Höhere Algorithmik II

Dozent: Prof. Dr. Helmut Alt

Sommersemester 2008

Inhaltsverzeichnis

Kapitel 1

Lineare Programmierung

1.1 Beispiele

Beispiel. 1

Eine Molkereigesellschaft hat zwei zentrale Sammelstellen. Eine erhält $15m^3$, die andere $20m^3$ Milch pro Tag. Es gibt drei Weiterverarbeitungsanlagen die jeweils 10, 15 und $20m^3$ Milch pro Tag verarbeiten können.

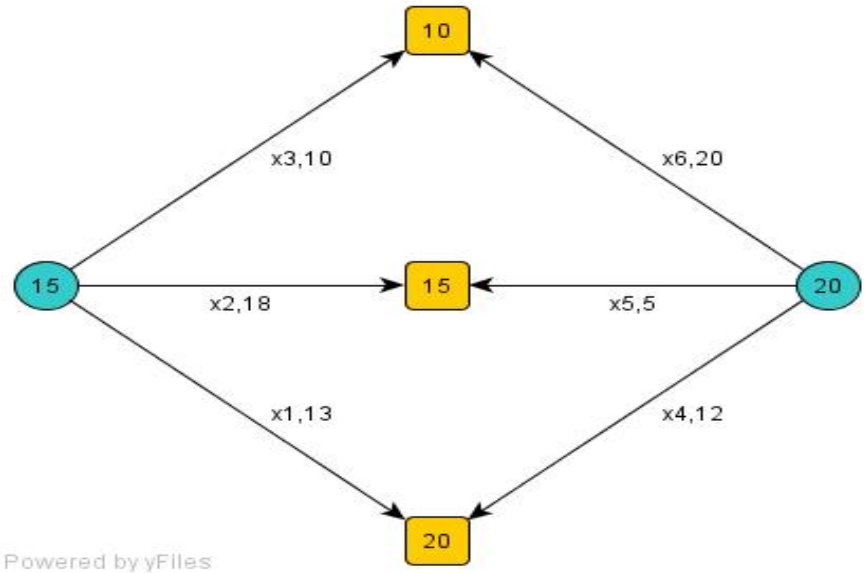


Abbildung 1.1: Kreise=Sammelstelle, Quadrate=Weiterverarbeitungsanlage

Die Milch soll nun so verteilt werden, dass die Transportkosten $(euro/m^3)$ minimiert werden. Der Ausdruck $13x_1 + 18x_2 + 10x_3 + 12x_4 + 5x_5 + 20x_6$ muss

also minimiert werden.
Dabei gilt es jedoch folgende, durch die Aufgabenstellung definierte, Nebenbedingungen einzuhalten:
$x_1 + x_2 + x_3 = 15$
$x_4 + x_5 + x_6 = 20$
$x_3 + x_6 \leq 10$
$x_2 + x_5 \leq 15$
$x_1 + x_4 \leq 20$
$x_i \geq 0, i = 1...6$

Solch eine Aufgabenstellung der Form
„minimiere eine lineare Funktion unter Einhaltung linearer Nebenbedingungen (Gleichungen und Ungleichungen)" heisst Lineares Programm.

Allgemein:
minimiere (bzw. maximiere) $f(x_1, ..., x_n) = c_1x_1 + ... + c_nx_n$
unter Nebenbedingungen:

$$\begin{array}{rcccccc}
a_{11}x_1 & + & ... & + & a_{1n}x_n & \leq & b_1 \\
. & . & . & . & . & . & . \\
. & . & . & . & . & . & . \\
a_{h1}x_1 & + & ... & + & a_{hn}x_n & \leq & b_h \\
a_{h+1,1}x_1 & + & ... & + & a_{hn}x_n & \geq & b_{h+1} \\
. & . & . & . & . & . & . \\
. & . & . & . & . & . & . \\
a_{j1}x_1 & + & ... & + & a_{jn}x_n & \geq & b_j \\
a_{l+1,1}x_1 & + & ... & + & a_{l+1,n}x_n & = & b_{l+1} \\
. & . & . & . & . & . & . \\
. & . & . & . & . & . & . \\
a_{m1}x_1 & + & ... & + & a_{mn}x_n & = & b_m
\end{array}$$

und Vorzeichenbedingungen $x_1 \geq 0, ..., x_d \geq 0, x_e \leq 0, ..., x_f \leq 0$

Die Aufgabenstellung kann leicht auf die kanonische Form gebracht werden:
minimiere $c^T * x$ (NICHT maximiere)
unter Nebenbedingung $A * x \leq b$
und Vorzeichenbedingung $x \geq 0$ (komponentenweise)

oder auch in die Standardform überführt werden:
minimiere $c^T * x$
unter Nebenbedingung $A * x = b$
und Vorzeichenbedingung $x \geq 0$

Umformung von allgemeiner Form in Standardform:
ersetze Nebenbedingung $a_{i1}x_1 + .. + a_{in}x_n \leq b$ ($bzw. \geq$)

durch $a_{i1}x_1 + ... + a_{in}x_n + s_i = b$
(s_i heisst slack variable, dt. Schlupfvariable)

sorge dafür, dass alle Vorzeichenbedingungen ein $\geq$ enthalten:
Falls keine VzB auf x_i besteht, dann ersetze jedes Vorkommen von x_i durch $x_i^+ - x_i^-$ und füge VzB $x_i^+ \geq 0$ und $x_i^- \geq 0$ ein.
Falls VzB $x_i \leq 0$ existiert, so ersetze überall x_i durch $-x_i$ und änder VzB auf $x_i \geq 0$

Beispiel. 2

Fragestellung wie im ersten Beispiel nur dass die Milch nun nur noch in Containern von $1m^3$ transportiert werden kann
d.h. statt einer optimalen Lösung $x^* \in \mathbb{R}^n$ (in polynomieller Zeit berechenbar)
suchen wir die optimale Lösung $x^* \in \mathbb{Z}^n$ (ganzzahlige LP, NP-vollständig)

Ganzzahlige Programmierung tritt in vielfacher Form in der Praxis auf:
z.B. bei Transportproblemen, Mischproblemen, Lagerhaltung etc.

Algorithmen:

1. Simplex-Methode (Danzig 1949)
   - benötigt im schlechtesten Fall exponentielle Laufzeit
   - zeigt aber im Mittel gut Laufzeit
   - ist die am häufigsten benutzte Methode
2. Ellipsoidmethode (Khachijan)
   - polynomielle Laufzeit
3. innere-Punkt-Methode (1984, Kharmarkar)
   - polynomielle Laufzeit
   - praktisch anwendbar

1.2 Geometrie des LP

Definition 1.2.1. Sei ein LP gegeben. Ein Vektor x der alle NB erfüllt heißt zulässige (feasible) Lösung. Die Menge aller zulässigen Lösungen heißt zulässiger Bereich.

Wie sieht solch ein zulässiger Bereich aus?

Sei Nb: $2x_1 + x_2 \leq 4$
und Vzb: $x_1 \geq 0, x_2 \geq 0$

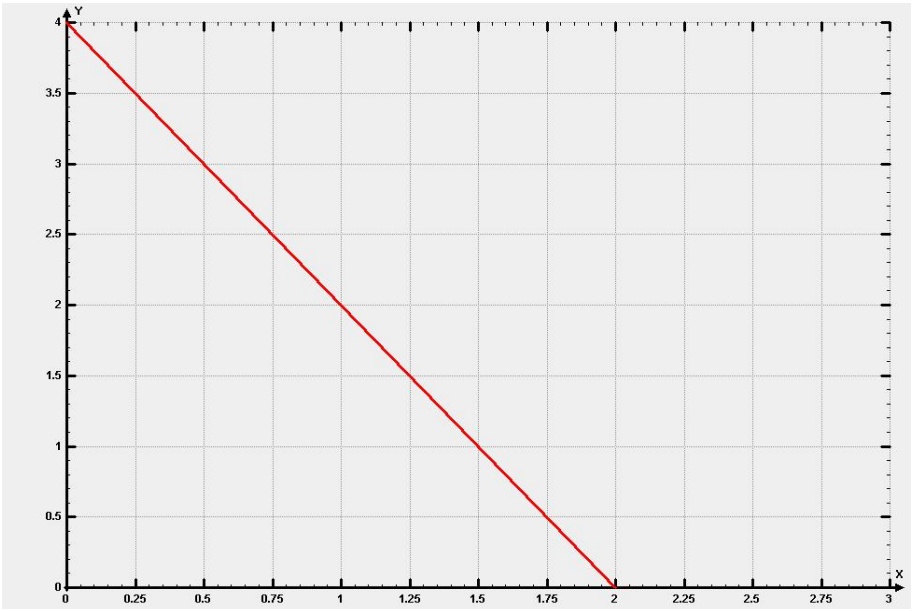


Abbildung 1.2: $2x_1 + x_2 \leq 4$,$x_1 \geq 0, x_2 \geq 0$ Zulässiger Bereich wird von den beiden Achsen und der Geraden umfasst

dann sieht der zulässige Bereich wie in Abbildung 2 aus:

Allgemein:

Definition 1.2.2. Eine Menge der Form $\{(x_1...x_n)|a_1x_1 + ... + a_nx_n = b\}$ heißt Hyperebene $(\in \mathbb{R}^n)$ Eine Menge der Form $\{(x_1...x_n)|a_1x_1 + ... + a_nx_n \leq b\}$ heißt Halbraum $(\in \mathbb{R}^n)$

Ein zulässiger Bereich ist der Durchschnitt von endlich vielen Halbräumen. Solch eine Menge heißt konvexes Polyeder.

Ein Punkt v eines konvexen Polyeders heißt Extrempunkt (oder Ecke) gdw. er NICHT als strikte Konvexkombination zweier Punkte $x, y \in P$ dargestellt werden kann.
D.h.: $v = \lambda x + (1 - \lambda) \cdot y$ mit $\lambda \in (0, 1)$

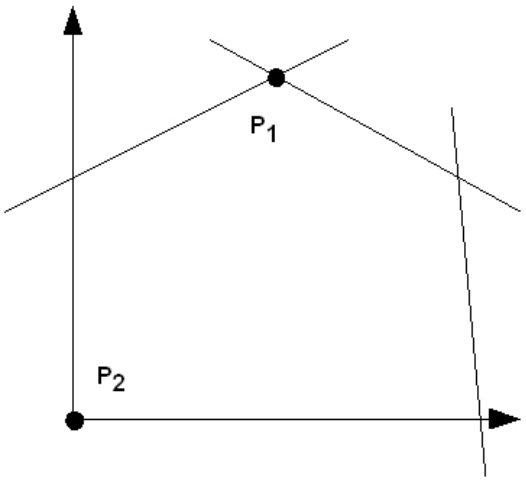


Abbildung 1.3: Ecken

Lemma 1.2.3. *Sei $P = \{x \in \mathbb{R}^n | Ax \leq b\}$ ein konvexes Polyeder mit $A \in \mathbb{R}^{n \times m}$. Dann gilt: v ist Ecke von $P \Leftrightarrow$ es gibt n linear unabhängige Nebenbedingungen in $Ax \leq b$ für die bei v Gleichheit gilt.*

Beispiel. (siehe Abbildung 1.3) Für p_1 wird die Gleichheit für zwei Nebenbedinungen erfüllt, für p_2 durch die beiden Vorzeichenbedingungen $x_1 = 0$ und $x_2 = 0$.

Beweis des Lemma. „$\Rightarrow$“: Sei $J = \{j_1, \ldots, j_k\}$ die Menge der Indizes der Nebenbedingungen (Zeilenvektoren von A), für die die Gleichheit gilt.

Falls $k < n$, nimm einen Vektor $t \neq \overrightarrow{0}$ (Nullvektor) mit $a_{j_i} t = 0$ und $i = 1, \ldots, k$. a_{j_i} ist j_i-te Zeile von A und t liegt im Orthogonalraum von $a_{j_1}, \ldots, a_{j_k}$. Dann gilt für $v^+ = v + \gamma t$ und $v^- = v - \gamma t$:

$$a_l v^{\pm} = a_l v \pm \gamma a_l t = a_l v$$

falls $l \in J$, da $a_l t = 0$. Auf Grund der oben genannten Gleichheit ($a_l v = b$) gilt dann ebenfalls:

$$a_l v^{\pm} = b_l$$

Falls jedoch $l \notin J$, dann gilt $a_l v < b_l$. Wird γ hinreichend klein gewählt, dann ist:

$$a_l v \pm \gamma a_l t = a_l v^{\pm} < b_l,$$

D.h.: Da $v^+, v^- \in P$ die Nebenbedingungen erfüllen und $v = \frac{1}{2}v^+ + \frac{1}{2}v^-$ ist, kann v keine Ecke sein. Dies steht jedoch im Widerspruch zu der oben gemachten Annahme und damit ist „$\Rightarrow$“ gezeigt.

„$\Leftarrow$“: Angenommen, v ist keine Ecke von P, dann existieren:

$$v_1, v_2 \in P, v_1 \neq v_2, \lambda \in (0,1) \text{ mit } v = \lambda v_1 + (1 - \lambda) v_2$$

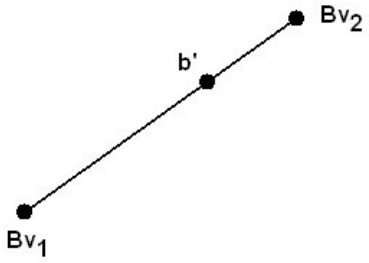


Abbildung 1.4: Konvexkombination von Bv_1 und Bv_2

Sei B eine Teilmatrix von A, welche aus jenen Zeilen besteht, für die für v die Gleichheit gilt:

$$B = \begin{pmatrix} a_{j_1} \\ \vdots \\ a_{j_n} \end{pmatrix}, B \in \mathbb{R}^{n \times n} \text{ und B regulär}$$

$$b' = \begin{pmatrix} bj_1 \\ \vdots \\ bj_k \end{pmatrix} = Bv = \lambda Bv_1 + (1-\lambda)Bv_2$$

Da b' eine Konvexkombination (siehe Abbildung 1.4) aus Bv_1 und Bv_2 ist und nach der Annahme $Bv_1 \leq b'$ und $Bv_2 \leq b'$ sind, muss gelten:

$$Bv_1 = b' \text{ und } Bv_2 = b'.$$

D.h. jedoch, da B regulär und damit invertierbar, dass ebenfalls gilt:

$$v_1 = B^{-1}b' \text{ und } v_2 = B^{-1}b'$$

und damit aber auch $v_1 = v_2$. Dies ist jedoch ein Widerspruch zur Annahme und damit ist „ $\Leftarrow$ “ gezeigt. □

Die Ecken von P entsprechen genau den Lösungen, für die bei v Gleicheit gilt. Ein solcher Punkt muss auf mindestens n Ebenen liegen, welche paarweise unabhängig sind.

Definition 1.2.4. Mengen von Punkten die k lineare unabhängige Nebenbedingungen mit Gleichheit erfüllen heißen $(n-k)$-*dimensionale* Facetten von P:

$$F_{i_1,\ldots,i_k} = \{x \in P | a_{i_1}x = b_{i_1}, \ldots, a_{i_k}x = b_{i_k}\}$$

Beispiel. Für $n = 3$:

- 0-dimensionale Facetten: Ecken
- 1-dimensionale Facetten: Kanten

- 2-dimensionale Facetten: Seitenflächen
- 3-dimensionale Facetten: gesamter Polyeder

Betrachtet man ein lineares Programm in Standardform, also mit den Eigenschaften

- minimiere $c^T x$ unte den
- Nebenbedingungen: $Ax = b$ und den
- Vorzeichenbedingungen: $x \geq 0$,

dann gelten die folgenden Eigenschaften:

Durch Hinzuhname von Schlupfvariablen wird die Dimension des untersuchten Raums erhöht.

Es kann davon ausgegangen werden, dass die Zeilenvektoren von A linear unabhängig sind, da andernfalls keine Lösung existiert oder Redundanz vorliegt und einige Nebenbedingungen gestrichen werden können.

A ist eine $m \times n$-Matrix vom Rang m und besitzt damit m unabhängige Zeilenvektoren. Damit besitzt A auch mindestens m linear unabhängige Spaltenvektoren. Jede solche Auswahl von unabhängigen Spaltenvektoren bildet eine *Basis* des Bildraums von linearen Abbildungen der Form $x \rightarrow Ax$ und spannt damit den Bildraum auf.

Die Spalten $j_1 < \ldots < j_m$ mit $j_i \in J$ bilden eine Basis und B sei eine $m \times m$-Matrix, welche aus diesen Spalten besteht. Dann ist B regulär und invertierbar.

Sei $a := B^{-1}b \in \mathbb{R}^m$, dann erfüllt $x \in \mathbb{R}^n$ mit

$$x_{j_i} = a_i \text{ für } i = 1, \ldots, m \text{ und}$$

$$x_l = 0 \text{ für } l \in J \quad (1)$$

die Nebenbedingungen des linearen Programms (2), denn

$$Ax = A \begin{pmatrix} 0 \\ \vdots \\ 0 \\ a_{j_1} \\ 0 \\ \vdots \\ 0 \\ a_{j_2} \\ 0 \\ \cdots \\ 0 \\ a_{j_m} \\ 0 \\ \vdots \\ 0 \end{pmatrix} = Ba = b$$

.

x heißt *Basislösung* des linearen Programms und ist der einzige Vektor mit den Eigenschaften (1) und (2). Insgesamt können, durch verschiedene Auswahlen der Spaltenvektoren, $\binom{n}{k}$ Basislösungen existieren. Erfüllt x auch die Vorzeichenbedingungen, dann wird x eine *zulässige Basislösung*, oder kurz *bfs* (basic feasible solution) genannt.

Korollar 1.2.5. *Es existieren höchstens* $\binom{n}{k}$ *Ecken.*

Beispiel. Für einen Würfel ist $n = 6$ (drei Variablen durch die Dimension und drei weitere durch Schlupfvariablen) und $m = 3$ (Vorzeichenbedingungen). Also ist die maximale Ecken-Anzahl eines Würfels höchstens $\binom{6}{3} = 20$.

Proposition 1.2.6. *$w \in \mathbb{R}^n$ ist zulässige Basislösung eines linearen Programms $\Leftrightarrow$ w ist eine Ecke des Polyeders F der zulässigen Lösung.*

Proof. „$\Leftarrow$“: wenn w eine Ecke des Polyeders, dann ist w auch eine zulässige Lösung.

Sei $I = \{i | w_i > 0\}$.

Fall 1) $E = \{A_j | j \in I\}$ sind linear unabhängig. A_i ist der i-te Spaltenvektor von A. Dann kann E zu einer Basis C (mit den Indizes $J \supset I$) von Spaltenvektoren erweitert werden und es ist $w_l = 0$ für $l \in J$. Dann ist w offensichtlich auch eine zulässige Lösung (siehe Definition).

Fall 2) E ist linear abhängig, dann existiert ein Vektor z mit der Eigenschaft $\sum_{j \in I} z_j A_j = 0$. Für alle $j \in J$ sei $z_j = 0$, dann ist $Az = 0$. Für hinreichend kleines $\theta > 0$ gilt:

$$w_j \pm \theta z_j \geq 0 \text{ für } j = 1, \ldots, n$$

Da $w_j > 0$ für $j \in I$ und $z_j = 0$ für $j \ni I$ ist das $\geq$ und damit auch die Vorzeichenbedingunen, immer erfüllt.

Für die Nebenbedingungen seien $w^+ := w + \theta z$ und $w^- := w - \theta z$, dann gelten $Aw^+ = Aw + \theta Az = Aw = b$ und $Aw^- = Aw - \theta Az = Aw = b$, also sind $w^+, w^- \in F$ und $w = \frac{1}{2}w^+ + \frac{1}{2}w^-$. w ist demnach, wie oben angenommen, keine Ecke, da es echt zwischen zwei Punkten liegt.

Damit ist „$\Leftarrow$“ bewiesen. □

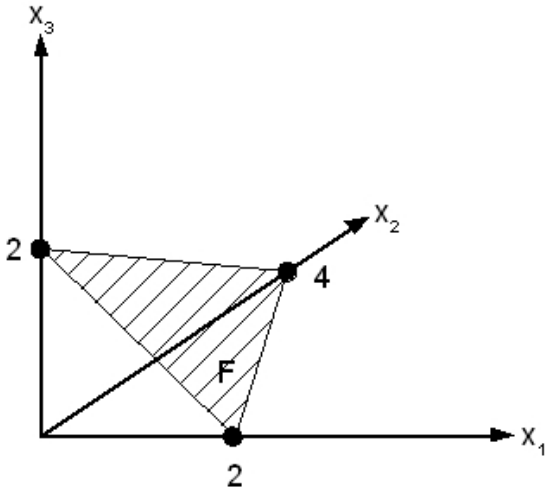


Abbildung 1.5: Graphische Darstellung eines linearen Programms

Proposition 1.2.7. *Sei F die Menge der zulässigen Lösungen eines linearen Programms und $p \in F$ ein Punkt in F. Dann ist $c^T x$ entwender auf F nach unten unbeschränkt (es ist kein Minimum vorhanden, da der Halbraum ins unendliche geöffnet ist) oder es existiert eine Ecke v von F mit $c^T v \leq c^T p$.*

Kurz: Ein Minimum existiert nicht, oder es wird an einer Ecke von F angenommen.

Beispiel (Geometrische Erklärung). Gegeben ist das lineare Programm mit der Nebenbedingung $2x_1 + x_2 + 2x_3 = 4$ und den Vorzeichenbedingungen $x_i \geq 0$ für $i = 1, \ldots, 3$. Der dazugehörige Polyeder ist in Abbildung 1.5 zu sehen.

Betrachtet man einen zweidimensionalen Polyeder F mit der Zielfunktion $c^T x$, so sind drei verschiedene Fälle zu unterscheiden. Bei $c^T x$ handelt es sich um die *c-Koordinate* von x, d.h. die Ausdehnung in c-Richtung. Um ein Mininum zu erhalten, muss eine zu c orthogonale Gerade (allg. Hyperebene) vom negativen Unendlichen entlang von c „geschoben“ werden, bis der Polyeder geschnitten wird. Bei dem ersten Schnittpunkt handelt es sich um das gesuchten Minimum. Statt eines Punktes kann der erste Schnitt auch mit einer höherdimensionalen Facette erfolgen. In dem fall gibt es unendlich viele optimale Lösungen.

Fall 1: Im einfachsten Fall liegt ein Minimum, wie in Abbildung 1.6 zu sehen, in einer Ecke (markiert durch einen großen Punkt). Es ist leicht zu sehen, dass es sich dabei um den Punkt mit der kleinsten c-Koordinate handelt.

Fall 2: Außerdem ist es möglich, wie in Abbildung 1.7 zu erkennen, dass das Minimum nicht eindeutig bestimmt ist. Dies geschieht genau dann wenn die zu c Orthogonale keine einzelne Ecke von F schneidet, sondern auf eine begrenzende Gerade fällt.

Fall 3: Außerdem ist es, wie in Abbildung 1.8 zu erkennen, möglich, dass gar kein Minimum existiert, da F zu einer Seite unbeschänkt ist.

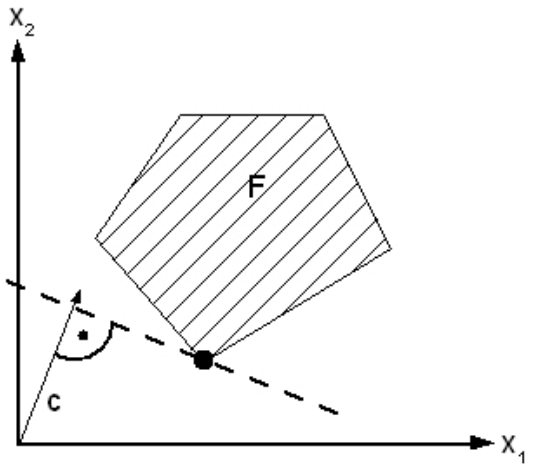


Abbildung 1.6: Minimum in Ecke

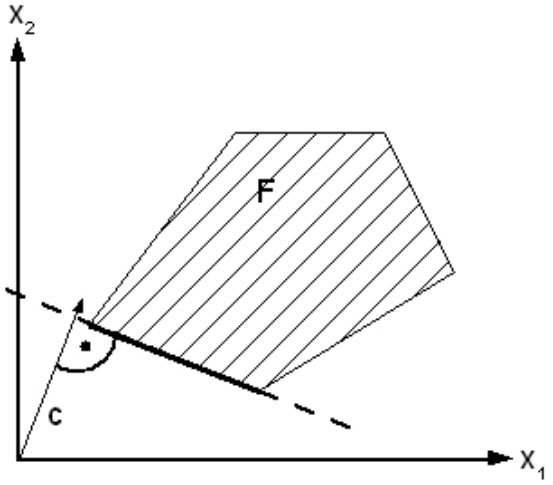


Abbildung 1.7: Minimum auf Kante

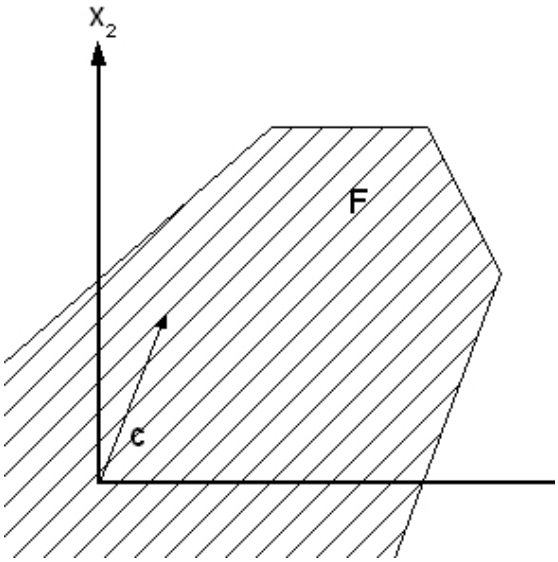


Abbildung 1.8: Kein Minimum vorhanden

Das Beispiel lässt sich auf n-dimensionale lineare Programme verallgemeinern: So wird an Stelle einer zu c orthogonalen Gerade eine zu c orthogonale Hyperebene verwendet. Minimas von $c^T x$ können auf einer d-dimensionalen Facette von F angenommen werden, wobei $0 \leq d \leq n-1$.

Proof. Ein lineares Programm mit den Vorzeichenbedingungen

$$\begin{aligned} x_1 &\geq 0 \\ x_2 &\geq 0 \\ x_3 &\geq 0 \\ &\vdots \\ x_n &\geq 0 \end{aligned}$$

und den Nebenbedingungen

$$\begin{aligned} a_{11}x_1 \quad \ldots \quad a_{1n}x_1 &= b_1 \\ a_{21}x_1 \quad \ldots \quad a_{2n}x_1 &= b_2 \\ a_{31}x_1 \quad \ldots \quad a_{3n}x_1 &= b_3 \\ &\vdots \\ a_{m1}x_1 \quad \ldots \quad a_{mn}x_1 &= b_m \end{aligned}$$

läßst sich folgendermaßen als Matirx darstellen:

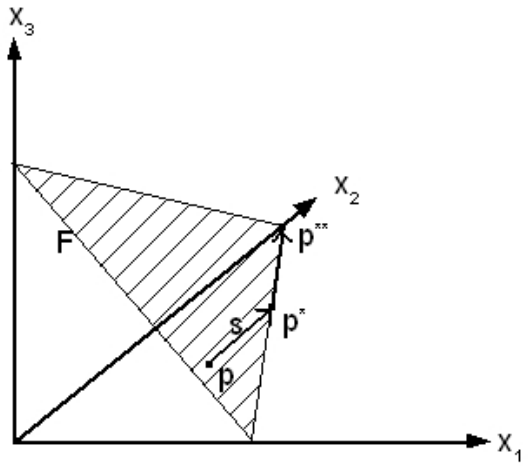


Abbildung 1.9: Verbesserte Wahl von I

$$\left(\begin{array}{cccc} 1 & 0 & \dots & 0 \\ 0 & 1 & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \dots & 0 & 1 \\ \hline & & A & \end{array}\right)$$

Sei $p \in F$ ein Punkt mit $p_i \geq 0$ für $i = 1, \dots, n$. Außerdem gelte $Ap = b$. Sei $J = \{i | p_i = 0\}$ eine Menge von Indizes und wähle I aus J ($I \subset J$) eine maximale Menge, so dass die dazugehörigen Vektoren die Form

$$0 \dots 010 \dots 0$$

haben, wobei die 1 an der i-ten Stelle steht; $i \in I$. Zusammen mit den Zeilenvektoren von A bilde eine linear unabhängige Menge S. Sei $r = |S|$ die Anzahl der in S linear unabhängigen Neben- und Vorzeichenbedingungen.

Falls $r = n$ (für die Neben- und Vorzeichenbedingungen ist die Gleichheit erfüllt), dann ist (nach dem oben genannten Lemma) eine Ecke des Polyeders erreicht.

Falls $r < n$, dann kann gezeigt werden: Entweder ist $c^T x$ auf F unbeschränkt, oder es existiert ein $p^* \in F$ mit einer noch größeren Menge von linear unabhängigen Neben- und Vorzeichenbedingungen, welche mit Gleichheit erfüllt sind.

