solange, bis man die Ecke erreicht hat, die der optimalen Lösung entspricht.

Sei nun x eine nicht–optimale Basislösung. Mit den Bezeichnungen des obigen Beweises betrachten wir die folgende Situation: Wir bilden die Menge

$$\Lambda \quad = \quad \left\{ \frac{b_j - a_j x}{a_j a} \mid (j \notin B);\ a_j a < 0 \right\}, \tag{1.2}$$

suchen darin ein minimales Element λ und bilden

$$\hat{x} \quad = \quad x + \lambda a. \tag{1.3}$$

Für diese neue Basislösung erhalten wir dann

$$c^T\hat{x} \quad \leq \quad c^T x.$$

Diese Konstruktion kann man aber nur dann durchführen, wenn die Menge Λ mindestens ein Element enthält. Weiter sehen wir, daß wir die Basislösung x nur dann wirklich verändern, wenn λ von Null verschieden ist. Also müssen wir die folgenden Fälle unterscheiden:

1. $\Lambda = \emptyset$

   Es ist vernünftig, das Minimum einer leeren Menge als ∞ zu definieren; also als das "neutrale Element" der Operation "min". Das bedeutet, daß wir λ in (1.3) beliebig groß wählen können; also können wir auch die Kosten $c^T\hat{x}$ beliebig verkleinern (s. (1.1) im Beweis auf Seite 12). Das Lineare Programm ist also unbeschränkt.

2. $\Lambda \neq \emptyset,\ \lambda > 0$

   Das Minumun λ werde angenommen für j^*. Dann ist $\hat{x}$ eine zulässige Basislösung bzgl. der Basis $\hat{B} = B\backslash\{b_1\} \cup \{j^*\}$ und es gilt $c^T\hat{x} < c^T x$.

3. $\Lambda \neq \emptyset,\ \lambda = 0$

   Das Minumun λ werde wieder angenommen für j^*. Dann ist $\hat{x}$ ebenfalls zulässige Basislösung bzgl. $\hat{B}$, es tritt aber wegen $c^T\hat{x} = c^T x$ keine Verbesserung des Zielfunktionswertes ein.

Obige Konstruktion nennen wir einen *Simplex–Schritt*. Der *Simplex–Algorithmus* besteht nun darin, aus einer Basislösung mit Hilfe einer Folge von Simplex–Schritten eine optimale Lösung des Linearen Programms zu finden. Nach jedem Simplex–Schritt kann man eine der folgenden Beobachtungen machen:

0. Man hat die optimale Lösung gefunden, oder
1. das Problem ist unbeschränkt, oder
2. es liegt nun eine bessere zulässige Basislösung $\hat{x}$ bzgl. einer neuen Basis $\hat{B}$ vor, wobei dann für den neuen Zielfunktionswert $c^T\hat{x} < c^T x$ gilt, oder
3. die ursprüngliche Lösung $x = \hat{x}$ ist auch Basislösung bzgl. der neuen Basis $\hat{B}$.

Der letzte Fall bedeutet für den Simplex–Algorithmus, daß außer der aktuellen Basislösung x keine andere Lösung mehr konstruiert wird; der Algorithmus wird von diesem Zeitpunkt an einen Zyklus von Basen erzeugen, die alle der Lösung x entsprechen. Diese Situation kann jedoch nur dann auftreten, wenn x mehr als $d = |B|$ der Ungleichungen $a_i x \leq b_i$ mit Gleichheit erfüllt, denn es gilt:

$$\begin{array}{rcll} a_i \cdot x & = & b_i & \text{für} \quad i \in B \\ a_{j^*} \cdot x & = & b_{j^*} & \text{wobei} \quad j^* \notin B; \end{array}$$

d.h., das System (A, b) ist degeneriert. Somit folgt, daß der Simplex–Algorithmus jedes nicht–degenerierte Lineare Programm in endlich vielen Schritten lösen kann. Es gibt nämlich nur endlich viele verschiedene Basen B, und da wir in jedem Schritt eine echt bessere Lösung als die bisherige erzeugen, kann keine Basis mehrfach auftauchen.
Es stellt sich nun die Frage, wie wir degenerierte Systeme behandeln. Eine Möglichkeit besteht darin, die Aufgabenstellung leicht zu stören, so daß sich die Degeneriertheit auflöst. Dies wollen wir im folgenden Abschnitt untersuchen.

1.2.6 Die Perturbationsmethode

Mit Hilfe der *Perturbationsmethode* wollen wir ein Lineares Programm so modifizieren, daß eine eventuell vorliegende Degeneriertheit aufgelöst wird. Die folgende Aussage erlaubt uns zu überprüfen, wann ein System nicht–degeneriert ist.

Proposition 1 *Das Gleichungssystem*

$$Ax = b$$

ist nicht–degeneriert, falls für je zwei voneinander verschiedene Basen B_1 und B_2 sowie für die zugehörigen Basislösungen x_1 und x_2 gilt:

$$A_{B_1} x_1 = b_{B_1} \quad \textit{und} \quad A_{B_2} x_2 = b_{B_2}, \quad \textit{d.h.} \quad A_{B_1}^{-1} b_{B_1} \neq A_{B_2}^{-1} b_{B_2}.$$

Sei nun das Lineare Programm "min $c^T x$ unter den Nebenbedingungen $Ax \geq b$" gegeben, und sei ε eine beliebig kleine reelle Zahl. Wir bilden dann

$$\vec{\varepsilon} = \begin{pmatrix} \varepsilon \\ \varepsilon^2 \\ \vdots \\ \varepsilon^m \end{pmatrix},$$

und betrachten die Nebenbedingungen

$$Ax \geq b + \vec{\varepsilon}.$$

Wir müssen nun zeigen, daß das so entstandene Programm nicht degeneriert sein kann. Dies bestätigt das folgende Lemma.

Lemma 1 *Ist A eine $m \times d$ Matrix vom Rang d, und $b \in \mathbb{R}^m$, dann gilt die folgende Aussage:*

$$\exists \delta > 0 : \forall \varepsilon \in \,]0, \delta[\, : (A, b + \vec{\varepsilon}) \quad \textit{ist nicht degeneriert.}$$

Beweis Zum Beweis dieser Aussage verwenden wir Proposition 1. Seien also B und B' voneinander verschiedene Basen, dann ist zu zeigen:

$$A_B^{-1}(b_B + \vec{\varepsilon}_B) \neq A_{B'}^{-1}(b'_{B'} + \vec{\varepsilon}_{B'}).$$

Beide Seiten dieser Ungleichungen sind Vektoren, deren Einträge Polynome in ε sind. Wir können dies in der Form

$$A_B^{-1}(b_B + \vec{\varepsilon}_B) = \begin{pmatrix} p_1(\varepsilon) \\ p_2(\varepsilon) \\ \vdots \\ p_m(\varepsilon) \end{pmatrix} \quad \text{und} \quad A_{B'}^{-1}(b'_{B'} + \vec{\varepsilon}_{B'}) = \begin{pmatrix} q_1(\varepsilon) \\ q_2(\varepsilon) \\ \vdots \\ q_m(\varepsilon) \end{pmatrix}$$

schreiben. Da sich die Basen B und B' in mindestens einem Element unterscheiden, gibt es auch mindestens eine Potenz von ε, die zwar unter den Polynomen q_i, aber in *keinem* der Polynome p_j vorkommt. Wir können also ohne Einschränkung annehmen, daß die Polynome p_1 und q_1 nicht identisch sind; $p_1 - q_1$ ist also nicht das Nullpolynom. Somit existiert eine positive reelle Zahl $\delta_{BB'}$ mit der Eigenschaft, daß $p_1 - q_1$ im Interval $]0, \delta_{BB'}[$ keine Nullstelle hat; d.h.,

$$\forall \varepsilon \in]0, \delta_{BB'}[\, : \; p_1(\varepsilon) - q_1(\varepsilon) \neq 0 \Longrightarrow A_B^{-1}(b_B + \vec{\varepsilon}_B) \neq A_{B'}^{-1}(b'_{B'} + \vec{\varepsilon}_{B'}).$$

Wir haben nun gezeigt, daß wir für jedes Paar von Basen B und B' eine Schranke $\delta_{BB'}$ angeben können, die die gewünschte Eigenschaft besitzt. Da es nur endlich viele verschiedene Basen gibt, erhalten wir auch nur endlich viele solcher Werte und können unter diesen das Minimum δ auswählen. Damit ist die Aussage des Lemmas gezeigt.

$\square$

Wie muß man den Parameter ε wählen, um ein konkretes Lineares Programm mit dem Simplex–Algorithmus zu lösen ? Die Basislösungen sind nun von der Form

$$x = A_B^{-1}(b_B + \vec{\varepsilon}_B) = x(\varepsilon) = (x_1(\varepsilon), \ldots, x_d(\varepsilon));$$

ihre Komponenten sind also Polynome in ε, und wir erhalten

$$\hat{x} = x(\varepsilon) + \lambda a. \tag{1.4}$$

Der Vektor a in dieser Gleichung, der auch bei der Konstruktion von λ in (1.2) benötigt wird, entsteht aus der Matrix A_B^{-1} und ist somit unabhängig von ε. Nun müssen wir λ aus einer Menge von Werten

$$\frac{(b_j + \varepsilon^j) - a_j x(\varepsilon)}{a_j a}$$

auswählen, d.h., wir müssen das Minimum aus einer Menge von Polynomen in ε auswählen. Die folgende Proposition gibt uns die Möglichkeit dies zu tun, ohne die Polynome explizit auszuwerten.

Proposition 2 *Gegeben seien zwei Polynome*

$$\begin{aligned} p(X) &= p_0 + p_1 X + \cdots p_k X^k \\ \textit{und } q(X) &= q_0 + q_1 X + \cdots q_l X^l. \end{aligned}$$

Dann gelten die folgenden Aussagen:

$$\begin{aligned} \exists \delta > 0 : \forall \varepsilon \in]0, \delta[: p(\varepsilon) = q(\varepsilon) &\Rightarrow p = q \\ \exists \delta > 0 : \forall \varepsilon \in]0, \delta[: p(\varepsilon) > q(\varepsilon) &\Leftrightarrow (p_0, p_1, p_2, \ldots, p_k) \textit{ ist lexikographisch größer als } (q_0, q_1, q_2, \ldots, q_l). \end{aligned}$$

Wir können die Polynome in Λ also vergleichen, indem wir die lexikographische Ordnung auf den zugehörigen Koeffizientenvektoren betrachten. Wir erhalten dann $\lambda = \lambda(\varepsilon)$ als Polynom in ε und somit gilt auch:

$$\hat{x} = \hat{x}(\varepsilon) = x(\varepsilon) + \lambda(\varepsilon)a. \tag{1.5}$$

In jedem Schritt des Simplex–Algorithmus rechnen wir nun also mit Polynomen statt mit skalaren Werten, und führen ansonsten dieselben Berechnungen durch wie zuvor.

1.2.7 Der Zwei–Phasen Simplex–Algorithmus

Wir haben bisher nur beschrieben, wie wir eine Basislösung eines Linearen Programmes in eine weitere überführen können. Um den Simplex–Algorithmus vollständig zu beschreiben müssen wir nun noch zeigen, wie wir eine Basislösung bestimmen können, von der aus wir die Simplex–Schritte iterieren können. Wir haben bereits in Satz 2 gesehen, daß es dann genügt, eine beliebige zulässige Lösung zu kennen; aus dieser können wir dann eine Basislösung konstruieren. Sind die Nebenbedingungen

$$Ax \geq b$$

gegeben, so führen wir zunächst eine neue Variable u ein, und betrachten das Lineare Programm:
minimiere u unter den Nebenbedingungen $(x, u) \geq 0$ und

$$\begin{array}{ccccccccc} a_{11}x_1 & + & \cdots & + & a_{1d}x_d & + & u & \geq & b_1 \\ & \vdots & & \vdots & & & & \vdots & \\ a_{m1}x_1 & + & \cdots & + & a_{md}x_d & + & u & \geq & b_m. \end{array}$$

Man kann leicht verifizieren, daß $(0, \ldots, 0, \overline{b})$ eine zulässige Lösung dieses Systems ist, wenn man $\overline{b}$ als das Maximum der $\{b_1, \ldots, b_m\}$ wählt. Zu dieser Lösung berechnen wir die zugehörige Basislösung gemäß der Konstruktion im Beweis von Satz 2 und starten damit den Simplex–Algorithmus. Dieser wird eine optimale Basislösung (x^*, u^*) berechnen. Ist darin $u^* > 0$, so kann das ursprüngliche System $Ax \geq b$ keine zulässige Lösung besitzen. Ist hingegebn $u^* = 0$, so ist x^* sogar eine zulässige Basislösung von $Ax \geq b$. Damit können wir dann auch dieses Lineare Programm mit dem Simplex–Algorithmus lösen. Diese Verfahren besteht also aus zwei Phasen des Simplex–Algorithmus.

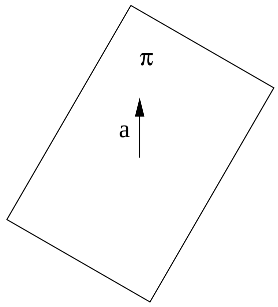
Vorlesung vom 29.4.1996
Patrick Brandmeier, Rainer Brunotte

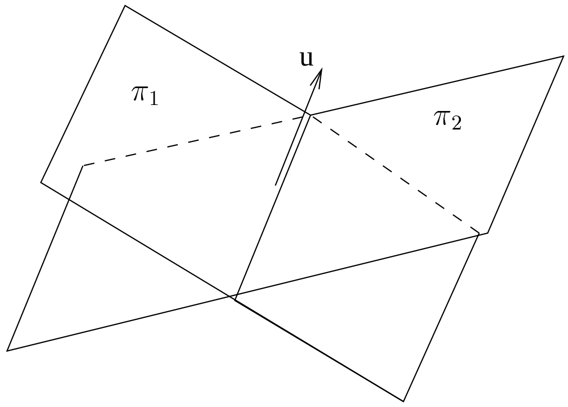
1.2.8 Geometrische Interpretation

Im folgenden betrachten wir die geometrische Interpretation des Simplex-Algorithmus. Der Simplex-Algorithmus startet mit einer zulässigen Basislösung, d.h. mit einem zulässigen Punkt, der Schnittpunkt von d Hyperebenen ist. Falls man nun aus den Normalenvektoren auf diesen d Hyperebenen den Richtungsvektor c nicht-negativ linear kombinieren kann, dann bilden die d Hyperebenen auch eine V-Kombination und die Basislösung ist sogar optimal. Die Skalare sind dabei genau die y_i, $i \in B$, mit $y_B^T = c^T A_B^{-1}$, siehe Beweis vom Satz der starken Dualität. Falls man, um den Richtungsvektor c zu kombinieren, negative Koeffizienten benötigt, so bilden die Hyperebenen noch keine V-Kombination. Die Basislösung ist nicht optimal, d.h. man kann noch entlang einer Kante herunterrutschen - bis man auf eine begrenzende Hyperebene trifft.
Wie läßt sich die Matrix A_B^{-1} geometrisch deuten? Betrachten wir dazu zunächst mal zwei Vektoren u und v im $\mathbb{R}^3$. Es gilt:

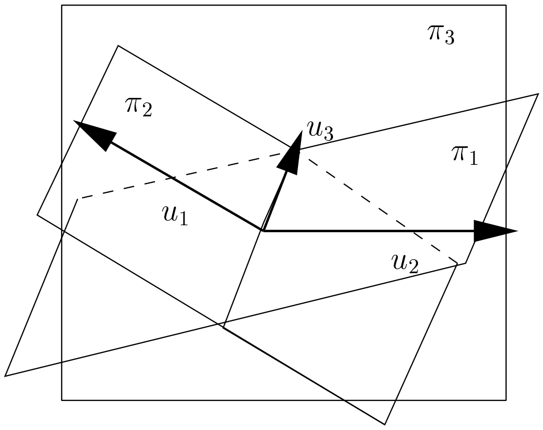
- Für $u, v \in \mathbb{R}^3$ gilt genau dann $u^T \cdot v = (\sum u_i \cdot v_i = 0)$, wenn der Winkel zwischen u und v 90 Grad beträgt.
- Für $u, v \in \mathbb{R}^3$ gilt genau dann $u^T \cdot v = (\sum u_i \cdot v_i > 0)$, wenn der Winkel zwischen u und v kleiner als 90 Grad ist.
- Für $u, v \in \mathbb{R}^3$ gilt genau dann $u^T \cdot v = (\sum u_i \cdot v_i < 0)$, wenn der Winkel zwischen u und v größer als 90 Grad ist.

Wenn wir also eine Ebene $\pi : a^T \cdot x = b_0$ gegeben haben, wobei a der Normalvektor auf der Ebene ist, dann gilt genau dann $a^T \cdot u = 0$, wenn u parallel zu π verläuft.
Dazu die folgenden Zeichnungen :





Wenn man zwei Ebenen π_1 mit Normalenvektor a_1 und π_2 mit Normalenvektor a_2 im $\mathbb{R}^3$ gegeben hat, so ist der Vektor u, der beide Gleichungen erfüllt $(a_1^T u = 0 = a_2^T u)$, Richtungsvektor der Schnittgeraden $\pi_1 \cap \pi_2$. Als nächstes nimmt man noch eine dritte Ebene π_3 mit Normalenvektor a_3 hinzu.



Wenn u_3 die Schnittgerade von π_1 und π_2 ist, u_2 die Schnittgerade von π_1 und π_3 ist und u_1 die Schnittgerade von π_2 und π_3 ist, dann gilt

$$\begin{array}{rclcl} a_1^T \cdot u_2 & = & 0 & = & a_1^T \cdot u_3 \\ a_2^T \cdot u_1 & = & 0 & = & a_2^T \cdot u_3 \\ a_3^T \cdot u_1 & = & 0 & = & a_3^T \cdot u_2. \end{array}$$

Oder anders ausgedrückt:

$$\begin{pmatrix} a_1^T \\ a_2^T \\ a_3^T \end{pmatrix} \cdot \quad \begin{pmatrix} u_1 & u_2 & u_3 \end{pmatrix} = \quad \begin{pmatrix} * & 0 & 0 \\ 0 & * & 0 \\ 0 & 0 & * \end{pmatrix}$$

Die Matrix, die durch die Spaltenvektoren u_1, u_2 und u_3 gebildet wird, ist also das Inverse zur Matrix A. Geometrisch bedeutet das für A_B^{-1}: Faßt man die Normalenvektoren der drei Ebenen zu einer Matrix A_B zusammen und invertiert diese Matrix, so erhält man eine Matrix, deren Einträge die Richtungsvektoren der Schnittgeraden sind.

Um den Simplex-Algorithmus starten zu können, benötigt man eine zulässige Lösung. Um die zu erhalten wendet man das Simplexverfahren auf das Lineare Programm minimiere u unter den Nebenbedingungen

$$\begin{array}{rcl} a_1 \cdot x + u & \geq & b_1 \\ a_2 \cdot x + u & \geq & b_2 \\ & \vdots & \\ a_m \cdot x + u & \geq & b_m \\ u & \geq & 0 \end{array}$$

an, für das man eine zulässige Lösung $(0, \ldots, 0, \bar{b})$ mit $\bar{b} = \max\{b1, \ldots b_m, 0\}$ kennt.
Wenn sich als optimaler Wert der Zielfunktion $u = 0$ ergibt, so ist das dazugehörige x eine zulässige Basislösung für unser Ausgangsproblem. Ansonsten gibt es für das eigentliche Lineare Programm keine zulässige Lösung.
Wie läßt sich dieses Verfahren geometrisch interpretieren?
Durch Hinzunahme der Variable u wird das Problem vom $\mathbb{R}^d$ in den $\mathbb{R}^{d+1}$ verschoben. In diesem Raum kennt man eine zulässige Lösung. Man versucht jetzt, u möglichst gering zu wählen, d.h. möglichst eine Lösung auf der Hyperebene $u = 0$ und damit in unserem ursprünglichen Raum zu erreichen.
Für $d = 2$, könnte eine Halbebene durch folgende Ungleichung gegeben sein: $2x_1 + 3x_2 \geq 2$. Diese Halbebene wird jetzt durch Hinzunahme einer weiteren Variable u zu einem Halbraum im $\mathbb{R}^3$: $2x_1 + 3x_2 + u \geq 2$, bzw. $u \geq 2 - 2x_1 - 3x_2$, siehe Abbildung 1.2.8.

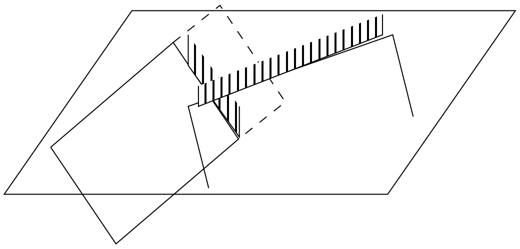


Abbildung 1.16: zwei Halbebenen, die durch Hinzunahme von u zu Halbräumen im $\mathbb{R}^3$ werden

1.2.9 Algebraischer Simplexalgorithmus

Bis jetzt haben wir uns überlegt, wie man unter den Nebenbedingungen $Ax \geq b$ das Optimierungsproblem $\min c^T x$ geometrisch lösen kann. Im folgenden wollen wir uns nun damit beschäftigen, einen algebraischen

Simplex-Algorithmus zu entwickeln, der das Problem löst. Dieser algebraische Algorithmus wird das duale Problem $\max z = y^T b$ unter den Nebenbedingungen $y^T A = c^T$, $y \geq 0$ (und damit nach dem Satz der starken Dualität auch das primale Problem) lösen. Dabei geht man von einer dual zulässigen Lösung aus und verbessert diese mit jedem Simplex-Schritt, indem man eine Hyperebene gegen eine andere austauscht und somit den dual zulässigen Punkt nach oben schiebt. Die Simplexschritte werden so lange durchgeführt, bis z maximal ist.
Zunächst schreiben wir das duale Problem (zur allgemeinen Verwirrung) anders auf:

$$\begin{aligned}
&\max y^T b, \quad y^T A = c^T, \quad y \geq 0 \\
&\Leftrightarrow \quad \max b^T y, \quad A^T y = c, \quad y \geq 0 \\
&\Leftrightarrow \quad \max c^T x, \quad Ax = b, \quad x \geq 0 \quad \text{(durch Umbenennen).}
\end{aligned}$$

Das zu lösende Lineare Programm hat also folgende Form:

$$\max z = c_1 \cdot x_1 + c_2 \cdot x_2 + \ldots + c_n \cdot x_n$$

$$\begin{aligned}
a_{11} \cdot x_1 + a_{12} \cdot x_2 + \ldots + a_{1n} \cdot x_n &= b_1 \\
a_{21} \cdot x_1 + a_{22} \cdot x_2 + \ldots + a_{2n} \cdot x_n &= b_2 \\
&\vdots \\
a_{m1} \cdot x_1 + a_{m2} \cdot x_2 + \ldots + a_{mn} \cdot x_n &= b_m \\
x_i &\geq 0
\end{aligned}$$

Der Algorithmus wird, bevor er formal aufgeschrieben wird, erst mal an einem Beispiel verdeutlicht:

Beispiel maximiere $z = 5x_1 + 4x_2 + 3x_3$, so daß $x_i \geq 0$ und

$$\begin{aligned}
2x_1 + 3x_2 + x_3 + x_4 &= 5 \\
4x_1 + x_2 + 2x_3 + x_5 &= 11 \\
3x_1 + 4x_2 + 2x_3 + x_6 &= 8.
\end{aligned}$$

Bilde ein Wörterbuch zu diesem Linearen Programm:

$$\begin{array}{rcl}
x_4 &=& 5 - 2x_1 - 3x_2 - x_3 \\
x_5 &=& 11 - 4x_1 - x_2 - 2x_3 \\
x_6 &=& 8 - 3x_1 - 4x_2 + 3x_3 \\
\hline
z &=& 5x_1 + 4x_2 + 3x_3
\end{array}$$

An dem Wörterbuch kann man ablesen, daß $B = \{4, 5, 6\}$ eine Basis ist und $x_4 = 5$, $x_5 = 11$, $x_6 = 8$ und $x_1 = x_2 = x_3 = 0$ eine zulässige Lösung ist mit $z = 0$.
Um z zu verbessern, also zu erhöhen, könnte man nun x_1, x_2, oder x_3 erhöhen. Wir entscheiden uns für x_1, da x_1 den größten Koeffizienten hat und so durch Erhöhung von x_1 der Zielvektor am schnellsten erhöht werden kann. Das Wörterbuch besagt, daß sich mit x_1 auch x_4, x_5 und x_6 ändern müssen; und zwar werden x_4, x_5 und x_6 mit Erhöhung von x_1 erniedrigt. Um wieviel darf man nun also x_1 erhöhen, ohne daß x_4, x_5 oder x_6 negativ werden? Antwort: x_1 darf nicht größer als $5/2$ werden, denn für $x_1 > 5/2$ gilt $x_4 < 0$ im Widerspruch zur Voraussetzung, daß alle x_i größer als 0 sein müssen. Damit ergibt sich als neue, bessere Lösung:

$$\begin{array}{rcl}
x_1 &=& 5/2 \\
x_2 = x_3 &=& 0 \\
x_4 &=& 0 \\
x_5 &=& 1 \\
x_6 &=& 1/2 \\
\hline
\end{array}$$

$$z = 25/2$$

Wir wollen jetzt zu dieser Lösung noch das neue Wörterbuch konstruieren. Das neue Wörterbuch soll die Basis $\{1,5,6\}$ haben. Deshalb lösen wir die Gleichungen entsprechend auf. x_1 ist also eintretende Variable, x_4 ist austretende Variable.

$$\begin{array}{rcl} x_1 &=& 5/2 - 3/2x_2 - 1/2x_3 - 1/2x_4 \\ x_5 &=& 11 - 4(5/2 - 3/2x_2 - 1/2x_3 - 1/2x_4) - x_2 - 2x_3 \\ x_6 &=& 8 - 3\cdot(5/2 - 3/2x_2 - 1/2x_3 - 1/2x_4) - 4x_2 - 2x_3 \\ \hline z &=& 5\cdot(5/2 - 3/2x_2 - 1/2x_3 - 1/2x_4) + 4x_2 + 3x_3 \end{array}$$

Durch Umformungen erhält man das Wörterbuch:

$$\begin{array}{rcl} x_1 &=& 5/2 - 3/2x_2 - 1/2x_3 - 1/2x_4 \\ x_5 &=& 1 + 5x_2 + 2x_4 \\ x_6 &=& 1/2 + 1/2x_2 - 1/2x_3 + 3/2x_4 \\ \hline z &=& 25/2 - 7/2x_2 + 1/2x_3 - 5/2x_4 \end{array}$$

In einem weiteren Schritt soll z erneut verbessert d.h. vergrößert werden. x_2, x_3 und x_4 können dazu verändert werden. Da x_2 und x_4 negative Koeffizienten haben, kann der Zielvektor nur durch Vergrößern von x_3 verbessert werden.
Die schärfste Bedingung für x_3 liefert die 3. Zeile mit $x_3 \leq 1$. Daraus folgt, daß x_3 in die Basis eingeht und x_6 die Basis verläßt.
Damit ergibt sich als neue, bessere Lösung:

$$\begin{array}{rcl} x_1 &=& 2 \\ x_2 &=& 0 \\ x_3 &=& 1 \\ x_4 &=& 0 \\ x_5 &=& 1 \\ x_6 &=& 0 \\ \hline z &=& 25/2 \end{array}$$

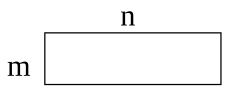
Das dazugehörige neue Wörterbuch lautet:

$$\begin{array}{rcl} x_3 &=& 1 + x_2 + 3x_4 - 2x_6 \\ x_5 &=& 1 + 5x_2 + 2x_4 \\ x_1 &=& 2 - 2x_2 + 2x_4 - x_6 \\ \hline z &=& 13 - 3x_2 - x_4 - x_6 \end{array}$$

Alle Koeffizienten der Zielfunktion sind negativ. Da x_2, x_4 und x_6 nur erhöht werden dürfen, ist z nicht mehr verbesserungsfähig. Die bisher gefundene Lösung ist optimal.

Der gerade an einem Beispiel vorgeführte Algorithmus wird nun formal aufgeschrieben.
Sei dazu wieder $\max c^T x$, $Ax = b$, $x \geq 0$ das Lineare Programm. Die Matrix A hat folgende Gestalt:
Wir nehmen an, daß wir bereits eine Basislösung x mit zugehöriger Basis B kennen. Für höchstens m Komponenten von x gilt also $x_i > 0$. Alle übrigen $n - m$ Komponenten sind gleich 0. Bilde zu dieser Basislösung ein Wörterbuch:

A:



$$\begin{aligned} x_i &= \beta_i + \sum_{j \notin B} a_{ij} \cdot x_j \qquad \forall i \in B, \qquad \beta_i \geq 0 \\ z &= v + \sum_{j \notin B} c_j x_j \end{aligned}$$

Die Lösung ist dann $x_i = \beta_i$ für $i \in B$ und $x_j = 0$ für $j \notin B$.

Simplex
1) Fall 1: $c_j < 0$ für alle $j \notin B$
2) Lösung ist optimal
3) Fall 2: es existiert ein $j^* \in B$ mit $c_{j^*} > 0$
4) Unterfall 1: $\forall i \in B : a_{i,j^*} \geq 0$
5) Das Problem ist unbeschränkt, x_{j^*} kann beliebig groß werden.
6) Unterfall 2: es existiert ein $i \in B$ mit $a_{i,j^*} < 0$
7) Setze $\mu := \min\{-\beta_i / a_{i,j^*} \mid a_{i,j^*} < 0,\ i \in B\} \geq 0$
Dieses Minimum werde durch i^* erreicht.
8) Bilde neue Basis $\hat{B} = B \setminus \{i^*\} \cup \{j^*\}$
9) Bilde neues Wörterbuch für Basis $\hat{B}$
(Isoliere dazu x_{j^*} aus $x_{i^*} = \beta_{i^*} + \sum_{j \notin B} a_{i^*,j} x_j$ und setze ein)
10) Mache nächsten Simplex- Schritt

Vorlesung vom 2.5.1996
Sven Jacobi

Wir wissen jetzt, wie der algebraische Simplex-Algorithmus eine Iteration durchführt. Es sind aber noch Fragen offen:

1. Wie bekommt man aus dem alten ein neues Wörterbuch?
2. Wie erhält man eine Basislösung und damit ein Anfangswörterbuch?
3. Welche Strategie verwendet man bei der Wahl der eintetenden Variable?
4. Welche Strategie verwendet man bei der Wahl der austretenden Variable, damit Zyklen vermieden werden?

Wir wollen im folgenden diese vier Fragen beantworten. Zunächst zu dem Problem, wie man von einem Wörterbuch durch Verändern einer Basisvariable zu einem neuen Wörterbuch kommt. Dafür gibt es zwei Möglichkeiten:

- Durch Einsetzen: Nachdem man die Variable x_{j^*}, die in die Basis eintreten soll, bestimmt hat, löst man die Zeile des Wörterbuches (für die $\mu = \min\{-\frac{\beta_i}{a_{ij^*}} \mid a_{ij^*} < 0,\ i \in B\}$ erreicht wird) nach x_{j^*} auf. Somit hat man eine Zeile des neuen Wörterbuches, und zwar die mit der Darstellung der neuen Basisvariable. Nun muß man noch im alten Wörterbuch in allen anderen Zeilen und in der Zielfunktion x_{j^*} durch den gerade bestimmten Term ersetzen, damit diese Variable nicht mehr im neuen Wörterbuch auf der rechten Seite vorkommt.

- Durch Zeilenoperationen: Auch hier bestimmt man zuerst die eintretende Variable x_{j^*} und die austretende Variable x_{i^*}. Nun addiert man zu jeder Zeile, in der x_{i^*} nicht als Variable auftritt, ein Vielfaches der Zeile, in der x_{i^*} als Variable auftritt, hinzu. Das Vielfache muß dabei so gewählt werden, daß x_{j^*} in der neu erzeugten Zeile verschwindet. Das gleiche geschieht mit der Zielfunktion z. Danach formt man die einzelnen Zeilen des Wörterbuch noch nach den in der neuen Basis stehenden Variablen um und ist fertig.

Beispiel Gegeben ist folgendes Wörterbuch:

$$\begin{array}{rcrcrcrcr} x_2 & = & 7 & + & 5x_1 & - & 3x_3 & - & 2x_5 \\ x_4 & = & 2 & + & 2x_1 & - & 2x_3 & + & x_5 \\ x_6 & = & 5 & - & 2x_1 & & & + & 3x_5 \\ \hline z & = & 13 & - & 3x_1 & + & 2x_3 & + & x_5. \end{array}$$

Man sieht, daß man die Zielfunktion z verbessern kann, indem man x_3 in die Basis aufnimmt. Die zweite Zeile stellt die größte Einschränkung dar, also verläßt x_4 die Basis. Um das neue Wörterbuch zu erhalten, formen wir zunächst die zweite Zeile nach x_3 um:

$$x_3 = 1 + x_1 - 1/2x_4 + 1/2x_5.$$

Jetzt ersetzen wir in der ersten und dritten Zeile und in der Zielfunktion x_3 durch den gerade berechneten Term. Zum Beispiel ergibt dies für die erste Zeile:

$$\begin{aligned} x_2 &= 7 + 5x_1 - 3 - 3x_1 + 3/2x_4 - 3/2x_5 - 2x_5 \\ &= 4 + 2x_1 + 3/2x_4 - 7/2x_5. \end{aligned}$$

Wendet man die zweite Methode an, so muß man zu jeder Zeile und zur Zielfunktion ein Vielfaches der zweiten Zeile addieren. Zum Beispiel multipliziert man die zweite Zeile mit $-3/2$ und addiert sie zur ersten Zeile:

$$x_2 - 3/2x_4 = 4 + 2x_1 - 7/2x_5.$$

Dies liefert das gleiche Ergebnis wie die erste Methode.

Kommen wir jetzt zur zweiten Frage, wie man eine Basislösung und damit ein Anfangswörterbuch findet. Dazu formen wir das Lineare Programm $\max c^T x$, $Ax = b$ (ohne Einschränkung nehmen wir $b \geq 0$ an), $x \geq 0$ um und betrachten das Lineare Programm

$$\begin{aligned} \max w &= -z_1 - z_2 - \ldots - z_m \\ &= -\sum b_i + (\sum a_i)x \end{aligned}$$

unter den Nebenbedingungen $x \geq 0$, $z \geq 0$ und

$$\begin{aligned} z_1 &= b_1 - a_1 x \\ z_2 &= b_2 - a_2 x \\ &\vdots \\ z_m &= b_m - a_m x \end{aligned}$$

Für dieses Lineare Programm kennen wir eine Anfangslösung: $z_i = b_i$ und $x_j = 0$ für alle i und j. Also können wir ein Anfangswörterbuch erstellen und den Simplex-Algorithmus anwenden (Phase I). Der Algorithmus liefert uns eine optimale Lösunge von Phase I: x^* und z^*. Dabei muß man zwei Fälle unterscheiden:

1. Es existiert ein i derart, daß $z_{i^*} > 0$ ist. Dann gilt für den optimalen Wert der Zielfunktion:

   $$w^* = -\sum z_{i^*} < 0$$

   und damit hat das Lineares Programm $Ax = b$, $x \geq 0$ keine Lösung.

2. Es gilt $z^* = 0$. Daraus folgt $Ax^* = b$ und $x^* \geq 0$. Wir haben also eine Basislösung x^* für unser ursprüngliches Lineares Programm $\max c^T x$, $Ax = b$, $x \geq 0$ gefunden. Durch Zeilenoperationen kann man das Gleichungssystem so umwandeln, daß man ein Wörterbuch für Phase II erhält.

Bleibt noch die Frage, nach welchen Regeln man die eintretenden und austretenden Variablen auswählen soll. Eine Möglichkeit, die eintretende Variable zu wählen, wäre, die mit dem größten Koeffizienten in der Zielfunktion zu nehmen. Also diejenige, die die größte Verbesserung des Zielwerts liefert. Hat man für die austretende Variable mehrere zur Auswahl, so könnte man die mit dem keinsten Index auswählen.

Dieses Verfahren hat den großen Nachteil, daß es zu Zyklen kommen kann, die den Simplex-Algorithmus in eine Endlosschleife schicken. Betrachte dazu folgendes Beispiel:

Beispiel Gegeben ist folgendes Wörterbuch:

$$\begin{array}{rcrcrcrcrcr}
x_5 & = & & - & 1/2x_1 & + & 11/2x_2 & + & 5/2x_3 & - & 9x_4 \\
x_6 & = & & - & 1/2x_1 & + & 3/2x_2 & + & 1/2x_3 & - & x_4 \\
x_7 & = & 1 & - & x_1 & & & & & & \\
\hline
z & = & & & 10x_1 & - & 57x_2 & - & 9x_3 & - & 24x_4
\end{array}$$

Nach unserer Regel wählen wir als eintretende Variable die mit dem größten Koeffizienten in der Zielfunktion: x_1. Als austretende Variable haben wir x_5 oder x_6 zur Auswahl. Wir wählen die mit dem kleinsten Index, also x_5. Unser neues Wörterbuch hat dann die Form:

$$\begin{array}{rcrcrcrcrcr}
x_1 & = & & & 11x_2 & + & 5x_3 & - & 18x_4 & - & 2x_5 \\
x_6 & = & & & -4x_2 & - & 2x_3 & + & 8x_4 & + & x_5 \\
x_7 & = & 1 & - & 11x_2 & - & 5x_3 & + & 18x_4 & + & 2x_5 \\
\hline
z & = & & & 53x_2 & + & 41x_3 & - & 204x_4 & - & 20x_5.
\end{array}$$

Nach sechs weiteren Simplexschritten kommt man wieder zum Anfangstableau des Beispiels zurück, ist also in einem Zyklus gelandet.

Eine Möglichkeit, Zyklen zu vermeiden, haben wir schon im Abschnitt über den geometrischen Simplex-Algorithmus kennengelernt: Perturbation. Bei der Perturbationsmethode addiert man zum Begrenzungsvektor b einen Vektor $\vec{\varepsilon}$ hinzu und rechnet dann nach irgendeinem Verfahren zur Wahl der ein- und austretenden Variablen weiter. Welche Methode man dabei verwendet ist egal, da es nie zu einem Zyklus kommen kann. Man ändert also das Gleichungssystem $Ax = b$ zu $Ax = b + \vec{\varepsilon}$, wobei

$$\vec{\varepsilon} = \begin{pmatrix} \varepsilon \\ \varepsilon^2 \\ \vdots \\ \varepsilon^m \end{pmatrix}$$

ist. Die Einträge im Wörterbuch haben dann die Form

$$x_i = \beta_{i0} + \sum_{j=1}^{m} \beta_{ij}\varepsilon^j + \sum_{j \notin B} a_{ij}x_j.$$

Eine weitere Methode, Zyklen zu vermeiden, stellt die *Regel von Bland* dar. Sowohl beim Verlassen der Basis als auch beim Eintritt, wähle unter den Variablenkandidaten die mit dem kleinstem Index. Wir müssen zeigen, daß damit wirklich Endlosschleifen vermieden werden:

Satz 4 *Mit der Regel von Bland kann es zu keinem Zyklus kommen.*

Beweis Angenommen, es kommt doch zu einem Zyklus. Sei $D_0, D_1, ..., D_k = D_0$ die dazugehörige Folge von Wörterbüchern.
Variablen, die in Basen und Nichtbasen des Zyklus vorkommen, heißen im folgenden *bunt*. Sei t der größte Index, so daß x_t bunt ist.
Es sei D das Wörterbuch mit zugehöriger Basis B, für das x_t aus B austritt und dafür x_s in B eintritt (siehe Abbildung 1.17). Das Wörterbuch hat also die Form

$$\begin{aligned} x_i &= \beta_i + \sum_{j \notin B} a_{ij}x_j \\ z &= v + \sum_{j \notin B} c_j x_j \end{aligned}$$

mit $t \in B$, $s \notin B$, $c_s \geq 0$ und $a_{ts} < 0$ (da x_s in die Basis eintritt).

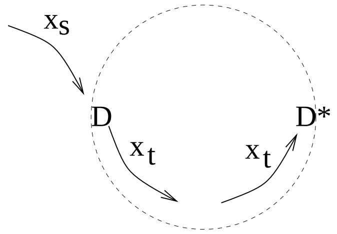


Abbildung 1.17: x_t tritt aus der Basis zum Wörterbuchs D aus und tritt in die Basis zum Wörterbuch D^* wieder ein.

Es sei D^* das zugehörige Wörterbuch mit Basis B^*, für das x_t wieder in die Basis eintritt. Die Zielfunktion z^* hat die Form

$$z^* = v^* + \sum_{j \notin B^*} c_j^* x_j,$$

beziehungsweise

$$z^* = v^* + \sum_j c_j^* x_j \qquad \text{mit } c_j^* = 0 \text{ für alle } j \in B^*.$$

Es gilt $t \notin B^*$, $c_t^* > 0$ und $v = v^*$, da sich der Wert der Zielfunktion während des Zyklus nicht ändert. Außerdem müssen während des Zyklus alle bunten Variablen den Wert Null haben ("β_i"$= 0$), da sie sonst beim Eintreten in die Basis den Wert der Zielfunktion erhöhen würden.

Angenommen, es existiert ein $r \in B$, $r \notin B^*$ mit $c_r^* a_{rs} > 0$. Dann gilt auf jeden Fall $r \neq t$, denn es ist $c_t^* > 0$ und $a_{ts} < 0$. Wir unterscheiden zwei Fälle:

1. $c_r^* > 0$ und $a_{rs} > 0$
   Dann hätte r in die Basis B^* eintreten können. Dies ist jedoch nicht passiert, obwohl $r < t$ gilt. Dieser Fall tritt also nicht ein.

2. $c_r^* < 0$ und $a_{rs} < 0$
   In diesem Fall hätte r die Basis B verlassen können. Dies ist jedoch nicht passiert, obwohl $r < t$ ist. Dieser Fall tritt also auch nicht ein.

Es kann also nicht $c_r^* a_{rs} > 0$ gelten, im Widerspruch zur Voraussetzung.

Wenn wir jetzt noch zeigen können, daß es ein $r \in B$, $r \notin B^*$ gibt mit $c_r^* a_{rs} > 0$, dann haben wir zwei sich gegenseitig ausschließende Aussagen bewiesen und damit einen Widerspruch hergeleitet. Die Annahme, daß die Regel von Bland einen Zyklus erzeugen kann, muß dann falsch gewesen sein und damit wäre der Satz bewiesen.

Die Behauptung ist also, daß es ein $r \in B$, $r \notin B^*$ gibt mit $c_r^* a_{rs} > 0$.
Da während des Zyklus nur degenerierte Simplexschritte ausgeführt werden, gilt

$$z = z^* = v^* + \sum_j c_j^* x_j = v + \sum_j c_j^* x_j$$

mit $c_j^* = 0$ für alle $j \in B^*$. Die Zielfunktion für D^* hat man durch algebraische Operationen aus der Zielfunktion für D erhalten, und daher ist jede Lösung (nicht unbedingt zulässig) für das Wörterbuch D auch eine Lösung für das Wörterbuch D^*. Insbesondere gilt dies für die folgende Lösung für D, wobei y beliebig gewählt werden kann:

$$\begin{aligned} x_s &= y \\ x_i &= \beta_i + a_{is} y \qquad \text{für } i \in B \\ x_j &= 0 \qquad \text{für } j \notin B,\ j \neq s. \end{aligned}$$

Also gilt:

$$v + c_s y = z = v + c_s^* y + \sum_{j \in B} c_j^* (\beta_j + a_{js} y).$$

Daraus folgt

$$(c_s - c_s^* - \sum_{j\in B} c_j^* a_{js})y = \sum_{j\in B} c_j^* \beta_j .$$

Auf der linken Seite hat man ein Polynom in y, auf der rechten Seite eine Konstante stehen. Da die Gleichheit für jeden Wert von y gelten soll, müssen die Ausdrücke $c_s - c_s^* - \sum_{j\in B} c_j^* a_{js}$ und $\sum_{j\in B} c_j^* \beta_j$ gleich Null sein. Nach Voraussetzung ist $c_s > 0$. Für c_s gilt $c_s \leq 0$, denn entweder ist x_s noch in der Basis, dann gilt $c_s = 0$. Oder x_s ist schon wieder aus der Basis ausgetreten. Dann folgt nach Definition von t, daß $s \leq t$ ist und daher gilt $c_s^* \leq 0$. Es gilt also:

$$\sum_{j\in B} c_j^* a_{js} > 0.$$

Es muß daher einen Index $r \in B$ geben mit $c_r^* a_{rs} > 0$.

