Ganzzahlige Optimierung

Minikurs im Cluster Intelligent Systems for Decision Support

Winfried Hochstättler

Diskrete Mathematik und Optimierung
FernUniversität in Hagen

16. Juni 2009

Outline I

Outline II

Outline

Ganzzahlige Optimierungsaufgabe

Seien $A \in \mathbb{Z}^{m \times n}, G \in \mathbb{Z}^{m \times r}, b \in \mathbb{Z}^m, c \in \mathbb{Z}^n, h \in \mathbb{Z}^r$. Ein

Ganzzahlige Optimierungsaufgabe

Seien $A \in \mathbb{Z}^{m \times n}, G \in \mathbb{Z}^{m \times r}, b \in \mathbb{Z}^m, c \in \mathbb{Z}^n, h \in \mathbb{Z}^r$. Ein gemischt ganzzahliges Programm in Standardform hat die Gestalt

$$
(MIP) \qquad \begin{array}{rrrrrrr} \max & c^\top x & + & h^\top y & & & \\ & Ax & + & Gy & & = & b \\ & x & \in & \mathbb{Z}^n_+ & , y & \geq & 0, \end{array}
$$

Ganzzahlige Optimierungsaufgabe

Seien $A \in \mathbb{Z}^{m\times n}, G \in \mathbb{Z}^{m\times r}, b \in \mathbb{Z}^m, c \in \mathbb{Z}^n, h \in \mathbb{Z}^r$. Ein gemischt ganzzahliges Programm in Standardform hat die Gestalt

$$(MIP) \qquad \begin{array}{lrcrcrcl} \max & c^\top x & + & h^\top y & & & \\ & Ax & + & Gy & & = & b \\ & x & \in & \mathbb{Z}^n_+ & , y & \geq & 0, \end{array}$$

ganzzahliges Programm

$$(IP) \qquad \begin{array}{lrcl} \max & c^\top x & & \\ & Ax & = & b \\ & x & \in & \mathbb{Z}^n_+, \end{array}$$

Ganzzahlige Optimierungsaufgabe

Seien $A \in \mathbb{Z}^{m\times n}, G \in \mathbb{Z}^{m\times r}, b \in \mathbb{Z}^m, c \in \mathbb{Z}^n, h \in \mathbb{Z}^r$. Ein

gemischt ganzzahliges Programm in Standardform hat die Gestalt

$$(MIP) \qquad \begin{array}{rcrcrcl} \max & c^\top x & + & h^\top y & & & \\ & Ax & + & Gy & & = & b \\ & x & \in & \mathbb{Z}^n_+ & , y & \geq & 0, \end{array}$$

ganzzahliges Programm

$$(IP) \qquad \begin{array}{rcrcl} \max & c^\top x & & & \\ & Ax & = & b & \\ & x & \in & \mathbb{Z}^n_+, & \end{array}$$

boolsches Programm

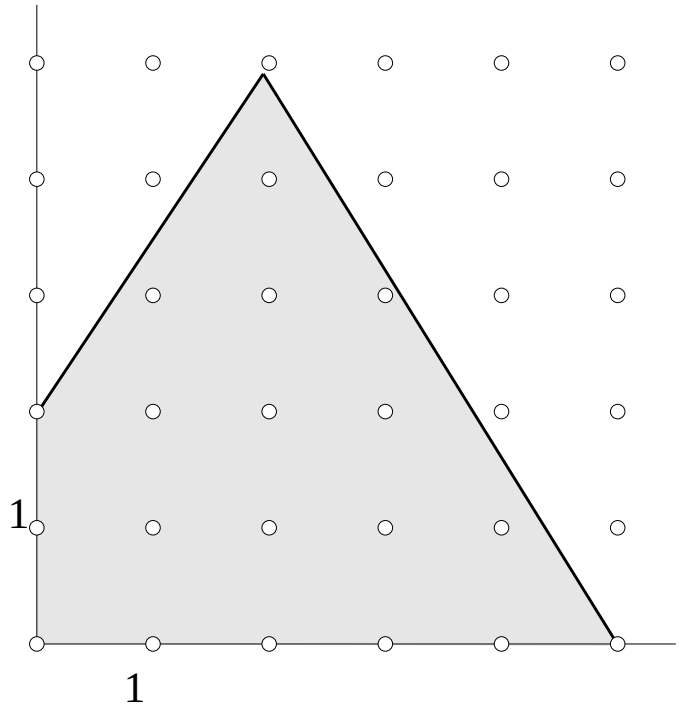
$$(BIP) \qquad \begin{array}{rcrcl} \max & c^\top x & & & \\ & Ax & = & b & \\ & x & \in & \{0,1\}^n. & \end{array}$$

Ergebnisse des Linearen Programms runden?

$$
\begin{array}{lrcll}
\max & 100x_1 & +64x_2 & & \\
 & 50x_1 & +31x_2 & \leq & 250 \\
 & -3x_1 & +2x_2 & \leq & 4 \\
 & & x_1, x_2 & \in & \mathbb{Z}_+.
\end{array}
$$

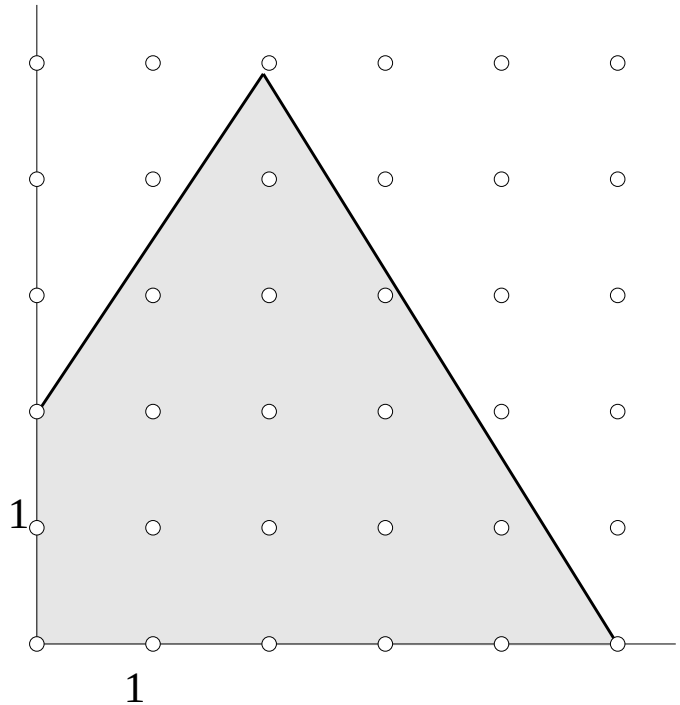
Ergebnisse des Linearen Programms runden?

$$
\begin{array}{lrrcl}
\max & 100x_1 & +64x_2 & & \\
 & 50x_1 & +31x_2 & \leq & 250 \\
 & -3x_1 & +2x_2 & \leq & 4 \\
 & & x_1, x_2 & \in & \mathbb{Z}_+.
\end{array}
$$



Ergebnisse des Linearen Programms runden?

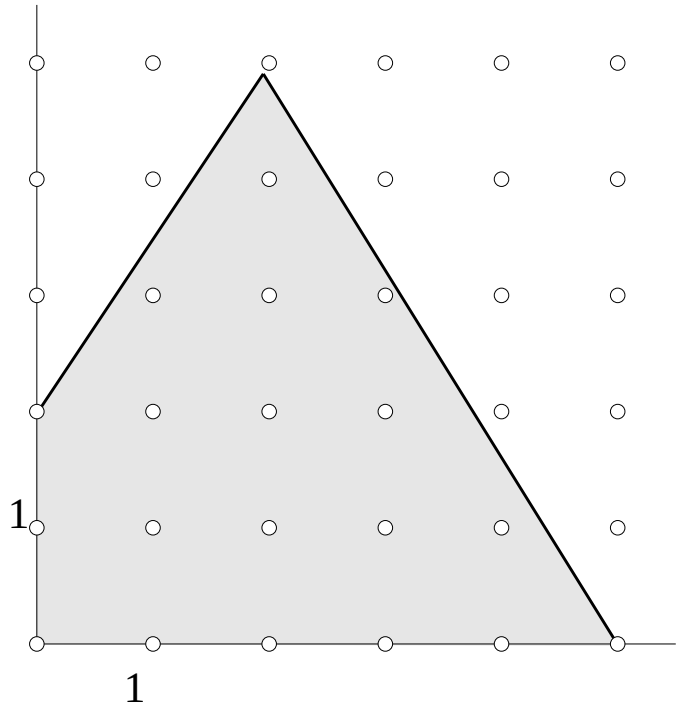
$$
\begin{array}{lrrcl}
\max & 100x_1 & +64x_2 & & \\
 & 50x_1 & +31x_2 & \leq & 250 \\
 & -3x_1 & +2x_2 & \leq & 4 \\
 & & x_1, x_2 & \in & \mathbb{Z}_+.
\end{array}
$$



Optimallösung ohne Ganzzahligkeitsbedingung $(\frac{376}{193}, \frac{950}{193})$.

Ergebnisse des Linearen Programms runden?

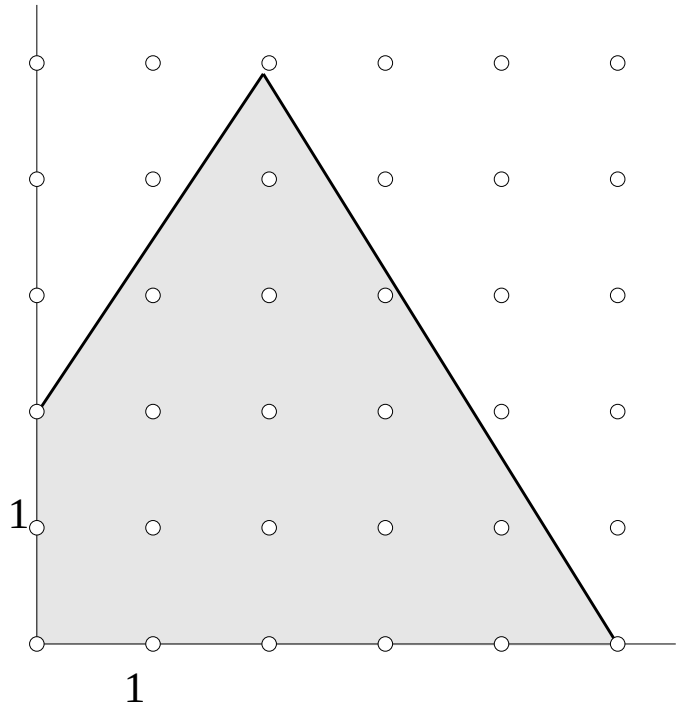
$$\begin{array}{lrrcl} \max & 100x_1 & +64x_2 & & \\ & 50x_1 & +31x_2 & \leq & 250 \\ & -3x_1 & +2x_2 & \leq & 4 \\ & & x_1, x_2 & \in & \mathbb{Z}_+. \end{array}$$



Optimallösung ohne Ganzzahligkeitsbedingung $(\frac{376}{193}, \frac{950}{193})$.
Optimallösung des IP ist $(5, 0)$ mit ZFW 500.

Ergebnisse des Linearen Programms runden?

$$\begin{array}{lrrcl} \max & 100x_1 & +64x_2 & & \\ & 50x_1 & +31x_2 & \leq & 250 \\ & -3x_1 & +2x_2 & \leq & 4 \\ & & x_1, x_2 & \in & \mathbb{Z}_+. \end{array}$$

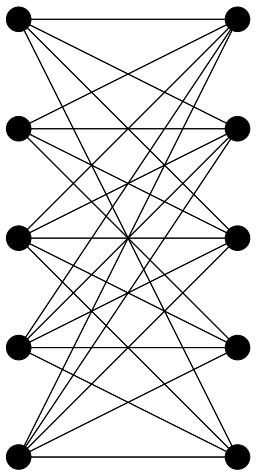


Optimallösung ohne Ganzzahligkeitsbedingung $(\frac{376}{193}, \frac{950}{193})$.
Optimallösung des IP ist $(5, 0)$ mit ZFW 500.
Mittels Runden $(2, 4)$ mit ZFW 456, 10% unter Optimalwert.

Outline

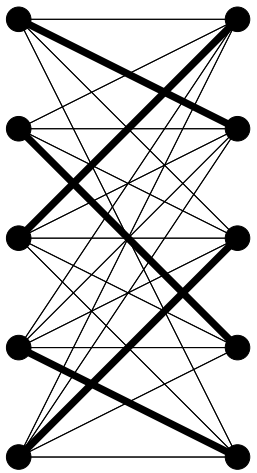
Das Zuweisungsproblem – Assignment Problem

Gegeben n Arbeitnehmer und n Arbeitsplätze. Besetzung von Arbeitsplatz j durch Arbeitnehmer i bringt Nutzen $c_{ij} \geq 0$. Maximiere den Gesamtnutzen.



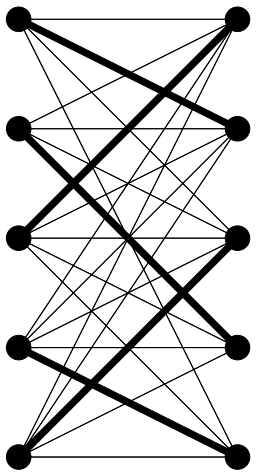
Das Zuweisungsproblem – Assignment Problem

Gegeben n Arbeitnehmer und n Arbeitsplätze. Besetzung von Arbeitsplatz j durch Arbeitnehmer i bringt Nutzen $c_{ij} \geq 0$. Maximiere den Gesamtnutzen.



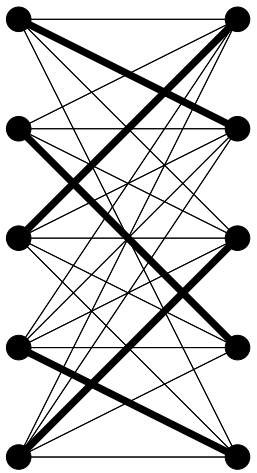
Das Zuweisungsproblem – Assignment Problem

Gegeben n Arbeitnehmer und n Arbeitsplätze. Besetzung von Arbeitsplatz j durch Arbeitnehmer i bringt Nutzen $c_{ij} \geq 0$. Maximiere den Gesamtnutzen. Setze $x_{ij} = 1$ falls i Platz j zugeordnet wird, $x_{ij} = 0$ sonst:



Das Zuweisungsproblem – Assignment Problem

Gegeben n Arbeitnehmer und n Arbeitsplätze. Besetzung von Arbeitsplatz j durch Arbeitnehmer i bringt Nutzen $c_{ij} \geq 0$. Maximiere den Gesamtnutzen. Setze $x_{ij} = 1$ falls i Platz j zugeordnet wird, $x_{ij} = 0$ sonst:

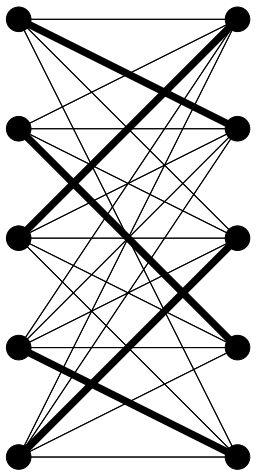


$$\max \quad \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$$

$$x_{ij} \quad \in \quad \{0, 1\}$$

Das Zuweisungsproblem – Assignment Problem

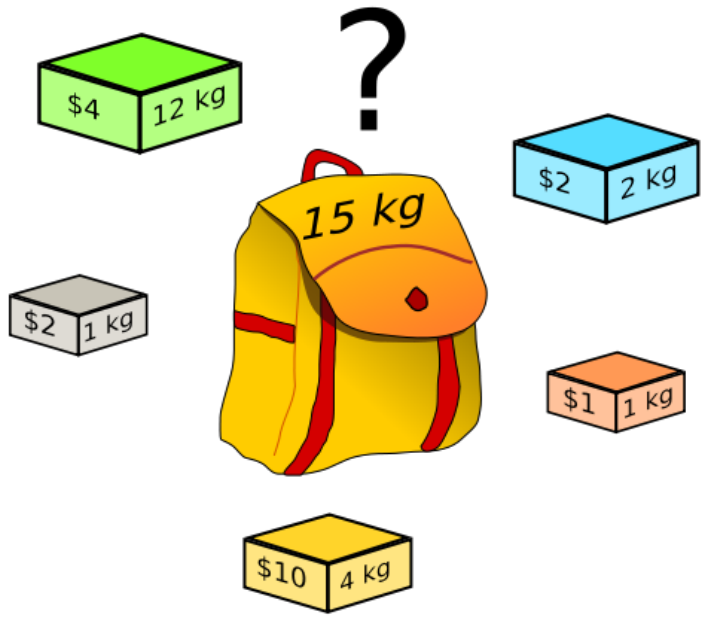
Gegeben n Arbeitnehmer und n Arbeitsplätze. Besetzung von Arbeitsplatz j durch Arbeitnehmer i bringt Nutzen $c_{ij} \geq 0$. Maximiere den Gesamtnutzen. Setze $x_{ij} = 1$ falls i Platz j zugeordnet wird, $x_{ij} = 0$ sonst:



$$
\begin{array}{lrcll}
\max & \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} & & & \\
(AP) & \sum_{j=1}^{n} x_{ij} & = & 1 & \forall i = 1, \ldots, n \\
& \sum_{i=1}^{n} x_{ij} & = & 1 & \forall j = 1, \ldots, n \\
& x_{ij} & \in & \{0, 1\} &
\end{array}
$$

Das Rucksack Problem (Knapsack Problem)

Gegeben sei ein Rucksack mit Kapazität $b \in \mathbb{N}$ und n Gegenstände. Gegenstand i hat Gewicht (Volumen) a_i und bringt Nutzen c_i. Packe Rucksack so, dass der Nutzen maximiert wird.



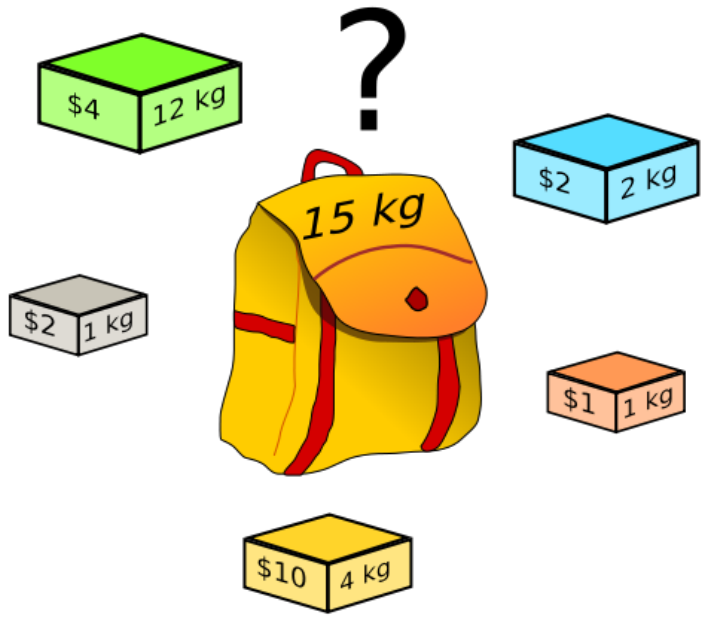
[a] http://en.wikipedia.org/wiki/Knapsack_problem

Das Rucksack Problem (Knapsack Problem)

Gegeben sei ein Rucksack mit Kapazität $b \in \mathbb{N}$ und n Gegenstände. Gegenstand i hat Gewicht (Volumen) a_i und bringt Nutzen c_i. Packe Rucksack so, dass der Nutzen maximiert wird.

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

$$x_j \quad \in \quad \{0, 1\}$$



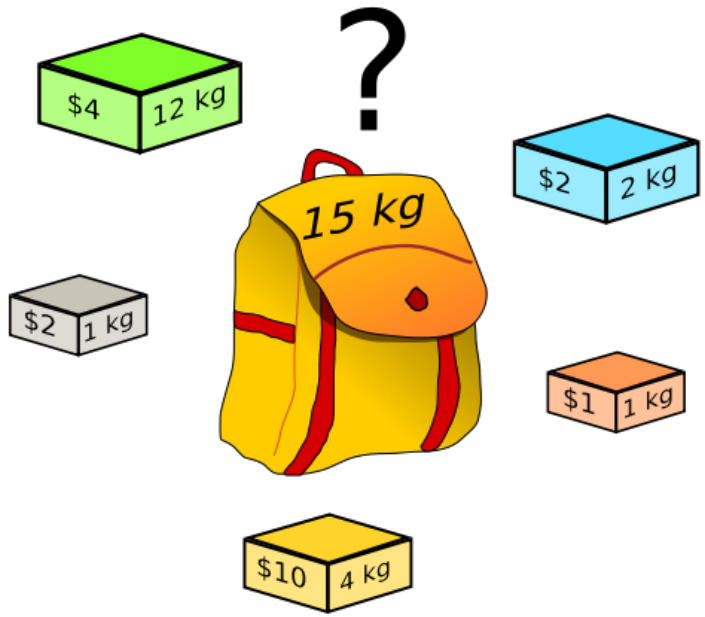
[a]

[a] http://en.wikipedia.org/wiki/Knapsack_problem

Das Rucksack Problem (Knapsack Problem)

Gegeben sei ein Rucksack mit Kapazität $b \in \mathbb{N}$ und n Gegenstände. Gegenstand i hat Gewicht (Volumen) a_i und bringt Nutzen c_i. Packe Rucksack so, dass der Nutzen maximiert wird.

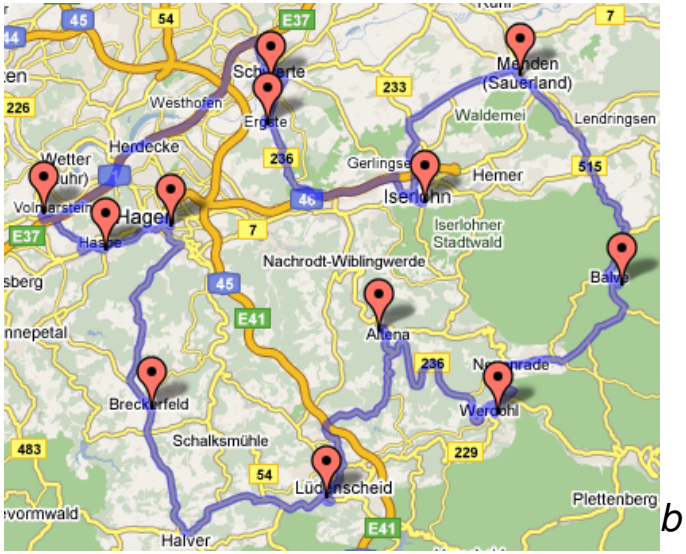
$$
\begin{array}{lrcl}
\max & \sum_{j=1}^{n} c_j x_j & & \\
(KP) & \sum_{j=1}^{n} a_j x_j & \leq & b \\
 & x_j & \in & \{0,1\}
\end{array}
$$



[a] http://en.wikipedia.org/wiki/Knapsack_problem

Rundreiseproblem (Travelling Salesman Problem TSP)

Man will n Städte in Rundreise besuchen und zum Ausgangspunkt zurückkehren. Reisezeit von Stadt i zu Stadt j ist c_{ij}. Minimiere Gesamtreisezeit.