Die Idee des Ansatzes ist es, wie in Abbildung 1.9 zu sehen: Wenn ein solches p nicht alle Vorzeichenbedingungen erfüllt, dann gibt es immer ein p^*, welches besser als p gewählt werden kann. Der Punkt p in der Abbildung 1.9 erfüllt offensichtlich alle Nebenbedingungen, jedoch keine einzige Vorzeichenbedingug.

p^* ist die erste Verbesserung von p und erfüllt bereits die Gleichheit für die Vorzeichenbedingungen von x_1 und x_3. p^{**} verbessert noch einmal p^* und nimmt für alle Vorzeichenbedingungen die Gleichheit an.

Dieses Verfahren kann man so lange fortsetzen, bis n Neben- und Vorzeichenbedingungen mit Gleichheit erfüllt sind. Dann ist eine Ecke erreicht.

Sei

$$F^* = \{x \in \mathbb{R}^n | \forall_{j \in J}\, x_j = 0 \wedge Ax = 0\}$$

. Dabei handelt es sich um den *Nullraum* einer $r \times n$-Matrix vom Rang r (so zu wählen). $x_j = 0$ bewirkt, dass bereits mit Gleichheit erfüllte Vorzeichenbedingungen auch in der nächsten Iteration erfüllt bleiben. Die Dimension von F^* ist $\dim F^* = n - r > 0$.

Sei $w \in F$, $w \neq 0$(Nullvektor).

Bemerkung.

$$p_j = 0 \Rightarrow w_j = 0 \text{ für } j = 1, \ldots, n$$

Beweis der Bemerkung. Wenn $j \in I$, dann gilt dies offensichtlich, da F^* so gewählt wurde. Ist $j \notin I$, dann ist

$$w_j = (0 \ldots 010 \ldots 0)w = \sum_{i \in I} a_i(s_i w) = 0$$

da an der j-ten Stelle von $(0 \ldots 010 \ldots 0)$ eine 1 steht (kann aus Linearkombination erzeugt werden: $\sum\limits_{i \in I} a_i s_i$) und $s_i = 0$. Daraus folgt dann auch

$$p_j = 0 \Rightarrow w_j = 0$$

□

Es gilt nun $\forall_{\lambda \in \mathbb{R}}\, A(p + \lambda w) = b$, da Aw=0. Falls nun $p_j = 0$, dann gilt auch $p + \lambda w = 0$ (nach Bemerkung). D.h., $p + \lambda w$ erfüllt die Neben- und Vorzeichenbedingungen mit 0 dort, wo es p auch tut.

Ohne weitere Einschränkgungen gilt für die Zielfunktion $c^T w = 0$. Ist dies nicht erfüllt, so kann w durch $-w$ ersetzt werden. Alle oben beschriebenen Eigenschaften bleiben dabei erhalten.

Sei $J^- = \{j | w_j < 0\}$. Es sind nun drei Fälle zu unterscheiden:

Fall 1) $c^T w < 0$ und $J^- = \varnothing$, also $\forall_{j \in J}\, w_j \geq 0$.

Dann gilt $\forall_{\lambda \geq 0}\, (p + \lambda)w \in F$, da $p \geq 0$ und da $w \geq 0$. Man beachte jedoch, für $\lambda \to \infty$, dass $c^T(p + \lambda w) = c^T + \lambda c^T w \to -\infty$. D.h., dass die Zielfunktion auf F nach unten unbeschränkt ist.

Fall 2) $J^- = \varnothing$, also $w_j < 0$.

Es gilt $S = \min\limits_{j \in J} \frac{p_j}{-w_j} > 0$, da $w_j < 0$ und damit $p_j > 0$.

Sei j^* mit $S = \frac{p_{j^*}}{-w_{j^*}}$ und $p^* = p + sw$, dann gilt:

a) Für alle j, für die $p_j = 0$, so ist ${p_j}^* = p_j + sw_j = 0$ (nach Bemerkung $p_j = 0 \Rightarrow w_j = 0$).

b) Für j^* ist $p_{j^*} > 0$, aber auch hier gilt: ${p_{j^*}}^* = p_{j^*} + \frac{p_{j^*}}{-w_{j^*}} {w_j}^* = 0$.

Also erfüllt p^* eine Vorzeichenbedingung mehr mit Gleichheit als p.

Fall 3): $J^- = \varnothing$ und $c^T w = 0$.

Als Übung

Also: Das Minimum von $c^T x$, falls es auf F existiert, wird von einer Ecke von F angenommen, damit wird das lineare Programm zu einem *endlichen Problem*. Damit müssen in der Zielfunktion endlich viele Punkte, höchstens $\binom{n}{m}$, betrachtet werden. □

Beispiel (Laufzeit des Brute-Force-Ansatz). Algorithmus: Bestimme alle Ecken des Polyeders, d.h. Basen und bfs. und minimiere $c^T x$ für diese.

Der größte Binomialkoeffizient ist $\binom{n}{\frac{n}{2}} \approx \frac{2^n}{\sqrt{\frac{\pi}{2} n}}$ (Stirling). Damit sind im *worst-case* exponentiell viele Punkte zu untersuchen $\Rightarrow$ exponentielle Laufzeit.

Anderes Beispiel:

$$\binom{n}{\frac{1}{4}n} = \frac{n(n-1)\cdots(\frac{3}{4}n+1)}{1 \times 2 \times \cdots \times \frac{1}{4}n} \leq \frac{(\frac{3}{4}n)^{\frac{1}{4}n}}{(\frac{1}{4}n)^{\frac{1}{4}n}} = (3n)^{\frac{1}{4}n}$$

Beispiel (Würfel). Ein n-dimensionaler Würfel ist beschrieben durch $x_i \geq 0$ und $x_i \leq 1$ für $i = 1, \ldots, n$. D.h., dass lineare Programm besteht aus n Variablen und $2n$ Nebenbedingungen. Die Anzahl der Ecken beträgt 2^n. Iteriert man über alle Ecken, so ist offensichtlich, dass dies exponentielle Laufzeit benötigt.

1.4. Das Simplexverfahren

Vorbemerkung: Ein Simplex ist das einfachste Polygon in einem Raum. Im zweidimensionalen ist ein Simplex ein Dreieck, im dreidimensionalen eine Pyramide mit dreieckiger Grundfläche, im eindimensionalen ein Intervall.

Zuerst probieren wir den Simplex-Algorithmus an einem Beispiel aus. Das lineare Programm liegt in kanonischer Form vor. Die zu maximierende Zielfunktion ist $5x_1 + 4x_2 + 3x_2$. Außerdem gibt es folgende Nebenbedingungen:
$2x_1 + 3x_2 + x_3 \leq 5$
$4x_1 + x_2 + 2x_3 \leq 11$
$3x_1 + 4x_2 + 2x_3 \leq 8$
die Vorzeichenbedingungen sind $x \geq 0$, also $x_1, x_2, x_3 \geq 0$

Mit Hilfe von Schlupfvariablen x_4, x_5, x_6 formen wir dieses Problem in eine lineares Programm in Standardform um. Dadurch ergibt sich:
$x_4 = 5 - 2x_1 - 3x_2 - x_3$
$x_5 = 11 - 4x_1 - x_2 - 2x_3$
$x_6 = 8 - 3x_1 - 4x_2 - 2x_3$
Anstatt die ursprüngliche Zielfunktion zu maximieren, minimieren wir $z = -5x_1 - 4x_2 - 3x_3$. Vorzeichenbedingungen sind weiterhin $x \geq 0$.

Zuerst müssen wir nun eine zulässige Basislösung suchen. Diese finden wir indem wir drei linear unabhängige Spalten aus der Matrix A wählen. Die anderen Variablen werden null gesetzt. Die Matrix A der Gleichung $A \cdot x = b$ für das Problem in Standardform sieht wie folgt aus:

$$A = \begin{pmatrix} 2 & 3 & 1 & 1 & 0 & 0 \\ 4 & 1 & 2 & 0 & 1 & 0 \\ 3 & 4 & 2 & 0 & 0 & 1 \end{pmatrix}$$

Auf der rechten Seite von A steht die Einheitsmatrix. Das erleichtert uns die Auswahl der ersten zulässigen Baasislösung. Wir setzen x_1,x_2 und x_3 auf null. Als Basisspalten werden damit die Spalten 4, 5 und 6 gewählt. Dann ergibt sich die folgene Basislösung:

$$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 5 \\ 11 \\ 8 \end{pmatrix}$$

Diese Lösung erfüllt $A \cdot x = b$ und alle Vorzeichenbedingungen. Als Lösung für die Zielfunktion ergibt sich nun $z = 0$. Lässt sich noch ein kleinerer Wert erreichen?
Ja - z kann verkleinert werden durch Erhöhung von x_1 auf einen Wert $\Theta \geq 0$. Dann muss gelten: $x_1 = \Theta$ und $x_2 = x_3 = 0$
Aus den Gleichungen für die übrigen Variablen und den Vorzeichenbedingungen ergibt sich eine Einschränkung von Θ.

$$\begin{array}{lcll} x_4 & = & 5 - 2\Theta & \\ x_5 & = & 11 - 4\Theta & \text{und } x > 0 \Rightarrow \\ x_6 & = & 8 - 3\Theta & \end{array} \quad \begin{array}{lcl} \Theta & \leq & \frac{5}{2} \\ \Theta & \leq & \frac{11}{4} \\ \Theta & \leq & \frac{8}{3} \end{array}$$

Wir wählen also $\Theta = \frac{5}{2}$, denn das ist die maximale Lösung, die alle Vorzeichenbedingungen erfüllt. Daraus erhalten wir nun eine neue zulässige Basislösung, wobei $x_4 = 0$ wird und x_1 größer null. Diesmal verwenden wir die Spalten 1,5 und 6 als Basisspalten. Wir schreiben die erste Gleichung um:
$x_1 = \frac{5}{2} - \frac{3}{2}x_2 - \frac{1}{2}x_3 - \frac{1}{2}x_4$
In den Gleichungen für x_5 und x_6 muss x_1 ersetzt werden.
$x_5 = 11 - 4(\frac{5}{2} - \frac{3}{2}x_2 - \frac{1}{2}x_3 - \frac{1}{2}x_4) - x_2 - 2x_3$
$x_5 = 1 + 5x_2 + 2x_4$
$x_6 = 8 - 3(\frac{5}{2} - \frac{3}{2}x_2 - \frac{1}{2}x_3 - \frac{1}{2}x_4) - 4x_2 - 2x_3$
$x_6 = \frac{1}{2} + \frac{1}{2}x_2 - \frac{1}{2}x_3 + \frac{3}{2}x_4$
Die entsprechende zulässige Basislösung ist:

$$\begin{pmatrix} \frac{5}{2} \\ 0 \\ 0 \\ 0 \\ 1 \\ \frac{1}{2} \end{pmatrix}$$

Anschaulich betrachtet ist dies eine andere Ecke des Polytops für die sich eine kleinere Lösung ergibt. Wir erhalten nun $z = -\frac{25}{2}$. Dieser Wert ist besser als null, aber ist er schon das Optimum?
Um das herauszufinden formen wir nun auch die Zielfunktion so um, dass sie unabhängig von x_1 ist:
$z = -5(\frac{5}{2} - \frac{3}{2}x_2 - \frac{1}{2}x_3 - \frac{1}{2}x_4) - 4x_2 - 3x_3$
$z = -\frac{25}{2} + \frac{7}{2}x_2 - \frac{1}{2}x_3 + \frac{5}{2}x_4$
Die Zielfunktion ist noch nicht optimal. Man erkennt, dass x_3 auf ein $\Theta > 0$ vergrößert werden kann, um die Zielfunktion zu verringern. Aus den Nebenbedingungen und Vorzeichenbedingungen ergeben sich die folgenden Einschränkungen für Θ.

$$\begin{array}{lcl} x_1 & = & \frac{5}{2} - \frac{\Theta}{2} \\ x_5 & = & 1 \\ x_6 & = & \frac{1}{2} - \frac{\Theta}{2} \end{array} \quad \text{und } \bar{x} > 0 \Rightarrow \quad \begin{array}{lcl} \Theta & \leq & 5 \\ \Theta & \leq & 1 \end{array}$$

Wir wählen $\Theta = 1$, da dies die maximale Lösung ist, die die Vorzeichenbedingungen erfüllt. Die neue zulässige Basislösung ist:

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

Die ausgewählten Basisspalten sind 1, 3 und 5 und der Wert von z ist jetzt -13. Die Nebenbedingungen passen wir wieder entsprechend an:
$x_3 = 1 + x_2 + 3x_4 - 2x_6$
$x_1 = \ldots$ und $x_5 = \ldots$
Auch die Zielfunktion wird umgeformt:
$z = -\frac{25}{2} + \frac{7}{2}x_2 - \frac{1}{2}(1 + x_2 + 3x_4 - 2x_6) + \frac{5}{2}x_4$
$z = -13 + 3x_2 + x_4 + x_6$
Diese Zielfunktion kann nicht weiter erniedrigt werden, weil alle Koeffizienten

positiv sind. Die Funktion nimmt ihren minimalen Wert an, wenn $x_2, x_4, x_6 = 0$. Wir haben also das Minimum gefunden. Die Lösung des ursprünglichen Problems (ohne Schlupfvariablen):

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 2 \\ 0 \\ 1 \end{pmatrix}$$

Es folgt eine formalere Beschreibung des Simplex-Algorithmus:
Ausgehend von der ursprünglichen zulässigen Basislösung führen wir einige Schritte wie oben beschrieben durch zu neuen zulässigen Basislösungen, wobei die Zielfunktion immer verkleinert wird. Wir hören auf, wenn sie nicht mehr verkleinert werden kann, wenn also in der Zielfunktion bezüglich der zulässigen Basis Lösung keinen negativen Koeffizienten mehr vorkommen. Die Schritte werden Pivot-Schritte genannt.

Um das ganze übersichtlicher zu gestalten arbeitet man mit Simplex-Tableaus. Wie im Beispiel wird immer dafür gesorgt, dass die Basisspalten eine Einheitsmatrix bilden.

	X_1	X_2	X_3	X_4	X_5	X_6	Vektor b	
Matrix A →	2	3	1	1	0	0	5	
	4	1	2	0	1	0	11	
	3	4	2	0	0	1	8	
-z →	5	4	3	0	0	0	0	← Wert

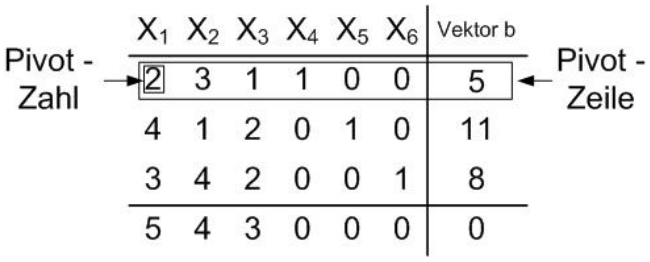
Ein Pivot-Schritt

Schritt 1: Wir finden das Maximum der untersten Zeile. (Dies ist eine Heuristik. Man wählt den steilsten Anstieg der Zielfunktion, um eine möglichst große Veränderung zu erreichen. Es wären auch andere Heuristiken möglich.) Die Spalte mit dem Maximum heißt Pivot-Spalte. Falls dieses Maximum kleiner gleich null ist, kann diese Zielfunktion nicht weiter verkleinert werden. In diesem Fall ist die optimale Lösung bereits gefunden. Andernfalls führen wir Schritt zwei durch.

	X_1	X_2	X_3	X_4	X_5	X_6	Vektor b
Pivot - Spalte →	2	3	1	1	0	0	5
	4	1	2	0	1	0	11
	3	4	2	0	0	1	8
	5	4	3	0	0	0	0

Schritt 2: Für jede Zeile, deren Eintrag r in der Pivot-Spalte positiv ist (Ne-

gative Einträge können vernachlässigt werden, da diese das Ergebnis nicht einschränken. Falls kein positiver Eintrag existiert ist die Zielfunktion unbeschränkt und es gibt kein Minimum.), betrachten wir den Eintrag s in der rechten Spalte (Vektor b) und wir bestimmen die Zeile mit dem kleinsten $\frac{s}{r}$. Diese Zeile heißt Pivot-Zeile. Der Eintrag der in der Pivot-Spalte und in der Pivot-Zeile liegt heißt Pivot-Zahl.



X_1	X_2	X_3	X_4	X_5	X_6	Vektor b
2	3	1	1	0	0	5
4	1	2	0	1	0	11
3	4	2	0	0	1	8
5	4	3	0	0	0	0

Schritt 3: Jeder Eintrag in der Pivot-Zeile wird durch die Pivot-Zahl geteilt.

X_1	X_2	X_3	X_4	X_5	X_6	Vektor b
0	$\frac{3}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	**0**	**0**	$\frac{5}{2}$
4	1	2	0	1	0	11
3	4	2	0	0	1	8
5	4	3	0	0	0	0

Schritt 4: Von den übrigen Zeilen des Tableaus subtrahieren wir ein geeignetes Vielfaches der Pivot-Zeile, so dass der Eintrag in der Pivot-Spalte null wird.

X_1	X_2	X_3	X_4	X_5	X_6	Vektor b
0	$\frac{3}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	0	0	$\frac{5}{2}$
0	**-5**	**0**	**-2**	**1**	**0**	**1**
0	$-\frac{1}{2}$	$\frac{1}{2}$	$-\frac{3}{2}$	**0**	**1**	$\frac{1}{2}$
0	$-\frac{7}{2}$	$\frac{1}{2}$	$-\frac{5}{2}$	**0**	**0**	$-\frac{25}{2}$

Es gibt noch einen positiven Eintrag in der unteren Zeile, daran sieht man, dass die Lösung noch nicht ideal ist. Es wird fortgefahren bis alle Einträge negativ sind. Dann steht der Wert im Feld rechts unten.

Vorlesung vom 28.04.2008
Skript erstellt von *David Kaltschmidt*

1.2.1 Einzelprobleme des Simplex-Algorithmus

Initialisierung Wie findet man ein geeignetes Start-Dictionary?

Iteration Findet man immer eine geeignete Eintrittsvariable und eine Austrittsvariable, die stattdessen herausfällt?

Terminierung Kann die Methode in eine unendliche Schleife laufen?

Initialisierung: Wahl eines Start-Dictionaries

$$x_{n+i} = b_i - \sum_{j=1}^{n} a_{i,j} x_j \quad i = 1 \dots m$$

Die Basislösung: $\begin{pmatrix} 0 \\ \vdots \\ 0 \\ b_1 \\ \vdots \\ b_m \end{pmatrix} \Leftarrow \begin{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \\ \begin{pmatrix} x_{n+1} \\ \vdots \\ x_{n+m} \end{pmatrix} \end{pmatrix}$ ist eine zulässige Lösung genau dann wenn $b_i \geq 0$.

Angenommen, LP in kanonischer Form mit Zielfunktion $\min c^T \overrightarrow{x}$, Nebenbedingungen $A\overrightarrow{x} \leq \overrightarrow{b}$ sowie Vorzeichenbedingungen $\overrightarrow{x} \geq \overrightarrow{0}$.

Daraus wird ein **Hilfsproblem** konstruiert. Es wird eine zusätzliche Variable x_0 eingeführt, die dann minimiert werden soll. Die Nebenbedingungen werden wie folgt abgeändert:

$$\sum_{j=1}^{n} a_{i,j} - x_0 \leq b_i \quad i = 1 \dots m$$

Die daraus resultierende Matrix A verändert sich wie folgt:

$$A = (a_{i,j})_{1 \leq i \leq m,\, 1 \leq j \leq n} \Rightarrow \begin{pmatrix} -1 & & \\ \vdots & & A \\ -1 & & \end{pmatrix}$$

Satz 1.2.8. *Für das ursprüngliche Problem und das Hilfsproblem sind folgende Aussagen äquivalent:*

1. *Das ursprüngliche Problem hat eine zulässige Lösung.*
2. *Das Hilfsproblem hat eine zulässige Lösung mit $x_0 = 0$.*
3. *Das Hilfsproblem hat ein Optimum von 0.*

Beweisskizze. Zur Lösung des Hilfsproblems betrachte man das sich ergebene Dictionary und die neue Zielfunktion w:

$$x_{n+i} = b_i - \sum_{j=1}^{n} a_{i,j} x_j + x_0 \quad i = 1 \dots m$$

$$\min w = x_0$$

Man beachte, dass das Dictionary nicht immer zulässig ist, da $b \geq 0$ nicht gewährleistet ist. Um es zulässig zu machen, setze man $x_0 = \max_{i=1\ldots m}(-b_i)$.

Beispiel. Minimiere $x_1 - 2x_2$ unter den Nebenbedingungen:

$$\begin{array}{rcl} -x_1 + x_2 & \leq -1 \\ 2x_1 - x_2 & \leq 4 \\ x & \geq 0 \end{array} \quad \text{wird dann zu} \quad \begin{array}{rcl} -x_1 + x_2 - \mathbf{x_0} & \leq -1 \\ 2x_1 - x_2 - \mathbf{x_0} & \leq 4 \\ x & \geq 0 \end{array}$$

Die Schlupfvariablen ergeben sich dann wie folgt:

$$\begin{array}{rrcrcrcr} x_3 = & -1 & + & x_1 & - & x_2 & + & \mathbf{x_0} \\ x_4 = & 4 & - & 2x_1 & + & x_2 & + & \mathbf{x_0} \end{array}$$

Wie man leicht sieht, ist die zugehörige Basislösung $(0\ 0\ 0\ -1\ 4)^T$ nicht zulässig. Man setze $x_0 = 1$ und behandele es als Eintrittsvariable. Mit x_3 als Austrittsvariable ergibt sich:

$$\begin{array}{rl} x_0 = & 1 - x_1 + x_2 \\ x_4 = & 5 - 3x_1 + 2x_2 + 3 \end{array} \quad \text{mit der Basislösung} \quad \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 5 \end{pmatrix}$$

Sobald x_0 als Austrittsvariable möglich ist, benutze es. Nächster Schritt im Beispiel: Erhöhe x_1 auf 1, so dass $x_0 = 0$ mit folgendem Dictionary:

$$\begin{array}{rl} x_1 = & 1 + x_1 + x_2 + x_3 - x_0 \\ x_4 = & 2 - x_2 - 2x_2 - 3x_0 \end{array} \quad \text{mit der Basislösung} \quad \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 2 \end{pmatrix}$$

Wenn $x_0 = 0$, dann wurde das Minimum erreicht und man kann x_0 aus der Basislösung und dem Dictionary entfernen. Die resultierende Basislösung ist eine zulässige Basislösung des ursprünglichen Programms:

$$\begin{array}{rl} x_1 = & 1 + x_1 + x_2 + x_3 \\ x_4 = & 2 - x_2 - 2x_2 \end{array} \quad \text{mit Basislösung} \quad \begin{pmatrix} 1 \\ 0 \\ 0 \\ 2 \end{pmatrix}$$

□

1.2.2 Iteration: Wahl einer Eintrittsvariablen

Es kommt jede Variable in Frage, deren Koeffizient in der momentanen Zielfunktion negativ ist (Heuristik: wähle die Variable mit dem betragsmäßig größten negativem Koeffizienten). Falls keine solche Variable existiert, so wurde das Minimum gefunden. Die Wahl der Austrittsvariablen war in den bisherigen Beispielen eindeutig, aber im Allgemeinen ist das nicht der Fall, z.B.:

$$\begin{array}{rl} x_2 = & 5 + 2x_3 - x_4 - 3x_1 \\ x_5 = & 7 - 3x_4 - 4x_1 \\ \hline z = & -5 - x_3 + x_4 + x_1 \end{array} \quad \text{mit Eintrittsvariable } x_3 \quad \begin{array}{l} x_3 = \Theta > 0 \\ 5 + 2\Theta \geq 0 \end{array}$$

In einem solchen Fall gibt es kein Minimum, da z beliebig klein werden kann. Eine weitere Möglichkeit ist **Degeneriertheit:** Mehr als ein Kandidat für Austrittsvariable, z.B.:

$$\begin{array}{rl} x_4 = & 1 - 2x_3 \\ x_5 = & 3 - 2x_1 + 4x_2 - 6x_3 \\ x_6 = & 2 + x_1 - 3x_2 - 4x_3 \\ \hline z = & -2x_1 + x_2 - 8x_3 \end{array}$$

mit Eintrittsvariable
$x_3 \to \frac{1}{2}$
$\Rightarrow x_4 = 0$
$\Rightarrow x_5 = 0$
$\Rightarrow x_6 = 0$

führt zu Basislösung $\begin{pmatrix} 0 \\ 0 \\ \frac{1}{2} \\ 0 \\ 0 \\ 0 \end{pmatrix}$

Es stehen dann mehrere Austrittsvariablen zur Verfügung. Vorerst wähle man eine beliebige Variable als Austrittsvariable, z.B. x_4:

$$\begin{array}{rl} x_3 = & \frac{1}{2} - \frac{1}{2}x_4 \\ \text{degenerierte } x_5 = & \mathbf{0} - 2x_1 + 4x_2 + 3x_4 \\ \text{Basisvariablen } x_6 = & \mathbf{0} + x_1 - 3x_2 + 2x_4 \\ \hline z = & -4 - 2x_1 + x_2 + 4x_4 \end{array}$$

mit Eintrittsvariable $x_1 \to \Theta$
$\Rightarrow x_5 : \ 2\Theta \geq 0$
$\Rightarrow x_6 : \Theta \geq 0$
$\Rightarrow \Theta = 0$

Angenommen, man lässt $x_1 = 0$, dann sind mehrere aufeinanderfolgende Schritte dieser Art möglich. Es kann sogar zu einer unendlichen Schleife kommen (**cycling**), das heisst, das gleiche Dictionary wird einige Schritte später wieder erreicht.

Beobachtung 1.2.9. *Zwei Dictionaries zu einem Linearen Programm mit der gleichen Basis sind identisch.*

Korollar 1.2.10. *Falls der Simplex-Algorithmus nicht terminiert, muss ein Zyklus vorliegen, denn es gibt nur endlich viele Basen:* $\begin{pmatrix} n+m \\ n \end{pmatrix}$

1.2.3 Terminierung: Bland's Regel

Bland's Regel Bei mehreren Möglichkeiten für Eintritt- und Austrittsvariablen wird immer die mit dem kleinsten Index gewählt.

Satz 1.2.11. *Falls im Simplex-Verfahren Eintritts- und Austrittsvariablen nach Bland's Regel gewählt werden, so terminiert das Programm.*

1.3 Blands Regel

Wie im letzten Kapitel gezeigt, lösen wir ein Hilfsproblem, um eine erste Basislösung zu finden. Ein anderes Problem ist folgendes:

$$x_3 = \frac{1}{2} - \frac{1}{2}x_4 \tag{1.1}$$
$$x_5 = -2x_1 + 4x_2 + 3x_4 \tag{1.2}$$
$$x_6 = x_1 - 3x_2 + 4x_4 \tag{1.3}$$
$$z = -4 - 2x_1 + x_2 + 4x_4 \tag{1.4}$$

Wir wollen x_1 auf θ erhöhen, so dass die Vorzeichenbedingungen noch erfüllt sind. x_3 bleibt immer größer 0. Bei x_5 darf man höchstens auf 0 erhöhen. Das ist die einzige Möglichkeit, obwohl noch nicht optimal. Sowohl x_5 als auch x_6 können Austrittsvariable sein, z wird nicht erhöht.

Diese degenerierten Fälle können durchaus mehrmals hintereinander auftreten und das Verfahren in eine Schleife laufen. Der Grund ist: Es gibt keine klare Regel über die Austrittsvariable.

Blades Regel für Ein- und Austrittsvariablen: Bei mehreren Möglichkeiten wird die mit dem kleinsten Index gewählt.

Satz 1.3.1. *Falls man Blands Regel anwendet und die Ein- und Austrittsvariablen wählt, so terminiert das Verfahren.*

Beweis. Angenommen nicht, das heißt es existiert eine Folge von Dictionaries $D_0, D_1, ..., D_k$ mit $D_k = D_0$.

Wir betrachten Variablen, die sowohl als Basisvariablen, als auch als Nicht-Basisvariablen in der Folge auftreten. Diese nennen wir ≫wackelige≪ Variablen. Sei x_t die wackelige Variable mit maximalem t.

Es gibt $D, D^* \in D_0, D_1, ..., D_k, D_1, ..., D_k$, dass x_t Austrittsvariable bei D und Eintrittsvariable bei D^* ist. D ist von der Form $x_i = b_i - \sum_{j \notin B} a_{ij} \cdot x_j \qquad i \in B$.

$B =$ Indizes der Basisvariablen von D.
$z = v + \sum_{j \notin B} c_j \cdot x_j$

z hat den gleichen Wert v in der ganzen Folge von Dictionaries. In D^* ist $z = v + \sum_{j=i}^{n+m} c_j \cdot x_j \qquad c_j^* = 0$ für die Basisvektoren in D^*.

Für ein beliebiges $a \in \mathbb{R}$ ist

$$x_s = a \tag{1.5}$$
$$x_j = 0 \qquad j \notin B; j \neq s \tag{1.6}$$
$$x_i = b_i - a_{is} \cdot a \qquad i \in B \tag{1.7}$$
$$z = v + c_s \cdot a \tag{1.8}$$

Das ist eine Lösung von D und gilt für alle a.

Was ist bei negativem a? Die Vorzeichenbedingungen werden nicht erfüllt, die Lösung ist nicht zulässig.

D^* entsteht aus D durch äquivalente Umformungen, also erfüllt die Lösung auch D^*.