$\square$

Vorlesung vom 6.5.1996
Jörg Brück, Heiko Schütz, Baré Said

1.2.10 Wie schnell ist der Simplex-Algorithmus

Nachdem wir in den letzten Vorlesungen sichere Verfahren für den Simplex-Algorithmus (Regel von Bland, Perturbationsmethode) kennengelernt haben, werden wir im folgenden dessen Laufzeit untersuchen. Hierbei untersuchen wir sowohl den geometrischen, als auch den algebraischen Simplex-Algorithmus. Diese sind folgendermaßen charakterisiert:

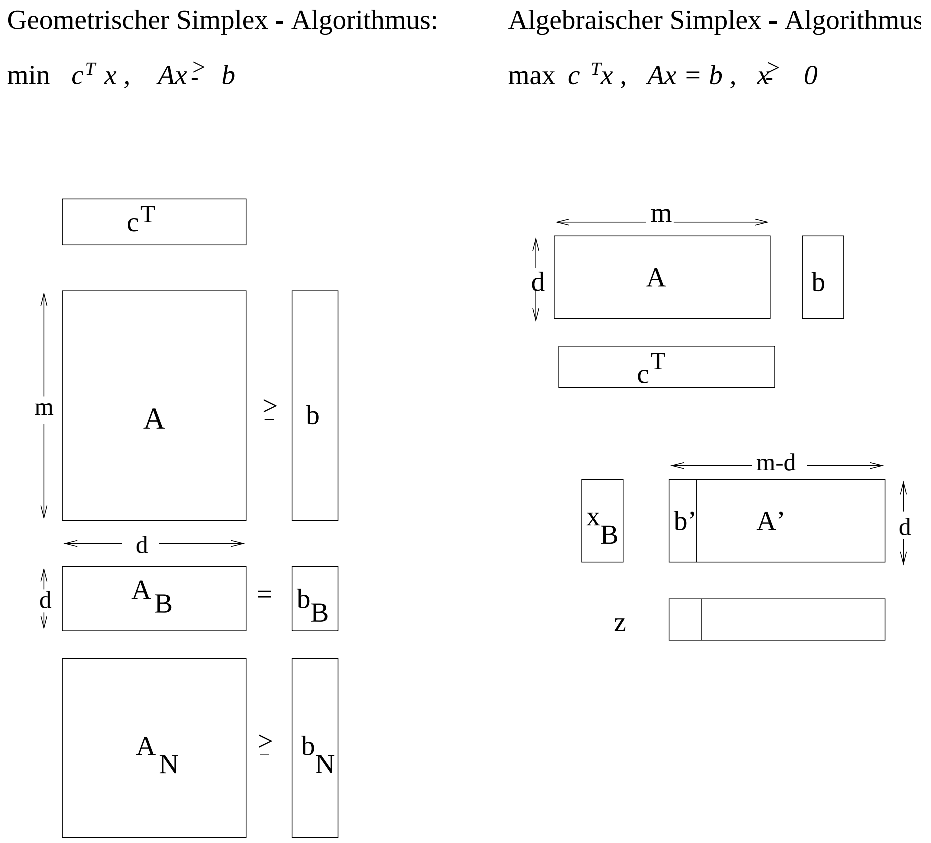


Abbildung 1.18: Geometrischer und algebraischer Simplex-Algorithmus

Bei der Berechnung der Laufzeit stellen sich folgende Fragen:

1. Wie lange braucht ein Simplex-Schritt ?

   (a) Algebraischer Simplex-Schritt:
   Beim algebraischen Simplex-Algorithmus hat man ein Gleichungssystem mit d Gleichungen und m Variablen und eine Zielfunktion, die zu maximieren ist. Um eine optimale Lösung zu finden, geht man folgendermaßen vor: Man wählt sich eine Basis aus und erstellt damit ein Anfangswörterbuch. Durch geeignete Auswahl einer in die Basis eintretenden und einer die Basis verlassenden Variable wird in jedem Simplex-Schritt ein neue Wörterbuch aufgebaut. Es werden so lange neue Wörterbücher aufgebaut, bis die Zielfunktion nur negative Koeffizienten enthält.

      1.) Finde eine Variable, die in die Basis eintritt. Dazu wählt man aus den (höchstens) $m-d$ Koeffizienten in z einen positiven aus. $\Rightarrow O(m-d)$

      2.) Finde eine Variable, die aus der Basis austritt. Dazu untersucht man die d Gleichungen des Wörterbuchs. $\Rightarrow O(d)$

      3.) Bilde neues Wörterbuch. Es werden höchstens $O(m-d)$ Operationen je Zeile ausgeführt.Das ergibt bei d Zeilen eine Laufzeit von $O(d \cdot (m-d))$.

   $\Longrightarrow$ Laufzeit für einen Simplex-Schritt: $O(d \cdot (m-d))$.

   (b) Geometrischer Simplex-Schritt:
   Beim geometrischen Simplex-Algorithmus betrachten wir folgendes Problem: minimiere $c^T x$ unter der Nebenbedingung $Ax \geq b$, wobei A eine $m \times d$ Matrix ist. Auch hier müssen wir einen Initialisierungsschritt machen, bevor wir mit dem ersten Simplex-Schritt starten können. In dem Initialisierungsschritt bestimmt man eine Anfangslösung x_0 sowie eine Basis A_B. Dabei besteht die Basis A_B aus d Ungleichungen und x_0 ist ein zulässiger Punkt, der die Unleichungen aus A_B mit Gleichheit erfüllt (siehe Abbildung 1.18).

   Ein Simplex-Schritt des geometrischen Simplex-Algorithmus besteht dann aus folgenden Phasen:

      1.) Finde die Ungleichung, die aus der Basis austritt. Berechne dazu $y_B^T = c^T A_B^{-1}$ und wähle ein $i \in B$ mit $y_i < 0$. Es sei a die i-te Spalte von A_B^{-1}. Es ist offensichtlich, daß dieser Subschritt $O(d^2)$ Operationen benötigt.

      2.) Finde die Ungleichung, die in die Basis eintritt. Um diese Ungleichung zu finden, bestimmt man $\lambda = \min\{\frac{b_j - a_j x_0}{a_j a} \mid j \notin B,\ a_j a \leq 0\}$, realisiert durch j^*. Falls die Menge nicht leer ist und $\lambda > 0$ ist, ist $x := x_0 + \lambda a$ eine zulässige, bessere Basislösung. Bilde neue Basis $B := B \backslash \{b_i\} \cup \{b_{j^*}\}$ und setze $x_0 := x$. Falls die Menge leer ist, ist das Lineare Programm unbeschränkt; falls $\lambda = 0$ ist, hat man die optimale Lösung gefunden.
      Dieser Subschritt kostet $O(d(m-d))$ Zeit, da wir das Minimum aus $m-d$ Elementen bestimmen müssen.

      3.) Bilde A_B^{-1}. Um die inverse Matrix zu bilden, braucht man normalerweise $O(d^3)$ Operationen. Da sich aber die neue Matrix von der alten Matrix nur in einer Spalte unterscheidet, kann man sich viele Operationen ersparen und die Kosten auf $O(d^2)$ reduzieren.

   Insgesamt braucht also auch hier ein Simplexschritt die Zeit $O(d(m-d))$.

2. Wieviele Simplex-Schritte braucht man ?

   Schon G. B. Dantzig stellte fest, daß für $m < 50$ und $m+d < 200$ gewöhnlich weniger als $3m/2$ Iterationen ausreichten und nur selten bis zu $3m$ Simplexschritte benötigt wurden. Diese Beobachtung stimmt mit empirischen Erfahrungen überein, die bei wesentlich größeren Aufgaben beobachtet wurden. Als "halbempirisch" ist eine Laufzeit von $O(m \cdot \log d)$ anzusehen, die jedoch noch nicht bewiesen werden konnte.

   Machen wir uns die einzelnen Simplex-Schritte zunächst an einem Beispiel klar:

   Beispiel Betrachte das Lineare Programm $\max x_1 + x_2 + x_3$ mit den Nebenbedingungen $0 \leq x_i \leq 1$, $i = 1, 2, 3$.

   Der durch diese Ungleichungen aufgespannte Ergebnisraum stellt einen Polyeder dar. Die Ecken des Polyeders sind jeweils Basislösungen. Wir beginnen mit der Basislösung $x_1 = 0, x_2 = 0, x_3 = 0$. Ausgehend von einer Basislösung führen wir einen Simplex-Schritt aus.Dieser entspricht graphisch dem Entlanggehen an einer Kante des Polyeders. Nach zwei weiteren Simplex-Schritten erreichen wir die Optimallösung $x_1 = 1, x_2 = 1, x_3 = 1$.

   Im folgenden sei:

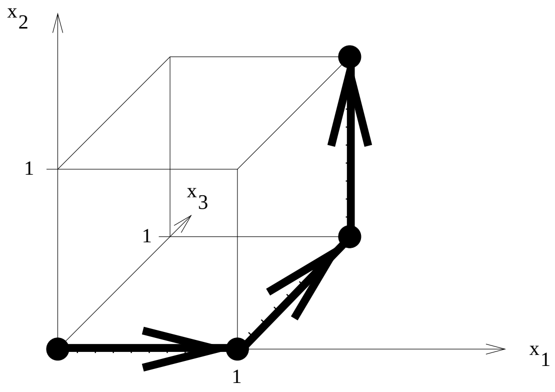


Abbildung 1.19: Beispiel für das Entlanggehen an den Kanten eines Polyeders

$\#\mathrm{B}(A,b)$ $\equiv$ Anzahl der Basen von $A \cdot x \geq b$

$\#\mathrm{BL}(A,b)$ $\equiv$ Anzahl der Basislösungen von $A \cdot x \geq b$
$\equiv$ "Ecken des Polyeders "

Diese beiden Terme sind obere Schranken für die Anzahl der Simplex-Schritte, falls das Gleichungssystemn nicht degeneriert ist (sonst kann es sein, daß der Simplex-Algorithmus unendlich viele Schritte ausführt. Die Anzahl der Simplexschritte ist höchstens gleich $\#\mathrm{B}(a,b)$ und auch höchstens gleich $\#\mathrm{BL}(A,b)$. Wie groß können nun diese beiden oberen Schranken werden? Darauf gibt das 'Upper Bound Theorem' von McMullen (1970) eine Antwort:

Wenn A eine $m \times d$ Matrix ist, so gilt:

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

Da diese Grenzen auch angenommen werden, haben wir bis jetzt nur sehr große obere Schranken gefunden.

Man vermutet, daß man bei geschickter Wahl der eintretenden und austretenden Ungleichungen mit $\leq m - d$ Simplex-Schritten auskommt, um von einer Basislösung zur optimalen Lösung zu gelangen:

Monotone Hirschvermutung (1957)
Es seien $A \cdot x \geq b$ die Nebendingungen sind, unter denen $c^T x$ maximiert werden soll. Sei weiterhin $\underline{x}$ die Basislösung, die $c^T \cdot x$ minimiert und $\bar{x}$ die Basislösung, die $c^T \cdot x$ maximiert. Dann gibt es eine Folge von $\leq m - d$ Simplexschritte, die $\underline{x}$ in $\bar{x}$ überführt.
Diese Vermutung konnte bis jetzt noch nicht bewiesen werden und ist eines der großen offenen Probleme in der Optimierung.

Man kennt bisher noch kein Verfahren zur Auswahl der ein- und austretenden Variablen von dem man nachweisen kann, daß es auch im worst-case polynomielle Laufzeit hat. Man weiß noch nicht einmal, ob es so eins überhaupt geben kann. Für einige Algorithmen, die für nahezu alle praktischen Probleme erfahrungsgemäß eine ganz gute Laufzeit haben, kann man Lineare Programme konstruieren, deren Lösung exponentielle Laufzeit erfordert.

Betrachten wir als Beispiel dazu im Folgenden den geometrischen Simplex-Algorithmus zusammen mit der Regel von Bland: Wenn mehrere Ungleichungen die Basis verlassen können, wähle die mit kleinstem Index.

Betrachte folgendes Lineare Programm (Klee Minty Würfel, 1972):
maximiere x_d unter den Nebenbedingungen

$$\begin{aligned} 0 &\leq x_1 \leq 1 \\ \epsilon \cdot x_{j-1} &\leq x_j \leq 1 - \epsilon \cdot x_{j-1} \end{aligned}$$

mit $\epsilon \in (0, \frac{1}{2})$ und $j = 2, \cdots, d$, bzw.

$$\begin{aligned} 0 &\leq x_1 \\ x_1 &\leq 1 \\ \epsilon \cdot x_1 &\leq 1 - \epsilon \cdot x_1 \\ &\vdots \end{aligned}$$

Behauptung 1 *Wenn der geometrische Simplex Algorithmus auf den Klee-Minty Würfel angewandt wird und man mit Basis $0 = x_1 = x_2 = \cdots = x_d$ beginnt und dabei Blands Regel verwendet, dann werden $2^d - 1$ Simplexschritte durchgeführt.*

Wenn man 1000 Simplex-Schritte pro Sekunde durchführen kann, braucht man bei 64 Variablen ($d = 64$) 585 Millionen Jahre.

Beispiel

- d=2: Siehe Abbildung 1.20 ($\varepsilon = 1/3$)
  $\max x_2$

  (1) $x_1 \geq 0$
  (2) $1- x_1 \geq 0$
  (3) $x_2 \geq \epsilon \cdot x_1$
  (4) $x_2 \leq 1 - \epsilon \cdot x_1$

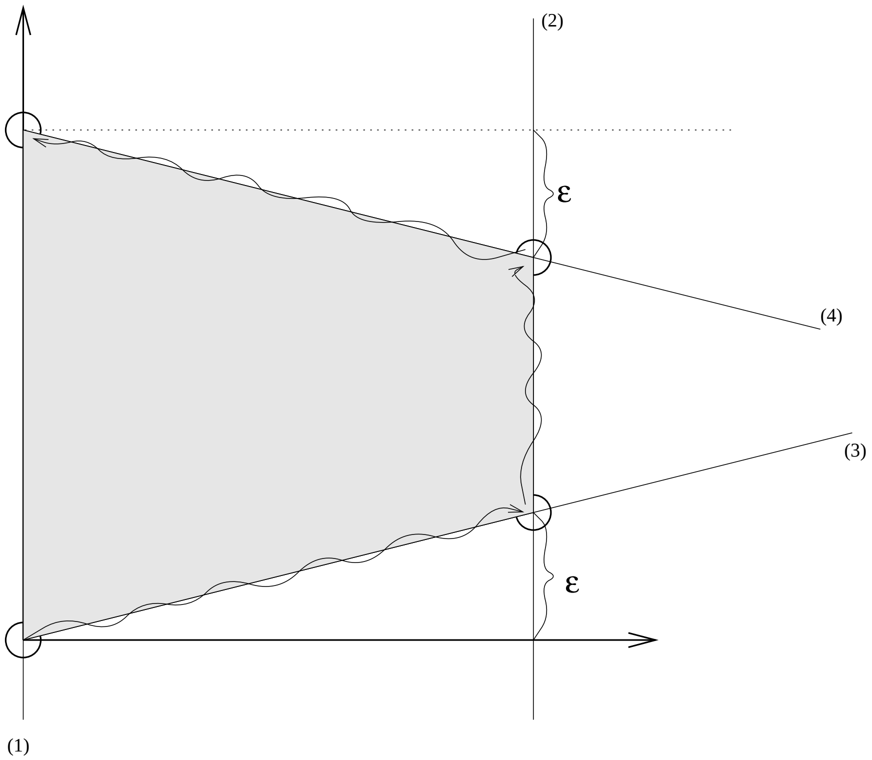


Abbildung 1.20: Maximierungsproblem an einem zweidimensionalen Beispiel

- d=3: Siehe Abbildung 1.21
  $\max x_3$

  (1) $x_1 \geq 0$
  (2) $1- x_1 \geq 0$

(3) $x_2 \geq \epsilon \cdot x_1$
(4) $x_2 \leq 1 - \epsilon \cdot x_1$
(5) $x_3 \geq \epsilon \cdot x_2$
(6) $x_3 \leq 1 - \epsilon \cdot x_2$

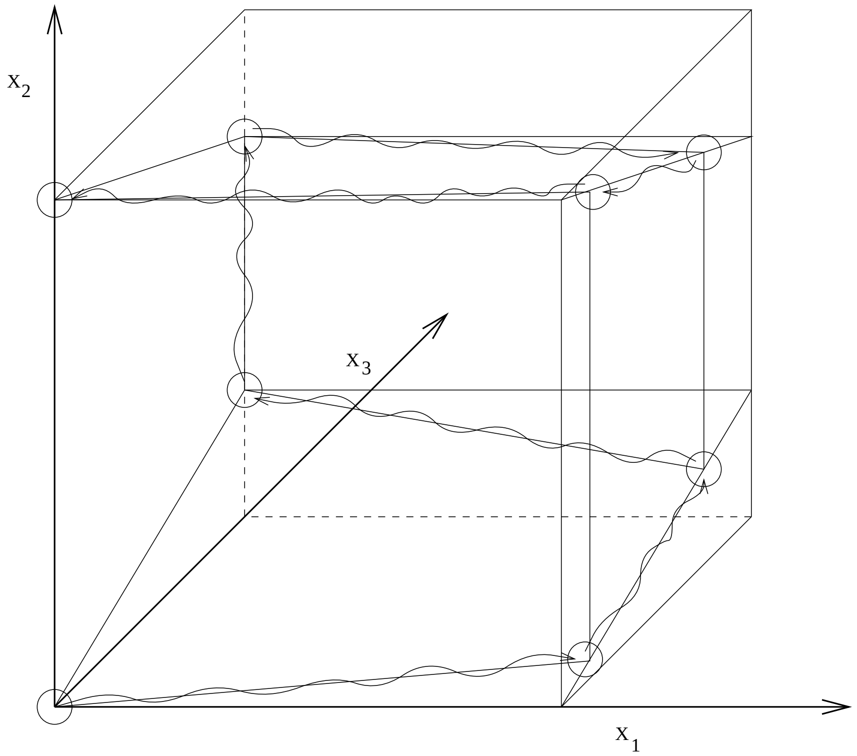


Abbildung 1.21: Maximierungsproblem an einem dreidimensionalen Beispiel

Die Abbildung 1.21 zeigt den von den Gleichungen (1) bis (6) aufgespannten Ergebnisraum. Hierbei stellen die Ecken des entstandenen Polyeders die Basislösungen des Gleichungssystems dar. Wir wollen nun x_3 maximieren. Dazu beginnen wir mit der Basislösung $(1, 3, 5)$. Diese finden wir im Schnittpunkt der angegebenen Gleichungen. Nun machen wir einen Simplexschritt, das heißt wir gehen an einer Kante des Polyeders entlang zur nächsten Basislösung. Wir haben bei $(1, 3, 5)$ nun die Möglichkeit zu $(2, 3, 5)$, zu $(1, 4, 5)$ oder zu $(1, 3, 6)$ zu wechseln. Wir entscheiden mittels Blands Regel und wandern zur Basislösung $(2, 3, 5)$. Hier haben wir zwei Möglichkeiten im Polyeder nach oben zu wandern und x_3 so weiter zu maximieren. Wir wandern weiter zu $(2, 4, 5)$ (Bland) und führen so einen weiteren Simplexschritt durch. Nach fünf weiteren Simplexschritten und dem Passieren der Eckpunkte $(1, 4, 5)$, $(1, 4, 6)$, $(2, 4, 6)$, $(2, 3, 6)$ (in dieser Reihenfolge) erreichen wir die Basislösung $(1, 3, 6)$. Es wurden alle Basislösungen tangiert und x_3 muss somit maximal sein.

Vorlesung vom 9.5.1996
Marc Langenbach, Thomas Martin, Michael Schmidt

1.3 Der Duale Simplex–Algorithmus

Nun betrachten wir eine Variation des Simplex–Algorithmus: wir gehen davon aus, daß wir ein Lineares Programm optimal gelöst haben und ändern nun die Problemstellung ein wenig ab. Man kann dieses neue Problem jetzt natürlich wieder von vorne lösen, oder aber man überlegt sich, wie man die ganze Arbeit nicht noch einmal machen muß. Hierzu ein paar Vorüberlegungen:

Als Ausgangspunkt haben wir wieder folgendes Lineares Programm:

$$\begin{array}{ll} max & c^T x \\ & Ax = b \\ & x \geq 0. \end{array}$$

Wir können dann ein Wörterbuch D mit Basis B erstellen:

$$\begin{array}{rcl} x_i & = & \overline{b_i} + \sum_{j \notin B} \overline{a_{ij}} x_j \quad (i \in B) \\ \hline z & = & v + \sum_{j \notin B} \overline{c_j} x_j . \end{array}$$

Dann haben wir auch eine zulässige Lösung:

$$x_i = \begin{cases} \overline{b_i} & (i \in B) \\ 0 & (i \notin B). \end{cases}$$

Wegen der Nebenbedingung $x \geq 0$ ist dies jedoch nur dann möglich, wenn auch alle $\overline{b_i} \geq 0$ sind. Betrachten wir nun das zugehörige duale Problem:

$$\begin{array}{ll} min & y^T b \\ & y^T A \geq c. \end{array}$$

Das Wörterbuch D ist dual zulässig, wenn alle $\overline{c_j} \leq 0$ sind. Wir wollen nun das dual zulässige Wörterbuch in ein anderes dual zulässiges Wörterbuch überführen, und zwar so, daß v, der aktuelle Wert der Zielfunktion, kleiner wird. Hierzu nehmen wir eine Variable aus der Basis heraus und fügen eine andere wieder hinzu.

DUALER SIMPLEXSCHRITT:

1. Finde ein $i^* \in B$ für das $\overline{b_{i^*}} \leq 0$ gilt. (Falls es kein solches i^* gibt, dann ist die Lösung optimal.)
2. Bestimme

$$\rho = min \left\{ \frac{-\overline{c_j}}{a_{i^* j}} \middle| a_{i^* j} \geq 0 \right\}$$

Dies werde erreicht durch den Index $j^* \notin B$. Falls es allerdings kein solches ρ gibt, dann ist das primale Problem unlösbar. Bilde nun die neue Basis $\hat{B} = (B \setminus \{i^*\}) \cup \{j^*\}$ und das zugehörige Wörterbuch.

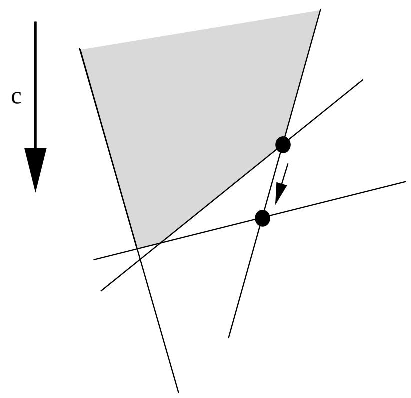


Abbildung 1.22: Verbesserung der V-Kombination beim Dual-Simplex-Schritt

Das heißt: momentan sind alle $\overline{c_j}$ kleiner oder gleich 0. Falls jetzt alle $\overline{b_i}$ größer oder gleich 0 sind, dann ist die Lösung sowohl primal als auch dual zulässig, also optimal. Andernfalls suchen wir einen Index i, so daß $\overline{b_i}$ kleiner als 0 ist und verwenden die entsprechende Zeile, um ein x_j in der Zielfunktion zu eliminieren.

Beispiel

$$\begin{array}{rcrcrcrcr} x_1 & = & -\ 4 & + & 3x_2 & - & 11x_4 & + & \mathbf{x_5} \\ x_3 & = & 3 & - & x_2 & + & 3x_4 & - & 2x_5 \\ \hline z & = & 12 & - & 4x_2 & - & x_4 & - & x_5 \end{array}$$

Die Basis ist dann $B = \{1, 3\}$. Die zugehörige Lösung

$$\begin{array}{rcl} x_1 & = & -4 \\ x_3 & = & 3 \\ x_2 = x_4 = x_5 & = & 0 \end{array}$$

ist dual zulässig. x_5 soll nun eliminiert werden. Hierzu wird die erste Zeile zu den übrigen addiert:

$$\begin{array}{rcrcrcrcrcr} x_3 & + & 2x_1 & = & -\ 5 & + & 5x_2 & - & 19x_4 \\ z & + & x_1 & = & 8 & - & x_2 & - & 12x_4, \end{array}$$

und man erhält damit

$$\begin{array}{rcrcrcrcr} x_3 & = & -\ 5 & - & 2x_1 & + & 5x_2 & - & 19x_4 \\ x_5 & = & 4 & + & x_1 & - & 3x_2 & + & 11x_4 \\ \hline z & = & 8 & - & x_1 & - & x_2 & - & 12x_4. \end{array}$$

Da in der Zielfunktion alle Koeffizienten negativ sind, ist das Wörterbuch dual zulässig.
Mit $i^* = 1$ und $j^* = 5$ ergibt sich dann die neue Basis $\hat{B} = \{3, 5\}$.

Iterieren wir diesen Simplexschritt, so erhalten wir einen Algorithmus, mit dem wir eine optimale Lösung berechnen können.

DUALER SIMPLEXALGORITHMUS:

1. Finde ein dual zulässiges Wörterbuch.
2. Wiederhole den dualen Simplexschritt, bis er nicht mehr anwendbar ist.

Zu beachten ist, daß auch bei diesem Algorithmus degenerierte Systeme zu einem Zyklus führen können. Abhilfe bieten hierbei die Regel von Bland und die Pertubationsmethode.

1.4 Sensitivitätsanalyse

Die Aufgabenstellung der Sensitivitätsanalyse besteht im Lösen eines leicht veränderten Systems. Ausgehend von einem Linearen Programm, dessen optimale Lösung – falls vorhanden – bereits bestimmt wurde, möchte man ein Problem lösen, das sich vom ursprünglichen nur wenig unterscheidet.
Angenommen, man hat zum Linearen Programm

$$\begin{array}{lll} (LP) & max & c^T x \\ & udN & Ax = b \\ & & x \geq 0 \end{array}$$

eine optimale Lösung x^* gefunden. Zu dieser Lösung gehört ein Wörterbuch D^* und eine Basis B^*, die aufgrund der Berechnung von x^* auch vorliegen. Das neue Problem hat dann die Form

$$\begin{array}{lll} (LP') & max & \tilde{c}^T x \\ & udN & \tilde{A}x = \tilde{b} \\ & & x \geq 0 \end{array}$$

Im folgenden werden nun drei Arten untersucht, in denen sich (LP') von (LP) unterscheiden kann:

1. (LP') besitzt eine neue Zielfunktion,
2. der Begrenzungsvektor $\tilde{b}$ unterscheidet sich von dem des ursprünglichen Programms oder
3. (LP') entsteht aus (LP) durch Hinzunahme einer neuen Nebenbedingung

In allen drei Fällen, die wir nun im einzelnen untersuchen wollen, bleiben die übrigen Komponenten unverändert.

1.4.1 Änderung der Zielfunktion

Da sich nur die Zielfunktion ändert, gestaltet sich (LP') als

$$\begin{array}{ll} max & \tilde{c}^T x \\ udN & Ax = b \\ & x \geq 0 \end{array}$$

In diesem Fall ist x^* immer noch eine primal zulässige Lösung, was leicht einzusehen ist, da sich die Nebenbedingungen nicht änderten. Aus dem Wörterbuch D^* läßt sich nun ein neues für (LP') ableiten, indem alle Zeilen außer der Zielfunktion übernommen werden (wobei sich die Basis nicht ändert). Die Zielfunktion kann durch

$$z = \sum_i \tilde{c} x_i$$

ausgedrückt werden, wobei diejenigen x_i durch die entsprechenden rechten Seiten zu ersetzen sind, die Basisvariable sind. Man erhält ein Wörterbuch $\tilde{D}$, das aus dem schon genannten Grund primal zulässig ist. Hierauf können nun ein oder mehrere Primal–Simplexschritte angewendet werden, bis man eine optimale Lösung erreicht oder das Verfahren wegen Unbeschränktheit o.ä. terminiert.

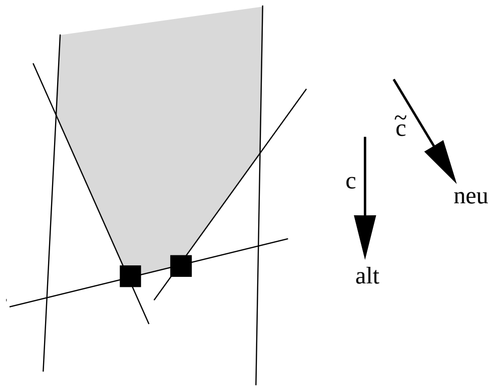


Abbildung 1.23: Änderung der Zielfunktion

Eine geometrische Interpretation für diesen Fall liefert Abbildung 1.23. Durch die Änderung der Zielfunktion ändert sich der Richtungsvektor c, der die Richtung der Optimierung angibt. Ausgehend von der optimalen Lösung x^* – dargestellt durch den unteren Punkt in der Zeichnung – wird der primale Simplex–Algorithmus fortgeführt und somit die Lösung x^* bezüglich $\tilde{c}$ verbessert.

1.4.2 Änderung des Begrenzungsvektors

Ändert sich der Begrenzungsvektor b, kann (LP') dargestellt werden als

$$\begin{array}{ll} max & c^T x \\ udN & Ax = \tilde{b} \\ & x \geq 0 \end{array}$$

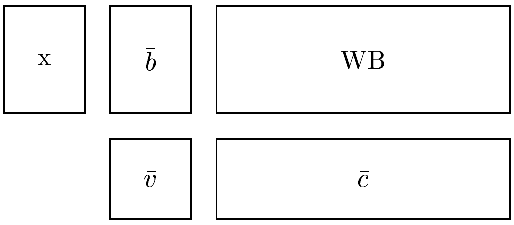


Abbildung 1.24: Wörterbuch D^* zur optimalen Lösung x^*

Anschaulich bedeutet die Änderung des Begrenzungsvektors eine Parallelverschiebung einer oder mehrerer Geraden (vergleiche hierzu Abbildung 1.25).

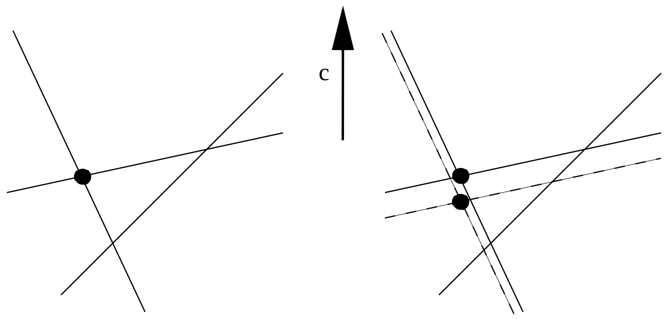


Abbildung 1.25: Parallelverschiebung der Begrenzungsgeraden

Wir haben früher bereits gesehen, daß man die Schritte des Simplex–Algorithmus auch mit Hilfe von Zeilenoperationen des Wörterbuchs darstellen kann (vgl. Seite 22). Vollzieht man diese Zeilenoperationen, ausgehend vom Anfangswörterbuch, nach, so kann man feststellen, daß in Abbildung 1.24 zwar die Koeffizienten von $\bar{b}$ und der Wert von $\bar{v}$ abhängig sind von b, nicht jedoch $\bar{c}$.
Da wir die Basis B^* zur optimalen Lösung von (LP) kennen, können wir uns durch auflösen der Gleichungen nach den Basisvariablen auch ein Wörterbuch für (LP') erstellen. Der Wert für $\bar{c}$ wird sich dadurch nicht verändern. Wir erhalten also ein für (LP') dual zulässiges Wörterbuch und können mit den Schritten des dualen Simplex–Algorithmus fortfahren, um eine optimale Lösoung für das neue Programm zu finden.

Eine andere Möglichkeit, ein Wörterbuch für das neue System (LP') aufzustellen, findet man, wenn man sich anschaut, was beim Simplex–Algorithmus eigentlich geschieht: Die im Anfangswörterbuch vorliegende Basis (siehe Abbildung 1.26) wird im Laufe des Simplexalgorithmus zur Einheitsmatrix umgeformt. Zugleich entsteht ihr Inverses. Multipliziert man nun den neuen Begrenzungsvektor mit dieser inversen Basismatrix, erhält man den benötigten Vektor $\bar{b}$. Mit anderen Worten: das Auflösen der Gleichungen nach gewissen Basisvariablen entspricht genau der Multiplikation der Matrizen und Vektoren mit der entsprechenden inversen Matrix.

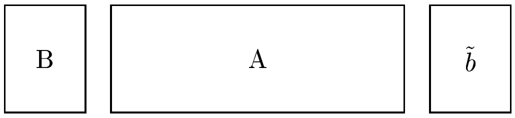


Abbildung 1.26: Wörterbuch zu Beginn des Simplex–Algorithmus

1.4.3 Hinzunahme einer Nebenbedingung

Wir betrachten nun ein Lineares Programm, das durch Hinzunahme einer weiteren Nebenbedingung aus einem bereits gelösten Programm entsteht, und unterscheiden danach, ob diese Nebenbedingung als Gleichung oder Ungleichung gegeben ist.

Hinzunahme einer Ungleichung

Ist die neue Ungleichung in der Form $\hat{a}x \leq \hat{b}$ gegeben, so können wir diese durch Einführen einer weiteren Schlupfvariablen $\hat{x}$ in eine Gleichung überführen:

$$\hat{a}x + \hat{x} \;=\; \hat{b}.$$

In dieser Gleichung müssen wir alle Basisvariablen durch Einsetzen der entsprechenden rechten Seiten im Wörterbuch eliminieren und können danach die Zeile

$$\hat{x} \;=\; \hat{b} - \hat{a}x$$

in das Wörterbuch aufnehmen. Dadurch erhalten wir eine neue Basis $B^* \cup \{\hat{x}\}$. Da sich die Zielfunktion nicht verändert, bleibt das so konstruierte Wörterbuch immer noch *dual* zulässig, und wir können mit dem dualen Simpex–Algorithmus fortfahren.

Hinzunahme einer Gleichung

Jetzt gestaltet sich die Umformung des Wörterbuchs D^* etwas schwieriger, da man die Basis nicht so leicht erweitern kann wie im letzten Fall. Sei $\hat{a}x = \hat{b}$ die Gleichung, die zu $\tilde{A}$ neu hinzugekommen ist. In einem ersten Schritt werden nun alle Basisvariablen aus dieser entfernt (wieder durch Einsetzen der jeweilgen rechten Seiten des Wörterbuchs). Man erhält das System aus Abbildung 1.27.

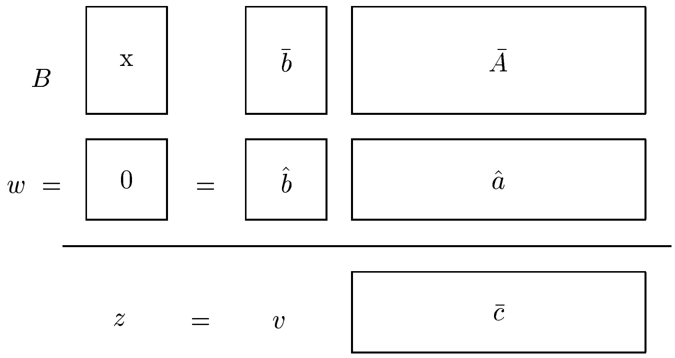


Abbildung 1.27: Hinzunahme einer neuen Gleichung

An der Stelle von w sollte aber eine Basisvariable stehen. Man müßte also eine Nicht–Basisvariable isolieren, auf die linke Seite bringen und sie dann noch aus den verbleibenden Gleichungen eliminieren.
Dazu geht man wie folgt vor: Unter der Annahme, die aktuelle Basis sei $B \cup \{w\}$, führt man einen Simplexschritt aus, bei dem w die Basis verläßt. Die somit auf der rechten Seite entstehende w-Spalte wird gestrichen. Da auch in dieser Situation nichts an der Zeile für die Kostenfunktion geändert wird, erhalten wir wieder ein dual zulässiges Wörterbuch, auf das wir den dualen Simplex–Algorithmus anwenden können.
Veranschaulichen wir auch hier wieder, was die Hinzunahme der Nebenbedingung bedeutet. Durch die Gleichung wird eine Hyperebene definiert; somit sind jetzt nur noch solche Punkte zulässig, die auch in dieser liegen. D.h., die Menge der zulässigen Punkte ist die Schnittmenge des zuvor zulässigen Bereichs mit dieser Hyperebene. Im allgemeinen wird also das bereits berechnete Optimum des ursprünglichen Programms nun nicht mehr primal zulässig sein.

1.5 Ellipsoidmethode

Vorlesung vom 13.5.1996
Henning Biermann, Markus Löckelt, René Beier, Matthias Fischmann

1.5.1 Komplexität der Linearen Programmierung

Wir wollen die Komplexität der Linearen Programmierung untersuchen, insbesondere wollen wir wissen, ob LP in $\mathcal{P}$ liegt. Wie wir sehen werden, hängt die Antwort von der betrachteten Eingabegröße und von dem Rechenmodell ab.
Bei dem algebraischen Rechenmodell nimmt man an, daß alle Operationen in konstanter Zeit unabhängig von der Operandengröße ausgeführt werden können. Die Eingabegröße eines LP ist in diesem Modell die Anzahl der Zahlen in der Matrix.
In dem Modell der Bitkomplexität zählt man die Operationen auf einzelne Bits; dazu betrachtet man die ganzzahlige Eingabematrix in binärer Darstellung.

Definition 10 *Sei A eine $m \times d$ Matrix über $\mathbb{Z}$. Als Größe von A definieren wir die Länge ihrer binären Kodierung*

$$L(A) = \sum_{ij} (\log_2 |a_{ij}| + 2).$$

Man kann tatsächlich eine Kodierung dieser Länge angeben: Jeder Matrixeintrag a_{ij} läßt sich mit $\lceil \log_2 |a_{ij}| \rceil + 1$ Bits darstellen, indem man $|a_{ij}|$ als Binärzahl abspeichert und ein zusätzliches Bit für das Vorzeichen von a_{ij} verwendet. Die einzelnen Bitstrings trennt man mit einem Sonderzeichen als Separator.

Wir werden nun in mehreren Schritten zeigen, daß für Bitkomplexität Lineares Programmieren in $\mathcal{P}$ liegt. Für das algebraische Rechenmodell ist so eine Tatsache bis heute unbekannt.

Lineares Programmieren und $\mathcal{NP}$

Wir überlegen uns, ob LP in $\mathcal{NP}$ liegt. Dazu formulieren wir LP als Entscheidungsproblem.

Definition 11 *Gegeben seien $A \in \mathbb{Z}^{m\times n}, b \in \mathbb{Z}^m$. Das Problem*

$$\exists x : Ax \leq b$$

heißt LP-Entscheidungsproblem.

Um ein LP-Entscheidungsproblem zu lösen, könnte man eine Lösung x raten und anschließend durch Einsetzen verifizieren. Damit dieses Vorgehen die Aussage „LP-Entscheidungsproblem $\in \mathcal{NP}$“ beweist, muß die Verifikation in Polynomialzeit erfolgen. Für das Modell der Bitkomplexität heißt das, daß die x_i nicht zu groß sein dürfen ($L(x)$ polynomiell in $L(A)$). Aus der Ausgabenstellung folgt aber nicht offensichtlich, daß es kleine x gibt, die das Problem lösen. Wir werden dies erst später (mit Lemma 4) zeigen.
Das LP-Entscheidungsproblem liegt auch in co$\mathcal{NP}$, d. h. das Komplement des LP-Entscheidungsproblems liegt in $\mathcal{NP}$. Der Beweis sei dem Leser überlassen (Hinweis: Dualität). Dieses Ergebnis läßt bereits vermuten, daß das LP-Entscheidungsproblem in $\mathcal{P}$ liegt, da $\mathcal{P} \subset \mathcal{NP} \cap \text{co}\mathcal{NP}$.
Wir stellen nun die Grundidee für einen (neuen) Algorithmus für das LP-Entscheidungsproblem vor. Im nächsten Abschnitt werden wir diese Methode verfeinern.
Es sei ein LP-Entscheidungsproblem gegeben. Wir setzen den zulässigen Bereich $P = \{x \mid Ax \leq b\}$. Durch Einsetzen kann man leicht überprüfen, ob ein Punkt z in P liegt. Falls $z \in P$, dann haben wir das Problem gelöst. Andernfalls verletzt der Punkt eine Ungleichung. Wir wollen dann aus dem alten Punkt eine Lösung x' finden, die diese Ungleichung erfüllt, und hoffen, daß wir mit x' einen „Fortschritt“ erreichen. Es ist aber nicht klar, ob ein Punkt x immer zu einem besseren x' führt.

Beispiel Die Abbildung 1.5.1 zeigt die geometrische Lösung eines Entscheidungsproblems im $\mathbb{R}^2$. Wir starten

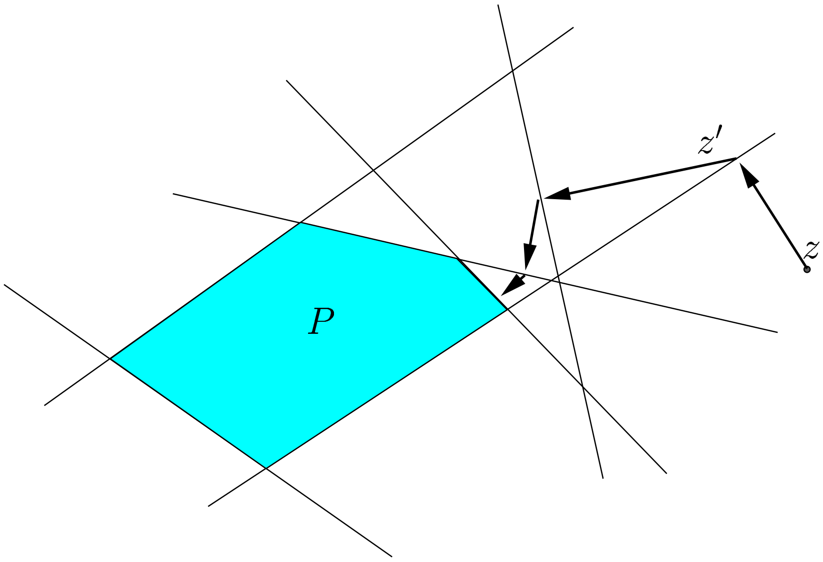


Abbildung 1.28: Geometrische Lösung eines Entscheidungsproblems im $\mathbb{R}^2$.

bei Punkt z und projizieren ihn in jedem Schritt senkrecht auf eine Halbebene, deren Ungleichung verletzt ist. Am Ende soll eine Projektion im Zulässigkeitsbereich P (grau) liegen.

Mit unserem Verfahren bleiben einige Punkte unklar:

1. Wie kann man „klaren Fortschritt“ erzielen?

2. Wie kann man feststellen, daß $P = \emptyset$ ist?

Zur zweiten Frage kann man sich überlegen, daß bei ganzzahligen Koeffizienten für A der Zulässigkeitsbereich ein Mindestvolumen abhänging von $L(A)$ hat. Damit kann man vielleicht die Zahl der Verbesserungsschritte abschätzen, die man braucht, bis x' in P liegt. Falls nach so vielen Schritten x' nicht in P liegt, ist P leer. Wir wollen im nächsten Abschnitt präziser werden.

1.5.2 Ellipsoid-Methode

Wir nehmen an, daß z Zentrum einer Ellipse E ($d > 2$: Ellipsoid) sei mit $S := E \cap P \neq \emptyset$ (falls $P \neq \emptyset$). Durch endlich viele Iterationsschritte wollen wir immer kleinere Ellipsen konstruieren, so daß das Zentrum schließlich in P liegt.

