6 Lineare Programmierung

Lineares Programm (LP): Finde optimale Werte für d **reelle Variablen** $x_1, \ldots, x_d \in \mathbb{R}$. Dabei soll eine **lineare Zielfunktion**

$$c_1 x_1 + \ldots + c_d x_d$$

für gegebene Koeffizienten $c_1, \ldots, c_d \in \mathbb{R}$ **minimiert** oder **maximiert** werden.

Es müssen m **lineare Nebenbedingungen** eingehalten werden. Für jedes $i \in \{1, \ldots, m\}$ sind Koeffizienten $a_{i1}, \ldots, a_{id} \in \mathbb{R}$ und $b_i \in \mathbb{R}$ gegeben. Eine Belegung der Variablen ist nur dann gültig, wenn sie die folgenden Nebenbedingungen einhält:

$$\begin{aligned} a_{11} x_1 + \ldots + a_{1d} x_d &\leq b_1 \\ &\vdots \\ a_{m1} x_1 + \ldots + a_{md} x_d &\leq b_m \end{aligned}$$

Statt $\leq$ ist auch $\geq$ erlaubt.

Sei $x^T = (x_1, \ldots, x_d)$ und $c^T = (c_1, \ldots, c_d)$.
Damit kann die **Zielfunktion als Skalarprodukt** $c \cdot x$ geschrieben werden.

Außerdem sei $A \in \mathbb{R}^{m \times d}$ die Matrix mit den Einträgen a_{ij} und $b^T = (b_1, \ldots, b_m) \in \mathbb{R}^m$.
Dann **entspricht jede Zeile der Matrix einer Nebenbedingung**.

Wir können die Nebenbedingungen als $Ax \leq b$ schreiben.

Lineares Programm

Eingabe:	$c \in \mathbb{R}^d$, $b \in \mathbb{R}^m$, $A \in \mathbb{R}^{m \times d}$
Lösungen:	alle $x \in \mathbb{R}^d$ mit $Ax \leq b$
Zielfunktion:	minimiere/maximiere $c \cdot x$

Beispiel: Maximaler Fluss

Eingabe: Flussnetzwerk $G = (V, E)$ mit Quelle $s \in V$ und Senke $t \in V$, Kapazitätsfunktion $c : E \to \mathbb{N}_0$

Aufgabe: Finde einen maximalen Fluss von s nach t in G.

Modellierung als LP:

Variablen: Für jedes $e \in E$ Variable $x_e \in \mathbb{R}$, die den Fluss auf e angibt.

Zielfunktion:

$$\sum_{e=(s,v)} x_e - \sum_{e=(v,s)} x_e$$

Nebenbedingungen:

$\forall e \in E : x_e \geq 0$ (Fluss nicht negativ)

$\forall e \in E : x_e \leq c(e)$ (Fluss nicht größer als Kapazität)

$\forall v \in V \setminus \{s, t\} : \sum_{e=(u,v)} x_e - \sum_{e=(v,u)} x_e = 0$ (Flusserhaltung)

kanonische Form	Gleichungsform
$\min c \cdot x$	$\min c \cdot x$
$Ax \leq b$	$Ax = b$
$x \geq 0$	$x \geq 0$

Sei $a_i = (a_{i1}, \ldots, a_{id})^T \in \mathbb{R}^d$ die i-te Zeile von A.

Transformationen

- „maximiere $c \cdot x$“ entspricht „minimiere $-c \cdot x$“.
- Variable x_i kann durch $x_i' - x_i''$ für zwei Variablen $x_i' \geq 0$ und $x_i'' \geq 0$ ersetzt werden.
- „$a_i \cdot x \geq b_i$“ entspricht „$-a_i \cdot x \leq -b_i$“.
- Gleichung $a_i \cdot x = b_i$ kann durch $a_i \cdot x \leq b_i$ und $a_i \cdot x \geq b_i$ ersetzt werden.
- $a_i \cdot x \leq b_i$ können wir durch $s_i + a_i \cdot x = b_i$ für eine **Schlupfvariable** $s_i \geq 0$ darstellen.

Geometrische Interpretation: Betrachte LP in kanonischer Form

Variablenbelegung $x \in \mathbb{R}^d$ **entspricht Punkt im** $\mathbb{R}^d$.

Eine Gleichung $a_i \cdot x = b_i$ definiert eine **affine Hyperebene** $\{x \in \mathbb{R}^d \mid a_i \cdot x = b_i\}$.

Jede solche affine Hyperebene definiert den **abgeschlossenen Halbraum**

$$\mathcal{H}_i = \{x \in \mathbb{R}^d \mid a_i \cdot x \leq b_i\}.$$

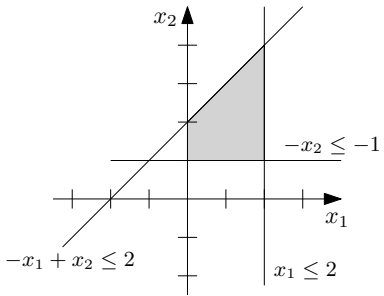
Eine Variablenbelegung $x \in \mathbb{R}^d$ **erfüllt genau dann Nebenbedingung i**, wenn $x \in \mathcal{H}_i$ gilt.