$z = v + \sum_j c_j^* \cdot x_j$, z in die Gleichung eingesetzt ergibt:
$v + c_s \cdot a = v + c_s^* \cdot a + \sum_{i \in B} c_i^* \cdot (B_i - a_{is} \cdot a \cdot \underbrace{(c_s - c_s^* + \sum_{i \in B} c_i^* \cdot a_{is})}_{=0} \cdot a = \sum_{i \in B} c_j^* \cdot b_i$

für alle a!

Also

$$c_s - c_s^* + \sum_{i \in B} c_i^* \cdot a_{is} = 0 \tag{1.9}$$

x_s war eine Eintrittsvariable in D. Daraus folgt $c_s < 0$. x_s ist nicht Eintrittsvariable in D^*, obwohl $s < t \rightarrow c_s^* c_s^* \geq 0$, also ist $c_s - c_s^* \leq 0$. Mit (1.9) folgt:

$$\exists r \in B : c_r^* \cdot a_{rs} > 0 \tag{1.10}$$

x_r is ein Basisvektor in D. Aus $c_r^* \neq 0$ folgt: x_r ist nicht Basisvektor in D^*. Also ist x_r wackelig. Daraus folgt: $r \leq t$. Da x_t Austrittsvariable und x_s ein Vektor in in D ist, folgt: $a_{ts} > 0$, denn $x_t = b_t ... - a_{ts} \cdot x_s ....$ Ansonsten könnte die Zeile t nicht Pivotzeile werden. Damit wissen wir, dass $c_t^* \cdot a_{ts} < 0$. Wegen (1.10) ist $r \neq t$, d.h. $r < t$.

Da trotzdem x_r kein Eintrittsvektor in D^* ist, ist $c_r^* \geq 0$. Mit 1.10 haben wir $a_r s > 0$. D, D^* haben den gleichen Lösungsvektor, da alle Zwischenschritte degeneriert sind. Da x_r kein Basisvektor in D^* ist, folgt: x_r hat den Wert 0 in beiden Dictionaries, also $b_r = 0$. x_r war Kandidat für den Austrittsvektors in D, es wurde aber x_t genommen. Das ist ein Widerspruch zu Blands Regel.

□

1.4 Laufzeit des Simplex-Algorithmus

Pivotschritte:

1. Finden der Pivotspalte $O(n)$
2. Finden der Pivotzeile $O(m)$
3. Division der Pivot-Zeile durch die Pivot-Zahl $O(n)$
4. Von jeder Zeile ein geeignetes Vielfaches der Pivotzeile subtrahieren $O(n \cdot m)$

Das ergibt eine Laufzeit von $O(n \cdot m)$ für einen Pivotschritt. Wieviele Pivotschritte gibt es?

Empirische Beobachtungen an praktischen Problemen zeigen, dass es etwas $\frac{3}{2}m$ Schritte sind, selten mehr als $3m$. (Dantzig)

Theoretische Ergebnisse: Bergwardt, Smale haben 1982 gezeigt, dass die mittlere Anzahl von Pivotschritten in $O(m)$ liegt.

In der Praxis sind Probleme mit mehreren hundert Variablen und Nebenbedingungen innerhalb von Sekunden lösbar.

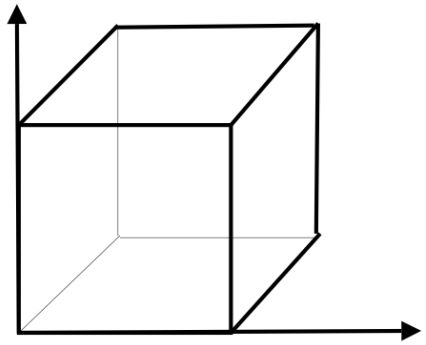
Im schlechtesten Fall (Klee, Minty 1972) kann der Algorithmus alle Ecken des Polyeders durchlaufen, es gibt exponentiell viele Ecken:

Minimiere die Zielfunktion: $\sum_{j=1}^{n} 10^{n-j} x_j$
Nb. $(2 \cdot \sum_{j=1}^{i-1} 10^{i-j} x_j) + x_i \leq 100^{i-1}$
Z.B. $100x_1 + 10x_2 + x_3$

Das führt zu 2^{n-1} Iterationen, wenn man als Pivotregel die Variante mit größtem Koeffizienten nimmt.

Andere Pivotregeln sind möglich, z.B. Blands Regel oder „Wähle die Variable mit der größten Abnahme der Zielfunktion“. Auch hier gibt es Beispiele mit exponentieller Laufzeit. (Jeroslav 1973)

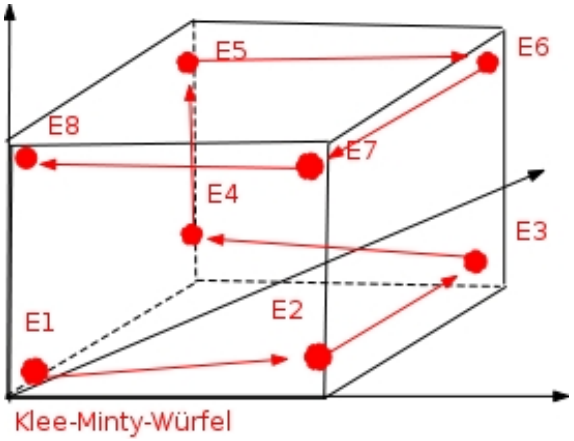
Der Klee-Minty-Würfel ist definiert durch: Sei $\epsilon \in (0, \frac{1}{2})$
Nb. $\epsilon \leq x_1 \leq 1$
$\epsilon x_{x-1} \leq x_i \leq 1 - \epsilon x_{i-1}$
Vzb. $x \geq 0$
Maximiere x_n



1.4.1 Klee-Minty (Wiederholung)

Der Klee-Minty-Würfel ist ein Beispiel dafür, dass der Simplex-Algorithmus in exponentiell vielen Schritten laufen kann. Es handelt sich um einen perturbierten n-dimensionalen Würfel, weswegen er etwas schief ist. Es gibt einen Weg, der alle Ecken durchläuft, während x, die zu maximierende Variable, monoton steigt. Dies geschieht bei einer ungeschickten Wahl der Pivot-Spalten, und man hat somit 2^{n-1} Pivotschritte.

maximiere x_n
$\epsilon \leq x_1 \leq 1$
$0 < \epsilon < \frac{1}{2}$
$\epsilon x_{i-1} \leq x_i \leq 1 - \epsilon x_{i-1}$
$x \geq 0$



Klee-Minty-Würfel

$E1 : (\epsilon, \epsilon^2, \epsilon^3)$
$E2 : (1, \epsilon, \epsilon^2)$
$E3 : (1, 1-\epsilon, \epsilon - \epsilon^2)$
$E4 : (\epsilon, 1-\epsilon^2, \epsilon - \epsilon^3)$
$E5 : (\epsilon, 1-\epsilon^2, 1 - \epsilon + \epsilon^3)$
$E6 : (1, 1-\epsilon, 1 - \epsilon + \epsilon^2)$
$E7 : (1, \epsilon, 1-\epsilon^2)$
$E8 : (\epsilon, \epsilon^2, 1-\epsilon^3)$

1.4.2 Dualität

Wir betrachten ein LP in Standardform. Die Dualität ist ein Phänomen, dass einem helfen kann, Lineare Programme zu lösen. Man erhält Schranken für das gesuchte Optimum.

Beispiel.
minimiere $x_1 + 2x_2 - 7x_3$
Nb:

$$\begin{aligned} 3x_1 + 2x_2 + 5x_3 &= 6 \\ 2x_1 + 0x_2 + x_3 &= 12 \end{aligned}$$

Vzb: $x_1 \geq 0, x_2 \geq 0, x_3 \geq 0$

Versuche, die untere Schranke für das Minimum z^* zu finden. Multipliziere zwei Mal die zweite Zeile und subtrahiere die erste Zeile:

$18 = x_1 + x_2 - 3x_3 \leq z \Rightarrow z^* \geq 18$

etwas systematischer: Sei $w = \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix}$ eine zulässige Lösung, und die Koeffizienten $y = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$ beliebig. Betrachte die Linearkombination der Nebenbedingungen mit den y's als Koeffizienten. x_i wird ersetzt durch w_i.

$$f = \underbrace{(3w_1 - w_2 + 5w_3)}_{=6} y_1 + \underbrace{(2w_1 + 0w_2 + w_3)}_{=12} y_2 = 6y_1 + 12y_2$$

Andererseits ist $f = (3y_1 + 2y_2)w_1 + (-y_1 + 0y_2)w_2 + (5y_1 + y_2)w_3$.

Wir möchten f gerne als untere Schranke für z^* haben. z^* ist die optimale Lösung, w^* ist die dazugehörige optimale Ecke des Polyeders. $z^* = c^T w^*$. Dies gilt wegen $w \geq 0$. Um die beste untere Schranke zu finden, muss man die neue Zielfunktion maximieren. Man erhält dadurch:

maximiere die neue Zielfunktion $6y_1 + 12y_2$
Nb:

$$\begin{aligned} 3y_1 + 2y_2 &\leq 1 \\ -y_1 + 0y_2 &\leq 2 \\ 5 - y_2 &\leq -7 \end{aligned}$$

Die linke Seite des Gleichungssystems ist die transponierte Matrix des originalen Problems, und die rechte Seite der Vektor c desselben. Die Koeffizienten der neuen Zielfunktion (hier 6 und 12) sind der ursprüngliche Vektor b.

Es gibt keine Vorzeichenbedingungen für die y's, daher gilt
$y_1 \leq 0, y_1 \geq 0, y_2 \leq 0, y_2 \leq 0$

Beispiel. für ein LP in allgemeiner Form

minimiere $3x_1 - x_2 - 4x_3$
Nb:

$$\begin{aligned} 5x_1 - 6x_2 + 7x_3 &= 9 \\ 6x_1 - 9x_2 - 2x_3 &\geq -7 \end{aligned}$$

Vzb: $x_1 \geq 0, x_2 \geq 0, x_2 \leq 0, x_3 \geq 0, x_3 \leq 0$

$f = y_1 \cdot \underbrace{1.Zeile}_{=9} + y_2 \cdot \underbrace{2.Zeile}_{=-7} \geq 9y_1 - 7y_2$, falls $y_2 \geq 0$.

$f = (5y_1 + 6y_2)w_1 + (-6y_1 - 9y_2)w_2 + (7y_1 - 2y_2)w_3 \leq c_1w_1 + c_2w_2 + c_3w_3$
mit $c_1 = 3$, $c_2 = 1$ und $c_3 = -4$.

Dies gilt nur, falls die folgenden Ungleichungen gelten:

$$\begin{aligned} 5y1 + 6y_2 &\leq 3 \\ -6y_1 - 9y_2 &= -1 \\ 7y_1 - 2y_2 &= -4 \end{aligned}$$

In der ersten Zeile ist ein $\leq$, weil $x_1 \geq 0$, und in den übrigen Zeilen ein $=$, weil $x_2 \leq 0, x_2 \geq 0$. Vzb: $y_1 \geq 0, y_1 \leq 0, y_2 \geq 0, y_2 \leq 0$

Wo kann man diese Vorzeichenbedingungen aus dem ursprï¿½nglichen Programm ablesen?
$=$ liefert keine Bedingung, also ≤ 0 und ≥ 0
$\geq$ liefert ≥ 0.

Fazit: Man kann alle LP in die Duale Form umformen.

Allgemein gilt: Das ursprüngliche Problem heißt "Primal", das neue Problem "Dual". Die allgemeine Konstruktion aus der allgemeinen Form des Primalen funktioniert wie folgt:

Primal	**Dual**
$\sum_{j=1}^{n} a_{ij}x_j = b_i$ Zeile i $1 \leq i \leq h$	Variable $y_1 \leq 0$ $y_1 \geq 0$
$\sum_{j=1}^{n} a_{ij}x_j \geq b_i$ $h+1 \leq i \leq m$ Zeile i	Variable $y_i \geq 0$
$x_j \geq 0$ und $x_j \leq 0$ Variable j $j = 1, .., l$	$\sum_{i=1}^{m} y_i a_{ij} = c_j$
$x_j \geq 0$ Variable j $j = l+1, .., n$	$\sum_{i=1}^{m} y_i a_{ij} \leq c_j$
min $c^T x$	max $y^T b$

Satz 1.4.1. *Das Duale des Dualen ist das Primale.*

Lemma 1.4.2. *Falls w eine zulässige Lösung im Primalen LP ist, und u eine im Dualen, dann gilt $c^T w \geq u^T b$.*

Beweis.

$$\begin{aligned} c^T w &= \sum_{j=1}^{l} c_j w_j + \sum_{j=l+1}^{n} c_j w_j \\ &\geq \sum_{j=1}^{l} \underbrace{(A_j^T u)}_{\sum_{j=1}^{m} u_i a_{ij}} w_j + \sum_{j=l+1}^{n} (A_j^T u) w_j \\ &= \sum_{i=1}^{m} u_i \underbrace{\left[\sum_{j=1}^{l} a_{ij} w_j + \sum_{j=l+1}^{m} a_{ij} w_j \right]}_{i-te Zeile von Aw (A^i w)} \end{aligned}$$

$c^T w \geq \sum_{i=1}^{h} u_i A^i w + \sum_{i=h+1}^{m} u_i A^i w$

Das u_i auf der linken Seite der Gleichung ist nicht vorzeichenbeschränkt, das auf der rechten Seite schon ($u_i \geq 0$). Weiterhin gilt für die linke Seite $A^i w = b_i$, und für die rechte Seite $A^i w \geq b_i$.

$\Rightarrow c^T w \geq \sum_{i=1}^{h} u_i b_i + \sum_{i=h+1}^{m} u_i b_i = u^T b$ □

Beispiel. Ernährungsproblem
gegeben:
m Nährstoffe
n Nahrungsmittel

a_{ij}	Anteil des i-ten Nährstoffes im Nahrungsmittel j
x_j	täglicher Verbrauch an Einheiten von Nahrungsmittel j, $j = 1, \ldots, n$
r_i	täglicher Bedarf am i-ten Nährstoff
c_j	Kosten pro Einheit am Nahrungsmittel j

Daraus ergibt sich das Lineare Programm

$min\ c^T x$	tägliche Kosten fürs Essen
$Ax \geq r$	täglichen Bedarf an allen Nährstoffen erfüllen
$x \geq 0$	

Das zugehörige duale LP (siehe Konstruktion Primal → Dual)

	Interpretation: Ernährung durch Pillen für einzelne Nährstoffe
$max\ y^T r$	y_i Kosten pro Einheit vom Nährstoff i
$A^T y \leq c$	Insgesamt sollen die Kosten für Pillen nicht höher sein
	als bei entsprechender herkömmlicher Ernährung
$y \geq 0$	

Satz 1.4.3. *Dualitätssatz*
Ein LP hat eine optimale Lösung gdw. sein duales Programm eine optimale Lösung hat und die Kosten beider sind gleich.

Beweis. (für die kanonische Form des Primalen)
Falls für irgendwelche zulässigen Lösungen w^* des Primalen
und u^* des Dualen gilt:

$$c^T w^* \quad = \quad u^{*T} b$$

dann müssen nach dem vorhergehenden Lemma beide optimal sein.

Betrachte Simplexalgorithmus beim Finden der optimalen Lösung des primalen LP.
Wir führen die Schlupfvariablen ein gemäss:

$$x_{n+i} \quad = \quad -b_i + \sum_{j=1}^{n} a_{ij} x_j,\ i = 1, .., m \tag{1.11}$$

das liefert schliesslich die Darstellung der Zielfunktion:

$$z \quad = \quad z^* + \sum_{k=1}^{n+m} \overline{c}_k x_k \tag{1.12}$$

mit $\overline{c}_j \geq 0\ \forall\, j$ und $\overline{c}_j = 0$, für Basisvariablen x_j

z^* ist optimaler Wert der Zielfunktion, also $z^* = c^T w^*$
Seien $u_i^* = \overline{c}_{n+i},\ i = 1, .., m$ Koeffizienten der Schlupfvariablen (3)

Wir wissen: $z = z^* + \overline{c}^T x$ nach (2)

andererseits $z = c^T x = z^* + \sum_{j=1}^{n} \overline{c}_j x_j + \sum_{i=1}^{m} \underbrace{u_i^*}_{=\overline{c}_{n+i}} \underbrace{(-b_i + \sum_{j=1}^{n} a_{ij} x_j)}_{=x_{n+i}}$ nach (1),(3)

damit ist $c^T x = \underbrace{z^* - \sum_{i=1}^{m} b_i u_i^*}_{\text{unabh. von x} \Rightarrow =0} + \sum_{j=1}^{n} (\overline{c}_j + \sum_{i=1}^{m} u_i^* a_{ij}) x_j \leftarrow$ muss für jeden

Vektor $x_1, \ldots, x_n$ gelten

also $\quad c^T w^* = z^* - b^T u^*$ $\square$

Korollar 1.4.4.

Genau *einer* der folgenden drei Fälle gilt für ein primal - duales Paar beim LP

(a) beide haben optimale Lösungen

(b) eines ist unbeschränkt (die Zielfunktion) und eines hat keine zulässige Lösung

(c) beide haben keine zulässigen Lösungen

Beweis.

(a) gilt nach dem Dualitätssatz

(b) Falls Zielfunktion im Primalen unbeschränkt, gilt wegen Lemma:
$u^T b \leq c^T w \quad \forall$ zulässige u im Dualen
$\forall$ zulässige w im Primalen
$\Rightarrow$ im Dualen existiert *keine* zulässige Lösung *und* Fall (b)

(c) kann auch auftreten. Beispiel als Übung

$\square$

Falls beide zulässige Lösungen haben, so haben beide optimale Lösungen!

Der Dualitätssatz liefert eine Methode zur *Verifikation* ob gegebene
$x^* = \begin{pmatrix} x_1^* \\ \vdots \\ x_n^* \end{pmatrix}$, $y^* = \begin{pmatrix} y_1^* \\ \vdots \\ x_m^* \end{pmatrix}$, optimale Lösungen eines LP und seines dualen sind:

gdw. $\quad Ax^* \leq b \quad \Rightarrow x^*$ zulässig *und* $A^T y^* \geq c \quad \Rightarrow y^*$ ist zulässig
$x^* \geq 0 \qquad\qquad\qquad\qquad\qquad y* \geq 0$

und $c^T x^* = y^{*T} b$

Es gilt sogar:

Satz 1.4.5. *(Komplementärer Schlupf)*
Sei P ein LP in allgemeiner Form, D sein duales Programm. Sei w zulässige Lösung von P, u eine von D.
Dann sind w,u optimal gdw.

$$(A^i w - b_i) u_i = 0, \; i = 1, .., m$$
und $\quad w_j (c_j - A_j^T u) = 0, \; j = 1, .., n$

Das heisst: Jede Nb des primalen oder entsprechende Vzb. des dualen LP muss gerade noch (mit =) erfüllt sein, und umgekehrt.

Damit ist die *Verifikation* ob zwei gegebene Lösungen w, u optimal sind, sehr leicht möglich.

Beweis.
Es gilt (i)

$$\underbrace{(A^i w - b_i)}_{\geq 0\ (1)}\ \underbrace{u_i}_{\geq 0,\ (2)} \quad \geq 0,\ i = 1, \ldots, m$$

$$\underbrace{w_j}_{\geq 0\ (2)}\ \underbrace{(c_j - A_j^T u)}_{\geq 0\ (1)} \quad \geq 0,\ j = 1, \ldots, n$$

da Nb. (1) bzw Vzb. (2) jeweils erfüllt sind.
Sei

$$\begin{aligned}
t &= \sum_{i=1}^{m} \underbrace{(A^i w - b_i) u_i}_{} + \sum_{j=1}^{n} \underbrace{w_j (c_j - A_j^T u)}_{} \\
t &= 0 \Leftrightarrow \qquad = 0\ \forall i \qquad\qquad = 0\ \forall j \quad \text{wegen } (i) \\
t &= -\sum_{i=1}^{m} b_i u_i + \sum_{j=1}^{n} w_j c_j + \underbrace{\sum_{i=1}^{m} u_i A^i w - \sum_{j=1}^{n} w_j A_j^T u}_{=0} \\
&= -u^T b + c^T w \\
&= 0 \overset{Dualitaetssatz}{\Longleftrightarrow} u,\ w \text{ optimal}
\end{aligned}$$

□

Anwendungsbeispiel: Kürzeste - Wege - Problem

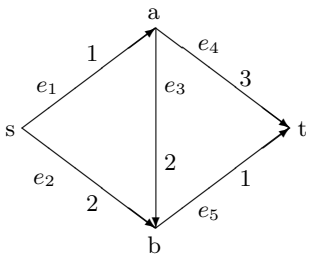
gegeben: gerichteter Graph $G = (V, E)$, $V = \{v_1, \ldots, v_n\}$, $E = \{e_1, \ldots, e_m\}$
Kosten der Kanten: $c_i \geq 0,\ i = 1, \ldots, m$,
$s \in V$ sei Quelle und $t \in V$ Senke des Graphen

gesucht: kürzester Weg von s nach t, repräsentiert als Kantenfolge $p = (e_{j_1}, \ldots, e_{j_k})$, mit $min\ c(p) = \sum_{j=1}^{k} c_{i_j}$

Definition 1.4.6. Inzidenzmatrix A von G
1 Zeile pro Ecke, 1 Spalte pro Kante

$$a_{ij} = \begin{cases} 1 & , \text{ falls } v_i \xrightarrow{e_j} \text{ d.h } \exists k \in \{1, .., |V|\} \text{ mit } e_j = (v_i, v_k) \\ -1 & , \text{ falls } v_i \xleftarrow{e_j} \text{ d.h } \exists k \in \{1, .., |V|\} \text{ mit } e_j = (v_k, v_i) \\ 0 & , \text{ sonst} \end{cases}$$

Beispiel.



liefert

	e_1	e_2	e_3	e_4	e_5
s	1	1	0	0	0
t	0	0	0	−1	−1
a	−1	0	1	1	0
b	0	−1	−1	0	1

Wir wollen das Flussproblem und das damit verknüpfte Kürzeste-Wege-Problem als lineares Programm formulieren. Seinen $e_1, \ldots, e_n$ die Kanten des Graphen und $c_1, \ldots, c_n$ deren zugehörige Kantengewichte. Zur Formulierung führen wir Variablen $f_1, \ldots, f_n$ ein, die den Fluss der zugehörigen Kante beschreiben sollen. Jede Kante hat eine Kapazität (also maximalen Fluss) von 1 und der Fluss soll von s nach t fließen. Um den kürzesten Fluss zu finden müssen wir also

$$c^T f \text{ minimieren.}$$

Ein Fluss ist zulässig, wenn in jedem Knoten (außer s und t) der einkommende Fluss gleich dem ausgehenden Fluss ist. Dies formulieren wir wie folgt:

$$Af = \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix}$$

mit

$$b_i = \begin{cases} 1 & \text{,wenn der zugehörige Knoten s ist} \\ -1 & \text{,wenn der zugehörige Knoten t ist} \\ 0 & \text{sonst.} \end{cases}$$

Die Vorzeichenbedingungen sind $f \geq 0$. Falls in einer Lösung alle $f_i \in \{0, 1\}$ sind und der Fluss den Wert 1 hat, so entspricht dies einem Pfad von s nach t in G. Obwohl wir kein ganzzahliges lineares Programm haben und damit nicht alle $f_i \in \{0, 1\}$ sein müssen, lässt sich zeigen, dass eine solche optimale Lösung existiert. Das duale Lineare Programm dazu sieht wie folgt aus:

$$\begin{gathered} \max\ y_s - y_t \\ A^T y = c \\ y \geq 0, y \leq 0 \end{gathered}$$

Eine zulässige Lösung entspricht hier einem Weg im Graphen und die y_i sind die negativen Kosten von i nach t. Aus den komplementären Schlupfbedingungen läßt sich eine Bedeutung für das Problem ableiten. Entweder $y_i^* - y_j^* = c_{ij}$ für Kante $e = (i, j)$ oder $f_{e^*} = 0$, was bedeutet, dass Kante e nicht zur Lösung hinzugenommen wird.

1.5 Die Ellipsoid-Methode

Lange Zeit war es ein offenes Problem, ob Lineare Programmierung in polynomieller Zeit (worst-case) möglich ist. Mit der Einführung der Ellipsoid-Methode bewies Leonid Khachiyan 1979, dass dieses möglich ist.

1.5.1 Bit-Komplexität

Was ist die Klasse P?
Die Klasse P ist die Klasse aller Probleme, die auf einer Turingmaschine oder einer Registermaschine im logarithmischen Kostenmaß in polynomieller Zeit lösbar sind.
Das bedeutet, dass ganze oder rationale Zahlen in binärer Form (oder einer anderen Basis) dargestellt werden. Die Problemgröße ist dann die Anzahl der Bits (bzw. Stellen), mit der die Eingabe dargestellt wird.

Definition 1.5.1. Sei $x \in \mathbb{Z}$, dann definieren wir die Funktion $gr : \mathbb{Z} \to \mathbb{N}$ wie folgt:

$$gr(x) = \begin{cases} \lceil \log(x) \rceil & |x| \geq 2 \\ 1 & x = 0, 1, -1 \end{cases}$$

Für $x = \frac{p}{q} \in \mathbb{Q}$ und p, q teilerfremd sei $gr : \mathbb{Q} \to \mathbb{N}$ definiert als

$$gr(x) = gr(p) + gr(q).$$

Falls $x = (x_1, \ldots, x_n)$ ein Tupel von Zahlen ist, dann definieren wir $gr : \mathbb{Q}^n \to \mathbb{N}$ als

$$gr(x) = \sum_{i=1}^{n} gr(x_i)$$

Man weiß, dass die Addition, die Subtraktion, die Multiplikation, die exakte Division in $\mathbb{Q}$, die ganzzahlige Division in $\mathbb{N}$ und die Funktionen $x \mapsto \lfloor \sqrt{x} \rfloor$ und $x \mapsto \lfloor \log_3(x) \rfloor$ in polynomieller Zeit berechenbar sind.

Satz 1.5.2. *Die Inversion von Matrizen ist in polynomieller Zeit möglich.*

Beweis. Übung □

Beispiel. Gaußscher Algorithmus
Der Gaußsche Algorithmus zum lösen linearer Gleichungssysteme benötigt für eine $n \times n$- Matrix (n Gleichugen mit n Variablen) $\Theta(n^3)$ arithmetische Operationen. Die Berechnung scheint in polynomieller Zeit zu laufen. Dies trügt aber, denn es können sich in jedem Eliminationsschritt die Anzahl der Bits jedes Koeffizientens verdoppeln, so dass am Ende deren Größe exponentiell in n ist. Man sieht also, dass der vorherige Satz nicht trivial ist (das Gaußverfahren kann auch zur Berechnung der inversen Matrix benutzt werden).

Korollar 1.5.3. *Es existiert ein Polynom p, so dass für jede reguläre Matrix $A \in \mathbb{Q}^{n \times n}$ gilt*

$$gr(A^{-1}) \leq p(gr(A))$$

Beweis. Das Korollar folgt unmittelbar aus 1.5.10. Wir wollen hier aber noch einen direkten Beiweis angeben, der diesen Satz nicht verwendet. Zunächst zeigen wir folgendes

Lemma 1.5.4. *Es existiert ein Polynom p_1, so dass für alle $B \in \mathbb{Q}^{n \times n}$ gilt*

$$gr(det(B)) \leq p_1(gr(B)).$$

Beweis. Nach der Leibniz-Formel kann die Determinante wie folgt berechnet werden:

$$det(B) = \sum_{\sigma \in S_n} \left(sgn(\sigma) \prod_{i=1}^{n} b_{i,\sigma i} \right).$$

Dann ist

$$|det(B)| \leq (max_{i,j}|b_{ij}|)^n n! \leq (2^{gr(B)})^n n^n.$$

Da wir uns nur für die Anzahl der Bits interessieren, mit der $det(B)$ dargestellt wird, wenden wir noch den Logarithmus an und erhalten:

$$gr(det(B)) \leq \log_2(2^{gr(B)})^n n^n = gr(B)n + n\lceil \log_2(n) \rceil.$$

Mit $n \leq gr(B)$ folgt dann

$$gr(det(B)) \leq 2(gr(B))^2 = p_1(gr(B)).$$

□

Die Inverse zu A berechnet sich nach der Cramerschen Regel wie folgt:

$$A^{-1} = (\bar{a}_{ij})_{1 \leq i,j \leq n}$$

mit $\bar{a}_{ij} = \frac{c_{ij}}{det(A)}$, wobei $c_{ij} = (-1)^{i+j} det(A \text{ ohne } i\text{-te Zeile und } j\text{-te Spalte})$. Also ist

$$\begin{aligned} gr(\bar{a}_{ij}) &= gr(c_{ij}) + gr(det(A)) \\ &\leq 2gr(det(A)) \\ \text{(wegen 1.5.4)} &\leq 2p_1(gr(A)). \end{aligned}$$

Dann gilt

$$\begin{aligned} gr(A^{-1}) &= \sum_{i,j} gr(\bar{a}_{ij}) \\ &\leq n^2 max_{i,j}(gr(\bar{a}_{ij})) \\ &\leq n^2 2p_1(gr(A)). \end{aligned}$$

□

Betrachte nun ein Lineares Programm in Standardform:

$$\begin{aligned} &\min c^T x \\ &Ax = b \\ &x \geq 0 \end{aligned} \tag{1.13}$$

mit $A \in \mathbb{Q}^{m \times n}$, $c \in \mathbb{Q}^n$ und $b \in \mathbb{Q}^m$. Dann ist

$$N = gr(A) + gr(b) + gr(c)$$

seine Größe.