Beispiel Die Abbildung 1.5.2 zeigt einen Iterationsschritt der Ellipsoidmethode im $\mathbb{R}^2$. Falls z nicht in P liegt, ist eine Ungleichung verletzt. Durch paralleles Verschieben erhält man eine Hyperebene durch z. Diese Hyperebene teilt E in zwei Hälften; wir nennen die Hälfte H, die S enthält. Nun konstruieren wir eine möglichst kleine Ellipse E', die H vollständig umschließt. E' hat ein kleineres Volumen als E.

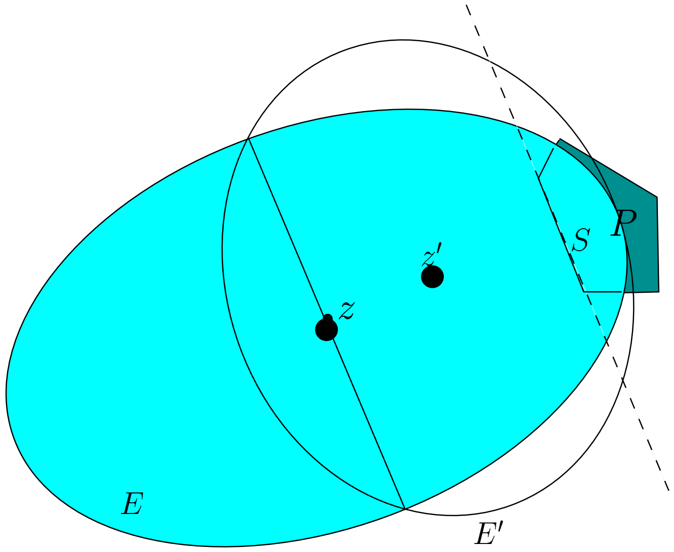


Abbildung 1.29: Konstruktion einer neuen Ellipse E' mit Zentrum z'

Iterationsschritt

1. Fall: $z \in P$. In diesem Fall gilt $Az \leq b$. Wir haben also eine Lösung gefunden und sind fertig.

2. Fall: $z \notin P$. Dann gibt es eine separierende Ungleichung $a_i z > b_i$, da für alle $x \in P$ gilt: $a_i x \leq b_i$.

   (i) Schneide E mit Halbraum H_i, der normal zu a_i liegt und z auf dem Rand liegen hat. $H_i = \{x \mid a_i(x - z) \leq 0\}$, wobei das Ungleichungszeichen vom Vorzeichen von b_i abhängt.

   (ii) Konstruiere ein Ellipsoid E' mit Zentrum z', das $E \cap H_i$ enthält und ein möglichst kleines Volumen hat.

Bei diesem Vorgehen stellen sich folgende Fragen, die wir mit einigen Lemmas beantworten wollen.

(0) Genügt es, das LP-Entscheidungsproblem zu lösen?

(1) Wie findet man ein Anfangsellipsoid?

(2) Wie führt man eine Iteration algebraisch durch?

(3) Muß die dargestellte Methode terminieren? Wie erkennt man Unerfüllbarkeit?

(3') Was tun, wenn der Zulässigkeitsbereich P das Volumen 0 hat (wenn beispielsweise P ein niedrigdimensionaler Polyeder ist - z. B. eine Strecke im $\mathbb{R}^2$)?

(4) Was ist die Laufzeit des Algorithmus? Wie lange braucht eine Iteration, und wie viele brauchen wir?

Lemma 2 *Wenn LP-Entscheidungsprobleme der Form $\exists x : Cx \leq d$ in polynomieller Zeit gelöst werden können, dann kann man auch LP-Optimierungsprobleme*

$$\max c^T x, \quad Ax \leq b$$

in polynomieller Zeit lösen.

Beweis Wir reduzieren das Optimierungsproblem auf ein LP-Entscheidungsproblem, dessen Eingabegröße nur polynomiell viel größer ist. Nach Voraussetzung können wir dann das Entscheidungsproblem (und damit die Optimierungsaufgabe) in Polynomzeit lösen. Gegeben sei also ein LP-Optimierungsproblem

$$(*) \qquad \max c^T x, \quad Ax \leq b.$$

Wir benutzen den Starken Dualitätssatz, wonach

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

falls keine der Mengen leer ist. Sei P der Zulässigkeitsbereich $\{x \mid Ax \leq b\}$ und

$$P' = \{(x,y) \in \mathbb{R}^{d+m} | Ax \leq b, \quad y \geq 0, \quad \begin{array}{rr} y^T A \leq c, & c^T x \leq y^T b \\ -y^T A \leq -c, & -c^T x \leq -y^T b\} \end{array}$$

Mit folgendem Algorithmus wollen wir $(*)$ lösen:

Algorithmus

1. Teste, ob $P = \emptyset$. Falls ja, hat $(*)$ keine Lösung, und wir sind fertig.

2. Sonst gilt $P \neq \emptyset$. Nun teste, ob $P' = \emptyset$. Falls ja, ist P unbeschränkt, das Maximum wird beliebig groß (Dualität). Andernfalls gibt es $(x_0, y_0) \in P'$ und x_0 ist eine Lösung des Optimierungsproblems, da nach Wahl von P' x_0 für das primale und y_0 für das duale Problem zulässig sind. Außerdem haben die Zielfunktionen den gleichen Wert: $c^T x_0 = y_0^T b$.

Das Entscheidungsproblem für P' ist polynomiell in der Eingabegröße $L(A)$, falls $(*)$ polynomiell ist, da sich die Bedingungen an P' in $O(L(A))$ Bits formulieren lassen. □

Damit ist die Frage (0) beantwortet. Wir zeigen jetzt mit zwei Hilfssätzen, wie man ein Anfangsellipsoid finden kann (1).

Lemma 3 *Sei A eine $n \times n$ Matrix über $\mathbb{Z}$ mit Bitkomplexität $L = L(A) = \sum_{ij}(2 + \log_2 |a_{ij}|)$. Wir setzen $L' = L'(A) = L(A) + n \log_2(n)$. Dann gilt*

$$|\underbrace{\det(A)}_{\in \mathbb{Z}}| \quad \leq \quad 2^{L'}.$$

Beweis Nach Definition gilt:

$$\det(A) = \sum_\pi \operatorname{sign}(\pi) \prod_i a_{i\pi(i)},$$

wobei die Summe über alle $n!$ Permutationen π läuft. $\det(A)$ liegt in $\mathbb{Z}$, da in der Summe nur ganze Zahlen vorkommen. Jeder Summand ist ein Produkt von n Elementen aus A (für jede Zeile wird ein Faktor ausgewählt). Wenn man alle Elemente aus A aufmultipliziert, erhält man eine Zahl $\leq 2^L$, da

$$\log_2 \prod_{ij} a_{ij} = \sum_{ij} \log_2 a_{ij} \leq L.$$

Daher gilt:

$$|\det(A)| \leq \sum_{\pi} \prod_{i} a_{i\pi(i)} \leq \sum_{\pi} \prod_{ij} a_{ij} \leq \sum_{\pi} 2^L = n! \cdot 2^L \leq n^n 2^L = 2^{n \log_2 n} \cdot 2^L = 2^{L'}$$

□

Lemma 4 *Seien $A \in \mathbb{Z}^{n\times n}$ invertierbar und $b \in \mathbb{Z}^n$. Wir setzen $L = L(A, b)$, wobei (A, b) die $n \times (n + 1)$ Matrix A mit zusätzlicher Spalte b ist. Seien ferner $L' = L + n \log_2 n$, und $x = (x_1, \ldots, x_n)^T$ eine Lösung von $Ax = b$. Dann gibt es ganze Zahlen $D_1, \ldots, D_n$ und D mit*

$$x_j = \frac{D_j}{D}, \quad |D_j| \leq 2^{L'} \quad \textit{und} \quad |D| \leq 2^{L'}.$$

Beweis Wir können die x_j mit der Cramerschen Regel ausrechnen.

$$x_j = \frac{\det(B_j)}{\det(A)}, \quad \text{mit} \quad B_j = (A_1, \ldots, A_{j-1}, b, A_{j+1}, \ldots, A_n).$$

A_i sei dabei die $i$te Spalte von A. Mit Hilfe des Lemmas 3 folgt die Behauptung. □

Beobachtung Dieses Lemma kann man nicht so leicht mit Gauss-Elimination zeigen, da sich in jedem Eliminationsschritt die Bitlänge verdoppeln kann. Man kann so nicht garantieren, daß die Größe des Lösungsvektors polynomiell in der Eingabegröße $L(A, b)$ ist.

Jetzt beschäftigen wir uns damit, wie wir ein Anfangsellipsoid finden. Sei das LP-Entscheidungsproblem $Ax \leq b$ gegeben mit $A \in Z^{m\times d}$. Wir wissen, daß jede Ecke x des Zulässigkeitsbereich ein quadratisches Teilsystem, also d Zeilen der Ungleichung, mit Gleichheit erfüllt. Nach Lemma 4 gilt daher für x: $|x_j| \leq 2^{L'}$, $\quad 1 \leq i \leq d$.
Wir wählen als Anfangsellipsoid E die Kugel um den Ursprung $z = 0$ mit Radius $2^{L'} \cdot d$. Falls der Zulässigkeitsbereich nicht leer ist, gibt es eine Ecke, die in unserem Anfangsellipsoid liegt. Wir haben also ein E gefunden, daß die Anfangsbedingung für unseren Algorithmus erfüllt.

Vorlesung vom 20.5.1996
Oliver Scherf, Valeria Samid, Patrick Stiffel, Igor Pouchkarev

Unter Punkt (2) stellte sich die Frage, wie die algebraische Form einer Iteration aussieht. Im folgenden soll diese Frage näher erläutert werden.
Was ist im algebraischen Sinne ein Ellipsoid ? Um eine entsprechende Definition geben zu können, müssen wir zunächst zwei andere Definitionen machen.

Definition 12 *Eine* affine Transformation *T ist eine Abbildung von $\mathbb{R}^n$ nach $\mathbb{R}^n$ mit $T(x) = Qx + t$, wobei Q eine invertierbare Matrix sein muß.*

Intuitiv gesprochen „verbiegt Q den Raum“ und t ist ein Translationsvektor der ihn „verschiebt“.

Definition 13 *Eine* Kugel *K mit Zentrum c und Radius r ist die Menge*

$$K(c, r) = \{x \mid (x - c)^T (x - c) \leq r^2\}.$$

Definition 14 *Ein* Ellipsoid *E mit Zentrum z ist eine affine Transformation der Einheitskugel $K(0, 1)$, d.h.*

$$E = \{y \mid (y - z)^T B^{-1} (y - z) \leq 1\}.$$

Dabei muß die Matrix B folgende äquivalente Eigenschaften erfüllen:

- *B^{-1} ist eine symmetrische[1], positiv definite Matrix.*
- *B^{-1} ist eine symmetrische Matrix, deren Eigenwerten alle positiv sind.*

[1] Zur Erinnerung: Symmetrische Matrizen besitzen nur *reelle* Eigenwerte.

- $B^{-1} = Q^{-1T}Q^{-1}$.

Wie leitet sich die Formel für ein Ellipsoid aus den obigen Defintionen ab ?
Die Einheitskugel hat folgende Form:

$$K(0,1) = \{x \mid x^T x \leq 1\}.$$

Unterwirft man die Einheitskugel einer affinen Transformation T

$$T(x) = Qx + t \qquad \Leftrightarrow \qquad x = Q^{-1}(T(x) - t),$$

so ergibt sich

$$T(K(0,1)) = \{ \underbrace{y}_{=Qx+t} \mid x^T x \leq 1\} = \{y \mid (y-t)^T \underbrace{Q^{-1T}Q^{-1}}_{=B^{-1}}(y-t) \leq 1\}$$

mit $b = QQ^T$.
Die Einträge auf der Diagonalen in B bewirken eine Streckung/Stauchung der Achsen der Einheitskugel, die restlichen Einträge die Rotation im Raum.

Wie wir vorher überlegt haben, soll bei der Ellipsoidmethode das in einem Iterationsschritt neu berechnete Ellipsoid möglichst klein sein. Deshalb wollen wir zunächst etwas über Volumen von Körpern im Raum aussagen.

- Sei $T(x) = Qx + t$ eine affine Transformation und M ein Körper mit $vol(M) = \mu$. Dann ergibt sich das Volumen des transformierten Körpers folgendermaßen:

  $$vol(T(M)) = \mu |\det Q|.$$

  Dabei gibt die Determinante von Q an, um wieviel sich das Volumen des Körpers M verkleinert oder vergrößert.

- Affine Transformationen erhalten Verhältnisse von Volumen. Zum Beispiel seien $vol(M)$ und $vol(N)$ Volumen der Körper M und N und T eine affine Transformation. Dann gilt:

  $$\frac{vol(M)}{vol(N)} = \frac{vol(T(M))}{vol(T(N))}.$$

Wie findet man nun ausgehend von einem Ellipsoid E mit Zentrum z das neue Ellipsoid E' mit Zentrum z'? Um das ganze zu vereinfachen, wollen wir zunächst das Problem für den Spezialfall, daß das Ausgangsellipsoid E die Einheitskugel und der Richtungsvektor a der separierenden Ungleichung der Einheitsvektor $e_1 = (1, 0, \ldots, 0)^T$ ist, lösen und dann mit Hilfe der affinen Transformation die Lösung verallgemeinern.
Sei also $E = K(0,1)$ und $a = e_1 = (1, 0, \ldots, 0)^T$. Welche Bedingungen muß das neue Ellipsoid E' erfüllen ?

- Das Zentrum z' von E' muß in der Halbebene $\{x \mid x_1 \leq 0\}$ liegen. In der Abbildung 1.30 ist dies die eingefärbte Fläche.

- E' muß das Halbellipsoid HE mit $HE = \{x \in K(0,1) \mid x_1 \leq 0\}$ einschließen. In der Abbildung 1.30 ist HE durch die schraffierte Fläche gekennzeichnet.

- E' soll möglichst klein sein, d.h. wir suchen ein möglichst kleines Ellipsoid, das die Punkte $-e_1, e_2, e_3, \ldots e_n$ als Randpunkte beinhaltet. So ein Ellipsoid hat notwendigerweise sein Zentrum auf der Hyperebene $x_2 = x_3 = \ldots = x_n = 0$.

Wähle also $z' = te_1$ mit $t < 0$. Das t muß noch bestimmt werden. Wenn man sich dies anhand der Abbildung 1.30 veranschaulicht, so sieht man, daß die Einheitskugel in der ersten Komponente x_1 gestaucht, in $x_2, x_3, \ldots, x_n$ gedehnt wird. Das Ellipsoid E' muß also folgende Form haben:

$$E' = \{x \mid (x - z')^T \begin{pmatrix} \frac{1}{a^2} & 0 & \cdots & 0 \\ 0 & \frac{1}{b^2} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \frac{1}{b^2} \end{pmatrix} (x - z') \leq 1\},$$

mit $a < 1$ negative Streckung in Richtung x_1 und $b > 1$ positive Streckung in $x_2, x_3, \ldots, x_n$ Richtung.

Wie sehen die Matrixeinträge $\frac{1}{a^2}$ und $\frac{1}{b^2}$ aus? Wir wissen, daß die Ungleichung für $-e_1$, $e_2, e_3, \ldots, e_n$ mit Gleichheit erfüllt sein muß. Somit erhalten wir ($z' = te_1$ eingesetzt):

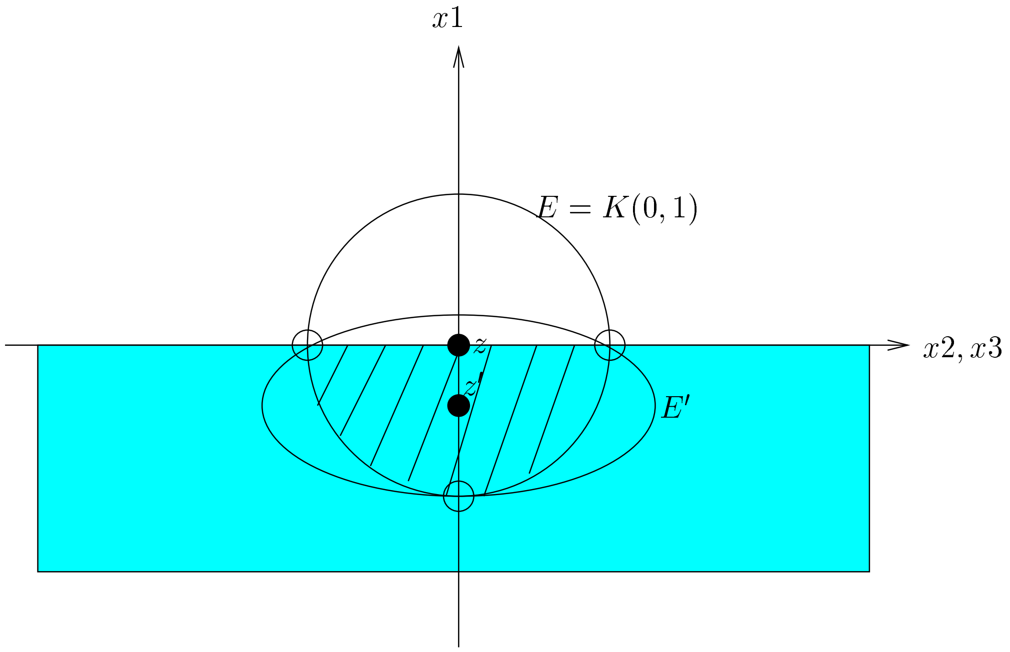


Abbildung 1.30: Konstruktion der Ellipse E', wenn E Einheitskugel $K(0,1)$

- Für e_1:

$$\underbrace{(-e_1 - te_1)^T}_{-\begin{pmatrix}1+t\\0\\\vdots\\0\end{pmatrix}} \begin{pmatrix} \frac{1}{a^2} & 0 & \cdots & 0 \\ 0 & \frac{1}{b^2} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \frac{1}{b^2} \end{pmatrix} (-e_1 - te_1) = 1.$$

Daraus folgt:

$$\frac{(1+t)^2}{a^2} = 1.$$

- Für e_2:

$$\underbrace{(e_2 - te_1)^T}_{\begin{pmatrix}-t\\1\\0\\\vdots\\0\end{pmatrix}} \begin{pmatrix} \frac{1}{a^2} & 0 & \cdots & 0 \\ 0 & \frac{1}{b^2} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \frac{1}{b^2} \end{pmatrix} (e_2 - te_1) = 1.$$

Daraus folgt:

$$\frac{t^2}{a^2} + \frac{1}{b^2} = 1.$$

Wir haben also für a und b Terme bekommen, die nur noch von dem noch zu berechnenden t abhängen:

$$\frac{1}{a^2} = \frac{1}{(1+t)^2} \qquad \text{und} \qquad \frac{1}{b^2} = \frac{1+2t}{(1+t)^2}.$$

Übung:
Zeige : Alle $x = (x_1, x_2, \ldots, x_n)$ mit $x^T x \leq 1$ und $x_1 \leq 0$ erfüllen auch $(x-\hat{z})\cdot\Lambda\cdot(x-\hat{z}) \leq 1$, wobei für $-1 < t < 0$

$$\hat{z} = \begin{pmatrix} t \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \qquad \Lambda = \begin{pmatrix} \frac{1}{a^2} & 0 & \cdots & 0 \\ 0 & \frac{1}{b^2} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \frac{1}{b^2} \end{pmatrix}$$

und

$$\frac{1}{a^2} = \frac{1}{(1+t)^2} \quad \text{und} \quad \frac{1}{b^2} = \frac{1+2t}{(1+t)^2}.$$

Wie muß man t wählen, damit das Volumen von E' minimal wird?

Es gilt:

$$vol(E') = vol(E) \cdot \underbrace{a \cdot b^{n-1}}_{\mu(t)}.$$

Wir möchten also den Streckungsfaktor $\mu(t)$ minimieren:

$$\begin{aligned} \mu^2(t) &= a^2 \cdot b^{2(n-1)} \\ &= (1+t)^2 \cdot \left(\frac{(1+t)^2}{(1+2t)}\right)^{n-1} \\ &= \frac{(1+t)^{2n}}{(1+2t)^{n-1}}. \end{aligned}$$

Ableiten und 0-setzen liefert:

$$t = -\frac{1}{n+1}.$$

Damit kann man jetzt auch explizit Werte für a und b angeben:

$$a = \frac{n}{n+1}, \qquad b = \frac{n}{\sqrt{n^2-1}}.$$

Wir wissen, daß sich das Volumen von E' umd den Faktor $\gamma_n := opt\mu(t)$ von dem Volumen von E unterscheidet. Ist dieser Faktor klein genug, um die gewünschte polynomielle Laufzeit zu gewährleisten?
Wir wissen:

$$\gamma_n^2 = \left(\frac{n}{n+1}\right)^2 \cdot \left(\frac{n^2}{n^2-1}\right)^{n-1} = \left(1 - \frac{1}{n+1}\right)^2 \cdot \underbrace{\left(\frac{\left(1-\frac{1}{n+1}\right)^2}{1-\frac{2}{n+1}}\right)^{n-1}}_{(*)}.$$

Um diese etwas unhandlichen Terme abschätzen zu können, benötigt man zwei Werkzeuge:

1) Für alle $x \in \mathbb{R}$ gilt $1 + x \leq e^x$.

2) Für alle $x \in \mathbb{R}$ und $a > 0$ gilt $(1+x)^a \leq e^{ax}$.

Betrachte den Term (*). Es gilt ($n+1$ wird durch v substituiert):

$$\begin{aligned} \frac{\left(1-\frac{1}{v}\right)^2}{1-\frac{2}{v}} &= 1 + \frac{\frac{1}{v^2}}{1-\frac{2}{v}} \\ &= 1 + \frac{1}{v^2 - 2v} \\ &= 1 + \frac{1}{v(v-2)} \\ &= 1 + \frac{1}{(n-1)(n+1)}. \end{aligned}$$

Also ergibt sich für γ_n^2:

$$\begin{aligned}\gamma_n^2 &= \left(1-\frac{1}{n+1}\right)^2 \cdot \left(1+\frac{1}{(n-1)(n+1)}\right)^{n-1}\\ &\leq e^{-\frac{2}{n+1}} \cdot e^{\frac{1}{n+1}}\\ &= e^{-\frac{1}{n+1}},\end{aligned}$$

und damit für unseren gesuchten Term γ_n:

$$\gamma_n \leq e^{-\frac{1}{2(n+1)}}.$$

Damit haben wir den Spezialfall, daß E die Einheitskugel und $a = e_1$ ist, vollständig abgehandelt. Wie gehen wir vor, wenn dieser Spezialfall nicht gegeben ist? Wir haben also einen beliebigen Ellipsoid $E = \{x|(x-z)^T \cdot B^{-1} \cdot (x-z) \leq 1\}$ und einen beliebigen Richtungsvektor a gegeben. Sei dazu $H = \{x|a^T(x-z) \leq 0\}$ der von a induzierte Halbraum. Wie findet man ein Ellipsoid E' mit Zentrum z' und Matrix B', das $E \cap H$ enthält ?
Die Idee ist (siehe Schemazeichung 1.31):

(i) Transformiere E durch eine Transformation T^{-1} zur Einheitskugel $K(0,1)$.

(ii) Rotiere den Halbraum H durch eine Rotation R^{-1} so, daß das Bild von H mit $\{x\,|\,x_1 \leq 0\}$ zusammenfällt. Jetzt hat man die Situation des Spezialfalles gegeben.

(iii) Bilde $\hat{E}$ mit Zentrum $\hat{z}$ wie im Spezialfall angegeben.

(iv) Mache die Transformationen (ii) und (i) rückgängig und bilde so E' mit Zentrum z'.

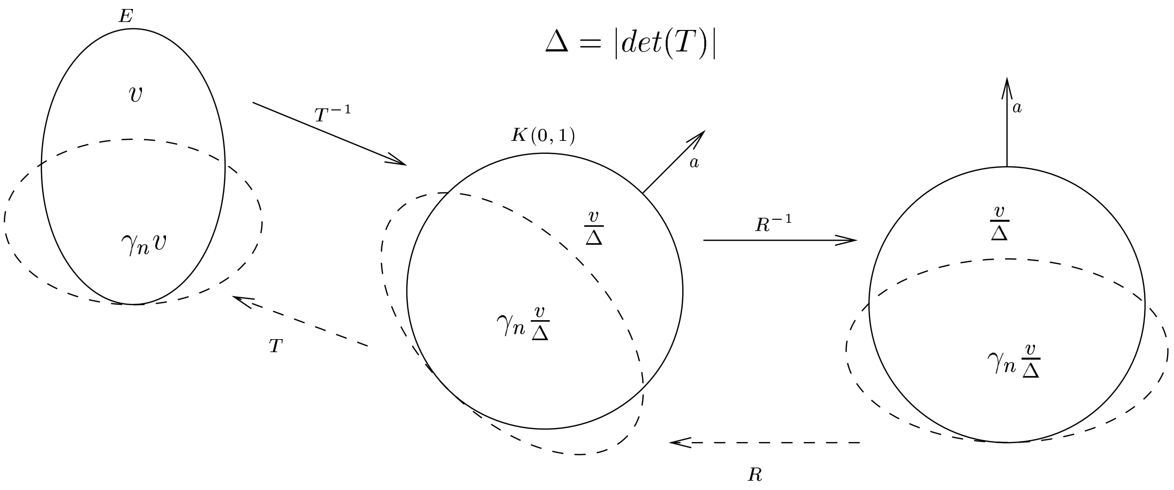


Abbildung 1.31: Transformation von E zu $K(0,1)$, Rotation, Bildung von $\hat{E}$ um $\hat{z}$ und zurück

Das Ellipsoid E ist nach Definition eine affine Transformation T der Einheitskugel: $E = T(K(0,1))$, mit $T(x) = Qx + z$, Q invertierbar. Die zu T inverse affine Transformation bildet also E auf die Einheitskugel ab: $T^{-1}(E) = K(0,1)$.
Wendet man die Transformation T^{-1} auf den Halbraum H an, so ergibt sich:

$$T^{-1}(H) = \{x \mid (Q^T a)^T x \leq 0\},$$

denn:

$$T^{-1}(H) \quad = \quad T^{-1}(\{x \mid a^T(x-z) \leq 0\}$$

$$\begin{aligned} &= \{T^{-1}(x) \mid a^T(x-z) \leq 0\} \\ &= \{x \mid a^T(T(x)-z) \leq 0\} \\ &= \{x \mid a^T(Qx+z-z) \leq 0\} \\ &= \{x \mid (Q^Ta)^T \leq 0\}. \end{aligned}$$

Diesen transformierten Halbraum wollen wir jetzt so rotieren, daß er auf den Halbraum $\{x \mid x_1 \leq 0\}$ abgebildet wird. Für diese Rotation R^{-1} muß dann gelten:

$$R^{-1}(\frac{Q^Ta}{\|Q^Ta\|}) = e_1,$$

und damit auch

$$\frac{Q^Ta}{\|Q^Ta\|} = Re_1.$$

Wir befinden uns jetzt in dem eben behandelten Spezialfall und können daher für das transformierte und rotierte Problem das Zentrum des neuen Ellipsoids angeben:

$$\hat{z} = te_1 = -\frac{1}{n+1}e_1.$$

Macht man nun zunächst die Rotation R^{-1} rückgängig, so erhält man:

$$R\hat{z} = -\frac{1}{n+1}Re_1 = -\frac{1}{n+1} \cdot \frac{Q^Ta}{\|Q^Ta\|}.$$

Wendet man darauf noch die inverse Transformation an, so ergibt sich das Zentrum z' des neu zu berechnenden Ellisoids E':

$$z' = T(R\hat{z}) = -\frac{1}{n+1} \cdot \frac{QQ^Ta}{\|Q^Ta\|} + z.$$

Auf die gleiche Weise kann man die das Ellipsoid E' bestimmende Matrix B' ausrechnen:

$$B' = \frac{n^2}{n^2-1}(B - \frac{2}{n+1} \cdot \frac{Ba(Ba)^T}{a^TBa}).$$

Vorlesung vom 23.5.1996
Jochen Wagner, Markus Walz, Boris Brandherm

1.5.3 Der Ellipsoid-Algorithmus

Wir haben nun hergeleitet, wie wir den Ellipsoid–Algorihtmus algebraisch beschreiben können, und wollen diesen nun zusammenfassend aufschreiben. Der Algorithmus löst das folgende Problem: "Gibt es eine Lösung x, die den Ungleichungen $Ax \leq b$ genügt ?"
Sei dazu K die Anzahl der Iterationen, die der Algorithmus höchstens durchlaufen muß, um ein x zu finden oder um festzustellen, daß es keine Lösung gibt.
Sei B das Anfangsellipsoid. Wir werden B als Kugel wählen, deren Radius so groß ist, daß mindestens ein Teil von $P = \{x : Ax \leq b\}$ in B enthalten ist (vorausgesetzt, P ist nicht leer).
Die Anzahl K der Iterationen und den Radius B werden wir auf den folgenden Seiten bestimmen.

Ellipsoid(A, b)
(1) $K := ?$ (* Anzahl der benötigten Iterationen *)
(2) $B := ?$ (* Anfangsellipsoid *)
(3) $z := (0, \ldots, 0)$ (* Ursprung = Zentrum des Anfangsellipsoids*)
(4) **for** $l := 0$ **to** $K-1$ **do**
(5) **if** $Az \leq b$ **then Return** JA(z) (* Lösung gefunden *)
(6) **else** finde ein i, sodaß $a_iz > b_i$ (* Finde Ungleichung, die verletzt wird *)
(7) $a := {a_i}^T$ (* Finde neues Ellipsoid *)
(8) $z := z - \frac{1}{n+1}\frac{Ba}{\sqrt{a^TBa}}$
(9) $B := \frac{n^2}{n^2-1}\left(B - \frac{2}{n+1}\frac{(Ba)(Ba)^T}{a^TBa}\right)$
(10) **Return** NEIN (* Es gibt keine Lösung *)

Bestimmung des Anfangsellipsoiden

Gesucht ist ein Radius $r \in \mathbb{R}^n$, so daß das Anfangsellipsoid B mindestens einen Teil von $P = \{x : Ax \le b\}$ enthält, falls P nicht leer ist. Wir zeigen zunächst, daß mindestens ein Punkt von P im n-dimensionalen Würfel W_n mit Zentrum $\vec{0}$ und Seitenlänge $2 \cdot 2^{L'} = 2^{L'+1}$ liegt.

1. **Fall**: P hat Ecken (siehe Abb. 1.32 und 1.33)
   Betrachte eine beliebige Ecke v von P. Dann gibt es n linear unabhängige Ungleichungen in $Ax \le b$, die mit Gleichheit erfüllt sind und als Lösung $v = (x_1, \ldots, x_n)$ haben. O.B.d.A. nehmen wir an, daß die ersten n Ungleichungen linear unabhängig sind.

$$\begin{array}{rcl} a_{11}x_1 + \cdots + a_{1n}x_n & = & b_1 \\ & \vdots & \\ a_{n1}x_1 + \cdots + a_{nn}x_n & = & b_n \end{array}$$

   Für dieses System gilt die gleiche Abschätzung der Größenordnung der Lösung wie für $Ax \le b$ (vgl. Lemma 4). Also liegt jede Ecke v von P in W_n.

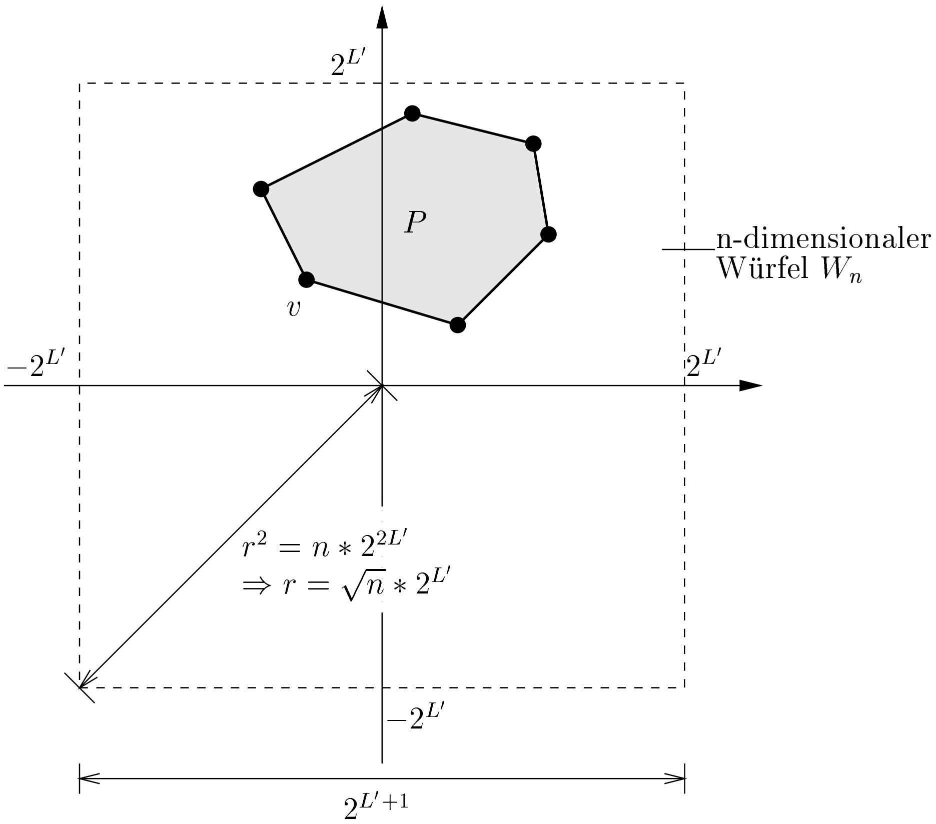


Abbildung 1.32: P liegt komplett in W_n

2. **Fall**: P hat keine Ecken (siehe Abb. 1.34)

   Behauptung 2 *W_n wird auf jeden Fall von P geschnitten.*

   Da P keine Ecken hat, können nur $k < n$ linear unabhängige Gleichungen mit Gleichheit erfüllt sein (O.B.d.A. die ersten k Ungleichungen).

$$\begin{array}{rcl} a_{11}x_1 + \cdots + a_{1n}x_n & = & b_1 \\ & \vdots & \end{array}$$

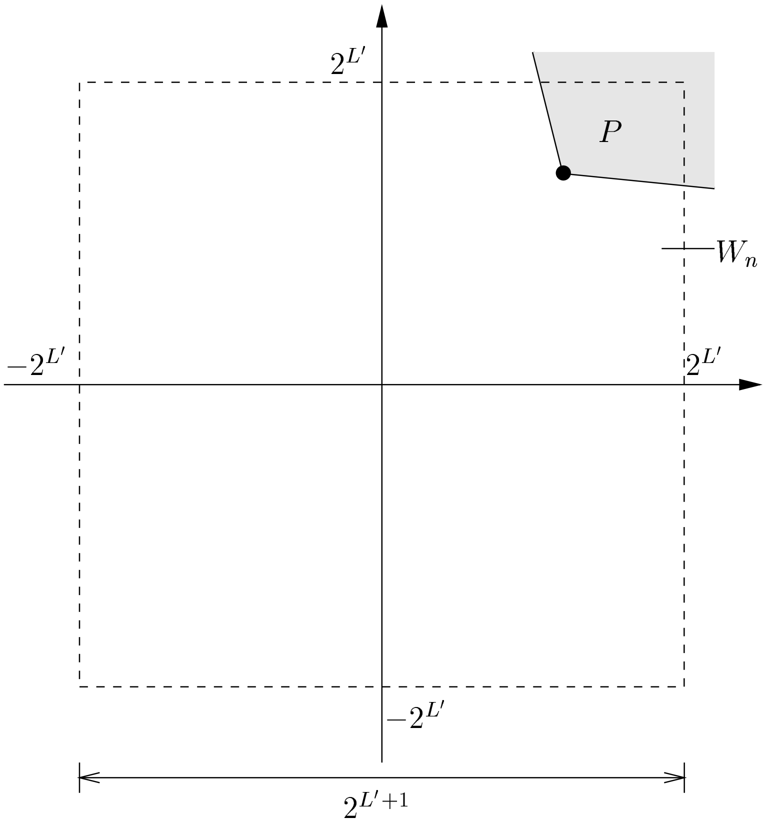


Abbildung 1.33: P hat nur eine Ecke

$$a_{k1}x_1 + \cdots + a_{kn}x_k \quad = \quad b_k.$$

Da die k Gleichungen dieses Systems linear unabhängig sind, gibt es auch k linear unabhängige Spalten (O.B.d.A. die ersten k Spalten). Durch Streichen der Spalten $k+1$ bis n entsteht folgendes $k \times k$–System:

$$\begin{aligned} a_{11}x_1 + \cdots + a_{1k}x_k &= b_1 \\ &\vdots \\ a_{k1}x_1 + \cdots + a_{kk}x_k &= b_k. \end{aligned}$$

Dieses System hat den Rang k, ist also regulär und hat eine eindeutige Lösung $(x_1, \ldots, x_k)$. $(x_1, \ldots, x_k, 0, \ldots, 0)$ ist dann eine Lösung des ursprünglichen Systems und es gilt: $|x_i| < 2^{L'}, 1 \leq i \leq n$. Also liegt auch in diesem Fall ein Punkt von P in W_n. Durch das Nullsetzen von x_{k+1} bis x_n wird ein Schnittpunkt von P mit der Hyperebene der ersten k Dimensionen bestimmt. Im zweidimensionalen Fall bedeutet dies, daß man z.B. $y = 0$ setzt, um den Schnittpunkt mit der x-Achse zu bestimmen. Betrachten wir die Koordinaten $x_1, \ldots, x_n$ eines solchen Punktes, so können wir wieder Lemma 4 anwenden und erhalten: $|x_i| < 2^{L'}, 1 \leq i \leq n$.
$\Rightarrow$ Ein solcher Punkt von P liegt im Würfel W_n mit Seitenlänge $2 \cdot 2^{L'} = 2^{L'+1}$ und Zentrum $\vec{0}$.

Also gilt in beiden Fällen: Falls P nicht leer ist, liegt ein Teil von P in W_n. Daher wählt man als Anfangsellipsoid E_0 eine Kugel um $\vec{0}$, bei der die Eckpunkte von W_n auf der Kugeloberfläche liegen. Diese Kugel enthält dann W_n komplett und somit einen Teil von P (falls P nicht leer ist). Der Kugelradius ergibt sich dabei als Abstand vom Ursprung zu einer beliebigen Ecke von W_n, z.B. $(2^{L'}, \ldots, 2^{L'})$:

$$r^2 = n2^{2L'} \Longleftrightarrow r = \sqrt{n} \cdot 2^{L'}.$$

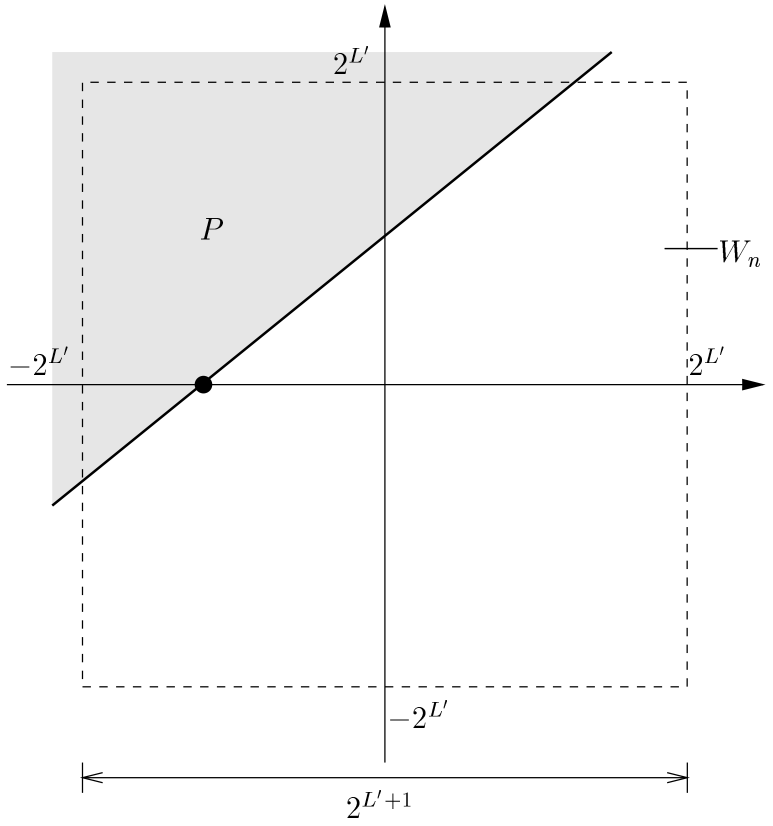


Abbildung 1.34: P hat keine Ecken

Die Matrix Q der Transformation von K(0,1) zu E_0 skaliert demnach die Einheitskugel mit diesem r:

$$\begin{aligned} Q &= rE_n = \sqrt{n} \cdot 2^{L'} E_n \\ B &= QQ^T = rE_n(rE_n)^T = r^2 E_n \end{aligned}$$

$$\boxed{\Rightarrow B = n2^{2L'} E_n}$$

Bestimmung von K

Sei P nicht leer. Falls das Volumen von P Null ist (2-dimensional: P ist ein Punkt oder eine Gerade), benutzen wir Lemma 6 und lösen anstatt $\exists x : Ax \leq b$ das äquivalente Problem $\exists x : Ax < b'$ mit $b' = b + \lambda(1, \ldots, 1)^T$; d.h., wir beschränken uns zunächst auf Systeme strikter Ungleichungen. Bei jeder Iteration wird das Volumen des Ellipsoids um den Faktor γ kleiner. Fällt das Volumen des Ellipsoids unter ein bestimmtes Mindestvolumen von P, so kann der Algorithmus abbrechen. Also brauchen wir zur Bestimmung von K eine untere Schranke für das Volumen von $P \cap E_0$.
Seien dazu$\mu > 0$ eine untere Schranke für vol$(P \cap E_0)$, M eine obere Schranke für vol(E_0) und E_l das Ellipsoid nach der l-ten Iteration.
Dann gilt: $\mu \leq \text{vol}(E_l) \leq \gamma^l \text{ vol}(E_0) \leq \gamma^l M$. Falls eine solche untere Schranke μ existiert, kann l bestimmt werden. Wie groß muß l sein, so daß $\gamma^l M < \mu$ ist?

$$\begin{array}{lrcll} & \gamma^l M & < & \mu & \\ \Longleftrightarrow & \gamma^l & < & \frac{\mu}{M} & \\ \Longleftrightarrow & l \ln \gamma & < & \ln \frac{\mu}{M} & \quad (\ln \gamma < 0, \text{ da} \gamma < 1) \\ \Longleftrightarrow & l & > & \ln \frac{\mu}{M} / \ln \gamma & \end{array}$$

Es gilt: $\ln \gamma < -\frac{1}{2(n+1)}$, da $\gamma < e^{-\frac{1}{2(n+1)}}$ (vgl. Abschätzung Seite 42, also ist $l > -\ln \frac{\mu}{M} \cdot 2(n+1)$.