Eine Variablenbelegung $x \in \mathbb{R}^d$ ist genau dann **gültig**, wenn
$x \in \mathcal{P} := \mathcal{H}_1 \cap \ldots \cap \mathcal{H}_m \cap \mathbb{R}^d_{\geq 0}$ gilt.

Beispiel:

Betrachte lineares Programm mit den folgenden Nebenbedingungen

$$x_1 \geq 0, \qquad x_2 \geq 0,$$
$$x_1 \leq 2, \quad -x_2 \leq -1,$$
$$-x_1 + x_2 \leq 2$$



Wir können $\mathbb{R}^d_{\geq 0}$ als einen Schnitt von d Halbräumen darstellen:

$$\mathbb{R}^d_{\geq 0} = \{x \in \mathbb{R}^d \mid -x_1 \leq 0\} \cap \ldots \cap \{x \in \mathbb{R}^d \mid -x_d \leq 0\}.$$

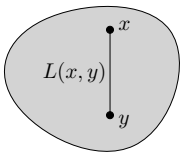
Somit ist $\mathcal{P}$ der **Schnitt von endlich vielen Halbräumen**.

Einen solchen Schnitt nennt man **Polyeder**. Wir sagen, dass ein lineares Programm **zulässig** ist, wenn sein **Lösungspolyeder** nichtleer ist.

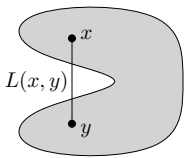
Eine Menge X heißt **konvex**, wenn für alle Punkte $x \in X$ und $y \in X$ gilt:

$$L(x, y) := \{\lambda x + (1 - \lambda)y \mid \lambda \in [0, 1]\} \subseteq X.$$

konvexe Menge



nichtkonvexe Menge



Lemma 6.1

Das Lösungspolyeder $\mathcal{P}$ ist konvex.

Theorem 6.2

Sei ein lineares Programm in kanonischer Form mit Lösungspolyeder $\mathcal{P}$ gegeben und sei $x \in \mathcal{P}$ eine lokal optimale Variablenbelegung. Dann ist x auch global optimal, d. h. es gibt kein $y \in \mathcal{P}$ mit $c \cdot y < c \cdot x$.

Ein LP heißt **unbeschränkt**, wenn der zu minimierende Zielfunktionswert innerhalb des Lösungspolyeders $\mathcal{P}$ beliebig klein werden kann. Ansonsten heißt es **beschränkt**.

Sei $c \cdot x$ eine beliebige lineare Zielfunktion und sei $w \in \mathbb{R}$ beliebig. Die Menge

$$\{x \in \mathbb{R}^d \mid c \cdot x = w\}$$

bildet eine **affine Hyperebene mit Normalenvektor c**.

Sei $c \cdot x$ eine beliebige lineare Zielfunktion und sei $w \in \mathbb{R}$ beliebig. Die Menge

$$\{x \in \mathbb{R}^d \mid c \cdot x = w\}$$

bildet eine **affine Hyperebene mit Normalenvektor c**.

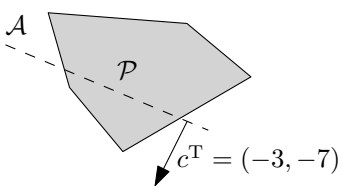
6.1 Grundlagen

Sei $c \cdot x$ eine beliebige lineare Zielfunktion und sei $w \in \mathbb{R}$ beliebig. Die Menge

$$\{x \in \mathbb{R}^d \mid c \cdot x = w\}$$

bildet eine **affine Hyperebene mit Normalenvektor c**.

1. Finde ein $w \in \mathbb{R}$, sodass $\{x \in \mathbb{R}^d \mid c \cdot x = w\} \cap \mathcal{P} \neq \emptyset$.



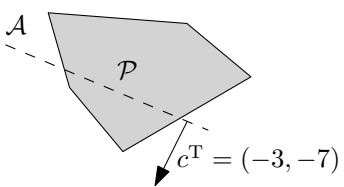
6.1 Grundlagen

Sei $c \cdot x$ eine beliebige lineare Zielfunktion und sei $w \in \mathbb{R}$ beliebig. Die Menge

$$\{x \in \mathbb{R}^d \mid c \cdot x = w\}$$

bildet eine **affine Hyperebene mit Normalenvektor c**.

1. Finde ein $w \in \mathbb{R}$, sodass $\{x \in \mathbb{R}^d \mid c \cdot x = w\} \cap \mathcal{P} \neq \emptyset$.
2. Verschiebe $\{x \in \mathbb{R}^d \mid c \cdot x = w\}$ solange parallel in Richtung $-c$ wie obiger Schnitt nichtleer ist.



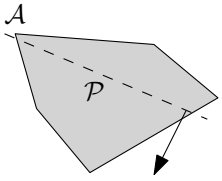
6.1 Grundlagen

Sei $c \cdot x$ eine beliebige lineare Zielfunktion und sei $w \in \mathbb{R}$ beliebig. Die Menge

$$\{x \in \mathbb{R}^d \mid c \cdot x = w\}$$

bildet eine **affine Hyperebene mit Normalenvektor c**.