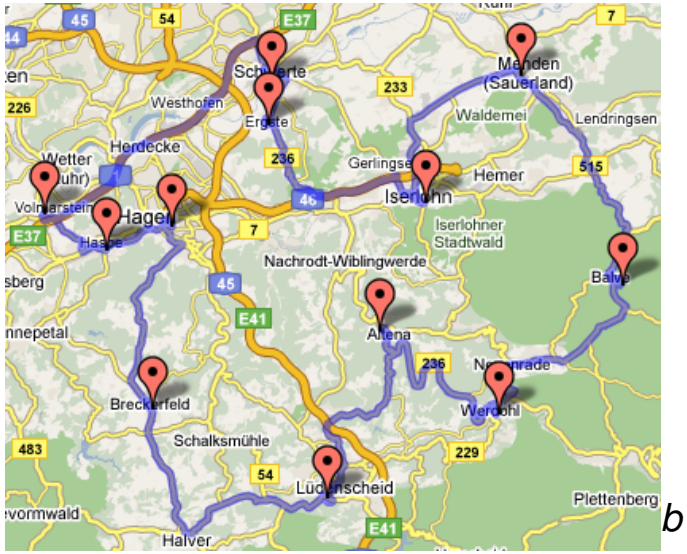


[b]

[b] ©TeleData PPWK 2009

Rundreiseproblem (Travelling Salesman Problem TSP)

Man will n Städte in Rundreise besuchen und zum Ausgangspunkt zurückkehren. Reisezeit von Stadt i zu Stadt j ist c_{ij}. Minimiere Gesamtreisezeit. Setze $x_{ij} = 1$, wenn Stadt i unmittelbar vor Stadt j besucht wird.



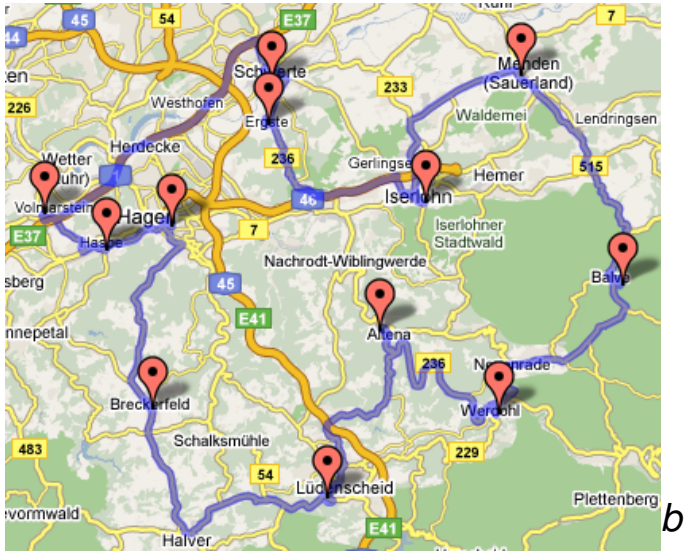
b

b©TeleData PPWK 2009

Rundreiseproblem (Travelling Salesman Problem TSP)

Man will n Städte in Rundreise besuchen und zum Ausgangspunkt zurückkehren. Reisezeit von Stadt i zu Stadt j ist c_{ij}. Minimiere Gesamtreisezeit. Setze $x_{ij} = 1$, wenn Stadt i unmittelbar vor Stadt j besucht wird.

$$\min \quad \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$$

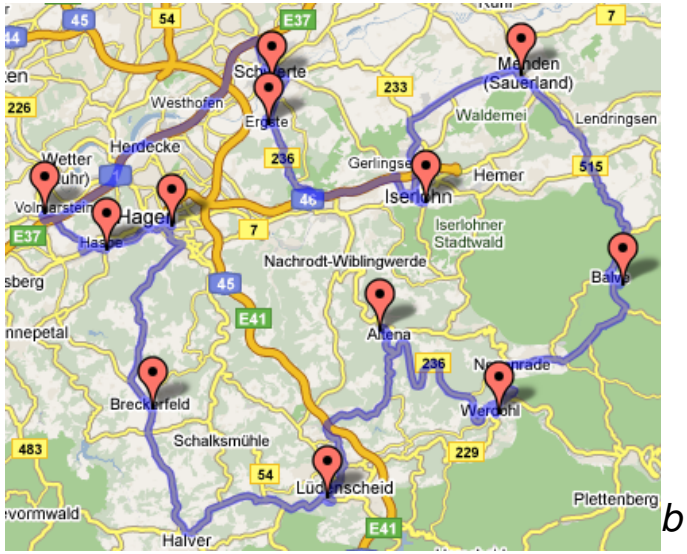


[b] ©TeleData PPWK 2009

$$x_{ij} \quad \in \quad \{0, 1\}$$

Rundreiseproblem (Travelling Salesman Problem TSP)

Man will n Städte in Rundreise besuchen und zum Ausgangspunkt zurückkehren. Reisezeit von Stadt i zu Stadt j ist c_{ij}. Minimiere Gesamtreisezeit. Setze $x_{ij} = 1$, wenn Stadt i unmittelbar vor Stadt j besucht wird.



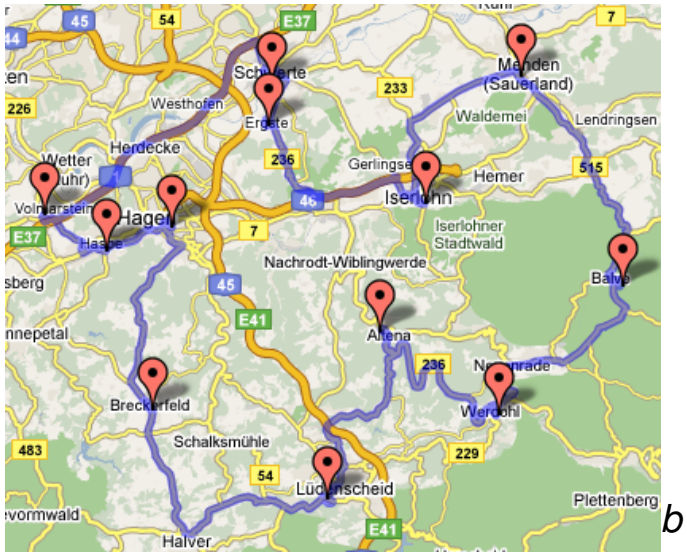
[b]

[b] ©TeleData PPWK 2009

$$
\begin{array}{rrcll}
\min & \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} & & & \\
(TSP) & \sum_{j=1}^{n} x_{ij} & = & 1 & \forall i = 1, \ldots, n \\
& \sum_{i=1}^{n} x_{ij} & = & 1 & \forall j = 1, \ldots, n \\
& & & & \\
& x_{ij} & \in & \{0, 1\} &
\end{array}
$$

Rundreiseproblem (Travelling Salesman Problem TSP)

Man will n Städte in Rundreise besuchen und zum Ausgangspunkt zurückkehren. Reisezeit von Stadt i zu Stadt j ist c_{ij}. Minimiere Gesamtreisezeit. Setze $x_{ij} = 1$, wenn Stadt i unmittelbar vor Stadt j besucht wird.



[b]

[b] ©TeleData PPWK 2009

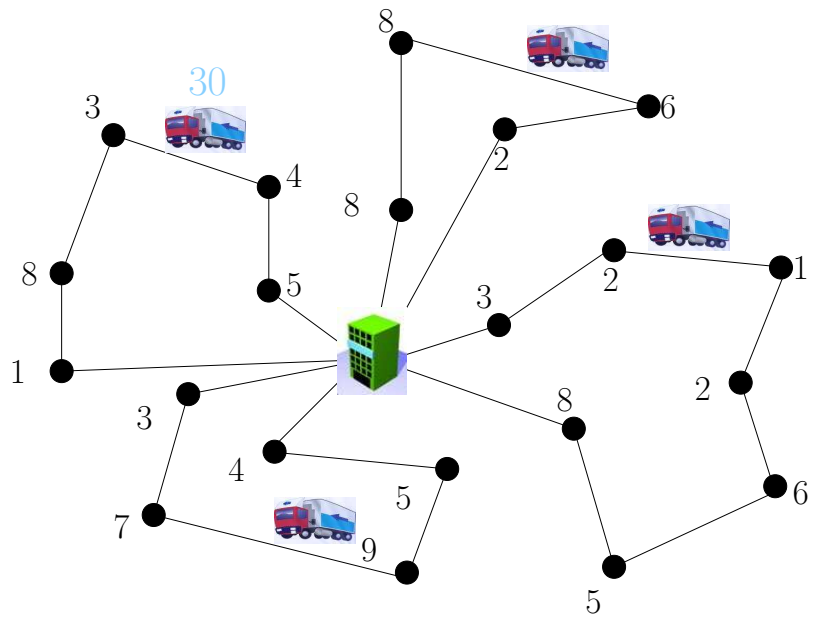
$$
\begin{array}{rrcll}
\min & \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} & & & \\
(TSP) & \sum_{j=1}^{n} x_{ij} & = & 1 & \forall i = 1,\ldots,n \\
& \sum_{i=1}^{n} x_{ij} & = & 1 & \forall j = 1,\ldots,n \\
& \sum_{i\in S}\sum_{j\notin S} x_{ij} & \geq & 1 & \forall \emptyset \neq S \subset \{1,\ldots,n\} \\
& x_{ij} & \in & \{0,1\} &
\end{array}
$$

Kapazitierte Tourenplanung CVRP

V_0 Menge von Knoten in gewichtetem Graph (Kunden) mit Bestellmengen $b_v \in \mathbb{N}$. $v_0 \notin V_0$ ein Depot. $C \in \mathbb{N}$ Kapazität der $k \in \mathbb{N}$ LKW. Seien c_e Kosten des Transports entlang Kante e. Beliefere Kunden kostenminimal.

Kapazitierte Tourenplanung CVRP

V_0 Menge von Knoten in gewichtetem Graph (Kunden) mit Bestellmengen $b_v \in \mathbb{N}$. $v_0 \notin V_0$ ein Depot. $C \in \mathbb{N}$ Kapazität der $k \in \mathbb{N}$ LKW. Seien c_e Kosten des Transports entlang Kante e. Beliefere Kunden kostenminimal.



Kapazitierte Tourenplanung CVRP

V_0 Menge von Knoten in gewichtetem Graph (Kunden) mit Bestellmengen $b_v \in \mathbb{N}$. $v_0 \notin V_0$ ein Depot. $C \in \mathbb{N}$ Kapazität der $k \in \mathbb{N}$ LKW. Seien c_e Kosten des Transports entlang Kante e. Beliefere Kunden kostenminimal.

$$\min \quad \sum_{e \in E} c_e x_e$$

Kapazitierte Tourenplanung CVRP

V_0 Menge von Knoten in gewichtetem Graph (Kunden) mit Bestellmengen $b_v \in \mathbb{N}$. $v_0 \notin V_0$ ein Depot. $C \in \mathbb{N}$ Kapazität der $k \in \mathbb{N}$ LKW. Seien c_e Kosten des Transports entlang Kante e. Beliefere Kunden kostenminimal.

$$\begin{array}{lll} \min & \sum_{e \in E} c_e x_e & \\ & \sum_{v \in e} x_e = 2 & \forall v \in V_0 \\ & \sum_{v_0 \in e} x_e = 2k & \end{array}$$

Kapazitierte Tourenplanung CVRP

V_0 Menge von Knoten in gewichtetem Graph (Kunden) mit Bestellmengen $b_v \in \mathbb{N}$. $v_0 \notin V_0$ ein Depot. $C \in \mathbb{N}$ Kapazität der $k \in \mathbb{N}$ LKW. Seien c_e Kosten des Transports entlang Kante e. Beliefere Kunden kostenminimal.

$$
\begin{array}{lllll}
\min & \sum_{e \in E} c_e x_e & & & \\
 & \sum_{v \in e} x_e & = & 2 & \forall v \in V_0 \\
 & \sum_{v_0 \in e} x_e & = & 2k & \\
 & \sum_{\substack{e=(u,v) \\ u \in S \not\ni v}} x_e & \geq & 2 \frac{\sum_{v \in S} b_v}{C} & \forall S \subseteq V_0
\end{array}
$$

Kapazitierte Tourenplanung CVRP

V_0 Menge von Knoten in gewichtetem Graph (Kunden) mit Bestellmengen $b_v \in \mathbb{N}$. $v_0 \notin V_0$ ein Depot. $C \in \mathbb{N}$ Kapazität der $k \in \mathbb{N}$ LKW. Seien c_e Kosten des Transports entlang Kante e. Beliefere Kunden kostenminimal.

$$
\begin{array}{lllll}
\min & \sum_{e \in E} c_e x_e & & & \\
 & \sum_{v \in e} x_e & = & 2 & \forall v \in V_0 \\
 & \sum_{v_0 \in e} x_e & = & 2k & \\
 & \sum_{\substack{e=(u,v) \\ u \in S \not\ni v}} x_e & \geq & 2 \frac{\sum_{v \in S} b_v}{C} & \forall S \subseteq V_0 \\
 & x_e & \in & \{0,1\} & \forall e \in V_0 \times V_0 \\
 & x_e & \in & \{0,1,2\} & \forall v_0 \in e
\end{array}
$$

Outline

Polyeder

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

Polyeder

Seien $A \in \mathbb{Z}^{m \times n}, b \in \mathbb{Z}^m$. Die Menge $P := \{x \in \mathbb{R}^n \mid Ax \leq b\}$ heißt ein Polyeder.

Polyeder

Seien $A \in \mathbb{Z}^{m \times n}, b \in \mathbb{Z}^m$. Die Menge $P := \{x \in \mathbb{R}^n \mid Ax \leq b\}$ heißt ein Polyeder. Sei $X \subseteq \mathbb{Z}^r \times \mathbb{R}^p$.

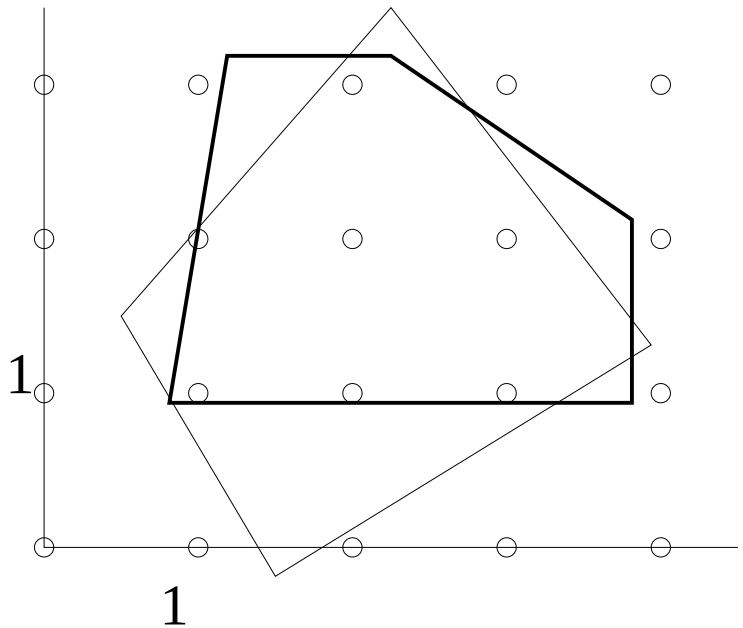
Polyeder

Seien $A \in \mathbb{Z}^{m \times n}, b \in \mathbb{Z}^m$. Die Menge $P := \{x \in \mathbb{R}^n \mid Ax \leq b\}$ heißt ein Polyeder. Sei $X \subseteq \mathbb{Z}^r \times \mathbb{R}^p$. Wir sagen, das Polyeder P modelliert X genau dann, wenn $X = P \cap (\mathbb{Z}^r \times \mathbb{R}^p)$.

Polyeder

Seien $A \in \mathbb{Z}^{m \times n}$, $b \in \mathbb{Z}^m$. Die Menge $P := \{x \in \mathbb{R}^n \mid Ax \leq b\}$ heißt ein Polyeder. Sei $X \subseteq \mathbb{Z}^r \times \mathbb{R}^p$. Wir sagen, das Polyeder P modelliert X genau dann, wenn $X = P \cap (\mathbb{Z}^r \times \mathbb{R}^p)$. Zwei Modellierungen von

$\{(1,1),(2,1),(3,1),(1,2),(2,2),(3,2),(2,3)\}$.



Gute und ideale Formulierungen

Eine Modellierung ist gut, wenn die Ecken des Polyeders möglichst gute Informationen über die gesuchten Lösungen liefern. Im Idealfall sind bereits alle Ecken ganzzahlig.

Gute und ideale Formulierungen

Eine Modellierung ist gut, wenn die Ecken des Polyeders möglichst gute Informationen über die gesuchten Lösungen liefern. Im Idealfall sind bereits alle Ecken ganzzahlig.

Warum betrachten wir Polyeder, wenn wir diskrete (ganzzahlige) Lösungsmengen betrachten?

Gute und ideale Formulierungen

Eine Modellierung ist gut, wenn die Ecken des Polyeders möglichst gute Informationen über die gesuchten Lösungen liefern. Im Idealfall sind bereits alle Ecken ganzzahlig.

Warum betrachten wir Polyeder, wenn wir diskrete (ganzzahlige) Lösungsmengen betrachten?

Seien $x_1, \ldots, x_k \in \mathbb{R}^n$ und $\lambda \in \mathbb{R}^k$. Dann heißt

$$x = \sum_{i=1}^{k} \lambda_i x_i$$

Konvexkombination, wenn $\lambda \geq 0, \sum_{i=1}^{k} \lambda_i = 1$. Die Menge aller Konvexkombinationen von Elementen in X, nennen wir die konvexe Hülle $\mathrm{Conv}(X)$.

Gute und ideale Formulierungen

Eine Modellierung ist gut, wenn die Ecken des Polyeders möglichst gute Informationen über die gesuchten Lösungen liefern. Im Idealfall sind bereits alle Ecken ganzzahlig.

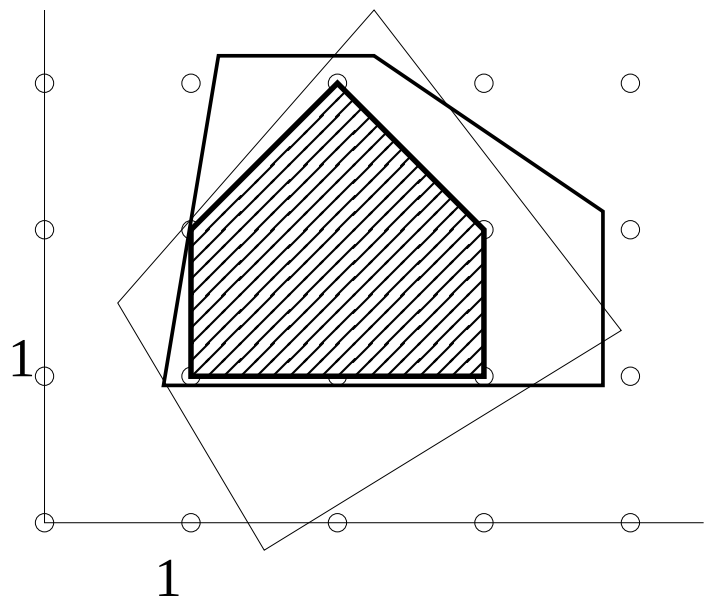
Warum betrachten wir Polyeder, wenn wir diskrete (ganzzahlige) Lösungsmengen betrachten?

Satz (Minkowski): Ist X endlich, so ist $\mathrm{Conv}(X)$ ein Polyeder und alle Ecken von $\mathrm{Conv}(X)$ liegen in X.

Gute und ideale Formulierungen

Eine Modellierung ist gut, wenn die Ecken des Polyeders möglichst gute Informationen über die gesuchten Lösungen liefern. Im Idealfall sind bereits alle Ecken ganzzahlig.

Satz: Die konvexe Hülle der zulässigen Punkte eines IP ist ein Polyeder.



Gute und ideale Formulierungen

Eine Modellierung ist gut, wenn die Ecken des Polyeders möglichst gute Informationen über die gesuchten Lösungen liefern. Im Idealfall sind bereits alle Ecken ganzzahlig.

Satz: Die konvexe Hülle der zulässigen Punkte eines IP ist ein Polyeder.

Wir können also die Optimierungsaufgabe $\max_{x \in X} c^\top x$ äquivalent ersetzen durch $\max_{x \in \mathrm{Conv}(X)} c^\top x$.

Gute und ideale Formulierungen

Eine Modellierung ist gut, wenn die Ecken des Polyeders möglichst gute Informationen über die gesuchten Lösungen liefern. Im Idealfall sind bereits alle Ecken ganzzahlig.

Satz: Die konvexe Hülle der zulässigen Punkte eines IP ist ein Polyeder.

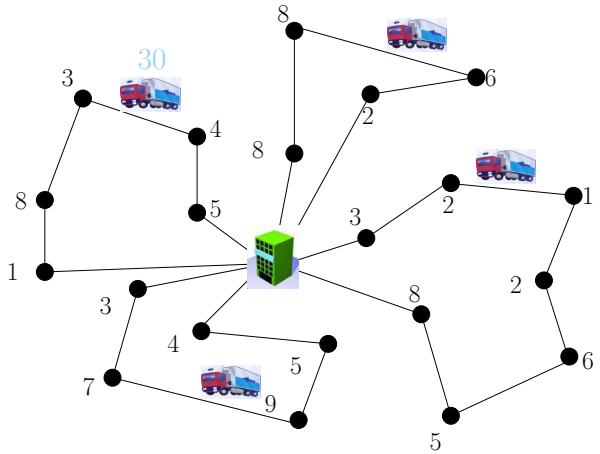
Wir können also die Optimierungsaufgabe $\max_{x \in X} c^\top x$ äquivalent ersetzen durch $\max_{x \in \mathrm{Conv}(X)} c^\top x$. Leider hat oftmals $\mathrm{Conv}(X)$ eine Beschreibung, die exponentiell in der Anzahl der Punkte von X ist oder aus prinzipiellen Gründen nicht in geschlossener Form bekannt sein kann (mehr dazu später).

Gute und ideale Formulierungen II

Seien P_1, P_2 zwei Modellierungen von X. Dann sagen wir P_1 ist besser als P_2, wenn $P_1 \subset P_2$.