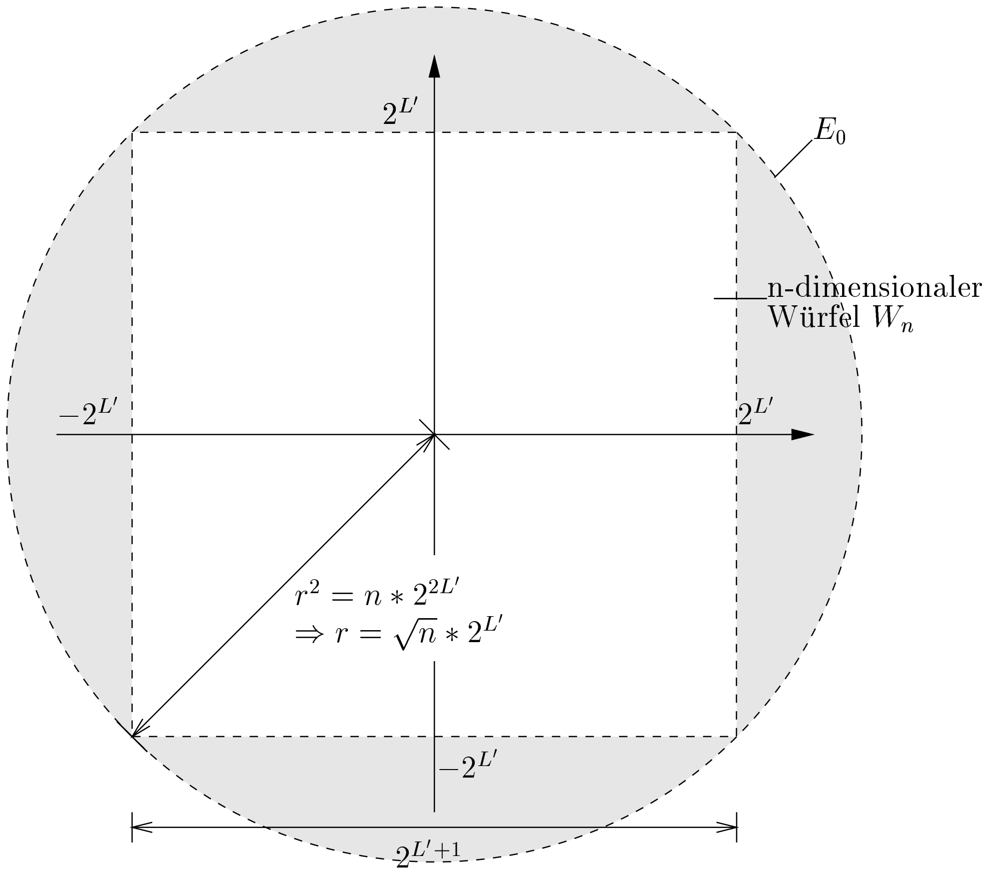


Abbildung 1.35: Kugel um W_n

Die kleinste natürliche Zahl l mit obiger Eigenschaft, ist dann eine obere Schranke für die Anzahl der Iterationen. Dieses l bezeichnen wir mit l_{min}. Hat P ein positives Volumen, so wird in höchstens l_{min} Iterationen ein passendes z gefunden. Wir müssen nun also noch die Größen M und μ bestimmen.

1. Bestimmung von M
   Als obere Schranke M für das Volumen der Kugel E_0 wählen wir das Volumen des umbeschriebenen Würfels (siehe dazu Abb. 1.36). $\text{vol}(E_0) < M := (2r)^n = (\sqrt{n} \cdot 2^{L'+1})^n = n^{\frac{n}{2}} \cdot 2^{n(L'+1)}$

2. Bestimmung von μ
   Wir betrachten den Schnitt $\hat{P}$ von P mit dem Würfel W_n. Es gibt mindestens $n+1$ Ecken $v_0, \ldots, v_n$, die nicht alle in einer Hyperebene liegen und einen Simplex S aufspannen (siehe Abb. 1.37). Würden alle Ecken auf einer Hyperebene liegen, so wäre $\text{vol}(P) = 0$. Dieses haben wir aber ausgeschlossen.

   Da S ganz in $\hat{P}$ liegt, ist sein Volumen eine untere Schranke für das Volumen von $\hat{P}$: vol(S)$\leq$vol($\hat{P}$)
   In der Ebene entspricht dem Simplex S ein Dreieck. Wir bestimmen das Volumen (in Dimension 2: Fläche) des Dreiecks:

$$\text{Fläche}(S_2) = \frac{1}{2!} \begin{vmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{vmatrix}$$

   Im $\mathbb{R}^n$ ist das Volumen des Simplex gegeben durch:

$$\text{vol}(S) = \left| \frac{1}{n!} \begin{vmatrix} 1 & v_0 \\ 1 & v_1 \\ \vdots & \vdots \\ 1 & v_n \end{vmatrix} \right|$$

   Die n-dimensionalen Zeilenvektoren $v_0, \ldots, v_n$ geben die $n+1$ Ecken von $\hat{P}$ an, die S aufspannen.

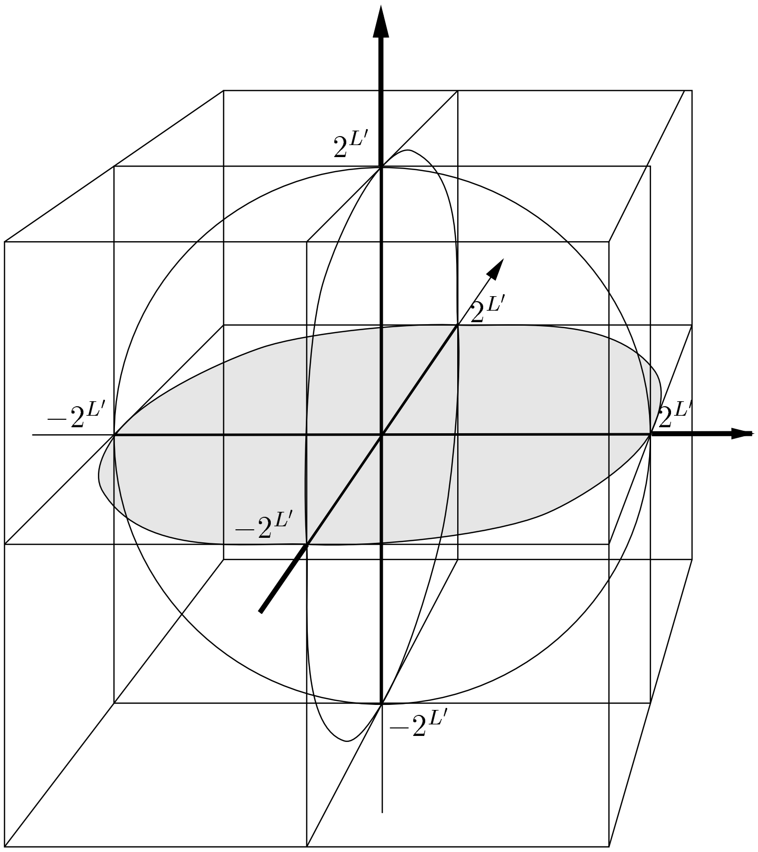


Abbildung 1.36: Obere Abschätzung für M

Jede Ecke ist eine Lösung eines Teilsystems von $Ax = b$ und es gilt daher:

$$v_{ij} = \frac{{D_j}^i}{D_i} \qquad ,1 \leq i, j \leq n, \qquad |{D_j}^i| < 2^{L'}, |D_i| < 2^{L'}.$$

$$\begin{aligned}
\Longrightarrow \mathrm{vol}(S) &= \left| \frac{1}{n!} \left\| \begin{matrix} 1 & \frac{{D_1}^0}{D_0} & \frac{{D_2}^0}{D_0} & \cdots & \frac{{D_n}^0}{D_0} \\ \vdots & \vdots & & & \vdots \\ 1 & \frac{{D_1}^n}{D_n} & \frac{{D_2}^n}{D_n} & \cdots & \frac{{D_n}^n}{D_n} \end{matrix} \right\| \right| \qquad D_0, \ldots, D_n \neq 0 \\
&= \left| \frac{1}{n!} \cdot \frac{1}{D_0 \cdot D_1 \cdots D_n} \right| \cdot \left\| \begin{matrix} D_0 & {D_1}^0 & {D_2}^0 & \cdots & {D_n}^0 \\ \vdots & & & & \vdots \\ D_n & {D_1}^n & {D_2}^n & \cdots & {D_n}^n \end{matrix} \right\|
\end{aligned}$$

Da die $n + 1$ Ecken $v_0, \ldots, v_n$ nicht in einer Hyperebene liegen, sind die Zeilenvektoren von

$$\begin{pmatrix} 1 & v_0 \\ \vdots & \\ 1 & v_n \end{pmatrix}$$

linear unabhängig, folglich gilt:

$$\begin{vmatrix} 1 & v_0 \\ \vdots & \vdots \\ 1 & v_n \end{vmatrix} \neq 0 \Longrightarrow \begin{vmatrix} D_0 & {D_1}^0 & {D_2}^0 & \cdots & {D_n}^0 \\ \vdots & & & & \vdots \\ D_n & {D_1}^n & {D_2}^n & \cdots & {D_n}^n \end{vmatrix} \neq 0$$

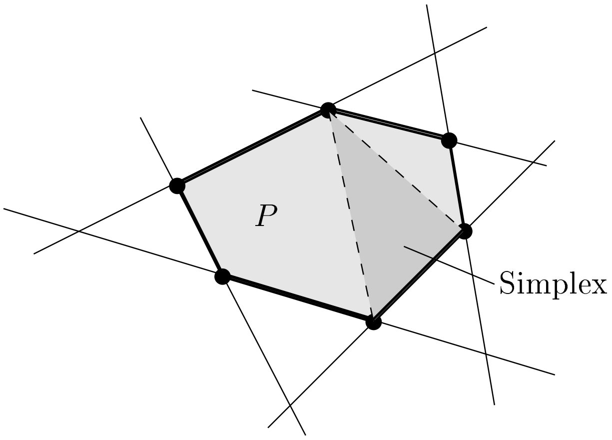


Abbildung 1.37: Simplex S

Da die $D_0, \ldots, D_n$ ganzzahlig sind und bei der Berechnung der Determinanten nur Additionen und Multiplikationen auftreten, muß gelten:

$$\left\| \begin{array}{ccccc} D_0 & {D_1}^0 & {D_2}^0 & \cdots & {D_n}^0 \\ \vdots & & & & \\ D_n & {D_1}^n & {D_2}^n & \cdots & {D_n}^n \end{array} \right\| \geq 1$$

Außerdem gilt: $|D_i| < 2^{L'} \iff \frac{1}{|D_i|} > 2^{-L'}$, und damit: vol(S)$> \frac{1}{n!} \cdot 2^{-L'(n+1)} > \frac{1}{n^n} \cdot 2^{-L'(n+1)}$

Als untere Schranke des Volumens von S (und damit von $\hat{P}$) wählen wir:

$$\boxed{\mu = \frac{1}{n^n} \cdot 2^{-L'(n+1)}}$$

Mit M und μ setzen wir die Berechnung von l fort:

$$\begin{aligned} l &> -2(n+1)\ln\frac{\mu}{M} \\ &= -2(n+1)\ln\frac{2^{-L'(n+1)} \cdot 2^{-n(L'+1)}}{n^{\frac{n}{2}}n^n} \\ &= -2(n+1)\ln(2^{-L'(2n+1)-n} \cdot n^{-\frac{3}{2}n}) \\ &= -2(n+1)(\ln 2^{-L'(2n+1)-n} + \ln n^{-\frac{3}{2}n}) \\ &= -2(n+1)(\frac{-L'(2n+1)-n}{\log_2 e} - \ln n^{\frac{3}{2}n}) \\ &= 2(n+1)[(L'(2n+1)+n)\frac{1}{\log_2 e} + \frac{3}{2}n\ln n] \end{aligned}$$

Damit haben wir eine untere Schranke für l. Für die Anzahl K der Iterationen bestimmen wir nun einen Wert von l, der obige Ungleichung erfüllt.

$$\ln n \leq n \Longrightarrow \frac{3}{2}n\ln n \leq 2n^2$$

$$[L'(2n+1)+n]\frac{1}{\log_2 e} < 2nL' + L' + n \overset{(L'=L+n\log n>n)}{<} 2nL' + 2L' \leq 2nL' + nL' = 3nL'$$

Es folgt, daß $\boxed{K = 6n(n+1)L' + 4(n+1)n^2}$ ein geeigneter Wert für l ist. Wie man sieht, ist die Anzahl der Iterationen polynomiell in n und L'.

1.5.4 Aufwand pro Iteration

Die Anzahl der arithmetischen Operationen pro Iteration beträgt $O((n+m)n)$. Einen wesentlichen Anteil daran macht die Multiplikation von Matrizen und Vektoren mit $O(n^2)$ aus.
Um polynomielle Laufzeitschranken einzuhalten, müssen wir den Zeitbedarf der arithmetischen Operationen einschränken. Dazu nutzen wir das folgende Lemma:

Lemma 5 *Es genügt, Operationen auf p Nachkomma-Stellen genau auszuführen mit $p = O(n^4 L')$.*

Durch die damit eingeführte Rechenungenauigkeit könnte das neue Ellipsoid das alte Halbellipsoid eventuell nicht ganz umfassen. Dann müßte B vergrößert werden, wodurch sich auch γ und die Laufzeit vergrößern.

Mit Hilfe des Ellipsoid–Algorithmus kann man also zeigen, daß man Lineare Programme in polynomieller Laufzeit lösen kann, allerdings stellte sich heraus, daß dieses Verfahren in der Praxis nur wenig Bedeutung erlangen kann. Erstens ist die reale Laufzeit schlecht, was sich auch bei feineren Schranken nur unwesentlich verbessert. Zweitens macht der Algorithmus aus einer schwach besetzten Anfangsmatrix in wenigen Schritten eine voll besetzte Matrix, kann den geringen Speicherbedarf (sowie den verminderten Rechenaufwand)für dünn besetzte Matrizen also nicht ausnutzen.
Für die theoretische Relevanz des Ellipsoid-Algorithmus ist entscheidend, daß die Tests $Az \leq b$ (Schritt 5) und $a_i z > b$ (Schritt 6) häufig mit schnelleren Verfahren durchgeführt werden können. In diesen Fällen ist der Algorithmus geeignet, zu beweisen, daß das Problem mit polynomiellem Aufwand lösbar ist.

Um Fälle mit $\mathrm{vol}(P) = 0$ bei $P \neq \emptyset$ auszuschließen, haben wir oben ein leicht verändertes System benutzt: wir haben uns darauf beschränkt, strikte Ungleichungen zu betrachten. Wir müssen noch zeigen, daß dieses veränderte System äquivalent zum alten System ist.

Lemma 6 *Seien $A \in \mathbb{R}^{m\times d}, x \in \mathbb{R}^d, b \in \mathbb{R}^m$. Dann ist $Ax \leq b$ lösbar genau dann, wenn $Ax < b + \lambda \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}$ lösbar ist.*

Beweis Jede Lösung der strikten Ungleichungen ist auch eine Lösung von $Ax \leq b$. Wir müssen nun also noch die umgekehrte Richtung zeigen. Dazu werden wir annehmen, daß eine Lösung x_0 von $Ax < b + \lambda(1,\ldots,1)^T$ existiert, und aus dieser dann eine für $Ax \leq b$ konstruieren. Zuerst erzeugen wir eine Matrix, die den Bedingungen der Cramerschen Regel genügt: Sei dazu J die Menge aller Zeilenvektoren $a_j, j = 1,\ldots,m$ von A.
Aus dieser bilden wir die Menge I aller linear unabhängigen Zeilenvektoren von A:
Sei $I = I(x_0) = \{a_i | b_i \leq a_i x_0 < b_i + \lambda, 1 \leq i \leq m\}$. Falls I_0 bereits leer ist, so ist x_0 eine Lösung von $Ax \leq b$, und wir sind fertig. Schrittweise finden wir alle $a_j \in J$, die von den $a_i \in I$ linear unabhängig sind:
Falls z_0 eine Lösung von $(a_i z = 0, a_i \in I, a_j z = 1, a_j \in J \backslash I)$ ist, löst $x_1 = x_0 + \varepsilon z_0$ mit hinlänglich kleinem ε ebenfalls $Ax < b + \lambda \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}$ und in $I(x_1) = \{a_i | b_i \leq a_i(x_0 + \varepsilon z_0) < b_i + \lambda\}$ ist jetzt auch a_j enthalten; d.h. es enthält mindestens einen Zeilenvektor mehr als $I(x_0)$. Nach höchstens m Schritten werden so alle linear unabhängigen Zeilen gefunden. Wir können also ohne Einschränkung annehmen, daß alle Zeilenvektoren von A linear abhängig sind von den Vektoren in I. Außerdem können wir I auch so verkleinern, daß I selbst eine linear unabhängige Menge von Vektoren ist: Falls das System $a_i z = 0, a_l z = 1$ für $i \neq l, a_i, a_l \in I$) eine Lösung hat, ist a_l linear unabhängig von den a_i. Sonst streichen wir a_l aus I.
Sei $|I| = k$ mit $k \leq d$, da es nur d Variablen $x_1,\ldots,x_d$ gibt. Falls $k < d$, füllen wir J mit $d-k$ von den $a_i \in I$ linear unabhängigen Einheitsvektoren auf (o.B.d.A. $e_1,\ldots,e_{d-k}$). Durch $d-k$–maliges Anwenden des obigen Verfahrens zum Aufspüren der $a_j \in J$, die von den $a_i \in I$ linear unabhängig sind, gelangen $e_1,\ldots,e_{d-k}$ in I.
I ist dann eine Basis des $\mathbb{R}^d$ ($|I| = d$) und nach Definition von I gilt auch für die ergänzten Einheitsvektoren: $b_i \leq e_i x_0 < b_i + \lambda, \quad i = 1,\ldots,d-k$.

Nun drücken wir alle Zeilen a_j von A als Linearkombination der $a_i \in I$ aus:

$$\underbrace{\begin{pmatrix} a_1 \\ \vdots \\ a_k \\ e_1 \\ \vdots \\ e_{d-k} \end{pmatrix}}_{d \times d\text{-Matrix}} \cdot \beta_j = a_j, \qquad \text{d.h.} \qquad a_j = \sum_{a_i \in I} \beta_{ji} a_i, \quad j = 1, \ldots, m$$

Die durch I definierte $d \times d$-Matrix hat vollen Rang. Also ist die Cramersche Regel anwendbar und es folgt:

$$\beta_{ji} = \frac{D_{ji}}{|D|} \qquad \text{mit} \qquad |D_{ji}| < 2^{L'} \quad ,|D| < 2^{L'}; \quad j = 1, \ldots, m; \quad i = 1, \ldots, d.$$

Als nächstes betrachten wir das $d \times d$–System $(a_i x = b_i, a_i \in I, i = 1, \ldots, k)$. Die a_i sind linear unabhängig, also existiert eine Lösung $\hat{x} \in \mathbb{R}^d$. Nach Definition von I gilt (auch für die ergänzten $e_1, \ldots, e_{d-k}$):

$$a_i \hat{x} < b_i + \lambda \quad ,\lambda = 2^{-L'} \Rightarrow \quad a_i \hat{x} < b_i + \frac{1}{|D|} \quad ,\text{da} \quad \frac{1}{|D|} > 2^{-L'} \Leftrightarrow \quad |D| \cdot (a_i \hat{x} - b_i) < 1.$$

Für $j = 1, \ldots, m$ rechnen wir dann folgendes nach:

$$\begin{aligned}
|D| \cdot (a_j \hat{x} - b_j) &= |D| a_j \hat{x} - |D| b_j \\
&= |D| \sum_{a_i \in I} \beta_{ji} a_i \hat{x} - |D| b_j, \quad \text{da} \quad a_j = \sum_{a_i \in I} \beta_{ji} a_i \\
&= \sum_{a_i \in I} D_{ji} a_i \hat{x} - |D| b_j, \quad \text{da} \quad \beta_{ji} = \frac{D_{ji}}{|D|} \\
&= \sum_{a_i \in I} D_{ji} b_i - |D| b_j, \quad \text{da} \quad a_i \hat{x} = b_i \quad \text{nach Def. von } \hat{x} \\
&= -\sum_{a_i \in I} D_{ji} a_i x_0 + \sum_{a_i \in I} D_{ji} b_i + \sum_{a_i \in I} D_{ji} a_i x_0 - |D| b_j \\
&= -\sum_{a_i \in I} D_{ji} a_i x_0 - \sum_{a_i \in I} (-D_{ji} b_i) + |D| a_j x_0 - |D| b_j, \quad \text{da} \quad \sum_{a_i \in I} D_{ji} a_i = |D| a_j \\
&= -\sum_{a_i \in I} D_{ji} (a_i x_0 - b_i) + |D| (a_j x_0 - b_j) \\
&\leq \sum_{a_i \in I} |D_{ji}| |a_i x_0 - b_i| + |D| (a_j x_0 - b_j) \\
&= \sum_{a_i \in I} |D_{ji}| (a_i x_0 - b_i) + |D| (a_j x_0 - b_j), \\
&\quad \text{da nach Def. von } I \text{ gilt:} \quad b_i \leq a_i x_0 \forall a_i \in I \Rightarrow a_i x_0 - b_i \geq 0; \\
&\quad \text{d.h.,} \quad a_i x_0 - b_i = |a_i x_0 - b_i| \\
&< \sum_{a_i \in I} \lambda |D_{ji}| + \lambda |D|, \quad \text{da nach Def. von } I \text{ gilt:} a_i x_0 - b_i < \lambda \\
&\quad \text{und } x_0 \text{ Lösung von} \quad Ax < b + \lambda \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \Rightarrow \quad a_j x_0 - b_j < \lambda \\
&= \lambda \left(\sum_{a_i \in I} |D_{ji}| + |D| \right) \\
&< 2^{-2L'} (m+1) 2^{L'}, \quad \text{da} \quad |D_{ji}| < 2^{L'}, |D| < 2^{L'}, |I| \leq m \\
&< 1.
\end{aligned}$$

Also ist $|D|(a_j\hat{x} - b_j) < 1$ für $j = 1, \ldots, m$. Nach Definition von D sind die Nenner aller Komponenten von $\hat{x}$ Teiler von $|D|$, und damit sind diese Größen ganzzahlig:

$$\begin{array}{rl} & |D|(a_j\hat{x} - b_j) \in \mathbb{Z} \quad \text{und} \quad |D|(a_j\hat{x} - b_j) < 1 \\ \Rightarrow & |D|(a_j\hat{x} - b_j) \leq 0 \\ \Rightarrow & a_j\hat{x} - b_j \leq 0, \quad \text{da} \quad |D| > 0 \\ \Leftrightarrow & a_j\hat{x} \leq b_j, \quad j = 1, \ldots, m; \end{array}$$

d.h. $\hat{x}$ ist Lösung von $Ax = b$. ◻

Vorlesung vom 30.5.1996
Oliver Plaehn, Henrik Stormer, Bianca Schröder

1.6 Karmarkars "Projektiver Skalierungsalgorithmus"

Wir werden uns jetzt mit einer weiteren Methode beschäftigen, Lineare Programme in polynomieller Zeit zu lösen, nämlich mit dem von Karmarkar entwickelten *Projektiven Skalierungsalgorithmus*.

Wir wollen folgendes Lineares Programm lösen:

$$\min v = c^T x, \qquad Ax = 0, \qquad x \in \Delta.$$

Dabei ist $\Delta = \{x \in \mathbb{R}^n \mid e^T x = 1, x \geq 0\}$ mit $e^T = (1, \ldots, 1)$ der Standardsimplex im $\mathbb{R}^n$ (siehe Abbildung 1.6).

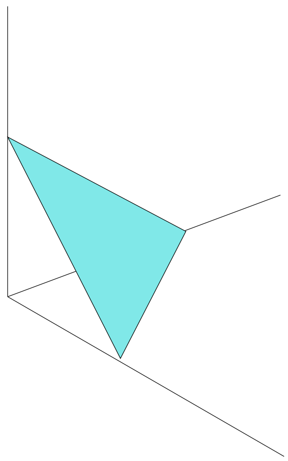


Abbildung 1.38: Der Standardsimplex im $\mathbb{R}^3$

Zusätzlich treffen wir noch folgende Annahmen:

1. A ist eine $m \times n$-Matrix mit Rang m,
2. $Ae = 0$, das heißt der Schwerpunkt e/n des Simplex ist eine zulässige Lösung,
3. der optimale Wert von v ist 0.

Übung: Man kann zeigen, daß man jedes Lineare Programm in die obige Standardform überführen kann, so daß auch die drei Annahmen erfüllt sind (Siehe Übungsblatt 4, Aufgabe 1).

Definition 15 *Ein Punkt x heißt genau dann* echt zulässig, *wenn x zulässig ist und $x > 0$ gilt.*

Die *Idee des Algorithmus* ist, eine Folge von echt zulässigen Lösungen $x^0 = e/n, x^1, x^2, \ldots, x^k, \ldots$ zu konstruieren, für die

$$c^T x^k \leq e^{-\frac{k}{5n}} c^T x^0$$

gilt. Hierbei ist e die Eulersche Zahl.
Das heißt, daß die x^k mit steigendem k immer besser werden, also gegen die Optimallösung streben.
Die Bitgröße der Matrix A ist $L = \sum_{i=0}^{m} \sum_{j=1}^{n} \lceil \log |a_{ij}| + 1 \rceil + 1$, mit $a_{0j} = c_j$.
Dann folgt:

$$\begin{aligned} c^T x^k &\leq e^{-\frac{k}{5n}} c^T x^0 \\ &= e^{-\frac{k}{5n}} c^T \frac{e}{n} \\ &\leq e^{-\frac{k}{5n}} 2^L \end{aligned}$$

Das bedeutet, daß man mit $k = \Theta(nL)$ bereits $c^T x^k \leq 2^{-L}$ erreicht, sich also bis auf Rechengenauigkeit der optimalen Lösung genähert hat. Man kann dann noch zu dieser Näherungslösung die Ecke des Polyeders ermitteln, die die genaue optimale Lösung repräsentiert, was wir hier aber nicht näher erläutern werden.
In der Praxis zeigt sich, daß mit leicht abgewandelten Algorithmen meist — auch bei größeren Problemen — 20 bis 50 Iterationen ausreichen, um sich der optimalen Lösung bis auf Rechengenauigkeit zu nähern.

Bleibt zu klären, wie man von x^k zu x^{k+1} kommt, wie also ein Iterationsschritt des Karmarkar-Algorithmus aussieht. Der Einfachheit halber treffen wir erst einmal die Annahme $x^k = e/n$, d.h. x^k ist der Schwerpunkt des Simplex Δ. Dann stellt sich die Frage, in welche Richtung man x^k bewegen muß, um eine bessere Lösung zu erhalten. Am schönsten wäre natürlich, wenn man x^k in negative c-Richtung verschieben könnte. Das geht aber nicht unbedingt immer, da sich der Zulässigkeitsraum auf einer Hyperebene im $\mathbb{R}^n$ befindet, der Vektor c aber nicht unbedingt auf dieser Hyperebene liegen muß. Also projezieren wir c in den Zulässigkeitsraum und verschieben x^k entlang dieser Projektion.
Die nächste Frage, die sich stellt, ist, wie weit man x^k verschieben kann, ohne den Zulässigkeitsraum zu verlassen. Eine Einschränkung des Zulässigkeitsraums stellt die Bedingung $x \in \Delta$ dar. Je nachdem, welche Richtung der Vektor c hat, kann man jedoch x^k verschieden weit verschieben, bis man auf die Simplexbegrenzung stößt. Um das zu vereinfachen schwächt man die Bedingung $x \in \Delta$ ab: Dazu legt man um den Schwerpunkt von Δ eine Kugel, die gerade noch im Simplex liegt. Der Radius dieser Kugel liefert dann die Entfernung, um die x^k verschoben werden darf.
Bleibt noch das Problem, wie man sich der Annahme $x^k = e/n$ entledigt. Dies erreicht man, indem man das ursprüngliche Problem durch eine Transformation so “umbiegt”, daß x^k Schwerpunkt des Standardsimplex wird.

Sei hierzu $\bar{x} = x^k$ und $\bar{Y} = diag(\bar{x})$, also $\bar{Y}e = \bar{x}$ (d.h. $\bar{Y}$ ist die Diagonalmatrix mit den Elementen von $\bar{x}$ auf der Hauptdiagonalen).
Jetzt können wir die Transformationsfunktion $F_{\bar{x}}$ angeben:

$$F_{\bar{x}} : x \mapsto \hat{x} = \frac{\bar{Y}^{-1} x}{e^T \bar{Y}^{-1} x}.$$

Die inverse Funktion hierzu ist dann:

$$F_{\bar{x}}^{-1} : \hat{x} \mapsto x = \frac{\bar{Y} \hat{x}}{e^T \bar{Y} \hat{x}}.$$

Schauen wir uns zur Verdeutlichung der Funktion $F_{\bar{x}}$ an, was diese mit $\bar{x}$ als Argument liefert:

$$\begin{aligned} F_{\bar{x}}(\bar{x}) &= \frac{\overbrace{\bar{Y}^{-1}\bar{x}}^{e}}{e^T \underbrace{\bar{Y}^{-1}\bar{x}}_{e}} \\ &= \frac{e}{\underbrace{e^T e}_{n}} \\ &= \frac{e}{n}. \end{aligned}$$

Das heißt, die Transformationsfunktion $F_{\bar{x}}$ bildet $\bar{x}$ auf den Schwerpunkt des Standardsimplex Δ ab.

Mit dem i-ten Einheitsvektor e_i als Argument liefert $F_{\bar{x}}$:

$$\begin{aligned} F_{\bar{x}}(e_i) &= \frac{\overbrace{\bar{Y}^{-1}e_i}^{\bar{x}_i^{-1}e_i}}{\underbrace{e^T \underbrace{\bar{Y}^{-1}e_i}_{\bar{x}_i^{-1}e_i}}_{\bar{x}_i^{-1}}} \\ &= e_i, \end{aligned}$$

d.h. die Ecken des Simplex werden auf sich selbst abgebildet.

Außerdem kann man zeigen, daß $F_{\bar{x}}(\Delta) = \Delta$ ist, was wir hier aber nicht prüfen werden.

Die allgemeine Vorgehensweise, um von x^k zu x^{k+1} zu gelangen, läßt sich also zusammenfassend folgendermaßen beschreiben:

- Das Ausgangsproblem wird der Transformation $F_{\bar{x}}$ unterworfen, wobei (wie oben gesehen) x^k auf den Schwerpunkt e/n des Standardsimplex abgebildet wird.
- Dann wird die transformierte Zielfunktion auf einer dem zulässigen Bereich eingeschriebenen Kugel minimiert und so ein Punkt $\hat{x}$ ermittelt.
- Schließlich wird die Transformation rückgängig gemacht, indem auf den ermittelten Punkt $\hat{x}$ die Umkehrfunktion $F_{\bar{x}}$ angewendet wird, und man erhält x^{k+1}.

Nach der Transformation erhalten wir das folgende Problem:

$$\min \frac{c^T\bar{Y}\hat{x}}{e^T\bar{Y}\hat{x}}, \qquad \frac{A\bar{Y}\hat{x}}{e^T\bar{Y}\hat{x}} = 0, \qquad \hat{x} \in \Delta.$$

Das heißt, wir haben jetzt keine lineare Funktion mehr, die wir zu optimieren versuchen, sondern eine rationale. Das macht aber nichts, denn wegen Annahme 3 muß der optimale Wert 0 sein, daß heißt, der Zähler muß 0 werden, wobei wir den Nenner ignorieren können.

Unser neues Problem lautet dann:

$$\min \hat{c}^T\hat{x}, \qquad \hat{A}\hat{x} = 0, \qquad \hat{x} \in \Delta,$$

mit $\hat{c} = \bar{Y}c$ und $\hat{A} = A\bar{Y}$.

Weil $\bar{Y}$ eine Diagonalmatrix ist, bedeuten $\hat{c} = \bar{Y}c$ und $\hat{A} = A\bar{Y}$ lediglich, daß c und jede Spalte von A skaliert werden. Mit der Berechnung von $\hat{c}$ und $\hat{A}$ ist also kein großer Rechenaufwand verbunden.

Um beim Übergang von x^k zu x^{k+1} nicht aus dem Standardsimplex "herauszufallen", muß die Kugel, auf der die neue Zielfunktion minimiert werden soll, vollständig im Standardsimplex enthalten sein.
Die Frage ist also, wie groß man den Radius der Kugel wählen kann, so daß sie gerade noch im Standardsimplex enthalten ist.

Formal sieht eine Kugel mit Radius ρ um den Mittelpunkt e/n folgendermaßen aus:

$$K(\frac{e}{n}, \rho) = \{x \in \mathbb{R}^n \;:\; e^Tx = 1, \| x - \frac{e}{n} \| \leq \rho\}.$$

Wir suchen den größtmöglichen Radius r mit $K(\frac{e}{n}, r) \subseteq \Delta$. r hat dann folgende Gestalt: $r = \| \frac{e}{n} - (e - e_1)\frac{1}{n} \|$, und daraus folgt $r = \frac{1}{\sqrt{n(n-1)}}$.

Das neue (etwas abgeschwächte) Problem lautet also:

$$\min \hat{c}^T\hat{x}, \qquad \hat{A}\hat{x} = 0, \qquad \hat{x} \in K(\frac{e}{n}, r),$$

mit $\hat{c} = \bar{Y}c$ und $\hat{A} = A\bar{Y}$.
Um den Wert von $\hat{c}^T\hat{x}$ optimal verbessern zu können, müssen wir $\hat{c}$ erst einmal in den Zulässigkeitsraum projezieren. Dies geschieht durch die orthogonale Projektion von $\hat{c}$ in den Nullraum von $B = \begin{bmatrix} \hat{A} \\ e \end{bmatrix}$. Der Projektionsvektor $\hat{p}$ hat folgende Gestalt:

$$\hat{p} = (I - B^T(BB^T)^{-1}B).$$

Dann ist

$$\hat{d} = -(I - B^T(BB^T)^{-1}B)\hat{c}$$

die gesuchte Projektion von $\hat{c}$.
Eine Lösung des obigen Optimierungsproblems ist damit gegeben durch

$$\hat{x}(\rho) = \frac{e}{n} + \rho\frac{\hat{d}}{\| \hat{d} \|}$$

für $\rho \leq r$.

Wähle $\rho = \frac{\alpha}{n} < \alpha r$, $\alpha = 1/3$.

Der projektive Skalierungsalgorithmus von Karmarkar sieht dann so aus:

Eingabe: $m \times n$ Matrix A, Vektor $c \in \mathbb{R}^n$, so daß die Annahmen 1 bis 3 erfüllt sind,
$q \in \mathbb{N}$

Ausgabe: ein zulässiger n-Vektor x mit $c^Tx \leq 2^{-q}c^T\frac{e}{n}$

$x^0 = \frac{e}{n}$
for $k = 0$ **to** $5nq$ **do**
  if $c^Tx^k \leq 2^{-q}c^T\frac{e}{n}$ **then** return $x = x^k$
  $\bar{Y} = diag(x^k)$
  $\hat{c} = \bar{Y}c$
  $\hat{A} = A\bar{Y}$
  $B = \begin{bmatrix} \hat{A} \\ e \end{bmatrix}$
  $\hat{d} = -(I - B^T(BB^T)^{-1}B)\hat{c}$
  $\hat{x} = \frac{e}{n} + \frac{\alpha}{n}\frac{\hat{d}}{\|\hat{d}\|} \qquad (\alpha = 1/3)$
  $x^{k+1} = \frac{\bar{Y}\hat{x}^k}{e^T\bar{Y}\hat{x}^k} \quad (= F^{-1}(\hat{x}))$

Pro Iteration benötigt der Algorithmus $O(n^3)$ arithmetische Operationen auf Genauigkeit L, also $O(n^3L)$ Bit-Operationen. Damit ergibt sich eine Gesamtlaufzeit von $O(n^4L^2)$.
Am Schluß bleibt noch, den Algorithmus zu analysieren. Warum gilt $c^Tx^k \leq e^{-k/5n}c^Tx^0$? Dazu definieren wir zuerst eine Potentialfunktion $f(x)$, die folgende Gestalt hat:

$$f(x) := f(x,c) = \sum_{1 \leq j \leq n} \ln(\frac{c^Tx}{x_j}) = n\ln(c^Tx) - \sum_{1 \leq j \leq n} \ln(x_j)$$

Weiterhin definieren wir $\hat{f}(\hat{x}) := f(\hat{x}, \hat{c})$.

Über diese Potentialfunktion werden im folgenden drei Aussagen gemacht, die für die Abschätzung der Funktion notwendig sind. Die Beweise dazu finden sich in den Übungen.

Lemma 7 *Sei* $\hat{x} = \frac{\bar{Y}^{-1}x}{e^T\bar{Y}^{-1}x}$ *gegeben. Dann gilt für die Potentialfunktion* $\hat{f}(\hat{x}) = f(x) + \ln(\det(\bar{Y}))$.

Lemma 8 *Sei $\hat{x} = \frac{e}{n} + \frac{\alpha}{n}\frac{\hat{d}}{\|\hat{d}\|}$. Dann gilt folgende Ungleichung:* $n\ln(\hat{c}^T\hat{x}) \leq n\ln(\frac{\hat{c}^Te}{n}) - \alpha$.

Lemma 9 *Sei $\hat{x} \in K(\frac{e}{n}, \frac{\alpha}{n})$. Dann gilt folgende Ungleichung:* $-\sum_{1\leq j\leq n} \ln(\hat{x}_j) \leq -\sum_{1\leq j\leq n} \ln(\frac{1}{n}) + \frac{\alpha^2}{2(1-\alpha)^2}$.

Zusätzlich wird noch das folgende Korollar aufgestellt:

Korollar 2 *Für die Potentialfunktion $\hat{f}$ gilt:*

$$\hat{f}(\hat{x}) \leq \hat{f}(\frac{e}{n}) - \alpha + \frac{\alpha^2}{2(1-\alpha)^2}$$

Mit $\alpha = \frac{1}{3}$ ergibt sich dann:

$$\hat{f}(\hat{x}) \leq \hat{f}(\frac{e}{n}) - \frac{1}{5}$$

Bevor wir das Korollar beweisen, zeigen wir erst einmal, wozu man es benutzen kann. Dieses Korollar impliziert, daß die für die Laufzeitabschätzung wichtige Bedingung $c^Tx^k \leq e^{-k/5n}c^Tx^0$ für unseren Algorithmus erfüllt ist. Zunächst einmal folgt aus dem Algorithmus, daß $F^{-1}(\hat{x}) = x^{k+1}$ ist. Außerdem gilt $F^{-1}(\frac{e}{n}) = x^k$. Mit dem Korollar und Lemma 7 können wir nun abschätzen:

$$f(x^{k+1}) \leq f(x^k) - \frac{1}{5},$$

und daraus folgt

$$f(x^k) \leq f(\underbrace{x^0}_{\frac{e}{n}}) - \frac{1}{5}k.$$

Setzt man in die letzte Ungleichung die Definition von f ein, so erhält man:

$$n\ln(c^Tx^k) - \underbrace{\sum_{1\leq j\leq n} \ln x_j}_{\leq \sum_{1\leq j\leq n} \ln(\frac{1}{n})} \leq n\ln(c^T\frac{e}{n}) - \sum_{1\leq j\leq n} \ln(\frac{1}{n}) - \frac{1}{5}k,$$

bzw.

$$\ln c^Tx^k \leq \ln c^T\frac{e}{n} - \frac{1}{5}\frac{k}{n}.$$

Potenziert man jetzt noch beide Seiten der Ungleichung, so ergibt sich die gewünschte Abschätzung:

$$c^Tx^k \leq c^T\frac{e}{n} \cdot e^{-\frac{1}{5}\frac{k}{n}}.$$

Als Letztes bleibt nun noch der Beweis des Korollars:

Beweis

$$\hat{f}(\hat{x}) = f(\hat{x}, \hat{c}) = n\ln(\hat{c}^T\hat{x}) - \sum_{1\leq j\leq n} \ln(\hat{x}_j).$$

Mit Hilfe von Lemma 8 und Lemma 9 läßt sich $\hat{f}(\hat{x})$ nach unten abschätzen:

$$\hat{f}(\hat{x}) \leq n\ln(\frac{\hat{c}^Te}{n}) - \alpha - \sum_{1\leq j\leq n} \ln(\frac{1}{n}) + \frac{\alpha^2}{2(1-\alpha)^2}.$$

Wegen $\sum_{1\leq j\leq n} \ln(\frac{1}{n}) = n\ln(\frac{1}{n})$ gilt also:

$$\hat{f}(\hat{x}) \leq n\ln(\hat{c}^Te) - \alpha + \frac{\alpha^2}{2(1-\alpha)^2}.$$

Das ist fast das Korollar. Es bleibt noch zu zeigen, daß $\hat{f}(\frac{e}{n})$ wirklich gleich $n\ln(\hat{c}^Te)$ ist:

$$\hat{f}(\frac{e}{n}) = f(\frac{e}{n}, \hat{c}) = n\ln(\hat{c}^T\frac{e}{n}) - \underbrace{\sum_{1\le j\le n} \ln(\frac{1}{n})}_{=n\ln(\frac{1}{n})} = n\ln(\hat{c}^Te).$$

$\square$

Vorlesung vom 3.6.1996
Holger Gebhard

1.7 Das Revidierte Simplex Verfahren

1.7.1 Simplexschritt via Wörterbuch

In diesem Abschnitt kommen wir noch einmal auf primale und duale Lineare Programme, die mit Hilfe des Simplex-Algorithmus gelöst werden, zurück. Wir werden sehen, daß man das bisherige Simplex-Verfahren auch ausschließlich durch Operationen auf Matrizen darstellen kann. Zunächst aber zur Erinnerung noch einmal die Problemstellung und das bisher entwickelte Simplexverfahren:

$$\begin{array}{llcll} \text{(primal)} & \max c^Tx & \qquad & \text{(dual)} & \min y^Tb \\ & \mathrm{Ax = b} & & & y^TA \ge c \\ & x \ge 0 & & & \end{array}$$

Dabei hat A vollen Rang und ist von der Form:

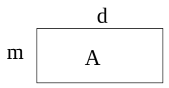


Abbildung 1.39: Koeffizientenmatrix

Der bisherige Simplex-Algorithmus arbeitete mit Hilfe eines Wörterbuch. Um das aufzustellen, benötigt man eine Basis B (eine Teilmenge von Spaltenindizes), und mit dieser Basis hat das Wörterbuch die Form:

$$\begin{aligned} x_i &= \overline{b_i} + \sum_{j\notin B} \overline{a_{ij}} x_j \qquad i \in B, \\ z &= v + \sum_{j\notin B} \overline{c}_j x_j. \end{aligned}$$

Eine Lösung des linearen Gleichungssystems besteht aus $x_i = b_i$ für alle Variablen x_i, die in der Basis vorkommen. Die Nicht-Basisvariablen setzt man auf 0.
Ein Simplexschritt besteht dann aus folgenden Schritten:

1. Finde j^*, so daß $\overline{c}_{j^*} > 0$ (j^* Index, mit dem der Zielfunktionswert verbessert werden kann).

2. In Spalte j^* bestimme $\overline{a}_{ij^*} < 0$ (Kandidaten, die die Basis verlassen).

3. Finde i^*, so daß $\frac{-b_i}{a_{ij^*}}$ minimal (Kandidat wird festgelegt).

4. Neue Basis $\hat{B} = B \setminus \{i^*\} \cup \{j^*\}$ (Austauschschritt durchführen).

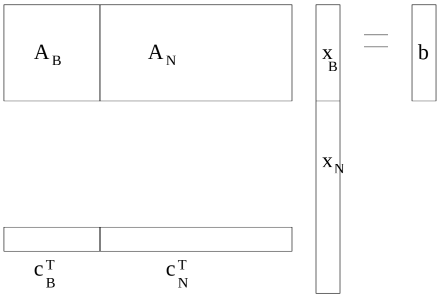


Abbildung 1.40: Anfangsform des Problems

1.7.2 Revidierter Simplexschritt

Hat man eine Basis B gegeben, so kann man das Wörterbuch und die Zielfunktion auch durch Matrizenoperationen erhalten. Dazu schreibt man die Matrix A, den Vektor b und den Zielvektor c so auf, daß zuerst die zu den Basisvariablen x_i, $i \in B$, gehörigen Spalten bzw. Einträge kommen (siehe Abbildung 1.40). Das Lineare Programm ändert sich dadurch nicht.
Das Gleichungssystem hat also folgende Form:

$$\begin{aligned} A_B x_B + A_N x_N &= b \quad \text{, woraus} \\ x_B + A_B^{-1} A_N x_N &= A_B^{-1} b \quad \text{, und schließlich} \\ A_B^{-1} b - A_B^{-1} A_N x_N &= x_B \end{aligned}$$

folgt. Setze die Gleichung für x_B in die Zielfunktion ein:

$$\begin{aligned} z &= c_B^T x_B + c_N^T x_N \\ &= c_B^T (A_B^{-1} - A_B^{-1} A_N x_N) + c_N^T x_N \\ &= \underbrace{c_B^T A_B^{-1}}_{y^T} b + (c_N^T - \underbrace{c_B^T A_B^{-1}}_{y^T} A_N) x_N . \end{aligned}$$

Man erält also das gleiche Wörterbuch wie beim normalen Simplex-Algorithmus, nur algebraisch durch Matrizenoperationen aufgeschrieben. Mit Hilfe dieses Wörterbuchs

$$\begin{aligned} x_B &= A_B^{-1} b - A_B^{-1} A_N x_N \\ z &= y^T b + (c_N^T - y^T A_N) x_N \end{aligned}$$

führen wir jetzt einen revidierten Simplex-Schritt durch, der das Gleiche wie der normale Simplex tut. Allerdings ergibt sich das neue Wörterbuch sofort aus der neuen Basis, nicht durch Umstellen und Verändern des alten Wörterbuchs.
Revidierter Simplex-Schritt:

1. Bestimme $y^T = c_B^T A_B^{-1}$, d.h. löse $y^T A_B = c_B^T$ nach y auf.
2. Finde j^*, so daß $c_{j^*}^T - y^T A_{j^*} > 0$. Gesucht wird also wieder der Index einer Nichtbasisvariable, die durch Erhöhen den Zielfunktionswert verbessert, denn $c_B^T A_B^{-1} b$ ist der bisherige Wert (v), der sich vergößern kann, falls $c_{j^*}^T - y^T A_{j^*} > 0$ und die zugehörige Variable ungleich 0 gesetzt wird (siehe Zielfunktionszeile).
3. Bestimme Spalte $\overline{A}_{j^*} = A_B^{-1} A_{j^*}$ und daraus die positiven Elemente, denn das sind Kandidaten für das Verlassen der Basis.
4. Bestimme $\overline{b} = A_B^{-1} b$. Finde mit Hilfe von A_{j^*} analog zum normalen Simplex-Verfahren den Index i^*, der die Basis verläßt.
5. Neue Basis $\hat{B} = B \setminus \{i^*\} \cup \{j^*\}$, siehe Abbildung 1.41.