1. Finde ein $w \in \mathbb{R}$, sodass $\{x \in \mathbb{R}^d \mid c \cdot x = w\} \cap \mathcal{P} \neq \emptyset$.
2. Verschiebe $\{x \in \mathbb{R}^d \mid c \cdot x = w\}$ solange parallel in Richtung $-c$ wie obiger Schnitt nichtleer ist.



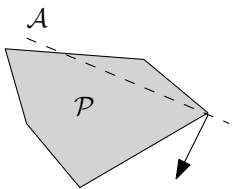
6.1 Grundlagen

Sei $c \cdot x$ eine beliebige lineare Zielfunktion und sei $w \in \mathbb{R}$ beliebig. Die Menge

$$\{x \in \mathbb{R}^d \mid c \cdot x = w\}$$

bildet eine **affine Hyperebene mit Normalenvektor c**.

1. Finde ein $w \in \mathbb{R}$, sodass $\{x \in \mathbb{R}^d \mid c \cdot x = w\} \cap \mathcal{P} \neq \emptyset$.
2. Verschiebe $\{x \in \mathbb{R}^d \mid c \cdot x = w\}$ solange parallel in Richtung $-c$ wie obiger Schnitt nichtleer ist.



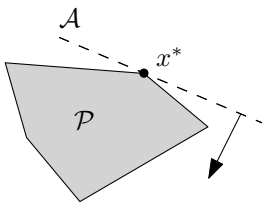
6.1 Grundlagen

Sei $c \cdot x$ eine beliebige lineare Zielfunktion und sei $w \in \mathbb{R}$ beliebig. Die Menge

$$\{x \in \mathbb{R}^d \mid c \cdot x = w\}$$

bildet eine **affine Hyperebene mit Normalenvektor c**.

1. Finde ein $w \in \mathbb{R}$, sodass $\{x \in \mathbb{R}^d \mid c \cdot x = w\} \cap \mathcal{P} \neq \emptyset$.
2. Verschiebe $\{x \in \mathbb{R}^d \mid c \cdot x = w\}$ solange parallel in Richtung $-c$ wie obiger Schnitt nichtleer ist.

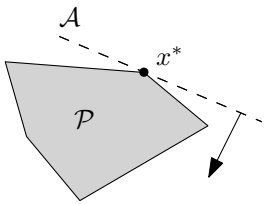


Sei $c \cdot x$ eine beliebige lineare Zielfunktion und sei $w \in \mathbb{R}$ beliebig. Die Menge

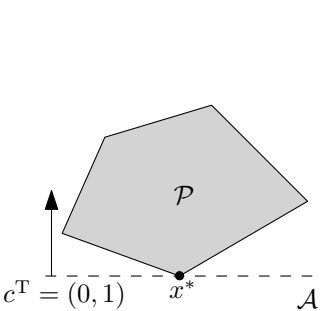
$$\{x \in \mathbb{R}^d \mid c \cdot x = w\}$$

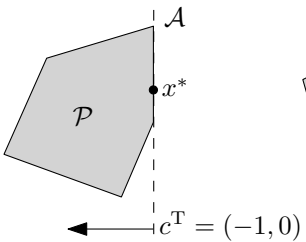
bildet eine **affine Hyperebene mit Normalenvektor c**.

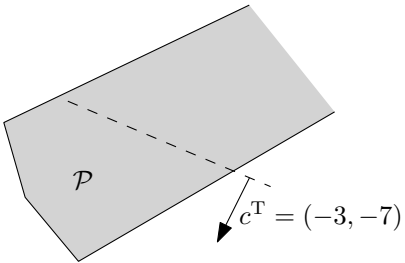
1. Finde ein $w \in \mathbb{R}$, sodass $\{x \in \mathbb{R}^d \mid c \cdot x = w\} \cap \mathcal{P} \neq \emptyset$.
2. Verschiebe $\{x \in \mathbb{R}^d \mid c \cdot x = w\}$ solange parallel in Richtung $-c$ wie obiger Schnitt nichtleer ist.
3. Terminiert der zweite Schritt nicht, so ist das LP unbeschränkt. Ansonsten sei $\mathcal{A} = \{x \in \mathbb{R}^d \mid c \cdot x = w\}$ die letzte Hyperebene mit $\mathcal{A} \cap \mathcal{P} \neq \emptyset$. Dann ist jeder Punkt $x^* \in \mathcal{A} \cap \mathcal{P}$ eine optimale Variablenbelegung des LPs.



Beispiele:


$\mathcal{P}$
$c^{\mathrm{T}} = (0, 1)$
x^*
$\mathcal{A}$


$\mathcal{A}$
$\mathcal{P}$
x^*
$c^{\mathrm{T}} = (-1, 0)$

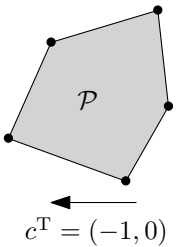

$\mathcal{P}$
$c^{\mathrm{T}} = (-3, -7)$

Simplex-Algorithmus (informelle Beschreibung)

Starte an einer Ecke $x^1 \in \mathcal{P}$ und teste, ob es eine benachbarte Ecke mit besserem Zielfunktionswert gibt.