Gute und ideale Formulierungen II

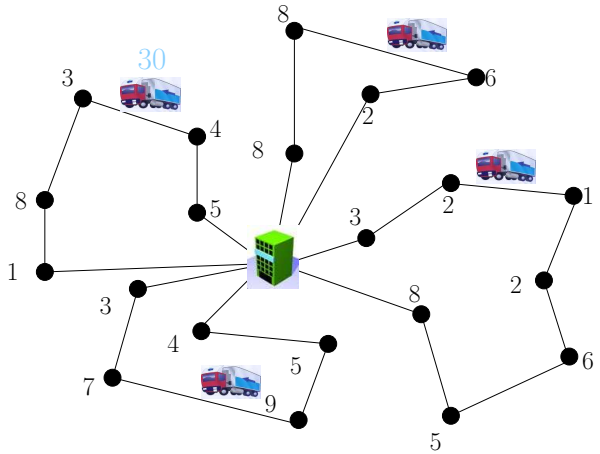
Seien P_1, P_2 zwei Modellierungen von X. Dann sagen wir P_1 ist besser als P_2, wenn $P_1 \subset P_2$.



$$
\begin{array}{lllll}
\min & \sum_{e \in E} c_e x_e & & & \\
& \sum_{v \in e} x_e & = & 2 & \forall v \in V_0 \\
& \sum_{v_0 \in e} x_e & = & 2k & \\
& \sum_{\substack{e=(u,v) \\ u \in S \not\ni v}} x_e & \geq & 2 \frac{\sum_{v \in S} b_v}{C} & \forall S \subseteq V_0 \\
& x_e & \in & \{0,1\} & \forall e \in V_0 \times V_0 \\
& x_e & \in & \{0,1,2\} & \forall v_0 \in e
\end{array}
$$

Gute und ideale Formulierungen II

Seien P_1, P_2 zwei Modellierungen von X. Dann sagen wir P_1 ist besser als P_2, wenn $P_1 \subset P_2$.

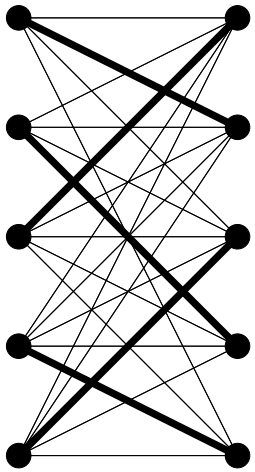


$$
\begin{array}{lllll}
\min & \sum_{e \in E} c_e x_e & & & \\
& \sum_{v \in e} x_e & = & 2 & \forall v \in V_0 \\
& \sum_{v_0 \in e} x_e & = & 2k & \\
& \sum_{\substack{e=(u,v) \\ u \in S \not\ni v}} x_e & \geq & 2 \left\lceil \frac{\sum_{v \in S} b_v}{C} \right\rceil & \forall S \subseteq V_0 \\
& x_e & \in & \{0,1\} & \forall e \in V_0 \times V_0 \\
& x_e & \in & \{0,1,2\} & \forall v_0 \in e
\end{array}
$$

Outline

Das Assignment Problem

Gegeben n Arbeitnehmer und n Arbeitsplätze. Besetzung von Arbeitsplatz j durch Arbeitnehmer i bringt Nutzen $c_{ij} \geq 0$. Maximiere den Gesamtnutzen.



$$
\begin{array}{lrcll}
\max & \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} & & & \\
(AP) & \sum_{j=1}^{n} x_{ij} & = & 1 & \forall i = 1, \ldots, n \\
& \sum_{i=1}^{n} x_{ij} & = & 1 & \forall j = 1, \ldots, n \\
& x_{ij} & \in & \{0, 1\} &
\end{array}
$$

Ganzzahlige Hülle

Zu $P = P(A, b) := \{x \in \mathbb{R}^n \mid Ax \leq b\}$ bezeichnen wir die konvexe Hülle der ganzzahligen Punkte als die ganzzahlige Hülle P_I von P.

Ganzzahlige Hülle

Zu $P = P(A, b) := \{x \in \mathbb{R}^n \mid Ax \leq b\}$ bezeichnen wir die konvexe Hülle der ganzzahligen Punkte als die ganzzahlige Hülle P_I von P.

Wir untersuchen hinreichende Kriterien dafür, dass $P = P_I$ für ein Polyeder $P = P(A, b)$ gilt.

Ganzzahlige Hülle

Zu $P = P(A, b) := \{x \in \mathbb{R}^n \mid Ax \leq b\}$ bezeichnen wir die konvexe Hülle der ganzzahligen Punkte als die ganzzahlige Hülle P_I von P.

Wir untersuchen hinreichende Kriterien dafür, dass $P = P_I$ für ein Polyeder $P = P(A, b)$ gilt.

Satz

$x \in P$ ist genau dann Ecke von $P = P(A, b)$, wenn es n linear unabhängige Zeilen I von A gibt, mit $A_I x = b_I$ also $x = A_I^{-1} b$.

Ganzzahlige Hülle

Zu $P = P(A, b) := \{x \in \mathbb{R}^n \mid Ax \leq b\}$ bezeichnen wir die konvexe Hülle der ganzzahligen Punkte als die ganzzahlige Hülle P_I von P.

Wir untersuchen hinreichende Kriterien dafür, dass $P = P_I$ für ein Polyeder $P = P(A, b)$ gilt.

Satz

$x \in P$ ist genau dann Ecke von $P = P(A, b)$, wenn es n linear unabhängige Zeilen I von A gibt, mit $A_I x = b_I$ also $x = A_I^{-1} b$.

Wie sehen Matrizen A aus, so dass $P(A, b) = P_I$ für beliebige ganzzahlige Vektoren b?

Outline

Totale Unimodularität

Eine Matrix $A \in \mathbb{Z}^{m \times n}$ heißt total unimodular wenn alle ihre Unterdeterminanten in $\{0, 1, -1\}$ liegen. Insbesondere $A \in \{0, \pm 1\}^{m \times n}$.

Totale Unimodularität

Eine Matrix $A \in \mathbb{Z}^{m \times n}$ heißt total unimodular wenn alle ihre Unterdeterminanten in $\{0, 1, -1\}$ liegen. Insbesondere $A \in \{0, \pm 1\}^{m \times n}$.

Satz

Sei $A \in \mathbb{Z}^{m \times n}$ total unimodular, $b \in \mathbb{Z}^m$ und $P = P(A, b)$. Dann ist $P = P_I$.

Totale Unimodularität

Eine Matrix $A \in \mathbb{Z}^{m \times n}$ heißt total unimodular wenn alle ihre Unterdeterminanten in $\{0, 1, -1\}$ liegen. Insbesondere $A \in \{0, \pm 1\}^{m \times n}$.

Satz

Sei $A \in \mathbb{Z}^{m \times n}$ total unimodular, $b \in \mathbb{Z}^m$ und $P = P(A, b)$. Dann ist $P = P_I$.

Beweis.

Sei x Ecke von P und I mit $A_I x = b$.

Totale Unimodularität

Eine Matrix $A \in \mathbb{Z}^{m \times n}$ heißt total unimodular wenn alle ihre Unterdeterminanten in $\{0, 1, -1\}$ liegen. Insbesondere $A \in \{0, \pm 1\}^{m \times n}$.

Satz

Sei $A \in \mathbb{Z}^{m \times n}$ total unimodular, $b \in \mathbb{Z}^m$ und $P = P(A, b)$. Dann ist $P = P_I$.

Beweis.

Sei x Ecke von P und I mit $A_I x = b$. Nach Cramerscher Regel ist

$$x_j = \frac{\det(\tilde{A})}{\det A_I} = \pm \det(\tilde{A}),$$

wobei $\tilde{A}$ die ganzzahlige Matrix ist, die aus A_I entsteht, wenn man die j-te Spalte durch b ersetzt.

Totale Unimodularität

Eine Matrix $A \in \mathbb{Z}^{m \times n}$ heißt total unimodular wenn alle ihre Unterdeterminanten in $\{0, 1, -1\}$ liegen. Insbesondere $A \in \{0, \pm 1\}^{m \times n}$.

Satz

Sei $A \in \mathbb{Z}^{m \times n}$ total unimodular, $b \in \mathbb{Z}^m$ und $P = P(A, b)$. Dann ist $P = P_I$.

Beweis.

Sei x Ecke von P und I mit $A_I x = b$. Nach Cramerscher Regel ist

$$x_j = \frac{\det(\tilde{A})}{\det A_I} = \pm \det(\tilde{A}),$$

wobei $\tilde{A}$ die ganzzahlige Matrix ist, die aus A_I entsteht, wenn man die j-te Spalte durch b ersetzt. Die Determinante einer ganzzahligen Matrix ist ganzzahlig. □

Totale Unimodularität II

Satz

Sind $c \in \mathbb{Z}^n, b \in \mathbb{Z}^m$ und ist $A \in \mathbb{Z}^{m \times n}$ total unimodular. Dann haben das primale Problem und das duale Problem

$$(P) \qquad \begin{array}{rcl} \max & c^\top x & \\ Ax & = & b \\ x & \geq & 0 \end{array} \qquad (D) \qquad \begin{array}{rcl} \min & u^\top b & \\ u^\top A & \geq & c^\top \end{array}$$

ganzzahlige Optimallösungen.

Totale Unimodularität II

Satz

Sind $c \in \mathbb{Z}^n, b \in \mathbb{Z}^m$ und ist $A \in \mathbb{Z}^{m \times n}$ total unimodular. Dann haben das primale Problem und das duale Problem

$$(P) \qquad \begin{array}{rcl} \max & c^\top x & \\ Ax & = & b \\ x & \geq & 0 \end{array} \qquad (D) \qquad \begin{array}{rcl} \min & u^\top b & \\ u^\top A & \geq & c^\top \end{array}$$

ganzzahlige Optimallösungen.

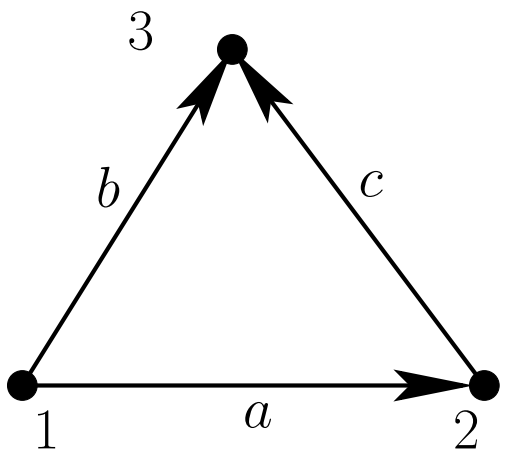
Satz (Hoffman und Kruskal 1956)

Sei $A \in \mathbb{Z}^{m \times n}$. Dann ist $P = \{x \in \mathbb{R}^n \mid Ax \leq b, x \geq 0\}$ genau dann für alle ganzzahligen $b \in \mathbb{Z}^m$ ganzzahlig, wenn A total unimodular ist.

Probleme vom Netzwerktyp

Ist $G = (V, A)$ ein Netzwerk, so ist die Inzidenzmatrix $A \in \{\pm 1, 0\}^{V \times A}$ gegeben durch

$$A_{v,a=(h,t)} = \begin{cases} 1 & \text{falls} \quad v = h \\ -1 & \text{falls} \quad v = t \\ 0 & \text{sonst} \end{cases}$$



$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} a & b & c \end{array} \\ \begin{pmatrix} 1 & 1 & 0 \\ -1 & 0 & 1 \\ 0 & -1 & -1 \end{pmatrix} \end{array}.$$

Probleme vom Netzwerktyp II

Satz

Die Inzidenzmatrix A eines Digraphen ist total unimodular.

Probleme vom Netzwerktyp II

Satz

Die Inzidenzmatrix A eines Digraphen ist total unimodular.

Beweis.

Sei B eine quadratische Untermatrix von A.

Probleme vom Netzwerktyp II

Satz

Die Inzidenzmatrix A eines Digraphen ist total unimodular.

Beweis.

Sei B eine quadratische Untermatrix von A.

1. Fall: Es gibt eine Nullspalte

Probleme vom Netzwerktyp II

Satz

Die Inzidenzmatrix A eines Digraphen ist total unimodular.

Beweis.

Sei B eine quadratische Untermatrix von A.

1. Fall: Es gibt eine Nullspalte $\Rightarrow \det(B) = 0$.

Probleme vom Netzwerktyp II

Satz

Die Inzidenzmatrix A eines Digraphen ist total unimodular.

Beweis.

Sei B eine quadratische Untermatrix von A.

1. Fall: Es gibt eine Nullspalte $\Rightarrow \det(B) = 0$.
2. Fall: Jede Spalte hat genau eine 1 und eine -1

Probleme vom Netzwerktyp II

Satz

Die Inzidenzmatrix A eines Digraphen ist total unimodular.

Beweis.

Sei B eine quadratische Untermatrix von A.

1. Fall: Es gibt eine Nullspalte $\Rightarrow \det(B) = 0$.
2. Fall: Jede Spalte hat genau eine 1 und eine -1 $\Rightarrow \det(B) = 0$.

Probleme vom Netzwerktyp II

Satz

Die Inzidenzmatrix A eines Digraphen ist total unimodular.

Beweis.

Sei B eine quadratische Untermatrix von A.

1. Fall: Es gibt eine Nullspalte $\Rightarrow \det(B) = 0$.
2. Fall: Jede Spalte hat genau eine 1 und eine -1
   $\Rightarrow \det(B) = 0$.
3. Fall: Es gibt eine Spalte, die genau ein Element $\neq 0$ enthält.

Probleme vom Netzwerktyp II

Satz

Die Inzidenzmatrix A eines Digraphen ist total unimodular.

Beweis.

Sei B eine quadratische Untermatrix von A.

1. Fall: Es gibt eine Nullspalte $\Rightarrow \det(B) = 0$.
2. Fall: Jede Spalte hat genau eine 1 und eine -1
   $\Rightarrow \det(B) = 0$.
3. Fall: Es gibt eine Spalte, die genau ein Element $\neq 0$ enthält.
   Entwicklung nach diese Spalte, Induktion, fertig.

□

Probleme vom Netzwerktyp III

Probleme vom „Netzwerktyp“ wie

Probleme vom Netzwerktyp III

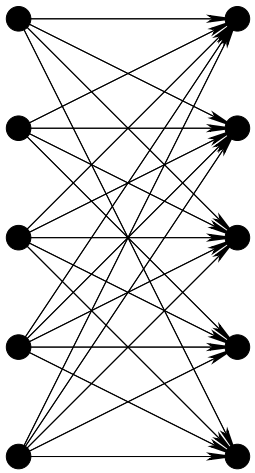
Probleme vom „Netzwerktyp“ wie

- Maximale Flüsse

Probleme vom Netzwerktyp III

Probleme vom „Netzwerktyp“ wie

- Maximale Flüsse
- Zuweisungsprobleme



Probleme vom Netzwerktyp III

Probleme vom „Netzwerktyp“ wie

- Maximale Flüsse
- Zuweisungsprobleme
- Minimale Schnitte

Probleme vom Netzwerktyp III

Probleme vom „Netzwerktyp“ wie

- Maximale Flüsse
- Zuweisungsprobleme
- Minimale Schnitte
- Kürzeste Wege

Probleme vom Netzwerktyp III

Probleme vom „Netzwerktyp“ wie

- Maximale Flüsse
- Zuweisungsprobleme
- Minimale Schnitte
- Kürzeste Wege
- Kostenminimale Flüsse

Probleme vom Netzwerktyp III

Probleme vom „Netzwerktyp“ wie

- Maximale Flüsse
- Zuweisungsprobleme
- Minimale Schnitte
- Kürzeste Wege
- Kostenminimale Flüsse

haben total unimodulare Restriktionsmatrizen. Lineare Optimierung liefert also schon ganzzahlige Lösungen.

Probleme vom Netzwerktyp III

Probleme vom „Netzwerktyp“ wie

- Maximale Flüsse
- Zuweisungsprobleme
- Minimale Schnitte
- Kürzeste Wege
- Kostenminimale Flüsse

haben total unimodulare Restriktionsmatrizen. Lineare Optimierung liefert also schon ganzzahlige Lösungen.

Satz (Seymour 1980)

Alle total unimodularen Matrizen lassen sich aus Netzwerkflussmatrizen, deren Transponierten und einer weiteren speziellen Matrix durch zwei Verknüpfungsoperationen konstruieren.

Outline

Nichtbipartites Matching

Sei $G = K_n$ der vollständige Graph mit n Knoten und Kantengewichtsfunktion c. Für $v \in V$ sei $\delta(v)$ die Menge der zu v inzidenten Kanten. Das gewichtete Minimum Perfect Matching Problem wird durch folgendes BP modelliert.

Nichtbipartites Matching

Sei $G = K_n$ der vollständige Graph mit n Knoten und Kantengewichtsfunktion c. Für $v \in V$ sei $\delta(v)$ die Menge der zu v inzidenten Kanten. Das gewichtete Minimum Perfect Matching Problem wird durch folgendes BP modelliert.

$$\min \quad \sum_{e \in E} c_e x_e$$

$$x_e \quad \in \quad \{0, 1\}$$

Nichtbipartites Matching

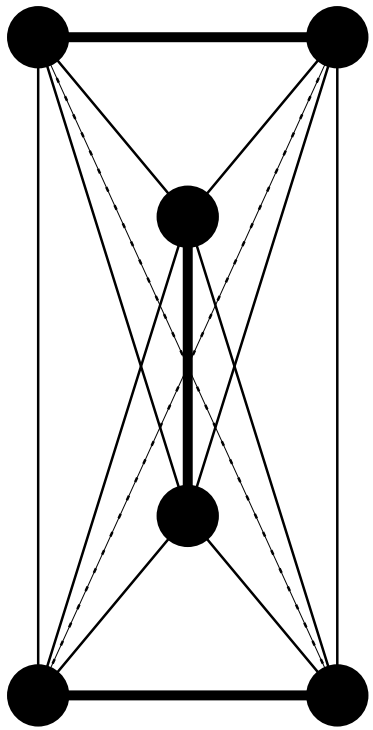
Sei $G = K_n$ der vollständige Graph mit n Knoten und Kantengewichtsfunktion c. Für $v \in V$ sei $\delta(v)$ die Menge der zu v inzidenten Kanten. Das gewichtete Minimum Perfect Matching Problem wird durch folgendes BP modelliert.

$$
\begin{array}{rlcll}
 & \min & \sum_{e \in E} c_e x_e & & \\
(MPMP) & & \sum_{e \in \delta(v)} x_e & = & 1 \qquad \forall v \in V \\
 & & x_e & \in & \{0,1\}
\end{array}
$$

Nichtbipartites Matching

Sei $G = K_n$ der vollständige Graph mit n Knoten und Kantengewichtsfunktion c. Für $v \in V$ sei $\delta(v)$ die Menge der zu v inzidenten Kanten. Das gewichtete Minimum Perfect Matching Problem wird durch folgendes BP modelliert.

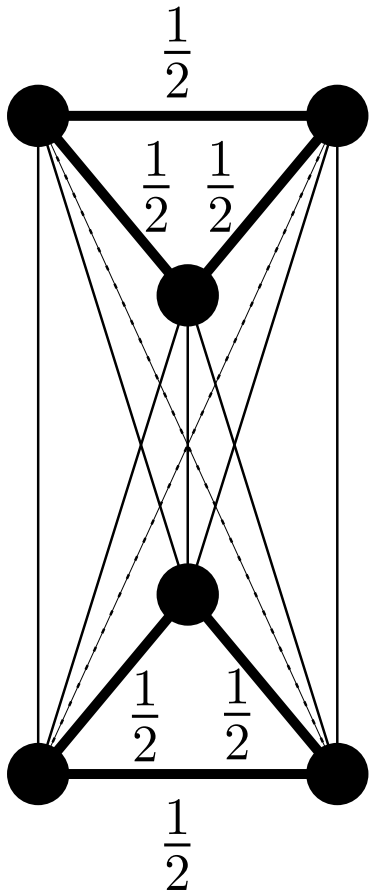
$$
\begin{array}{rlcll}
 & \min & \sum_{e \in E} c_e x_e & & \\
(MPMP) & \sum_{e \in \delta(v)} x_e & = & 1 & \forall v \in V \\
 & x_e & \in & \{0,1\} &
\end{array}
$$



Nichtbipartites Matching

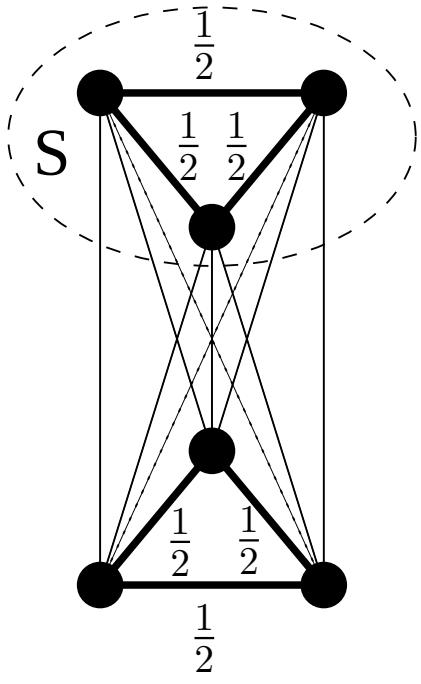
Sei $G = K_n$ der vollständige Graph mit n Knoten und Kantengewichtsfunktion c. Für $v \in V$ sei $\delta(v)$ die Menge der zu v inzidenten Kanten. Das gewichtete Minimum Perfect Matching Problem wird durch folgendes BP modelliert.

$$
\begin{array}{rlcll}
 & \min & \sum_{e \in E} c_e x_e & & \\
(MPMP) & & \sum_{e \in \delta(v)} x_e & = & 1 \qquad \forall v \in V \\
 & & x_e & \geq & 0
\end{array}
$$



Schnittebenen

Die halbzahlige Lösung verletzt die folgende, offensichtlich für alle perfekten Matchings gültige Ungleichung.

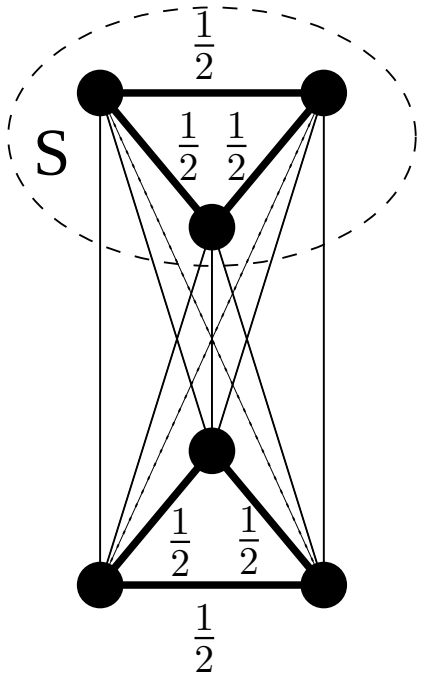


Schnittebenen

Die halbzahlige Lösung verletzt die folgende, offensichtlich für alle perfekten Matchings gültige Ungleichung.

$$\sum_{e \in \delta(S)} x_e \geq 1.$$

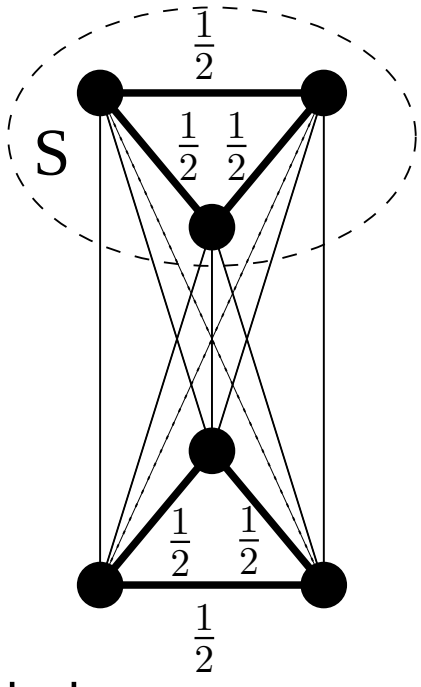
Dabei bezeichne $\delta(S)$ die Menge aller Kanten, die genau einen Endknoten in S haben.