Korollar 1.5.5. *Es existiert ein Polynom p, so dass für jedes Lineare Programm der Form 1.13 gilt, dass für jede zulässige Basislösung $a = \begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix}$ mit $a_i \in \mathbb{Q}$ ist*

$$gr(a) \leq L = p(N).$$

(Insbesondere mit $a_i = \frac{r_i}{s_i}$ ist dann $|r_i|, |s_i| \leq 2^L$).

Beweis. Die zulässige Basislösung ist von der Form $\bar{a} = B^{-1}b$ mit B Teilmatrix von A. a enthält dann alle Einträge von $\bar{a}$ und die zusätzlichen Nullen. Die Einträge von B^{-1} sind nach Korollar 1.5.4 polynomiell in $gr(A)$. □

Korollar 1.5.6.

a. *Sei x eine bfs(basic feasible solution) eines linearen Programms*

$$\begin{gathered} Ax = b \\ x \geq 0 \\ min\, c^T x \end{gathered}$$

dann gilt

$$\left| c^T x \right| \leq 2^{Ln}$$

wobei $L := p(n)$

b. *Seien x_1, x_2 bfs von diesem linearen Programm mit*

$$\exists k \in \mathbb{Z}, k2^{-2Ln} < c^T x_1, c^T x_2 \leq (k+1)2^{-2Ln}$$

Dann gilt:

$$c^T x_1 = c^T x_2$$

Beweis.

a. Sei x bfs und $x = \begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix}$ mit $a_i = \frac{r_i}{s_i}$ und $|r_i|, |s_i| \leq 2^L$. Weil $c^T x = \sum_{i=1}^{n} c_i a_i$, es folgt dann

$$\left| c^T \right| \leq \sum_{i=1}^{n} \underbrace{|c_i|}_{\leq N} \underbrace{|a_i|}_{2^L} \leq n \cdot N \cdot 2^L$$

b. Angenommen, dass $c^T x_1 \neq c^T x_2$. Wir setzen $x_1 = \begin{pmatrix} \frac{p_1}{a_1} \\ \vdots \\ \frac{p_n}{a_n} \end{pmatrix}$ und $x_2 = \begin{pmatrix} \frac{r_1}{s_1} \\ \vdots \\ \frac{r_n}{s_n} \end{pmatrix}$, wobei $p_i, q_i, r_i, s_i \in \mathbb{Z}$, und $|p_i|, |q_i|, |r_i|, |s_i| \leq 2^L$, aber

$$\left| c^T x_1 - c^T x_2 \right| = \underbrace{\frac{r}{\Pi_{i=1}^{n} q_i \cdot s_i}}_{\leq 2^{2nL}} \geq 2^{-2nL} \text{ mit } r \in \mathbb{Z}\, r \neq 0$$

also $c^T x_1 = c^T x_2$.

□

1.5.2 Erfüllbarkeit linearer Ungleichungen

Definition 1.5.7. Problem LI (engl. "linear inequalities"). Gegeben $A \in \mathbb{Q}^{m \times n}, b \in \mathbb{Q}^m$, die Frage ist, existiert x mit $Ax \leq b$? (d. h. gibt es eine zulässige Lösungen?)

Satz 1.5.8. *Falls es einen Polynomialzeit-Algorithmus für LI gibt, so gibt es auch einen für LP.*

Beweis. Sei LP o.B.d.A in Standardform und L wie oben im Korollar 1.5.3 definiert. Sei $\mathcal{A}$ ein Algorithmus für LI(in polyn. Laufzeit). Wir entwickeln einen Algorithmus für LP von polyn. Laufzeit plus polynomiell viele Orakelanfragen an $\mathcal{A}$, so dass der Algorithmus insgesamt in Polynomialzeit für LP ist, falls $\mathcal{A}$ ein Algorithmus für LI ist.

1. wende $\mathcal{A}$ an auf
$$\begin{gathered} Ax \geq b \\ Ax \leq b \\ x \geq 0 \end{gathered}$$
falls "nein", keine zulässige Lösung für LP, sonst:
2. wende $\mathcal{A}$ an auf
$$\begin{gathered} Ax \geq b \\ Ax \leq b \\ x \geq 0 \\ c^T \leq -2^{Ln} - 1 \end{gathered}$$
falls "ja", gib aus "Zielfunktion ist unbeschränkt", sonst:
3. LP hat dann eine optimale zulässige Basislösung x^*. Bestimme $k \in \{-2^{3Ln}, \ldots, 2^{3Ln}\}$ mit $k2^{-2Ln} \leq c^T x^* \leq (k+1)2^{-2Ln}$ durch Aufrufe von $\mathcal{A}$ mit
$$\begin{gathered} Ax \geq b \\ Ax \leq b \\ x \geq 0 \\ 2^{Ln} c^T x \leq a \end{gathered}$$
für verschiedene $a \in \mathbb{Z}$. Das können wir durch Binärsuche mit maximal $3Ln + 1$ Tests. Das Intervall ist so klein dass alle Basislösungen, für die Wert der Zielfunktion darin liegt, den gleichen Wert haben. Es bleibt noch
4. Basis zu bestimmen. Für $l = 1, 2, \ldots,$ prüfe ob
$$\underbrace{\begin{gathered} Ax \geq b \\ Ax \leq b \\ k \leq 2^{2Ln} c^T x \leq k+1 \\ x \geq 0 \\ x_l \leq 0 \\ x_j \leq 0 \,\text{für}\, j \in S(l) \end{gathered}}_{\text{höchstens } n \text{ Aufrufe von } \mathcal{A}}$$
wobei $S(l) = \{r \in \{1, 2, \ldots, l-1\} | \text{Antwort für r war "ja"}\}$. Wir rufen $\mathcal{A}$ auf, solange bis nur noch m x_i vorhanden, die nicht auf 0 gesetzt werden. Diese bilden die Basis. Es führt zu einer $m \times m$ Untermatrix $\widetilde{A}$, weil die Spalten den Basisvariablen entsprechen, liefert $\widetilde{A}^{-1} b$ die restlichen Einträge für die optimale bfs, wobei wir die Invertierung von $\widetilde{A}$ in polyn. Zeit berechnen können.

□

Poly. Zeit mit poly. viele Aufrufen des "Orakels" $\mathcal{A}$. Sprechweise: Solch eine Reduktion heißt "Turing-Reduktion", im Gegensatz: Beim Beweis von "NP-Vollständigkeit" benutzen wir "many-one-reduktion".

Definition 1.5.9. Problem LSI (engl. "linear strict inequalities"). Gegeben sei $A \in \mathbb{Z}^{m \times n}, b \in \mathbb{Z}^m$, die Frage ist: existiert ein $x \in \mathbb{R}$ mit $Ax < b$?

Satz 1.5.10. *LSI ist in poly. Zeit lösbar $\Longrightarrow$ LI ist in poly. Zeit lösbar*

Beweis über

Lemma 1.5.11. *$Ax \leq b$ (3) hat Lösung $\Longleftrightarrow$ $Ax < b + \begin{pmatrix} \varepsilon \\ \varepsilon \\ \vdots \\ \varepsilon \end{pmatrix}$ (4) hat eine Lösung,*

wobei $\varepsilon = 2^{-P(L)}$

Beweis.

$\Longrightarrow$: $\surd$

$\Longleftarrow$: Sei x_0 eine Lösung von (4), so dass

$$I = \{i | b_i \leq A_i^T x < b_i + \varepsilon\}$$

maximal viele Elemente hat. Dann gilt:

$$\forall j \, \exists \beta_{ji}, \, i \in I \text{ mit } A_j = \sum_{i \in I} \beta_{ji} A_i$$

Also alle übrigen Zeilenvektoren lassen sich als lineare Kombination von denen mit Index $\in I$ darstellen.

Denn: angenommen das ist falsch, dann existiert $A_j, j \notin I$, das linear unabhängig von $\{A_i | i \in I\}$ ist. Dann hat das Gleichungssystem

$$\begin{cases} A_i^T z = 0 \text{ für } i \in I \\ A_j^T z = 1 \text{ sonst} \end{cases}$$

eine Lösung z_0, dann ist $x_1 = x_0 + \lambda z_0$ Lösung von (4) für $|\lambda|$ hinreichend klein. Wir können dann $|\lambda|$ soweit erhöhen, dass I um j erweitert werden kann, d.h. $A_j^T \geq b_i$, aber Widerspruch zur Maximalität von I.

$\square$

Es existiert ein $I' \subseteq I$ für das gilt: A_i mit $i \in I'$ ist linear unabhängig und β_{ji} mit $j \in \{1, ..., m\}$ und $i \in I'$.

Dann sei

$$A_j = \sum_{i \in I'} \beta_{ji} A_i$$

$$\text{mit } I' = \{i_1, ..., i_k\}$$

und

$$\tilde{A} := \begin{pmatrix} A_{i1} \\ \vdots \\ A_{im} \end{pmatrix}$$

so gilt

$$\begin{pmatrix} \beta_{j1} \\ \vdots \\ \beta_{jm} \end{pmatrix} = \tilde{A}^{-1} \cdot A_j$$

$$j \in \{1, ..., n\}$$

Dann gilt nach Cramers Regel

$$\beta_{ji} = \frac{\alpha_{ji}}{d}$$

wobei

$$d = |det\tilde{A}|$$

$$gr(\alpha_{ji}), gr(d) \leq q(L) \text{ für ein Polynom q nach vorherigem Korrolar}$$

Sei ferner

$$\hat{x} = \tilde{A}^{-1} \cdot b$$

dann gilt

$$d(A_j^T \cdot \hat{x} - b_j) = \sum_{i \in I'} \alpha_{ji} \cdot A_i^T \hat{x} - db_j$$

$$= -\sum_{i \in I'} \alpha_{ji}(A_i^T \cdot x_0 - b_i) + d(A_j^T \cdot x_0 - b_j)$$

$$< \epsilon(\sum_{i \in I'} |\alpha_{ji}| + d) < \epsilon 2^{q(L)}(m+1) \leq 1$$

Falls nun

$$\epsilon \leq \frac{1}{2^{q(L)}(m+1)} \geq 2^{-p(L)} \text{ (für ein Polynom p)}$$

also

$$\epsilon \geq 2^{-p(L)}$$

dann gilt

$$\epsilon(\sum_{i \in I'} |\alpha_{ji}| + d) < 1$$

Außerdem gilt, dass $d \cdot A_j^T \cdot \hat{x}$ eine ganze Zahl ist. Nach Cramers Regel regelt folgt dann

$$\hat{x}_i = \frac{ganzeZahl}{d} \ \forall i$$

also

$$d(A_j^T \cdot \hat{x} - b_j) < 1$$

Da $d > 0$ und $(A_j^T \cdot \hat{x} - b_j) \geq 0$, muss $(A_j^T \cdot \hat{x} - b_j) = 0$ gelten. Daraus folgt nun

$$A_j^T \cdot \hat{x} = b_j$$

Damit haben wir eine Lösung von (3).

Beweis. Beweis des vorherigen Satzes

Statt LI überprüfe die Erfüllbarkeit von einem strikten Gleichungssystem

$$Ax < b + \begin{pmatrix} 2^{-p(L)} \\ \vdots \\ 2^{-p(L)} \end{pmatrix} \text{ (p nach Lemma)}$$

$$\Leftrightarrow 2^{p(L)} \cdot A_i^T x < 2^{p(L)} \cdot b_i + 1$$

Welches nun ein LSI ist, womit wir den Satz bewiesen haben. □

1.5.3 Ellipsoide

Definition 1.5.12. n-dimensionale Einheitskugel

Betrachten wir die n-dimensionale Einheitskugel. Sie ist definiert durch:

$$B_n = \{x \in \mathbb{R}^n | x^T \cdot x \leq 1\}$$

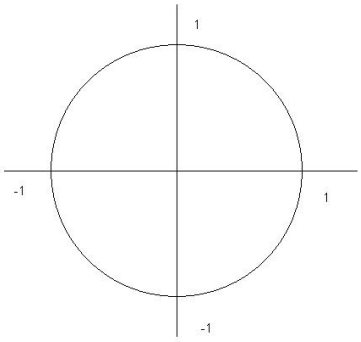


Abbildung 1.10: Einheitskreis

Definition 1.5.13. affine Transformation

Die affine Transformation ist definiert durch:

$$T : \mathbb{R}^n \to \mathbb{R}^n$$
$$x \to Q \cdot x + t$$

Wobei $Q \in \mathbb{R}^{n \times n}$ und regulär.
$t \in \mathbb{R}^n$.

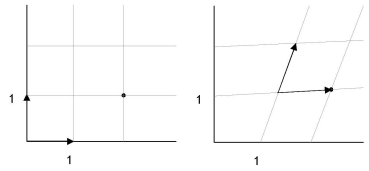


Abbildung 1.11: Affine Transformation

Definition 1.5.14. Ellipsoid
Mengen der Form $T(B_n)$ heißen Ellipsoide, wobei T eine affine Transformation ist.
Im 2-dimensionalen Raum nennt man einen Ellipsoid Ellipse.

Definition 1.5.15. alternative Ellipsoid Definition
Ein Ellipsoid ist eine Menge der Form

$$E = \{x \in \mathbb{R}^n | (x-t)^T C(x-t) \leq 1\}$$

wobei $t \in \mathbb{R}^n$ und $C \in \mathbb{R}^{n \times n}$ und C positiv definit.
(Positiv definit bedeutet $x^T Cx > 0$ für alle $x \neq 0$ und $x \in \mathbb{R}^n$
alternativ: Es existiert eine reguläre Matrix $Q \in \mathbb{R}^{n \times n}$ mit $C = QQ^T$)

Beispiel. 1

$$C = I_n$$
$$x^T Ix = x_1^2 + x_2^2 + ...x_n^2$$

Beispiel. 2

$$C = \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix} \text{ ist positiv definit}$$

dann

$$x \cdot C \cdot x^T = 2x_1^2 + x_2^2$$
$$\{x | x^T C x \leq 1\}$$

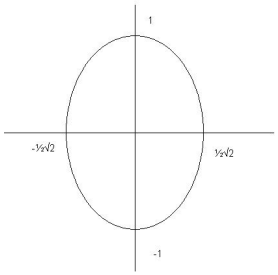


Abbildung 1.12: Ellipse

Lemma 1.5.16. *Sei $S \subset \mathbb{R}^n$ mit existierendem Volumen (Leqbreque-meßbar) und T eine affine Transformation mit Matrix Q.*

Dann ist $Vol(T(S)) = Vol(S) \cdot |detQ|$
(ohne Beweis)

Beispiel.

$$T(x) = \begin{pmatrix} 2 & 0 \\ 0 & 3 \end{pmatrix}$$

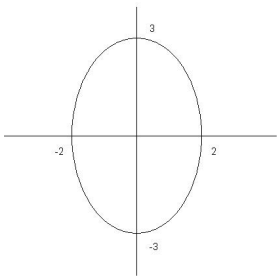


Abbildung 1.13: Transformierter Einheitskreis

Lemma 1.5.17. *Sei P ein konvexer Polyeder, definiert durch*

$$P = \{x \in \mathbb{R}^n | A \cdot x \leq b\}$$
$$A \in \mathbb{R}^{m \times n}$$
$$b \in \mathbb{R}^m$$

Falls das Innere von $P = \{x \in \mathbb{R}^n | A \cdot x < b\} \neq 0$, so existieren $n+1$ affin unabhängige Ecken von P.

Beispiel.

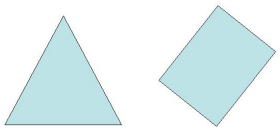


Abbildung 1.14: konvexe Polyeder

1.5.4 Der Ellipsoid Algorithmus

Eingabe $A \in \mathbb{Z}^{m \times n} \quad b \in \mathbb{Z}^m$

Ausgabe $x \in \mathbb{R}^n$ mit $Ax < b$ falls existent, ansonsten NEIN

(1) $j := 0 \quad t_0 := 0 \quad C_0 := n^2 2^{2L} I_n$
($L =$ Bitlänge der Eingabe $\quad I_n =$ n-dimensionale Einheitsmatrix)

(2) if ($At_j < b$) then return t_j

(3) if ($j > 16n(n+1)L$) then return NEIN

(4) Sei a die i-te Zeile von A mit $a^T t_j \geq b_i$

(5) $t_{j+1} := t_j - \frac{1}{n+1} \frac{C_j a}{\sqrt{a^T C_j a}}$

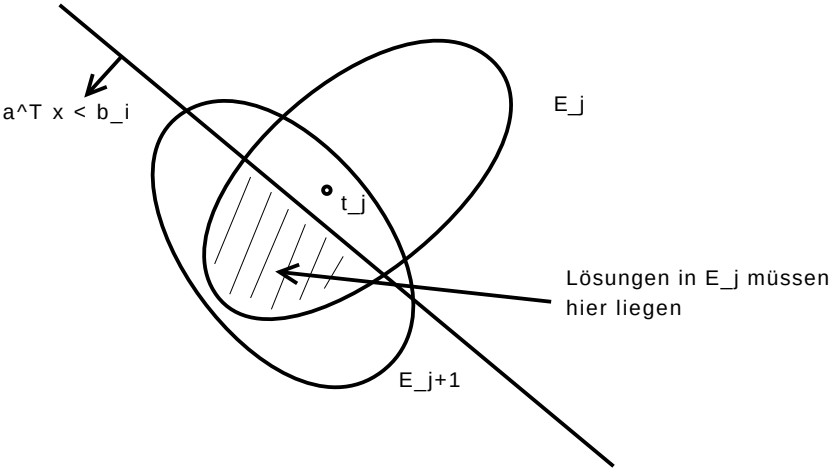
(6) $C_j + 1 := \frac{n^2}{n^2-1}[C_j - \frac{2}{n+1} \frac{(C_j a)(C_j a)^T}{a^T C_j a}]$

(7) $j := j + 1$, goto (2)

Wenn es für das Problem eine Lösung gibt, so ist sie nun in folgendem Ellipsoid enthalten:

$$E_j = \{x | (x - t_j)^T C_j^{-1} (x - t_j)\}$$

Idee: Definiere $E_j = \{x|(x-t_j)^T C_j^{-1}(x-t_j) \leq 1\}$



Eine Ellipse mit dem Mittelpunkt t_j, falls C_j von der Form $C_j = QQ^T$ (C regulär). C_j wird so konstruiert, dass es den schraffierten Bereich enthält.

1.5.5 Korrektheit

Satz 1.5.18. *Sei C_j eine symmetrische, positiv definite Matrix, $t_j \in \mathbb{R}^n$, $a \in \mathbb{R}^n$, $a \neq 0$, c_{j+1} und t_{j+1} wie in den Zeilen (5) und (6) des Ellipsoid-Algorithmus. Dann gilt:*

a) C_{j+1} ist symmetrisch positiv definit.

b) Für das „Halbellipsoid“ $E_j = \{x|(x-t_j)^T C_j^{-1}(x-t_j) \leq 0\}$ gilt, dass $E_j' \subseteq E_{j+1}$

c) $\frac{vol(E_{j+1})}{vol(E_j)} < 2^{-\frac{1}{2(n+1)}}$

Lemma 1.5.19. *Wir betrachten die Kugel B_n und wir betrachten das Ellipsoid $E = \{x|x \in \mathbb{R}^n$ und $(x-t)^T C^{-1}(x-t) \leq 1\}$ mit $t = \begin{pmatrix} \frac{-1}{n+1} \\ 0 \\ \vdots \\ 0 \end{pmatrix}$*

$$c = \begin{pmatrix} \frac{n^2}{(n+1)^2} & & & 0 \\ & \frac{n^2}{n^2-1} & & \\ & & \ddots & \\ 0 & & & \frac{n^2}{n^2-1} \end{pmatrix}$$

Dann gilt:

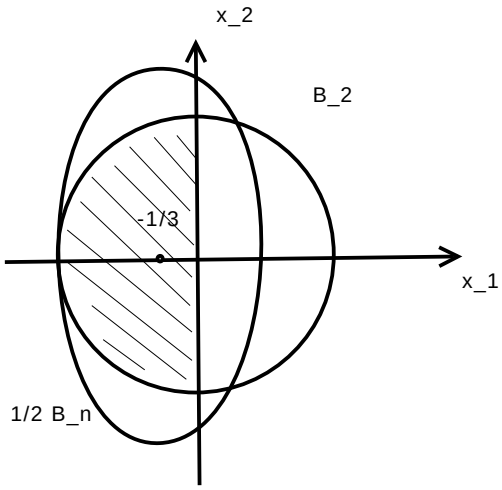
a) C ist symmetrisch positiv definit, d.h. von der Form $C = QQ^T$, Q regulär. Also ist E ein Ellipsoid.

b) Halbkugel $\frac{1}{2}B_n = \{x|X^T X \leq$ und $x_1 \leq 0\}$ ist Teilmenge von E.

c) $\frac{vol(E)}{vol(B_n)} < 2^{-\frac{1}{2(n+1)}}$

Beispiel zu Lemma 1.5.19 (n=2)

$C = \begin{pmatrix} \frac{4}{9} & 0 \\ 0 & \frac{4}{3} \end{pmatrix}$
$C^{-1} = \begin{pmatrix} \frac{9}{4} & 0 \\ 0 & \frac{3}{4} \end{pmatrix}$
$t = \begin{pmatrix} \frac{-1}{3} & 0 \end{pmatrix}$



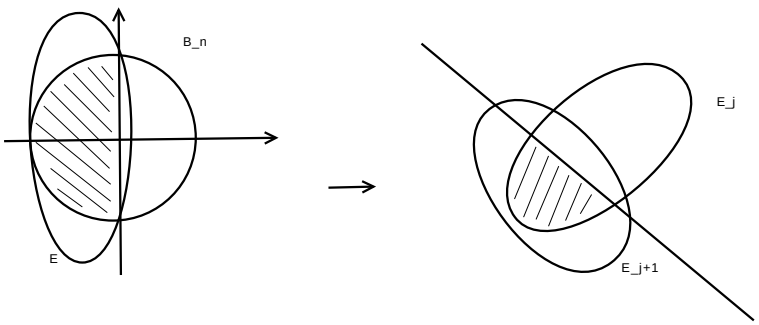
Halbachsen von E:
$(x-t)^T C^{-1}(x-t) = \frac{(x_1-t_1)^2}{\frac{4}{9}} + \frac{(x_2-t_2)^2}{\frac{4}{3}}$
$a = \sqrt{\frac{4}{9}} = \frac{2}{3} \qquad b = \sqrt{\frac{4}{3}} = \frac{2}{3}\sqrt{3}$

Das ist die normierte Situation des Satzes 1.5.18. Der Beweis von Lemma 1.5.19 ist eine Übung.

$\frac{vol(E)}{vol(B_2)} = \frac{\pi ab}{\pi} = \frac{4}{9}\sqrt{3} < 2^{\frac{-1}{6}}$

Lemma 1.5.20. *Sei $C_j \in \mathbb{R}^{n\times n}$ symmetrisch positiv definit und $0 \neq t \in \mathbb{R}^n$, t_{j+1}, c_{j+1} wie im Algorithmus. $\frac{1}{2}B_n$ und E wie im Lemma 1.5.19. Dann existiert eine affine Transformation T, so dass*

a) $T(B_n) = E_j = \{x|(x-t_j)^T C_j (x-t_j) \leq 1\}$
b) $T(E) = E_{j+1}$
c) $T(\frac{1}{2}B_n) = \{x|(x-t_j)^T C_j^{-1}(x-t_j) \leq 1$ *und* $a^T(x-t_j) \leq 0\} = E_j'$



Beweis von Lemma 1.5.20

Sei $C_j = QQ^T$ und es existiert eine Matrix $R \in \mathbb{R}^{n\times n}$ mit $RR^T = I$ und $Q^T a$ mit $R^T Q^T a = \begin{pmatrix} \|Q^T a\| \\ 0 \\ \vdots \\ 0 \end{pmatrix}$. Wir definieren $T(x) = QRx + t_j \qquad x \in \mathbb{R}^n$
$T^{-1}(x) = R^{-1}Q^{-1}(x - t_j)$.

a) Als Übung

b) Es ist

$$\begin{aligned}
C_{j+1} &= \frac{n^2}{n^2-1}\left(C_j - \frac{2}{n+1}\frac{C_j a a^T C_j^T}{a^T C_j a}\right) \\
&= \frac{n^2}{n-1}\left(C_j - \frac{2}{n+1}\frac{QRR^TQ^T aa^T QRR^TQ^T}{a^T QRR^TQ^T a}\right) \\
&= \frac{n^2}{n-1}\left(C_j - \frac{2}{n+1}\frac{QR\begin{pmatrix}\|Q^Ta\|^2 & & 0\\ & 0 & \\ 0 & & \ddots\end{pmatrix}R^TQ^T}{\|Q^Ta\|^2}\right) \\
&= \frac{n^2}{n-1}\left(QRI_nR^TQ^T - \frac{2}{n+1}QR\begin{pmatrix}1 & & 0\\ & 0 & \\ 0 & & \ddots\end{pmatrix}R^TQ^T\right) \\
&= \frac{n^2}{n^2-1}QR\begin{pmatrix}\frac{n-1}{n+1} & & 0\\ & 1 & \\ 0 & & \ddots\end{pmatrix}R^TQ^T \\
&= QR\begin{pmatrix}\frac{n^2}{(n+1)^2} & & & 0\\ & \frac{n^2}{n^2-1} & & \\ & & \ddots & \\ 0 & & & \frac{n^2}{n^2-1}\end{pmatrix}R^TQ^T \\
&= QRCR^TQ^T
\end{aligned}$$

Also

$$\begin{aligned}
x - t_{j+1} &= x - t_j + \frac{QRR^TQ^Ta}{(n+1)\sqrt{a^TQRR^TQ^Ta}} \\
&= x - t_j + QR\frac{\begin{pmatrix}\|Q^Ta\|\\ 0\\ \vdots\\ 0\end{pmatrix}}{(n+1)\|Q^Ta\|} \\
&= QR(T^{-1}(x) - t_j) \longleftarrow \textit{aus Lemma 1.5.19}
\end{aligned}$$

Deswegen

$$\begin{aligned} T(E) &= \{T(x)|(x-t)^T C^{-1}(x-t) \leq 1\} \\ &= \{x|(T^{-1}(x)-t)^T C^{-1}(T^{-1}(x)-t) \leq 1\} \\ &= \{x|(x-t_{j+1})^T (Q^{-1})^T R C^{-1} R^{-1} Q^{-1}(x-t_{j+1}) \leq 1\} \\ &= \{x|(x-t_{j+1})^T C_{j+1}^{-1}(x-t_{j+1}) \leq 1\} \\ &= E_{j+1} \end{aligned}$$

c) Als Übung

Beweis von Satz 1.5.18

(a) nach *Lemma 1.5.20 (b)* : $T(E) = E_{j+1}$
nach *Lemma 1.5.19 (a)* : $E = T'(Bn)$ für eine affine Transformation T'

also gilt: $E_{j+1} = T \circ T'(Bn)$
ist also ein Ellipsoid für alle j (E_0 ist sowieso einer)

(b)

$$\frac{1}{2}E_j = T(\frac{1}{2}Bn) \underbrace{<}_{\text{Lemma 1.5.19 (b)}} T(E) \underbrace{=}_{\text{Lemma 1.5.20(b)}} E_{j+1}$$

(c)

$$\frac{Vol(E_{j+1})}{Vol(E_j)} = \frac{Vol(T(E))}{Vol(T(Bn))} \underbrace{=}_{\text{Lemma ??}} \frac{|det(QR)| * (Vol(E))}{|det(QR)| * Vol(Bn} \underbrace{=}_{\text{Lemma 1.5.19 (c)}} 2^{-\frac{1}{2(n+1)}}$$

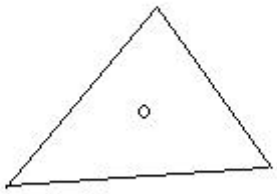
Lemma 1.5.21. *Falls ein LSI - System der Größe L eine Lösung hat, dann hat die Menge der Lösungen in der Kugel* $||x|| \leq n * 2^L$ *ein Volumen* $\geq 2^{-3n*L}$

$A * x < b$ hat eine Lösung => $A * x, x < \begin{pmatrix} 2^L \\ \vdots \\ 2^L \end{pmatrix}$ hat eine

=> Polyeder $A * x \leq b, x \leq \begin{pmatrix} 2^L \\ \vdots \\ 2^L \end{pmatrix}$ hat einen inneren Punkt

=> Polyeder hat $n+1$ affin unabhängige Ecken $v_0, ..., v_n$.

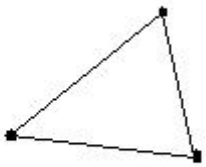
Erläuterung:

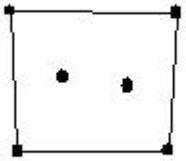


Im 2-dimensionalen Raum brauchen wir mindestens 3 linear unabhängige Ecken, im 3-dimensionalen 4 u.s.w.