Gibt es eine solche benachbarte Ecke $x^2 \in \mathcal{P}$, so mache mit x^2 analog weiter und teste, ob es eine **bessere benachbarte Ecke** gibt.

Den Übergang von einer Ecke zu einer besseren in der Nachbarschaft nennen wir auch **Pivotschritt**.

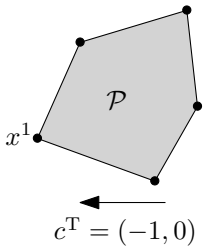


Simplex-Algorithmus (informelle Beschreibung)

Starte an einer Ecke $x^1 \in \mathcal{P}$ und teste, ob es eine benachbarte Ecke mit besserem Zielfunktionswert gibt.

Gibt es eine solche benachbarte Ecke $x^2 \in \mathcal{P}$, so mache mit x^2 analog weiter und teste, ob es eine **bessere benachbarte Ecke** gibt.

Den Übergang von einer Ecke zu einer besseren in der Nachbarschaft nennen wir auch **Pivotschritt**.

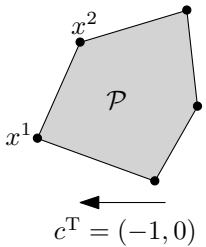


Simplex-Algorithmus (informelle Beschreibung)

Starte an einer Ecke $x^1 \in \mathcal{P}$ und teste, ob es eine benachbarte Ecke mit besserem Zielfunktionswert gibt.

Gibt es eine solche benachbarte Ecke $x^2 \in \mathcal{P}$, so mache mit x^2 analog weiter und teste, ob es eine **bessere benachbarte Ecke** gibt.

Den Übergang von einer Ecke zu einer besseren in der Nachbarschaft nennen wir auch **Pivotschritt**.

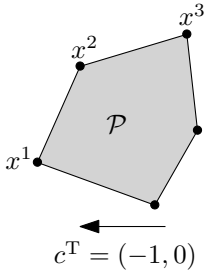


Simplex-Algorithmus (informelle Beschreibung)

Starte an einer Ecke $x^1 \in \mathcal{P}$ und teste, ob es eine benachbarte Ecke mit besserem Zielfunktionswert gibt.

Gibt es eine solche benachbarte Ecke $x^2 \in \mathcal{P}$, so mache mit x^2 analog weiter und teste, ob es eine **bessere benachbarte Ecke** gibt.

Den Übergang von einer Ecke zu einer besseren in der Nachbarschaft nennen wir auch **Pivotschritt**.

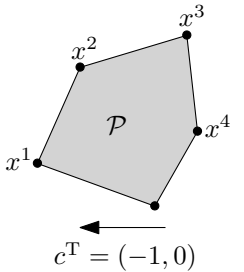


Simplex-Algorithmus (informelle Beschreibung)

Starte an einer Ecke $x^1 \in \mathcal{P}$ und teste, ob es eine benachbarte Ecke mit besserem Zielfunktionswert gibt.

Gibt es eine solche benachbarte Ecke $x^2 \in \mathcal{P}$, so mache mit x^2 analog weiter und teste, ob es eine **bessere benachbarte Ecke** gibt.

Den Übergang von einer Ecke zu einer besseren in der Nachbarschaft nennen wir auch **Pivotschritt**.

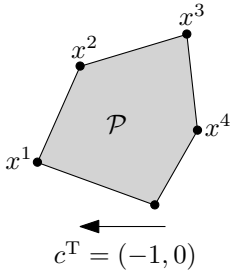


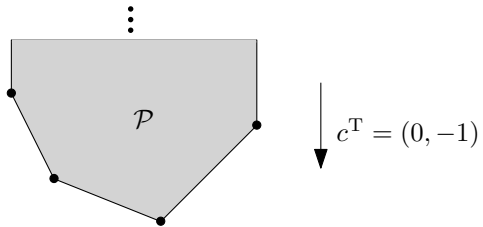
Simplex-Algorithmus (informelle Beschreibung)

Starte an einer Ecke $x^1 \in \mathcal{P}$ und teste, ob es eine benachbarte Ecke mit besserem Zielfunktionswert gibt.

Gibt es eine solche benachbarte Ecke $x^2 \in \mathcal{P}$, so mache mit x^2 analog weiter und teste, ob es eine **bessere benachbarte Ecke** gibt.

Den Übergang von einer Ecke zu einer besseren in der Nachbarschaft nennen wir auch **Pivotschritt**.