Schnittebenen

Die halbzahlige Lösung verletzt die folgende, offensichtlich für alle perfekten Matchings gültige Ungleichung.

$$\sum_{e \in \delta(S)} x_e \geq 1.$$



Dabei bezeichne $\delta(S)$ die Menge aller Kanten, die genau einen Endknoten in S haben.

Verletzte Ungleichungen dieses Typs kann man beim Matchingproblem effizient finden. Das Separationsproblem für das Perfect Matching Polytope ist polynomiell lösbar.

Grötschel, Lovász, Schrijver

Sei $P \subseteq \mathbb{Q}^n$ eine konvexe Menge $x \in \mathbb{Q}^n$.

Grötschel, Lovász, Schrijver

Sei $P \subseteq \mathbb{Q}^n$ eine konvexe Menge $x \in \mathbb{Q}^n$. Das Separationsproblem besteht darin,

Grötschel, Lovász, Schrijver

Sei $P \subseteq \mathbb{Q}^n$ eine konvexe Menge $x \in \mathbb{Q}^n$. Das Separationsproblem besteht darin,

- entweder zu bestätigen, dass $x \in P$ oder

Grötschel, Lovász, Schrijver

Sei $P \subseteq \mathbb{Q}^n$ eine konvexe Menge $x \in \mathbb{Q}^n$. Das Separationsproblem besteht darin,

- entweder zu bestätigen, dass $x \in P$ oder
- ein $c \in \mathbb{Q}^n$ anzugeben mit

$$\forall z \in P : c^\top x < c^\top z.$$

Grötschel, Lovász, Schrijver

Sei $P \subseteq \mathbb{Q}^n$ eine konvexe Menge $x \in \mathbb{Q}^n$. Das Separationsproblem besteht darin,

- entweder zu bestätigen, dass $x \in P$ oder
- ein $c \in \mathbb{Q}^n$ anzugeben mit

$$\forall z \in P : c^\top x < c^\top z.$$

Satz (Grötschel, Lovász, Schrijver '81)

Separieren und Optimieren sind polynomiell äquivalent für jede gegebene Klasse von Polyedern.

Grötschel, Lovász, Schrijver

Sei $P \subseteq \mathbb{Q}^n$ eine konvexe Menge $x \in \mathbb{Q}^n$. Das Separationsproblem besteht darin,

- entweder zu bestätigen, dass $x \in P$ oder
- ein $c \in \mathbb{Q}^n$ anzugeben mit

$$\forall z \in P : c^\top x < c^\top z.$$

Satz (Grötschel, Lovász, Schrijver '81)

Separieren und Optimieren sind polynomiell äquivalent für jede gegebene Klasse von Polyedern.

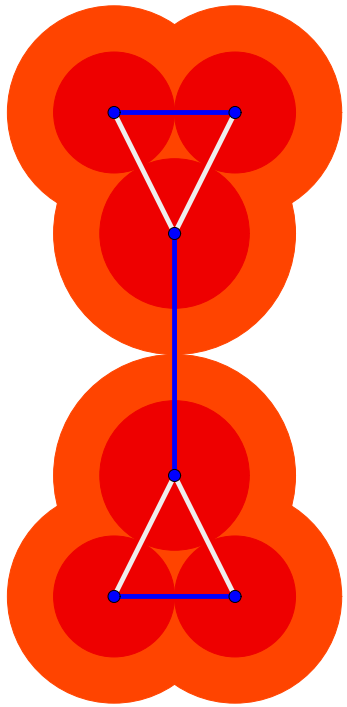
Der Beweis fußt auf der Polynomialität der Ellipsoidmethode.

Das Perfect Matching Polytope

$$
\begin{array}{rlll}
\min & \sum_{e \in E} c_e x_e & & \\
(MPMP) & \sum_{e \in \delta(v)} x_e & = & 1 \qquad \forall v \in V \\
& x_e & \in & \{0,1\}
\end{array}
$$

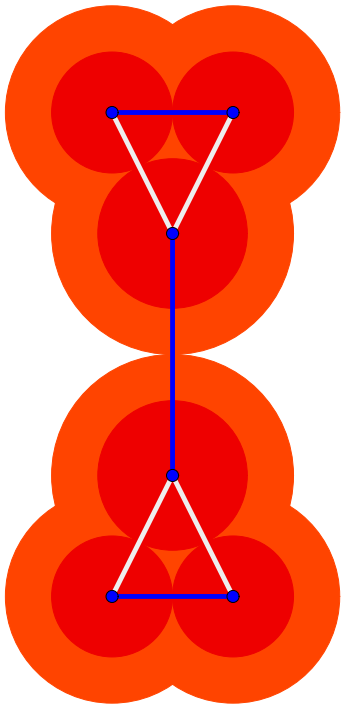
Das Perfect Matching Polytope

$$
\begin{array}{rlcll}
\min & \sum_{e\in E} c_e x_e & & & \\
(MPMP) & \sum_{e\in\delta(v)} x_e & = & 1 & \forall v \in V \\
 & \sum_{e\in\delta(S)} x_e & \geq & 1 & \text{für } S\subseteq V \\
 & & & & |S| \text{ ungerade} \\
 & x_e & \geq & 0 &
\end{array}
$$



Das Perfect Matching Polytope

$$
\begin{array}{rlcll}
\min & \sum_{e\in E} c_e x_e & & & \\
(MPMP) & \sum_{e\in\delta(v)} x_e & = & 1 & \forall v \in V \\
 & \sum_{e\in\delta(S)} x_e & \geq & 1 & \text{für } S\subseteq V \\
 & & & & |S| \text{ ungerade} \\
 & x_e & \geq & 0 &
\end{array}
$$



Das Perfect Matching Polytope $MPMP_I$ lässt sich vollständig beschreiben – allerdings mittels exponentiell vieler Ungleichungen.

Outline

Spezielle rechte Seiten

Darüber hinaus untersucht man Ganzzahligkeitseigenschaften für spezielle Paare von Matrizen und rechten Seiten. Solche findet etwa man unter den Stichworten

- TDI-Systeme

Spezielle rechte Seiten

Darüber hinaus untersucht man Ganzzahligkeitseigenschaften für spezielle Paare von Matrizen und rechten Seiten. Solche findet etwa man unter den Stichworten

- TDI-Systeme
- perfekte Matrizen (bei Packungsproblemen)

Spezielle rechte Seiten

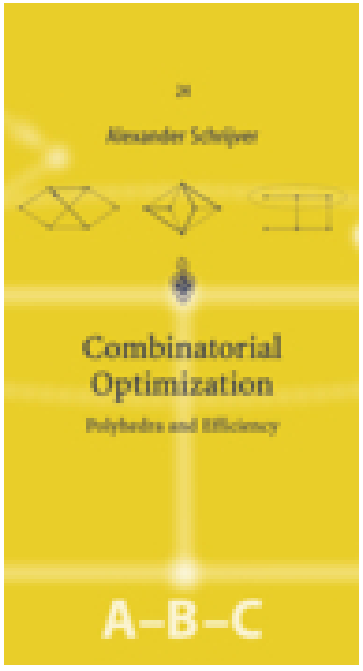
Darüber hinaus untersucht man Ganzzahligkeitseigenschaften für spezielle Paare von Matrizen und rechten Seiten. Solche findet etwa man unter den Stichworten

- TDI-Systeme
- perfekte Matrizen (bei Packungsproblemen)
- ideale Matrizen (bei Überdeckungsproblemen).

Spezielle rechte Seiten

Darüber hinaus untersucht man Ganzzahligkeitseigenschaften für spezielle Paare von Matrizen und rechten Seiten. Solche findet etwa man unter den Stichworten

- TDI-Systeme
- perfekte Matrizen (bei Packungsproblemen)
- ideale Matrizen (bei Überdeckungsproblemen).

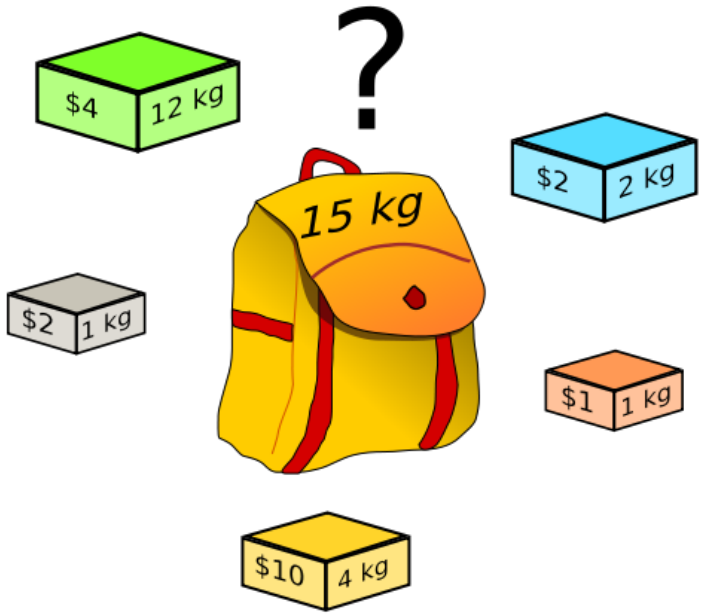


Outline II

Das Rucksack Problem (Knapsack Problem)

Gegeben sei ein Rucksack mit Kapazität $b \in \mathbb{N}$ und n Gegenstände. Gegenstand i hat Gewicht (Volumen) a_i und bringt Nutzen c_i. Packe Rucksack so, dass der Nutzen maximiert wird.

$$
\begin{array}{lrcl}
\max & \sum_{j=1}^{n} c_j x_j & & \\
(KP) & \sum_{j=1}^{n} a_j x_j & \leq & b \\
 & x_j & \in & \{0,1\}
\end{array}
$$



Outline

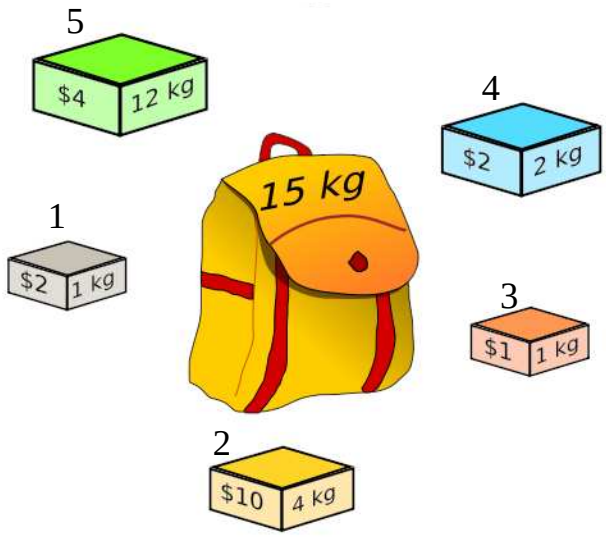
Entscheidungsprobleme, $\mathcal{P}$

$\Pi : \Sigma^* \rightarrow \{\text{ja, nein}\}$ Entscheidungsproblem

Entscheidungsprobleme, $\mathcal{P}$

$\Pi : \Sigma^* \rightarrow \{\text{ja, nein}\}$ Entscheidungsproblem

Beim Knapsackproblem lautet die Version als Entscheidungsproblem etwa: „Kann ich mit dem Rucksack Waren im Wert von mindestens 14$ mitnehmen?“



Entscheidungsprobleme, $\mathcal{P}$

$\Pi : \Sigma^* \rightarrow \{\text{ja, nein}\}$ Entscheidungsproblem

A_Π Algorithmus für Π

Entscheidungsprobleme, $\mathcal{P}$

$\Pi : \Sigma^* \to \{\text{ja, nein}\}$	Entscheidungsproblem
A_Π	Algorithmus für Π
$\ell_{A_\Pi} : \Sigma^* \to \mathbb{N}$	Laufzeitfunktion

Entscheidungsprobleme, $\mathcal{P}$

$\Pi : \Sigma^* \to \{\text{ja, nein}\}$	Entscheidungsproblem
A_Π	Algorithmus für Π
$\ell_{A_\Pi} : \Sigma^* \to \mathbb{N}$	Laufzeitfunktion
$\ell^{wc}_{A_\Pi} : \mathbb{N} \to \mathbb{N}$	$\ell^{wc}_{A_\Pi}(n) := \max\{\ell_{A_\Pi}(\sigma) \mid \langle\sigma\rangle = n\}$

Dabei ist $\langle\sigma\rangle$ die Kodierungslänge der Eingabe σ.

Entscheidungsprobleme, $\mathcal{P}$

$\Pi : \Sigma^* \to \{\text{ja, nein}\}$	Entscheidungsproblem
A_Π	Algorithmus für Π
$\ell_{A_\Pi} : \Sigma^* \to \mathbb{N}$	Laufzeitfunktion
$\ell^{wc}_{A_\Pi} : \mathbb{N} \to \mathbb{N}$	$\ell^{wc}_{A_\Pi}(n) := \max\{\ell_{A_\Pi}(\sigma) \mid \langle\sigma\rangle = n\}$

Dabei ist $\langle\sigma\rangle$ die Kodierungslänge der Eingabe σ. Beim Knapsackproblem etwa $\log_2 b + \sum_{i=1}^{n}(\log_2 a_i + \log_2 c_i)$.

Entscheidungsprobleme, $\mathcal{P}$

$\Pi : \Sigma^* \to \{\text{ja, nein}\}$	Entscheidungsproblem
A_Π	Algorithmus für Π
$\ell_{A_\Pi} : \Sigma^* \to \mathbb{N}$	Laufzeitfunktion
$\ell^{wc}_{A_\Pi} : \mathbb{N} \to \mathbb{N}$	$\ell^{wc}_{A_\Pi}(n) := \max\{\ell_{A_\Pi}(\sigma) \mid \langle\sigma\rangle = n\}$

Dabei ist $\langle\sigma\rangle$ die Kodierungslänge der Eingabe σ. A_Π heißt polynomiell, wenn es $k \in \mathbb{N}$ gibt mit

$$\ell^{wc}_{A_\Pi}(n) \leq n^k \text{ für alle hinreichend großen } n \in \mathbb{N}.$$

Entscheidungsprobleme, $\mathcal{P}$

$\Pi : \Sigma^* \to \{\text{ja, nein}\}$	Entscheidungsproblem
A_Π	Algorithmus für Π
$\ell_{A_\Pi} : \Sigma^* \to \mathbb{N}$	Laufzeitfunktion
$\ell^{wc}_{A_\Pi} : \mathbb{N} \to \mathbb{N}$	$\ell^{wc}_{A_\Pi}(n) := \max\{\ell_{A_\Pi}(\sigma) \mid \langle\sigma\rangle = n\}$

Dabei ist $\langle\sigma\rangle$ die Kodierungslänge der Eingabe σ. A_Π heißt polynomiell, wenn es $k \in \mathbb{N}$ gibt mit

$$\ell^{wc}_{A_\Pi}(n) \leq n^k \text{ für alle hinreichend großen } n \in \mathbb{N}.$$

$\mathcal{P}$ bezeichne die Menge aller Entscheidungsprobleme, die mit einem Algorithmus mit polynomieller Laufzeit gelöst werden können.

Outline

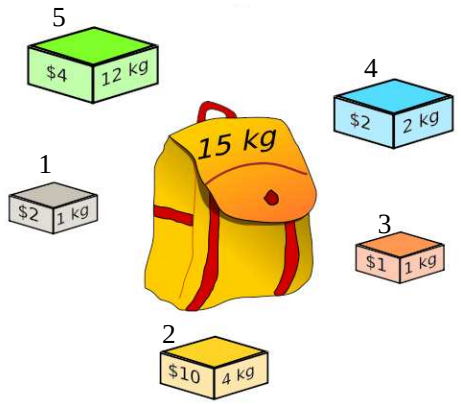
$\mathcal{NP}$ und $\mathcal{NP}$-vollständige Probleme

Mit $\mathcal{NP}$ bezeichnen wir die Menge aller Entscheidungsprobleme, bei denen Instanzen, bei denen die Antwort ja lautet, mit dem richtigen Zertifikat (Hinweis von außen) mit einem Algorithmus mit polynomieller Laufzeit verifiziert werden können.

$\mathcal{NP}$ und $\mathcal{NP}$-vollständige Probleme

Mit $\mathcal{NP}$ bezeichnen wir die Menge aller Entscheidungsprobleme, bei denen Instanzen, bei denen die Antwort ja lautet, mit dem richtigen Zertifikat (Hinweis von außen) mit einem Algorithmus mit polynomieller Laufzeit verifiziert werden können.

Beim Knapsackproblem lautet das Zertifikat hier etwa $\{1, 2, 4\}$.



$\mathcal{NP}$ und $\mathcal{NP}$-vollständige Probleme

Mit $\mathcal{NP}$ bezeichnen wir die Menge aller Entscheidungsprobleme, bei denen Instanzen, bei denen die Antwort ja lautet, mit dem richtigen Zertifikat (Hinweis von außen) mit einem Algorithmus mit polynomieller Laufzeit verifiziert werden können.

Problem

Ist $\mathcal{P} \neq \mathcal{NP}$?

$\mathcal{NP}$ und $\mathcal{NP}$-vollständige Probleme

Mit $\mathcal{NP}$ bezeichnen wir die Menge aller Entscheidungsprobleme, bei denen Instanzen, bei denen die Antwort ja lautet, mit dem richtigen Zertifikat (Hinweis von außen) mit einem Algorithmus mit polynomieller Laufzeit verifiziert werden können.

Problem

Ist $\mathcal{P} \neq \mathcal{NP}$?

Ein Problem heißt $\mathcal{NP}$-vollständig, wenn es in $\mathcal{NP}$ liegt und von einem Algorithmus mit polynomieller Laufzeit für dieses Problem ebensolche Algorithmen für beliebige Probleme in $\mathcal{NP}$ abgeleitet werden können.

$\mathcal{NP}$ und $\mathcal{NP}$-vollständige Probleme

Mit $\mathcal{NP}$ bezeichnen wir die Menge aller Entscheidungsprobleme, bei denen Instanzen, bei denen die Antwort ja lautet, mit dem richtigen Zertifikat (Hinweis von außen) mit einem Algorithmus mit polynomieller Laufzeit verifiziert werden können.

Problem

Ist $\mathcal{P} \neq \mathcal{NP}$?

Ein Problem heißt $\mathcal{NP}$-vollständig, wenn es in $\mathcal{NP}$ liegt und von einem Algorithmus mit polynomieller Laufzeit für dieses Problem ebensolche Algorithmen für beliebige Probleme in $\mathcal{NP}$ abgeleitet werden können.

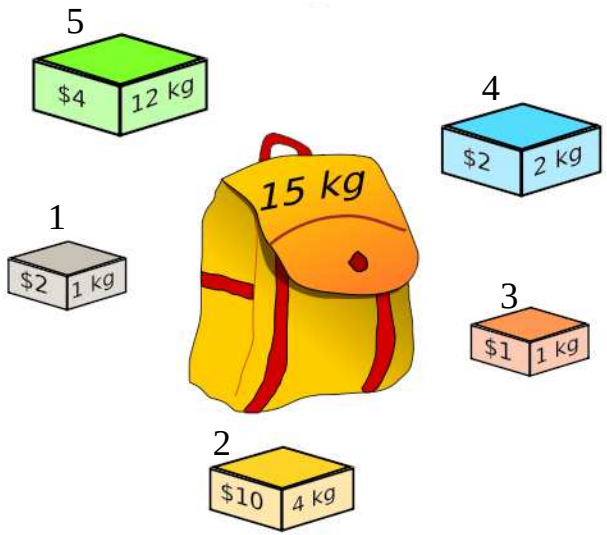
Satz

Das Knapsack Problem ist $\mathcal{NP}$-vollständig (und damit insbesondere auch die ganzzahlige Optimierung).

