Kombinatorische Optimierung

Theorie, Algorithmen und Anwendungen

Prof. Dr. Peter Becker

Fachbereich Informatik
Hochschule Bonn-Rhein-Sieg

Sommersemester 2020

Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

Allgemeines zur Vorlesung

Homepage:

`http://www2.inf.h-brs.de/~pbecke2m/kombopt/`

Die Vorlesung wird überwiegend folienbasiert gehalten.

Folien sind knapp gehalten (Definiton, Satz, Beweis, Beispiel).

Die Folien zur Vorlesung (Skript) stehen auf der Homepage vor der Vorlesung zur Verfügung.

Übungen

Wöchentlich erscheint ein Aufgabenblatt mit Punkten, das

- eine Sollpunktzahl hat,
- in Zweiergruppen bearbeitet werden kann,
- in der folgenden Woche vor den Übungen abgegeben werden muss,
- in den Übungen besprochen wird und
- von mir bewertet wird.

Heute: Ausgabe des ersten Aufgabenblatts!

Zu erfüllende Vorleistung: $\geq$ 50% der gesamten Sollpunktzahl.

Wer die Vorleistung nicht erfüllt, wird nicht zur Prüfung zugelassen!

Lernziele

- Algorithmen zur Lösung kombinatorischer Optimierungsprobleme kennen, anwenden und in Grundzügen implementieren können,
- Eigenschaften der Algorithmen wissen, mathematisch beschreiben und beweisen können,
- in der Lage sein, praktische Problemstellungen in ein geeignetes mathematisches Modell überführen zu können und
- solch ein Modell unter Einsatz von Softwarewerkzeugen lösen können.

Inhalt (geplant)

1. Totale Unimodularität
2. Komplexität
3. Schnittebenenverfahren
4. Branch-and-Bound
5. Branch-and-Cut
6. Heuristiken und Approximation

Organisatorisches und Formales

Inhaltliche Voraussetzungen:

- Kenntnisse in Linearer Programmierung und der Nutzung eines LP-Solvers (Gurobi oder GLPK)
- Lineare Algebra und Graphentheorie
- Datenstrukturen und Algorithmen
- Theoretische Informatik

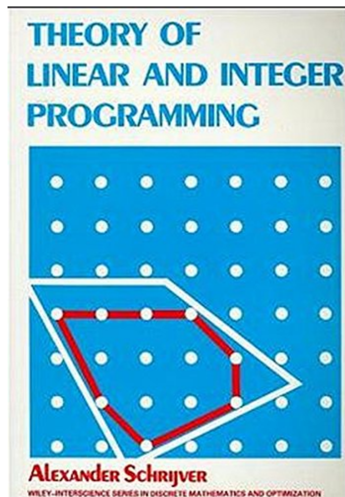
Umfang: 2V + 2Ü

Übungen/Vorleistung: $\geq$ 50% der Sollpunkte aus den Übungen

Prüfungsform: mündlich

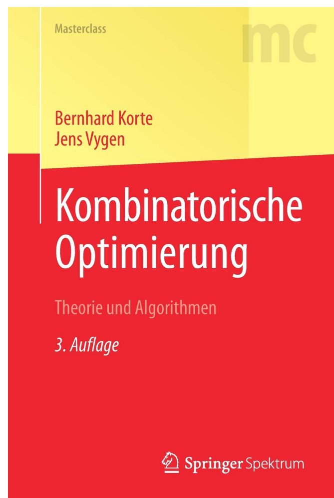
Softwarewerkzeuge: GNU Linear Programming Kit, Gurobi

Literatur

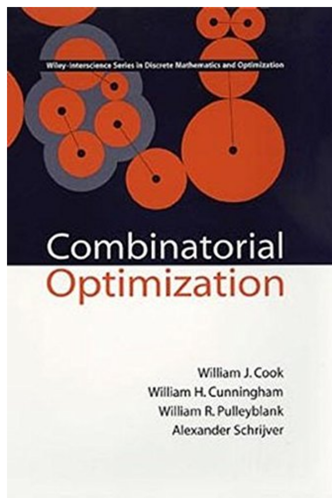


- Klassiker zum Thema lineare und ganzzahlige Programmierung
- umfassend und tiefgehend
- trocken, nicht immer leicht verständlich