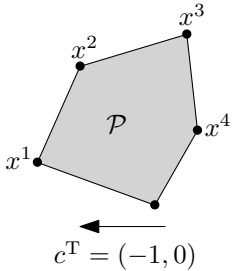


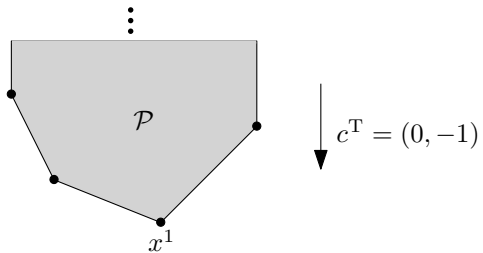
Simplex-Algorithmus (informelle Beschreibung)

Starte an einer Ecke $x^1 \in \mathcal{P}$ und teste, ob es eine benachbarte Ecke mit besserem Zielfunktionswert gibt.

Gibt es eine solche benachbarte Ecke $x^2 \in \mathcal{P}$, so mache mit x^2 analog weiter und teste, ob es eine **bessere benachbarte Ecke** gibt.

Den Übergang von einer Ecke zu einer besseren in der Nachbarschaft nennen wir auch **Pivotschritt**.



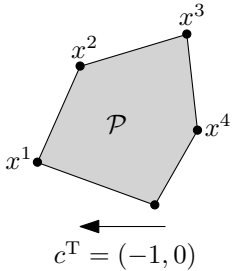


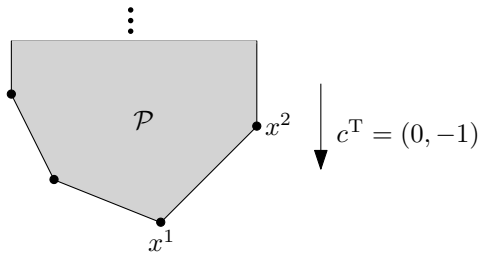
Simplex-Algorithmus (informelle Beschreibung)

Starte an einer Ecke $x^1 \in \mathcal{P}$ und teste, ob es eine benachbarte Ecke mit besserem Zielfunktionswert gibt.

Gibt es eine solche benachbarte Ecke $x^2 \in \mathcal{P}$, so mache mit x^2 analog weiter und teste, ob es eine **bessere benachbarte Ecke** gibt.

Den Übergang von einer Ecke zu einer besseren in der Nachbarschaft nennen wir auch **Pivotschritt**.





Finden einer initialen Lösung

Sei ein LP mit den Nebenbedingungen $Ax = b$ gegeben und sei o. B. d. A. $b \geq 0$.

Für die m Nebenbedingungen führen wir **Hilfsvariablen** $h_1 \geq 0, \ldots, h_m \geq 0$ ein.
Die NB $a_i \cdot x = b_i$ ersetzen wir für jedes i durch die NB $a_i \cdot x + \mathbf{h_i} = b_i$.
Wir ignorieren die Zielfunktion und definieren als **neue Zielfunktion** $h_1 + \ldots + h_m$.

Zulässige Lösung für dieses LP:

$h_i = b_i$ für jedes $i \in \{1, \ldots, m\}$ und $x_i = 0$ für alle $i \in \{1, \ldots, d\}$.

Initialisiere Simplex-Algorithmus mit dieser Lösung und berechne eine opt. Lösung (x^*, h^*).

Gilt $h^* \neq 0$, dann ist das ursprüngliche LP **nicht zulässig**.

Gilt $h^* = 0$, dann ist x^* eine **zulässige Lösung für das ursprüngliche LP**.

Theorem 6.3

In (nicht-degenerierten) LPs terminiert der Simplex-Algorithmus immer. Er findet eine optimale Lösung oder stellt fest, dass es keine oder keine optimale Lösung gibt.

Theorem 6.3

In (nicht-degenerierten) LPs terminiert der Simplex-Algorithmus immer. Er findet eine optimale Lösung oder stellt fest, dass es keine oder keine optimale Lösung gibt.

Theorem 6.5

Die Laufzeit eines einzelnen Pivotschrittes ist polynomiell in der Eingabelänge des LPs beschränkt.

Theorem 6.3

In (nicht-degenerierten) LPs terminiert der Simplex-Algorithmus immer. Er findet eine optimale Lösung oder stellt fest, dass es keine oder keine optimale Lösung gibt.

Theorem 6.5

Die Laufzeit eines einzelnen Pivotschrittes ist polynomiell in der Eingabelänge des LPs beschränkt.

Theorem 6.6

Für jedes $n \in \mathbb{N}$ gibt es ein LP in Gleichungsform mit $3n$ Variablen und $2n$ Nebenbedingungen, in dem alle Koeffizienten ganzzahlig sind und Absolutwert höchstens 4 haben, und auf dem der Simplex-Algorithmus $2^n - 1$ Pivotschritte durchführen kann.

Theorem

Existiert ein polynomieller Algorithmus, der entscheidet, ob ein LP eine Lösung besitzt oder nicht, so existiert auch ein polynomieller Algorithmus zur Optimierung von LPs.

6.3 Komplexität von linearer Programmierung

Theorem

Existiert ein polynomieller Algorithmus, der entscheidet, ob ein LP eine Lösung besitzt oder nicht, so existiert auch ein polynomieller Algorithmus zur Optimierung von LPs.

Theorem

Es existiert ein polynomieller Algorithmus zur Lösung von linearen Programmen.