Outline

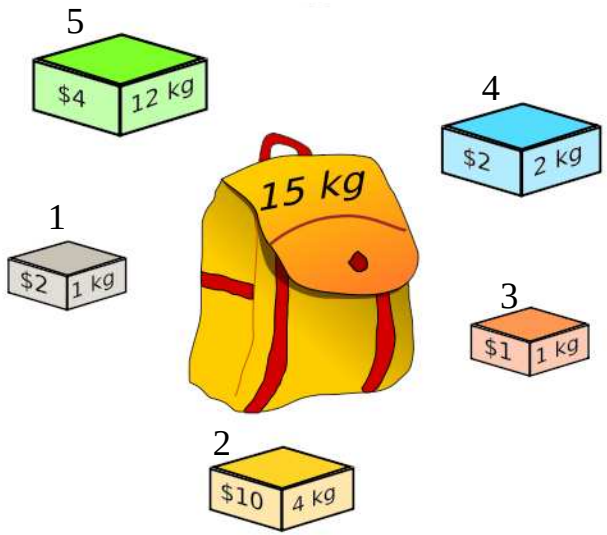
Dynamic Programming

Bezeichnen wir mit $R(b', i)$ den maximalen Nutzen eines Rucksacks mit Kapazität b', der mit den Items $\{1, 2, \ldots, i\}$ gepackt wird, so ist



Dynamic Programming

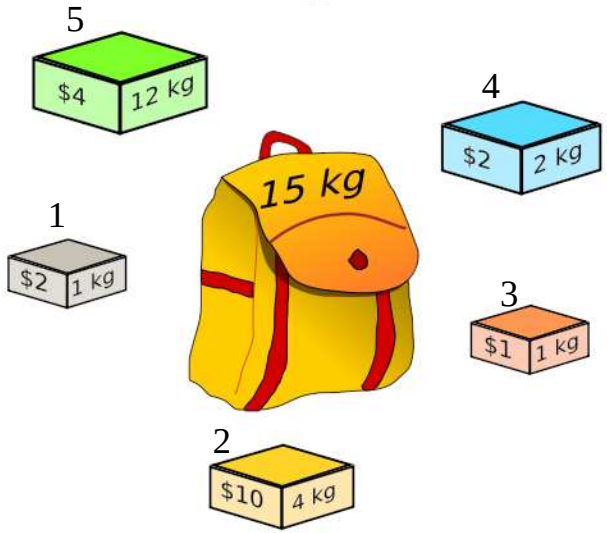
Bezeichnen wir mit $R(b', i)$ den maximalen Nutzen eines Rucksacks mit Kapazität b', der mit den Items $\{1, 2, \ldots, i\}$ gepackt wird, so ist



$$R(b', i) = \max\{R(b', i-1), R(b' - a_i, i-1) + c_i\}$$

Dynamic Programming

Bezeichnen wir mit $R(b', i)$ den maximalen Nutzen eines Rucksacks mit Kapazität b', der mit den Items $\{1, 2, \ldots, i\}$ gepackt wird, so ist

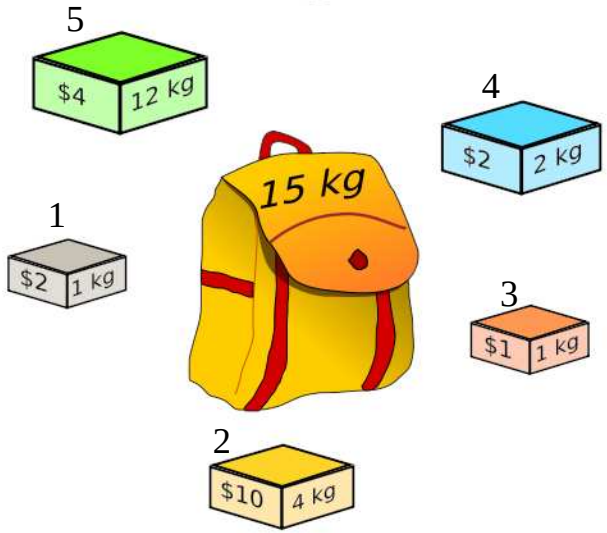


$$R(b', i) = \max\{R(b', i-1), R(b' - a_i, i-1) + c_i\}$$

$$R(5, 2) = 12,$$

Dynamic Programming

Bezeichnen wir mit $R(b', i)$ den maximalen Nutzen eines Rucksacks mit Kapazität b', der mit den Items $\{1, 2, \ldots, i\}$ gepackt wird, so ist

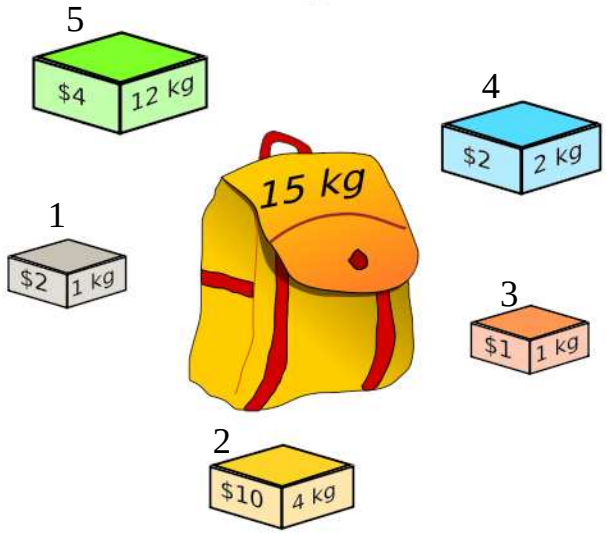


$$R(b', i) = \max\{R(b', i-1), R(b' - a_i, i-1) + c_i\}$$

$$R(5, 2) = 12, R(8, 3) = R(6, 3) = R(5, 2) + c_3 = 13,$$

Dynamic Programming

Bezeichnen wir mit $R(b', i)$ den maximalen Nutzen eines Rucksacks mit Kapazität b', der mit den Items $\{1, 2, \ldots, i\}$ gepackt wird, so ist

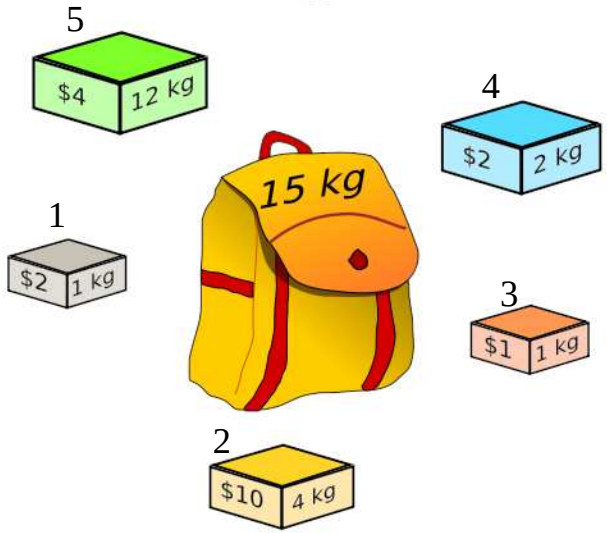


$$R(b', i) = \max\{R(b', i-1), R(b' - a_i, i-1) + c_i\}$$

$$R(5,2) = 12, R(8,3) = R(6,3) = R(5,2) + c_3 = 13,$$
$$R(8,4) = R(6,3) + c_4 = 15,$$

Dynamic Programming

Bezeichnen wir mit $R(b', i)$ den maximalen Nutzen eines Rucksacks mit Kapazität b', der mit den Items $\{1, 2, \ldots, i\}$ gepackt wird, so ist

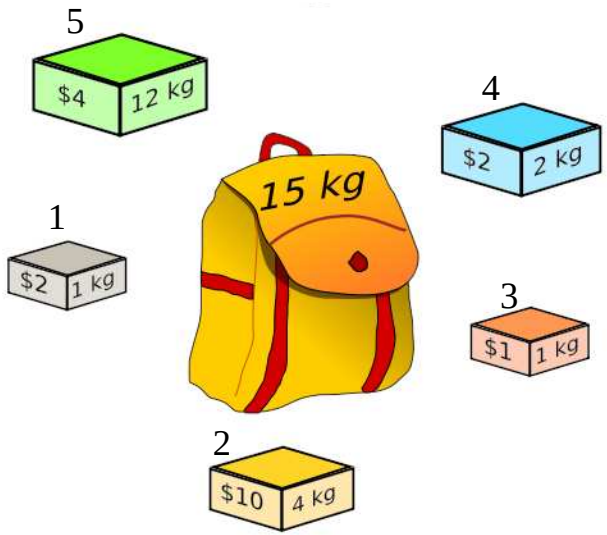


$$R(b', i) = \max\{R(b', i-1), R(b' - a_i, i-1) + c_i\}$$

$$R(5,2) = 12, R(8,3) = R(6,3) = R(5,2) + c_3 = 13,$$
$$R(8,4) = R(6,3) + c_4 = 15,$$
$$R(15,5) = \max\{R(3,4) + 4, R(15,4)\} = 15.$$

Dynamic Programming

Bezeichnen wir mit $R(b', i)$ den maximalen Nutzen eines Rucksacks mit Kapazität b', der mit den Items $\{1, 2, \ldots, i\}$ gepackt wird, so ist



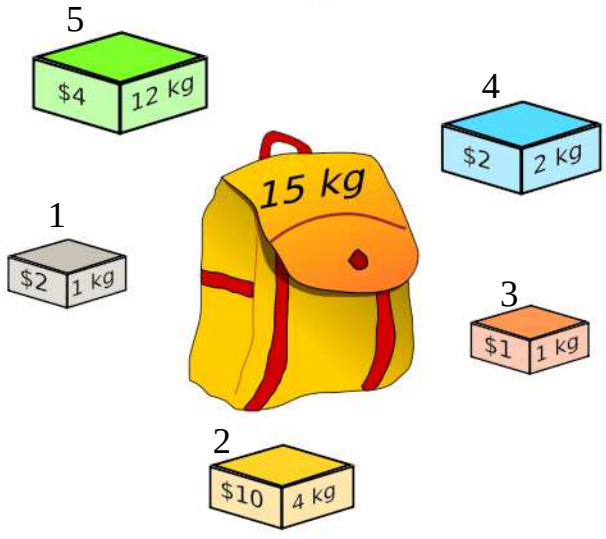
$$R(b', i) = \max\{R(b', i-1), R(b' - a_i, i-1) + c_i\}$$

Satz:

Das Knapsack Problem kann in $O(bn)$ gelöst werden.

Dynamic Programming

Bezeichnen wir mit $R(b', i)$ den maximalen Nutzen eines Rucksacks mit Kapazität b', der mit den Items $\{1, 2, \ldots, i\}$ gepackt wird, so ist



$$R(b', i) = \max\{R(b', i-1), R(b' - a_i, i-1) + c_i\}$$

Satz:

Das Knapsack Problem kann in $O(bn)$ gelöst werden.

Der Algorithmus ist pseudopolynomiell, polynomiell in der Anzahl der Gegenstände aber exponentiell in der Kodierungslänge.

Outline

Egon Balas “Facets of the Knapsack Polytope”

A necessary and sufficient condition is given for an inequality with coefficients 0 or 1 to define a facet of the knapsack polytope, i.e., of the convex hull of 0—1 points satisfying a given linear inequality. A sufficient condition is also established for a larger class of inequalities (with coefficients not restricted to 0 and 1) to define a facet for the same polytope, and a procedure is given for generating all facets in the above two classes. The procedure can be viewed as a way of generating cutting planes for 0–1 programs.

Egon Balas “Facets of the Knapsack Polytope”

A necessary and sufficient condition is given for an inequality with coefficients 0 or 1 to define a facet of the knapsack polytope, i.e., of the convex hull of 0–1 points satisfying a given linear inequality. A sufficient condition is also established for a larger class of inequalities (with coefficients not restricted to 0 and 1) to define a facet for the same polytope, and a procedure is given for generating all facets in the above two classes. The procedure can be viewed as a way of generating cutting planes for 0–1 programs.

Egon Balas “Facets of the Knapsack Polytope”

A necessary and sufficient condition is given for an inequality with coefficients 0 or 1 to define a facet of the knapsack polytope, i.e., of the convex hull of 0—1 points satisfying a given linear inequality. A sufficient condition is also established for a larger class of inequalities (with coefficients not restricted to 0 and 1) to define a facet for the same polytope, and a procedure is given for generating all facets in the above two classes. The procedure can be viewed as a way of generating cutting planes for 0–1 programs.

Egon Balas “Facets of the Knapsack Polytope”

A necessary and sufficient condition is given for an inequality with coefficients 0 or 1 to define a facet of the knapsack polytope, i.e., of the convex hull of 0–1 points satisfying a given linear inequality. A sufficient condition is also established for a larger class of inequalities (with coefficients not restricted to 0 and 1) to define a facet for the same polytope, and a procedure is given for generating all facets in the above two classes. The procedure can be viewed as a way of generating cutting planes for 0–1 programs.

Beobachtung (Edmonds ’68?)

Es gibt für kein Polytop P_I eines $\mathcal{NP}$-vollständigen Problems eine explizite Beschreibung aller Facetten, es sei denn $\mathcal{NP} =$ co-$\mathcal{NP}$.

Egon Balas “Facets of the Knapsack Polytope”

Beobachtung (Edmonds ’68?)

Es gibt für kein Polytop P_I eines $\mathcal{NP}$-vollständigen Problems eine explizite Beschreibung aller Facetten, es sei denn $\mathcal{NP} =$ co-$\mathcal{NP}$.

Beweis:

Wir haben zu zeigen, dass die Frage: „Gibt es eine Packung des Rucksacks mit Wert 16$“ mit Hilfe von außen in Polynomialzeit verneint werden kann, wenn man die Facetten des Knapsack-Polytops kennt.

Egon Balas “Facets of the Knapsack Polytope”

Beobachtung (Edmonds ’68?)

Es gibt für kein Polytop P_I eines $\mathcal{NP}$-vollständigen Problems eine explizite Beschreibung aller Facetten, es sei denn $\mathcal{NP} = \text{co-}\mathcal{NP}$.

Beweis:

Wir haben zu zeigen, dass die Frage: „Gibt es eine Packung des Rucksacks mit Wert 16$“ mit Hilfe von außen in Polynomialzeit verneint werden kann, wenn man die Facetten des Knapsack-Polytops kennt. Da $\sum_{j=1}^{n} c_j x_j \leq 15$ eine gültige Ungleichung für das Polytop ist, kann man diese durch positive Linearkombinationen von Facettenungleichungen herleiten.

Egon Balas “Facets of the Knapsack Polytope”

Beobachtung (Edmonds ’68?)

Es gibt für kein Polytop P_I eines $\mathcal{NP}$-vollständigen Problems eine explizite Beschreibung aller Facetten, es sei denn $\mathcal{NP} = \text{co-}\mathcal{NP}$.

Beweis:

Wir haben zu zeigen, dass die Frage: „Gibt es eine Packung des Rucksacks mit Wert 16$“ mit Hilfe von außen in Polynomialzeit verneint werden kann, wenn man die Facetten des Knapsack-Polytops kennt. Da $\sum_{j=1}^{n} c_j x_j \leq 15$ eine gültige Ungleichung für das Polytop ist, kann man diese durch positive Linearkombinationen von Facettenungleichungen herleiten. Diese Facettenungleichungen und die Koeffizienten werden als Hinweis von außen gegeben.

Typen von Facetten

Erinnerung (Satz von Minkowski)

Jedes Polytop P_I lässt sich durch ein Ungleichungssystem beschreiben. Ungleichungen, die zu einer Facette gehören, nennen wir selber Facetten.

Typen von Facetten

Erinnerung (Satz von Minkowski)

Jedes Polytop P_I lässt sich durch ein Ungleichungssystem beschreiben. Ungleichungen, die zu einer Facette gehören, nennen wir selber Facetten.

Wir unterscheiden bei Polytopen P_I von $\mathcal{NP}$-vollständigen Problemen zwischen Facetten, die

Typen von Facetten

Erinnerung (Satz von Minkowski)

Jedes Polytop P_I lässt sich durch ein Ungleichungssystem beschreiben. Ungleichungen, die zu einer Facette gehören, nennen wir selber Facetten.

Wir unterscheiden bei Polytopen P_I von $\mathcal{NP}$-vollständigen Problemen zwischen Facetten, die

- wir in Polynomialzeit separieren können,

Typen von Facetten

Erinnerung (Satz von Minkowski)

Jedes Polytop P_I lässt sich durch ein Ungleichungssystem beschreiben. Ungleichungen, die zu einer Facette gehören, nennen wir selber Facetten.

Wir unterscheiden bei Polytopen P_I von $\mathcal{NP}$-vollständigen Problemen zwischen Facetten, die

- wir in Polynomialzeit separieren können,
- wir kennen, wir aber nicht in Polynomialzeit separieren können,

Typen von Facetten

Erinnerung (Satz von Minkowski)

Jedes Polytop P_I lässt sich durch ein Ungleichungssystem beschreiben. Ungleichungen, die zu einer Facette gehören, nennen wir selber Facetten.

Wir unterscheiden bei Polytopen P_I von $\mathcal{NP}$-vollständigen Problemen zwischen Facetten, die

- wir in Polynomialzeit separieren können,
- wir kennen, wir aber nicht in Polynomialzeit separieren können,
- wir kennen, bei denen das Separationsproblem selbst wieder $\mathcal{NP}$-vollständig ist,

Typen von Facetten

Erinnerung (Satz von Minkowski)

Jedes Polytop P_I lässt sich durch ein Ungleichungssystem beschreiben. Ungleichungen, die zu einer Facette gehören, nennen wir selber Facetten.

Wir unterscheiden bei Polytopen P_I von $\mathcal{NP}$-vollständigen Problemen zwischen Facetten, die

- wir in Polynomialzeit separieren können,
- wir kennen, wir aber nicht in Polynomialzeit separieren können,
- wir kennen, bei denen das Separationsproblem selbst wieder $\mathcal{NP}$-vollständig ist,
- wir nicht kennen, bzw. nicht in Polynomialzeit verifizieren können.

Outline

Schnittebenen

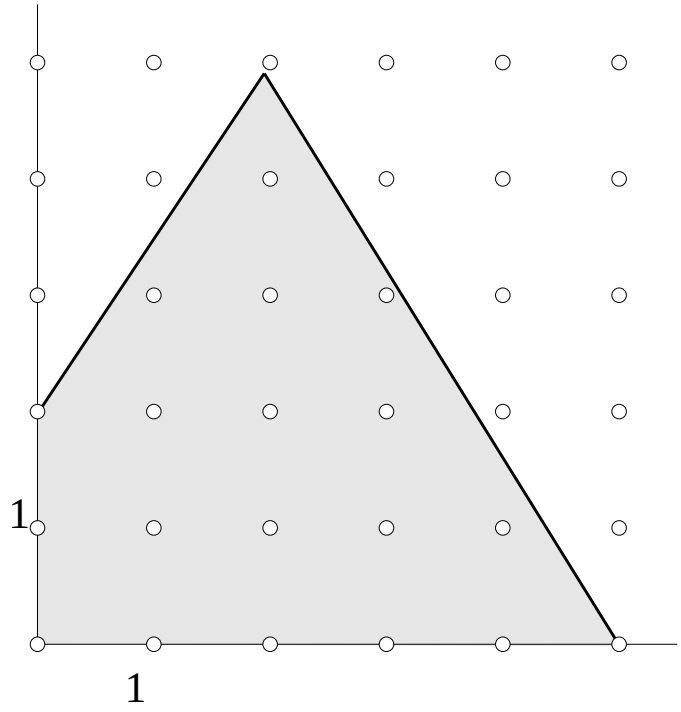
Schnittebenenverfahren kombiniert mit Branch & Bound sind die erfolgreichsten Methoden, IPs zu lösen.

Die Idee von Schnittebenenverfahren ist die Folgende:

Schnittebenen

Schnittebenenverfahren kombiniert mit Branch & Bound sind die erfolgreichsten Methoden, IPs zu lösen.

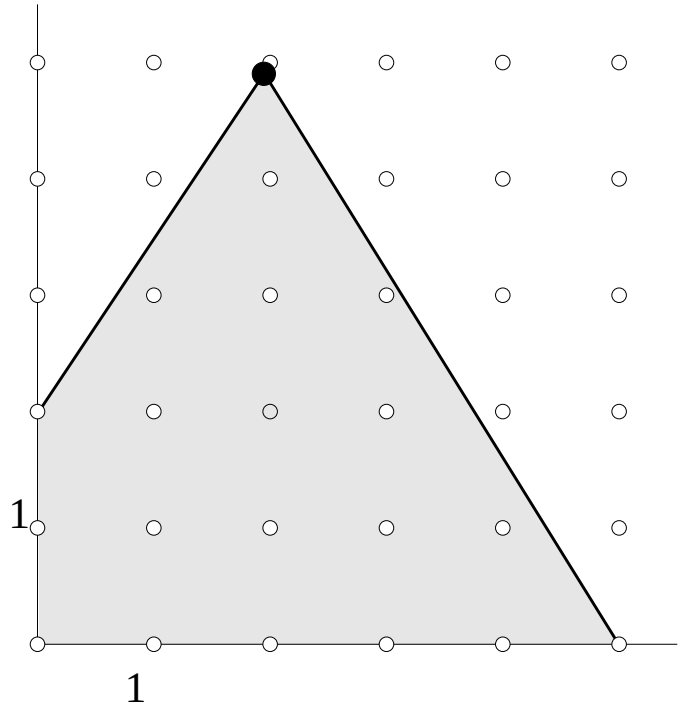
Die Idee von Schnittebenenverfahren ist die Folgende: Ausgehend von einer Modellierung P_1 des Problems



Schnittebenen

Schnittebenenverfahren kombiniert mit Branch & Bound sind die erfolgreichsten Methoden, IPs zu lösen.

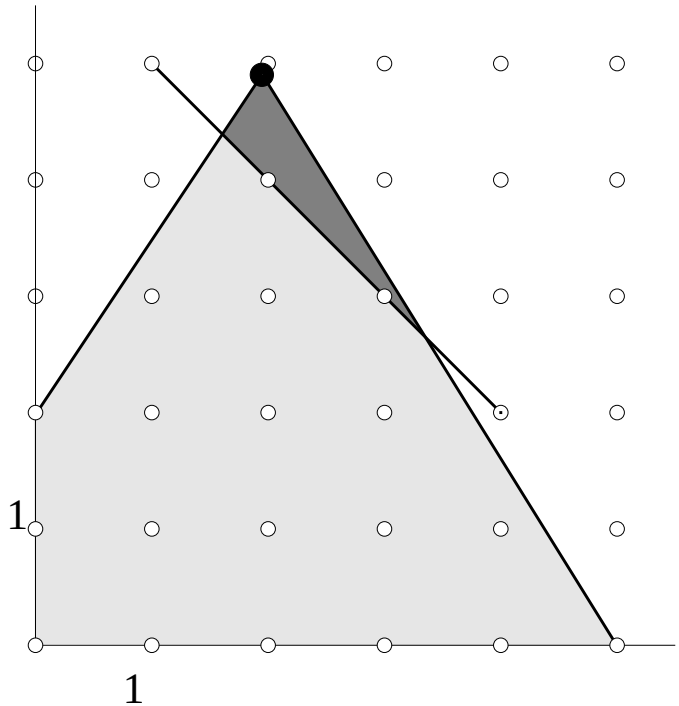
Die Idee von Schnittebenenverfahren ist die Folgende: Ausgehend von einer Modellierung P_1 des Problems mit einer fraktionalen Optimallösung x_1,



Schnittebenen

Schnittebenenverfahren kombiniert mit Branch & Bound sind die erfolgreichsten Methoden, IPs zu lösen.

Die Idee von Schnittebenenverfahren ist die Folgende: Ausgehend von einer Modellierung P_1 des Problems mit einer fraktionalen Optimallösung x_1, finde Ungleichung $ax \leq b$, die für $x \in P_I$ gültig ist mit $ax_1 > b$. Füge diese zum Modell hinzu und erhalte neues Modell P_2.



Generisches Schnittebenenverfahren

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

Generisches Schnittebenenverfahren

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

Ausgabe: $x^* \in \mathbb{Z}^n$ mit $c^\top x^* = \max\{c^\top x \mid Ax \leq b, x \in \mathbb{Z}^n_+\}$.

Generisches Schnittebenenverfahren

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

Ausgabe: $x^* \in \mathbb{Z}^n$ mit $c^\top x^* = \max\{c^\top x \mid Ax \leq b, x \in \mathbb{Z}^n_+\}$.

Verfahren: Sei $P = P(A, b)$. Iteriere

Generisches Schnittebenenverfahren

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

Ausgabe: $x^* \in \mathbb{Z}^n$ mit $c^\top x^* = \max\{c^\top x \mid Ax \leq b, x \in \mathbb{Z}^n_+\}$.

Verfahren: Sei $P = P(A, b)$. Iteriere

- Löse das lineare Programm $x^* := \max_{x \in P} c^\top x$.

Generisches Schnittebenenverfahren

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

Ausgabe: $x^* \in \mathbb{Z}^n$ mit $c^\top x^* = \max\{c^\top x \mid Ax \leq b, x \in \mathbb{Z}^n_+\}$.

Verfahren: Sei $P = P(A, b)$. Iteriere

- Löse das lineare Programm $x^* := \max_{x \in P} c^\top x$.
- Falls $x^* \in \mathbb{Z}^n$. Stop, gib x^* aus.

Generisches Schnittebenenverfahren

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

Ausgabe: $x^* \in \mathbb{Z}^n$ mit $c^\top x^* = \max\{c^\top x \mid Ax \leq b, x \in \mathbb{Z}^n_+\}$.

Verfahren: Sei $P = P(A, b)$. Iteriere

- Löse das lineare Programm $x^* := \max_{x \in P} c^\top x$.
- Falls $x^* \in \mathbb{Z}^n$. Stop, gib x^* aus.
- Falls $x^* \notin \mathbb{Z}^n$, löse das Separationsproblem und finde neue Ungleichung $a^\top x \leq \beta$.

Generisches Schnittebenenverfahren

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

Ausgabe: $x^* \in \mathbb{Z}^n$ mit $c^\top x^* = \max\{c^\top x \mid Ax \leq b, x \in \mathbb{Z}^n_+\}$.