Definition 1.5.22. Eine konvexe Hülle von $v_0, ..., v_n$ $CH(v_0, ..., v_n)$ ist die kleinste konvexe Menge, die $v_0, ..., v_n$ enthält.

$$CH(v_0, ..., v_n) = \{\lambda_0 v_0 + ... + \lambda_n v_n | \sum \lambda_i = 1, \lambda_i > 0\}$$





Alle inneren Punkte einer konvexen Hülle $CH(v_0, .., v_n$ sind Lösungen von $A * x < b$ in der Kugel $||x|| < n * 2^L$.

Aus der Geometrie wissen wir, dass das Volumen der konvexen Hülle wie folgt berechnet wird:

$$\frac{1}{n!}|det \begin{pmatrix} 1 & ... & 1 \\ v_0 & ... & v_0 \end{pmatrix}|$$

$$\text{jedes } v_i = \frac{U_i}{D_i} \text{und } |D_i| \leq 2^L$$

$$=> Vol_{CH} \geq \frac{1}{n! \prod(D_i)} \geq \frac{1}{n! 2^{Ln}} \underbrace{>}_{n! < n^n = 2^{n*logn}} 2^{-(n \underbrace{logn}_{\leq L} + Ln)} > 2^{3nL}$$

Satz 1.5.23. *Der Ellipsoid - Algorithmus entscheidet (korrekterweise) ob ein LSI eine Lösung hat.*

Beweis. Falls t_j in Schritt (2) zurückgegeben wird, dann ist t_j eine Lösung.
Angenommen, dass *"nein"* zurückgegeben wird, das System aber doch eine Lösung hat.
Nach 1.5.20 existiert eine Lösungsmenge in

$$E_0 = \{x \in |R^n, \underbrace{x^i C_0^{-1} x}_{\frac{||x||}{(n 2^{2L})^2}}\} \leq 1 \text{mit} Volumen \geq 2^{-3nL}$$

nach 1.5.18 (b) : $S \subset E_j$ für $j = 0, ..., k = 16n(n+1)L$
nach 1.5.18 (c) :

$$vol(E_k) < vol(E_0) * 2^{-\frac{k}{2(n+1)}} < \underbrace{(\overbrace{2n^2}^{2^{n*log(n)+n}} 2^{2L})^n}_{\text{Volumen des Würfels der } E_0 \text{ enthält}} 2^{-\frac{16n(n+1)L}{2(n+1)}} < 2^{-3nL}$$

□

also ist $vol(S) < 2^{-3nL}$ also gibt es nach 1.5.20 keine Lösung, was aber ein Widerspruch zur Annahme ist.

1.5.6 Arithmetische Genauigkeit

Problem: Die Wurzel in Zeile(5) kann im Bitmodell nicht genau berechnet, sondern muss approximiert werden aof endlich viele Bits.
Wieviel bleibt von der allgemeinen korrektheit? (geht das überhaupt)

Beweis. Zeilen (5) und (6):
$t_{j+1} = F(t_j, C_j)$
$C_{j+1} = G(t_j, C - j)$
Betrachte die Folge $E_0 = (t_0, C_0), E_1 = (t_1, C_1), ...$
Gemeint ist der Ellipsoid $x|(x - t_0)^t C_0^{-1}(x - t_0) \leq 1$
jetzt mit endlicher Genauigkeit:

$$t'_{j+1} = \overbrace{\hat{F}}^{F \text{ mit Rundungsfehler}} \underbrace{(t'_j, C'_j)}_{\text{durch Approximationsalgorithmen berechnet}}$$

$C'_{j+1} = \hat{G}(t'_j, G'_j)$

Ellipsoide :
$E'_j = (t'_j, C'_j(1 + \delta)^j)$ Modifikation des Algorithmus: Matrix wird in jedem Schritt um $1 + \delta$ vergrößert für ein geeignetes δ
=> Das Volumen vergrößert sich bei jedem Schritt um $(1 + \delta)^n$ um zu garantieren, dass E'_j für alle j trotz der Rundungsfehler noch die gesamte Lösungsmenge S enthält.
Funktioniert mit:
$k = 32n(n + 1)L)$
$\delta = \frac{1}{kn}$
$P = Ckn^2$ für hinreichend große C.

□

Satz 1.5.24. *Der Ellipsoid Algorithmus hat Polynomielle Laufzeit.*

1.6 Varianten

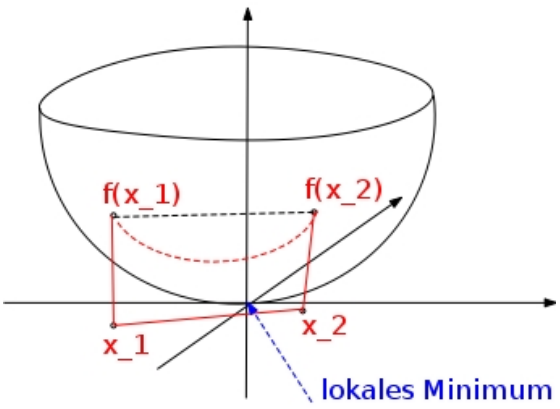
Der Ellipsoid-Algorithmus von 1979 ist nicht wirklich von praktischer Bedeutung beim Lösen von LP. Er zeigt aber, dass LP in polynomieller Zeit lösbar ist.
Ein anderer Algorithmus wurde 1984 von Kharmarkar entwickelt. Dieser Algorithmus durchsucht auch Punkte im Inneren des Polyeders, daher der Name "Innere-Punkt-Methode".
Die zur Zeit schnellsten LP-Löser sind meist eine Kombination vom Simplex-Algorithmus und der Inneren-Punkt-Methode.
Der Simplex-Algorithmus hat im Mittel über alle Eingaben eine gute Laufzeit und eine lineare Anzahl von Pivotschritten (gemäß Borgwardt und Smale).
Der Ellipsoid-Algorithmus wird auch für die **konvexe Optimierung** verwendet.

1.6.1 Konvexe Optimierung

Bei der konvexen Optimierung geht es um Probleme der Form

$$\begin{array}{ll} \text{minimiere} & f_0,\ x \in \mathbb{R}^n \\ \text{Nb} & f_i(x) \leq 0, i = 1, \ldots, m \\ & Ax = 0 \end{array}$$

$f_0, f_1, \ldots, f_m$ sind konvexe Funktionen, z.B. $f(x, y) = x_1^2 + x_2^2$

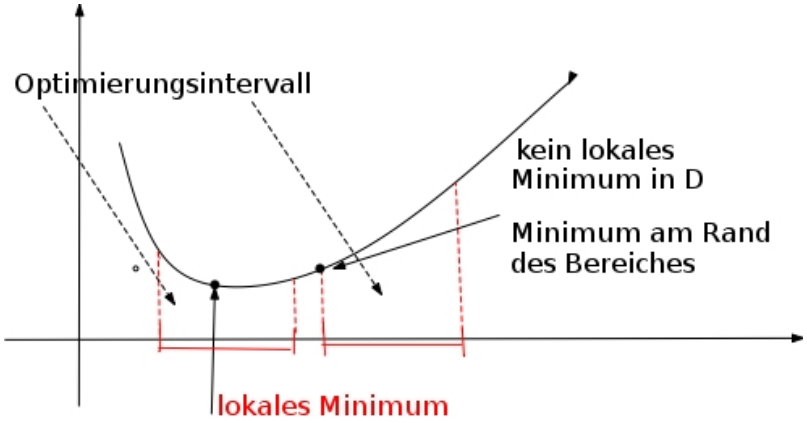


f heißt konvex von $\mathbb{R}^n \longrightarrow \mathbb{R}$, wenn gilt
$f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda) f(y), \forall x, y \in D$ und $\lambda \in [0, 1]$, wobei D (der Definitionsbereich) konvex ist.

Da alle linearen Funktionen konvex sind, ist LP ein Spezialfall von konvexer Programmierung.
Es gibt höchstens EIN lokales Minimum von f_0, und dies ist zugleich das globale Minimum.
Das Minimium hängt vom Definitionsbereich ab. Lösungen liegen am Rand der Menge

der zulässigen Lösungen.

Beispiel für ein 2D konvexes Programm:



Die Ellipsoid-Methode ist anwendbar und liefert einen Polynomzeit-Algorithmus für konvexes Programmieren unter gewissen Voraussetzungen an die Komplexität von $f_0, f_1, \ldots, f_m$.

1.6.2 Quadratische Optimierung

Bei der quadratischen Optimierung geht es um Probleme der Form

$$\begin{array}{ll} \text{minimiere} & x^T Q x + c^T x \\ \text{Nb} & Ax \leq b \end{array}$$

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \ldots \\ x_n \end{pmatrix}$$

z.B. $x_1^2 + 2x_2^2 + x_1 + 3x_1x_2$, $\qquad Q = \begin{pmatrix} 1 & 3 \\ 0 & 2 \end{pmatrix}$, $\qquad c = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$

Die quadratische Optimierung ist NP-schwer.

Kapitel 2

Das Lösen schwerer Probleme mit Hilfe von LP

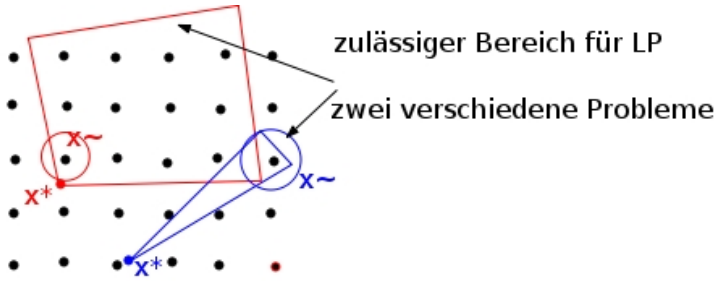
Wir haben auf dem 1. Übungsblatt gesehen, dass SAT und CLIQUE (beide sind NP-vollständig) in polynomieller zeit auf lineare Programme reduzierbar sind, wobei allerdings nach der optimalen *ganzzahligen* Lösung gesucht wird. Das nennt man ganzzahliges lineares Programmieren (ILP).
Damit ist auch ILP NP-schwer. Sogar bei Einschränkung auf Lösungen, deren Koordinaten 0 oder 1 sind, ist das ganze NP-schwer. Dieses Programm heißt dann "0/1-LP".

2.1 lP-Relaxierung

gegeben: ein ILP-Problem

Idee: Löse das LP in polynomieller Zeit ohne Einschränkung auf ganze Zahlen und man erhält das reellwertige Optimum $x*$.
Dieses ist eine approximative Lösung für ILP.

Aber: wie gut ist diese Methode, die die Lösung in polynomieller Zeit berechnet? Ist die Lösung optimal? Unterscheidet sich die Lösung maximal um den Faktor 2 von der optimalen Lösung?
Das muss noch geklärt werden, aber auf jeden Fall ist $f(x*)$ eine untere Schranke für das ganzzahlige Optimum.



$x*$ ist die reellwertige Lösung, und $\hat{x}$ die ganzzahlige. Wie man sehen kann, ist die rote reellwertige Lösung ziemlich dicht an der gesuchten, ganzzahligen Lösung, aber die blaue reellwertige Lösung überhaupt nicht.

2.1.1 Beispiele

Knapsack-Problem

Beispiel. Das Knapsack-Problem (NP-schwer)

Eingabe:

$$\begin{array}{ll} b \in \mathbb{N} & b \text{ ist das maximal tragbare Gewicht im Rucksack} \\ w_1, w_2, \ldots, w_n & w_i \in \mathbb{N} \\ c_1, c_2, \ldots, c_n & c_i \in \mathbb{N} \end{array}$$

Finde:

$$\begin{array}{l} T \subseteq \{1, \ldots, n\} \\ \text{maximiere } \sum\limits_{i \in T} c_i \\ \text{wobei } \sum\limits_{i \in T} w_i \leq b \end{array}$$

Sogar die vereinfachte Version: $w_1 = w_2 = \cdots = w_n = 1$ ist NP-schwer.

Das läßt sich als 0/1-LP formulieren:

Variablen: $x_1, x_2, \ldots, x_n$, $x_j = 1$ bedeutet: $j \in T$ (= im Rucksack)

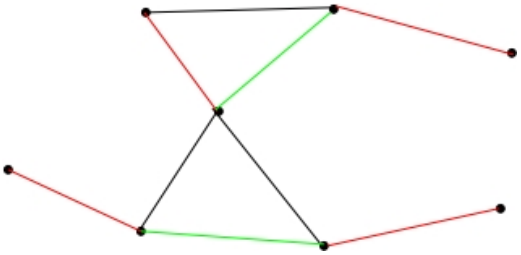
$$\begin{array}{lll} \text{maximiere } c^T x & & \\ \text{Nb} & \sum\limits_{i=1}^{n} w_i x_i \leq b & \\ & x_i \in \{0,1\} \longrightarrow & x_i \in [0,1] \\ & & x_i \geq 0 \\ & & x_i \leq 1 \end{array}$$

`Schnittmengenverfahren:` Für ein Problem werden zusätzliche Nebenbedingungen mit in das LP aufgenommen.

Matching

`Matching (Paarung)`: Eine Teilmenge der Kanten eines Graphen so auswählen, dass keine zwei Kanten einen gemeinsamen Endpunkt haben.

Beispiel für ein Matching:



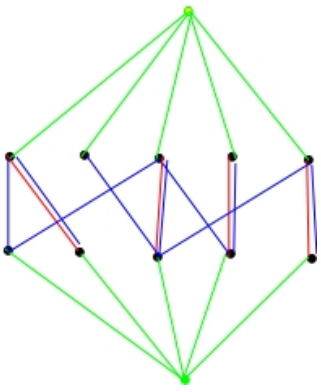
Grün und rot sind jeweils korrekte Matchings. Grün stellt ein maximales Matching dar, das heisst, dass man keine Kante mehr hinzufügen kann, ohne die Matching-Bedingung zu verletzten. Rot ist hingegen das eigentlich gesuchte, **größte** Matching.

Ein **perfektes** Matching ist gegeben, wenn alle Knoten daran beteiligt sind. Rot ist auch ein perfektes Matching.

Ein **bipartiter Graph**: Ein ungerichteter Graph $G = (V, E)$ heisst bipartit, wenn die Knotenmenge $V = L \cup R$ in zwei disjunkte Teilmengen L und R unterteilt werden kann, so dass für jede Kante (u, v) gilt $u \in L$ und $v \in R$.

Ein **bipartites Matching** ist ein Matching auf einem bipartiten Graphen.

Beispiel für ein bipartites Matching:



Heiratsvermittlung: Die oberen Punkte sind Frauen, die unteren Punkte sind Männer. Die blauen Linien stehen für Interesse an einem Vertreter des anderen Geschlechts. Der Anbieter möchte seinen Gewinn maximieren, und muss daher maximal viele Paare matchen. Da Hochzeit monogam ist, kann offiziell immer nur ein Mann mit einer Frau gematcht werden. Die roten Linien zeigen das letztendliche Matching.
Man löst dieses Problem, indem man eine künstliche Quelle und Senke einführt (grüne Punkte), und diese mit allen Punkten einer Gruppe verbindet (grüne Linien). Man lößt das Maximaler-Fluss-Problem mit dieser Eingabe und gibt jeder Kante den Fluss 1,

und bekommt automatisch auch die Lösung für das Matching. Die Laufzeit ist $O(n^3)$.

Schwerstes Matching

Beispiel. Schwerstes Matching

Zusätzlich zum normalen Matching-Problem werden noch Kantengewichte $w_1, w_2, \ldots, w_n$ berücksichtigt.

gegeben: Graph $G = (V, E)$
gesucht: ein Matching, dass die Summe der Kantengewichte maximiert.

Auch dieses Problem läßt sich als 0/1-LP formulieren:

Pro Kante e gibt es die Variable x_e.

$$\begin{array}{ll} \text{maximiere} & \sum_{e \in E} W_e x_e \\ \text{Nb} & \sum_{e \in E(v)} x_e \leq 1 \qquad \forall v \in V,\ x_e \in \{0,1\} \\ & E(v)\text{: die zu v inzidenten Kanten} \end{array}$$

MAXSAT

Beispiel. MAXSAT

SAT
gegeben: eine Boolesche Formel in KNF: $\phi = c_1 \wedge c_2 \wedge \cdots \wedge c_m$, wobei c_i sogenannte Klauseln von der Form $c_i = (l_{i1} \vee l_{i2} \vee \cdots \vee l_{ik})$ sind. Die l_{ik} wiederum sind einfache oder negierte Variablen $x_1, x_2, \ldots, x_n$.

Frage: Gibt es eine erfüllende Belegung $\mu : \{x_1, \ldots, x_n\} \longrightarrow \{W, F\}$, d.h. ϕ wird mit dieser Belegung ausgewertet, so dass $\phi = 1$ ist ?

MAXSAT:

gegeben ist ϕ in KNF.
gesucht: eine Belegung, die die größtmögliche Anzal von Klauseln auf 1 setzt.

SAT ist ein NP-vollständiges Entscheidungsproblem. MAXSAT ist noch schwerer als SAT. SAT ist auf MAXSAT in polynomieller Zeit reduzierbar, daher ist MAXSAT NP-schwer. In NP (= NP-schwer) liegen nur Entscheidungsprobleme.

MAXSAT≥kSAT
MAXSAT eingeschränkt auf auf Formeln mit $k_i \geq k$ für $i = 1, \cdots m$, dass heisst mindestens k verschiedene Literale pro Klausel.

randomisierter Algorithmus (RandSAT)
für i = 1 , ... , n {
belege x_i zufällig mit 1 oder 0 (mit Wahrscheinlichkeit 0.5)
}(Die so entstehende Belegung nennen wir: $\mu : \{x_1, \cdots ; x_n\} \to \{0, 1\}$) return μ

Satz 1:
Die erwartete Anzahl an erfüllten Klauseln bei RandSAT ist $\geq (1 - \frac{1}{2^k})m$
Damit ist RandSAT ein randomisierter Algorithmus für MAXSAT≥kSAT, wobei die erwartetet Approximationsgüte $\geq 1 - \frac{1}{2^k}$ ist.
Güte = $\frac{\text{Grösse der vom Algorithmus zurückgegebenen Lösung}}{\text{Grösse der optimalen Lösung}}$

Beweis. Sei X_i eine Zufallsvariable, dass deine zufällig gezogene Folge $(z_1, \cdots, z_n) \in \{0, 1\}^n$ c_i wahr macht.

$$X = \sum_{i=1}^{m} X_i \text{ (wie viele Klauseln werden wahr gemacht)}$$

Erwatungswert:

$$E(X) = \sum_{i=1}^{m} E(X_i)$$

$$E(X_i) = Pr(X_i = 1) = 1 - Pr(X_i = 0) \geq 1 - 2^{-k_i} \geq 1 - 2^{-k}$$

Damit ist

$$E(X) \geq m(1 - 2^{-k}) \geq OPT(1 - 2^{-k})$$

wobei OPT die maximale Anzahle an Klauseln ist die erfüllt werden können.

□

ILP für MAXSAT

Variable: $x_1, \cdots, x_n, z_1, \cdots, z_m$
wobei x_i die Variablen der Boolschen Formel sind und z_i bedeuted, dass die i-te Klausel wahr ist.
Zielfunktion:

$$\text{maximiere } \sum_{i=1}^{m} z_i$$

Nebenbedingungen:

$$\sum_{x \in V_j^+} X + \sum_{x \in V_j^-} (1 - x) \geq z_j \ \forall j \in \{1, \cdots, m\}$$

wobei V_j^+ die Menge der nicht negierten Variablen in c_j und V_j^- die Menge der negierten Variablen in c_j

$$z_j \in \{0, 1\}$$
$$x_i \in \{0, 1\}$$

LP - Relaxierung
Das LP ist das gleiche wie das ILP gerade, ausser dass bei den Nebenebedingungen gilt:

$$0 \leq z_j \leq 1 \; \forall \; j \in \{1, \cdots, m\}$$
$$0 \leq x_i \leq 1 \; \forall \; i \in \{1, \cdots, n\}$$

Dieses LP liefert eine *fraktionale* Lösung, dass heisst sie ist nicht unbedingt ganzzahlig.

Idee: zufälliges Runden auf $\{0, 1\}$- Lösung

randomisierter Algorithmus (RandRundMAXSAT)

1. - finde die optimale Lösung $(x_1^*, \cdots, x_n^*, z_1, \cdots, z_m)$ der LP Relaxierung
2. - für $i = 1, \cdots, n$ wähle $x_i = 1$ mit Wahrscheinlicheit x_i^*
3. - return $(x_1, \cdots, x_n)$ als Belegung

Satz 2:
Der Algorithmus RandRundMAXSAT ist ein randomisierter Approximationsalgorithmus für MAXSAT mit erwarteter Güte $\geq 1 - e^{-1}$.

Beweis. Sei X_i eine Zufallsvariable, dass deine zufällig gezogene Folge $(z_1, \cdots, z_n) \in \{0, 1\}^n$ c_i wahr macht.

$$X = \sum_{i=1}^{m} X_i \text{ (wie viele Klauseln werden wahr gemacht)}$$

Erwatungswert:

$$E(X) = \sum_{i=1}^{m} E(X_i)$$
$$E(X_i) = Pr(X_i = 1) = 1 - Pr(X_i = 0)$$

mit

$$Pr(X_j = 0) = \prod_{i: x_i \in V_j^+} (1 - X_i^*) \prod_{i: x_i \in V_j^-} X_i^*$$

und der Aussage:
für $t_1, \cdots, t_m, t \in \mathbb{R}_{\geq 0}$ mit $\sum\limits_{i=1}^{k} t_i = t$ gilt : $\prod\limits_{i=1}^{k} (1 - t_i) \leq (1 - \frac{t}{k})^k$ und der Nebenbedingung: $\sum\limits_{i: x_i \in V_j^+} X_i^* + \sum\limits_{i: x_i \in V_j^-} (1 - X_i^*) \geq z_j \; \forall \; j \in \{1, \cdots, m\}$ ergibt sich:

$$Pr(X_j = 0) \leq (1 - \frac{z_j}{k_j})^{k_j} \; \forall \; j \in \{1, \cdots, m\}$$

$$\Rightarrow Pr(X_j = 1) \geq 1 - (1 - \frac{z_j}{k_j})^{k_j} \geq (1 - (1 - \frac{1}{k_j})^{k_j}) z_j \geq (1 - e^{-1}) z_j$$

also ergibt sich:

$$E(X) = \sum_{j=1}^{m} Pr(X_j = 1) \geq (1 - e^{-1}) \sum_{j=1}^{m} z_j \geq (1 - e^{-1}) OPT$$

□

Satz 3:
Wendet man RandRundMAXSAT und RandSAT auf eine Formel φ an und nimmt diejenige Lösung, die mehr Klauseln erfüllt, so ist dies ein randomisierter Algorithmus mit erwarteter Güte $\geq \frac{3}{4}$

Beweis. Sei n_1 Anzahl der erfüllenden Klauseln bei RandSAT
Sei n_2 Anzahl der erfüllenden Klauseln bei RandRundMAXSAT
Wir wissen schon:

$$n_2 \geq \sum_{j=1}^{m} (1 - (1 - \frac{1}{k_j})^{k_j}) z_j$$

ausserdem wissen wir:

$$n_1 \geq \sum_{j=1}^{m} (1 - 2^{-k_j})$$

da $z_j \leq 1$ gilt:

$$n_1 \geq \sum_{j=1}^{m} (1 - 2^{-k_j}) z_j$$

also gilt:

$$\max(n_1, n_2) \geq \frac{n_1 + n_2}{2} \geq \frac{1}{2} \sum_{j=1}^{m} \underbrace{[(1 - 2^{-k_j}) + 1 - (1 - \frac{1}{k_j})^{k_j}]}_{\geq \frac{3}{2} \ \forall k_j \in \mathbb{N}} z_j$$

und

$$\sum_{j_1}^{m} z_j \geq OPT$$

$$\Rightarrow \max(n_1, n_2) \geq \frac{3}{4} OPT$$

□

2.1.2 Minimum Set Cover und die primal duale Methode

Problem: MSC (Minimum Set Cover)
Eingabe: $S = \{1, \cdots, n\}, F = \{S - 1, \cdots, S_m\}$ mit $S_i \subseteq S$ und $\bigcup_{i=1}^{m} S_i = S$
Ausgabe: $F' \leq F$ mit $\bigcup_{S' \in F'} S' = S$ und $|F'|$ minimal

Dieses Problem ist NP -schwer. Es gibt jedoch einen einfachen Greedy - Approximationsalgorithmus mit Approximations - Güte $O(\log n)$

Wir Formen unser Prolem zunächst in ein ILP um:
Zielfunktion:

$$\text{minimiere } \sum_{i=1}^{m} x_i$$

unter den Nebenbedingungen:

$$\sum_{i:s \in S} x_i \geq 1 \ \forall \ s \in S$$

$$x_i \in \{0, 1\} \ \forall 1 \leq i \leq n$$

Nun betrachten wir das relaxierte duale LP:
Zielfunktion:

$$\text{maximiere } \sum_{i=1}^{n} y_i$$

unter den Nebenbedingungen:

$$\sum_{j \in S_i} y_i \leq 1 \ \forall \ s_i \in F$$

$$y_j \in [0, 1]$$

Die primal duale Methode benutzt den 'Satz vom komplementären Schlupf' (siehe Satz 1.4.7)

Bei unserem Problem muss also für die optimale Lösung $\hat{x}, \hat{y}$ gelten:

$$\hat{x_i}(\sum_{j \in S_i} y_j - 1) = 0 \ \forall \ i \in \{1, \cdots, m\}$$

ILP-Formulierung für MSC:

min $\sum x_i$

Nb: $\sum x_i \geq 1$

$x_i \in \{0,1\}$

Matrix-Notation für die Nb:

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

Die Zeilen (der linken Matrix) sind als die s_i und die Spalten als die S_i zu verstehen

Relaxieren:
$x_i \geq 0$ statt $x_i \in \{0,1\}$
und das duale Programm:

max $\sum_{j=1}^{n} y_j$

Nb: $\sum_{j:s_j \in S_i} y_j \leq 1$ (i=1...m)
$1 \geq y_j \geq 0$

Bei unserem Problem muss für die optimalen Lösungen vom relaxierten Problem gelten:

$\hat{x}$ *($\sum_{j:s_j \in S_i} \hat{y}_j$ - 1)=0 (für i=1,..,m)
Dies folgt aus dem Satz vom komplementären Schlupf.

Idee:
Bestimme eine Lösung y=$y_1 ... y_n$ des aktuellen Programms.
Wenn die i-te Nebenbedingung scharf ist, dann setze x_i=1 und sonst x_i=0.
Falls dieses x=($x_1 ... x_m$) eine zulässige Lösung des primalen LP ist
(falls also die Bedingung vom komplementären Schlupf in eine Richtung erfüllt ist),
dann scheint es sich um eine gute Approximation zu handeln.

Algorithmus: DualSC1

(1) bestimme eine optimale Lösung y=(y_1...y_n) des dualen Programm

(2) falls i-te Nb des dualen Programms durch diese Lösung scharf ist,
dann setze $x_i = 1$
sonst $x_i = 0$
für i=1...m

(3) return x =(x_1,...,x_m)

Sei $\Delta_s = \max_{s \in S} |\{S_i \in F | s \in S_i\}|$

Satz 4:
Der Algorithmus DualSC1 approximiert MSC mit einer Güte von Δ_s

Beweis. DualSC1 liefert eine zulässige Lösung für das primale (ganzzahlige) LP
Begründung:
Angenommen dies wäre nicht der Fall,
d.h. $\exists s_j \in$S das nicht überdeckt wird
d.h. x_i=0 für alle $S_i \in$F mit $s_j \in S_i$
→ alle Nb mit y_j sind nicht scharf (wg. Zeile (2) des Alg)
→ die Zielfunktion des dualen Programms ist nicht maximiert
da y_j noch erhöht werden kann (→ Widerspruch!!!)

Seien OPT die Kosten einer optimalen Lösung von MSC

Durch den schwachen Dualitätssatz wissen wir,dass die Zielfunktion bei einer zulässigen Lösung des Dualen immer kleiner gleich der Zielfunktion bei einer zulässigen Lösung im Primalen ist. Also:

$$\sum_{j=1}^{n} y_j \leq \text{OPT}$$

jedes y_j ist in höchstens Δ_s Nebenbedingungen →

$$\sum_{i=1}^{m} = \sum_{i:x_i=1} 1 = \sum_{i:x_i=1} \sum_{j:s_j \in S_i} y_j \leq \Delta_s * \sum_{j=1}^{n} y_j \leq \Delta_s * \text{OPT} \qquad \square$$

verbesserter Algorithmus (bezüglich der Laufzeit):
Algorithmus: DualSC2(S,F)

Für alle i=1..n und alle j=1..m gilt:
x_i=0, y_j=0, dualschlupf[i]=1

WHILE es existiert noch ein nicht überdecktes $s_j \in$S DO
bestimme $S_i \in$F mit $s_j \in S_i$ und dualschlupf[i] ist minimal
x_i=1
y_j=dualschlupf[i]
FOR ALL k mit $s_j \in S_k$ DO dualschlupf[k] -= y_j
RETURN x=(x_1...x_n)

Satz 2.1.1. *Der Algorithmus DualSC2 hat eine Approximationsgüte von Δ_s und eine Laufzeit von O(nm)*

Beweis: → Übung

Zu DualSC2 ist noch anzumerken, dass die Anfangslösung y=(0,...,0) zulässig ist, und die While-Schleife die Zulässigkeit erhält.