6.3 Komplexität von linearer Programmierung

Theorem

Existiert ein polynomieller Algorithmus, der entscheidet, ob ein LP eine Lösung besitzt oder nicht, so existiert auch ein polynomieller Algorithmus zur Optimierung von LPs.

Theorem

Es existiert ein polynomieller Algorithmus zur Lösung von linearen Programmen.

Ellipsoidmethode (Khachiyan, 1979)

Innere-Punkte-Verfahren (Karmarkar, 1984)

Rucksackproblem:

$$\begin{aligned} \text{maximiere} \quad & p_1 x_1 + \cdots + p_n x_n \\ \text{sodass} \quad & w_1 x_1 + \cdots + w_n x_n \leq t, \\ & \forall i : x_i \in \{0, 1\}. \end{aligned}$$

Dies ist fast ein LP. Einziger Unterschied $x_i \in \{0, 1\}$ statt $x_i \in [0, 1]$ gefordert.

Rucksackproblem:

$$\begin{aligned} \text{maximiere} \quad & p_1 x_1 + \cdots + p_n x_n \\ \text{sodass} \quad & w_1 x_1 + \cdots + w_n x_n \leq t, \\ & \forall i : x_i \in \{0, 1\}. \end{aligned}$$

Dies ist fast ein LP. Einziger Unterschied $x_i \in \{0, 1\}$ statt $x_i \in [0, 1]$ gefordert.

Ganzzahliges LP (ILP): Ein **ganzzahliges LP** ist ein LP, bei dem für (einige) Variablen gefordert wird, dass sie nur ganzzahlige Werte annehmen dürfen.

Rucksackproblem:

$$\begin{aligned} \text{maximiere} \quad & p_1 x_1 + \cdots + p_n x_n \\ \text{sodass} \quad & w_1 x_1 + \cdots + w_n x_n \leq t, \\ & \forall i : x_i \in \{0, 1\}. \end{aligned}$$

Dies ist fast ein LP. Einziger Unterschied $x_i \in \{0, 1\}$ statt $x_i \in [0, 1]$ gefordert.

Ganzzahliges LP (ILP): Ein **ganzzahliges LP** ist ein LP, bei dem für (einige) Variablen gefordert wird, dass sie nur ganzzahlige Werte annehmen dürfen.

Viele NP-schwere Probleme können als ganzzahlige LPs dargestellt werden.

Vertex-Cover-Problem (VC)

Eingabe: ungerichteter Graph $G = (V, E)$ mit $V = \{1, \ldots, n\}$

Aufgabe: Finde kleinste Menge $V' \subseteq V$, sodass jede Kante aus E zu mindestens einem Knoten aus V' inzident ist?

Formulierung als ILP:

Variablen: Für $i \in V$ gibt Variable $x_i \in \{0, 1\}$ an, ob i in der Auswahl V' enthalten ist.

Vertex-Cover-Problem (VC)

Eingabe: ungerichteter Graph $G = (V, E)$ mit $V = \{1, \ldots, n\}$

Aufgabe: Finde kleinste Menge $V' \subseteq V$, sodass jede Kante aus E zu mindestens einem Knoten aus V' inzident ist?

Formulierung als ILP:

Variablen: Für $i \in V$ gibt Variable $x_i \in \{0, 1\}$ an, ob i in der Auswahl V' enthalten ist.

$$\begin{aligned} \text{minimiere} \quad & x_1 + \cdots + x_n \\ \text{sodass} \quad & \forall e = (i, j) \in E : x_i + x_j \geq 1, \\ & \forall i : x_i \in \{0, 1\}. \end{aligned}$$

Scheduling auf identischen Maschinen

Eingabe: Menge $J = \{1, \ldots, n\}$ von **Jobs**, **Jobgrößen** $p_1, \ldots, p_n \in \mathbb{R}_{>0}$
Menge $M = \{1, \ldots, m\}$ von **Maschinen**

Aufgabe: Finde **Schedule** $\pi : J \to M$ mit minimalem Makespan.

Scheduling auf identischen Maschinen

Eingabe: Menge $J = \{1, \ldots, n\}$ von **Jobs**, **Jobgrößen** $p_1, \ldots, p_n \in \mathbb{R}_{>0}$
Menge $M = \{1, \ldots, m\}$ von **Maschinen**

Aufgabe: Finde **Schedule** $\pi : J \to M$ mit minimalem Makespan.

Formulierung als ILP:

Variablen: Reellwertige Variable t, die den Makespan codiert.

Für $i \in M$ und $j \in J$ Variable x_{ij} die binär codiert, ob Job j Maschine i zugewiesen wird.

Scheduling auf identischen Maschinen

Eingabe: Menge $J = \{1, \ldots, n\}$ von **Jobs**, **Jobgrößen** $p_1, \ldots, p_n \in \mathbb{R}_{>0}$
Menge $M = \{1, \ldots, m\}$ von **Maschinen**

Aufgabe: Finde **Schedule** $\pi : J \to M$ mit minimalem Makespan.