Verfahren: Sei $P = P(A, b)$. Iteriere

- Löse das lineare Programm $x^* := \max_{x \in P} c^\top x$.
- Falls $x^* \in \mathbb{Z}^n$. Stop, gib x^* aus.
- Falls $x^* \notin \mathbb{Z}^n$, löse das Separationsproblem und finde neue Ungleichung $a^\top x \leq \beta$.
- Setze $P := P \cap \{x \in \mathbb{R}^n \mid a^\top x \leq \beta\}$ und iteriere weiter.

In diesem speziellen Falle ist das Separationsproblem stets lösbar, z.B. mittels Chvátal-Gomory-Schnitten.

Chvátal-Gomory-Schnitte

Satz

Ist $u \geq 0$ und $Ax \leq b$ gültig für P_I so auch

$$\sum_{j=1}^{m} \lfloor u^\top A_{\cdot j} \rfloor x_j \leq \lfloor u^\top b \rfloor. \tag{1}$$

Chvátal-Gomory-Schnitte

Satz

Ist $u \geq 0$ und $Ax \leq b$ gültig für P_I so auch

$$\sum_{j=1}^{m} \lfloor u^\top A_{\cdot j} \rfloor x_j \leq \lfloor u^\top b \rfloor. \tag{1}$$

Tatsächlich erhalten wir auf „diese Weise“ auf Dauer alle gültigen Ungleichungen.

Beispiel für einen Chvátal-Gomory-Schnitt

$$
\begin{array}{lrcrcl}
\max & x_1 & - & x_2 & & \\
\text{unter} & 7x_1 & - & 2x_2 & \leq & 14 \\
 & & & x_2 & \leq & 3 \\
 & 2x_1 & - & 2x_2 & \leq & 3 \\
 & & & & & x_1, x_2 \in \mathbb{Z}_+
\end{array}
$$

wir lösen das Lineare Programm

Beispiel für einen Chvátal-Gomory-Schnitt

$$
\begin{array}{lrcrcl}
\max & x_1 & - & x_2 & & \\
\text{unter} & 7x_1 & - & 2x_2 & \leq & 14 \\
 & & & x_2 & \leq & 3 \\
 & 2x_1 & - & 2x_2 & \leq & 3 \\
 & & & & & x_1, x_2 \in \mathbb{Z}_+
\end{array}
$$

wir lösen das Lineare Programm

1	−1	0	0	0	0
7	−2	1	0	0	14
0	1	0	1	0	3
$\boxed{2}$	−2	0	0	1	3

0	0	0	0	−0.5	−1.5
0	5	1	0	−3.5	3.5
0	1	0	1	0	3
1	−1	0	0	0.5	1.5

Beispiel für einen Chvátal-Gomory-Schnitt

$$
\begin{array}{lrcrcl}
\max & x_1 & - & x_2 & & \\
\text{unter} & 7x_1 & - & 2x_2 & \leq & 14 \\
 & & & x_2 & \leq & 3 \\
 & 2x_1 & - & 2x_2 & \leq & 3 \\
 & & & & & x_1, x_2 \in \mathbb{Z}_+
\end{array}
$$

wir lösen das Lineare Programm

1	−1	0	0	0	0
7	−2	1	0	0	14
0	1	0	1	0	3
$\boxed{2}$	−2	0	0	1	3

0	0	0	0	−0.5	−1.5
0	5	1	0	−3.5	3.5
0	1	0	1	0	3
1	−1	0	0	0.5	1.5

aus der letzten Zeile schließen wir nun $x_1 - x_2 \leq 1$

Beispiel für einen Chvátal-Gomory-Schnitt

$$
\begin{array}{lrcrcl}
\max & x_1 & - & x_2 & & \\
\text{unter} & 7x_1 & - & 2x_2 & \leq & 14 \\
 & & & x_2 & \leq & 3 \\
 & 2x_1 & - & 2x_2 & \leq & 3 \\
 & & & & & x_1, x_2 \in \mathbb{Z}_+
\end{array}
$$

wir lösen das Lineare Programm

1	−1	0	0	0	0
7	−2	1	0	0	14
0	1	0	1	0	3
$\boxed{2}$	−2	0	0	1	3

0	0	0	0	−0.5	−1.5
0	5	1	0	−3.5	3.5
0	1	0	1	0	3
1	−1	0	0	0.5	1.5

aus der letzten Zeile schließen wir nun $x_1 - x_2 \leq 1 \Rightarrow y_3 \geq 1$.

Beispiel für einen Chvátal-Gomory-Schnitt

1	−1	0	0	0	0
7	−2	1	0	0	14
0	1	0	1	0	3
2	−2	0	0	1	3

0	0	0	0	−0.5	−1.5
0	5	1	0	−3.5	3.5
0	1	0	1	0	3
1	−1	0	0	0.5	1.5

aus der letzten Zeile schließen wir nun $x_1 - x_2 \leq 1 \Rightarrow y_3 \geq 1$.

0	0	0	0	−0.5	0	−1.5
0	5	1	0	−3.5	0	3.5
0	1	0	1	0	0	3
1	−1	0	0	0.5	0	1.5
0	0	0	0	-1	1	−1

0	0	0	0	0	−0.5	−1
0	5	1	0	0	−3.5	7
0	1	0	1	0	0	3
1	−1	0	0	0	0.5	1
0	0	0	0	1	−1	1

Outline

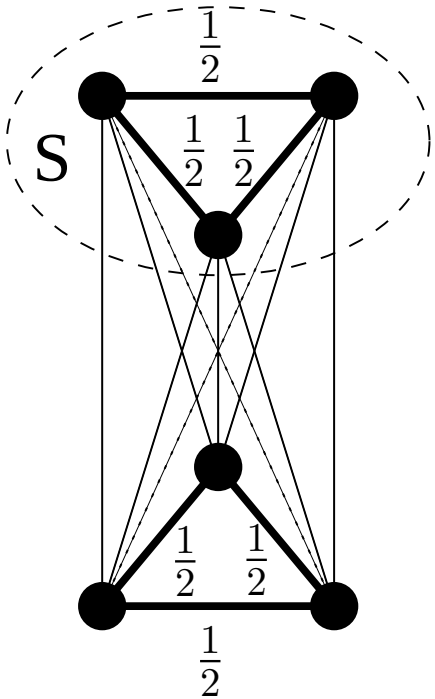
Schnittebenen

Obwohl Gomory-Chvátal-Schnitte einen endlichen Algorithmus liefern, werden sie in der Praxis „nur im Notfall“ eingesetzt.

Schnittebenen

Obwohl Gomory-Chvátal-Schnitte einen endlichen Algorithmus liefern, werden sie in der Praxis „nur im Notfall“ eingesetzt.

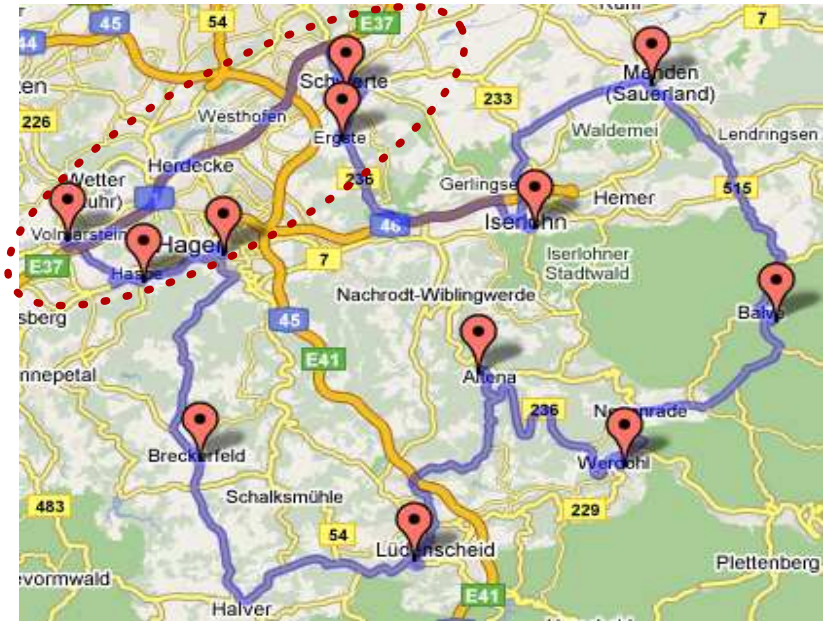
Wichtiger sind zunächst kombinatorisch motiverte Schnitte. Kennengelernt haben wir einen solchen Typ bei Matching.



Schnittebenen

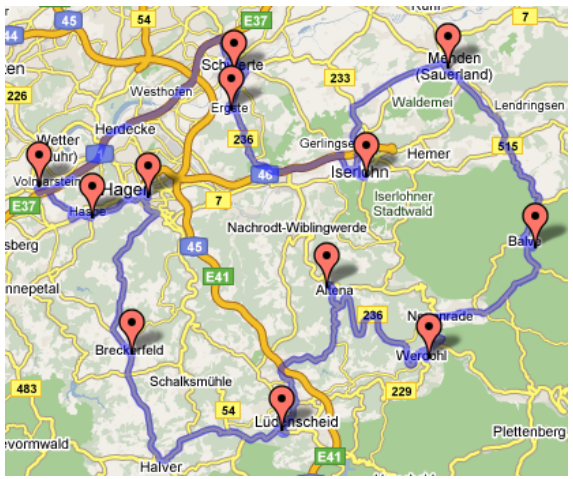
Obwohl Gomory-Chvátal-Schnitte einen endlichen Algorithmus liefern, werden sie in der Praxis „nur im Notfall“ eingesetzt.

Wichtiger sind zunächst kombinatorisch motiverte Schnitte. Kennengelernt haben wir einen solchen Typ bei Matching. Analoges gilt für das Rundreiseproblem.



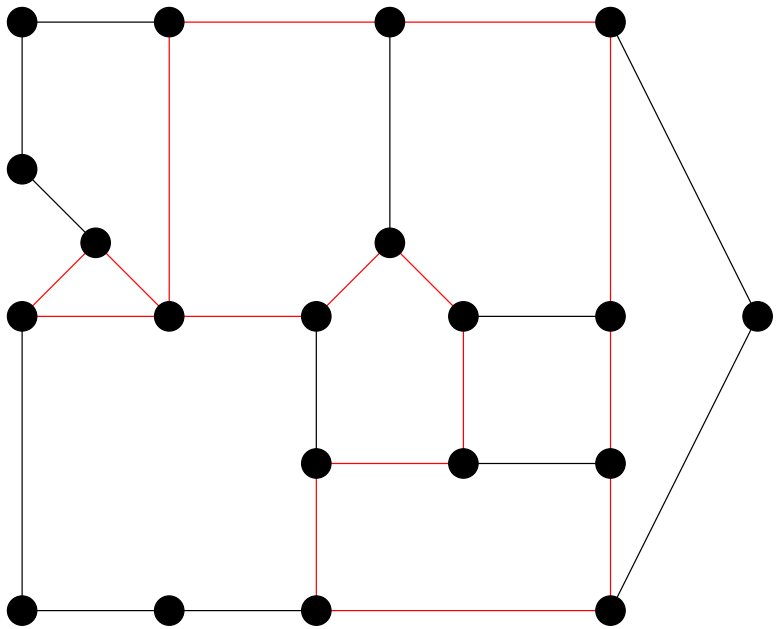
Das symmetrische TSP

$$
\begin{aligned}
\min \quad & \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} & & & \\
& \sum_{(ij)\in\delta(v)} x_{ij} & = & 2 & \forall v \in V \\
& \sum_{(ij)\in\delta(S)} & \geq & 2 & \forall \emptyset \neq S \neq V \\
& x_{ij} & \in & \{0,1\} &
\end{aligned}
$$



Dantzig, Fulkerson und Johnson 1954

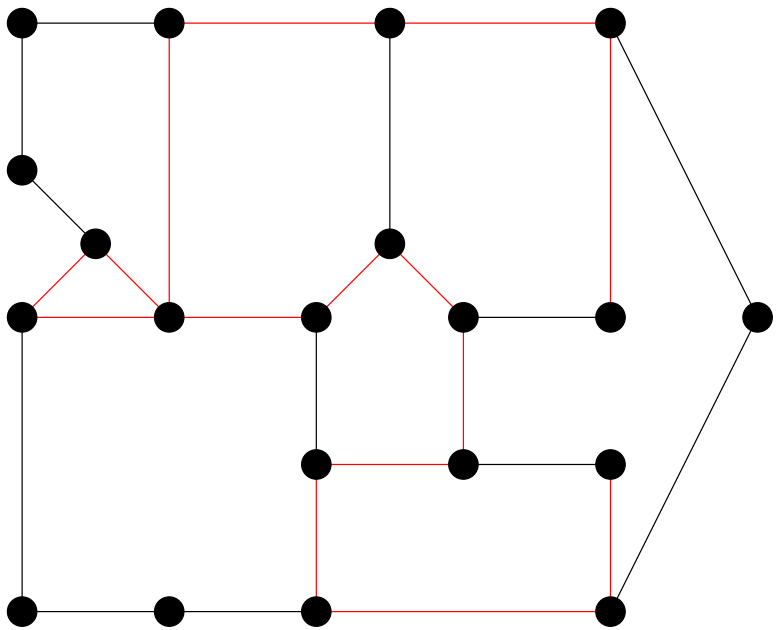
Beispiel einer Ungleichung auf einem Graphen mit 20 Knoten, die wir nicht in Polynomialzeit verifizieren können.



Dantzig, Fulkerson und Johnson 1954

Beispiel einer Ungleichung auf einem Graphen mit 20 Knoten, die wir nicht in Polynomialzeit verifizieren können.

$$\sum_{\text{rote und schwarze Kanten}} x_{ij} \leq 19.$$

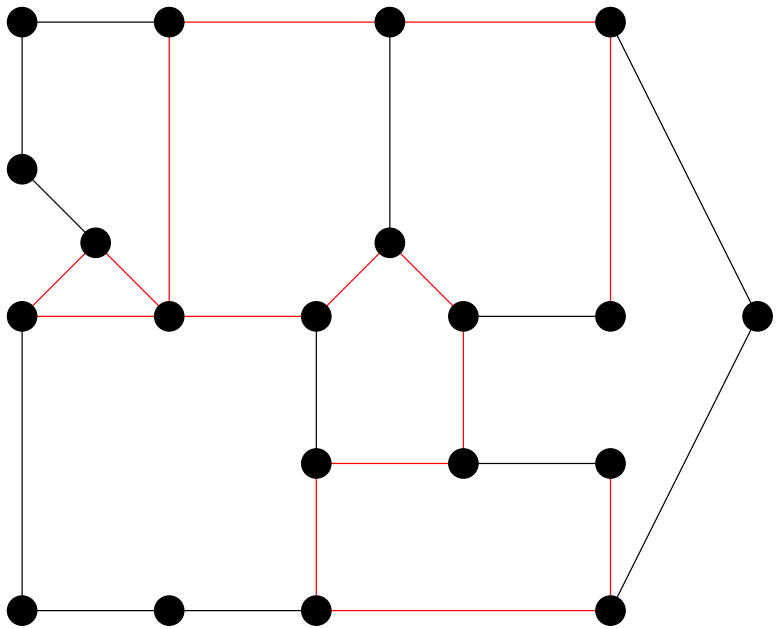


Dantzig, Fulkerson und Johnson 1954

Beispiel einer Ungleichung auf einem Graphen mit 20 Knoten, die wir nicht in Polynomialzeit verifizieren können.

$$\sum_{\text{rote und schwarze Kanten}} x_{ij} \leq 19.$$

Der Graph der roten und schwarzen Kanten hat keinen Hamiltonkreis.

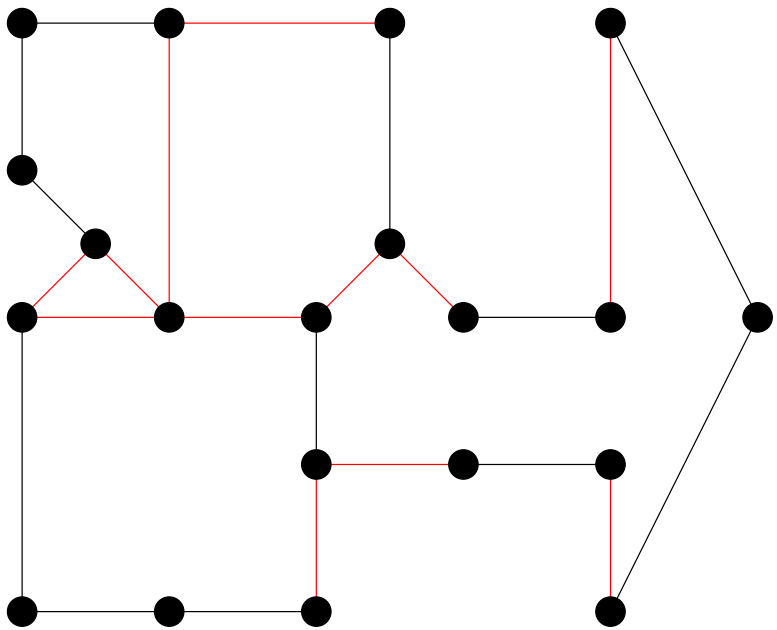


Dantzig, Fulkerson und Johnson 1954

Beispiel einer Ungleichung auf einem Graphen mit 20 Knoten, die wir nicht in Polynomialzeit verifizieren können.

$$\sum_{\text{rote und schwarze Kanten}} x_{ij} \leq 19.$$

Der Graph der roten und schwarzen Kanten hat keinen Hamiltonkreis.

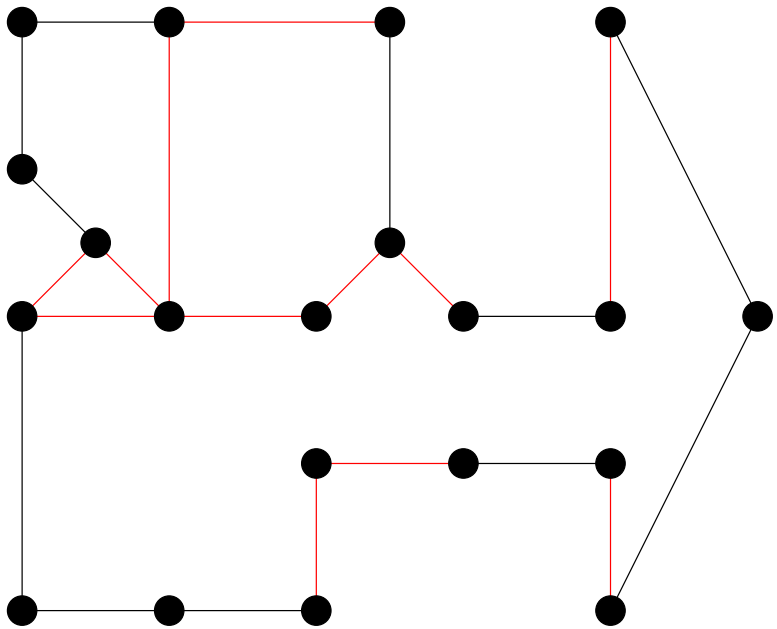


Dantzig, Fulkerson und Johnson 1954

Beispiel einer Ungleichung auf einem Graphen mit 20 Knoten, die wir nicht in Polynomialzeit verifizieren können.

$$\sum_{\text{rote und schwarze Kanten}} x_{ij} \leq 19.$$

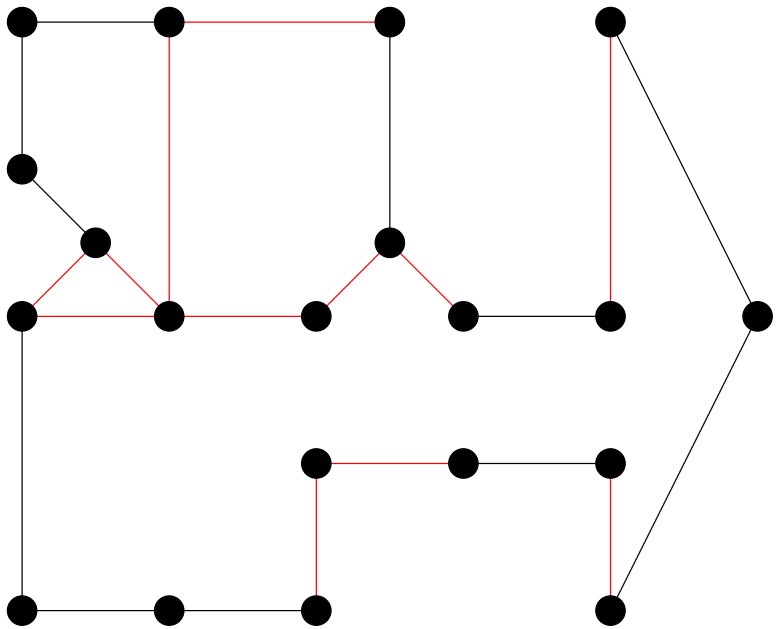
Der Graph der roten und schwarzen Kanten hat keinen Hamiltonkreis.



Dantzig, Fulkerson und Johnson 1954

Beispiel einer Ungleichung auf einem Graphen mit 20 Knoten, die wir nicht in Polynomialzeit verifizieren können.

$$\sum_{\text{rote und schwarze Kanten}} x_{ij} \leq 19.$$

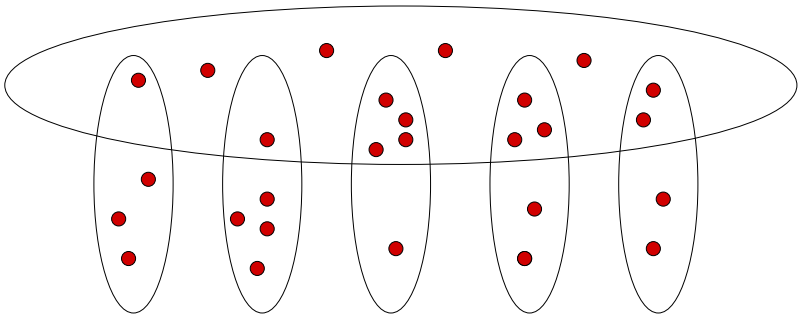


Der Graph der roten und schwarzen Kanten hat keinen Hamiltonkreis. Das Hamiltonkreisproblem ist $\mathcal{NP}$-vollständig.