Greedy-Algorithmus für MSC:

Seien U ⊂ S die Menge der noch nicht überdeckten Elemente
und C ⊂ F die Menge der bisher ausgewählten Elemente

Algorithmus: SC (S,F)

U:= S
C:= ∅
WHILE U ≠ ∅ DO bestimme $S_i \in$ F mit $|S_i \cap U|$ maximal
  C:= C ∪ $\{S_i\}$
  U:= U \ S_i
RETURN C

Analyse des Algorithmus‘ mit Hilfe von LP:

Definition:
Seien $S_{t1}...S_{tk}$ die vom Algorithmus ausgewählten Mengen in zeitlicher Reihenfolge
Dann sei der Preis eines s∈S:
  Preis(s) = $\frac{1}{|S_{ti} \setminus (S_{t1} \cup ... \cup S_{ti-1})|}$

Dabei beschreibt der Nenner die Anzahl der Elemente die erst im i-ten Schritt überdeckt werden.

Lemma 2.1.2. *Sei $H_m = \sum_{i=1}^{m} \frac{1}{i}$ die m-te harmonische Zahl.*
dann ist
  y=($y_1...y_n$) mit $y_i = \frac{Preis(s_i)}{H_n}$

Beweis. z.Z.: $\sum_{j:S_j \in S_i} y_j$ (i=1...m)
Sei i≤m, k = $|S_i|$ und S_i=$\{s_1...s_k\}$ und symbolisiere der Index von s o.B.d.A. die Reihenfolge in der sie überdeckt wurden
dann betrachte die Iteration, in der $s_t \in S_i$ zum ersten Mal überdeckt wird

→ S_i hat zu diesem Zeitpunkt noch mindestens k-t+1 unüberdeckte Elemente
also gilt:

$$\text{Preis}(s_t) \leq \frac{1}{k-t+1} * \sum_{j \in S_i} y_j = \sum_{j=1}^{k} y_j = \sum_{j=1}^{k} \frac{Preis(s_j)}{H_n}$$

$$\rightarrow \leq \sum_{j=1}^{k} \frac{1}{k-j+1} * \frac{1}{H_n} = \frac{H_k}{H_n}$$

□

Minimal Set Cover

Primales LP
$\min \sum_{i=1}^{m} x_i$
$\sum_{i:s\in S_i} x_i \geq 1 \quad \forall s \in S$
$x_i \in \{0,1\}$

Duales LP
$\max \sum_{i=1}^{n} y_i$
$\sum_{j:s\in S_i} y_j \leq 1$
$y_i \in [0,1]$

Lemma 2.1.3. *$y = (y_1, \ldots, y_n)$ mit $y_i = \frac{Preis(s_i)}{H_n}$ ist eine zulässige Lösung für das duale lineare Programm.*

Beweis. siehe letzte Vorlesung □

Damit gilt für die Überdeckung, die der Algorithmus liefert $S_1, \ldots, S_{t_r}$:
$\Rightarrow x = (x_1, \ldots, x_n)$ mit $x_{t_1} = \cdots = x_{t_r} = 1$ und $x_i = 0$ sonst.

Frage 2.1.4. *Was ist die zugehörige Lösung des ILP?*

Wir betrachten den Vektor $\overline{y} = (Preis(s_1) \ldots Preis(s_n))$
Es gilt $\sum_{i=1}^{m} x_i = \sum_{j=1}^{n} \overline{y_j}$ da $\sum_{s\in S_{t_i}\setminus(S_{t_1}\cup\cdots\cup S_{t_{i-1}})} Preis(s) = 1$
$\sum_{i=1}^{m} x_i = \sum_{i=1}^{n} \overline{y_i} = H_n \cdot \sum_{j=1}^{n} y_j \leq H_n \cdot \widetilde{OPT} \leq H_n \cdot OPT$

Wenn die $\overline{y_i}$ eine zulässige Lösung für das Duale wären, dann wäre die Lösung optimal. $\widetilde{OPT}$ ist die optimale Lösung des ursprünglichen relaxierten LP. Wir verwenden den schwachen Dualitätssatz. OPT ist die optimale Lösung des ursprünglichen ganzzahligen LP.

Satz 2.1.5. *Der Algorithmus SC liefert eine approximative Lösung für das Problem MSC mit Approximationsgüte $H_n = \Theta(\log(n))$. Das ist besser als Δ_s.*

Idee dahinter:
Wir betrachten das Problem als (0,1)-LP und bringen es auf die Form:
$\min c^T x$
$Ax \geq b$
$x \in \{0,1\}^n$

Wir suchen eine zulässige Lösung x und eine Lösung $\overline{y}$ des dualen relaxierten Programms mit $c^T x \leq \overline{y}^T \cdot b$. Wenn $\overline{y}$ zulässig wäre hätten wir das Optimum gefunden. Das ist es aber im Allgemeinen nicht. Desfalb suchen wir ein α, so dass $y = \frac{\overline{y}}{\alpha}$ eine zulässige Lösung ist.

Dann ist $c^T x \leq \overline{y}^T \cdot b \leq \alpha y^T b$. Nach dem schwachen Dualitätssatz ist das kleiner als

$\alpha \cdot OPT$ für das duale Programm also auch kleine r$\alpha \cdot OPT$ für das (0,1)-LP. Daraus folgt, dass der Algorithmus eine Approximationsgüte von α hat.

innere Punkt Methoden (Karmarkar '84)

Die innere Punkt Methode ist iterativ und hat polynomielle Laufzeit. Angenommen wir haben einenen inneren Punkt aus der Menge der zulässigen Lösungen gefunden. Dann konstruieren wir eine Kugel um diesen inneren Punkt, die noch ganz in der Menge der zulässigen Lösungen liegt. Dafür nimmt man den minimalen Abstand der Nebenbedingungen zum Punkt und wählt die Kugel mit diesem Radius. Dann sucht man eine bessere Kugel. Vom inneren Punkt geht man in die Gegenrichtung der Zielfuktion bis zum Rand der Kugel und definiert dies als neuen Punkt. Um diesen wird wieder eine Kugel gelegt. Das Problem ist, dass der Fortschritt immer kleiner wird. Deshalb führt man eine projektive Transformation des Ganzen durch. Dadurch können Bereiche ausgedehnt oder geschrumpft werden. Der interessante Bereich wird ausgedehnt und der Punkt wandert in die Mitte. Schnelle LP-Löser benutzen eine Kombination aus der inneren Punktmethode und dem Simplex-Algorithmus.

Kapitel 3

Algorithmische Geometrie

Dieses Gebiet gibt es seit Mitte der 80er. Es beschäftigt sich mit Algorithmen für geometrische Probleme. Die Anwendungen sind vielfältig, weil die Geometrie durch Computergraphik sehr wichtig geworden ist. Eine dreidimensionale Szene die man im Rechener modelliert hat, auf den Bildschirm zu projezieren ist nicht so einfach wie es scheint. Eine andere Anwendung ist die Mustererkennung. Es gibt in der numerischen Mathematik Methoden (finite Elemente) die darauf beruhen, komplexe Flächen in einfachere zu zerlegen und das Problem darauf zu lösen. (z.B. Triangulierung)

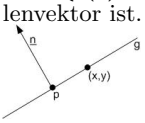
3.1 Elementare Objekte im zweidimensionalen Raum

In vielen Fällen ist die Übertragung auf höhere Dimensionen klar. Das elementarste Objekt ist ein Punkt.

Definition 3.1.1. Nehmen wir an, ein Punkt(point) ist dargestellt durch seine kartesischen Koordinaten (x,y).

Definition 3.1.2. Eine Gerade (straight line) kann auf verschiedene Arten dargestellt werden.
z.B. durch $(a, b) \in \mathbb{R}^2$ im zweidimensionalen Raum, wobei gemeint ist $y = ax + b$. Aber eine parallele zur y-Achse lässt sich nicht darstellen. Deshalb gibt es auch andere Darstellungen.
$\{\lambda p + (1 - \lambda)q | \lambda \in \mathbb{R}\}$ ist die Gerade durch die Punkte $p, q \in \mathbb{R}^2$.

oder $\left\{ \begin{pmatrix} x \\ y \end{pmatrix} \mid \left(p - \begin{pmatrix} x \\ y \end{pmatrix} \right) \cdot \underline{n} = 0 \right\}$ in der Hessesche Normalform. Wobei $\underline{n}$ ein Normalenvektor ist.

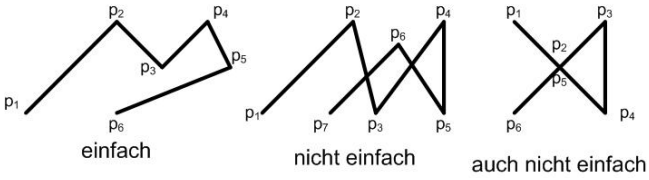


Dabei haben die Punkte links (in Richtung des Normalenvektors) von der Geraden g die Eigenschaft $\left(p - \begin{pmatrix} x \\ y \end{pmatrix} \right) \cdot \underline{n} < 0$ die Punkte rechts $\left(p - \begin{pmatrix} x \\ y \end{pmatrix} \right) \cdot \underline{n} > 0$. Diese Mengen heißen offenen Halbebenen.

Definition 3.1.3. Eine Strecke (straight line segment) zwischen den Punkten p und q ist definiert durch $\overline{pq} = \{x | x = \lambda p + (1 - \lambda)q, \lambda \in [0, 1]\}$

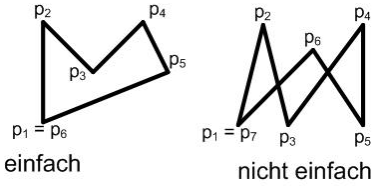
Definition 3.1.4. Aus Strecken lassen sich Streckenzüge bzw. Polygonzüge (polygon chain) zusammenbauen. Ein Streckenzug ist eine Folge von Strecken der Form $\overline{p_1p_2}, \overline{p_2p_3} \dots \overline{p_{n-1}p_n}$. p_i sind die Ecken und $\overline{p_ip_{i+1}}$ die Kanten des Polygonzugs.

Definition 3.1.5. Ein Polygonzug heißt einfach, wenn für $i < j$ gilt
$\overline{p_ip_{i+1}} \cap \overline{p_jp_{j+1}} = \begin{cases} p_j \text{ falls } j = i + 1 \\ \emptyset \text{ sonst} \end{cases}$.

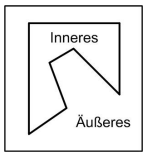


Definition 3.1.6. Ein Polygonzug heißt Polygon, falls $p_n = p_1$

Definition 3.1.7. Ein Polygon heißt einfach, wenn für $i < j$ gilt
$\overline{p_ip_{i+1}} \cap \overline{p_jp_{j+1}} = \begin{cases} p_j \text{ falls } j = i + 1 \\ p_1 \text{ falls } i = 1 \wedge j = n - 1 \\ \emptyset \text{ sonst.} \end{cases}$.



Definition 3.1.8. Ein einfaches Polygon teilt den $\mathbb{R}^2$ in ein Inneres und ein Äußeres.



Definition 3.1.9. $A \subset \mathbb{R}^d$ heißt konvex $\Leftrightarrow \forall p, q \in A : \overline{pq} \subset A$ Der Durchschnitt beliebig vieler konvexer Mengen ist konvex.

Definition 3.1.10. Sei $A \subset \mathbb{R}^d$, dann ist die konvexe Hülle (convex hull) $CH(A) = \bigcap_{A \subset B, B konvex} B$ enthalten in jeder konvexen Menge, die A enthält.

Definition 3.1.11. Ein einfaches Polygon heißt konvexes Polygon, wenn sein Inneres konvex ist.

Definition 3.1.12. Die konvexe Hülle einer endlichen Punktmenge ist ein konvexes Polygon.

Frage 3.1.13. *Wie berechnet man die Punkte, die auf der konvexen Hülle liegen?*

Viele Probleme sind für konvexe Polygone einfacher zu lösen als im Allgemeinen. Deswegen werden viele Probleme auf konvexe Polygone zurückgeführt indem man kompliziertere Formen in konvexe Polygone zerlegt. Zum Beispiel kann man die Fläche eines einfachen Polygons berechnen, indem man es trianguliert und die Flächen der Dreiecke addiert. Wobei das Triangulieren nur bei konvexen Polygonen trivial ist. (Man muss nur von einer Ecke Strecken zu allen anderen Ecken ziehen.)

Definition 3.1.14. Ein Punkt p einer konvexen Menge $K \in \mathbb{R}^d$ heißt Extrempunkt g.d.w. es keine $a, b \in K$ gibt, $a \neq b$ mit $p \in \overline{ab}$. Beim konvexen Polygon sind das genau die Ecken.

Satz 3.1.15. *Der Rand der konvexe Hülle einer endlichen Punktmenge S ist ein konvexes Polygon dessen Ecken Punkte von S sind.*

3.2 Berechnung der konvexen Hülle

3.2.1 Graham-Scan

gegeben: endliche Punktemenge S
gesucht: $CH(S)$ als Polygonzug

Algorithmus:

(1) Bestimme den Schwerpunkt s von S.

(2) Bestimme alle Strecken von s zu allen Punkten in S.

(3) Sortiere die Strecken nach Steigung. Falls mehrere Strecken die gleich Steigung haben, entferne alle Punkte und Strecken, bis auf den Punkt und seine zugehörige Strecke, der am weitesten vom Schwerpunkt s entfernt ist. Seien dann $s_1, ..., s_n$ die Strecken und $p_1, ..., p_n$ die zugehörigen Punkte.

(4) Sei $i = 3, ..., n, 1$. Betrachte nun die Strecke $\overline{p_{i-1}p_i}$. Falls sie mit der Strecke $\overline{p_{i-2}p_{i-1}}$ einen Innenwinkel von mehr als 180° bildet, verbinde p_i mit p_{i-2}. Falls nun $\overline{p_ip_{i-2}}$ mit $\overline{p_{i-2}p_{i-3}}$ einen Winkel von mehr als 180° bildet, inspiziere p_{i-3} usw bis p_{i-k}, so dass $\sphericalangle p_{i-k-1}p_{i-k}p_i < 180°$ oder bis p_1 erreicht wird, falls dies vorher eintritt.

(5) Entferne falls notwendig $p_1, p_2, ..., p_k$ bis $\sphericalangle p_np_{k+1}p_{k+2} < 180°$.

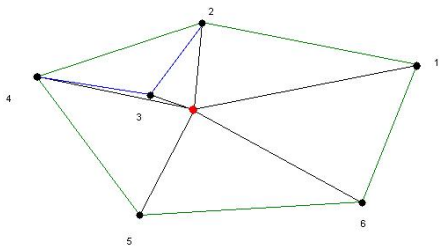


Abbildung 3.1: Graham-Scan

Laufzeit:

(1) $O(n)$

(2) $O(n)$

(3) $O(n \log n)$

(4) $O(n)$, denn jeder Punkt wird einmal aufgenommen und höchstens einmal entfernt aus CH.

(5) $O(n)$

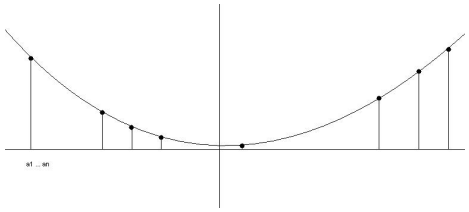
insgesamt $O(n \log n)$

Satz 3.2.1. *Der Graham-Scan berechnet die konvexe Hülle einer Menge von n Punkten in* $O(n \log n)$ *Zeit.*

untere Schranken:

Sortieren von n reelen Zahlen kann in linearer Zeit auf Berechung der konvexen Hülle reduziert werden.

gegeben: $a_1, ..., a_n \in \mathbb{R}$



Betrachte Punkte (a_i, a_i^2) mit $i = 1, ..., n$. Wir berechnen ihre konvexe Hülle beginnend mit dem linkestens Punkt. Durchlaufen wir nun die konvexe Hülle, so erhalten wir die Zahlen aufsteigend sortiert, d.h. kann man in Zeit $T(n)$ die konvexe Hülle berechnen, so kann man in $O(T(n) + n)$ sortieren.
Sortieren von n Zahlen benötigt $\Omega(n \log n)$ Zeit, wenn als Operationen $+, -, *, /, \leq$ von reelen Zahlen erlaubt sind. (siehe Ben-Or)

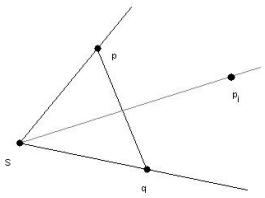
Satz 3.2.2. *Im algebraischen Berechnungsmodell benötigt die Berechnung der konvexen Hülle von n Punkten in $\mathbb{R}^2$ $\Omega(n \log n)$ Zeit.*

Ein inkrementeller Algorithmus: Seien $p_1, ..., p_n$ gegebene Punkte. Die konvexe Hülle kann online bestimmt werden, d.h. es sind nicht alle Punkte zu Beginn bekannt, die konvexe Hülle wird inkrementell Punkt für Punkt erweitert.

Algorithmus:

(1) Bestimme den Schwerpunkt s von p_1, p_2, p_3 und die Strahlen $s_i = \overrightarrow{sp_i}$ für $i = 1, 2, 3$. Sortiere diese nach ihrer Steigung.

(2) Sei $i = 4, ..., n$. Bestimme die Lage des Strahls $\overrightarrow{sp_i}$ innerhalb der vorherigen Strahlen durch Binärsuche. Falls p_i diesseits, also auf der gleichen Seite wie s, der Kante p, q liegt, ignoriere p_i. Ansonsten füge Strahl $\overrightarrow{sp_i}$ zwischen $\overrightarrow{sq}$ und $\overrightarrow{sp}$ ein und lösche die Kante $\overline{pq}$ aus der konvexen Hülle. Gibt es ausgehend von $\overrightarrow{p_ip}$ und $\overrightarrow{p_iq}$ Tangenten von p_i an der bisherigen konvexen Hülle, so füge diese ein und entferne die Stücke dazwischen.

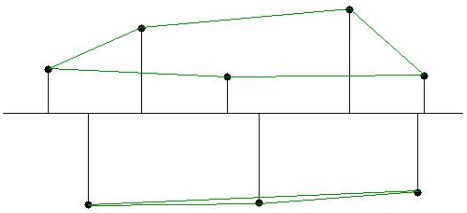


Laufzeit:

(1) $O(n)$

(2) $O(n)$ für die Binärsuche in einem balanciertem binären Suchbaum, ebenso für Einfügen und Streichen. Das Finden der Tangenten kostet $O(\log n)$, da jeder Punkt einmal eingefügt und höchstens einmal gestrichen wird. Insgesamt für (2) haben wir nach n Schritten $O(n \log n)$.

insgesamt $O(n \log n)$



weitere Möglichkeit:

Wir sortieren die Punkte nach ihren x-Koordinaten und berechnen die konvexe Hülle für oben und unten getrennt.

3.3 planare Unterteilungen und Voronoi-Diagramme

Definition 3.3.1. Ein Graph $G = (V, E)$ heißt planar, wenn er überschneidungsfrei in die Ebene eingebettet werden kann, d.h. es gibt folgende Abbildungen:

$$f : V \to \mathbb{R}^2$$
$$g : E \to \text{ Kurven im } \mathbb{R}^2$$

so dass gilt: Für jede Kante $e = u, v$ gilt: $f(n), f(v)$ sind Endpunkte der Kurve $g(e)$ und $e \neq e'$ und $g(e), g(e')$ schneiden sich nicht in ihrem Inneren.

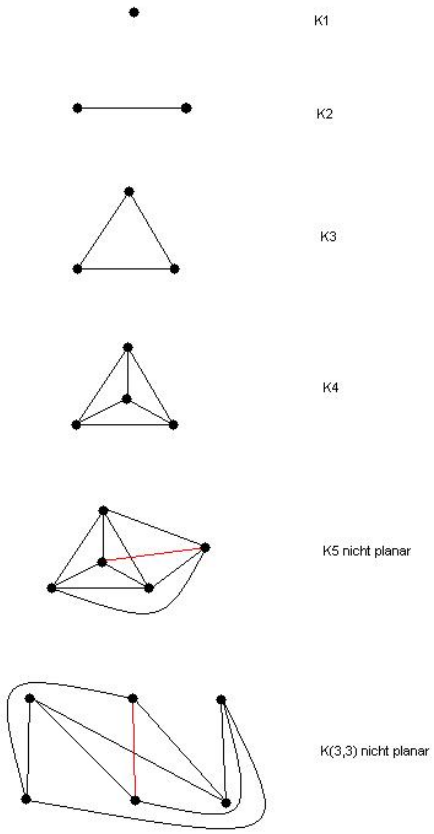


Abbildung 3.2: Beispiele für planare Graphen

Satz 3.3.2 (von Kuratowski(1930))**.** *Jeder nicht planare Graph enthält als Teilgraph eine Unterteilung des K_5 oder des $K_{3,3}$. Unterteilung heißt, dass Kanten durch knotendisjunkte Wege ersetzt werden dürfen.*

Satz 3.3.3 (Fary 1948)**.** *Zu jedem planaren Graphen existiert eine geradlinige Einbettung, d.h. alle Kanten werden durch Strecken realisiert.*

Beispiel.

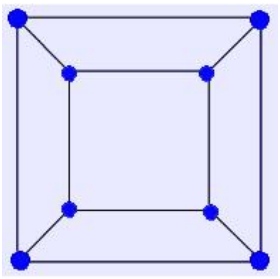


Abbildung 3.3: geradlinige Einbettung von Würfel

Es gilt:

Satz 3.3.4. *Jedes konvexe Polyeder im dreidimensionalen Raum ist ein planarer Graph.*

Beispiel. Dodekaeder und platonischer Körper

Beweisidee: Sei π konvexes Polyeder, wähle Punkt m im Inneren von π und Kugel S mit Mittelpunkt m, die π ganz umfasst. Oberfläche, projizieren Ecken von π von m aus auf S. Die Kanten werden auch über kreuzungsfreie auf die Kugelfläche einbettet. Mit stereographischer Projektion gibt es eine homöomorphe Abbildung S: Kugeloberfläche ohne einen Punkt in die Ebene $\mathbb{R}^2$. Die Abbildung erhält auch die topologischen Eigenschaften, insbesondere überkruzungsfreiheit. Wenden S auf Einbettung von π auf Kugeloberfläche S an, $\rightarrow$ Einbettung in die Ebene. Keine Ecken auf "Nordpol" (proj. Zentrum) abbilden, falls planarer Graph 3-fach zusammenhängend ist, dann ist er als Polyeder realisierbar.

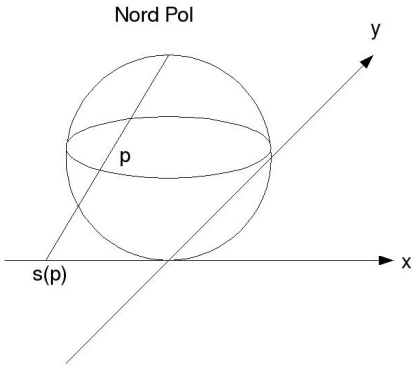


Abbildung 3.4: stereograhpische Projektion

Definition 3.3.5. Sei η Einbettung eines Graphen in die Ebene. Facette von η ist Zusammenhangskomponente von $\mathbb{R}^2 \backslash \eta$.

Beispiel:

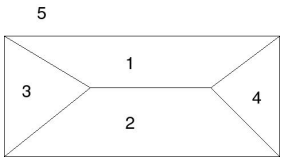


Abbildung 3.5: 5 Facetten

Definition 3.3.6. Dualer Graph eines eingebettet planaren Graphen $G = (V, E)$ ist $G^* = (V^*, E^*)$ mit $V^* =$ Facetten von G, $E^* \widehat{=} E$. $e \in E$ trennt zwei Facetten F_1, F_2 $\Rightarrow$ zugehörige duale Kante verbindet F_1, F_2.

Beispiel:

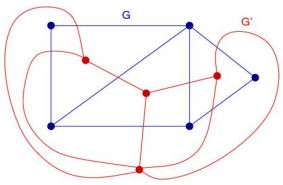


Abbildung 3.6: Beispiele für dualen Graphen

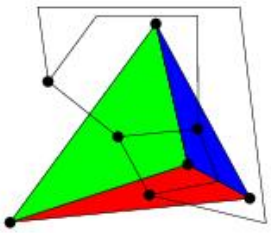


Abbildung 3.7: k_4 ist selbst dual

Eigenschaft: Sei G planar und zusammenhängend, dann:

a. G^* hat auch diese Eigenschaft und $G^{**} = G$

b. $n :=$ Anzahl der Ecken, $f =$ Anzahl von Facetten, $e =$ Anzahl der Kanten, dann $n + f = e + 2$("Eulerscher Polyedersatz")

c. $e \leq 3n - 6$ (G planar und einfach)

d. $f \leq 2n - 4$ (G planar und einfach)

e. $n \leq 2f - 4$ (G und G^* planar und einfach)

Definition 3.3.7. Betrachten endl. Punktmenge S und euklidische Metrik als Abstandsmaß also $\|p - q\| = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}$ $\forall p, q \in \mathbb{R}, p = (p_x, p_y), q = (q_x, q_y)$

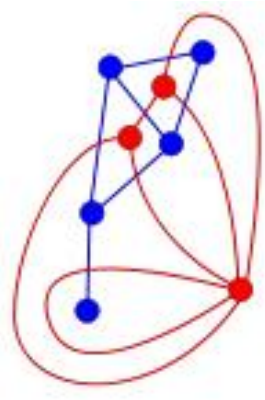


Abbildung 3.8: dualer Graph kann Schleife und Mehrfachkanten enthalten

Teile $\mathbb{R}^2$ auf in Gebiete, die jeweils zu einem Punkt von S näher sind als zu allen anderen.

Beispiel:

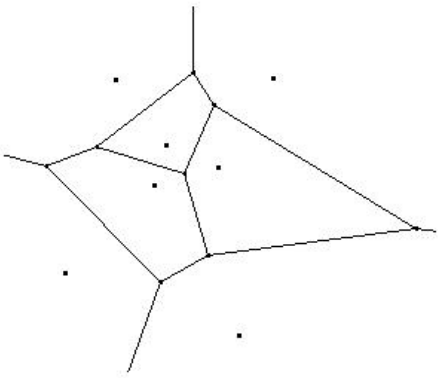


Abbildung 3.9: Beispiele für Voronoi-Diagramm

diese Aufteilung heißt Voronoi-Diagramm von S

formal: $S = \{p_1, p_2, \ldots, p_n\}$

$$VR(p_i) := \{p \in \mathbb{R}^2 | \, \|p - p_i\| < \|p - p_j\| \text{ für alle} j \neq i\}$$

also Voronoi-Region(-Zelle) von p_i.

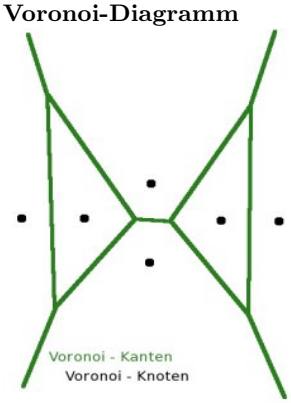
$VR(p_i)$ ist ein konvexes Polygon, falls beschränkt. Das ist auf jeden Fall konvex und durch endlich viele Geraden begrenzt, denn Durchschnitt von endlich vielen Halbebenen(-räumen), nämlich $\{p | \, \|p - p_i\| < \|p - p_j\| \, , j = 1, \ldots, n, j \neq i\}$. Voronoi-Kanten sind Kanten, die die Voronoi-Regionen begrenzen, Voronoi-Ecken sind Ecken der VR.

Anwendungen:

- post office problem

- kristallogrophie
- Robotik Bewegungsplanung
- Biologie
- Soziologie

Voronoi-Diagramm



Eigenschaften

a) Jeder Punkt auf einer Voronoi-Kante hat den kürzesten Abstand zu mindestens zwei Punkten in S

b) Jeder Voronoi-Knoten hat den kürzesten Abstand zu mindestens drei Punkten in S

c) genau die Punkte in S, die auf dem Rand von $CH(S)$ liegen, haben eine unbeschränkte Voronoi-Region

d) die Anzahl der Voronoi-Kanten ist $e \leq 3n - 6$
die Anzahl der Voronoi-Knoten ist $v \leq 2n - 5$ (bei $n \geq 3$)

Informationen zu d)

- mit hinreichend weit entferntem Zusatzknoten, mit dem alle unbeschränkten Kanten verbunden sind erhalten wir einen **eingebetteten planaren Grapen**
- die Komplexität von $VD(S)$ ist linear in $n = |S|$

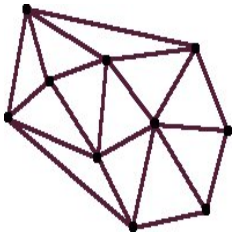
Bei der Betrachtung von $VD(S)$ als planaren eingebetteten Graphen betrachten wir den **dualen Graphen**

d.h. jeder Punkt $p \in S$ wird verbunden mit $q \in S$, falls deren Voronoi-Regionen durch eine gemeinsame Kante getrennt sind.