Formulierung als ILP:

Variablen: Reellwertige Variable t, die den Makespan codiert.

Für $i \in M$ und $j \in J$ Variable x_{ij} die binär codiert, ob Job j Maschine i zugewiesen wird.

$$\begin{aligned} \text{minimiere} \quad & t \\ \text{sodass} \quad & \forall i \in \{1, \ldots, m\} : \sum_{j=1}^{n} p_j x_{ij} \leq t, \\ & \forall j \in \{1, \ldots, n\} : \sum_{i=1}^{m} x_{ij} = 1, \\ & \forall i, j : x_{ij} \in \{0, 1\}. \end{aligned}$$

Traveling Salesman Problem (TSP)

Eingabe: $V = \{1, \ldots, n\}$ mit symmetrischen Distanzen d_{ij} für $i \in V$ und $j \in V$

Traveling Salesman Problem (TSP)

Eingabe: $V = \{1, \ldots, n\}$ mit symmetrischen Distanzen d_{ij} für $i \in V$ und $j \in V$

Formulierung als ILP:

Variablen: Für $i, j \in V$ mit $i \neq j$ codiert x_{ij} binär, ob die Tour die Kante (i, j) enthält.

Traveling Salesman Problem (TSP)

Eingabe: $V = \{1, \ldots, n\}$ mit symmetrischen Distanzen d_{ij} für $i \in V$ und $j \in V$

Formulierung als ILP:

Variablen: Für $i, j \in V$ mit $i \neq j$ codiert x_{ij} binär, ob die Tour die Kante (i, j) enthält.

$$\text{minimiere} \quad \sum_{i=1}^{n} \sum_{j \neq i, j=1}^{n} d_{ij} x_{ij}$$

$$\text{sodass} \quad \forall j \in \{1, \ldots, n\} : \sum_{i \neq j, i=1}^{n} x_{ij} = 1$$

$$\forall j \in \{1, \ldots, n\} : \sum_{i \neq j, i=1}^{n} x_{ji} = 1$$

$$\forall i, j : x_{ij} \in \{0, 1\}$$

Traveling Salesman Problem (TSP)

Eingabe: $V = \{1, \ldots, n\}$ mit symmetrischen Distanzen d_{ij} für $i \in V$ und $j \in V$

Formulierung als ILP:

Variablen: Für $i, j \in V$ mit $i \neq j$ codiert x_{ij} binär, ob die Tour die Kante (i, j) enthält.

$$\text{minimiere} \quad \sum_{i=1}^{n} \sum_{j \neq i, j=1}^{n} d_{ij} x_{ij}$$

$$\text{sodass} \quad \forall j \in \{1, \ldots, n\} : \sum_{i \neq j, i=1}^{n} x_{ij} = 1$$

$$\forall j \in \{1, \ldots, n\} : \sum_{i \neq j, i=1}^{n} x_{ji} = 1$$

$$\forall i, j : x_{ij} \in \{0, 1\}$$

Problem: Das LP stellt nicht sicher, dass die Lösung ein Kreis ist. Es können auch mehrere Kreise sein, die insgesamt alle Knoten abdecken.

Wir benötigen weitere Nebenbedingungen.

zusätzliche Variablen: Für $i \in V$ mit $i \neq 1$ codiere $u_i \in \{1, \ldots, n-1\}$, an welcher Stelle der Knoten i in der Tour vorkommt. Knoten 1 komme an Stelle 0.

Wir benötigen weitere Nebenbedingungen.

zusätzliche Variablen: Für $i \in V$ mit $i \neq 1$ codiere $u_i \in \{1, \ldots, n-1\}$, an welcher Stelle der Knoten i in der Tour vorkommt. Knoten 1 komme an Stelle 0.

Wir möchten codieren, dass u_j größer als u_i sein muss, wenn die Kante (i, j) in der Tour enthalten ist. Dies erreichen wir durch die folgenden Nebenbedingungen:

$$\forall i, j \in \{2, \ldots, n\}, i \neq j : u_i - u_j + n x_{ij} \leq n - 1, \tag{1}$$
$$\forall i \in \{2, \ldots, n\} : u_i \in \{1, \ldots, n-1\}.$$

Wir benötigen weitere Nebenbedingungen.

zusätzliche Variablen: Für $i \in V$ mit $i \neq 1$ codiere $u_i \in \{1, \ldots, n-1\}$, an welcher Stelle der Knoten i in der Tour vorkommt. Knoten 1 komme an Stelle 0.

Wir möchten codieren, dass u_j größer als u_i sein muss, wenn die Kante (i, j) in der Tour enthalten ist. Dies erreichen wir durch die folgenden Nebenbedingungen:

$$\begin{aligned} &\forall i, j \in \{2, \ldots, n\}, i \neq j : u_i - u_j + n x_{ij} \leq n - 1, \\ &\forall i \in \{2, \ldots, n\} : u_i \in \{1, \ldots, n-1\}. \end{aligned} \tag{1}$$

Gilt $x_{ij} = 0$, so entspricht (1) der Bedingung $u_i - u_j \leq n - 1$, die für jede Belegung der Variablen u_i und u_j aus $\{1, \ldots, n-1\}$ automatisch erfüllt ist.