Kammungleichungen (Grötschel & Padberg 1979)

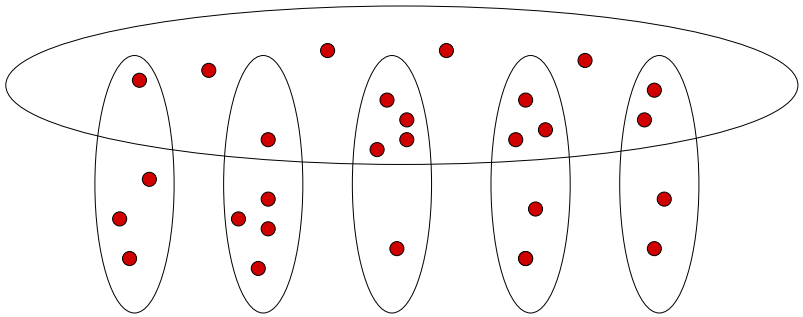
Kammungleichungen (Grötschel & Padberg 1979)

Ein Kamm besteht aus einer Knotenmenge H (wie handle) und einer ungeraden Anzahl paarweise disjunkter Zähne $T_1, \ldots, T_p$, $p \geq 3$ mit $\forall j : H \cap T_j \neq \emptyset \neq T_j \cap H$. Dann gilt die Kammungleichung



Kammungleichungen (Grötschel & Padberg 1979)

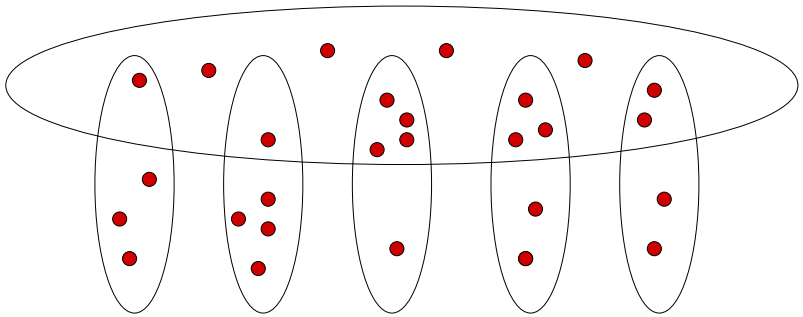
Ein Kamm besteht aus einer Knotenmenge H (wie handle) und einer ungeraden Anzahl paarweise disjunkter Zähne $T_1, \ldots, T_p$, $p \geq 3$ mit $\forall j : H \cap T_j \neq \emptyset \neq T_j \cap H$. Dann gilt die Kammungleichung



$$\sum_{(ij)\in H} x_{ij} + \sum_{k=1}^{p} \sum_{(ij)\in T_k} x_{ij} \leq |H| + \sum_{k=1}^{p} |T_k| - \lceil \frac{3p}{2} \rceil.$$

Kammungleichungen (Grötschel & Padberg 1979)

Ein Kamm besteht aus einer Knotenmenge H (wie handle) und einer ungeraden Anzahl paarweise disjunkter Zähne $T_1, \ldots, T_p$, $p \geq 3$ mit $\forall j : H \cap T_j \neq \emptyset \neq T_j \cap H$. Dann gilt die Kammungleichung



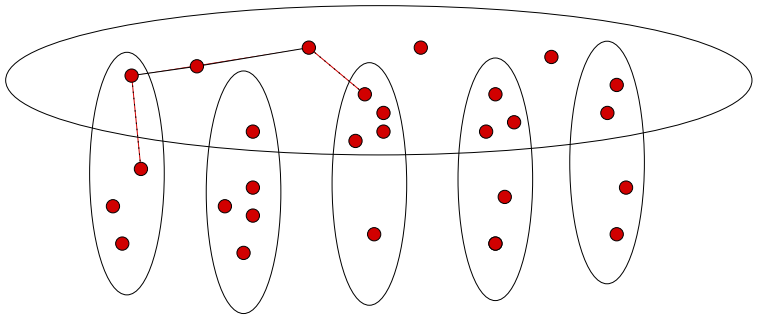
$$\sum_{(ij)\in H} x_{ij} + \sum_{k=1}^{p} \sum_{(ij)\in T_k} x_{ij} \leq |H| + \sum_{k=1}^{p} |T_k| - \lceil \frac{3p}{2} \rceil.$$

Beweis

Wir müssen zeigen, dass Rundtouren die Ungleichung stets erfüllen.

Kammungleichungen (Grötschel & Padberg 1979)

Ein Kamm besteht aus einer Knotenmenge H (wie handle) und einer ungeraden Anzahl paarweise disjunkter Zähne $T_1, \ldots, T_p$, $p \geq 3$ mit $\forall j : H \cap T_j \neq \emptyset \neq T_j \cap H$. Dann gilt die Kammungleichung



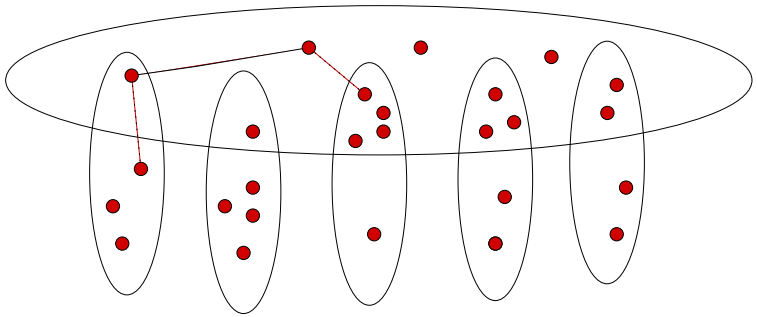
$$\sum_{(ij)\in H} x_{ij} + \sum_{k=1}^{p} \sum_{(ij)\in T_k} x_{ij} \leq |H| + \sum_{k=1}^{p} |T_k| - \lceil \frac{3p}{2} \rceil.$$

Beweis

O.E. liegen keine Knoten in $H \setminus \bigcup T_j$.

Kammungleichungen (Grötschel & Padberg 1979)

Ein Kamm besteht aus einer Knotenmenge H (wie handle) und einer ungeraden Anzahl paarweise disjunkter Zähne $T_1, \ldots, T_p$, $p \geq 3$ mit $\forall j : H \cap T_j \neq \emptyset \neq T_j \cap H$. Dann gilt die Kammungleichung



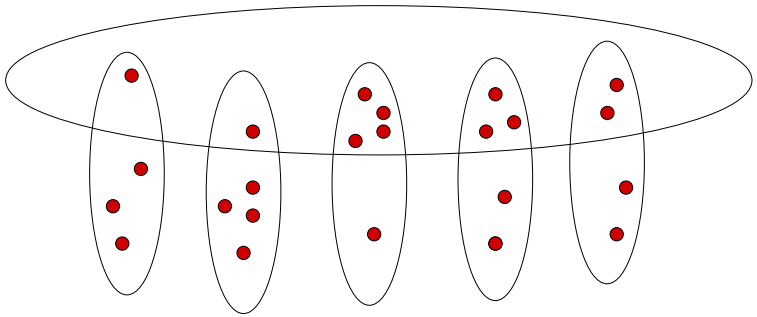
$$\sum_{(ij)\in H} x_{ij} + \sum_{k=1}^{p} \sum_{(ij)\in T_k} x_{ij} \leq |H| + \sum_{k=1}^{p} |T_k| - \lceil \frac{3p}{2} \rceil.$$

Beweis

O.E. liegen keine Knoten in $H \setminus \bigcup T_j$.

Kammungleichungen (Grötschel & Padberg 1979)

Ein Kamm besteht aus einer Knotenmenge H (wie handle) und einer ungeraden Anzahl paarweise disjunkter Zähne $T_1, \ldots, T_p$, $p \geq 3$ mit $\forall j : H \cap T_j \neq \emptyset \neq T_j \cap H$. Dann gilt die Kammungleichung



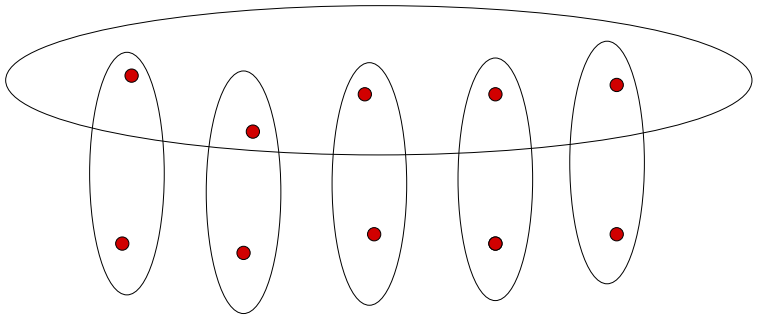
$$\sum_{(ij)\in H} x_{ij} + \sum_{k=1}^{p} \sum_{(ij)\in T_k} x_{ij} \leq |H| + \sum_{k=1}^{p} |T_k| - \lceil \frac{3p}{2} \rceil.$$

Beweis

O.E. liegen keine Knoten in $H \setminus \bigcup T_j$.

Kammungleichungen (Grötschel & Padberg 1979)

Ein Kamm besteht aus einer Knotenmenge H (wie handle) und einer ungeraden Anzahl paarweise disjunkter Zähne $T_1, \ldots, T_p$, $p \geq 3$ mit $\forall j : H \cap T_j \neq \emptyset \neq T_j \cap H$. Dann gilt die Kammungleichung



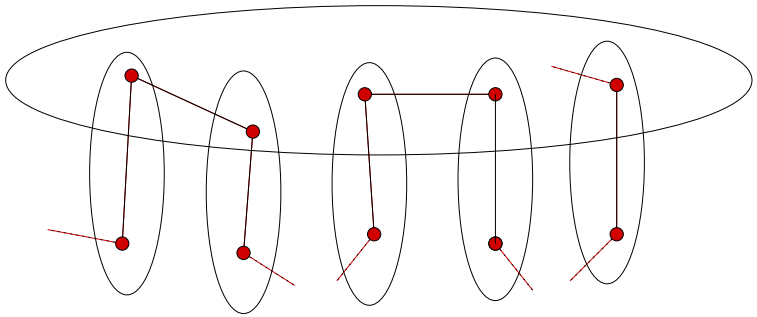
$$\sum_{(ij)\in H} x_{ij} + \sum_{k=1}^{p} \sum_{(ij)\in T_k} x_{ij} \leq |H| + \sum_{k=1}^{p} |T_k| - \lceil \frac{3p}{2} \rceil.$$

Beweis

O.E. liegen genau zwei Knoten in jedem T_j.

Kammungleichungen (Grötschel & Padberg 1979)

Ein Kamm besteht aus einer Knotenmenge H (wie handle) und einer ungeraden Anzahl paarweise disjunkter Zähne $T_1, \ldots, T_p$, $p \geq 3$ mit $\forall j : H \cap T_j \neq \emptyset \neq T_j \cap H$. Dann gilt die Kammungleichung



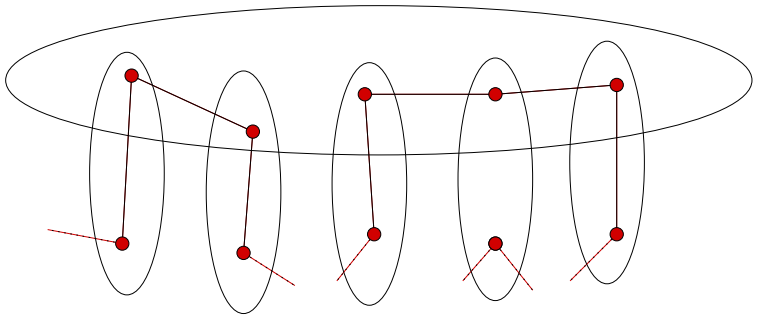
$$\sum_{(ij)\in H} x_{ij} + \sum_{k=1}^{p} \sum_{(ij)\in T_k} x_{ij} \leq |H| + \sum_{k=1}^{p} |T_k| - \lceil \frac{3p}{2} \rceil.$$

Beweis

Die maximale Anzahl Kanten, die links gezählt werden ist $p + \lfloor \frac{p}{2} \rfloor$.

Kammungleichungen (Grötschel & Padberg 1979)

Ein Kamm besteht aus einer Knotenmenge H (wie handle) und einer ungeraden Anzahl paarweise disjunkter Zähne $T_1, \ldots, T_p$, $p \geq 3$ mit $\forall j : H \cap T_j \neq \emptyset \neq T_j \cap H$. Dann gilt die Kammungleichung



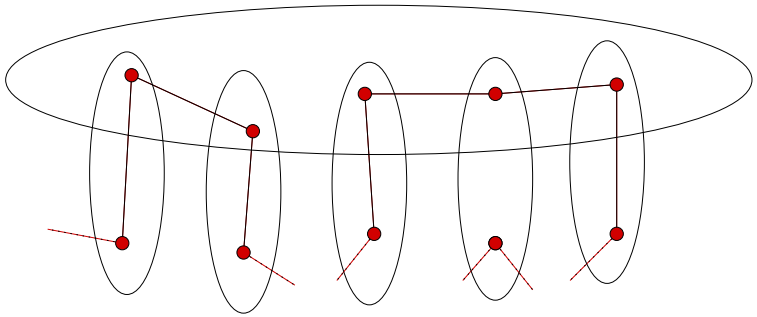
$$\sum_{(ij)\in H} x_{ij} + \sum_{k=1}^{p} \sum_{(ij)\in T_k} x_{ij} \leq |H| + \sum_{k=1}^{p} |T_k| - \lceil \frac{3p}{2} \rceil.$$

Beweis

Die maximale Anzahl Kanten, die links gezählt werden ist $p + \lfloor \frac{p}{2} \rfloor$.

Kammungleichungen (Grötschel & Padberg 1979)

Ein Kamm besteht aus einer Knotenmenge H (wie handle) und einer ungeraden Anzahl paarweise disjunkter Zähne $T_1, \ldots, T_p$, $p \geq 3$ mit $\forall j : H \cap T_j \neq \emptyset \neq T_j \cap H$. Dann gilt die Kammungleichung



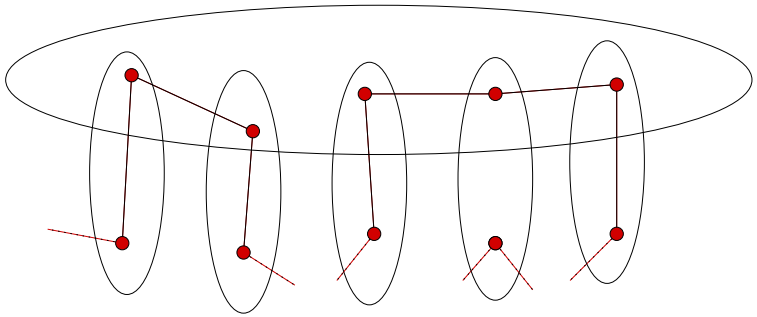
$$\sum_{(ij)\in H} x_{ij} + \sum_{k=1}^{p} \sum_{(ij)\in T_k} x_{ij} \leq |H| + \sum_{k=1}^{p} |T_k| - \lceil \frac{3p}{2} \rceil.$$

Beweis

Die maximale Anzahl Kanten, die links gezählt werden ist $p + \lfloor \frac{p}{2} \rfloor$.
Rechts werden $3p$ Knoten gezählt.

Kammungleichungen (Grötschel & Padberg 1979)

Ein Kamm besteht aus einer Knotenmenge H (wie handle) und einer ungeraden Anzahl paarweise disjunkter Zähne $T_1, \ldots, T_p$, $p \geq 3$ mit $\forall j : H \cap T_j \neq \emptyset \neq T_j \cap H$. Dann gilt die Kammungleichung



$$\sum_{(ij)\in H} x_{ij} + \sum_{k=1}^{p} \sum_{(ij)\in T_k} x_{ij} \leq |H| + \sum_{k=1}^{p} |T_k| - \lceil \frac{3p}{2} \rceil.$$

Beweis

$$3p - \lceil \frac{3p}{2} \rceil = p + \lfloor \frac{p}{2} \rfloor.$$

Kammungleichungen (Grötschel & Padberg 1979)

Ein Kamm besteht aus einer Knotenmenge H (wie handle) und einer ungeraden Anzahl paarweise disjunkter Zähne $T_1, \ldots, T_p$, $p \geq 3$ mit $\forall j : H \cap T_j \neq \emptyset \neq T_j \cap H$. Dann gilt die Kammungleichung

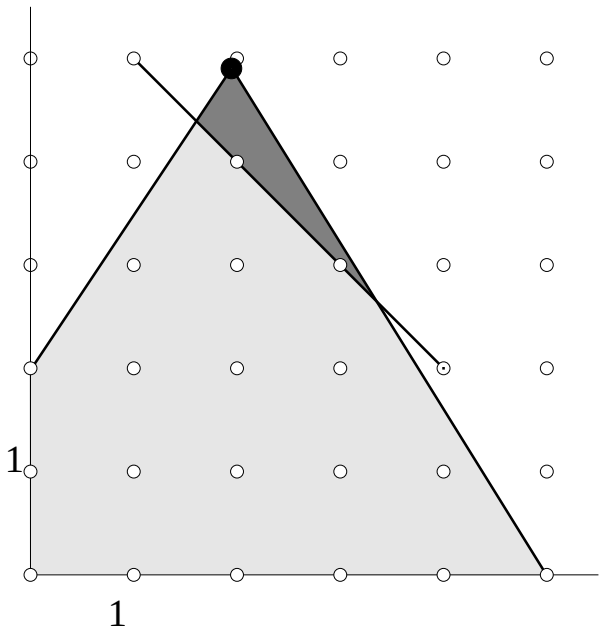
$$\sum_{(ij)\in H} x_{ij} + \sum_{k=1}^{p} \sum_{(ij)\in T_k} x_{ij} \leq |H| + \sum_{k=1}^{p} |T_k| - \lceil \frac{3p}{2} \rceil.$$

Die Komplexität der Separation von Kammungleichungen scheint im allgemeinen Fall noch offen zu sein.

Outline

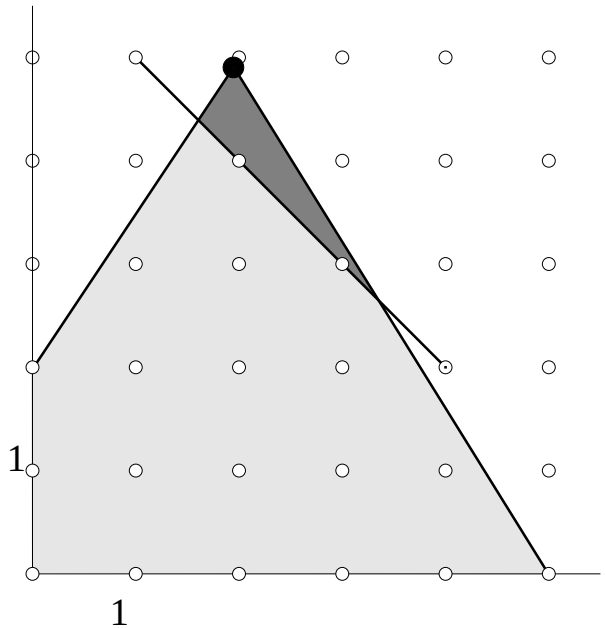
Branch and Cut

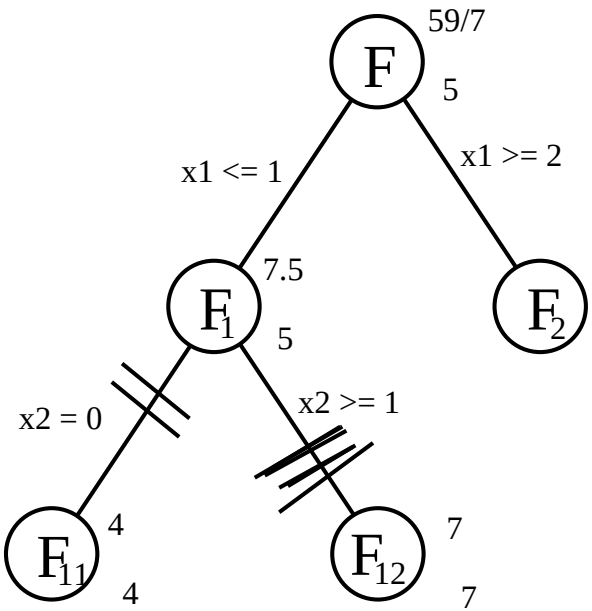
Die Schnittebenenverfahren kombiniert man mit Branch & Bound.



Branch and Cut

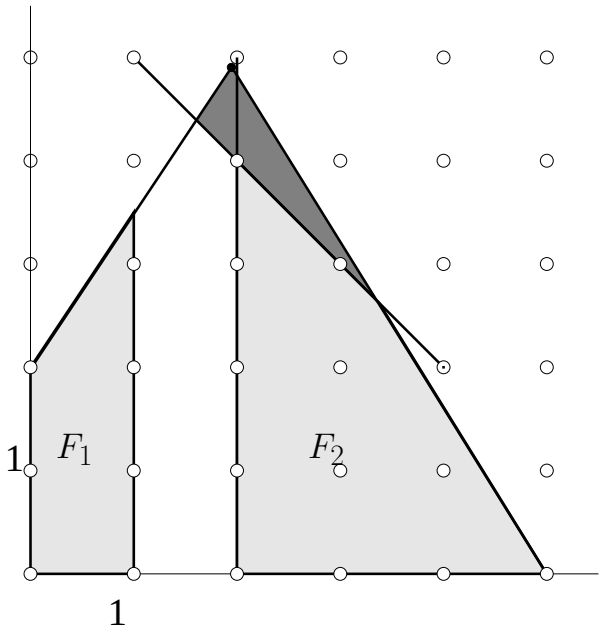
Die Schnittebenenverfahren kombiniert man mit Branch & Bound.

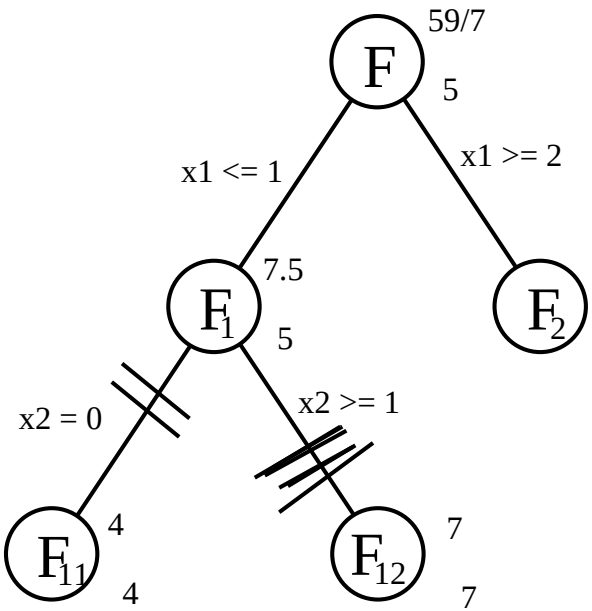




Branch and Cut

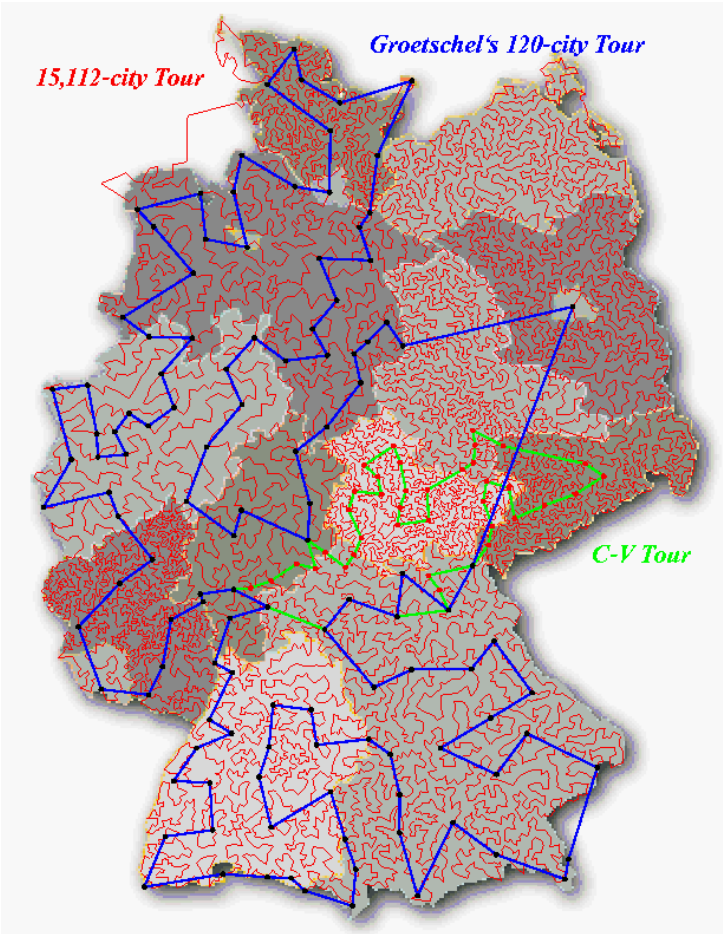
Die Schnittebenenverfahren kombiniert man mit Branch & Bound.