Definition: Triangulierung einer Punktmenge S

Hierbei handelt es sich um einen eingebetteten Graphen mit geradlinigen Kanten ('geometrischer Graph'),

- dessen Knoten die Punkte von S sind,
- dessen Facetten (außer der Äußeren unbeschränkten) Dreiecke sind und
- der alle Kanten der konvexen Hülle enthält



Eigenschaften des dualen Graphen vom VD

a) bei 'allgemeiner Lage' von S, d.h. keine vier Punkte liegen auf einem Kreis, ist der duale Graph eine **Triangulierung** von S, und zwar eine **Delaunay-Triangulierung**

b) zwischen zwei Punkten $p, q \in S$ existiert eine Kante $\Leftrightarrow$ es gibt einen 'leeren' Kreis durch p und q, d.h. er enthält (weder direkt noch im Inneren) keine weiteren Punkte aus S

c) drei Punkte p, q, r bilden eine Facette $\Leftrightarrow$ der Umkreis von p, q, r ist leer

d) aus dem dualen Graphen lässt sich in $O(n)$ Zeit das VD konstruieren und umgekehrt

Konstruktion des Voronoi-Diagramms

Divide and Conquer - Algorithmus

falls $|S| > 1$:

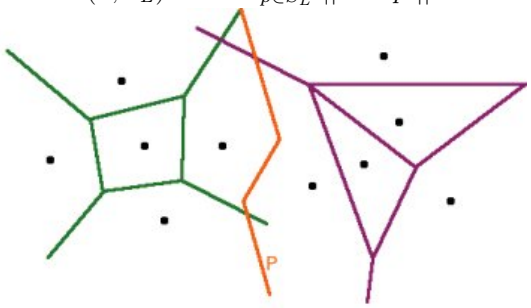
a. sortiere S lexikographisch bezüglich x-, dann y-Koordinate
teile S in eine linke Hälfte S_L und eine rechte Hälfte S_R

b. konstruiere rekursiv $VD(S_L)$, $VD(S_R)$

c. mische beide $V(D)$ zu einem: $VD(S)$

Details zu Schritt 3:
Konstruktion der 'Trennlinie' P zwischen S_L und S_R
$P = \{x \in \mathbb{R}^2 \mid d(x, S_L) = d(x, S_R)\}$

wobei $d(x, S_L) := min_{p \in S_L} \mid\mid x - p \mid\mid$



Eigenschaften (ohne Beweis):

a) $P = \{x \in \mathbb{R}^2 \mid x \in$ Kante von $VD(S)$, die ein $p \in S_L, q \in S_R$ trennt $\}$

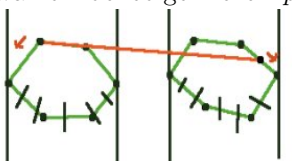
b) P Kantenzug bestehend aus zwei Strahlen an seinen Enden und einigen Strecken. Die Strahlen sind Teile der Bisektoren zwischen Punkten, an denen obere und untere Tangenten zwischen Punkten von $CH(S_L)$ und $CH(S_R)$ anliegen

c) Sind L, R die Gebiete links und rechts von P, dann ist $VD(S) = (VD(S_L) \cap L) \cup P \cup (VD(S_R) \cap R)$, wobei $VD(S)$ als Menge der Punkte auf den Kanten zu verstehen ist

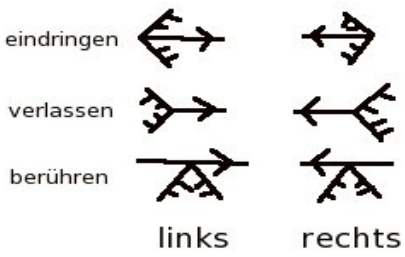
Konstruktion der oberen Tangente (Eigenschaft b)

angenommen die konvexen Hüllen $CH(S_L), CH(S_R)$ werden mitkonstruiert

a. wählen beliebige Ecken p von oberen $CH(S_L)$, q von oberen $CH(S_R)$



b. bestimmen rechtesten und linkesten Ecken von $CH(S_L), CH(S_R)$ und entfernen unteren Teile der CH



Fälle für Verbindungsstrecke der ausgewählten Punkte

Daraus ergeben sich neun Kombinationen.

Beispiel 1: Bei beiden Polygonen tritt der Fall 'berühren' auf - dann wurde die Tangente gefunden.
Beispiel 2: Beim linken Polygon tritt der Fall 'berühren' auf und bei dem Rechten der Fall 'verlassen' - dann kommen die Punkte unter dem 'verlassenen' Knoten des rechten Polygons nicht mehr in Frage.
Beispiel 3: Beim linken Polygon tritt der Fall 'eindringen' auf und bei dem Rechten der Fall 'verlassen' - dann kommen bei beiden Polygonen die Knoten unter dem 'eindringenden'- und unter dem 'verlassenen'-Knoten nicht mehr in Frage.

In jedem Fall: der Bereich der oberen konvexen Hülle unterhalb mindestens eines Punktes kommt nicht mehr in Frage

mit einer Art Binärsuche findet man die Tangente in $O(logn)$ Zeit (wobei $O(n)$ reichen würde)

Mit den vorherigen Überlegungen ergibt sich Schritt 3 des Algorithmus wie folgt:

a. Finde die obere Tangente t an $CH(S_i)$ und $CH(S_R)$ und deren Tangentialpunkte $p \in S_L$, $q \in S_R$. Weiterhin sei s der Strahl die Mittelsenkrechte zu $p - q$.

b. Verlängere den Strahl s nun von hinreichend weit oben kommend nach unten bis er das erste Mal $VD(S_l) \cup VD(S_R)$ schneidet. Dieser Schnittpunkt sei r. Falls es keinen Schnittpunkt gibt, sind wir fertig. Liegt r in $VD(S_L)$ (bzw. $VD(S_R)$), so löse p (bzw. q) durch denjenigen Punkt $p' \in VD(S_L)$ (bzw. $q' \in VD(S_R)$) ab, in deren Voronoizelle s eintritt. s sei nun die Mittelsenkrechte zwischen p' und q (bzw. p und q').
Iteriere (2) bis die untere Tangente erreicht ist, also es keinen weiteren Schittpunkt mehr gibt.

c. Entferne die Voronoikanten, die nun nicht mehr zum VD gehören.

Bemerkung. Das Finden von r in Schritt (2) geschieht durch Ablaufen des Randes von $VP(p)$ und $VP(q)$ bis der Schnittpunkt mit dem Strahl gefunden ist. Insgesamt werden dazu $O(n)$ Kanten getestet, wobei man hierzu z.B. doppelt verkettete Listen der Kanten des Randes jeder Zelle pflegt.

Für das Mischen in Schritt 3 ergibt sich also eine Laufzeit von $O(n)$. Damit erhalten wir folgende Rekursionsgleichung

$$T(n) = 2T(n/2) + O(n),$$

was einer Laufzeit von $T(n) = n \log n$ entspricht.

Satz 3.3.8. *Das VD von n Punkten im* $\mathbb{R}^2$ *kann in* $O(n \log n)$ *berechnet werden.*

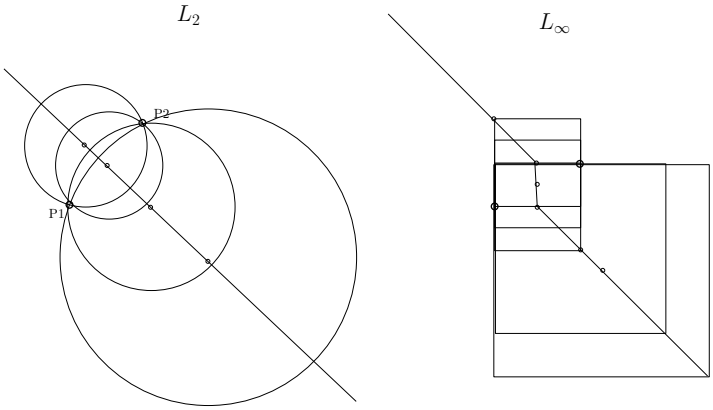
Bemerkung. Die Berechnugn de VD in 2 Dimensionen kann man auf die Berechnung der CH in 3 Dimensionen reduzieren.

Variationen:

- Man kann Voronoi-Diagramme bzgl. anderer Metriken berechnen, z.B. der L_1- oder der L_∞-Metrik, wobei

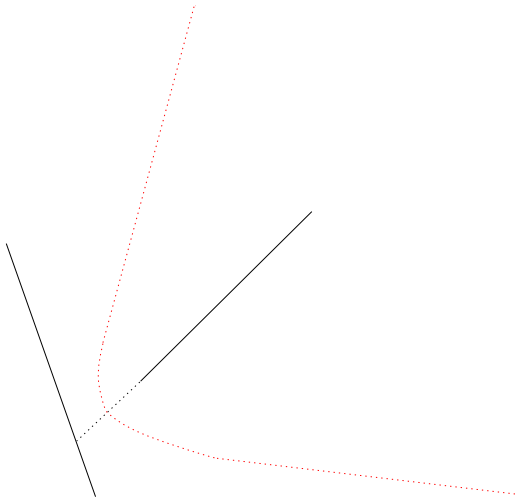
$$L_1 = \|p - q\|_1 = |p_x - q_x| + |p_y - q_y|$$
$$L_\infty = \|p - q\|_\infty = \max(|p_x - q_x|, |p_y - q_y|)$$

  Beispiel. VD für zwei Punkte in der L_2-Metrik und in der L_∞-Metrik. Bei der L_2-Metrik ergibt sich die VK aus den Mittelpunkten der Kreise, auf deren Rand sowohl $P1$ als auch $P2$ liegen. Bei der L_∞-Metrik sind dies Quadrate, wodurch diese Abzweigung entsteht.

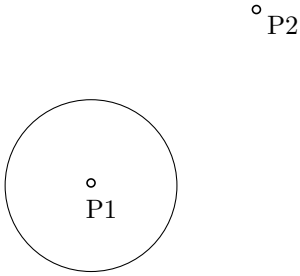


Man kann auch Voronoidiagramme von anderen Objekten, wie Strecken und Kreisen erstellen. Diese können dann als Hindernisse interpretiert werden und finden damit z.B. in der Robotik Anwendung (siehe Anwendungen).

Beispiel. VD von zwei Strecken. Die VK sind weiter weg von den Strecken die Winkelhalbierenden und unmittelbar zwischen den Strecken Parabeln.



Eine weitere Möglichkeit besteht darin die VD zu wichten. Diese nennt man dann „Johnson-Mehl-Diagramme“. Die VK sind dann Kreisbögen. Im Beispiel wird $P1$ etwas weniger Gewicht zugeordnet, was man direkt an dem Verhältnis der Strecke von $P1$ zum Kreisbogen zu $P2$ zum Kreisbogen erkennen kann.



Weitester-Punkt-VD. Diese VD sind geradlinig beschränkt. Ihre Zellen sind alle unbeschränkt, somit bilden die VK einen Baum.

Anwendungen:

- Bestimmen der beiden nächstgelegenen Punkte einer Punktemenge S. Brute-Force dauert dies $O(n^2)$ Zeit. Man beobachtet, dass die Strecke zwischen den beiden Punkten mit kürzestem Abstand eine Delaunay-Kante sein muss, da es einen leeren Kreis durch diese beiden gibt. Man konstruiert also die Delaunay-Triangulierung von S und bestimmt deren kürzeste Strecke, was einen $O(n \log n)$-Zeit-Algorithmus liefert.
- Bewegungsplanung in der Robotik. Man hat Hindernisse in der Ebene gegeben und einen Roboter (z.B. Kreis mit Radius r), der einen Weg durch die Hindernisse suchen soll. Man kann dieses Problem lösen indem man das zugehörige Voronoi-Diagramm berechnet und die Kanten herauslöscht, die zwei Zellen voneinander trennt, deren Hindernisse einen zu geringen Abstand voneinander haben. Dann führt man einen Graphen-Wegfinde-Algorithmus durch.

3.4 Suchen in ebenen Unterteilungen

Definition 3.4.1. Ebene Unterteilungen sind Unterteilungen der Ebene in Regionen durch Kurven (wir nehmen Strecken, Strahlen, Geraden an).

Das algorithmische Problem, das wir hier betrachten möchten ist:
Problem:

- Gegeben: Ein Anfragepunkt q.
- Finde: Die Region, die q enthält.

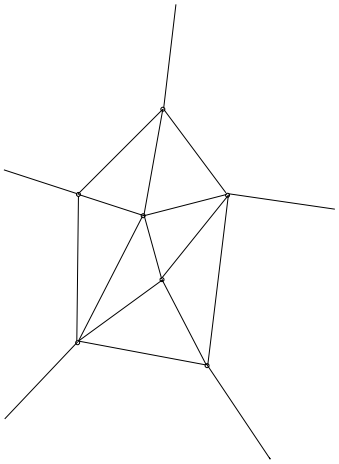
Beispiel. Ein Beispiel für eine ebene Unterteilung ist das Voronoidiagramm. Angewandt auf diese Unterteilung suchen wird denjenigen Punkt p, der q am nächsten ist. Dieses Problem heißt auch „post office problem“

Die der Lösung zugrundeliegende Idee soll die Folgende sein. Die ebene Unterteilung wird als statisch angesehen. Die Eingabe wird dann vorverarbeitet (in einer adäquaten Datenstruktur), damit mehrere unterschiedliche Anfragen auf der selben Unterteilung effizient beantwortet werden können. Die zu optimierenden Ressourcen sind also:

- Vorverarbeitungszeit
- Speicherbedarft
- Anfragezeit

Triangulierte ebene Unterteilungen:

Beispiel. Beispiel einer Triangulierten Unterteilung. Die Vorstellung für die ins Unendlliche gehenden Strahlen ist, dass sie alle in einem unendlich fernen Punkt zusammentreffen und dieser dann die äußeren Dreiecke schließt.



Lemma 3.4.2. *Es gibt Konstanten $c > 0$ und $k \in \mathbb{N}$, so dass gilt: Für jede triangulierte ebene Unterteilung $\widehat{G}$ mit n Knoten (einschließlich des unendlich fernen Knotens) gibt es eine unabhängige Knotenmenge I mit mindestens cn Elementen, in der jeder Knoten maximal Grad k hat. I kann in $O(n)$ Zeit gefunden werden.*

Proof. Die Anzahl der Kanten ist höchstens $3n - 6$, da die Triangulierung ein planarer Graph ist. Also ist die Summe der Kontengrade höchstens $6n-12 \leq 6n$. Es folgt, dass es höchstens $n/2$ Knoten gibt die mindestens den Grad 12 haben. Sei K die Menge der Übrigen also derer, dessen Grad kleiner als 12 ist. $|K|$ ist also mindestens $n/2$. Iteriere nun Folgendes bis K leer ist: Nimm einen Knoten aus K und füge ihn in I ein. Entferne diesen und alle zu ihm adjazenten Knoten aus K. Dann ist

$$I \geq \frac{n}{24}$$

und alle Knoten in I haben höchstens Grad 11. Wähle also $c = 1/24$ und $k = 11$ □

Algorithmus zur Konstruktion einer Suchstruktur

gegeben sei eine triangulierte ebene Unterteilung G_0, falls G_0 ein Dreieck ist, speichere G_0, sonst:

a. *bestimme die Menge I gemäß Lemma*

b. *entferne die Knoten aus I einschließlich anhängender Kanten → ebene Unterteilung G_1 mit $\leq (1-c) * n$ Knoten*

   Zusatzinfo:

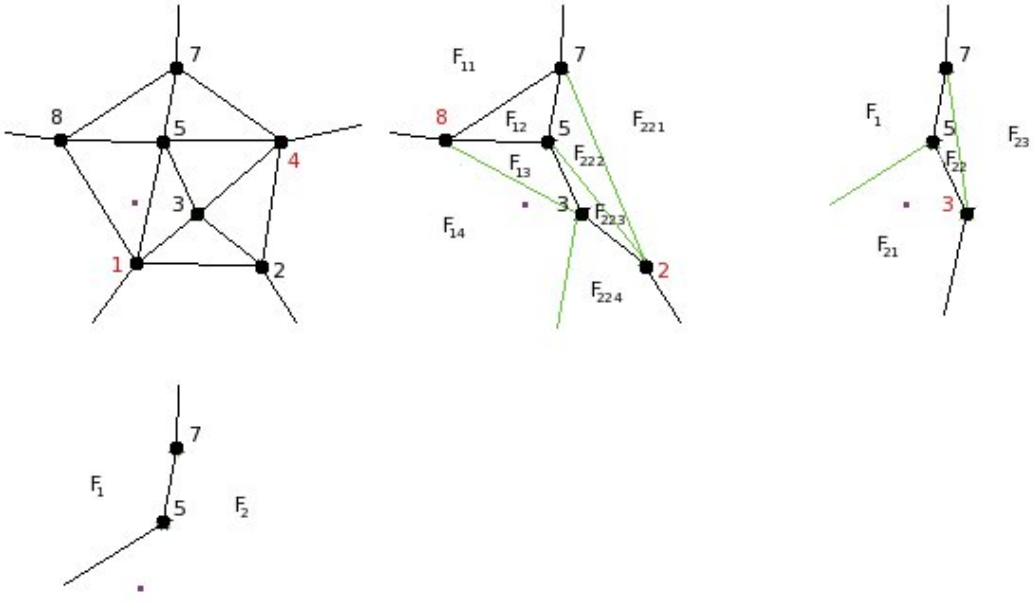
   - *die Facetten von G_1 : m-Ecke mit $m \leq 12$*
   - *jedes enthält höchstens einen Knoten aus I*

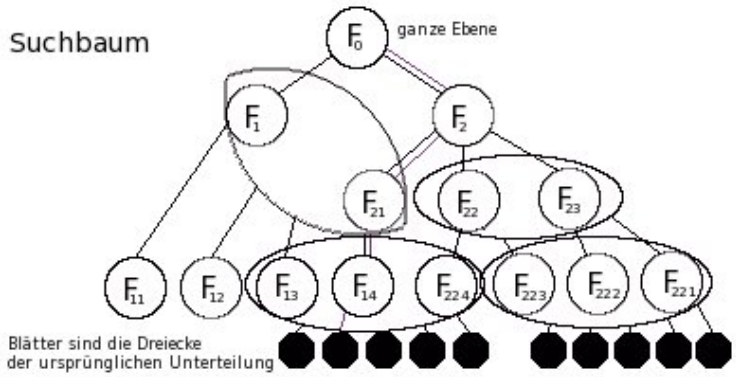
c. *trianguliere die Facetten → triangulierte ebene Unterteilung G_2*

d. *konstruiere rekursiv Datenstruktur D_2 für G_2*

e. *fasse entsprechende Blätter zusammen (Facetten entsprechend, die durch Kante aus Schritt 3 getrennt sind)*

f. *→ Suchstruktur D_0 für G_0 durch Anhängen der in Schritt 2 entfernten Dreiecke*





Höhe des 'Baumes' = Anzahl der Vergröberungsschritte
in jedem wird die Anzahl der Knoten um den Faktor $1-c$ *gesenkt*
$n \to (1-c)*n \to ... \to (1-c)^h * n \Rightarrow h = log_{1/(1-c)} n/3 = O(logn)$

Zeit

Zur Konstruktion der Suchstruktur (Vorverarbeitung)

a. $O(n)$

b. $O(n)$

c. $O(n)$ *Facetten konstanter Größe* $O(n)$

d. $T((1-c)*n)$

e. $O(n)$

f. $O(n)$

$T(n) = T((1-c)*n) + b*n$ *für Konstante* b
$T(n) = b*n + (1-c)*b*n + (1-c)^2*b*n... \leq b*n*\sum_{i=0}^{\infty}(1-c)^i = O(n)$
(da $(1-c) < 1$ *)*

Platzbedarf

Gesamtzahl der vorkommenden Kanten ist $O(n)$ *wegen der Laufzeit des Algorithmus*
Von jedem Knoten des Baumes: Zeiger auf neu hinzukommenden Kanten: $O(n)$

Suchzeit

Weg im Baum wird durchlaufen, konstante Zeit pro Knoten $O(\log n)$

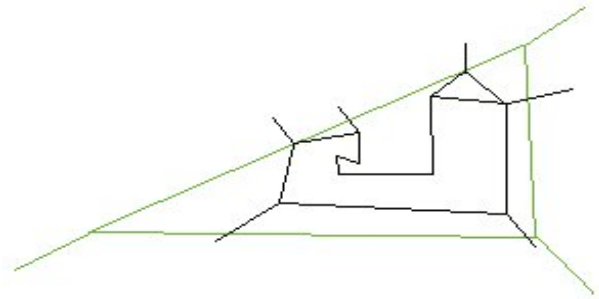
Verallgemeinerung auf nichttriangulierte ebene Unterteilung ist leicht möglich durch Triangulierung der Facetten.
Kantenanzahl erhöht sich nur linear.

a. *lege Dreieck D um alle beschränkten Teile der ebenen Unterteilung*

b. *füge alle Schnittpunkte mit Strahlen/Geraden als Knoten hinzu*

c. *drei neue Strahlen von Ecken von D*

d. *triangulieren alle Polygone im Inneren von $D \Rightarrow$ triangulierte Unterteilung in $O(n)$ Zeit möglich*



Die Triangulierung einfacher Polygone in Linearer Zeit nach Chazelle '90 (sehr kompliziert)

Satz

Zu einer ebenen Unterteilung der Größe n kann in $O(n)$ Zeit eine Datenstruktur konstruiert werden, die eine point-location-Anfrage in $O(\log n)$ Zeit beantwortet.

Beispiel:

Post-Office-Problem:
gegeben: Menge $S \subset \mathbb{R}^2, |S| = n$
gesucht: Datenstruktur, die zu einem gegebenen Anfragepunkt effizient den nächsten Punkt $\in S$ findet

Kombination vom Voronoi-Diagramm + point-location
Vorverarbeitung: $O(n \log n)$ fr VD, $O(n)$ für DS $\rightarrow O(n \log n)$
Speicher: $O(n)$
Suchzeit: $O(\log n)$

3.5 Sweepline-Verfahren

Beispiel *n Strecken in der Ebene, finde alle Schnittpunkte*
'brute force' Algorithmus benötigt $O(n^2)$
... es geht auch nicht besser, falls es $\Omega(n^2)$ *Schnittpunkte gibt,*
aber:
angenommen es gibt k Schnittpunkte, kann man es in subquadratischer Zeit in n und k lösen? (k ist in Anwendungen meist wesentlich kleiner als n^2*)*

Schnittpunkte von n Strecken bestimmen:
Die Strecken sind gegeben als Anfangs- und Endpunkt. Es gibt Verfahren, die können alle Schnittpunkte in $O(n \log n + k)$ *Zeit berechen. Hierbei ist* k *die Anzahl der Schnittpunkte. Wir lernen hier ein Verfahren kennen, dass* $O((n + k) \log n)$ *Zeit braucht. Solch einen Algorithmus nennt man* output sensitiv.

Idee: mit einer senkrechten Geraden, von links nach rechts, über die Szene „hinwegfegen". Dabei merken und aktualisieren wir (im Sweepline Status, kurz SLS) die Reihenfolge der Strecken, die von der Sweepline geschnitten werden. Diese ändert sich bei Endpunkten der Strecken und bei Schnittpunkten. Dies sind die sogenannten Ereignispunkte bzw „event points"(e.p.). Diese Ereignispunkte, genauer gesagt ihre x-Koordinaten, werden in einer Datenstruktur Namens „event point schedule" (EPS) verwaltet. An jedem Ereignispunkt wird der SLS aktualisiert (Strecken werden entfernt, hinzugefügt oder die Reihenfolge von ihnen getauscht). Als Datenstruktur für den SLS nimmt man am besten einen balancierten Scuhbaum (z.B.: AVL-Baum). Die Datenstruktur für den EPS ist eine Prioritätswaretschlange (Heap).

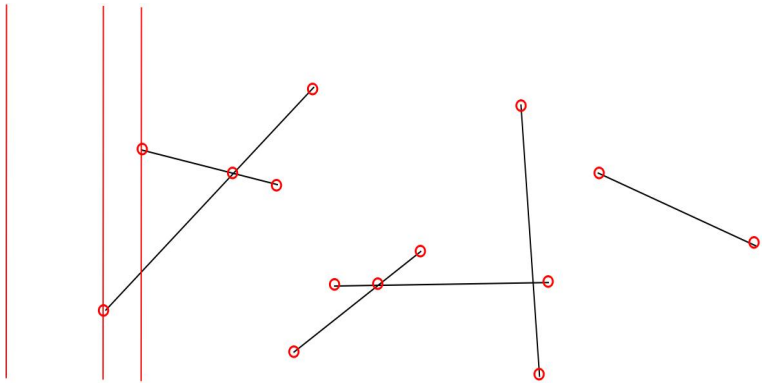


Abbildung 3.10: Die Schnittpunkte der schwarzen Linien sollen bestimmt werden. Die rote Sweepline „fegt" von links nach rechts über die Szene. Die rote Kreise sind die event points

Der genaue Algorithmus sieht wie folgt aus:

```
SLS := {-infinty, + infinity}
EPS := Menge der 2n Endpunkte
while EPS != leer  do{
    p:= streich Minimum von EPS
    if p linker Endpunkt einer Strecke e aus S then {
        suche oberen und unteren Nachbarn e' und e''
        von e auf der Sweepline und füge e in SLS ein.
        teste ob e mit e' und e'' einen Schnittpunkt hat
        und füge diese gegebenenfalls in EPS ein.
    }
    if p rechter Endpunkt von e then{
```

```
        streiche e aus SLS
        teste ob e' und e'' einen Schnittpunkt haben und füge diesen
        gegebenfalls in EPS ein
    }
    if p Schnittpunkt zweier Strecken e1 und e2 then {
    vertausche e1 und e2 in SLS und gib p aus.
    teste e1 und e1 mit neuen Nachbarn auf neue Schnittpunkte
    udn füge diese in EPS ein
    }
}
```

Laufzeit:

a. *EPS initialisieren:* $O(n)$

b. *Zeit pro Einfügung/Streichung oder Vertauschung* $O(\log n)$

c. *2. muss für jeden EPS (dass sind 2n+k) durchgeführt werden*

damit ergibt sich insgesamt eine Laufzeit von $O((n+k)\log n)$

Sweepline - Verfahren zur Triangulierung einfacher Polygone

Ein einfache Polygon zu triangulieren bedeuted es in Dreiecke zu unterteilen, deren Ecken ausschliesslich Ecken des Polygons sind.

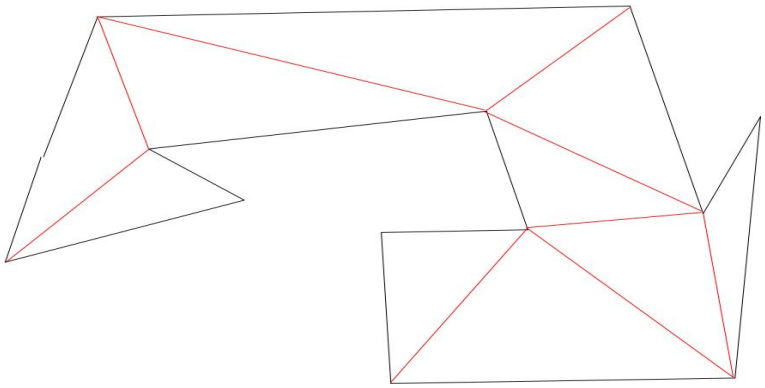


Abbildung 3.11: Beispiel für die Triangulierung eines einfachen Polygons

Konstruktion der Triangulierung:
EPS initialisiert mit den x-Koordinaten der Ecken des Polygons.
Wir unterscheiden drei Arten von Ecken, wie sie in Abbildung 3 zu sehen sind.

SLS enthält „aktive" Kanten, das sind Kanten die die Sweepline gerade schneiden (siehe Abbildung 4).

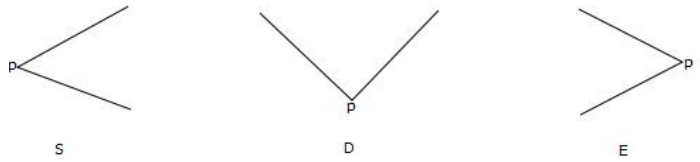


Abbildung 3.12: S steht für Startecke, D für Durchgangsecke und E für Endecke

Abbildung 3.13: Intervalle die mit a markiert sind befinden sich ausserhalb des Polygons; die mit i markierten innerhalb

Zu jedem i- Intervall merken wir uns einen Polygonzug $v_1, \cdots, v_k$ und einen Zeiger auf den Punkt des Polygonzuges der am weitesten rechts liegt.

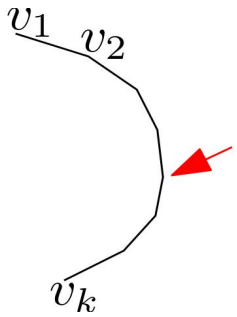


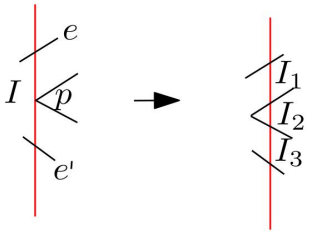
Abbildung 3.14: Der Polygonzug der für sich jedes i-Intervall gemerkt wird, mit dem Zeiger auf den rechtesten Punkt

Es gilt: $\angle(v_i, v_{i+1}, v_{i+2}) \geq \pi$
Folgende Änderungen müssen bei Eventpoints geamcht werden:

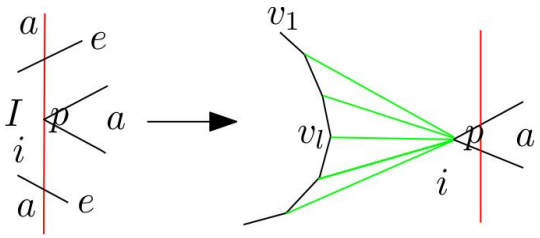
a. p ist ein S-Punkt:

(a) I ist ein a-Intervall:
Spalte I in I_1, I_2, I_3. Hierbei werden I_1 und I_3 a-Intervalle und I_2 eine i-Intervall.
Der Polygonzug für I_2 besteht nur aus p. p ist gleichzeitig auch der

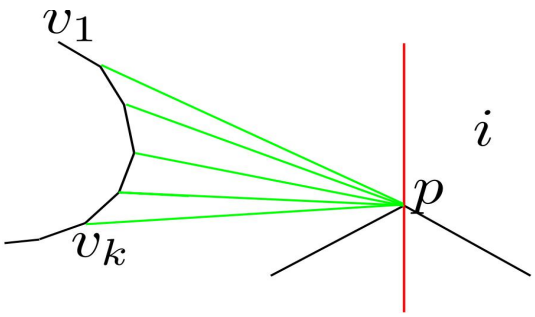
rechteste Punkt des Polygonzuges.