Gilt $x_{ij} = 1$, so entspricht (1) der gewünschten Bedingung $u_i \leq u_j - 1$.

$$\forall i, j \in \{2, \ldots, n\}, i \neq j : u_i - u_j + n x_{ij} \leq n - 1, \qquad (1)$$
$$\forall i \in \{2, \ldots, n\} : u_i \in \{1, \ldots, n-1\}.$$

- **Aus Tour ergibt sich Lösung für das ILP:** Codiere in den x_{ij}-Variablen die Kanten der Tour und in den u_i-Variablen für jeden Knoten, an welcher Stelle er in der Tour vorkommt.
  Wichtig: Die Bedingung (1) muss nur für $i \neq 1$ und $j \neq 1$ gelten muss.

$$\forall i, j \in \{2, \ldots, n\}, i \neq j : u_i - u_j + n x_{ij} \leq n - 1, \qquad (1)$$
$$\forall i \in \{2, \ldots, n\} : u_i \in \{1, \ldots, n-1\}.$$

- **Aus Tour ergibt sich Lösung für das ILP:** Codiere in den x_{ij}-Variablen die Kanten der Tour und in den u_i-Variablen für jeden Knoten, an welcher Stelle er in der Tour vorkommt.
  Wichtig: Die Bedingung (1) muss nur für $i \neq 1$ und $j \neq 1$ gelten muss.
- **Aus Lösung des ILPs ergibt sich Tour:** Erlaubte Lösungen ohne (1): Menge disjunkter Kreise, die gemeinsam alle Knoten abdecken.
  (1) stellt sicher, dass ein Kreis nur dann erlaubt ist, wenn er den Knoten 1 enthält.

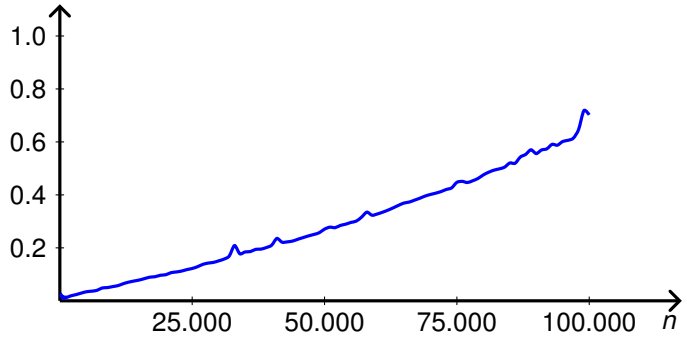
ILP-Solver: Softwarepaket zur Lösung (ganzzahliger) linearer Programme.

Beispiele: Gurobi[1], CPLEX[2], SCIP[3]

[1] https://www.gurobi.com/
[2] https://www.ibm.com/de-de/analytics/cplex-optimizer
[3] https://www.scipopt.org/

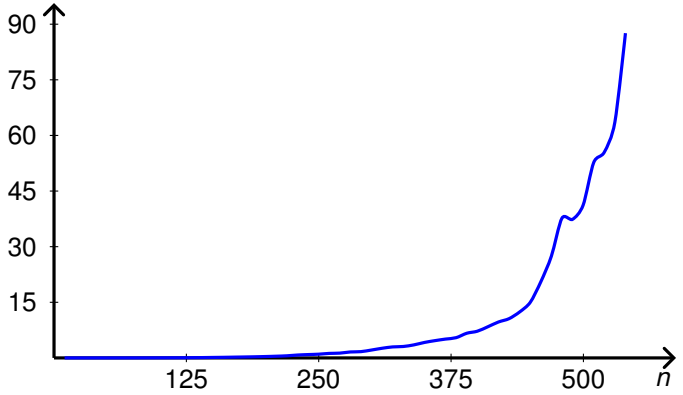
Laufzeit von Gurobi zur Lösung des Rucksackproblems: Eingaben mit n Objekten, jeder Nutzen p_i und jedes Gewicht w_i uniform zufällig aus $[0, 1]$, Kapazität $n/4$

y-Achse zeigt die durchschnittliche Laufzeit in Sekunden über jeweils 100 Durchläufe.



Laufzeit von Gurobi zur Lösung von Clique: Eingaben mit n Knoten, jede der $\binom{n}{2}$ vielen möglichen Kanten ist mit Wahrscheinlichkeit $1/2$ enthalten

y-Achse zeigt die durchschnittliche Laufzeit in Sekunden über jeweils 100 Durchläufe.



Laufzeit von Gurobi zur Lösung des TSP:

Eingaben mit n Knoten, jedes Knotenpaar erhält uniform zufälligen Abstand aus $[0, 1]$

Experimente bis $n = 80$, durchschnittliche Laufzeiten nicht aussagekräftig, da **sehr hohe Varianz**.

Für $n = 80$ konnten die meisten Instanzen in weniger als 10 Sekunden optimal gelöst werden. Bei anderen Instanzen haben wir die Berechnung dafür **nach über 4 Stunden ohne Ergebnis abgebrochen**.