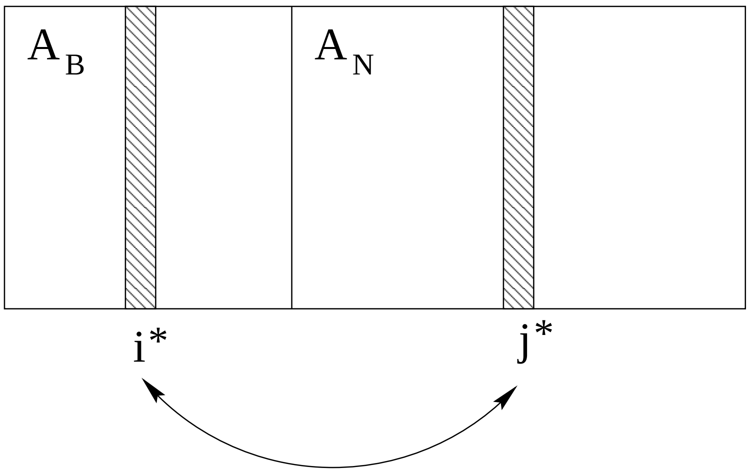


Abbildung 1.41: Matrix A beim revidierten Simplexschritt

Die Idee bei diesem Algorithmus ist, daß er schneller als der herkömmliche Simplex ist, falls die Rechenoperationen auf den gegebenen Matrizen und Vektoren billig sind (dünn besetzte Matrizen bleiben dünn besetzt, im Gegensatz zur Ellipsoidmethode).

Die Variable y, die man sich bei diesem Algorithmus definiert, bezeichnet man als *duale Variable*. Für die optimale Basis B gibt $y^T = c_B^T A_B^{-1}$ die optimale duale Lösung an.

1.8 Netz-Simplex (Network Simplex)

1.8.1 Das Transportproblem

Es sei $G = (V, E)$ ein gerichteter Graph mit folgender Interpretation:

- Die Knoten stellen Städte dar. Für alle Knoten $i \in V$ sei b_i das Angebot ($b_i \leq 0$) oder die Nachfrage ($b_i > 0$) des Knotens i an Orangen. Wir nehmen an, daß der Graph einen geschlossenen Wirtschaftsraum darstellt, in dem sich Angebot und Nachfrage entsprechen:
$$\sum_{i \in V} b_i = 0.$$
- Die Kanten sind die Verbindungswege zwischen den Städten. Für jede Kante $e \in E$ seien c_e die Kosten des Transports von Orangen über die Kante e.

Notation: Hat eine Kante $e \in E$ die Form von Abb. 1.42, dann schreibe $e = [tail(e), head(e)\rangle = [tl(e), hd(e)\rangle$.

Abbildung 1.42: gerichtete Kante

Die Aufgabe, die Nachfrage möglichst kostengünstig zu befriedigen, kann man mit Hilfe eines Linearen Programms folgendermaßen darstellen:
$$\min \sum_{e \in E} c_e x_e,$$
unter den Bedingungen, daß $x_e \geq 0$ und für alle Knoten $v \in V$
$$\sum_{e:hd(e)=i} x_e - \sum_{e:tl(e)=i} x_e = b_i$$
gilt. x_e ist dann die Menge, die über die Kante e transportiert wird, und die Bedingungen besagen, daß in jedem Knoten die Gesamtmenge von dem, was eintrifft, minus die Gesamtmenge von dem, was verschickt wird, gleich dem Lagerbestand sein soll.

Die Aufgabenstellung kann man noch variieren, indem man den Transportwegen haben Kapazitäten ($l_e \le x_e \le u_e$) zuordnet, d.h. die Menge der Ware, die über eine Kante tranportiert wird, muß in einem bestimmten Bereich liegen.

Beispiel Gegeben sei der Graph wie in Abbildung 1.43.

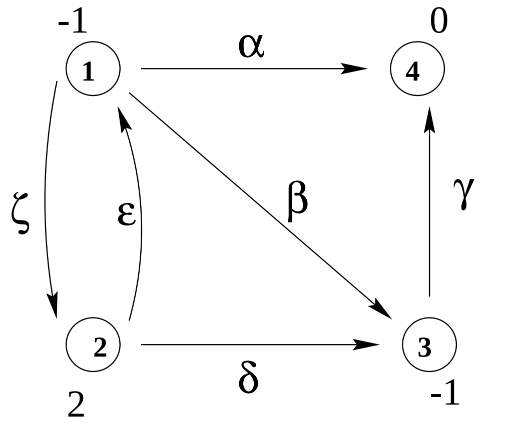


Abbildung 1.43: Ein Transportproblem

Dieses Transportproblem führt zu folgendem Optimierungsproblem:

$$\min c_\alpha x_\alpha + c_\beta x_\beta + c_\gamma x_\gamma + c_\delta x_\delta + c_\epsilon x_\epsilon + c_\zeta x_\zeta$$

unter den Bedingungen $x_\alpha, x_\beta, \ldots, x_\zeta \ge 0$ und

$$\begin{array}{lrrrrrrrrrrrrl}
\text{1. Knoten} & - & x_\alpha & - & x_\beta & & & & & + & x_\epsilon & - & x_\zeta & = -1 \\
\text{2. Knoten} & & & & & & & - & x_\delta & - & x_\epsilon & + & x_\zeta & = 2 \\
\text{3. Knoten} & & & & x_\beta & - & x_\gamma & + & x_\delta & & & & & = -1 \\
\text{4. Knoten} & & x_\alpha & & & + & x_\gamma & & & & & & & = 0
\end{array}$$

Beobachtung

- Die Koeffizientenmatrix hat nicht vollen Rang, da alle Zeilen aufaddiert 0 ergeben.
- Jeder Spalte entspricht einer Kante.

Beispiel Kürzeste Wege in Graphen
Gegeben sei ein Graph $G = (V, E)$ mit $|V| =$ n. Jede Kante $e \in E$ habe ein Gewicht w_e. Den Knoten ordnet man Nachfragen zu: $b_1 = -n + 1$ und $b_i = 1$ für alle anderen Knoten $i \in \{2, 3, 4...n\}$. Die Kanten erhalten als Transportkosten $c_e = w_e$ für Transportprobleme, bei denen die Kanten Kapazitätsschranken haben, setzt man noch $l_e = 0$ und $u_e = n$).
Knoten 1 hat damit für jeden anderen Knoten eine Lieferung von 1 Einheit. Da die Kosten beim Transportproblem minimiert werden, werden dabei auch die kürzesten Wege von Knoten 1 zu jedem anderen Knoten bestimmt.

Beispiel Maximaler Fluß
Es sei wieder $G = (V, E)$ ein Graph und s und t seien Knoten des Graphen. Es seien weiterhin Kantenkapazitäten $k_e > 0$ gegeben. Um den maximalen Fluß mit Hilfe eines Transportproblems berechnen zu können, füge in den Graphen eine neue Kante $e' = [t, s\rangle$ ein. Setze für den erweiterten Graphen $b_i = 0$ für alle Knoten i und $l_e = 0$, $u_e = k_e$, $c_e = 0$ für alle Kanten $e \in E$. Die neu hinzugefügte Kante e' erhält als Transportkosten $c_{e'} = -1$.

Je mehr man von s nach t transportieren kann (maximaler Fluß), desto größer wird der Rückfluß und damit werden die Kosten immer kleiner.

Beispiel Gewichtetes Zuweisungsproblem
Gegeben seien N_1 Personen und N_2 Dinge mit $|N_1| = |N_2| = n$. Betrachte die Elemente aus N_1 und N_2 als Knoten in einem bipartiten Graphen. Kanten verlaufen nur zwischen Knoten aus N_1 und N_2, wobei jede Kante $[i,j]$ ein Gewicht $c_{i,j}$ hat. Bestimmte Dinge können also bestimmten Personen mit bestimmten Kosten zugeordnet werden. Die Aufgabe besteht nun darin, eine $1-1$-Zuweisung zwischen den Dingen und den Personen zu finden, die die Gesamtkosten minimiert. Setze dazu $b_i = -1$ für $i \in N_1$, $b_j = +1$ für $j \in N_2$ und $l_e = 0$, $u_e = 1$ für alle Kanten e in dem Graphen. Das so definierte Transportproblem löst das Zuweisungsproblem. Jede Person muß eine Einheit an ein Ding abgeben und jedes Ding kann genau eine Einheit bekommen.

Im folgenden wird vorausgesetzt, daß ein Graph $G = (V, E)$ immer zusammenhängend ist, und daß auch G ohne den ersten Knoten und die dazu adjazenten Kanten noch zusammenhängend ist.

Definition 16 *Eine $n \times d$ Matix A heißt* Knoten-Kanten-Inzidenzmatrix *von $G = (V, E)$ mit $|V| = n$ und $|E| = d$, wenn die Einträge a_{ie} folgende Form haben:*

$$a_{i,e} = \begin{cases} +1 & : \ hd(e) = i \\ -1 & : \ tl(e) = i \\ 0 & : \ sonst. \end{cases}$$

Wir bezeichnen im weiteren mit $\hat{A}$ die Matrix, die aus der Matix A durch Weglassen der ersten Spalte entsteht. Für $B \subset E$ mit $|B| = n - 1$ sei $\hat{A}_B$ die Matrix mit den Spalten $\hat{A}_e$ für Kanten $e \in B$.

Wie muß B beschaffen sein, damit $\hat{A}_B^{-1}$ existiert?
Wir überlegen uns die Antwort auf diese Frage mit Hilfe des obigen Beispiels:

Sei $B = \{\beta, \epsilon, \delta\}$, d.h. wir haben einen Teilgraphen von Abbildung 1.43:

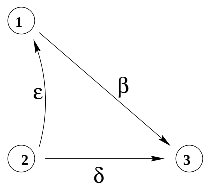


Abbildung 1.44: Teilgraph des Standardgraphen

Betrachtet man die zu diesem Graphen gehörige Knoten-Kanten-Adjazenzmatrix

$$\begin{pmatrix} -1 & 0 & +1 \\ 0 & -1 & -1 \\ +1 & +1 & 0 \end{pmatrix},$$

so sieht man, daß diese Matrix nicht invertierbar ist (alle Zeilen summieren zu 0 auf). Enthält also die Kantenmenge B einen Zyklus (ungerichtet betrachtet), dann ist $\hat{A}_B$ nicht invertierbar, da die Zeilen sich zu 0 addieren.

Behauptung 3 *Sei A eine $n \times d$ Knoten-Kanten-Inzidenzmatrix eines Graphen $G = (V, E)$ wie oben und $B \subset E$ mit $|B| = n - 1$. Dann gilt: $\hat{A}_B$ ist genau dann invertierbar, wenn B (ungerichtet betrachtet) ein aufspanndender Baum von G ist*

Beweis
"$\Rightarrow$" Ist B mit $|B| = n - 1$ kein aufspannender Baum von G, dann enthält der Teilgraph einen Zyklus. Das bedeutet, daß $\hat{A}_B$ nicht invertierbar ist.
"$\Leftarrow$" Betrachte Knoten 1 als Wurzel des Baumes und nummeriere die restlichen Knoten gemäß Präordnung durch. Anschließend nummeriere die Kanten folgendermaßen: Wenn e =(Vater, Sohn) , dann nummeriere e mit der Nummer des Sohnes (Siehe Abbildung 1.45).

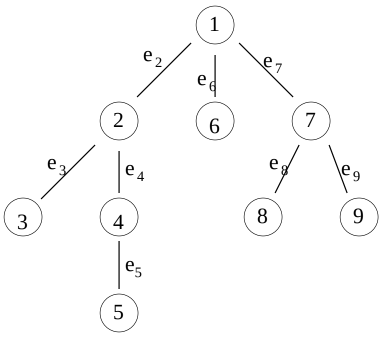


Abbildung 1.45: Beispiel eines durchnummerierten Graphen

Ordne die Matrix A_B nach der Nummerierung ihrer Spalten (Knotennummern) um, und erhalte eine Matrix der Form

$$\begin{array}{cccccc}
 & e_2 & e_3 & e_4 & e_5 & e_6 \\
1 & 1 & & & & \\
2 & 1 & 1 & 1 & & \\
3 & & 1 & & & \\
4 & & & 1 & 1 & 1 \\
5 & & & & 1 & \\
6 & & & & & 1 \\
7 & & & \ldots & & \\
\vdots & & & & &
\end{array}$$

Die so umgeordnete Matrix $\hat{A}_B$ ist eine obere Dreiecksmatrix, auf deren Diagonale nur die Werte $+1$ und -1 stehen. $\hat{A}_B$ ist also invertierbar. $\square$

Vorlesung vom 10.6.1996
Jörg Wallerich, Wolfgang Wonner

1.8.2 Netzwerk-Simplexschritt

Im folgenden wollen wir ein Verfahren angeben, wie man ausgehend von einer zulässigen Lösung für das Transportproblem, die durch einen aufspannenden Baum gegeben wird, einen Netzwerk-Simplexschritt durchführt und damit die Zielfunktion verbessert. Betrachte dazu ein Beispiel:

Beispiel Zu folgendem Graphen (siehe Abbildung 1.46)

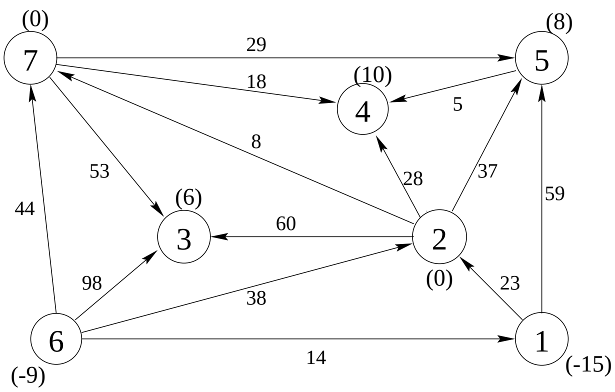


Abbildung 1.46: Beispiel-Graph

zeigt Abbildung 1.47 einen aufspannenden Baum.

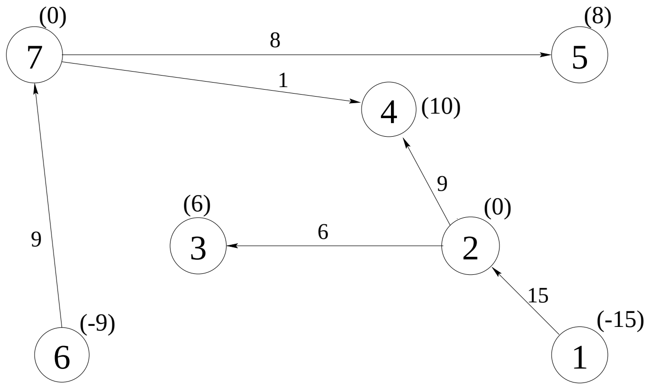


Abbildung 1.47: Aufspannender Baum

Die Kanten sind mit dem Fluß, der über sie fließt, bezeichnet. Man sieht, daß man eine zulässige Lösung für das Transportproblem gefunden hat, wenn man noch über die restlichen Kanten, die nicht zum auspannenden Baum gehören, Fluß mit Wert Null laufen läßt.

Allgemein gilt: Falls $(x_1^*, \ldots, x_n^*) = x*$ eine zulässige Lösung für einen aufspannenden Baum ist, so erhält man eine zulässige Lösung für $Ax = b$, $x \geq 0$, indem man x^* mit Nullen auffüllt. Man setzt $x_{[i,j\rangle} = 0$, falls $[i,j\rangle \notin B$.

Wir wollen jetzt die so gegebene zulässige Lösung mit Hilfe des Netzwerk-Simplex-Algorithmus verbessern. Jede Iteration in diesem Algorithmus besteht aus drei Phasen:

1. Erstelle Menge von "fairen" Preisen y_i aus Einkaufs- und Transportkosten: Wenn an Knoten i die Orangen y_i DM kosten und Knoten i und Knoten j durch eine Kante mit Transportkosten $c_{i,j}$ verbunden sind, dann sollen die Orangen an Knoten j $y_i + c_{i,j} = y_j$ DM kosten. Jeder Knoten i bekommt also einen fairen Preis y_i für Orangen zugeordnet:

$$y_i + c_{ij} = y_j, \qquad \text{für alle Kanten } [i,j\rangle \in B.$$

Man sieht leicht, daß dann natürchlich auch $y_1 + d, y_2 + d, \ldots, y_n + d$ für eine beliebige Konstante d faire Preise sind.

Wie sieht das bei unserem Beispiel aus?

Beispiel Setze $y_1 = 0$. Daraus folgt

$$\begin{array}{rcllr} y_2 & = & 23 & & \\ -y_2 + y_3 & = & 60 & \Longrightarrow & y_3 = 83 \\ -y_2 + y_4 & = & 28 & \Longrightarrow & y_4 = 51 \\ -y_7 + y_4 & = & 18 & \Longrightarrow & y_7 = 33 \\ -y_7 + y_5 & = & 29 & \Longrightarrow & y_5 = 62 \\ y_7 - y_6 & = & 44 & \Longrightarrow & y_6 = -11 \end{array}$$

Man sieht, daß das Gleichungssystem die Form $y^T \hat{A}_B = c_B$ hat. Die Lösung ist also eindeutig und entspricht der Berechnung der dualen Variable im 1. Schritt des Revidierten Simplex-Algorithmus.

Für eine Menge von fairen Preisen $y^T := (y_1, y_2, \cdots, y_n)$ definiere

$$\bar{c}^T := c^T - y^T A.$$

Es gilt

$$\begin{array}{rcl} \bar{c}_{ij} & = & c_{ij} + y_i - y_j = 0, \qquad \forall\ [i,j\rangle \in B, \text{ sowie} \\ x_{ij} & = & 0, \qquad \forall\ [i,j\rangle \notin B \end{array}$$

Daraus folgt $\bar{c}^T x = 0$.

Umgekehrt sei $\bar{x}$ eine andere zulässige Lösung für $A\bar{x} = b$, $\bar{x} \geq 0$. Dann gilt:

$$\begin{aligned} c^T \bar{x} &= \bar{c}^T \bar{x} + y^T A \bar{x} \\ &= \bar{c}^T \bar{x} + y^T b. \end{aligned}$$

Insbesondere folgt daraus für $x = \bar{x}$:

$$c^T x = \bar{c}^T x + y^T b = y^T b,$$

also

$$c^T \bar{x} = c^T x + \bar{c}^T \bar{x}.$$

2. Ein Konkurrent könnte jetzt versuchen, die Preise zu verbessern, indem er die Orangen über andere, billigere Kanten transportiert: Finde Kante $[u, v\rangle$ mit $y_u + c_{uv} < y_v$, z.B. die Kante $[1, 5\rangle$: $y_5 - y_1 - c_{1,5} = 62 - 0 - 59 = 3 \geq 0$. Dieser Schritt entspricht dem zweiten Schritt im Revidierten Simplex-Algorithmus.

   Wenn keine solchen Kanten mehr existieren, dann ist die Lösung optimal, denn aus $c_{ij} + y_i - y_j \geq 0$ für alle Kanten $[i, j\rangle$ im Baum folgt $\bar{c} \geq 0$, also $\bar{c}^T \bar{x} \geq 0$ für alle $\bar{x} \geq 0$. Damit ergibt sich $c^T \bar{x} \geq c^T x$ für alle zulässigen Lösungen $\bar{x}$.

3. Wir kennen den Plan des Gegners und versuchen ihm zuvorzukommen, indem wir t Orangen über den Weg $[u, v\rangle$ schicken. Je größer dabei das t ist, desto geringer wird die Zielfunktion.

   Da B ein aufspannender Baum ist, existiert genau ein Pfad zwischen u und v. $B \cup \{[u, v\rangle\}$ enthält also genau einen Zyklus. Gehe in B von u nach v und ändere den Flüß, der über die Vorwärtskanten (Kanten, die in Richtung von u nach v verlaufen) und Rückwärtskanten fließt:

$$\bar{x}_{ij} = \begin{cases} x_{ij} - t \ , [i, j\rangle & \text{Rückwärtskante} \\ x_{ij} + t \ , [i, j\rangle & \text{Vorwärtskante} \\ x_{ij} \qquad , [i, j\rangle & \text{Kante nicht im Zyklus.} \end{cases}$$

Beispiel Die Kante $[1, 5\rangle$ kommt zu dem aufspannenden Baum B dazu und erhält den Fluß t: $x_{1,5} = t$. Der zugehörige Graph ist in Abbildung 1.48 dargestellt.

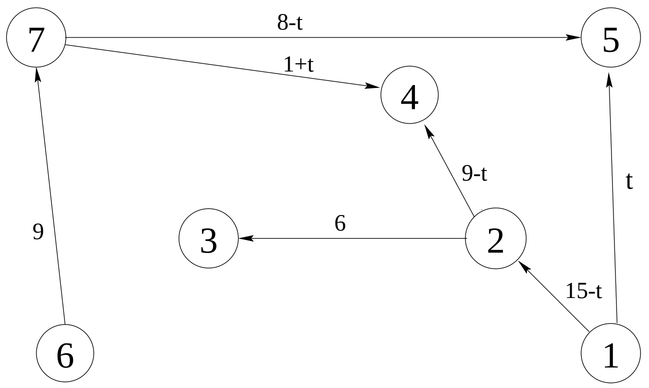


Abbildung 1.48:

Über jede Kante muß positiver Fluß fließen und daher geben $x_{7,5} = 8 - t \geq 0$, $x_{2,4} = 9 - t \geq 0$ und $x_{1,2} = 15 - t \geq 0$ obere Schranken für t an. Die kleinste obere Schranke ist $t = 8$. Damit wir haben wir auch die austretende Variable $x_{7,5}$ gefunden. Der neue Graph hat dann die Form wie in Abbildung 1.49.

Man muß jetzt wieder für jeden Knoten faire Preise berechnen. Die einzige Änderung ergibt sich beim Knoten 5: $y_5 = 62 - 3 = 59$, die restlichen y_i bleiben gleich.

Im nächsten Schritt ist $[2, 7\rangle$ die eintretende Kante (siehe Abbildung 1.50), denn

$$y_7 - y_2 - c_{27} = 33 - 23 - 8 = 2 \geq 0.$$

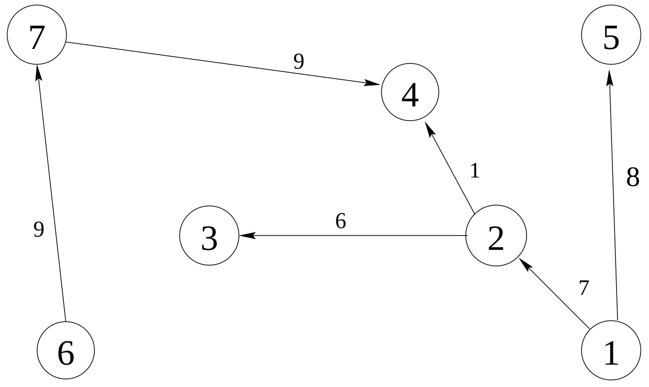


Abbildung 1.49:

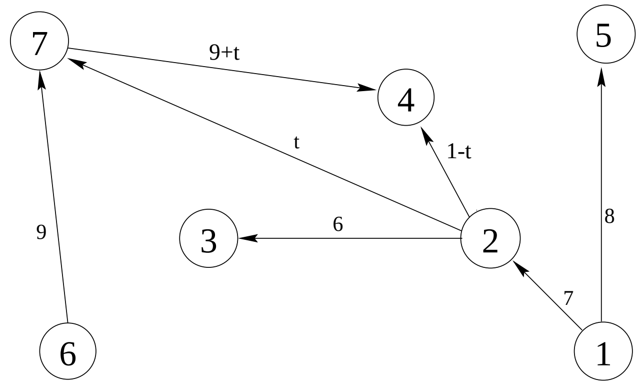


Abbildung 1.50:

Die Kante $[2,4\rangle$ stellt die größte Einschränkung für t dar, also $t = 1$ und $x_{2,4}$ tritt aus der Basis aus, sie Abbildung 1.51.

Bei der Berechnung der neuen Preise y_i verändern sich $y_7 = 33 - 2 = 31$, $y_4 = 51 - 2 = 49$ und $y_6 = -11 - 2 = -13$.

Jetzt gilt für alle Kanten $[i,j\rangle \in E$, daß $y_j - y_i - c_{i,j} \leq 0$ ist, d. h. die Lösung ist optimal.

Wir müssen noch zeigen, daß dieser dritte Schritt des Algorithmus wieder eine zulässige Lösung mit einem aufspannenden Baum liefert.

Der Vektor $\bar{x}$ ist eine zulässige Lösung, denn es gilt $A\bar{x} = Ax$, da sich in jedem Knoten $\pm t$ herausheben. Daraus folgt $c^T\bar{x} = c^Tx + \bar{c}^T\bar{x}$. $[u,v\rangle$ ist die einzige Kante mit $\bar{c}_{ij} \neq 0$ und $\bar{x}_{ij} \neq 0$, der letzte Term hat also die Form:

$$\bar{c}^T\bar{x} = \bar{c}_{uv}^T\bar{x}_{uv} + \bar{c}_{uv}t.$$

Daraus folgt

$$c^T\bar{x} = c^Tx + \bar{c}_{uv}t.$$

Wir wollen t so, daß $\bar{x}$ zulässig ist, also $\bar{x} \geq 0$, und $c\bar{x}$ minimal ist. Nach Wahl von $[u,v\rangle$ gilt $\bar{c}_{uv} < 0$. Je größer t gewählt wird, desto besser wird die Zielfunktion. Wir dürfen t aber nur soweit vergrößern, daß $\bar{x} \geq 0$ noch gilt. Finde dazu eine Vorwärtskante f mit $x_f \leq x_{ij}$ für alle Rückwärtskanten $[i,j\rangle$. Setze $t := x_f$. Nun gilt $\bar{x}_f = 0$, $\bar{x}_{ij} = 0$ für alle Kanten $[i,j\rangle \in B \cup [u,v\rangle$. Die neue Basis ist $\hat{B} = B \cup [u,v\rangle \setminus f$. Nach Konstruktion ist $\hat{B}$ wieder ein aufspannender Baum und die Lösung ist zulässig.

Bei diesem Algorithmus kann es auch wieder zu Zyklen kommen. Mit der Regel von Bland lassen sich diese vermeiden. Dann terminiert der Algorithmus, da es nur eine endliche Menge von aufspannenden Bäumen gibt und jeder maximal einmal betrachtet wird.

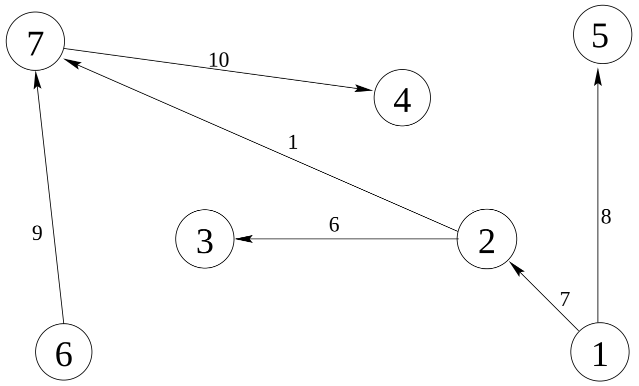


Abbildung 1.51:

Bevor wir uns überlegen, wie man zu einer zulässigen Anfangslösung kommen kann, vergewissern wir uns anhand des Beispiels, daß der Algorithmus in einem Simplexschritt das gleiche tut wie der Revidierte Simplexalgorithmus.
Wir haben die Knoten-Kanten-Inzidenzmatrix A, den Zielvektor c und den Nachfragevektor b gegeben und kennen zu dem Linearen Programm $\max c^T x$, $Ax = b$, $x \geq 0$ eine zulässige Lösung x_B, aufgefüllt mit Nullen, mit dazugehöriger Basis B. In dem Beispiel sind das:

$$\begin{aligned} B &= \{[1,2\rangle, [2,3\rangle, [2,4\rangle, [6,7\rangle, [7,4\rangle, [7,5\rangle\} \\ c_B^T &= \{23, 60, 28, 44, 18, 29\} \\ x_B &= (15, 6, 9, 9, 1, 8) \end{aligned}$$

Die auf die Basis B eingeschränkte Matrix A hat die Form:

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

Durch Streichen der ersten Zeile und Inversion erhält man:

$$\hat{A}_B^{-1} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 & -1 & -1 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}.$$

Jetzt werden folgende Schritte ausgeführt:

- Schritt 0 und 1 haben wir schon beim Netzwerksimplex ausgeführt: $j^* = [1,5\rangle$
- Schritt 2:

$$\begin{aligned} \bar{\bar{A}}_{[1,5\rangle} &= -\hat{A}_B^{-1} \cdot (0,0,0,1,0,0)^T \\ &= (\underbrace{-1}_{[1,2\rangle}, 0, \underbrace{-1}_{[2,4\rangle}, 0, 1, \underbrace{-1}_{[7,5\rangle}) \end{aligned}$$

- Schritt 3:

$$\begin{aligned} \bar{\bar{b}} &= \hat{A}_B^{-1} \hat{b} \\ &= \hat{A}_B^{-1} \cdots (0, 6, 10, 8, -9, 0) \\ &= (15, 6, 9, 9, 1, 8) \\ &= x_B \end{aligned}$$

Die Kante $[7,5\rangle$ mit $\bar{\hat{b}}_i = 8$ liefert den minimalen Wert.

- Schritt 4: $\hat{B} = B \cup \{[1,5\rangle\} - \{[7,5\rangle\}$ wird neue Basis.

Hier erkennt man, daß der Netzwerksimplex dasselbe tut wie der Revidierte Simplex.

Bleibt nur noch Frage, wie man zu einer Anfangslösung kommt.

1.8.3 Wie bekommt man eine Anfangslösung?

Betrachte den aufspannenden Baum in Abbildung 1.52:

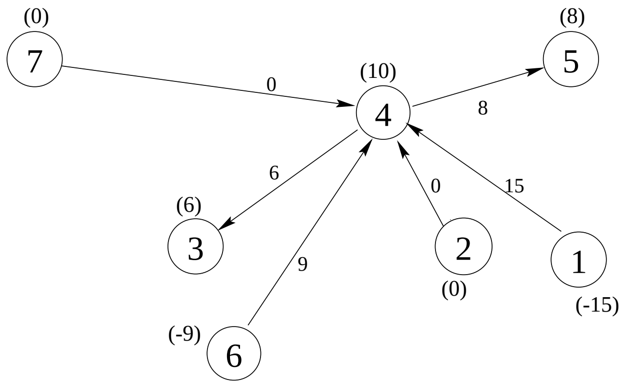


Abbildung 1.52:

Bei diesem aufspannenden Baum B_w, bei dem es einen Knoten w (hier $w = 4$) gibt, von dem alle Kanten ausgehen, bzw. in dem alle Kanten enden, ist es einfach, eine zulässige Lösung anzugeben: Über alle Kanten $[i,w\rangle$ wird $-b_i$ transportiert, über alle Kanten $[w,i\rangle$ wird b_i transportiert.

Es kann leider passieren, daß nicht alle Kanten aus B_w vorhanden sind. Füge sie hinzu, aber verbiete ihren Gebrauch: Wähle einen Knoten w beliebig aus, und füge die fehlende Kanten zum Graphen hinzu. Dabei nennen wir Kanten, die es im Ursprungsgraphen nicht gibt *künstliche Kanten*.
Belege die künstlichen Kanten mit einer "Strafe":

$$p_{ij} = \begin{cases} 1 \,\forall & \text{künstliche Kanten } [i,j\rangle \notin E \\ 0 \,\forall & \text{Kanten } [i,j\rangle \in E \end{cases}$$

Minimiere nun $\sum p_{ij}x_{ij}$ mit Hilfe des Netzwerk-Simplex-Algorithmus. Wir können den Algorithmus sofort anwenden, da wir für dieses Problem, wie oben gezeigt, eine Anfangslösung kennen.

Sei x^* die optimale Lösung für das Problem mit zugehörigem Baum B. Es gibt drei Möglichkeiten:

1. B enthält eine künstliche Kante $[u,v\rangle$ mit $x^*_{u,v} > 0$. Dann hat das ursprüngliche Problem keine Lösung.

2. B enthält keine künstlichen Kanten. Dann ist x^* mit aufspannendem Baum B eine zulässige Anfangslösung für das ursprüngliche Problem.

3. B enthält mindestens eine künstliche Kante, aber es gilt $x^*_{ij} = 0$ für alle künstlichen Kanten. Dann existiert eine Lösung für das ursprüngliche Problem (x^* beschränkt auf die originalen Kanten des Graphen ist eine zulässige Lösung), die allerdings keinen aufspanndenden Baum darstellt. Wir zeigen, daß man in diesem Fall das ursprüngliche Problem in disjunkte Teilprobleme zerlegen und diese einzeln mit dem Netzwerk-Simplex lösen kann.

   Seien $y_1, y_2, \ldots, y_n$ die duale Variablen zur optimalen Lösung x^*. Wähle irgendeine künstliche Kante $[u,v\rangle \in B$ und teile die Menge der Knoten in zwei disjunkte Teilmengen R und S auf:

$$\begin{aligned} y_k \le y_n &\iff k \in R, \\ y_k > y_n &\iff k \in S. \end{aligned}$$

Da $u \in R$ ist, ist $R \neq \emptyset$, und wegen $y_v = y_u + p_{uv} = y_u + 1$ ist $v \in S$ und es gilt daher auch $S \neq \emptyset$. Weiterhin gilt, da die Lösung X^* optimal ist, $y_i + p_{ij} \geq y_j$. Daraus folgt, daß es keine Kante $[i,j\rangle \in E$ gibt mit $i \in R$ und $j \in S$, denn sonst wäre $y_i + p_{ij} = y_i \leq y_u < y_j$. Es existiert also keine Kante, die vom Teilgraphen R in den Teilgraphen S führt.

Wir können zeigen, daß

$$\sum_{k \in S} b_k = \sum_{i \in S, j \notin S} x^*_{ij} - \sum_{i \notin S, j \in S} x^*_{ij} = \sum_{[i,j\rangle \in E, i \in S, j \in R} x^*_{ij} = 0$$

ist, denn keine der Kanten $[i,j\rangle \in E$ mit $i \in S$, $j \in R$ kann in B sein, sonst würde gelten:

$$y_j = y_i + p_{ij} = y_i \qquad \text{und} \qquad y_i > y_u \geq y_j$$

was nicht sein kann.

S ist daher ein eigener abgeschlossener Markt, der weder Exporte noch Importe hat. Also sind S und R abgeschlossene Systeme, auf die man jeweils den Netzwerk-Simplexalgorithmus anwenden kann.

Vorlesung vom 13.6.1996
Carsten Gutwenger, Florian La Roche, Armin Juhlke

Satz 5 *Ist der Vektor b (also die Beschriftung der Knoten) ganzzahlig, dann gelten folgende Aussagen: Existiert eine zulässige Lösung, dann existiert auch eine ganzzahlige zulässige Lösung.*
Existiert eine optimale Lösung, dann existiert auch eine ganzzahlige optimale Lösung.

Satz 6 *(Satz von König)*
Gegeben seien n Buben, von denen jeder genau k Mädchen kennt, und n Mädchen, von denen jedes genau k Buben kennt. Dann existiert eine eineindeutige Zuordnung zwischen Mädchen und Buben, die einander kennen. (Jeder reguläre bipartite Graph hat ein perfektes Matching)

Beweis Betrachte den bipartiten Graphen 1.53. Jeder obere Knoten repräsentiert einen Buben und bekommt das Angebot -1. Jeder untere Knoten repräsentiert ein Mädchen und bekommt die Nachfrage $+1$. Von jedem Jungen gehen genau k gerichtete Kanten zu den Mädchen, die er kennt.

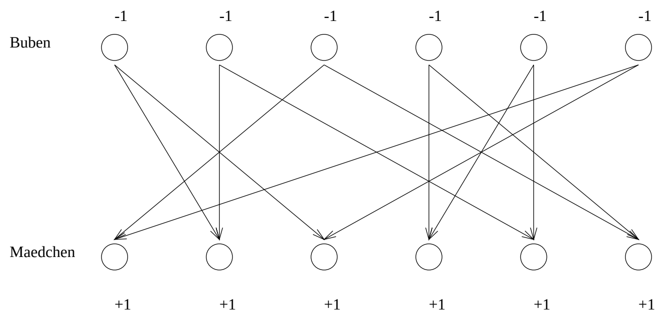


Abbildung 1.53: Buben und Mädchen Zuordnung

Für das Transportproblem ist $x_e = \frac{1}{k}$ für alle Kanten $e \in E$ sicher eine zulässige Lösung. Aus ihrer Existenz folgt sofort mit dem vorigen Satz die Existenz einer ganzzahligen Lösung $\bar{x}$ für die Kantenflüsse. Da $\bar{x}_e$ für jede Kante e zwischen 0 und 1 liegt, gilt $\bar{x}_e = 0$ oder $\bar{x}_e = 1$. Die Menge $\{e \ : \ \bar{x}_e = 1\}$ bildet die gewünschte eineindeutige Zuordnung. $\square$

Definition 17 *Eine $n \times n$–Matrix D mit positiven Einträgen heißt* doppelt stochastisch*, wenn sowohl alle Zeilen, als auch alle Spalten zu 1 aufsummieren:*

$$\begin{aligned} \sum_{i=1}^{n} d_{i,j} &= 1 \qquad \textit{für alle Spalten } j = 1, \dots, n \\ \sum_{j=1}^{n} d_{i,j} &= 1 \qquad \textit{für alle Zeilen } i = 1, \dots, n. \end{aligned}$$

Doppelt stochastische Matrizen mit ganzzahligen Einträgen heißen Permutationsmatrizen.

Satz 7 *(Hilfssatz)*
Für jede doppelt stochastische Matrix $M = (m_{i,j})$ gibt es eine Permutationsmatrix $P = (p_{i,j})$, so daß aus $m_{i,j} = 0$ folgt, daß auch $p_{i,j} = 0$ ist.

Beweis Wir schreiben das Problem wieder als Transportproblem auf: siehe Abbildung 1.54.

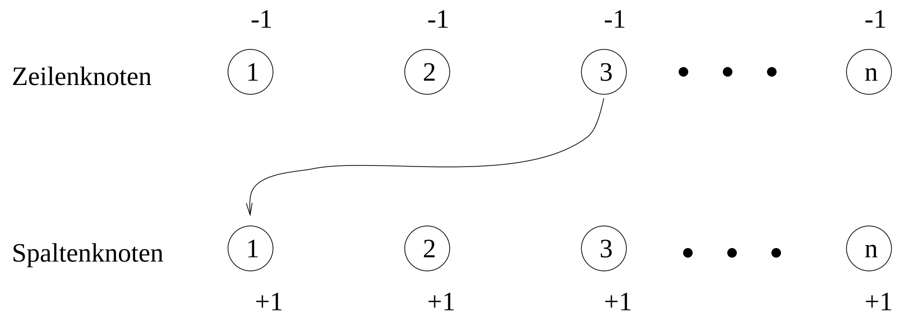


Abbildung 1.54: Zeilenknoten und Spaltenknoten bilden einen bipartiten Graphen

In dem Graphen existiert genau dann eine Kante von i nach j, wenn $m_{i,j} \geq 0$ ist. Setzt man die Kantenflüsse auf $x_{i,j} = m_{i,j}$ für alle i und j, so ist das eine zulässige Lösung. Dann muss es aber auch wieder eine ganzzahlige zulässige Lösung geben. Diese muß lauter Nullen und Einsen enthalten, d.h. wir haben eine Permutationsmatrix.
□

Definition 18 *Eine Kombination von Matrizen M_i gemäß*

$$\sum_{i=1}^{n} \lambda_i M_i \quad \textit{für} \quad 0 \leq \lambda_i \leq 1, \quad \sum_{i=1}^{n} \lambda_i = 1$$

heißt konvexe Kombination *von Matrizen.*

Satz 8 *(Birkhoff – von Neumann)*
Jede doppelt stochastische Matrix M ist darstellbar als konvexe Kombination von Permutationsmatritzen.

Beweis Zum Beweis geben wir einen Algorithmus an, der für eine doppelt stochastischen $n \times n$-Matrix M die konvexe Kombination von Permutationsmatrizen berechnet.

1) $k = 1; \quad \lambda_1 = 1; \quad X = M$
2) Falls X Permutationsmatrix ist, setze $P_k = X$. Fertig!
3) Solange X keine Permutationsmatrix ist
4) Verwende Hilfssatz 7, um eine Permutationsmatrix P^* für X zu finden.
5) $c = \min \left\{ x_{i,j} \mid p^*_{i,j} = 1 \right\}$
6) $\lambda = \lambda_k$
7) $\lambda_k = c\lambda$
8) $\lambda_{k+1} = (1-c)\lambda$
9) $P_k = P^*$
10) $X = \frac{1}{1-c}(X - cP^*)$
11) $k = k + 1$

Während der Schleifendurchläufe wird folgende Invariante immer aufrecht erhalten:

1. $M = \sum_{i=1}^{k-1} \lambda_1 P_i + \lambda_k X$,
2. $\sum \lambda_i = 1$, $\lambda_i \geq 0$,
3. P_i ist Permutationsmatrix,
4. X ist doppelt stochastisch.

Mit jeder Iteration wird ein zusätzlicher Eintrag von X zu 0. Deshalb brauchen wir weniger als n^2 Iterationen.
□

1.8.4 Matchings und Coverings

Definition 19 *Sei $G = (V, E)$ ein ungerichteter Graph.*

1. *Eine Knotenmenge $C \subset V$ heißt genau dann* Vertex–Cover, *wenn jede Kante $e \in E$ inzident zu mindestens einem Knoten in C ist.*

2. *Eine Kantenmenge $M \in E$ heißt genau dann* Matching, *wenn jeder Knoten $v \in V$ inzident zu höchstens einer Kante in M ist.*

Abbildung 1.55 zeigt ein Beispiel für einen Graphen mit eingezeichnetem Matching und Covering.