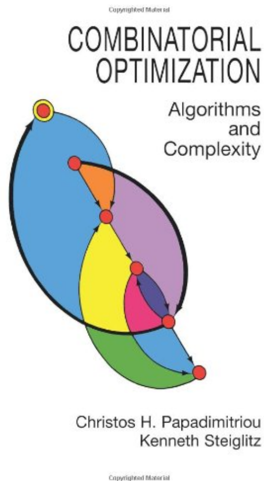
- umfassend: für Lineare und kombinatorische Optimierung
- geeignet für die Komplexitätsbetrachtungen
- in der kombinatorischen Optimierung nicht so tiefgehend



- deutscher Klassiker zum Thema kombinatorische Optimierung
- legt etwas andere Schwerpunkte als diese Veranstaltung
- als PDF in der Bibliothek verfügbar



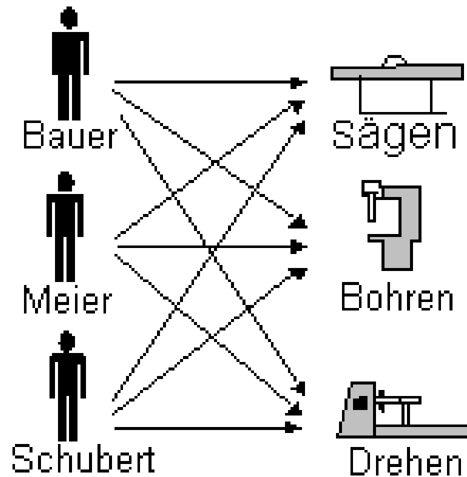
- bedeutendes internationales Lehrbuch
- kompakt, Darstellungen ziemlich knapp
- ohne lineare Programmierung



- bedeutendes internationales Lehrbuch
- legt andere Schwerpunkte

Kapitel 1

Totale Unimodularität



Inhalt

1 Totale Unimodularität

- Wiederholung: Transport- und Zuordnungsproblem
- Total unimodulare Matrizen
- Inzidenzmatrix
- Optimierungsprobleme auf Graphen

Transportproblem

Definition 1.1

Das Optimierungsproblem $\min \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij}$

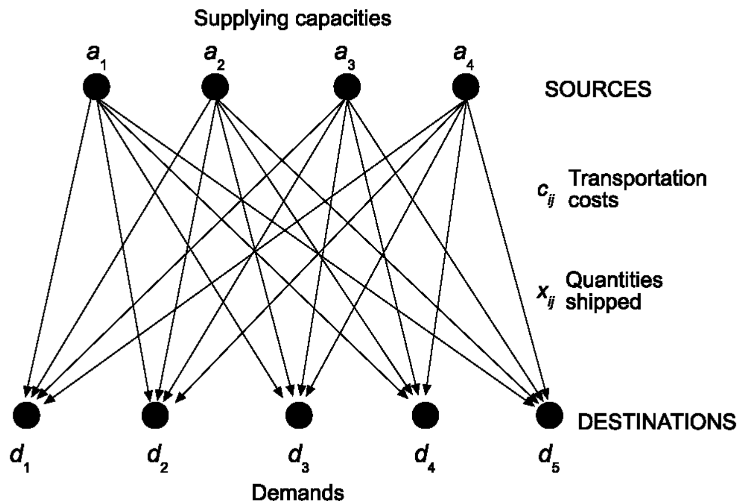
unter den Nebenbedingungen

$$\begin{aligned} \sum_{j=1}^{n} x_{ij} &= a_i \quad \text{für } i = 1, \ldots, m \\ \sum_{i=1}^{m} x_{ij} &= b_j \quad \text{für } j = 1, \ldots, n \end{aligned}$$

und den Vorzeichenbedingungen

$$x_{ij} \geq 0 \text{ für } i = 1, \ldots, m \text{ und } j = 1, \ldots, n$$

heißt Transportproblem.