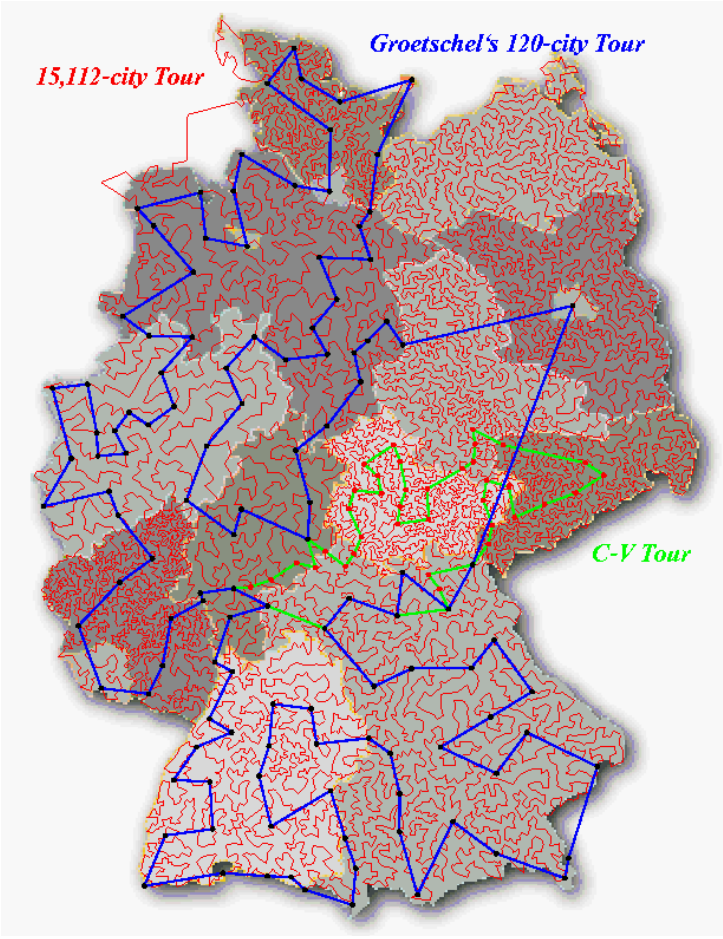
Das TSP als Referenzproblem

1832 Der Handlungsreisende - wie er sein soll und was er zu thun hat, um Auftraege zu erhalten und eines gluecklichen Erfolgs in seinen Geschaeften gewiss zu sein - Von einem alten Commis-Voyageur



Das TSP als Referenzproblem

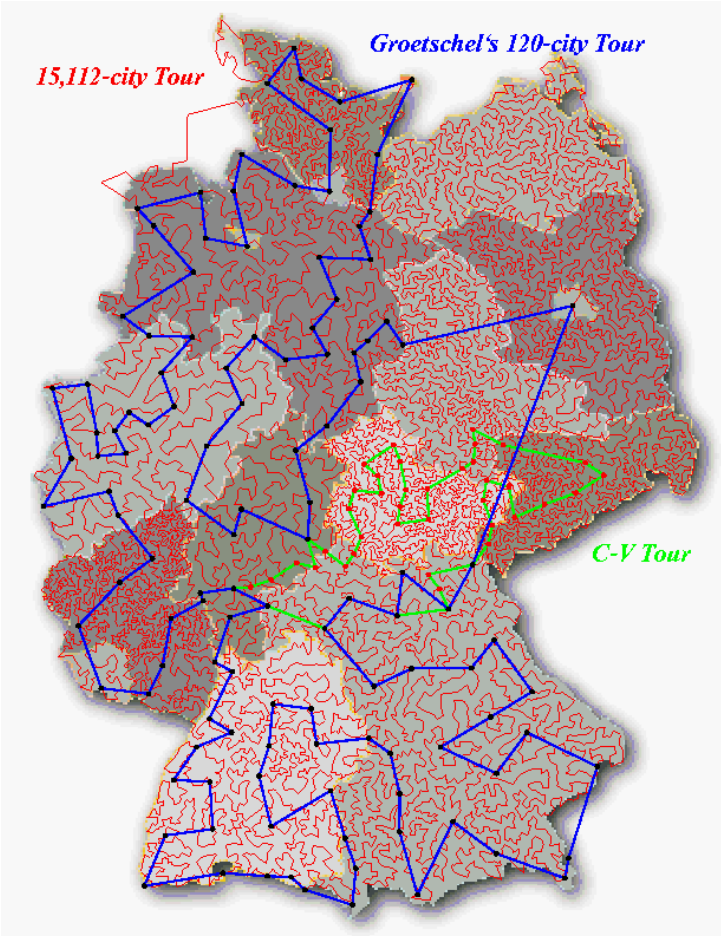
1954 Dantzig, Fulkerson & Johnson: 49 Städte in USA



Das TSP als Referenzproblem

1954 Dantzig, Fulkerson & Johnson: 49 Städte in USA

1977 Grötschel 120 Städte (Deutscher General Atlas)

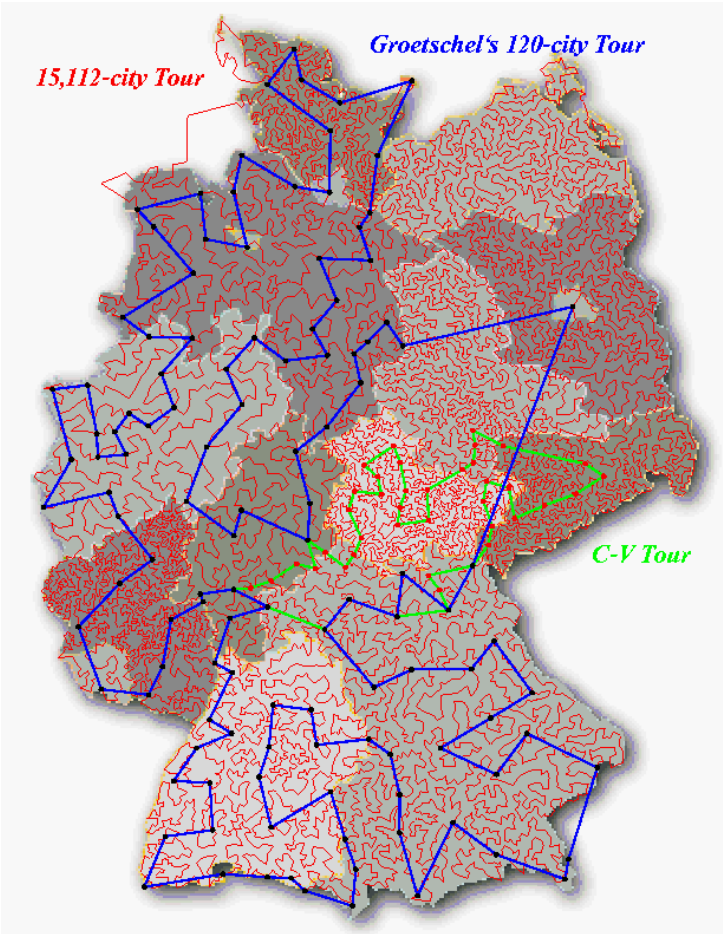


Das TSP als Referenzproblem

1954 Dantzig, Fulkerson & Johnson: 49 Städte in USA

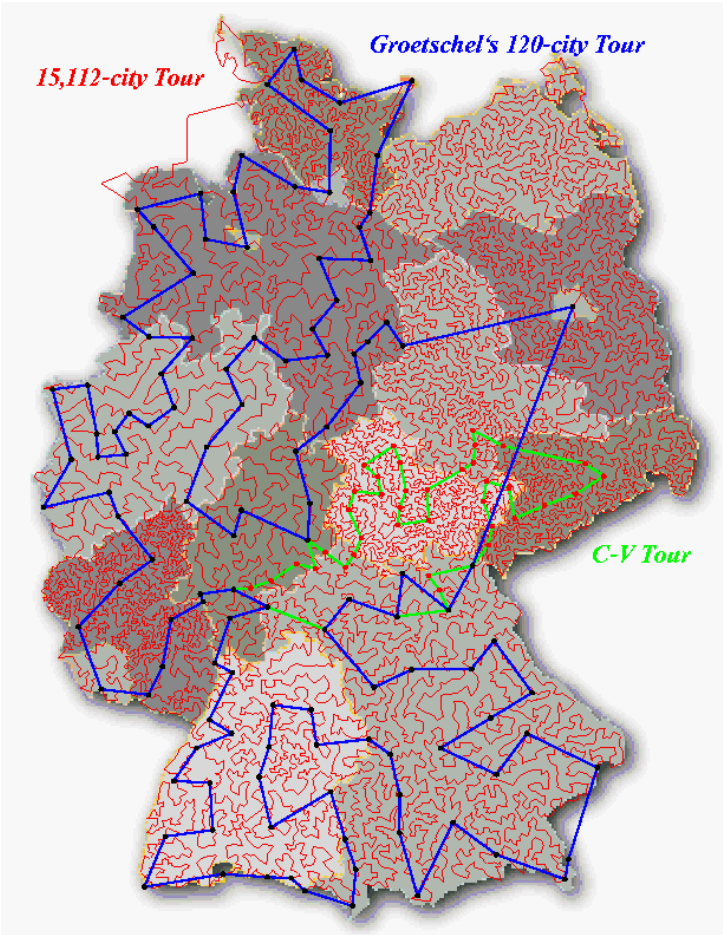
1977 Grötschel 120 Städte (Deutscher General Atlas)

2001 Applegate, Bixby, Chvátal, Cook (ABCC): 15112 Städte (22,6 Prozessorjahre auf 110 Alpha 500 MHz Prozessoren).



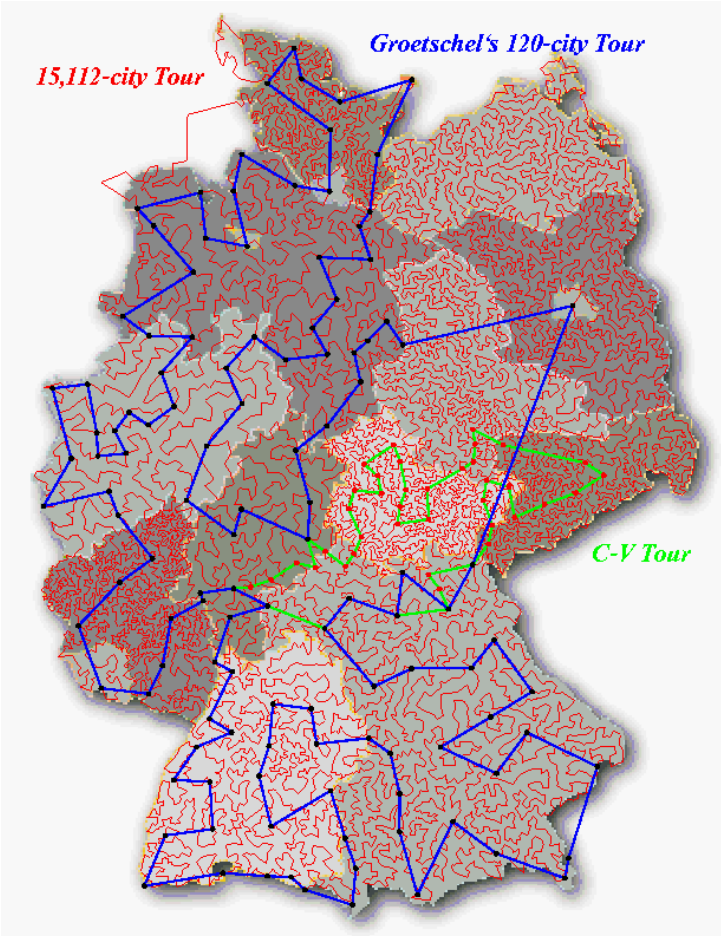
Das TSP als Referenzproblem

Alle jüngsten Rekorde wurden von ABCC mit dem Programm Concorde erzielt. Derzeitiger Rekord (2006) liegt bei 85,900 Städten in einem VLSI Problem.



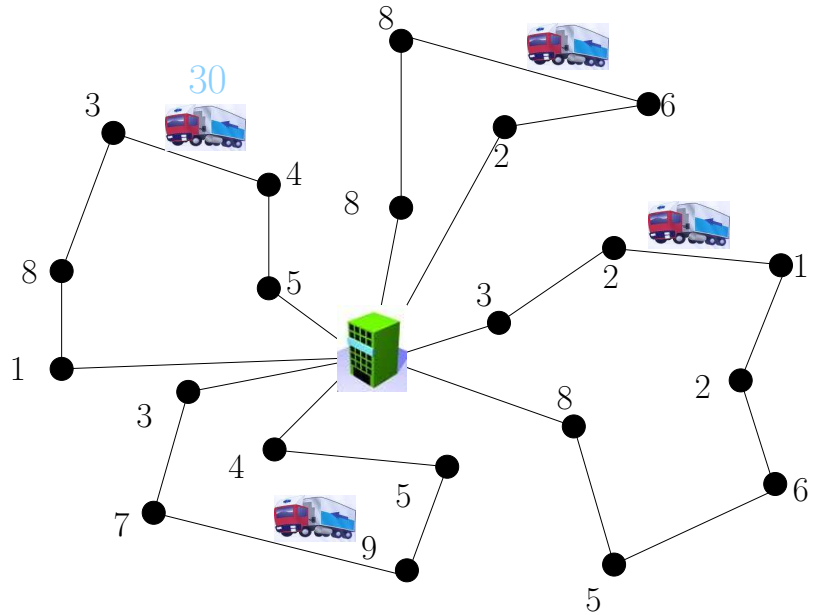
Das TSP als Referenzproblem

Alle jüngsten Rekorde wurden von ABCC mit dem Programm Concorde erzielt. Derzeitiger Rekord (2006) liegt bei 85,900 Städten in einem VLSI Problem. Damit sind alle Probleme aus der TSPLIB von Günter Reinelt (Heidelberg) optimal gelöst.

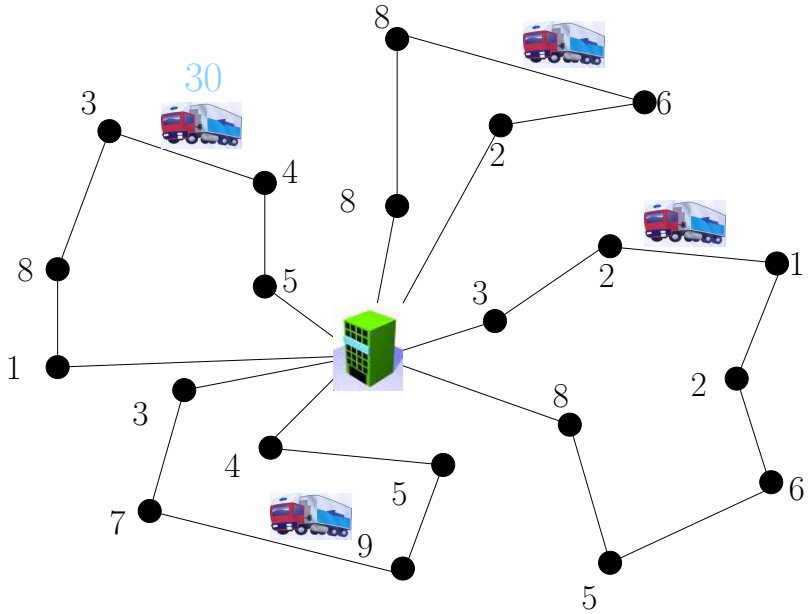


Jenseits des TSP

Während man beim TSP Probleme mit mehreren Tausend Knoten fast routinemäßig exakt lösen kann, gibt es beim CVRP Benchmark-Instanzen mit weniger als 100 Kunden, bei denen die Optimallösung nicht bekannt ist.



Jenseits des TSP



Hier wendet man auch einen zu den Schnittebenenverfahren dualen Ansatz an, bei dem man nicht neue Ungleichungen sondern neue Variablen generiert. Diese Verfahren nennt man Spaltengenerierungsverfahren.

Column Generation

Seien $T_1, \ldots, T_k$ die Menge aller zulässigen Touren die vom Depot ausgehen und die Kapazitätsbeschränkung berücksichtigen.

Column Generation

Seien $T_1, \ldots, T_k$ die Menge aller zulässigen Touren die vom Depot ausgehen und die Kapazitätsbeschränkung berücksichtigen. Sei $\chi(T_i) \in \{0,1\}^{V_0}$ der Inzidenzvektor, der anzeigt, welche Kunden von T_i bedient werden. Seien schließlich $c(T_i)$ die Kosten von Tour T_i.

Column Generation

Seien $T_1, \ldots, T_k$ die Menge aller zulässigen Touren die vom Depot ausgehen und die Kapazitätsbeschränkung berücksichtigen. Sei $\chi(T_i) \in \{0,1\}^{V_0}$ der Inzidenzvektor, der anzeigt, welche Kunden von T_i bedient werden. Seien schließlich $c(T_i)$ die Kosten von Tour T_i. Wir setzen $x_{T_i} = 1$, wenn wir Tour T_i fahren und $x_{T_i} = 0$ sonst.

Dann haben wir folgendes Boolsches Programm für das VRCP

Column Generation

Seien $T_1, \ldots, T_k$ die Menge aller zulässigen Touren die vom Depot ausgehen und die Kapazitätsbeschränkung berücksichtigen. Sei $\chi(T_i) \in \{0,1\}^{V_0}$ der Inzidenzvektor, der anzeigt, welche Kunden von T_i bedient werden. Seien schließlich $c(T_i)$ die Kosten von Tour T_i. Wir setzen $x_{T_i} = 1$, wenn wir Tour T_i fahren und $x_{T_i} = 0$ sonst.

Dann haben wir folgendes Boolsches Programm für das VRCP

$$\begin{array}{lrcl} \min & \sum_{i=1}^{k} c(T_i) x_{T_i} & & \\ & \sum_{i=1}^{k} \chi(T_i) x_{T_i} & \geq & 1 \\ & x_{T_i} & \in & \{0,1\} \end{array}$$

Column Generation II

Wir haben also nun ein Problem mit mehr als exponentiell vielen Variablen.

- Diese Variablen betrachtet man nicht alle, sondern man hält einen Pool von „aussichtsreichen Touren“ und löst das Problem mit diesen.

Column Generation II

Wir haben also nun ein Problem mit mehr als exponentiell vielen Variablen.

- Diese Variablen betrachtet man nicht alle, sondern man hält einen Pool von „aussichtsreichen Touren“ und löst das Problem mit diesen.
- Hat man ein solches Problem optimal gelöst, sucht man nach weiteren Touren, deren Variable positive reduzierte Kosten im Tableau hat. Dieses Problem nennt man das Pricing-Problem.

Column Generation II

Wir haben also nun ein Problem mit mehr als exponentiell vielen Variablen.

- Diese Variablen betrachtet man nicht alle, sondern man hält einen Pool von „aussichtsreichen Touren“ und löst das Problem mit diesen.
- Hat man ein solches Problem optimal gelöst, sucht man nach weiteren Touren, deren Variable positive reduzierte Kosten im Tableau hat. Dieses Problem nennt man das Pricing-Problem.
- Gibt es keine solche Tour ist das Problem optimal gelöst.

Column Generation II

Wir haben also nun ein Problem mit mehr als exponentiell vielen Variablen.

- Diese Variablen betrachtet man nicht alle, sondern man hält einen Pool von „aussichtsreichen Touren“ und löst das Problem mit diesen.
- Hat man ein solches Problem optimal gelöst, sucht man nach weiteren Touren, deren Variable positive reduzierte Kosten im Tableau hat. Dieses Problem nennt man das Pricing-Problem.
- Gibt es keine solche Tour ist das Problem optimal gelöst.
- Um Ganzzahligkeit zu erreichen, kombiniert man auch dies mit Branch& Bound und nennt dies Branch & Price.

ColumnGeneration III

Allgemein verwendet man Column Generation insbesondere bei Problemen, die ein Überdeckungsteilproblem haben.

Column Generation alleine scheint beim kapazitierten Tourenplanungsproblem nicht konkurrenzfähig zu sein. In Kombination mit Schnittebenen erhält man Branch & Price & Cut mit dem 2005 einige bisher ungelöste Benchmarkinstanzen gelöst werden konnten.

Zusammenfassung

- Ganzzahlige Programme haben üblicherweise Struktur.

Zusammenfassung

- Ganzzahlige Programme haben üblicherweise Struktur.
- Netzwerkstrukturen liefern gutartige (Teil-)Probleme.

Zusammenfassung

- Ganzzahlige Programme haben üblicherweise Struktur.
- Netzwerkstrukturen liefern gutartige (Teil-)Probleme.
- Die Kenntnis auch weiterer gutartiger Probleme ist oft hilfreich.

Zusammenfassung

- Ganzzahlige Programme haben üblicherweise Struktur.
- Netzwerkstrukturen liefern gutartige (Teil-)Probleme.
- Die Kenntnis auch weiterer gutartiger Probleme ist oft hilfreich.
- Pack- und Seuqnzierungsprobleme lassen sich häufig mit dynamischer Programmierung lösen.

Zusammenfassung

- Ganzzahlige Programme haben üblicherweise Struktur.
- Netzwerkstrukturen liefern gutartige (Teil-)Probleme.
- Die Kenntnis auch weiterer gutartiger Probleme ist oft hilfreich.
- Pack- und Seuqnzierungsprobleme lassen sich häufig mit dynamischer Programmierung lösen.
- Cutting Plane Methods sind ein allgemeines und oft erfolgreiches Verfahren.

Zusammenfassung

- Ganzzahlige Programme haben üblicherweise Struktur.
- Netzwerkstrukturen liefern gutartige (Teil-)Probleme.
- Die Kenntnis auch weiterer gutartiger Probleme ist oft hilfreich.
- Pack- und Seuqnzierungsprobleme lassen sich häufig mit dynamischer Programmierung lösen.
- Cutting Plane Methods sind ein allgemeines und oft erfolgreiches Verfahren.
- Das TSP hat man recht gut im Griff, das CVRP weniger.

Geschafft

Vielen Dank für Ihre Aufmerksamkeit