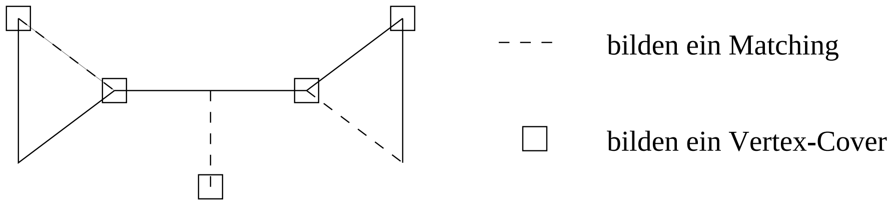


Abbildung 1.55: Beispiel für Matching und Covering

Behauptung 4 *Für alle Vertex–Cover C und alle Matchings M gilt: $|C| \geq |M|$.*

Beweis Sei M ein Matching und C ein Vertex–Cover. Für eine Kante $e = [i, j\rangle \in M$ liegt mindestens einer der Endpunkte i oder j in C. Da alle Kantenendpunkte in M verschieden sind, "braucht" jede Kante in M mindestens einen Knoten in C. Daraus folgt die Behauptung. □

Vorlesung vom 17.06.1996
Ludger Bünger, Holger Sabo

1.8.5 Bipartite Graphen

Definition 20 *Ein Graph $G = (V, E)$ heißt* bipartit, *wenn man die Menge V der Knoten in zwei nichtleere, disjunkte Teilmengen L und R unterteilen kann, so daß die Kanten nur zwischen Knoten aus L und R verlaufen.*

Für diese Klasse von Graphen wollen wir nun untersuchen, welcher Zusammenhang zwischen Machtings und Vertex–Covers besteht.

Satz 9 (von König - Egervary) *Ist $G = (L, R, E)$ ein bipartiter Graph, so gilt für ein maximales Matching M^* und ein minimales Vertex–Cover C^*: $|M^*| = |C^*|$.*

Behauptung 5 *Ein optimales Matching M^* und ein Cover C^* können durch Lösen eines Transportproblems ermittelt werden.*

Ein gegebener bipartiter Graph $G = (L, R, E)$ kann wie folgt in ein geeignetes Transportproblem G' transformiert werden:

1. Füge zwei neue Knoten v und w zu G hinzu.

2. Wandle alle (bisher ungerichteten) Kanten aus E in von L nach R gerichtete Kanten um.

3. Füge für jeden Knoten l_i aus L eine Kante $[l_i, v\rangle$ zu G hinzu.

4. Füge für jeden Knoten r_j aus R eine Kante $[w, r_j\rangle$ zu G hinzu.

5. Füge die Kante $[w, v\rangle$ zu G hinzu.

6. Weise den Knoten folgende Kapazitäten zu:

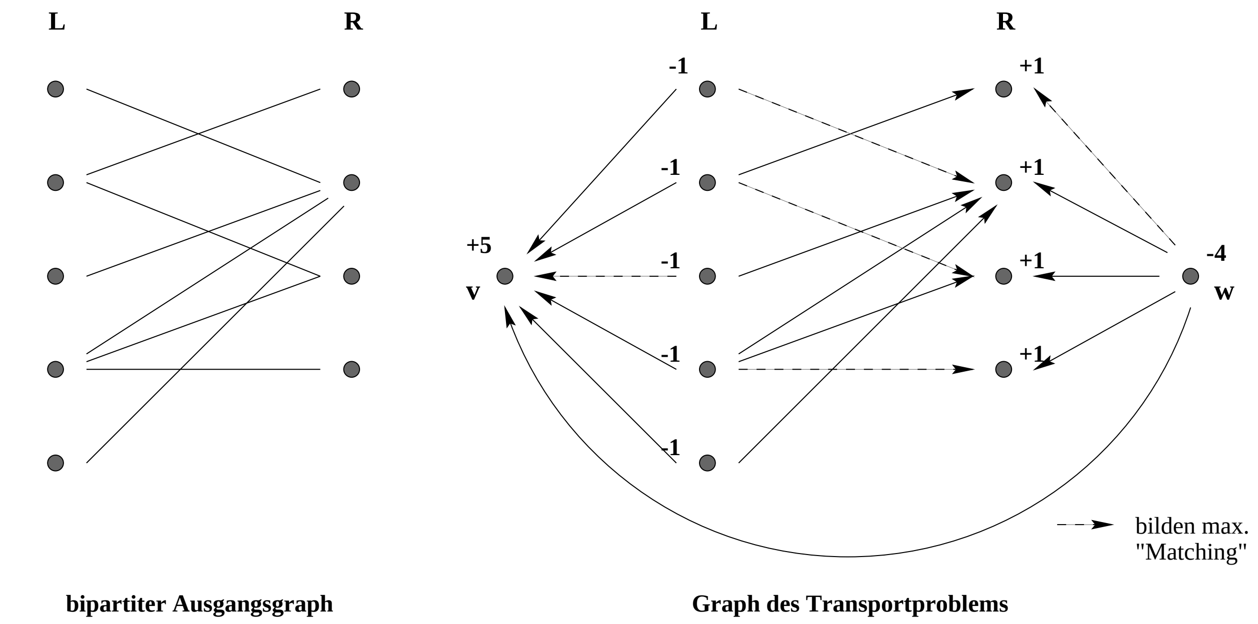


Abbildung 1.56: Beispiel für eine Transformation nach den Regeln

$$\begin{array}{ccl} l \in L & : & b_l = -1 \\ r \in R & : & b_r = +1 \\ v & : & b_v = |L| \\ w & : & b_w = -|R|, \end{array}$$

so daß damit $\sum_i b_i = 0$ erfüllt ist.

Wir können dies folgendermaßen veranschaulichen: Jeder Knoten aus L besitzt eine Orange, und bei jedem Knoten aus R herrscht eine Nachfrage nach einer Orange. Die Knoten in L "müssen sich entscheiden", ob sie ihr Angebot an Knoten v weitergeben, oder an einen Knoten aus R. Demenstprechend müssen die Knoten aus R die Orangen entweder aus L oder von w beziehen. Dies gilt auch für v: alle Orangen, die nicht direkt von L zu v transportiert werden, muß w an v schicken. Damit werden über die Kante $[w,v\rangle$ ebensoviele Orangen versendent, wie zwischen L und R; auf jeder anderen Kante wird höchstens eine Orange transportiert. Außerdem wird von jedem Knoten aus L nur eine Orange wegtransportiert, und jeder Knoten aus R erhält auch nur eine Orange.

Wir wollen uns nun anschauen, wie wir aus einer Lösung dieses Transportproblems ein Matching für den ursprünglichen Graphen konstruieren können.

Bemerkungen:

1. Da es sich um ein Transportproblem mit Kapazitäten aus $\mathbb{Z}$ handelt, ist sichergestellt, daß wir uns auf ganzzahlige Lösungen beschränken können. Insbesondere gilt

   $\begin{array}{lcl} \text{für alle } l \in L & : & b_l = -1 \\ \text{für alle } r \in R & : & b_r = +1 \end{array}$, so daß alle Flüsse x_e aus $\{0,1\}$ sind ($e \neq [w,v\rangle$).

2. Es ergibt sich: An jedem Knoten aus $L \cup R$ kann höchstens eine Kante e mit $x_e = 1$ anliegen. Dies bedeutet, daß alle Kanten e von L nach R mit $x_e = 1$ ein Matching M bilden (vgl. Definition Matching). Somit gilt

   $$|M| = \sum_{\substack{e \text{ von } L \\ \text{nach } R}} x_e.$$

   Wie man sich leicht klar machen kann, gilt die Umkehrung, d.h. *„Für jedes Matching M gibt es eine entsprechende zulässige $\{0,1\}$-Belegung der Kantenflüsse x_e"*, ebenfalls.

3. Da über die Kante $[w,v\rangle$ ebensoviel transportiert werden muß wie von L nach R, können wir feststellen, daß

   $$x_{[w,v\rangle} = \sum_{\substack{e \text{ von } L \\ \text{nach } R}} x_e = |M|$$

gilt. Damit können wir ein maximales Matching finden, indem wir die Kante $x_{[w,v\rangle}$ mit $c_{[w,v\rangle} = -1$, alle anderen Kanten e mit $c_e = 0$ gewichten und das resultierende Transportproblem lösen.

Wir haben nun gesehen, daß wir ein maximales Matching durch Lösen eines Transportproblems bestimmen können. Wir wollen aber auch ein minimales Vertex–Cover berechnen und zeigen, daß dieses ebenso groß ist wie ein maximales Matching. Diese Aufgabenstellung sieht dem Lösen dualer Linearer Programme sehr ähnlich. Daher wollen wir hier auch diese untersuchen.
Sind x^* und y^* optimale primale bzw. duale Lösungen des Transportproblems, so gilt wegen der Dualität folgende Gleichheit:

$$\sum_{Knoten\ i} b_i \cdot y_i^* = \sum_{Kanten\ e} c_e \cdot x_e^* = -|M^*|.$$

Die Werte der y_i^* müssen wir noch berechnen. Um diese zu finden, setzen wir (o.B.d.A.) $y_v^* = 0$. Aus der Annahme, daß Matchingkanten existieren und $x^*_{[w,v\rangle} = |M^*| > 0$ gilt, wissen wir, daß $[w,v\rangle$ im „Basisbaum" T enthalten ist. Damit gilt wegen der Dualität:

$$\underbrace{y_v^*}_{0} = y_w^* + \underbrace{c_{[w,v\rangle}}_{-1} \quad \text{also} \quad y_w^* = 1.$$

Weiterhin gilt für alle anderen Baumkanten e: $c_e = 0$, so daß sich für alle Knoten i y_i^* zu 0 bzw. 1 ergibt, je nachdem, ob i zuerst von v oder w erreichbar ist.

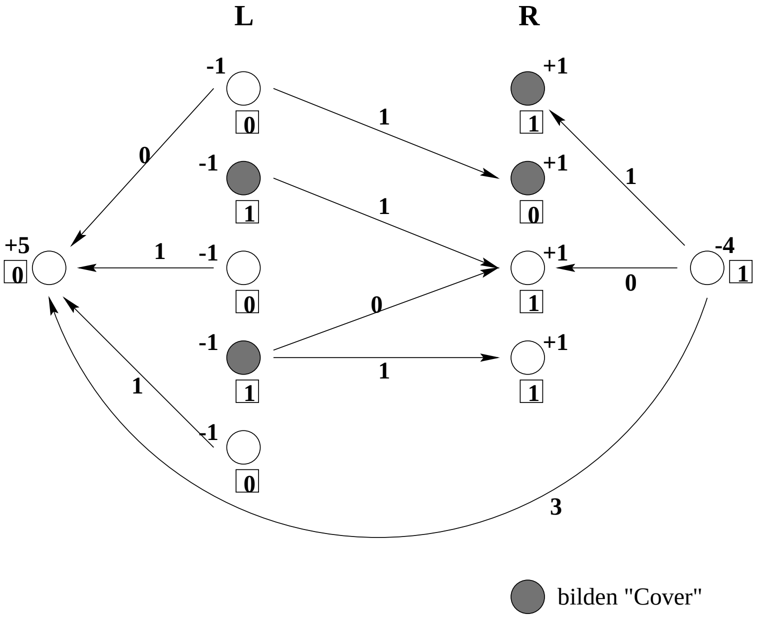


Abbildung 1.57: Optimale Lösung des Transportproblems

Mit Hilfe der dualen Lösung wollen wir nun eine Vertex–Cover bestimmen.

Behauptung 6 *Sei $C_L = \{l \in L : y_l^* = 1\}$ und $C_R = \{r \in R : y_r^* = 0\}$. Dann ist $C = C_L \cup C_R$ ein Vertex-Cover.*

Beweis Wir nehmen an, daß $[l,r\rangle \in L$ nicht von C überdeckt wird, d.h. C ist kein Vertex-Cover.
Dann gilt nach Konstruktion $y_l^* = 0$ und $y_r^* = 1$. Dies steht aber im Widerspruch zu

$$y_r^* \leq y_l^* + \underbrace{c_{[l,r\rangle}}_{=0}.$$

Somit wurde gezeigt, daß C ein Vertex-Cover ist. □

Wir müssen jetzt noch zeigen, daß das so konstruierte Cover tatsächlich ebenso groß ist wie M^*; d.h.,$|M^*| = |C|$.

Beweis *von Satz 9 (Satz von König-Egervary)*

$$\begin{aligned}
|M^*| &= -\sum b_i \cdot y_i^* \\
&= \underbrace{-b_v \cdot \overbrace{y_v^*}^{=0}}_{=0} - \sum_{l \in L} b_l \cdot y_l^* - \sum_{r \in R} b_r \cdot y_r^* - b_w \cdot y_w^* \\
&\stackrel{\substack{b_l=-1 \\ b_r=1}}{=} \underbrace{+\sum_{l \in L} y_l^*}_{=|C_L|} - \sum_{r \in R} y_r^* \underbrace{-b_w \cdot y_w^*}_{=|R|} \\
&= |C_L| + |C_R| \quad = \quad |C|.
\end{aligned}$$

Dabei wurde im letzten Schritt verwendet, daß $\sum_{r \in R} y_r^* + |C_R| = |R|$ gilt. (Die darin erscheinende Summe entspricht genau der Anzahl der Elemente aus R, die *nicht* in C_R liegen.)

□

Definition 21 *Sei* $G = (L, R, E)$ *ein bipartiter Graph und* $\ell \in L$. *Die Menge* $N(\ell) = \{r \in R | r \text{ ist adjazent zu } \ell\}$ *heißt* Nachbarschaft *von* ℓ.

Satz 10 (von Hall) *G hat* kein *Matching der Größe* $|L|$, *d.h. kein perfektes Matching, genau dann, wenn es eine Teilmenge I von L gibt, für die gilt, daß die „Gesamtnachbarschaft“ von I kleiner ist als I selbst; also wenn*

$$\exists I \subseteq L \ : \ \left| \bigcup \{N(\ell) | \ell \in I\} \right| \ < \ |I|.$$

Beweis
„ $\Leftarrow$ “ : Wenn die Nachbarschaft einer Knotenmenge kleiner als die Menge selbst ist, kann kein perfektes Matching gefunden werden.
„ $\Rightarrow$ “ : Ist M ein maximales Matching mit $|M| < |L|$ und C ein zugehöriges minimales Vertex-Cover mit $C = C_L \cup C_R$, so gilt nach Satz 9 $|C| = |M|$.
Sei $I = L \setminus C_L$, also $|I| = |L| - |C_L|$. Damit müssen alle Nachbarn von Knoten in I in C_R liegen ($N(I) \subseteq C_R$), denn da sie nach Konstruktion nicht von links überdeckt werden, müssen sie von rechts überdeckt werden. Hieraus schliessen wir

$$|N(I)| \ \leq \ |C_R| \ = \ \underbrace{|C|}_{=|M|<|L|} \ - |C_L| \ < \ |L| - |C_L| \ = \ |I|.$$

□

1.8.6 Optimale Zuweisungsprobleme

Unter einem „optimalen Zuweisungsproblem“ versteht man die Fragestellung nach einem perfekten Matching für einen bipartiten Graphen $G = (L, R, E)$, das „besonders gut“ ist. Dabei geht man davon aus, daß G folgende Eigenschaften besitzt:

- Die beiden Knotenmengen L und R sind gleich groß, $n := |L| = |R|$.
- Alle möglichen Verbindungskanten zwischen Knoten aus L und R existieren, $E = L \times R$.
- Alle Kantengewichte s_e sind ganzzahlig, $s_e \in \mathbb{Z}$.

Man unterscheidet zwischen zwei Grundformen von optimalen Zuweisungsproblemen:

1. **Das (allgemeine) optimale Zuweisungsproblem:**
   Gesucht wird ein perfektes Matching M, so daß die Summe der Kantengewichte von M $\left(\sum_{e \in M} s_e\right)$ je nach Problemstellung minimal bzw. maximal ist.

2. **Das Flaschenhals-Zuweisungsproblem:**

   Gesucht wird ein perfektes Matching M, so daß das kleinste in M vorkommende Kantengewicht möglichst groß ist. ($min\{s_e | e \in M\}$ maximal)

Zur Lösung des *Flaschenhals-Zuweisungsproblems* können wir folgendes bemerken:

Beobachtung Ist G_t ein Teilgraph von G, indem die Kantengewichte s_e aller Kanten e die Schranke t nicht unterschreiten ($s_e \geq t, \forall e \in E$), so hat G genau dann ein perfektes Matching M, mit $min\{s_e | e \in M\} \geq t$, wenn G_t ein perfektes Matching hat.

Dies führt sofort zu einer Idee für einen Algorithmus:
Suche (z.B. mit binärer Suche) das größte t, für das es noch ein Matching gibt. Das dabei gefundene Matching ist notwendigerweise eine Lösung für das Flaschenhals-Zuweisungsproblem.

Für das *optimale Zuweisungsproblem* gibt es zwei grundlegende Lösungsansätze:

Die naive Vorgehensweise:
Das Matchingproblem kann, wie schon gesehen, durch ein geeignetes Transportproblem modelliert werden, in welchem die Summe der Kantengewichte durchflossener Kanten minimiert (bzw. maximiert) wird.

Wir konstruieren also wieder einen Graphen G' aus G nach den folgenden Regel:

- Richte die Kanten von links nach rechts.
- Versieh alle Knoten aus L mit einem Angebot von eins ($b_l = -1, \forall l \in L$).
- Versieh alle Knoten aus R mit einer Nachfrage von eins ($b_r = 1, \forall r \in R$).
- Beschränke den Durchluß durch alle Knoten auf eins ($x \geq 1$):

$$\begin{aligned} \forall i \in L \quad &: \quad \sum_{j \in R} x_{[i,j\rangle} = -1 \\ \forall j \in R \quad &: \quad \sum_{i \in L} x_{[i,j\rangle} = 1 \end{aligned}$$

- Minimiere bzw. maximiere die Summe der Kantengewichte durchflossener Kanten:

$$\sum_{\substack{i \in L \\ j \in R}} s_{ij} \cdot x_{[i,j\rangle}.$$

Die „Ungarische Methode“:
Im folgenden suchen wir ein minimales Matching und stellen die Kantengewichte s_{ij} in einer Matrix dar. Dabei entsprechen die Zeilen genau den Knoten aus L, die Spalten der Matrix den Knoten aus R.
Intuition: Da jede Permutationsmatrix einem Matching entspricht, greift man aus jeder Zeile und jeder Spalte genau ein Element heraus, so daß deren Summe minimal ist.

Beobachtung Addiert man eine Konstante γ zu einer Zeile, bzw. Spalte, so ändert sich das optimale Matching nicht.

Algorithmus “Die Ungarische Methode”

1. Subtrahiere von allen Zeilen i das Minimum $c_i = min\{s_{ij} | j \in R\}$, wodurch man in jeder Zeile mindestens eine 0 erhält.
2. Subtrahiere analog von allen Spalten j das Minimum $c_j = min\{s_{ij} | i \in L\}$.
3. Bilde den bipartiten Graphen der 0-Einträge der erzeugten Matrix. Dies ist der bipartite Graph, der genau dann eine Kante $[i, j]$ enthällt, wenn der Eintag s_{ij} der Matrix 0 ist.

4. Suche in diesem Graphen ein maximales Matching M, sowie das zugehörige minimale Vertex-Cover $C = C_L \cup C_R$.

5. Es können folgende Fälle auftreten:

   - $|M| = n$: *FERTIG*, das minimale Vertex-Cover und das maximale Matching, mit minimalem Gewicht $\sum_{e \in M} s_e$ wurden gefunden.
   - $|M| < n$: Fahre mit Schritt 6 fort.

6. Bilde $\gamma = min\{s_{ij} | i \notin C_L, j \notin C_R\}$.

7. Subtrahiere γ von allen Zeilen $i \notin C_L$.

8. Addiere γ zu allen Spalten $j \in C_R$ und fahre fort mit Schritt 3.

Durch die Schritte 6 bis 8 wird sichergestellt, daß keine der bisher in der Matrix erzeugten Null–Einträge negativ wird. Wir müssen noch zeigen, daß dieser Algorithmus wirklich terminiert.

Beweis

Idee: Man zeigt, daß das Gesamtgewicht $\sum_{\substack{i \in L \\ j \in R}} s_{ij}$ bei jeder Iteration des Algorithmus echt kleiner wird. Da alle Einträge ganzzahlig sind, folgt damit die Behauptung.

In Schritt 7 verringert sich das Gesamtgewicht um $\gamma n(n - |C_L|)$.
In Schritt 8 vergrößert sich das Gesamtgewicht um $\gamma n(|C_R|)$.

Zeige nun, daß sich damit tatsächlich eine Verringerung des Gesamtgewichts ergibt:

$$\gamma n(n - |C_L|) - \gamma n(|C_R|) \;=\; \gamma n(n \underbrace{-|C_L| - |C_R|}_{=-|C|}) \;=\; \gamma n(n - \underbrace{|C|}_{<n}) \geq \gamma n \;>\; 0$$

$\square$

Beispiel zur Anwendung der „Ungarischen Methode“:

$$\begin{pmatrix} 6 & 6 & 7 & 8 & 6 & 8 \\ 6 & 5 & 3 & 6 & 4 & 5 \\ 9 & 8 & 6 & 9 & 7 & 9 \\ 4 & 5 & 5 & 6 & 3 & 4 \\ 5 & 3 & 2 & 6 & 4 & 3 \\ 9 & 8 & 8 & 9 & 6 & 8 \end{pmatrix} \xrightarrow{Schritt\ 1} \begin{pmatrix} 0 & 0 & 1 & 2 & 0 & 2 \\ 3 & 2 & 0 & 3 & 1 & 2 \\ 3 & 2 & 0 & 3 & 1 & 3 \\ 1 & 2 & 2 & 3 & 0 & 1 \\ 3 & 1 & 0 & 4 & 2 & 1 \\ 3 & 2 & 2 & 3 & 0 & 2 \end{pmatrix} \xrightarrow{Schritt\ 2} \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 1 \\ 3 & 2 & 0 & 1 & 1 & 1 \\ 3 & 2 & 0 & 1 & 1 & 2 \\ 1 & 2 & 2 & 1 & 0 & 0 \\ 3 & 1 & 0 & 2 & 2 & 0 \\ 3 & 2 & 2 & 1 & 0 & 1 \end{pmatrix}$$

Jetzt bildet man den bipartiten Graph der 0–Einträge (siehe Abbildung 1.8.6).

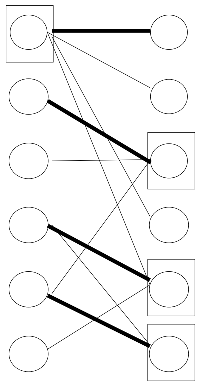


Abbildung 1.58:

Wie man leicht sieht, gilt für das Matching $M : |M| = 4 < n = 6$.
Man bildet nun $\gamma = \min\{s_{ij} | i \notin C_l, j \notin C_r\} = 1$.

$$\overset{Schritt\ 7}{\Longrightarrow} \begin{pmatrix} 6 & 6 & 7 & 8 & 6 & 8 \\ 6 & 0 & 0 & 6 & 0 & 0 \\ 9 & 8 & 6 & 9 & 7 & 9 \\ 0 & 0 & 0 & 6 & 0 & 0 \\ 0 & 0 & 0 & 6 & 0 & 0 \\ 9 & 8 & 8 & 9 & 6 & 8 \end{pmatrix} \overset{Schritt\ 8}{\Longrightarrow} \begin{pmatrix} 6 & 6 & 7 & 8 & 6 & 8 \\ 6 & 5 & 0 & 6 & 0 & 5 \\ 9 & 8 & 6 & 9 & 7 & 9 \\ 0 & 5 & 5 & 6 & 0 & 0 \\ 5 & 0 & 0 & 6 & 0 & 0 \\ 9 & 8 & 8 & 9 & 6 & 8 \end{pmatrix} \overset{usw.}{\Longrightarrow}$$

Fahre so lange fort, bis $|M| = n = 6$ und somit ein perfektes Matching mit $\min_{e \in M}\{s_e\} \geq 5$ erreicht ist (komplettes Beispiel: siehe Musterlösung zu Übung 5, Aufgabe 3).

Vorlesung vom 24.6.1996
Guido Schäfer, Michael Rovatsos

1.9 Ganzzahlige und gemischte lineare Programme

1.9.1 Motivation

Bisher haben wir nur lineare Programme der Form $\min c^T x, \quad Ax \geq b$ betrachtet. Nimmt man die Bedingung hinzu, daß der Lösungsvektor ganzzahlig sein muß, also $x \in \mathbb{Z}^n$, so stellt sich heraus, daß diese kleine Modifikation die Lösung des linearen Programms erheblich erschwert. Da solche **ILP**-Probleme ("**I**nteger **L**inear **P**rogramming") in der Praxis häufig auftreten, sei ihnen der folgende Abschnitt gewidmet. Ähnlich interessant erscheinen sogenannte gemischte lineare Programme ("**M**ixed **ILP**"), bei denen für den Lösungsvektor $x \in \mathbb{Z}^n$ zusätzlich gilt: $x_1, \ldots, x_k \in \mathbb{R}, \quad x_{k+1}, \ldots, x_n \in \mathbf{Z}, \quad 1 \leq k \leq n$.

Beispiel Wenn ein maximales Matching M bei minimalen Kosten in einem ungerichteten Graphen $G = (V, E)$ gesucht ist, so ergibt sich folgende ILP-Formulierung:

$$\max \sum_{e \in E} c_e \cdot x_e, \text{ wobei } x_e = \begin{cases} 0 & falls \quad e \notin M \\ 1 & falls \quad e \in M, \end{cases}$$

wobei für jeden Knoten $i \in V$ gelten muß

$$\sum_{e: i \in e} x_e = 1 \text{ (Matchingbedingung).}$$

Die Bedingung für die x_e kann man auch schreiben als

$$x_e \in \mathbb{Z}, \quad 0 \leq x_e \leq 1.$$

Durch diese Bedingung schränkt man den Lösungsraum auf ganzzahlige Vektoren ein.

Da man Lösungsverfahren für allgemeine lineare Programme kennt, stellt sich nun die Frage, ob es möglich ist, mit Hilfe der (unter Umständen nicht ganzzahligen) Lösung, die z.B. der Simplex-Algorithmus liefert, eine ganzzahlige Lösung anzunähern oder sie exakt zu ermitteln. Man bezeichnet diesen Ansatz häufig als "**Relaxation**". Sei also x^* die optimale Lösung zu einem LP und $z^* = c^T x^*$;

- Falls x^* (zufälligerweise) ganzzahlig ist, ist die Nebenbedingung schon erfüllt.
- Ist das nicht der Fall, so stellen sich folgende Fragen:
  1. z^* ist eine untere Schranke für das wahre Optimum der Zielfunktion. Wie gut ist diese Schranke?
  2. Kann man sich x^* zunutze machen, um das ganzzahlige Optimum zu ermitteln?
  3. Gibt es Ungleichungen, die in $Ax = b$ eingebaut werden können, und durch die die Nebenbedingung $x \in \mathbb{Z}^n$ ersetzt werden kann?

Vor allem ist natürlich Frage (3) interessant, da sie es möglich macht, ILP-Probleme mit Hilfe der bekannten Algorithmen zu lösen. Tatsächlich gibt es bei dem Matchingproblem folgende Möglichkeit, um die Nebenbedingung $x_e \in \mathbb{Z}, \quad 0 \leq x_e \leq 1$ in Ungleichungen zu kodieren: Schreibe für alle Teilmengen $U \subset V$ mit $|U|$ ungerade, $|U| > 1$ folgende Ungleichung:

$$\sum_{e \in E(U)} x_e \leq \left\lfloor \frac{|U|}{2} \right\rfloor .$$

Da dieses Verfahren aber $\approx 2^n - 1$ Ungleichungen fordert von denen - wie sich herausstellt - man keine entbehren kann, ist es für die praktische Realisierung eines Algorithmus so gut wie unbrauchbar.

Beispiel Das TRAVELLING SALESMAN PROBLEM (TSP): Sei $G = (V, E)$ ein ungerichteter, vollständiger Graph, c_e seien die Kantenlängen. Wenn man keinen vollständigen Graphen gegeben hat, so fügt man die fehlenden Kanten ein aber verbietet ihren Gebrauch, indem man die Kosten $c_e = \infty$ setzt. Gesucht ist die Rundreise $(v_{i_0}, v_{i_1}, \ldots, v_{i_{n-1}})$ mit $v_{i_j} \in V$, bei der jeder Knoten aus V genau einmal besucht wird, und bei welcher $\sum_{e_j=\{v_{i_{j-1}}, v_{i_j}\}} c_{e_j}$ (die "Kosten"der Tour) minimal sind. Wir wollen zunächst das Problem als ILP formulieren:

$$\forall e \in E: \quad x_e \in \{0,1\}, \qquad x_e = \begin{cases} 0 & falls \quad e \text{ nicht auf der Tour} \\ 1 & falls \quad e \text{ auf der Tour} \end{cases}$$

$\min \sum_{e \in E} c_e \cdot x_e$ (Kostenminimierung)

$\forall i \in V: \qquad \sum_{e: i \in e} x_e = 2$ (Gradbedingung)

$\forall S \subset V: \qquad \emptyset \neq S \neq V: \qquad \sum_{e: |e \cap S| = 1} x_e \geq 2$ (Untertourbedingung)

Die Gradbedingung sagt, daß jeder Knoten genau einmal betreten und wieder verlassen wird. Die Untertourbedingung stellt sicher, daSS die Tour zusammenhängend ist.

Problem: Sehr viele Ungleichungen!

Im folgenden sollen nun einige Lösungsmöglichkeiten für das TRAVELLING SALESMAN PROBLEM näher betrachtet werden.

1.9.2 Das TRAVELLING SALESMAN PROBLEM – Lösungsansätze

Es gibt - wie bei nahezu jedem Problem dieser Art - zwei grundsätzliche Vorgehensweisen: eine exakte Lösung zu finden oder diese möglichst genau anzunähern, wenn das mit erheblich geringerem Aufwand verbunden ist. Zunächst stellen wir drei Möglichkeiten zur exakten Lösung des TSP vor; im darauffolgenden Abschnitt dann approximative Methoden.

1. **Aufzählung aller Touren:** Die intuitive Vorgehensweise, alle möglichen Touren aufzuzählen und ihre Kosten zu vergleichen, ist praktisch unbrauchbar, denn es gibt $\frac{(n-1)!}{2}$ Touren ($n\mathbf{-1}$, da der Startknoten nicht mitgezählt wird und $\frac{\mathbf{1}}{\mathbf{2}}\cdot$, weil der Graph ungerichtet ist).

2. **Dynamisches Programmieren:**
   Bemerkung: Im folgenden seien die Knoten durch ihre Indizes und Wege als Indexmengen repräsentiert.

   Idee: Für jede Teilmenge $S \subset V \backslash \{1\}$ und jeden Knoten $j \notin S$, $j \neq 1$ definiere $C(j, S)$ als die Länge des kürzesten Weges von 1 nach j, der jeden Graphknoten aus S genau einmal enthält.

   Ziel: Ermittle rekursiv
   $$\min_{j \in V \backslash \{1\}} \{C(j, V \backslash \{1, j\} + c_{\{j,1\}}\}.$$

   Es gilt: $C(j, \emptyset) = c_{\{1,j\}}$ und $C(j, S) = \min_{i \in S} \{C(i, S \backslash \{i\}) + c_{\{i,j\}}\}$. Damit kann man die Rekursion berechnen.

   Auch dieser Algorithmus hat sehr hohe Laufzeit- und Speicherkomplexität:
   Laufzeit (Iterationen): $\sum_{k=0}^{n-1} \binom{n-1}{k} \cdot (n-1-k) \cdot k = (n-1) \cdot (n-2) \cdot 2^{n-3}$
   Speicherkomplexität: $n \cdot \binom{n}{n/2}$

3. **“Branch and Bound”-Methoden:** Auf diese Möglichkeit wird später eingegangen.

Folgerung: Sucht man exakte Lösungen des TSP, so benötigt man exponentiell viel Zeit. Man kann das TSP also nicht effizient lösen. Daher bedient man sich Approximationsalgorithmen, in der Hoffnung, eine möglichst exakte Lösung zu finden.

1.9.3 Approximative Lösungsansätze

Heuristik

- “nearest neighbour”: Beginne mit “Stadt 1”. Gehe jeweils zur nächstgelegenen, noch nicht besuchten Stadt.
- “furthest neighbor”: Beginne mit Tour $1 \longrightarrow 2 \longrightarrow 1$. Erweitere die Rundreise jeweils um eine Stadt. Wähle aus den sich anbietenden Möglichkeiten (“Umleitungen”) diejenige Stadt aus, die die Tour am meisten verlängert, wenn sie in die bereits existierende Tour optimal eingefügt wird.

Beispiel

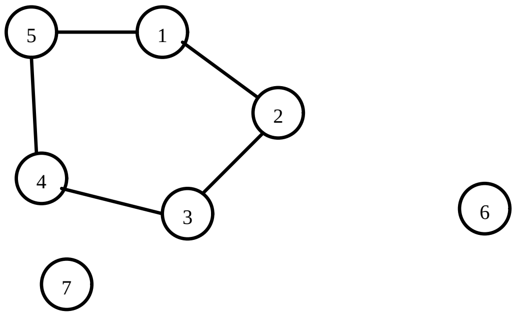


Abbildung 1.59: Tour $t = (1, 2, 3, 4, 5, 1)$

Man hat in Abbildung 1.59 fünf Knoten zur Auswahl, um die vorhandene Tour $t = (1, 2, 3, 4, 5, 1)$ zu erweitern. Eine Erweiterung um den Knoten 6 verlängert die Tour am meisten. Wähle die Kanten zu 6 so aus, daß ‘minimaler’ Aufwand entsteht, d.h. $t' = (1, 2, 6, 3, 4, 5, 1)$, siehe Abbildung 1.60.

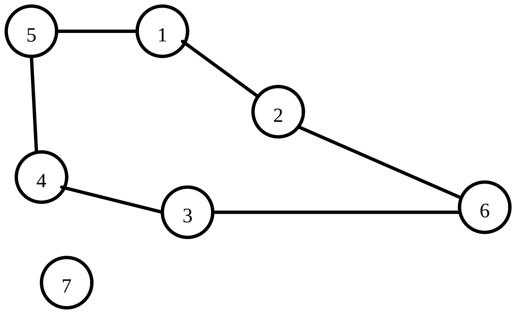


Abbildung 1.60: Tour nach Anwendung des “furthest neighbor”-Prinzips

Lokale Verbesserungen: Entferne zwei Kanten $\{a, b\}, \{c, d\}$ und eliminiere Überkreuzungen, siehe Abbildung 1.61.
Eliminiere so alle vorkommenden Kreuzungen und hoffe, daß das Ergebnis möglichst gut wird, siehe Abbildung 1.62.
Approximationsalgorithmen (mit Garantie): Man versucht, Approximationsverfahren anzugeben, bei dem man Aussagen über die maximale Abweichung von der optimalen Lösung machen kann. Beim TSP-Problem ist es nicht möglich, es sei denn es gilt $\mathcal{P} = \mathcal{NP}$, einen Algorithmus anzugeben, der bis auf eine multiplikative Konstante die optimale Lösung berechnet:

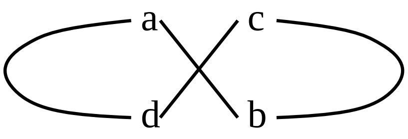


Abbildung 1.61: vorher

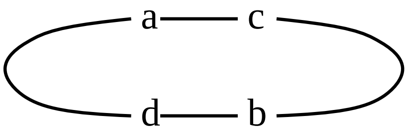


Abbildung 1.62: nachher

Satz 11 *Für jedes $M > 0$ gilt: Wenn es einen polynomiellen Algorithmus gibt, der garantiert eine Lösung des TSP mit Wert $< M \cdot L_{opt}$ (L_{opt} = Länge der optimalen Tour) liefert, dann gilt $\mathcal{P} = \mathcal{NP}$.*

Beweis Mit so einem Algorithmus könnte man das Hamiltonsche Kreisproblem lösen: in einem ungerichteten Graph G möchte man einen Zyklus finden, der jeden Knoten genau einmal enthält.
Sei $G = (V, E)$ ein ungerichteter Graph mit $|V| = n$. Erzeuge den gewichteten, vollständigen Graphen

$$G' = (V, \begin{pmatrix} V \\ 2 \end{pmatrix}) \quad mit \quad c(e) = \begin{cases} 1 & : \ wenn \quad e \in E \\ n \cdot M & : \ wenn \quad e \notin E. \end{cases}$$

Wenn in G ein Hamilton-Zyklus der Länge n existiert, so existiert in G' eine Tour t mit den Kosten $C(t) = \sum_{e \in t} c(e) = n$ (was der Länge der optimalen Tour entspricht); ansonsten hat die Tour $\geq n \cdot M$ Kosten. Je nachdem welches Ergebnis der polynomielle TSP-Approximationsalgorithmus liefert kennen wir also einen Hamilton-Zyklus oder wir wissen, daß keiner existiert. Da wir wissen, daß das Hamilton-Zyklus-Problem $\mathcal{NP}$-vollständig ist, folgt daraus, daß $\mathcal{P} = \mathcal{NP}$ ist.
Sie dazu auch das folgende Beispiel mit den Abbildungen 1.63 und 1.64. □

Beispiel

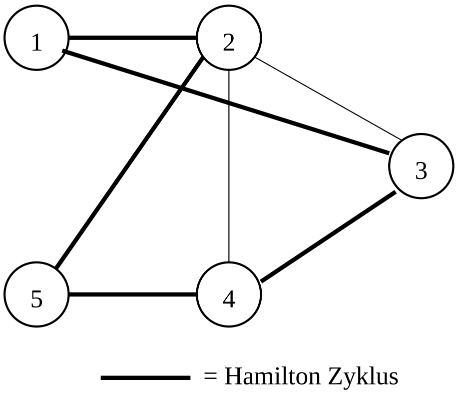


Abbildung 1.63: $G = (V, E)$

Graphen mit Dreiecksungleichung
Beschränkt man sich auf gewichtete Graphen, die der Dreiecksungleichung genügen, d.h. bei denen für alle Knoten $i, j, k \in V$

$$c_{ij} \leq c_{ik} + c_{kj}$$

gilt mit $c_{ii} = 0$, so kann man Approximationsalgorithmen finden, die in polynomieller Zeit gute (angenäherte) Lösungen konstruieren.

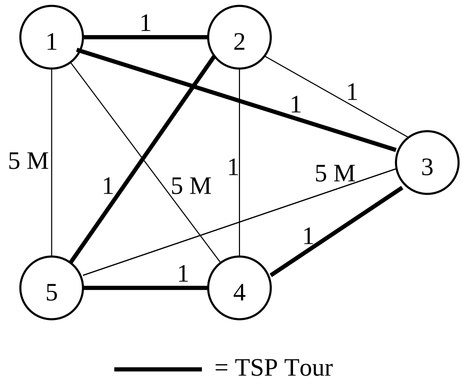


Abbildung 1.64: $G' = (V, E')$

Satz 12 *Wenn ein Graph $G = (V, E)$ der Dreiecksungleichung genügt, so kann man in polynomieller Zeit eine Approximation des TSP finden, die höchstens um einen Faktor 2 schlechter ist.*

Beweis

1. Bilde MST (Minimum Spanning Tree) T von G (Verfahren hierzu siehe Vorlesung "Datenstrukturen und Algorithmen" im WS '95/96)
2. Verdopple alle Kanten in T und erhalte T'
3. Bilde "Euler"-Tour t in T': Eine Euler-Tour eines zusammenhängenden, ungerichteten Graphen G ist ein Zyklus, der jede Kante von G genau einmal besucht, wobei Knoten mehrfach besucht werden dürfen.
4. Finde Abkürzungen über mehrfach besuchte Knoten. Bezeichne diese Tour mit t_{app}.

$\square$

Warum leistet der Algorithmus das Verlangte?

Behauptung 7 $C(t_{app}) = \sum_{e \in t_{app}} c(e) \leq 2 \cdot L_{opt}$, *wobei* L_{opt} *die Länge der optimalen TSP-Tour bezeichnet.*

Beweis Für die Kosten des MST $T = (V, E_T)$ gilt:

$$C(T) = \sum_{e \in E_T} c(e) \leq C(t_{opt}),$$

wobei t_{opt} die optimale TS-Tour bezeichnet. Dies erklärt sich folgendermaßen: Jede Tour in G kann in einen aufspannenden Baum transformiert werden, indem man eine Kante aus der Tour löscht. Für die Euler-Tour t in $T' = (V, E_{T'})$ gilt somit

$$C(t) = C(T') = \sum_{e \in E_{T'}} c(e) = 2 \cdot C(T) \leq 2 \cdot C(t_{opt}).$$

t_{app} ist eine verkürzte Tour von t. Somit gilt:

$$C(t_{app}) \leq C(t) \leq 2 \cdot C(t_{opt}).$$

$\square$

Behauptung 8 *In Schritt 3 muß eine Euler-Tour immer existieren, da in T' alle Knoten geraden Grad haben.*

Behauptung 9 *Ist t eine Euler-Tour in T', so existiert eine Tour t' (mit Kanten aus G), für die gilt: $C(t') \leq C(t)$.*

Beweis Sei $t = (v_0, v_1, ..., v_{m-1}, v_0)$ die Euler-Tour in T'. Falls $m = |V|$, so ist $t = t'$; ansonsten existiert nach dem Schubfachprinzip ein kleinster Index j_0 mit $v_i = v_{j_0}$ für ein $i < j_0$. Betrachte nun die Tour:

$$t' = (v_0, v_1, ..., v_{j_0-1}, v_{j_0+1}, ..., v_{m-1}, v_0).$$

Es gilt:

$$C(t') = C(t) \underbrace{-c_{v_{(j_0-1)}v_{j_0}} - c_{v_{j_0}v_{(j_0+1)}} + c_{v_{(j_0-1)}v_{(j_0+1)}}}_{\substack{\leq 0 \\ (Dreiecksungleichung)}} \leq C(t).$$

□

Wendet man dieses Verfahren wiederholt an, so erhält man t_{app} mit $C(t_{app}) \leq C(t)$.

Beispiel Betrachte die Abbildungen 1.65 und 1.66.

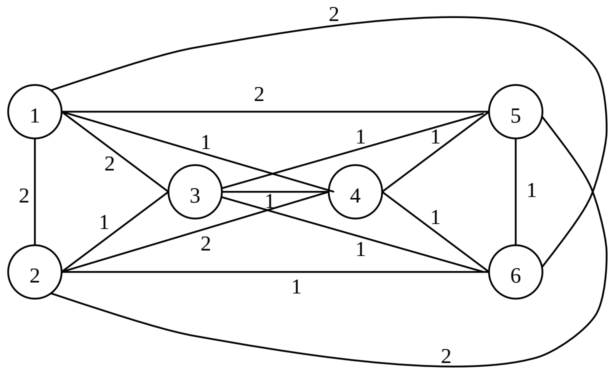


Abbildung 1.65: Beispielgraph $G = (V, E)$

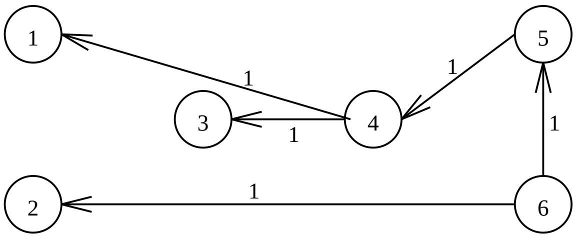


Abbildung 1.66: MST von G

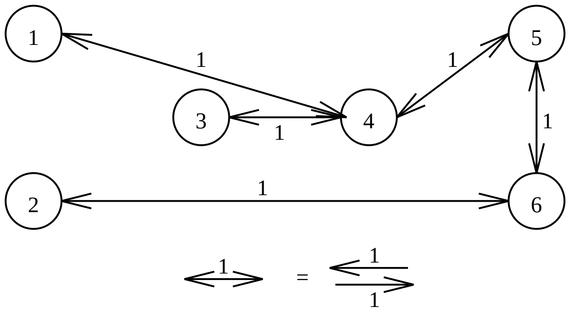


Abbildung 1.67: MST nach Verdopplung der Kanten

Offensichtlich ist $p = (2, 6, 5, 4, 3, 4, 1, 4, 5, 6, 2)$ eine Euler-Tour für T', die auf die Tour $p' = (2, 6, 5, 4, 3, 1, 2)$ gekürzt werden kann.