Supplying capacities
a1
a2
a3
a4
SOURCES
cij Transportation costs
xij Quantities shipped
DESTINATIONS
d1
d2
d3
d4
d5
Demands

Bemerkungen zum Transportproblem

Wir setzen ein geschlossenes Transportproblem voraus: $a_i > 0, b_j > 0$ und $\sum_{i=1}^{m} a_i = \sum_{j=1}^{n} b_j$, also Gesamtangebot = Gesamtnachfrage.

Für den Fall $\sum_{i=1}^{m} a_i > \sum_{j=1}^{n} b_j$ führen wir ein zusätzliches Warenhaus mit $b_{n+1} = \sum_{i=1}^{m} a_i - \sum_{j=1}^{n} b_j$ und $c_{i,n+1} = 0$ ein.

Für den Fall $\sum_{i=1}^{m} a_i < \sum_{j=1}^{n} b_j$ führen wir eine zusätzliche Produktionsstätte mit $a_{m+1} = \sum_{j=1}^{n} b_j - \sum_{i=1}^{m} a_i$ ein.

Die $c_{m+1,j}$ modellieren dann die Kosten pro ME für das mangelnde Angebot in Warenhaus j.

Anzahl Variablen: $m \cdot n$

Beispielproblem

Beispiel 1.2

Wir gehen von folgenden Kosten, Angebot und Nachfrage aus:

	B_1	B_2	B_3	
A_1	9	1	3	50
A_2	4	5	8	70
	40	40	40	

Fortsetzung Beispiel.

Damit lautet das zugehörige Transportproblem

$$\min 9x_{11} + x_{12} + 3x_{13} + 4x_{21} + 5x_{22} + 8x_{23}$$

unter den Nebenbedingungen

$$\begin{array}{ccccccccccccc}
x_{11} & + & x_{12} & + & x_{13} & & & & & & & = & 50 \\
 & & & & & & x_{21} & + & x_{22} & + & x_{23} & = & 70 \\
x_{11} & & & & & + & x_{21} & & & & & = & 40 \\
 & & x_{12} & & & & & + & x_{22} & & & = & 40 \\
 & & & & x_{13} & & & & & + & x_{23} & = & 40
\end{array}$$

und Vorzeichenbedingungen

$$x_{11}, x_{12}, x_{13}, x_{21}, x_{22}, x_{23} \geq 0.$$

Lösungsalgorithmen

- Simplexalgorithmus
  Anzahl Basisvariablen: $n + m - 1$
- Stepping-Stone-Methode
  Kombinatorische Entsprechung des Simplex-Algorithmus
- u-v-Methode
  Verbesserung der Stepping-Stone-Methode durch Dualitätseigenschaften

Zuordnungsproblem

Definition 1.3

Das Optimierungsproblem $\min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$ unter den Nebenbedingungen

$$
\begin{aligned}
\sum_{j=1}^{n} x_{ij} &= 1 \quad \text{für } i = 1, \ldots, n \\
\sum_{i=1}^{n} x_{ij} &= 1 \quad \text{für } j = 1, \ldots, n
\end{aligned}
$$

sowie

$$x_{ij} \in \{0, 1\} \text{ für } i = 1, \ldots, n \text{ und } j = 1, \ldots, n$$

heißt Zuordnungsproblem.

Ecken des Zuordnungsproblems

Definition 1.4

Ein Zuordnungsproblem mit den Vorzeichenbedingungen

$$0 \leq x_{ij} \leq 1 \text{ für } i,j = 1,\ldots,n$$

statt $x_{ij} \in \{0,1\}$ heißt relaxiertes Zuordnungproblem.

Beispiel 1.5

Wir betrachten ein relaxiertes Zuordnungproblem mit Kostenmatrix

$$\mathbf{C} = (c_{ij}) = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}.$$

Fortsetzung Beispiel.