(b) *I ist ein i-Intervall:*
sei v_l rechtester Knoten des Polygonzugs von I. Die Kante $\overline{pv_l}$ wird eine Kante der Triangulierung.
Alle Kanten $\overline{pv_{l-1}}, \cdots$ und $\overline{pv_{l+1}}, \cdots$ werden kanten der Triangulierung, wobei hier im Polygonzug nach oben und nach unten gesucht wird, bis entweder das Ende erreicht ist oder $\angle(p, v_{i-1}, v_i) < \pi$, für die Punkte auf der oberen Seite des Polyginzuges, und $\angle(v, v_{i+1}, p) < \pi$, für die der unteren Seite, gilt.
Ausserdem wird das Intervall I gespalten in I_1, I_2, I_3, wobei I_1, I_3 i-Intervalle sind und I_2 ein a-Intervall.
I_1 wird der obere Rest des Polygonzuges von I. Diesem wird noch die Kante $\overline{v_- p}$ hinzugefügt, wobei v_- der unterste Knoten des Restes vom Polygozug ist.
I_3 wird der untere Rest des Polygonzuges von I.Diesem wird noch die Kante $\overline{v_+ p}$ hinzugefügt, wobei v_+ der oberste Knoten des Restes vom Polygozug ist.
Für beide neuen Polygonzüge ist der rechteste Punkt p.

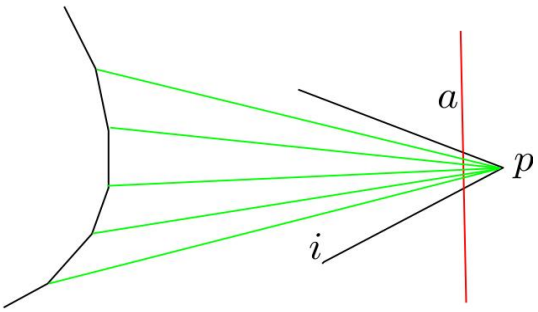


b. *p ist ein D Punkt:*
Füge soviele Triangulierungskanten (zwischen p und dem Polygonzug von I) ein wie möglich.
Neuer Polygonzug von I ist der Rest des alten und die Verbindung zwischen dem Rest und p

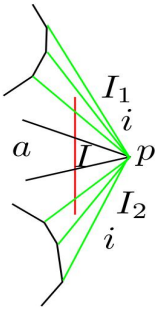


c. p ist ein E Punkt:

(a) I ist ein i-Intervall:
alle Triangulierungskanten einfügen. Das ist möglich.



(b) I ist ein a-Intervall:
zeichen Triangulierungskanten von p zu den Ecken des Polygonzugs von I_1 und I_2 soweit dies möglich ist.
verschmelze I_1 und I_2 zu einem Intervall I.
Der neue Polygonzu von I besteht aus dem oberen Rest des Polygonzugs von I_1, dem unteren Rest des Polygonzugs von I_2 die über den Punkt p verbunden werden. p ist dabei rechtester Punkt.



Kapitel 4

Algorithmische Spieltheorie

Definition 4.0.1. Spiele sind alle Situationen, wo verschiedene Parteien in Wettbewerb miteinander stehen und Aktionen ausführen.

Anwendungen:

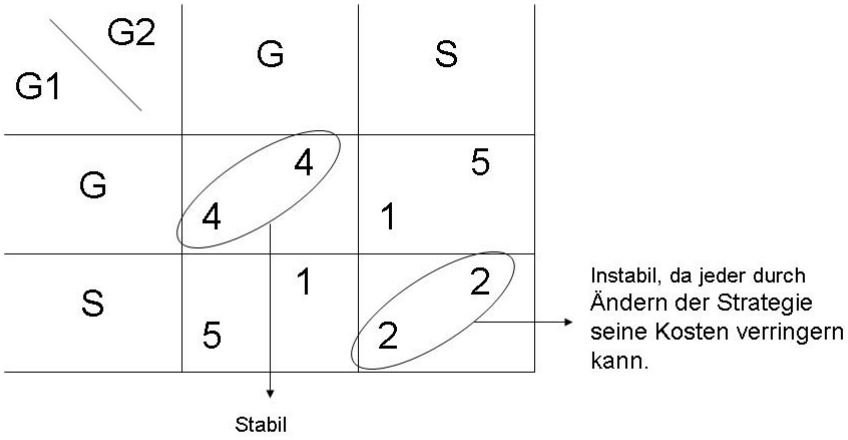
- *Wirtschaft: Börse, Verkauf, Versteigerung*
- *Internet: Wettbewerb der verschiedenen Nutzer um Ressourcen(Verbindungen, Rechenzeit, e.t.c.)*

Als erster hat John v. Neumann (1903-1957) eine Verbindung der Spieltheorie mit der Algorithmik hergestellt.

4.1 Beispiele

4.1.1 Gefangenendilemma (prisoners dilemma)

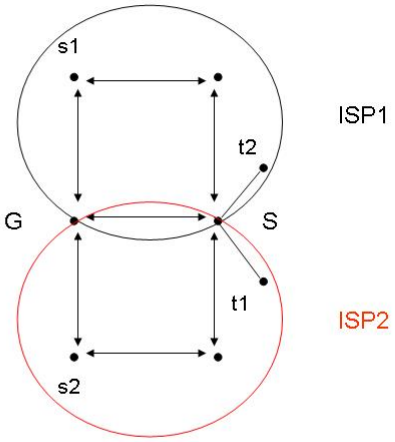
Zwei Gefangene sind wegen einer Straftat vor Gericht. Sie können wählen zwischen Gestehen und Schweigen.
Resultat (in Jahren Gefängnis):



Jeder hat zwei Strategien: Gestehen oder Schweigen

4.1.2 ISP routing

ISP steht für Internet service provider. In unserem Beispiel haben wir zwei davon: ISP_1 und ISP_2. Die Netzwerke sehen folgendermaßen aus.



Der Internetverkehr soll von s_1 nach t_1 und von s_2 nach t_2 stattfinden.
Die Kosten für $ISP_{1/2}$ entsprechen der ANzahl der Kanten, die in jedem Netz durchlaufen werden.
G und S sind die Transitknoten zwischen den Netzen.
ISP_1 ist für $s_1 \rightarrow t_1$ zuständig.

Egoistische Strategie: Schicke nach G. Kostet 1 für ISP_1 und 3 für ISP_2
Selbstlose Strategie: Schicke nach G. Kostet 2 für ISP_1 und 0 für ISP_2
Analog verhält es sich mit ISP_2 $s_2 \rightarrow t_2$

ISP1 \ ISP2	G	S
G	3 1	0 2
S	2 0	1 3

4.1.3 Umweltschutz-Spiel (viele Teilnehmer)

n Länder „spielen". Jedes kann Umweltschutz betreiben. Das verursacht 3 *eigene Kosten und* 0 *Kosten für andere. Oder es tut nichts, was* 1 *eigene Kosten und* 1 *Kosten für andere verursacht.*

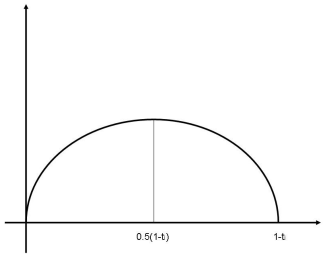
Angenommen k Länder betreiben keinen Umweltschutz, dann entstehen für sie jeweils Kosten k.
Die Kosten der anderen $n - k$ Länder betragen je $k + 3$. Jedes dieser Länder kann durch Änderung der eigenen Strategie seine Kosten auf $k + 1$ senken.
Die einzige Stabile Lösung ist $k = n$, also haben alle Länder Kosten n.
Im Fall $k = 0$ haben alle Länder Kosten 3*, ist aber keine Stabile Lösung.*

4.1.4 Tragedy of the commons

Bei diesem Spiel hängt die Strategie davon ab was die anderen tun.
n Spieler wollen eine gemeinsame Ressource benutzen. Sei dies ein Kanal mit der Bandbreite 1*. Spieler i kann $x_i \in [0, 1]$ Kapazität wählen. Es gibt also unendlich viele Strategien! Angenommen die Qualität nimmt bei starker Nutzung des Kanals ab, dann ergibt sich folgender Gewinn für den Spieler:*

$$Gewinn = \begin{cases} 0 \quad , falls \quad \sum_{j=1}^{n} x_j > 1 \\ x_i * (1 - \sum_{j=1}^{n} x_j) \quad , sonst \end{cases}$$

Sei $t_i = \sum_{j=1}^{n} x_j$ *Fluss aller anderen Spieler.*
Wenn Spieler i x *Einheiten schickt, folgt daraus der Gewinn* $x(1 - t_i - x)$



Wie man sieht ist $x = 0.5(1 - t_i)$ *optimal.*
Falls alle diese Strategie benutzen ergibt sich:

$$x_i = \frac{(1 - \sum_{j \neq i} x_j)}{2} \Longleftrightarrow 2x_i + \sum_{j \neq i} x_j = 1$$

$$\begin{pmatrix} 2 & 1 & ... & ... & 1 \\ 1 & 1 & & & \\ 1 & & \ddots & & \\ \vdots & & & \ddots & 1 \\ 1 & ... & ... & 1 & 2 \end{pmatrix} * \begin{pmatrix} x_1 \\ \vdots \\ \vdots \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} 1 \\ \vdots \\ \vdots \\ \vdots \\ 1 \end{pmatrix}$$

Eindeutige Lösung ist

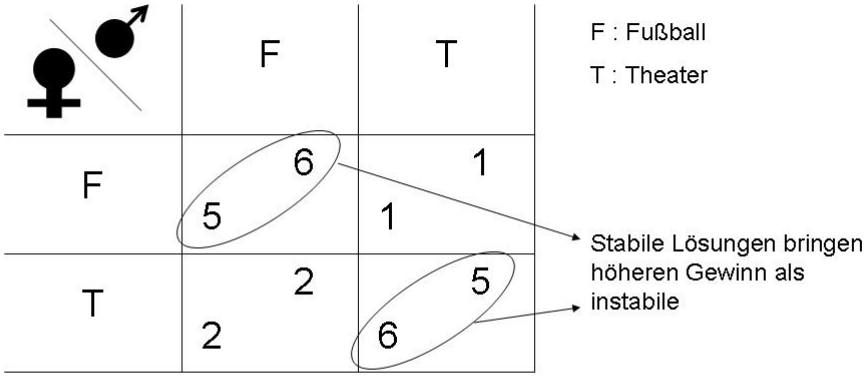
$$x_i = \frac{1}{n+1} \quad für \quad i = 1, ..., n$$

Der Gewinn pro Spieler ist $\frac{1}{(n+1)^2}$.
Der Gesammtgewinn ist $\frac{n}{(n+1)^2} < \frac{1}{n+1}$

Würde man dafür sorgen, dass $\sum_{i=1}^{n} x_i = \frac{1}{2}$, *dann wäre der Gesammtgewinn* $\frac{1}{4}$

4.1.5 Kampf der Geschlechter (battle of the sexes)

Kampf der Geschlechter ist ein Koordinationsspiel, bei dem die beste Lösung herauskommt wenn beide kooperieren.



4.1.6 Übereinstimmende Münzseiten

Es spielen zwei Spieler. Jeder wählt eine Münzseite.
*Spieler*1 *gewinnt wenn die Seiten gleich sind.*
*Spieler*2 *gewinnt wenn die Seiten ungleich sind.*

S1 \ S2	Kopf	Zahl
Kopf	-1 1	1 -1
Zahl	1 -1	-1 1

Keine stabile Lösung. Spiel eher dem Zufall überlassen.

4.2 Definitionen

Es gibt n Spieler. Jeder hat Menge von möglichen Strategien $S_1, ..., S_n$.
$(s_1, ..., s_n) \in S_1 \times ... \times S_n = S$ *: Strategievektor*
Präferenzen der Ereignisse für Spieler i werden ausgedrückt durch die Gewinnfunktion $U_i : S \to \mathbb{R}$ *und den Kostenfaktor* $c_i : S \to \mathbb{R}$

$$U_i(s) = -c_i(s), \forall s \in S$$

4.3 Lösungskonzepte

Definition 4.3.1. Ein Spiel hat eine Lösung mit **dominanter Stretegie**, falls jeder Spieler eine optimale Strategie hat, die nicht von den Strategien der anderen abhängt. Das Ergebniss ist natürlich von den Strategien der anderen abhängig, aber mit der dominanten Strategie ist das eigene Ergebnis immer besser als mit allen anderen Strategien.

$$\forall i \in \{1, ..., n\} : \exists s_i \in S_i : \forall s_i' \in S_i : \forall j \neq i : \forall s_j \in S_j : u_i(s_1, ..., s_i, ..., s_n) \geq u_i(s_1, ..., s_i', ..., s_n)$$

Beispiele dafür sind Gefangenendilemma und Umweltsimulation.

Vorlesung vom 2008-07-11
Skript erstellt von David Kaltschmidt

4.3.1 Entwurf von Spielen mit dominanter Strategie

Vickrey-Auktion

Jeder der n Spieler gibt einen verschlossenen Umschlag mit seinem Gebot ab. Der Höchstbietende bekommt den Zuschlag für den zweithöchsten Preis. Dies erzwingt eine dominante Strategie.

Bemerkung. Bei einer traditionellen Auktion erhält der Höchstbieter den Zuschlag zu dem Preis, den *er* geboten hat.

Wieso ist das sinnvoll? Jeder Spieler weiss, dass er nur den zweithöchsten Preis zahlen muss. Bei der traditionellen Variante jedoch spekulieren die Mitbieter über die Gebote der anderen. Das Ergebnis der Spekulation wird kleiner-gleich dem Preis sein, den sie maximal bieten würden.

Proposition 4.3.2. *Bei einer Vickrey-Auktion ist die dominante Strategie, seinen persönlichen Maximalpreis zu bieten.*

Warum ist das eine dominante Strategie? Jeder Spieler i ordnet dem Gemälde einen Wert v_i zu. Sein ***Gewinn*** *wird 0, falls er den Zuschlag nicht erhält. Ist er jedoch Höchstbieter und muss den Preis p zahlen, so ist sein Gewinn $v_i - p$. Bei einer Vickrey-Auktion ist es dann eine dominante Strategie für Spieler i, den Preis v_i zu bieten. Dazu betrachte folgende Fälle:*

a. *Angenommen Spieler i würde einen Preis $p_i > v_i$ bieten. Wenn dann ein zweiter Spieler k einen Preis $p_i > p_k > v_i$ bieten würde, dann würde Spieler i einen Gewinn von $v_i - p_k < 0$ machen (Verlust, da mehr gezahlt wurde, als das Gemälde dem Spieler wert war).*
b. *Angenommen Spieler i würde einen Preis $p_i > v_i > p_k$ bieten, dann würde Spieler i einen Gewinn von $v_i - p_k > 0$ machen.*
c. *Angenommen Spieler i würde einen Preis $v_i > p_i > p_k$ bieten, dann würde Spieler i einen Gewinn von $v_i - p_k > 0$ machen.*
d. *Wenn Spieler i nicht Höchstbietender ist, ist sein Gewinn 0.*

Der Gewinn von Punkt 3 hätte aber auch erreicht werden können, wenn Spieler i den Preis $p_i = v_i$ geboten hätte. Das Verhalten von 3 hätte nur riskiert, die Auktion zu verlieren (wenn $p_i < p_k < v_i$). Direkt v_i zu bieten ist also eine dominante Strategie, da man seinen Gewinn durch eine andere Strategie nicht vergrößern kann.

Weniger anbieten, als einem das Objekt Wert ist, bringt also nichts (man verschlechtert sich nicht, wenn man direkt sein Maximalgebot abgibt). Diese Strategie führt auch zum ***sozialoptimalen Ergebnis****: Derjenige, der dem Objekt den größten Wert zuweist, bekommt es. Bei der traditionellen Variante muss dies nicht so sein, da es durch die Spekulationen zu Verzerrungen kommen kann (in der Regel gewinnt der beste Spekulant). Desweiteren kommt es bei steigender Spieleranzahl die zu einer Annäherung von Wert v_i und dem Preis p_k, den Spieler i dann zu zahlen hat.*

Nash-Gleichgewicht

Spiele mit dominanter Strategie sind einfach für den Spieler und eher selten. Bei komplexere Spielen wird, statt nach einer Strategie, nach einem Strategievektor gesucht.

Ein Zustand (Strategie-Vektor, entspricht Kästchen in Strategiematrix aus den vorherigen Beispielen) indem sich niemand durch Wechsel seiner Strategie verbessern kann (die Strategie der anderen bleibt gleich), heißt NASH-*Gleichgewicht*[1].

Definition 4.3.3. Ein Strategievektor $s = (s_1, \ldots, s_n)$ ist genau dann im Nash-Gleichgewicht, wenn sich kein Spieler durch alleiniges Abweichen besserstellen kann, das heißt genau dann, wenn

$$\forall i \in 1 \ldots n \, \forall s_i \in S_i' : \quad u_i(s_1', \ldots, s_{i-1}', \sim_{\beth}, s_{i+1}', \ldots s_n') \leq u_i(s_1', \ldots, s_{i-1}', s_i', s_{i+1}', \ldots s_n')$$

Niemand kann sich also aus eigener Kraft verbessern (ohne dass andere ihre Strategie ändern). Entscheidend für das Gleichgewicht ist dabei die ***Stabilität des Zustands****, nicht der Gewinn.*

Beispiel. Gefangenendilemma: Solange nicht beide gestehen, ist der Zustand nicht im Nash-Gleichgewicht
Kampf der Geschlechter: Beide gemeinsamen Aktivitäten sind im Nash-Gleichgewicht.
Münzauswahl: Es ist kein Nash-Gleichgewicht möglich (in jedem Fall kann sich immer einer verbessern).

Proposition 4.3.4. *Ein Spiel mit dominanter Strategielösung hat auch ein Nash-Gleichgewicht.*

Wenn alle Spieler die dominante Strategielösung wählen, dann kann sich keiner mehr durch Änderung seiner Strategie verbessern.

Proposition 4.3.5. *Falls man eine* ***streng*** *dominante Strategielösung hat (wenn man von dieser dominanten Strategie abweicht, verschlechtert man sich), dann existiert ein eindeutiges Nash-Gleichgewicht.*

Wenn alle Spieler diese streng dominante Strategielösung wählen, erhält man einen Strategievektor, der nur diese streng dominante Strategie enthält. Angenommen, es gebe noch ein anderes Nash-Gleichgewicht, könnte ein Spieler zu diesem Wechseln. Er kann sich jedoch nur verschlechtern, da seine neue Strategie nicht die streng dominante sein kann.

Bisher wurden nur ***reinen Strategien*** *behandelt, das heißt es wird deterministisch eine Strategie ausgewählt. Es gibt auch noch die* ***gemischte Strategie****, bei der aus der Menge der Strategien gemäß einer Wahrscheinlichkeitsverteilung zufällig eine Strategie ausgewählt wird. Die eigentlich "Strategie" ist dabei, die Wahrscheinlichkeitsverteilung günstig zu wählen. Das Nash-Gleichgewicht bezüglich einer gemischten Strategie ist der Fall, wenn keiner seine Wahrscheinlichkeitsverteilung so ändern kann, dass der erwartete Gewinn sich erhöht falls die anderen ihre Strategie beibehalten.*

Beispiel. Übereinstimmende Münzauswahl: hat ein Nash-Gleichgewicht für gemischte Strategien bei der Wahrscheinlichkeitsverteilung $(\frac{1}{2}, \frac{1}{2})$ für beide Spieler. Keiner kann sich verbessern, der erwartete Gewinn ist immer 0.

[1] John Forbes Nash, Jr. (* 13. Juni 1928 in Bluefield, West Virginia) ist ein US-amerikanischer Mathematiker, der besonders in den Bereichen Spieltheorie und Differentialgeometrie sowie auf dem Gebiet der partiellen Differentialgleichungen arbeitet. Im Jahr 1994 erhielt er zusammen mit Reinhard Selten und John Harsanyi den Nobelpreis für Wirtschaftswissenschaften für die gemeinsamen Leistungen auf dem Gebiet der Spieltheorie. Damit ist er einer der wenigen Mathematiker, die einen Nobelpreis erhalten haben, obwohl es keinen speziellen Nobelpreis für Mathematik gibt. [Wikipedia]

Beispiel zu gemischten Strategien: Chicken (Feigling)

Zwei Autos fahren aufeinander zu, wer zuerst ausweicht, hat verloren. Wenn keiner ausweicht, haben beide verloren.[2] *Die Strategien sind W für weiterfahren, und A für ausweichen.*

Spieler 1 / 2	*W*	*A*
W	*-100/-100*	*2 / 0*
A	*0 / 2*	*1 / 1*

Es existieren zwei Nash-Gleichgewichte für reine Strategien: (W, A) und (A, W). Für gemischte Strategien gibt es ein weiteres Nash-Gleichgewicht gegeben durch die Wahrscheinlichkeitsverteilung der Strategien. Die Wahrscheinlichkeiten für die Ereignisse A und W seien für die beiden Spieler wie folgt:

$$\textit{Spieler } 1 : p(W) = \epsilon_1, p(A) = 1 - \epsilon_1 \quad \textit{Spieler } 2 : p(W) = \epsilon_2, p(A) = 1 - \epsilon_2$$

Der erwartete Gewinn für Spieler 1 errechnet sich dann wie folgt:

$$\begin{aligned} u_1 &= -100\epsilon_1\epsilon_2 + 2\epsilon_1(1-\epsilon_2) + 1(1-\epsilon_1)(1-\epsilon_2) && (4.1)\\ &= \epsilon_1(-100\epsilon_2 + 2(1-\epsilon_2) - 1 + \epsilon_2) + 1 - \epsilon_2 && (4.2)\\ &= \epsilon_1(1 - 101\epsilon_2) + 1 - \epsilon_2 && (4.3) \end{aligned}$$

Bei einem Nash-Gleichgewicht muss gelten, dass ϵ_1 sich nicht ändern kann, so dass sich Spieler 1 verbessert. Die einzige Möglichkeit das zu erreichen, tritt ein wenn $1 - 101\epsilon_2 = 0$ ist, da sonst Spieler 1 durch Erhöhung oder Erniedrigen von ϵ_1 seinen Gewinn erhöhen könnte. Im Nash-Gleichgewicht folgt dann aus $1 - 101\epsilon_2 = 0$:

$$\epsilon_2 = 1/101 \textit{ sowie analog } \epsilon_1 = 1/101$$

Wenn Spieler 1 und 2 ihre Wahrscheinlichkeitsverteilungen so wählen, kann sich keiner verbessern. Im Nash-Gleichgewicht für gemischte Strategien haben bei der Wahrscheinlichkeitsverteilung $(\frac{1}{101}, \frac{100}{101})$ beide Spieler einen erwarteter Gewinn von $\frac{100}{101}$. Die Wahrscheinlichkeit für einen Crash ist $\frac{1}{101^2}$.

Satz 4.3.6. *Satz von Nash (1951): Jedes Spiel mit einer endlichen Menge von Spielern und einer endlichen Menge von Strategien besitzt ein Nash-Gleichgewicht bezüglich gemischter Strategien.*

[2] Jede direkte Konfrontation, die einen fatalen Ausgang hat, falls beide Mitspieler stur bleiben, kann so modelliert werden. Vgl. Kubakrise

4.4 Gleichgewicht finden

Wie ist die Komplexität in einem gegebenen Spiel ein Gleichgewicht zu finden?

Definition 4.4.1. Zwei-Personen-Nullsummen-Spiele

2 Spieler Sp_1, Sp_2

$\forall s \in S_1 \times S_2 : u_1(s) + u_2(s) = 0$

kann repräsentiert werden durch Matrix A mit u_1(s)

Seien p^*, q^* die Verteilung, die Sp_1 und Sp_2 beim Nash-GG (existiert nach Nash) benutzen, dann ist der erwartete Gewinn für Sp_1 $\sum_{i,j} p_i A_{ij} q_j$

$p^* = (p_1, ..., p_n)$

$q^* = (q_1, ..., q_m)$

$v^* = p^{*T} A q^*$

O.B.d.A ist p dem Sp_2 bekannt, denn bei Nash-GG kann er sich dadurch auch nicht verbessern.

Der Zeilenvektor $p^T A$ gibt für Sp_2 an wie viel Verlust seine verschiedenen Strategien zur Folge haben.

Sp_2 wählt natürlich das Minimum. Sp_1 wählt eine gemischte Strategie, so dass dieses Minimum maximiert wird.

$$\begin{aligned} \max & \, v \\ p & \geq 0 \\ \sum_{i=1}^{n} p_i & = 1 \\ (p^T)_j & \geq v \text{ für } j = 1, ..., m \end{aligned}$$

v_1 muss kleiner v^* sein, weil Sp_1 so spielen kann, dass er Gewinn v_1 macht unabhängig von der Strategie von Sp_2.

Wenn allerdings Sp_2 p^* kennt, dann wählt er im GG eine Strategie mit

$\min(p^* A)_j \Rightarrow v^* \leq v_1$ *also* $v^* = v_1$

LP liefert Gewinn von Sp_1 im Nash-GG:

$$\begin{aligned} \min & \, vq & \geq & \; 0 \\ \sum_{j=1}^{m} & q_j & = & \; 1 \\ & (Aq)_i & \leq & \; v \; \forall i \end{aligned}$$

also $v_1 = v_2$

Satz: Die opt. Lösung der obigen LP liefert die W-Verteilung für das Nash-GG.

Also: In diesem Fall Nash-GG in polynomieller Zeit bestimmbar.

4.5 Problem NASH

Gegeben ist ein Spiel (endlich viele Spieler und endlich viele Strategien) durch Matrizen. Finde ein Nach-GG bezüglich einer gemischten Strategie.

Es ist nicht bekannt, ob NASH in polynomieller Zeit lösbar ist.

Gegeben eine gemischte Strategie, so ist in polynomieller Zeit entscheidbar, ob sie ein Nash-GG ist.

Anscheinend ist das Problem also schwer berechenbar, ab leicht verifizierbar. Somit ist es den NP-vollständigen Problemen ähnlich.

Außerdem gilt: Falls der Träger der gemischten Strategie vorgegeben wird, d.h. die reinen Strategien, mit Wahrscheinlichkeit > 0, dann lässt sich in polynomieller Zeit entscheiden, ob Nash-GG mit diesem Träger existiert und es kann auch berechnet werden.

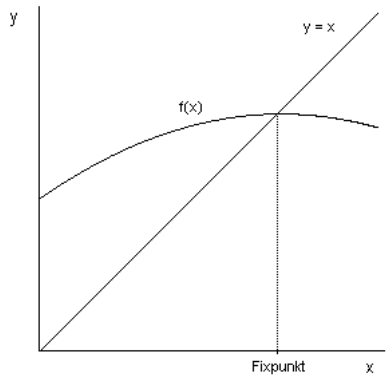
Folgende Probleme sind NP-vollständig:

Gegeben ein Spiel mit 2 Spielern. Hat es:

- *mehr als zwei Nash-GG?*
- *Nash-GG mit Gewinn $\geq c$ für Sp_1, für gegebenes $c \in \mathbb{R}$?*
- *Nash-GG mit Gewinn $\geq c$ für für beide Spieler, für gegebenes $c \in \mathbb{R}$?*
- *Nash-GG mit $\geq k$ Wahrscheinlichkeiten > 0 für gegebenes $k \in \mathbb{N}$?*

Der Beweis für den Satz von Nash liefert keinen Algorithmus zum finden des Nash-GG, denn er ist nicht konstruktiv. Er beruht auf dem Brouwer'schen Fixpunktsatz:

Sei B_n (n-dimensionale Kugel) $= \{(x_1, ..., x_n) | \sum_{i=1}^{n} x_i^2 \leq 1\}$, dann hat jede stetige Abbildung $f : B_n \to B_n$ einen Fixpunkt d.h. $x \in B_n$ mit $f(x) = x$



Der Brouwer'sche Fixpunktsatz ist auch nicht konstruktiv.

Die Berechnung eines Fixpunktes kann sogar in polynomieller Zeit auf NASH reduziert werden.

NASH ist vollständig für die Klasse PPAD. Sie liegt zwischen P und NP. PPAD enthält außerdem:

- *FIXPUNKT finden für* $f : B_n \to B_n$
- *Ham-Sandwich-Cut: Gegeben n Punkte in d-dimensionalem Raum mit k Farben gefärbt. Frage: Gibt es eine Hyperebene, die alle Farbklassen halbiert? d.h. gleich viele auf beiden Seiten der Hyperebene*