Die gerade angegebene Approximationsmethode kann man noch verbessern:

Satz 13 *Wenn ein Graph $G = (V, E)$ der Dreiecksungleichung genügt, so kann man in polynomieller Zeit eine Approximation des TSP finden, die höchstens um einen Faktor $3/2$ schlechter ist.*

Beweis Methode von "Christofides"

1. Bilde MST (Minimum Spanning Tree) T von G.
2. Sei W die Menge der Knoten mit ungeradem Grad in T (die Mächtigkeit von W ist gerade).
3. Bilde den Graphen $G' = (W, E(W))$.
4. Finde ein perfektes Matching P in G' mit minimalen Kosten.
5. Füge Kanten von P in T ein und erhalte T'. Alle Knoten haben jetzt geraden Grad.
6. Bilde Euler-Tour t in T'.
7. Finde Abkürzung über mehrfach besuchte Knoten. Bezeichne diese Tour mit t_{app}.

□

Warum leistet der Algorithmus das Verlangte?

Behauptung 10 $C(t_{app}) = \sum_{e \in t_{app}} c(e) \leq 3/2 \cdot L_{opt}$, *wobei* L_{opt} *die Länge der optimalen TSP-Tour bezeichnet.*

Beweis Für die Kosten des MST $T = (V, E_T)$ gilt wieder:

$$C(T) = \sum_{e \in E_T} c(e) \leq C(t_{opt}),$$

wobei t_{opt} die optimale TS-Tour bezeichnet. Für die Euler-Tour t in T' gilt:

$$C(t) = C(T') = C(T) + C(P) \leq L_{opt} + C(P).$$

Sei $\{i_1, i_2, ..., i_{2m}\}$ die Menge der Knoten aus W, in der Reihenfolge ihres Vorkommens in der optimalem TS-Tour t_{opt}. Mit anderen Worten ist $t_{opt} = (\alpha_0, i_1, \alpha_1, i_2, ..., \alpha_{2m-1}, i_{2m}, \alpha_{2m})$, wobei die α's (möglicherweise leere) Folgen von Knoten aus V sind. Betrachte die zwei Matchings mit Knoten aus W:

$$\begin{array}{rcl} M_1 & = & \{[i_1, i_2], [i_3, i_4], ..., [i_{2m-1}, i_{2m}]\}, \\ M_2 & = & \{[i_2, i_3], [i_4, i_5], ..., [i_{2m}, i_1]\}. \end{array}$$

Nach der Dreiecksungleichung gilt:

$$C(t_{opt}) \geq C(M_1) + C(M_2) \geq 2 \cdot C(P).$$

Somit ergibt sich:

$$C(t) = C(T') = C(T) + C(P) \leq L_{opt} + \frac{1}{2} \cdot L_{opt} = \frac{3}{2} \cdot L_{opt}.$$

□

Behauptung 11 *Die Schritte erzeugen einen Graph, für den man eine Euler-Tour finden kann.*

Beweis Wenn ein Knoten in T geraden Grad hat, so hat er den selben Grad in T'. Wenn ein Knoten in T ungeraden Grad hat, so kommt eine Matchingkante aus P hinzu. Somit haben alle Knoten in T' geraden Grad und damit findet man eine Euler-Tour. □

Beispiel Wir wenden den Algorithmus von Christofides auf den Graphen in Abbildung 1.68 an.
Zuerst bilden wir den MST T von G, siehe Abbildung 1.69. Die Knoten mit ungeradem Grad sind dabei schraffiert.

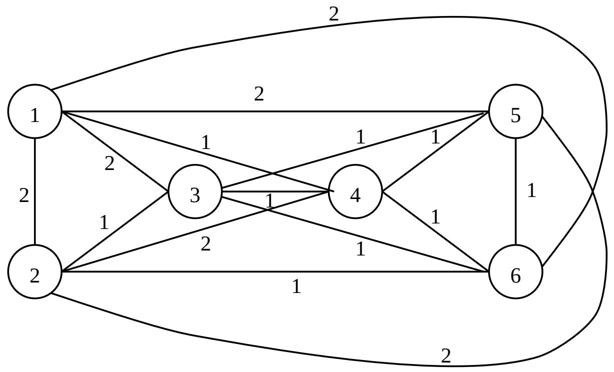


Abbildung 1.68: Beispielgraph $G = (V, E)$

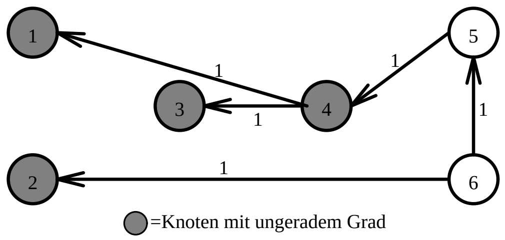


Abbildung 1.69: MST T von G

Für den Teilgraphen G' muß man ein maximales Matching P finden, siehe Abbildung 1.70.
Fügt man die Kanten von P in T ein so erhält man T', siehe Abbildung 1.71.
In T' ist die Tour $t = (2, 6, 5, 4, 1, 4, 3, 2)$ eine Euler-Tour, die auf die Tour $t_{app} = (2, 6, 5, 4, 1, 3, 2)$ gekürzt werden kann.

Vorlesung vom 20.06.1996
Michaela Altherr, Ulrich Laux und Jürgen Zimmer

1.10 Matching-Probleme in allgemeinen Graphen

Nachdem wir bisher nur Matchings in bipartiten Graphen betrachtet haben, wollen wir nun auch in allgemeinen ungerichteten Graphen *maximale* Matchings finden, d.h. Teilmengen $M \subset E$ mit maximaler Kardinalität $|M|$. Die Definition eines Matchings bleibt gleich. Zusätzlich brauchen wir aber noch ein paar einfache weiterführende Begriffe.
Dazu seien im folgenden $G = (V, E)$ ein ungerichteter Graph und $M \subset E$ ein Matching in G.

Definition 22 *Ist $v \in V$ ein Knoten von G.*

- *v heißt* besetzt*, wenn v inzident zu einer Kante $e \in M$ ist.*
- *v heißt* frei*, falls es keine solche Kante gibt.*

Definition 23 *Ein* alternierender Pfad *bezüglich M ist ein Pfad, dessen Kanten in Folge entlang des Pfades abwechselnd in M und nicht in M — d.h. in $E \setminus M$ liegen.*

Definition 24 *Ein* augmentierender Pfad *ist ein alternierender Pfad, der 2 freie Knoten verbindet.*

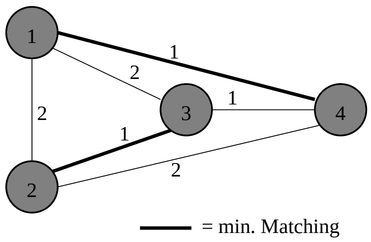


Abbildung 1.70: Vollständiger Graph G' mit minimalem Matching

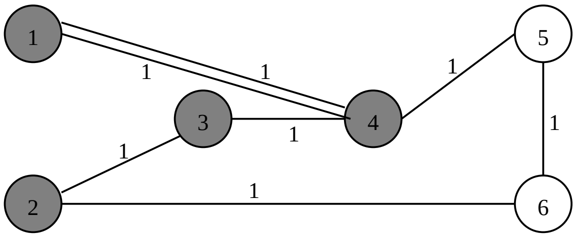


Abbildung 1.71: Graph T'

Wie ihr Name schon sagt, sind es gerade die augmentierenden Pfade, die es uns ermöglichen, ein Matching zu vergrößern. Ein solcher Pfad, der laut Definition zwei freie Knoten verbindet, besteht offensichtlich aus einer ungeraden Anzahl $k+1$ von Kanten $e \in E \setminus M$ und k Kanten $e \in M$. Durch „Invertieren" eines augementierenden Pfades erhält man also ein neues Matching, das eine Kante mehr enthält. Etwas genauer ausgeführt wird diese Idee im Beweis des folgenden ...

Lemma 10 *Ein Matching M ist nicht maximal genau dann, wenn ein augmentierender Pfad existiert.*

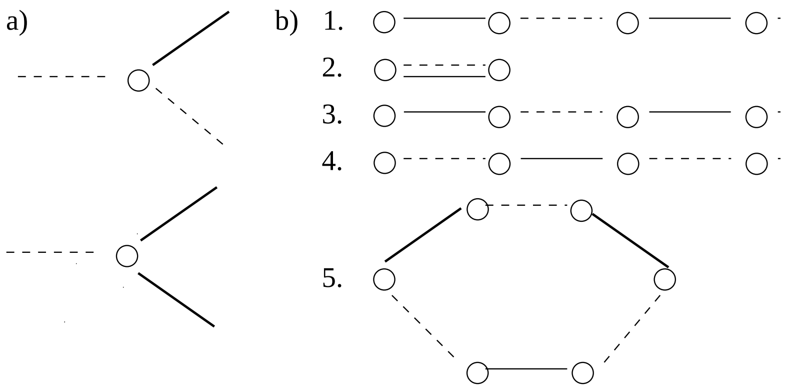


Abbildung 1.72: die unmöglichen (a) und möglichen (b) Fälle bei Vereinigung zweier Matchings

Beweis

Angenommen, es existiert ein augmentierender Pfad bezüglich eines Matchings M und es bezeichnen - - - - - Kanten in M und ——— Kanten, die nicht im Matching liegen.
Ein augmentierender Pfad ist alternierend und verbindet zwei freie Knoten, hat also folgende Gestalt:
o—–o- - - -o—–o- - - -o—–o

Nun kann man aber ein größeres Matching konstruieren, indem man in dem Graphen die Matchingkanten in Nicht-Matchingkanten umwandelt und umgekehrt.
o- - - -o—–o- - - -o—–o- - - -o
Deshalb kann M nicht maximal sein.

Sei das Matching M nicht maximal, dann gibt es sicher ein Matching R mit $|R| > |M|$. Man stelle sich vor, daß die Kanten von R (in 1.72 durchgezogen) rot, und die von M (gestrichelt) blau sind.

Betrachten wir $R \cup M$, so kann keiner der beiden Fälle in der Abbildung (a) eintreten, denn jede Ecke ist jeweils inzident zu höchstens einer Kante eines Matchings.

Wir zeigen nun, daß die Vergrößerung des Matchings M auf das Matching R nur durch „Invertieren" eines augementierenden Pfades geschehen kann, d.h. daß es einen augmentierenden Pfad geben muß.

Für Pfade in $R \cup M$ sind nur noch die in 1.72 (b) dargestellten Fälle möglich:

1. Der Pfad besteht aus einer geraden Anzahl von Kanten $\Rightarrow$ Gleichstand.
2. Eine Kante kommt sowohl in M als auch in R vor $\Rightarrow$ Gleichstand.
3. Ein augmentierenden Pfad für M.
4. Ein augmentierender Pfad für R.
5. Ein Zyklus mit alternierenden Kanten $\Rightarrow$ Gleichstand.

In den Fällen 1,2 und 3 treten genauso viele rote wie blaue Kanten auf, d.h. sie stellen keine Möglichkeiten der Matchingvergrößerung dar.

Da $|R| > |M|$ ist, muß aber der dritte Fall häufiger eintreten, als der vierte, da sich das Matching nur dabei vergrößert. In allen anderen Fällen würde sich das Matching entweder verkleinern oder gleichgroß bleiben.

Deshalb muß es für M mindestens einen augmentierenden Pfad geben.

□

Natürlich kann man mit dieser Erkenntnis gleich einen einfachen Algorithmus angeben, der ein maximales Matching berechnet. Er beginnt mit einem Matching, das aus einer beliebigen Kante besteht und erweitert dieses durch „Invertieren" augmentierender Pfade, solange ein solcher gefunden wird:

Algorithmus Max-Matching (V, E)

M={e}
wiederhole
  suche einen augmentierenden Pfad: Find-Augmenting-Path (V, E, M)
    falls es keinen gibt, dann ist man fertig: M ist maximal
    sonst vergrößere das Matching M

Dieser Algorithmus scheint sehr einfach zu sein. Wenn man einen augmentierenden Pfad gefunden hat, ist es ein leichtes, das Matching zu vergrößern. Allerdings ist es gerade die Suche nach einem solchen Pfad, die unerwartete Probleme bereiten kann. Welche Schwierigkeiten auftauchen können, sehen wir an dem folgenden Abschnitt.

1.11 Der Blüten-Schrumpf Algorithmus

Wie schon gesagt, ist es nicht trivial, in einem beliebigen (ungerichteten) Graphen einen augementierenden Pfad zu finden, falls er überhaupt existiert. Man sieht dies an dem in 1.73 gezeigten Beispiel.

Angenommen ein Algorithmus, der einen augementierenden Pfad sucht, beginnt bei dem freien Knoten v und verfolgt den alternierenden Pfad, entlang des Stengels. Erreicht der Algorithmus den Knoten w, so muß er entscheiden, welcher Knoten als nächstes besucht wird. Analog zur Breitensuche werden alle bereits besuchten Knoten markiert, um ein Backtracking zu ermöglichen. Man sieht folgendes:

- Falls die Kante (a) existiert, gibt es einen augmentierenden Pfad von v nach x, der gegen den Uhrzeigersinn um die Blüte herum läuft.
- Im Fall (b) gibt es einen augmentierenden Pfad von v nach y.

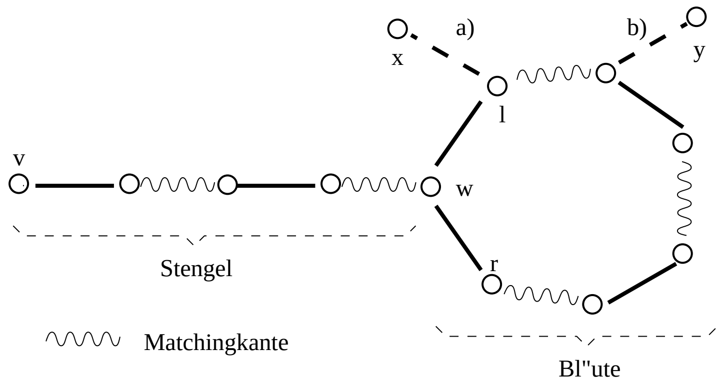


Abbildung 1.73: Eine Blütensituation beim Suchen eines augementierenden Pfades.

Entscheidet sich der Algorithmus im Fall (b), den Knoten r (für rechts) als nächstes zu besuchen, so stößt er nach Umrundung der Blüte auf den Knoten w, den er bereits besucht hat und der natürlich nicht frei ist. Es muß ein Backtracking durchgeführt werden und der Algorithmus müßte nun den Knoten l (links) wählen. Allerdings ist dieser schon als besucht markiert. Das gleiche gilt, falls im Fall (a) zuerst der Knoten l gewählt wird.
Die Idee von Edmonds war, eine durch eine solche Umrundung entdeckte Blüte zu einem Superknoten zusammenschrumpfen zu lassen und danach den Suchalgorithmus fortzusetzen. Daß diese Vorgehensweise nichts an der Existenz eines augmentierenden Pfades ändert, besagt der folgende

Satz 14 *Entsteht der Graph G' aus G durch Schrumpfen von Blüten, dann gilt*

$$G' \text{ enthält augmentierenden Pfad} \Leftrightarrow G \text{ enthält augmentierenden Pfad.}$$

Beweis

- Wenn in G' ein augmentierender Pfad existiert, so war dieser auch in G vorhanden, da G' aus G durch Schrumpfen einer Blüte zu einem Superknoten entsteht.
- Die Korrektheit des folgenden Algorithmus beweist, daß durch das Schrumpfen von Blüten keine augmentierenden Pfade zerstört werden.

$\square$

Wir benötigen einige Vorbereitungen und Begriffe:

- Zunächst werden anstelle der Kanten $\{v, w\} \in E$ die zwei Kanten $[v, w\rangle$ und $[w, v\rangle$ eingeführt.
- Außerdem werden drei Knotenprädikate eingeführt, von denen für jeden Knoten jeweils genau eines erfüllt ist. Ein Knoten ist entweder **unerreicht, gerade oder ungerade.**
- Für eine Kante $\{v, w\}$ des aktuellen Matchings M ist w =Partner(v) und v =Partner(w).
- Wir erzeugen im Laufe des Algorithmus einen Wald, der durch die Elternzeiger $p(v)$ dargestellt wird.

Algorithmus FIND-AUGMENTING-PATH (V, E, M)

Anfangs werden alle (bzgl. M) freien Knoten als *gerade* gekennzeichnet und alle Elternzeiger $p(v) = NULL$ gesetzt.
Wiederhole: Wähle eine noch nicht bearbeitete Kante $[v, w\rangle$ aus E, wobei v gerade ist.
1.Fall: Ist w ungerade, dann tue nichts.
2.Fall: Ist w unerreicht, dann setzte
w auf ungerade,
Partner(w) auf gerade,
$p(w) = v$ und
$p(Partner(w)) = w$.
3.Fall: Ist w gerade und v und w liegen in verschiedenen Bäumen, dann gibt es
einen augmentierenden Pfad zwischen den Wurzeln der beiden Bäume, also war

die Suche erfolgreich.
4.Fall: Ist w gerade und v und w liegen im gleichen Baum, dann haben wir eine Blüte entdeckt und müssen diese schrumpfen. Der entstehende Superknoten erhält den Zustand „gerade“, damit er in den Baum paßt.

Die Korrektheit dieses Algorithmus soll hier nicht gezeigt werden. Allerdings kann man eine Aussage über sein Laufzeitverhalten machen, das sich als äußerst ungünstig herausstellt. Dazu sei im folgenden $n = |V|$ die Anzahl der Knoten unseres Graphen.
Bei jedem Aufruf von FIND-AUGMENTING-PATH werden höchstens n^2 Kanten betrachtet, denn in einem Graphen mit n Knoten kann es höchstens n^2 Kanten geben. Im Extremfall muß man bei jeder Kante eine Blüte schrumpfen, was in Zeit $O(n)$ von statten geht, da höchstens n Knoten betrachtet werden. Also benötigt das Suchen eines augmentierenden Pfades mit FIND-AUGMENTING-PATH $O(n^3)$ Schritte.
In dem Graphen (V, E) kann es aber höchstens $\frac{n}{2}$ augmentierende Pfade geben. Für unseren Algorithmus MAX-MATCHING ergibt sich eine Gesamtlaufzeit von $O(n^4)$, was nicht gerade sehr effizient ist, allerdings wurden bei den obigen Betrachtungen auch sehr grobe obere Schranken benutzt.
Neben der schlechten Laufzeit treten außerdem große Probleme bei der Blütenverwaltung auf.

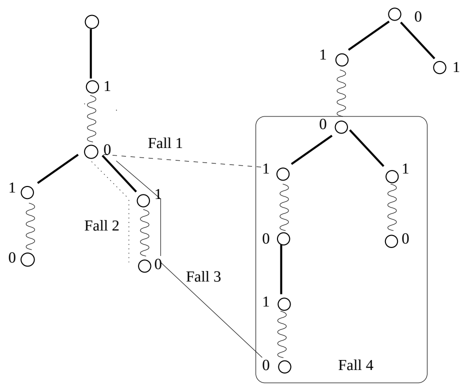


Abbildung 1.74: Ein Beispiel.

Die verschiedenen Fälle des Schleifenrumpfes sind noch einmal etwas anschaulicher in dem in 1.74 dargestellten Beispiel aufgeführt.

Kapitel 2

Approximationsalgorithmen

Vorlesung vom 26.6.1996
Stefanie Zacher, Peter Noss

2.1 Planungsprobleme

2.1.1 PCS - Precedence Constrained Scheduling

Wir wollen das Problem PCS, das Planen mit Vorrangbedingungen, näher untersuchen: Gegeben ist ein gerichteter, azyklischer Graph G und eine positive, ganze Zahl $m \in \mathbb{N}$. Gesucht ist ein "Schedule" (Plan) $S : V \to \mathbb{N}$ mit den Eigenschaften

1. $[v, w\rangle \in E \quad \Rightarrow \quad S(v) \leq S(w)$ und
2. $\forall k \in \mathbb{N} : \quad | S^{-1}(k) | \leq m$,

der $max\{S(v) \mid v \in V\}$ minimiert.
Anschaulich besagt dies, daß man m Maschinen zur Verfügung hat, die pro Zeiteinheit jeweils einen Knoten abarbeiten können. Für jede Kante muß der Startknoten zeitlich vor dem Endknoten bearbeitet werden. Insgesamt sollen die Aufgaben an den Knoten so auf die Maschinen verteilt werden, daß die insgesamt benötigten Zeiteinheiten minimal sind. Siehe dazu auch das folgende Beispiel:

Beispiel
Gegeben ist ein Graph G mit 5 Knoten und 6 Kanten, siehe Abbildung 2.1. Für $m = 2$ benötigt man mindestens 4 Zeiteinheiten, um alle Aufgaben zu erledigen.

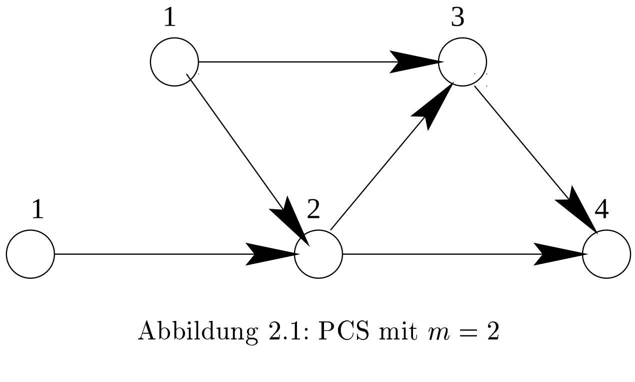


Abbildung 2.1: PCS mit $m = 2$

Das PCS-Problem ist $\mathcal{NP}$-vollständig, wie wir nachher noch sehen werden. Es wäre also schon sehr gut, wenn wir einen Algorithmus angeben könnten, dessen Ergebnisse nicht allzu sehr von der optimalen Lösung abweichen. Solche Algorithmen nennt man *Approximationsalgorithmen.* Wir werden jetzt einen Approximationsalgorithmus angeben, von dem man beweisen kann, daß die Ergebnisse höchstens um einen Faktor 2 schlechter sind als die optimalen Lösungen.

APPROXIMATIONSALGORITHMUS

1. Markiere jeden Knoten mit seiner “Tiefe”, d.h. mit der Länge des längsten gerichteten Pfades (wir zÄhlen die Anzahl der Knoten), der dort endet.

2. Sei d_i die Anzahl der Knoten mit Tiefe i. Verwende die ersten $\lceil \frac{d_1}{m} \rceil$ Zeiteinheiten für Knoten der Tiefe 1, verwende die nächsten $\lceil \frac{d_2}{m} \rceil$ Zeiteinheiten für Knoten der Tiefe 2, usw.

Wir werden jetzt zeigen, daß dieser Algorithmus die optimale Lösung gut approximiert:

Satz 15 *Sei L die “Länge” des konstruierten Plans und L_{opt} die “Länge” eines optimalen Plans. Es gilt:*

$$L \leq 2 \cdot L_{opt}.$$

Beweis

1. Sei t die maximale Tiefe eines Knotens im Graphen. Dann folgt sofort $L_{opt} \geq t$.

2. Andererseits gilt

$$L_{opt} \geq \frac{\#\ \text{Knoten}}{m} = \frac{d_1 + d_2 \ldots + d_t}{m}.$$

Mit diesen beiden Aussagen erhält man

$$L = \sum_{i=1}^{t} \left\lceil \frac{d_i}{m} \right\rceil \leq \sum_{i=1}^{t} \left(\frac{d_i}{m} + 1 \right) + \underbrace{\sum_{i=1}^{t} \frac{d_i}{m}}_{\leq L_{opt} \text{ wegen 2.}} + \underbrace{\sum_{i=1}^{t} 1}_{\leq L_{opt} \text{ wegen 1.}} \leq 2 \cdot L_{opt}.$$

□

Es bleibt noch zu zeigen, das PCS ein $\mathcal{NP}$-vollständiges Problem ist, daß wir also nicht hoffen können, einen einfachen polynomiellen Algorithmus zu finden. Man kann sogar noch eine stärkere Aussage beweisen: man wird keinen Algorithmus finden, der besser als $4/3 \cdot L_{opt}$ approximiert, es sei denn es gilt $\mathcal{P} = \mathcal{NP}$.

Satz 16 *Wenn es einen polynomiellen Approximationsalgorithmus gibt, der stets $L_{opt} < \frac{4}{3} L_{opt}$ garantiert, dann gilt $\mathcal{P} = \mathcal{NP}$.*

Beweis Man reduziert PCS auf das CLIQUE-Problem, von dem man weiß, daß es $\mathcal{NP}$-vollständig ist: Gegeben ist ein Graph G und eine positive, ganze Zahl $k \in \mathbb{N}$. Gibt es in G einen vollständigen Teilgraphen (einen Graphen, in dem alle Knoten untereinander verbunden sind) mit k Knoten?
Wir konstruieren aus einer beliebigen Instanz (V, E, k) eines CLIQUE-Problems eine Instanz eines PCS-Problems, das

1. auf jeden Fall in vier Zeiteinheiten lösbar ist und

2. das genau dann in drei Zeiteinheiten lösbar ist, wenn (V, E) eine Clique der Größe k hat.

Das bedeutet, daß ein PCS-Algorithmus, der besser als $4/3 \cdot L_{opt}$ approximiert, genau dann die Instanz des PCS-Problems in 3 Zeiteinheiten löst, wenn (V, E, k) eine k-Clique besitzt. Damit könnte man das k-Clique Problem in polynomieller Zeit lösen und es wäre $\mathcal{P} = \mathcal{NP}$.
Nun zur Konstruktion des PCS-Problems: Als “Aufgaben” nimmt man die Menge $V \cup E \cup F_1 \cup F_2 \cup F_3$, wobei die F_i “Füllaufgaben” sind, deren Notwendigkeit man gleich sieht. Als Vorrangbedingungen sollen gelten:

1. Ein Knoten v hat Vorrang vor einer Kante e, wenn v inzident zu e ist, d.h. eine Kante darf erst dann abgearbeitet werden, wenn beide Endknoten schon abgearbeitet sind.

2. Alles in F_i hat Vorrang vor allem in F_{i+1} für $i = 1, 2$.

Wähle dabei $|F_i|$ so, daß

a) $m = k + |F_1|$,

b) $m = (n - k) + \binom{k}{2} + |F_2|$,

c) $m = |E| - \binom{k}{2} + |F_3|$ und

d) $m, |F_1|, |F_2|, |F_3| > 0$

gelten. So eine Konstruktion geht immer.

Behauptung 12 *Es existiert ein Plan, der nur vier Zeiteinheiten braucht:*

ZE 1: F_1 und k Knoten in V,

ZE 2: F_2 und $n - k$ Knoten in V,

ZE 3: F_3 und einige Kanten in V,

ZE 4: restliche Kanten; das ist immer möglich, denn man kann $m \geq |E|$ noch zu den Bedingungen a),... d) hinzufügen.

Behauptung 13 *Wenn es in dem Graphen eine k-Clique gibt, dann ist das PCS-Problem in drei Zeiteinheiten lösbar:*

ZE 1: F_1 und die k Knoten der Clique,

ZE 2: F_2, die restlichen $n - k$ Knoten und $\binom{k}{2}$ Kanten,

ZE 3: F_3 und die restlichen Kanten.

Behauptung 14 *Wenn das PCS-Problem in drei Zeiteinheiten lösbar ist, dann gibt es eine k-Clique:* Wenn k Knoten keine Clique bilden, dann können die m Maschinen im 2. Schritt keine $\binom{k}{2}$ Kanten abarbeiten. Da F_1 im ersten, F_2 im zweiten und F_3 im dritten Schritt wegen der Vorrangbedingungen abgearbeitet werden müssen, kann es keine Lösung geben, die nur drei Schritte benötigt.

Damit ist der Satz bewiesen. □

2.1.2 SIT - Scheduling of Independent Tasks

Als nächstes betrachten wir das Planen unabhängiger Aufgaben. Die Eingabe ist dabei eine Menge von Zahlen $w_1, \ldots, w_n \in \mathbb{N}$, die Länge der Aufgaben, und eine positive, ganze Zahl $m \in \mathbb{N}$, die Anzahl der Maschinen. Die Aufgabe ist, eine Zuordnung $S : \{1, \ldots, n\} \to \{1, \ldots, m\}$ zu finden, so daß der Ausdruck $\left(\max \sum_{i:S(i)=k} w_i\right)$ minimal wird. Die Aufgaben sollen also so auf die einzelnen Maschinen verteilt werden, daß jede Maschine möglichst wenig arbeitet.

Auch dieses Problem ist $\mathcal{NP}$-vollständig und daher suchen wir wieder einfache, aber effektive Approximationsalgorithmen.

Einfacher Approximationsalgorithmus

1. Sortiere die Aufgaben der Länge nach aufsteigend: $w_1 \geq w_2 \geq \ldots \geq w_n$.

2. Plane eine Aufgabe nach der anderen: Ordne w_i der Maschine zu, die bis jetzt am wenigsten zu tun hat.

Beispiel Gegeben sind 5 Aufgaben der Längen 5, 3, 2, 2 und 1, siehe Abbildung 2.2. Es sind drei Maschinen zur Verfügung. Der Approximationsalgorithmus ordnet die Aufgaben den Maschinen wie in Abbildung 2.3 dargestellt zu.

Abbildung 2.2: Aufgaben nach Länge sortiert

Auch für diesen Approximationsalgorithmus kann man eine obere Schranke bezüglich der Abweichung von der optimalen Lösung angeben:

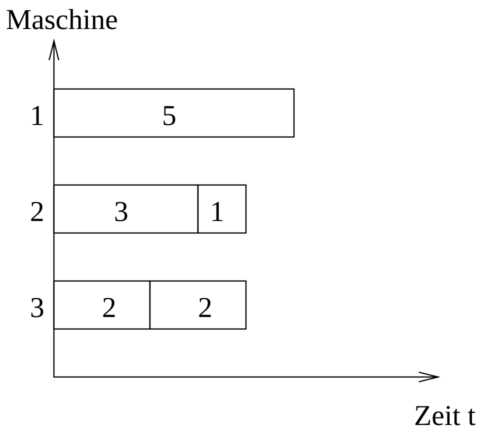


Abbildung 2.3: Aufgaben nach Länge den Maschinen zugeteilt

Satz 17 *Sei L die Länge des durch den Approximationsalgorithmus erzeugten Plans und L_{opt} die Länge des optimalen Plans. Es gilt:*

$$L \leq (\frac{4}{3} - \frac{1}{3m}) \cdot L_{opt}.$$

Beweis
Wir unterscheiden verschiedene Fälle:

Fall 1: w_n wird nicht "als letztes" fertig, d.h. die Maschine, die Aufgabe w_n abarbeitet ist nicht die Maschine, die am längsten arbeitet. Dann hat die letzte Aufgabe keinen Einfluß auf die Länge des erzeugten Plans:

$$\begin{aligned} L(w_1, \ldots, w_n) &= L(w_1, \ldots, w_{n-1}) \\ &\overset{IV}{\leq} (\frac{4}{3} - \frac{1}{3m}) \underbrace{L_{opt}(w_1, \ldots, w_{n-1})}_{\leq L_{opt}(w_1, \ldots, w_n)}. \end{aligned}$$

Fall 2: w_n wird nicht als letztes fertig. Setze $W := \sum w_i$. In diesem Fall ist jede Maschine mindestens $L - w_n$ lang beschäftigt, da die Aufgabe immer der Maschine zugeteilt wird, die bis jetzt am wenigsten ausgelastet ist. Daher gilt

$$m \cdot L \leq W + (m-1)w_n,$$

und aus der Definition von L_{opt} folgt

$$W \leq m \cdot L_{opt}.$$

Daraus kann man

$$L \leq L_{opt} + \left(\frac{m-1}{m}\right) w_n$$

folgern. Jetzt kann man wiederum zwei Fälle unterscheiden:

Fall a)

$$\underbrace{\frac{m-1}{m}}_{1-\frac{1}{m}} w_n \leq \underbrace{\left(\frac{1}{3} - \frac{1}{3m}\right)}_{\frac{1}{3}\left(1-\frac{1}{m}\right)} L_{opt},$$

bzw. äquivalent

$$w_n \leq \frac{1}{3} L_{opt}.$$

Daraus folgt sofort die Behauptung.

Fall b)

$$w_n > \frac{1}{3} L_{opt}, \quad \text{d.h.} \quad w_i > \frac{1}{3} L_{opt} \qquad \forall i.$$

Die einzelnen Aufgaben sind so lang, daß im optimalen Plan jede Maschine höchstens zwei Aufgaben erledigt, d.h. $n \leq 2m$. Ohne Einschränkung nehmen wir $n = 2m$ an (füge Aufgaben der Länge 0 dazu). Auf Übungsblatt 7 wird gezeigt, daß in diesem Fall der vom Approximationsalgorithmus erstellte Plan sogar optimal ist.

□

Vorlesung vom 27.06.1996
Daniel Kästner, Daniel Kreutzer

Wir betrachten nun eine leicht veränderte Aufgabenstellung für SIT. Dabei nehmen wir an, daß die Anzahl m der Maschinen nicht mehr Teil der Eingabe, sondern fest vorgegeben ist.

- SIT_m:
  - Eingabe: Zahlen $w_1, w_2, \ldots, w_n \in \mathbb{N}$, die Längen von Aufgaben repräsentieren.
  - Aufgabe: Finde einen Plan S mit $S : \{1, \ldots, n\} \to \{1, \ldots, m\}$, so daß $(\max_k \sum_{i:S(i)=k} w_i)$ minimal wird. Dadurch wird die maximale Auslastzeit aller Maschinen minimiert.

Algorithmus „Sozialer“ Algorithmus SA

1. Sortiere die w_i absteigend nach ihrer Größe, so daß gilt:
   $w_1 \geq w_2 \geq \ldots \geq w_n$
2. **für** $i = 1$ **bis** n
   ordne w_i der am wenigsten ausgelasteten Maschine zu

Da lange Aufgaben eine größere Auswirkung auf die Planlänge haben, können wir den obigen Algorithmus verbessern, indem wir ihnen mehr Aufmerksamkeit schenken, als den kürzeren. Daraus ergibt sich dann der folgende Algorithmus:

AlgorithmusSA(k)

1. Sortiere die w_i absteigend nach ihrer Größe, so daß gilt:
   $w_1 \geq w_2 \geq \ldots \geq w_n$.
2. Berechne den *optimalen* Plan für die k längsten Aufgaben, also für $w_1, w_2, \ldots, w_k$.
3. **für** $i = k + 1$ **bis** n
   ordne w_i der am wenigsten ausgelasteten Maschine zu

Satz 18 *Sei $L_{SA(k)}$ die Länge des von SA(k) erzeugten Planes und L_{opt} die Länge des optimalen Planes. Dann gilt:*

$$L_{SA(k)} \leq \left(1 + \frac{m-1}{k+1}\right) \cdot L_{opt}.$$

Beweis Sei L die Länge des optimalen Plans für $w_1, w_2, \ldots, w_k$.

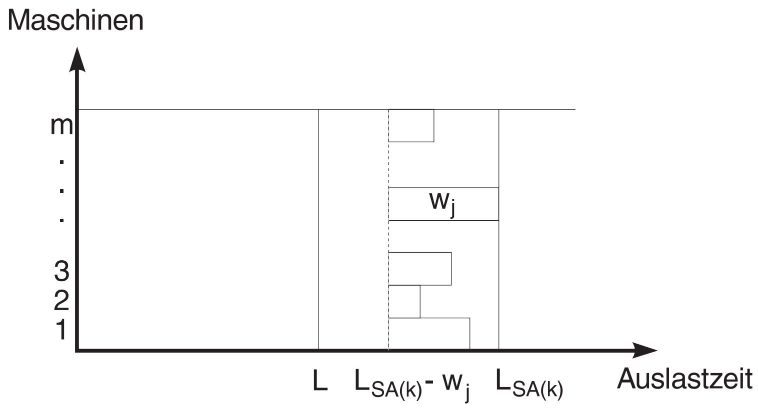


Abbildung 2.4: Maschinenauslastung

1. Gilt $L_{SA(k)} = L$, dann wissen wir folgendes:

$$\begin{aligned} & L = L_{opt}(w_1, w_2, \ldots, w_k) \leq L_{opt}(w_1, w_2, \ldots, w_n) \leq L_{SA(k)} = L \\ \Rightarrow \quad & L_{opt}(w_1, w_2, \ldots, w_k) = L_{opt}(w_1, w_2, \ldots, w_n) \end{aligned}$$

   Der Algorithmus berechnet somit tatsächlich den optimalen Plan.

2. Sei nun $L_{SA(k)} > L$. Betrachte dazu die Abbildung 2.4.

   Annahme: Aufgabe j endet mit Zeit $L_{SA(k)}$

   Als w_j eingefügt wurde, war Maschine k diejenige, welche am wenigsten ausgelastet war.

   Betrachten wir nun den Flächeninhalt aller Rechtecke, die die Maschinenauslastung für die einzelnen Aufgaben repräsentieren, so ist er größer als der des Rechtecks mit Seitenlängen $L_{SA(k)} - w_j$ und m. (Dabei haben die Seiten des Einzelrechtecks einer Aufgabe l die Längen w_l und 1.) Das bedeutet: Da jede Maschine mindestens $L_{SA(k)} - w_j$ Zeiteinheiten verbrauchte, sind für die ersten $j-1$ Aufgaben auch mindestens $m \cdot (L_{SA(k)} - w_j)$ Einheiten notwendig.

   Daraus ergibt sich die folgende Ungleichung:

$$\frac{(w_1 + w_2 + \ldots + w_{j-1})}{m} \geq L_{SA(k)} - w_j. \tag{2.1}$$

   Andererseits gilt auch:

$$L_{opt} \geq \frac{(w_1 + w_2 + \ldots + w_j)}{m}.$$

   Durch Umformung erhalten wir

$$-L_{opt} \leq -\frac{(w_1 + w_2 + \ldots + w_j)}{m}. \tag{2.2}$$

   Durch Addition von 2.1 und 2.2 ergibt sich schließlich

$$L_{SA(k)} - L_{opt} \leq \left(1 - \frac{1}{m}\right) \cdot w_j$$

   Da die w_i's absteigend sortiert sind, gilt

$$\begin{aligned} \left(1 - \frac{1}{m}\right) \cdot w_j &\leq \left(1 - \frac{1}{m}\right) \cdot w_{k+1} \\ &= (m-1) \cdot \frac{w_{k+1}}{m} \\ &\leq \frac{m-1}{k+1} \cdot L_{opt}. \end{aligned}$$

Die letzte Umformung gilt wegen Gleichung 2.2, denn

$$L_{opt} \geq \frac{(w_1 + w_2 + \ldots + w_{k+1})}{m}.$$

Wegen $w_1 \geq w_2 \geq \ldots \geq w_{k+1}$ folgt daraus:

$$\frac{(w_1 + w_2 + \ldots + w_{k+1})}{m} \geq \frac{(k+1) \cdot w_{k+1}}{m}.$$

Insgesamt erhalten wir somit:

$$\begin{aligned} L_{SA(k)} &\leq \underbrace{\left(1 - \frac{1}{m}\right) \cdot w_j}_{\leq \frac{m-1}{k+1} \cdot L_{opt}} + L_{opt} \\ &\leq \left(1 + \frac{m-1}{k+1}\right) \cdot L_{opt}. \end{aligned}$$

□

Sehen wir uns nun noch eine *Laufzeitabschätzung* für SA(k) an:

- Schritt 1: $\mathcal{O}(n \log n)$ zum Sortieren der w_i.
- Schritt 2: Probiere alle m^k Möglichkeiten durch und teste für jede, welche Maschine am wenigsten ausgelastet ist; das erfordert Zeit $\mathcal{O}(m^{k+1})$. Je größer dabei k wird, desto kleiner wird die Konstante $1 + \frac{m-1}{k+1}$, die Approximation wird also immer besser. Drücken wir k in Abhängigkeit von einer Konstante ε aus, so führt der Algorithmus SA(k(ε)) zu einer Approximation mit Genauigkeit $1 + \varepsilon$, also

  $$L_{SA(k(\varepsilon))} \leq (1 + \varepsilon) \cdot L_{opt}.$$

  Dazu muß für ε gelten

  $$\begin{aligned} \varepsilon &\geq \frac{m-1}{k+1} \\ \Leftrightarrow \qquad k+1 &\geq \frac{m-1}{\varepsilon}. \end{aligned}$$

  Damit erhalten wir eine Laufzeit von $\mathcal{O}(m^{\frac{m-1}{\varepsilon}})$. Da m fest vorgegeben ist, ist die Laufzeit also konstant.
- Schritt 3: Da m konstant ist, $\mathcal{O}(n)$
- Die Gesamtlaufzeit beträgt also $\mathcal{O}(m^{\frac{m-1}{\varepsilon}}) + \mathcal{O}(n \log n)$.

Wir haben bisher einige Approximations–Algorithmen kennengelernt, und haben für diese stets angegeben, wie "gut" die jeweilig berechneten Lösungen sind. Im nächsten Abschnitt werden wir nun bestimmte Klassen von Approximationen betrachtet, und im Anschluß untersuchen, wie sich die uns bekannten Algorithmen darin einordnen lassen.

Definition 25 *Sei Π ein Minimierungsproblem mit Optimum > 0, I eine Instanz von Π und L_{opt} eine optimale Lösung von I.*

1. *Eine Lösung von I mit Wert $L \leq (1 + \varepsilon) \cdot L_{opt}$ heißt* ε**-Approximation**.
2. *Ein* ε**-Approximationsalgorithmus** *für Π berechnet für jedes I eine ε-Approximation.*
3. *Ein* **Approximationsschema** *für Π ist eine Methode, die für jedes $\varepsilon > 0$ einen ε-Approximationsalgorithmus für Π liefert.*

4. *Ein* **PTAS** *(Polynomial Time Approximation Scheme; Polynomialzeit-Approximationsschema) für* Π *ist ein Schema, das für jedes* $\varepsilon > 0$ *einen* ε*-Approximationsalgorithmus für* Π *mit polynomieller Laufzeit (in der Größe von* I*) liefert.*

5. *Ein* **FPTAS** *(Fully PTAS) ist ein PTAS, bei der die Laufzeit der erzeugten Algorithmen polynomiell in Größe von* I *und in* $\frac{1}{\varepsilon}$ *ist.*

Einordnung der bisher behandelten Probleme in diese Kategorien:

1. TSP: Es gibt keinen ε-Approximationsalgorithmus für TSP mit polynomieller Laufzeit für alle ε – es sei denn $\mathcal{P} = \mathcal{NP}$. TSP ist somit sehr schwer approximierbar.

2. $\triangle$TSP: Für $\triangle$TSP gibt es einen polynomiellen $\frac{1}{2}$- Approximationsalgorithmus. Es ist jedoch nicht bekannt, ob es für $\triangle$TSP ein PTAS gibt.

   Für das „euklidische TSP" gibt es ein PTAS; die Laufzeit beträgt $\mathcal{O}(n^{\frac{100}{\varepsilon}})$, bzw $\mathcal{O}(n^{\frac{30}{\varepsilon}})$ (euklidisches TSP: Gegeben seien n Punkte einer Ebene; die euklidischen Distanzen zwischen ihnen definieren einen Graphen. Für den soll dann das TSP-Problem gelöst werden.)

3. PCS: Für jedes PCS gibt es einen polynomiellen 1-Approximationsalgorithmus. Gäbe es ein PTAS für PCS, dann wäre $\mathcal{P} = \mathcal{NP}$ (da es keinen ε-Approximationsalgorithmus für $\varepsilon < \frac{1}{3}$ gibt).

4. SIT_m: Für SIT_m gibt es ein PTAS, aber kein FPTAS – es sei denn $\mathcal{P} = \mathcal{NP}$.

Vorlesung vom 1.7.1996
Roland Schlosser

2.2 Das Gewichtete Rucksackproblem

In diesem Abschnitt wollen wir uns nun weitere Probleme ansehen, für deren Lösung wir Approximationsalgorithmen angeben können.

2.2.1 Das Rucksackproblem

Die Aufgabenstellung des Rucksackproblems besteht darin, einen Rucksack der Tragekapaziät b so mit n Gegenständen (von jeweiligem Gewicht w_i, $1 \leq i \leq n$) so zu füllen, daß b genau ausgeschöpft wird. Wir können dieses Problem also wie folgt aufschreiben:

- Eingabe: $w_1, ...w_n, b \in \mathbb{N}$.
- Frage: $\exists J \subset \{1, ..., n\} : \sum_{j \in J} w_j = b$

Es ist bekannt, daß das Rucksackproblem NP-vollständig ist. Folgender Algorithmus berechnet eine Lösung:

Algorithmus
Der Algorithmus benutzt ein Feld $B[0..b]$, so daß wir folgende Invariante in jeder Iteration aufrecht erhalten können:

$$B[\beta] = 1 \quad \Leftrightarrow \quad \exists J \subset \{1,..,i\} : \sum_{j \in J} w_j = \beta, \quad \text{für} \quad (1 \leq i \leq n).$$

$B[i] = 0$ für $(1 \leq i \leq b)$; B[0] = 1;
for $i = 1$ **to** n **do**
  for $\beta = 0$ **to** $b - w_i$ **do**
    if $B[\beta] = 1$ **then** $B[\beta + w_i] = 1$
  endfor
endfor
return $(B[b])$

Man sieht leicht ein, daß die Laufzeit dieses Algorithmus $\mathcal{O}(n * b)$ ist. Wir nennen diese Laufzeit "pseudopolynomiell"; d.h., polynomiell in der Anzahl der Gegenstände und in der Größe der vorkommenden Zahlen.

2.2.2 Das gewichtete Rucksackproblem

Wir betrachten nun eine Variante des Rucksackproblems. Dabei erhält jeder Gegenstand j zusätzlich einen Preis c_j. Die Aufgabe besteht dann darin, den Rucksack so zu packen, daß er möglichst teuer wird; dabei darf die Größe b des Rucksacks nicht überschritten werden. Schreiben wir für eine Teilmenge J von $\{1, \dots, n\}$ abkürzend

$$c_J = \sum_{j \in J} c_j \quad \text{und} \quad w_J = \sum_{j \in J} w_j,$$

so können wir das Problem folgendermaßen beschreiben: Wir suchen einen Algorithmus mit den Eigenschaften:

Eingabe:	$w_1, .., w_n$	"Gewichte"
	b	Rucksackgröß (oBdA $w_i \leq b \quad, \forall i$)
	$c_1, .., c_n$	Kosten

Finde: $J \subseteq \{1, .., n\}$ so daß c_J maximal wird unter der Nebenbedingung $w_J \leq b$.
Zur Lösung des Problems wollen wir eine Tafel F berechnen, die die Einträge

$$F(c, i) := \min\{w_J | J \subset \{1, .., i\}, c_J = c\}$$

enthält. Dann erhalten wir eine Lösung des Problems, indem wir in der letzten Zeile dieser Tafel, die noch Einträge kleiner als b enthält einen maximalen Eintrag suchen. Die Tafel selbst können wir leicht rekursiv berechnen, wenn wir folgende Beobachtungen machen:

$F(0, 0) = 0$
c $\neq$ 0: $F(c, 0) = \infty$
i $\geq$ 0: $F(c, i) = \min\{\ F(c, i-1), F(c - c_i, i-1) + w_i\}$
$F(c, i-1)$ gilt, falls $w_i \notin J^*$
$F(c - c_i, i-1)$ gilt, falls $w_i \in J^*$

Da die maximalen Kosten, die der Rucksack haben wird, durch $\overline{C} = \sum_{j=1}^{n} c_j$ abgeschätzt werden können (wir können höchstens *alle* Gegenstände hineinlegen) genügt es, die Tafel F nur für $c \leq \overline{C}$ zu berechnen. Damit können wir einen Algorithmus angeben, der das gewichtete Rucksackproblem löst.

Algorithmus
for $i = 1$ **to** n **do**
  for $c = 1$ **to** $\overline{C}$ **do**
    $F(c, i) = \min\{\ F(c, i-1), F(c - c_i, i-1) + w_i\}$
  endfor
endfor

Die Laufzeit dieses Algorithmus ist $\mathcal{O}(n\overline{C})$, also wieder " pseudopolynomiell". Im folgenden Abschnitt werden wir zeigen, wie wir das Problem approximativ lösen können; wir werden sehen, daß es sogar möglich ist, ein FPTAS anzugeben.

2.2.3 FPTAS für gewichtetes Rucksackproblem

Wir wollen zeigen, daß ein FPTAS für das gewichtete Rucksackproblem existiert. Dazu müssen wir für jedes ε einen ε –Approximations–Algorithmus angeben, dessen Laufzeit sowohl in der Eingabegröße als auch in $1/\varepsilon$ polynomiell ist. Zuvor haben wir einen Algorithmus kennengelernt, der das Problem in Laufzeit $\mathcal{O}(n\overline{C})$ exakt löst. Diese Laufzeit hängt also von den Kosten der Gegenstände, die verpackt werden sollen, ab; je billiger die Gegenstände, desto schneller können wir eine Lösung berechnen. Die Idee für einen Approximations–Algorithmus besteht nun darin, das Problem zu skalieren. Wir werden einen Faktor S wählen, und alle Preise durch S dividieren. Wir erhalten also zu jeder Instanz $Q = (w_1, ..w_n, b, c_1, .., c_n)$ des gewichteten Rucksackproblems eine Instanz $Q_S := (w_1, ..w_n, b, \lfloor c_1/S \rfloor, .., \lfloor c_n/S \rfloor)$ des skalierten Problems. Q_S kann dann in Zeit $\mathcal{O}(n\overline{C}/S)$ gelöst werden, und jede optimale Lösung des skalierten Problems ist eine Näherungslösung für das ursprüngliche Problem. Wir müssen noch darauf achten, daß der Faktor S so gewählt wird, daß die Approximation noch hinreichend gut ist. Dazu werden wir nun die Kosten zweier jeweils optimaler Lösungen miteinander vergleichen. Seien dazu

$K \subset \{1, ..n\}$ optimal für Q_S und
$J \subset \{1, ..n\}$ optimal für Q.

Dann erhalten wir

$$c_K \;=\; \sum_{j \in K} c_j \;=\; S \cdot \sum_{j \in K} c_j/S \;\geq\; S \cdot \sum_{j \in K} \lfloor c_j/S \rfloor,$$

weil k optimal für Q_S; und weiter

$$\geq\; S \cdot \sum_{j \in J} (c_j/S - 1) \;=\; \sum_{j \in J} c_j - S \cdot |J| \;\geq\; c_J - S \cdot n.$$

Also gilt

$$L \quad \geq \quad L_{opt} - S \cdot n.$$

Für $-S \cdot n \geq -\varepsilon \cdot L_{opt}$ gilt also

$$L \;\geq\; (1 - \varepsilon) \cdot L_{opt}.$$

Wir können also für jedes ε einen Faktor S angeben, der uns eine entsprechend gute Näherungslösung für unser Problem liefert. Allerdings haben wir S bisher nur in Abhängigkeit von L_{opt} ausgedrückt, welches wir jedoch nicht kennen. Daher verwenden wir noch eine weitere Abschätzung. Wir können ohne Einschränkung davon ausgehen, daß jeder der Gegenstände alleine in den Rucksack hineinpaßt. Also können wir in jedem Fall die Kosten $U = \max c_j \ (1 \leq j \leq n)$ erzielen, wenn wir nur den teuersten Gegenstand in den Rucksack legen. Damit erhalten wir für S die folgende Schranke:

$$S \leq (\varepsilon \cdot U)/n \quad \text{und damit auch} \quad S \leq (\varepsilon \cdot L_{opt})/n.$$

Daraus ergibt sich folgende Laufzeit:

$$\mathcal{O}(n \cdot \overline{C}/S) \;\leq\; \mathcal{O}\left(\frac{n \cdot \overline{C}}{\frac{\varepsilon \cdot \max_i c_i}{n}}\right) \;=\; \mathcal{O}\left(\frac{n^2 \cdot \overline{C}}{\varepsilon \cdot \max_i c_i}\right) \;\leq\; O(n^3/\varepsilon),$$

da $\max_i c_i \geq \overline{c}/n$.
Dieses Ergebnis fassen wir noch in einem Satz zusammen.

Satz 19 *Für das gewichtete Rucksackproblem gibt es ein FPTAS mit Laufzeit $O(n^3/\varepsilon)$*

Kapitel 3

Branch and Bound

In diesem Abschnitt werden wir ein weiteres Verfahren kennenlernen, mit dem man Optimierungsprobleme lösen kann, die sogenannte *Branch and Bound* Methode. Ziel dabei ist es, das Minimum einer Funktion auf einer gegebenen Menge zu suchen. Bei dieser Suche behält man die jeweils beste bereits bekannte Lösung, und verwendet diese dazu den Suchraum für weitere Lösungen einzuschränken.

3.1 Der Algorithmus

Wir werden die Branch and Bound Methode nun genauer formulieren. Wir betrachten das folgende Problem:

- Gegeben ist eine endliche Menge X sowie eine Abbildung $f : X \to \mathbb{R}$. Die Menge X ist in der Regel sehr groß und nur implizit gegeben .
- Gesucht ist $z = min\{f(x)|x \in X\}$ (realisiert durch x_0).

Dabei legen wir noch weitere Annahmen zugrunde:

1. $f(x)$ läßt sich leicht berechnen.
2. Sei $Y \subset X$ eine Teilmenge von X, dann kann man einen Wert μ_Y (leicht) berechnen, für den gilt:

$$\mu_Y \leq f(y), \quad \forall y \in Y.$$

   Das Minimum von f auf der Menge Y ist dann sicherlich nicht kleiner als μ_Y.

Damit können wir nun einen Algorithmus formulieren.

Algorithmus Findmin ($Y \subset X$)

if $Y = \{\}$ **then return**
if $Y = \{y\}$ **then** $currentmin = \min\{currentmin, f(y)\}$
  else
    if $currentmin \leq y$ **then return**
    Partitioniere Y in Teilmengen $Y_1, .., Y_n$
    for each $i \in \{1, .., n\}$ **do** Findmin(Y_i)

In dieser Form ist der Algorithmus sehr allgemein beschrieben. In den folgenden Punkten kann man diesen also noch variieren:

1. Wie kann man eine untere Schranke μ_Y berechnen?
2. Wie partitioniert man eine Teilmenge Y?
3. In welcher Reihenfolge werden die Y_i abgearbeitet?

Vorlesung vom 4.7.1996
Thorsten Oelgart

Beispiel Betrachten wir zur Verdeutlichung der Arbeitsweise des FINDMIN-Algorithmus eine Instanz des *Scheduling of Independent Tasks*-Problems mit zwei Maschinen $M1$ und $M2$ (SIT_2). Die Maschinen seien in der Notation zwecks Unterscheidung geordnet. Wir schreiben $(29,7)$ für: Maschine M_1 hat Aufgaben der Länge 29 bearbeitet, Maschine M_2 Aufgaben der Länge 7. Die einzelnen Aufgabenlängen seien wie folgt gegeben:

$$w_1 = 24, \quad w_2 = 5, \quad w_3 = 7, \quad w_4 = 17, \quad w_5 = 3, \quad w_6 = 15.$$

Die Menge X in unserem Algorithmus ist damit die Menge aller möglichen Zuweisungen:

$$X : \{1, \ldots, 6\} \to \{1, 2\}.$$

Eine Teilmenge $Y \subset X$ ist dann eine Teilmenge von Zuweisungen und für ein bestimmtes $f \in X$ ist $f(i)$, $i = 1, \ldots, k$ und $1 \leq k \leq 6$, festgelegt. Da alle Aufgaben zusammen $\sum_{i=1}^{n} w_n = 71$ Zeit benötigen, ist $36 = \lceil \frac{71}{2} \rceil$ eine obere Schranke für die Auslastung jeder Maschine. Man wählt deshalb

$$\mu_Y = \max\{36, a1, a2\},$$

wobei $a1$ die schon erzielte Auslastung der Maschine $M1$, $a2$ die von $M2$ ist.

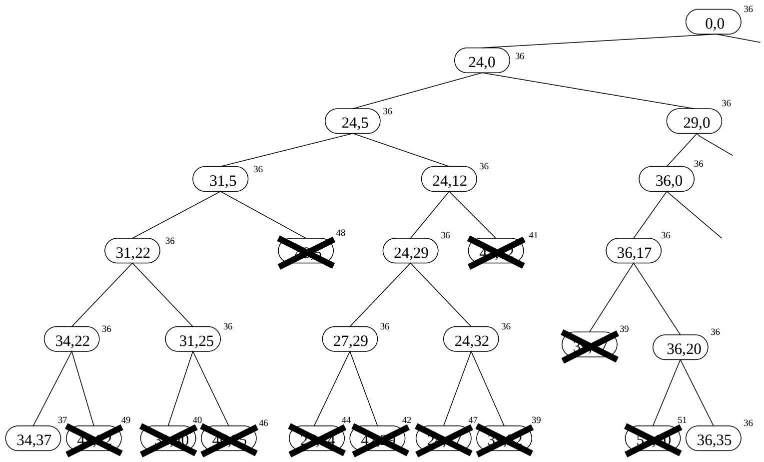


Abbildung 3.1: Finden der optimalen Lösung mit Hilfe der BB-Methode anhand des Beispiels SIT_2

In Abbildung 3.1 ist ein möglicher Baum dargestellt, den der BB-Algorithmus aufbaut (BB-Baum). Welcher Maschine er bei jedem Knoten des Baumes die neue Aufgabe zuteilt ist dabei willkürlich. Die globale Variable *currentmin* wird als Maximum der beiden Maschinenauslastungen in den tiefsten Blättern des Baumes berechnet, also erst wenn alle w_i verteilt sind. In unserem Beispiel wird sie also zuerst von ∞ (Initialisierungswert) auf 37 gesetzt (Baumdurchlauf via DFS). Nun greift die BB-Methode in den Fällen, in denen eine der Maschinenauslastungen über diesen Wert steigt (durch die jeweilig vorgenommene Zuteilung der Aufgaben). An diesen Knoten brauchen die darunterliegenden Teilbäume nicht weiter untersucht zu werden, da das Ergebnis bestenfalls gleichgut wäre. Das Optimum wird hier erreicht, wenn *currentmin*= 36 gesetzt wird.

Bei dem Beispiel durchquerte FINDMIN den BB-Baum in DFS-Art. Das muß nicht unbedingt die optimale Strategie sein. Man sieht, wie sich der Suchbaumes durch das BB-Verfahren verkleinert. Allerdings wird immer noch ein großer Teil des Aufzählungsbaums durchquert.

3.1.1 Allgemeine BB-Methode

Die Blätter des BB-Baumes bezeichnet man als *offen* wenn es bisher nicht ausgeschlossen werden kann, daßdurch ihre weitere Expansion eine optimale Lösung gefunden wird, ansonsten als *abgeschlossen*. Algorithmus:

wähle offenes Blatt l ("Branching Strategie")
produziere alle Kinder $l_1, \ldots, l_k,\ k > 1$ von l ("Partitions Strategie")
für jedes l_i berechne Schranke μ_i
  if $\mu_i \geq$ *currentmin* **then** schließe Blatt
  else if (l_i stellt vollständige Lösung dar) **then** *update currentmin*
  if *currentmin*$= \mu_\lambda$ für Vorfahren λ von Blatt l_i **then**
    schneide Teilbaum unter λ ab und schließe Knoten λ

Die Partitions-Strategie ist problemabhängig. Für die Branching-Strategie gibt es verschiedene Möglichkeiten:

- Wähle das offene Blatt mit dem größten μ.
- Wähle das offene Blatt mit dem kleinsten μ.
- Arbeite die offenen Blätter nach dem $LIFO$-Prinzip ab. Dies führt zu DFS im BB-Baum.
- Arbeite die offenen Blätter nach dem $FIFO$-Prinzip ab. Das führt zu BFS im BB-Baum.

Beispiel Wendet man auf unser obiges Beispiel diesen allgemeinen Algorithmus mit der Branching Strategie, immer das Blatt mit dem kleinsten μ zu wählen, an, so erhält man einen BB-Baum wie in Abbildung 3.2.

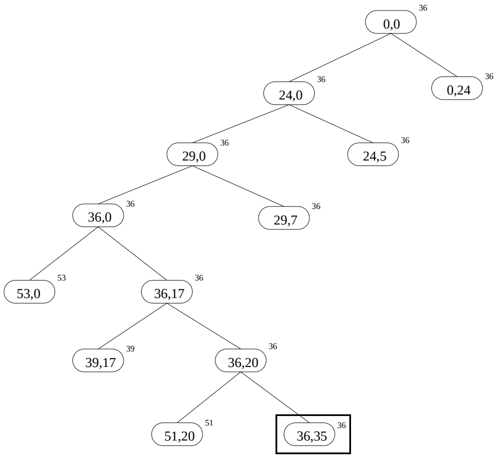


Abbildung 3.2: Die BB-Methode mit "minimales μ"-Branching Strategie

3.1.2 Branch and Bound für ganzzahlige Lineare Programme (ILP)

Die allgemeine Form eines ganzzahligen linearen Programmes ist:

$$\min c^T x, \quad Ax \leq b, \quad l \leq x \leq u, \quad x \in \mathbb{Z}^n,$$

wobei A, b und c ganzzahlig sind. Überführt man dies in ein mit Hilfe der BB-Methode lösbares Problem, so ergibt sich:

$$X = \{x \in \mathbb{Z}^n | Ax \leq b, l \leq x \leq u\} \quad \text{mit} \quad f(x) = c^T x.$$

Für $Y \subset X$ gilt $Y = \mathbb{Z}^n \cap P$, wobei P ein n-dimensionales Polytop ist, das sich aus den Einschränkungen durch die Ungleichungen ergibt. Die im Algorithmus notwendige Partitionierung von Y läßt sich durch das Hinzufügen von Ungleichungen der Art

$$a^T x \leq a_0 \quad \text{und} \quad a^T x \geq a_0 + 1$$

mit $a \in \mathbb{Z}^n$ und $a_0 \in \mathbb{Z}$ realisieren.
Typischerweise nimmt man für a einen Einheitsvektor wie in Abbildung 3.3. Den Wert für die untere Schranke μ erhält man, indem man das ILP zu einem LP relaxiert, also die Forderung der Ganzzahligkeit wegläßt. Man mußdeswegen nicht in jedem Schritt ein LP lösen, um das μ_i zu berechnen, da in jedem Schritt jeweils nur eine

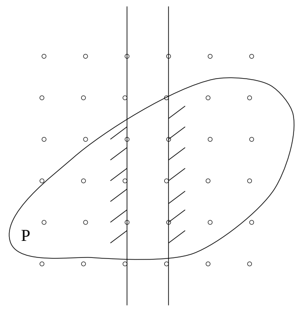


Abbildung 3.3: Durch Einheitsvektor partitionierter Raum.

Ungleichung hinzukommt. Das neue LP kann man mit Hilfe der alten Lösung schnell mit dem Dualen Simplex lösen.

Beispiel

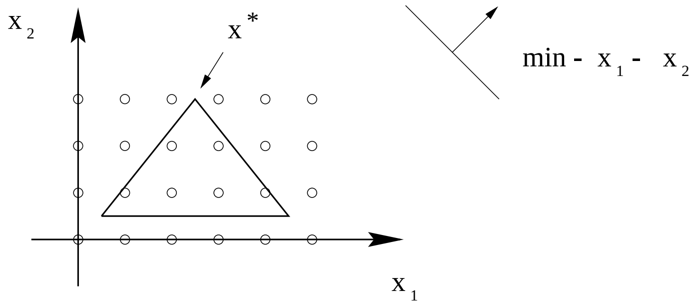


Abbildung 3.4: ILP in der Ebene

Sei P ein Dreieck in der Ebene (siehe Abbildung 3.4). Der Algorithmus zur BB-Methode sieht nun wie folgt aus:

> Zum Partitionieren nimmt man einen noch zu bestimmenden Einheitsvektor x_i.
> Wahl von x_i:
> - Sei x^* die optimale Lösung des relaxierten Problems.
> - Falls $x_j^* \in \mathbb{Z} \quad \forall j \longrightarrow$ fertig,
> - sonst
>   - wähle j mit $x_j^* \notin \mathbb{Z}$,
>   - $i = j$
>   - Partitioniere gemäß $x_i \leq \lfloor x_i^* \rfloor$ und $x_i \geq \lfloor x_i^* \rfloor + 1$.

Im Beispiel sieht das folgendermaßen aus (siehe Abbildung 3.5): Die optimale Lösung des relaxierten Problems ist die Spitze des Dreiecks. Für diesen optimalen Punkt gilt, daß die x_1-Koordinate keine ganze Zahl ist und zwischen 2 und 3 liegt. Wir unterteilen also zunächst den zulässigen Bereich in die Menge der Punkte mit $x_1 \leq 2$ und die Menge der Punkte mit $x_1 \geq 3$. Im zweiten Schritt wenden wir den Algorithmus zunächst auf das linke Teilproblem an, auf die Menge der zulässigen Punkte mit $x_1 \leq 2$. Der optimale Punkt ist hier die Spitze des linken schraffierten Dreiecks. Für diesen Punkt ist die x_1-Koordinate ganzzahlig aber $x_2 \notin \mathbb{Z}$ und $2 \leq x_2 \leq 3$. Daher partitionieren wir gemäß $x_2 \geq 3$ und $x_2 \leq 2$. Im dritten Schritt betrachten wir die Menge der zulässigen Lösungen mit $x_1 \leq 2$ und $x_2 \leq 2$. Der optimale Punkt des relaxierten Problems ist hier schon ganzzahlig, nämlich der Punkt $(2, 2)$. Damit haben wir dieses Teilproblem gelöst und müssen jetzt für die anderen Teilprobleme überprüfen, ob man die Lösung noch verbessern kann.
Der Algorithmus terminiert, weil bei jedem Durchlauf der Lösungsraum kleiner wird.

Vorlesung vom 8.7.1996
Daniel Kästner, Daniel Kreutzer

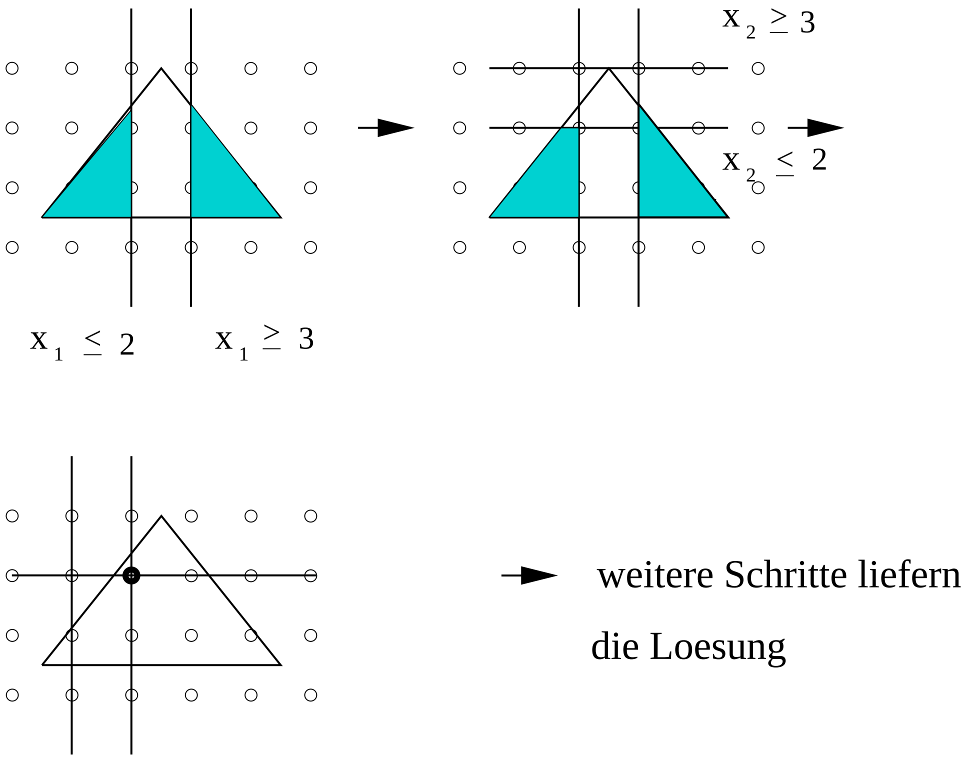


Abbildung 3.5: Branch and Bound

Bei der Anwendung der Branch & Bound-Strategie zum Lösen ganzzahliger linearer Programme $\max c^T x$, $Ax = b$, $x \geq 0$, x ganzzahlig, kann es zu dem Problem kommen, daß der zulässige Bereich des relaxierten Problems ein Streifen ist, der keinen ganzzahligen Gitterpunkt enthält, siehe Abbildung 3.6. In diesem Fall gibt es keine Lösung für das ILP. Es stellt sich die Frage, ob man eine Aussage treffen kann, wann ein Gitterpunkt im zulässigen Bereich liegt.
Unser Ziel ist es nun, eine Aussage der folgenden Form zu treffen: Falls das ILP eine ganzzahlige Lösung hat, dann gibt es solch eine Lösung $\widehat{x}$ mit $\widehat{x} \leq M(A, b)$, d.h. dann kann man die Lösung gegen eine obere Konstante abschätzen.

Bei der Ellipsoidmethode haben wir bereits gesehen, daß man zulässige Lösungen mit Hilfe der Cramerschen Regel und der Darstellung von Determinanten als Summe von Permutationen nach oben durch eine ganze Zahl abschätzen kann. Das liefert uns die Aussage des folgenden Lemmas:

Lemma 11 *Wenn $Ax = b$ mit $x \geq 0$ eine rationale Lösung hat, dann gibt es eine Lösung $\widehat{x}$ mit*

$$\widehat{x} \leq m!\alpha^{m-1}\beta \qquad (\ x_i = \frac{y_i}{D},\ D \leq m!\alpha^m\ ),$$

wobei $A \in \{-\alpha, \ldots, \alpha\}^{m \times n}$ und $b \in \{-\beta, \ldots, \beta\}^m$. Es ist also α der größte Absolutbetrag in A und β der größte Absolutbetrag in b.

Lemma 12 *Gegeben seien die k Vektoren $v_1, \ldots, v_k \in \{-\alpha, \ldots, \alpha\}^m$, sowie die Schranke $M_1 = (m\alpha)^{m+1}$. Dann sind die folgenden Aussagen äquivalent:*

1. *Es existiert ein $\lambda = (\lambda_1, \ldots, \lambda_k) \in \mathbb{R}^k$, $\lambda \neq 0$, $\lambda \geq 0$ mit $\sum_{i=1}^k \lambda_i v_i = 0$.*
2. *Es existiert ein $\lambda = (\lambda_1, \ldots, \lambda_k) \in \{0, \ldots, M_1\}^k$, $\lambda \neq 0$, $\lambda \geq 0$ mit $\sum_{i=1}^k \lambda_i v_i = 0$.*
3. *Es gibt kein $h \in \mathbb{R}^m$, so daß $h^T v_i > 0$ für $i = 1, \ldots, k$.*
4. *Es gibt kein $h \in \{-M_1, \ldots, M_1\}^m$, so daß $h^T v_i \geq 1$ für $i = 1, \ldots, k$.*

Die Äquivalenz der Aussagen 1) und 3) ist bekannt als *Farkas Lemma*. In den Abbildungen 3.7 und 3.8:
Beweis (durch Ringschluß)

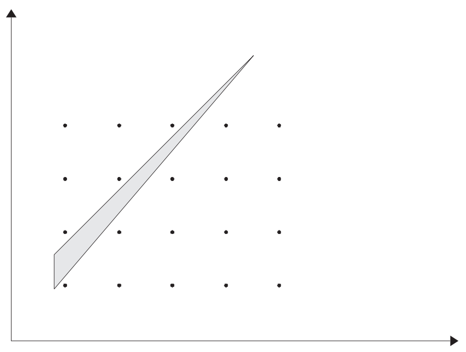


Abbildung 3.6: ILP hat keine Lösung

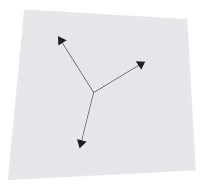


Abbildung 3.7: Die Vektoren lassen sich nicht auf einen Halbraum einschränken; die Aussagen 1) und 3) gelten

1. $\Rightarrow$ 2. folgt direkt aus Lemma 11.

2. $\Rightarrow$ 3. Aus 2. folgt: Es existiert ein $\lambda \geq 0$, $\lambda \neq 0$ mit $\sum_i \lambda_i v_i = 0$. Angenommen, 3. gilt nicht. Dann gibt es also ein h mit $h^T v_i > 0$ für alle i. Daraus folgt:

$$\underbrace{h^T 0}_{=0} = h^T (\sum_i \lambda_i v_i) = \sum_i \lambda_i \underbrace{h^T v_i}_{>0}.$$

Wir haben damit einen Widerspruch hergeleitet. Also muß aus 2. direkt 3. folgen.

3. $\Rightarrow$ 4. Wenn es keine reelle Lösung gibt, dann gibt es auch keine ganzzahlige!

4. $\Rightarrow$ 1. Betrachte dazu folgendes lineares Programm:

$$\min h^T 0 \qquad h^T v_i \geq 1 \quad \text{für alle } i.$$

Wenn es eine Lösung für dieses Lineare Programm gibt, dann gibt es eine Lösung $\widehat{h}$ mit folgenden Eigenschaften: $\widehat{h}_i = \frac{g_i}{D}$ mit $0 \leq g_i \leq M_1$ und $1 \leq D \leq M_1$. Daraus folgt

$$(\underbrace{D\widehat{h}}_{=g})^T v_i \geq D \cdot 1 \geq 1 \quad \forall i.$$

Da 4. gilt, gibt es kein solches h, d.h. das lineare Programm ist unlösbar. Daraus folgt, daß das zugehörige duale lineare Programm entweder unlösbar oder unbeschränkt ist. Das duale Lineare Programm hat die Form:

$$\max \sum_{i=1}^{k} \lambda_i, \qquad \sum_{i=1}^{k} \lambda_i v_i = 0, \qquad \lambda_i \geq 0.$$

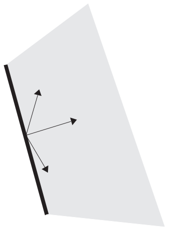


Abbildung 3.8: Alle Vektoren zeigen in einen Halbraum; die Aussagen 1) und 3) gelten nicht

Offensichtlich ist $\lambda = 0$ eine Lösung des dualen Programms. Es ist also nicht unzulässig, und muß daher unbeschränkt sein. Dann gibt es aber auch eine nichtnegative Lösung λ.

$\square$

Satz 20 *Wenn das ILP eine ganzzahlige Lösung hat, dann gibt es eine solche Lösung x mit $x \in \{0, 1, \ldots, M\}^n$, wobei $M = n \cdot (m\alpha)^{2m+3} \cdot (1 + \beta)$ ist.*

Beweis Sei x eine Lösung des ILP, für die $\sum_{i=1}^n x_i$ minimal ist. Dann müssen wir 2 Fälle unterscheiden:

1. $x_i \le \underbrace{(m\alpha)^{m+1}}_{=:M_1} < M$ für alle i. Die Aussage des Satzes ist somit trivial.

2. Es gibt mindestens ein i mit $x_i > M_1$. Ohne Einschränkung seien $x_1, \ldots, x_k > M_1$ und $x_{k+1}, \ldots, x_n \le M_1$. Seien weiterhin $v_1, \ldots, v_k$ die ersten k Spalten von A. Es sind nun wieder 2 Fälle zu betrachten:

   (a) Es existiert ein $\lambda \in \{0, \ldots, M_1\}^k$ mit $\lambda > 0$ und $\sum_{i=1}^k \lambda_i v_i = 0$. Dann ist aber auch
   $$(x_1 - \lambda_1, \ldots, x_k - \lambda_k, x_{k+1}, \ldots, x_n)^T$$
   eine Lösung des ILP. Diese ist kleiner als die ursprüngliche, was einen Widerspruch zur Annnahme darstellt, x sei die kleinste Lösung.

   (b) Ein solches λ existiert nicht. Dann wissen wir nach Lemma 12, daß es ein $h \in \{-M_1, \ldots, M_1\}^m$ gibt mit $h^T v_i \ge 1$ für alle $i = 1, \ldots, k$. x ist eine Lösung für das ILP, also gilt $Ax = b$ und daraus folgt $h^T Ax = h^T b$, bzw.
   $$\sum_{i=1}^k h^T v_i x_i + \sum_{i=k+1}^n h^T v_i x_i = h^T b.$$
   Daraus kann man folgende Abschätzung erhalten
   $$\sum_{i=1}^k x_i \;\le\; \sum_{i=1}^k \underbrace{h^T v_i}_{\ge 1} x_i \;\le\; \underbrace{h^T b}_{\le m M_1 \beta} - \sum_{i=k+1}^n \overbrace{h^T v_i}^{\le m M_1 \alpha} \underbrace{x_i}_{\le M_1}.$$
   Damit erhalten wir schließlich
   $$x_i \;\le\; \sum_{i=1}^k x_i \;\le\; m M_1 \beta + (n-k) m M_1^2 \alpha \;\le\; M.$$

$\square$

Betrachte nun ein ganzzahliges Lineares Programm ILP mit der zugehörigen Relaxation LP:

$$\begin{array}{rr} \min & c^T x \\ & Ax = b \\ & x \ge 0 \\ & x \in \mathbb{Z} \end{array} \qquad\qquad \begin{array}{rr} \min & c^T x \\ & Ax = b \\ & x \ge 0 \end{array}$$

Satz 21 *Ist das relaxierte Problem LP unbeschränkt und hat das ganzzahlige Problem ILP eine Lösung, dann ist auch das ILP unbeschränkt.*

Satz 22 *Wenn das ILP eine endliche optimale Lösung besitzt, dann hat es eine optimale Lösung x^* mit*

$$x^* \leq n^3((m+2)\gamma)^{4m+12}.$$

Dabei ist $\gamma = max_{i,j}\{|a_{ij}|, |b_{ij}|, |c_{ij}|\}$.

3.1.3 "Cutting Plane" - Algorithmen

Wir wollen nun versuchen, von der Lösung des relaxierten Problems ausgehend, eine optimale Lösung des ganzzahligen Problems zu finden. Es kann dabei der Fall auftreten, daß x^*_{ILP} sehr weit von x^*_{LP} entfernt ist, so daß eine Nachbarschaftssuche nach der optimalen Lösung des ganzzahligen linearen Programmes nicht erfolgversprechend ist. Können wir aber nun den Zulässigkeitsbereich des relaxierten Problems so einschränken, daß wir solche Bereiche, von denen wir wissen, daß sie keine ganzzahligen Lösungen enthalten können, einfach wegfallen lassen?
Aus dieser Fragestellung ergibt sich eine Idee für einen Algorithmus:

Grundschema eines Cutting Plane-Algorithmus Sei P der Zulässigkeitsbereich des relaxierten linearen Programms LP.

Führe zusätzliche Bedingungen (Schnittebenen, Cutting Planes) ein, die P auf $\widehat{P}$ verkleinern, so daß kein ganzzahliger Punkt in $P - \widehat{P}$ enthalten ist und der optimale Punkt in $\widehat{P}$ ganzzahlig ist.

Betrachte hierzu die Abbildungen 3.9, 3.10 und 3.11.

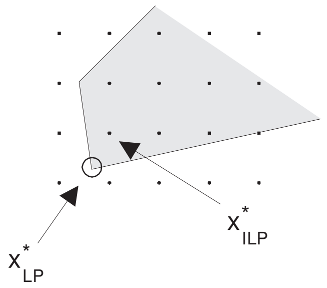


Abbildung 3.9: Anfangssituation

Zur Bestimmung solcher Schnittebenen gibt es einmal die Möglichkeit, sich selbst intuitiv neue Ungleichungen zu überlegen, aber glücklicherweise gibt es auch verschiedene automatische Verfahren.

1. **Gomory Cuts**

   Sei gegeben ein optimales Wörterbuch

$$\begin{array}{rcl}
x_i &=& \overline{b_i} - \sum_{j \notin B} \overline{a_j} x_j \qquad \forall i \in B \\
\hline
z &=& v - \sum_{j \notin B} \overline{c_j} x_j \\
\overline{b_i} &\geq& 0
\end{array}$$

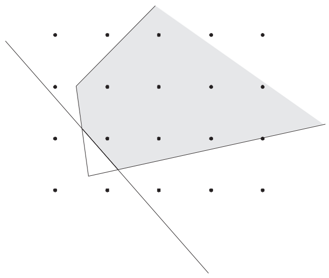


Abbildung 3.10: Hinzufügen einer zusätzlichen Ungleichung

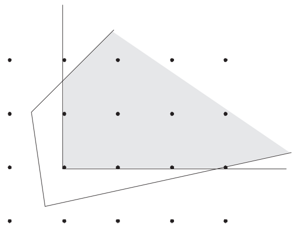


Abbildung 3.11: Zielsituation

Für die sich daraus ergebende optimale Basislösung x gilt:

$$\begin{array}{llll} x_i & = \overline{b_i} & & i \in B \\ x_i & = 0 & & i \notin B \end{array}$$

Wir betrachten nun 2 Fälle:

(a) Gilt $\overline{b_i} \in \mathbb{Z}$ für alle $i \in B$, dann gibt es nichts mehr zu tun.

(b) Es gebe ein i^*, so daß $\overline{b_{i^*}} \notin \mathbb{Z}$. Dann gilt

$$x_{i^*} \; + \; \sum_{j \notin B} \overline{a_j} x_j \;\; = \;\; \overline{b_{i^*}} \notin \mathbb{Z}.$$

Der Nachkommateil von $\overline{b_{i^*}}$ kann sich hierbei jedoch nur als Summe der Nachkommateile der $\overline{a_j}$ ergeben. Wegen $x_j \geq 0$ gilt dann

$$\sum_{j \notin B} \overline{a_j} x_j \;\; \geq \;\; \sum_{j \notin B} \lfloor \overline{a_j} \rfloor x_j.$$

Damit ergibt sich

$$\underbrace{x_{i^*}}_{\in\mathbb{Z}} + \underbrace{\sum_{j\notin B} \lfloor\overline{a_j}\rfloor x_j}_{\in\mathbb{Z}} \;\le\; x_{i^*} + \sum_{j\notin B} \overline{a_j} x_j \;=\; \overline{b_{i^*}}.$$

Da die linke Seite also ganzzahlig ist, können wir in der obigen Gleichung $\overline{b_{i^*}}$ durch $\lfloor\overline{b_{i^*}}\rfloor$ ersetzen und erhalten damit

$$x_{i^*} + \sum_{j\notin B} \lfloor\overline{a_j}\rfloor x_j \;\le \lfloor\overline{b_{i^*}}\rfloor.$$

Auf diese Weise erhalten wir eine völlig neue Ungleichung für unser Problem:

$$\sum_{j\notin B} (\overline{a_j} - \lfloor\overline{a_j}\rfloor) x_j \;\ge \overline{b_{i^*}} - \lfloor\overline{b_{i^*}}\rfloor.$$

Wir führen nun eine neue Notation ein:

$$\{z\} =: z - \lfloor z\rfloor.$$

Damit erhalten wir als neue Gleichung beim *Gomory Cut*

$$\sum_{j\notin B} \{\overline{a_j}\} x_j \;\ge\; \{\overline{b_{i^*}}\}.$$

Vorlesung vom 11.7.1996
Henrik Stormer

Diese Ungleichung fügen wir nun zu dem ursprünglichen Wörterbuch hinzu und erhalten:

$$\begin{array}{rcrcl} S & = & -\{\bar{b}_i\} & + & \sum_{j\notin B} \{\bar{a}_{ij}\} x_j \\ x_k & = & \bar{b}_k & - & \sum_{j\notin B} \bar{a}_{kj} x_j \\ \hline z & = & v & - & \sum_{j\notin B} \bar{c}_j x_j. \end{array}$$

Dieses liefert eine dual zulässige Lösung des Problems, und wir können mit den Schritten des dualen Simplex–Algorithmus fortfahren. Zusammenfassend erhalten wir damit den folgenden Algorithmus zu Lösung eines ILP.

Algorithmus NACHKOMMA–DUALER–SIMPLEXALGORITHMUS FÜR ILP

1. Löse Relaxation des ILP
   - LP unzulässig $\longrightarrow$ ILP unzulässig
   - LP unbeschränkt $\longrightarrow$ füge Einschränkungen $x_i \le M$ hinzu (Satz 20) und beginne neu
   - LP beschränkt $\longrightarrow$ sei x^* die optimale Lösung mit Wörterbuch
2. Solange x^* nicht ganzzahlig
   (a) Wähle Zeile aus Wörterbuch mit $b_i^* \notin \mathbb{Z}$ und erzeuge Gomory Schnitt. Füge neue Gleichung mit Schlupfvariable oben zum Wörterbuch hinzu.
   (b) Führe dualen Simplex aus.
   - duales LP unbeschränkt $\longrightarrow$ primal unzulässig $\longrightarrow$ ILP unzulässig
   - beschränkt $\longrightarrow$ x^* ist neue optimale Lösung mit Wörterbuch
3. return x^*

Als letztes muß jetzt noch die Frage geklärt werden, ob der Algorithmus wirklich terminiert. Es könnte ja auch sein, daß er in Punkt 2 in eine Endlosschleife gerät, weil keine ganzzahlige Lösung gefunden wird.

Satz 23 *Dieser Algorithmus terminiert, wenn bei (a) immer die unterste mögliche Zeile gewählt wird und bei (b) im Simplexverfahren die lexikographische Methode zur Wahl der neuen Basisvariablen verwendet wird.*

Diesen Satz werden wir an dieser Stelle nicht beweisen. Es bleibt damit also nur noch zu klären, wie man mit Hilfe der lexikographischen Methode Basisvariable auswählt. Dies wird anhand eines Beispiels erläutert. Sei folgendes lineare Programm gegeben, für welches hier schon das Wörterbuch aufgeschrieben ist:

$$\begin{array}{rcrcrcrcrcr}
x_1 &=& -5 &+& 2x_4 &-& x_5 &+& x_6 &+& 3x_7 \\
x_2 &=& 3 &+& 8x_4 &+& 3x_5 &+& 5x_6 &+& 9x_7 \\
x_3 &=& 1 &+& 6x_4 &+& 7x_5 &+& 4x_6 &+& 6x_7 \\
\hline
z &=& 12 &+& 4x_4 &+& 5x_5 &+& 2x_6 &+& 6x_7
\end{array}$$

Dabei soll z maximiert werden. Der Wert von z kann noch durch die Veränderung der 1. Zeile vergrößert werden. Dazu kann man beispielsweise das Zweifache der ersten Zeile von der letzten subtrahieren. Dies sieht dann folgendermaßen aus:

$$z - 2x_1 = 22 \quad + \quad 0x_4 \quad + \quad 7x_5 \quad + \quad 0x_6 \quad + \quad 0x_7.$$

Als Eintrittsvariable wird diejenige Variable ausgewählt, deren Koeffizient in dieser Gleichung Null ist. In diesem Fall hat man mehrere Möglichkeiten zur Auswahl, nämlich x_4, x_6 und x_7. Mit Hilfe der lexikographischen Ordnung wird jetzt bestimmt, welche Variable ausgewählt werden soll. Dazu betrachten wir die 3. Zeile und führen dieselbe Operation mit dieser aus. Dann ergibt sich:

$$x_3 - 2x_1 = 11 \quad + \quad 2x_4 \quad + \quad 9x_5 \quad + \quad 2x_6 \quad + \quad 0x_7.$$

Würden auch hier mehrere Koeffizienten 0 auftauchen, so müßten wir erneut eine Zeile auswählen und von dieser ein Vielfaches von x_1 subtrahieren. Dieses Verfahren muß terminieren, da ansonsten zwei Spalten des Wörterbuchs gänzlich übereinstimmen würden.