Dann sind

$$\begin{aligned} \mathbf{x} &= (x_{11}, x_{12}, x_{13}, x_{21}, x_{22}, x_{23}, x_{31}, x_{32}, x_{33}) \\ &= (1, 0, 0, 0, 1, 0, 0, 0, 1) \\ \mathbf{y} &= (0, 1, 0, 1, 0, 0, 0, 0, 1) \text{ und} \\ \mathbf{z} &= \left(\frac{1}{2}, \frac{1}{2}, 0, \frac{1}{2}, \frac{1}{2}, 0, 0, 0, 1\right) \end{aligned}$$

optimale Lösungen.
Wegen

$$\mathbf{z} = \frac{1}{2}\mathbf{x} + \frac{1}{2}\mathbf{y}$$

ist aber $\mathbf{z}$ keine Ecke und würde damit vom Simplexalgorithmus niemals als optimale Lösung ermittelt.

Transport- vs. Zuordnungsproblem

- Das relaxierte Zuordnungsproblem ist ein spezielles Transportproblem.
- $n = m$ und $a_1 = \cdots = a_m = b_1 = \cdots = b_n = 1$
- Alle Lösungsalgorithmen für das Transportproblem können auch auf das relaxierte Zuordnungsproblem angewendet werden.

Ganzzahligkeit der Ecken beim Zuordnungsproblem

Satz 1.6

Für jedes relaxierte Zuordnungsproblem sind alle Ecken ganzzahlig.

Für ein relaxiertes Zuordnungsproblem der Größe $n \times n$ gilt also

$$\mathbf{x} \text{ ist Ecke } \Rightarrow \mathbf{x} \in \{0, 1\}^{n \times n}$$

Beweis.

Induktion über n.

$n = 1$: $x_{11} = 1$ ist die einzige zulässige und damit optimale Lösung.

$n - 1 \rightarrow n$: Es sei $\mathbf{x}$ Ecke eines relaxierten $n \times n$-Zuordnungsproblems.

Fall 1: Es existieren $1 \leq i, j \leq n$ mit $x_{ij} = 1$.

- Dann streiche aus dem Zuordnungsproblem Zeile i und Spalte j und aus $\mathbf{x}$ alle entsprechenden Komponenten.
- Der Restvektor von $\mathbf{x}$ muss dann eine Ecke des $(n-1) \times (n-1)$ Zuordnungsproblems sein, das nach I. V. nur ganzzahlige Ecken hat.

Fortsetzung Beweis.

Fall 2: Es existiert kein i, j mit $x_{ij} = 1$.

- Damit folgt $0 \leq x_{ij} < 1$ für alle i, j.
- Wegen $\sum_{j=1}^{n} x_{ij} = 1$ für alle i folgt: Für jedes i gibt es mindestens zwei Variablen $x_{ij} > 0$.
- Damit existieren mindestens $2n$ Variablen $x_{ij} > 0$.
- Widerspruch, denn eine Ecke $\mathbf{x}$ und damit eine zulässige Basislösung hat nur $2n - 1$ Basisvariablen.

Folgerung 1.7

Wir können Zuordnungsprobleme mit dem Simplexalgorithmus optimal lösen.

Konsequenz

Wir können Zuordnungsprobleme lösen, indem wir

- zum relaxierten Problem übergehen und
- das relaxierte Problem mit dem Simplexalgorithmus lösen.

Wir wollen nun untersuchen,

- für welche weiteren kombinatorischen Probleme solch ein Vorgehen möglich ist, bzw.
- welche Bedingungen hinreichend für ganzzahlige Ecken sind.

Unimodulare Matrizen

Definition 1.8

Eine ganzzahlige Matrix $\mathbf{A} \in \mathbb{Z}^{n \times n}$ heißt unimodular, wenn

$$|\det(\mathbf{A})| = 1$$

gilt, d. h. $\det(\mathbf{A}) = 1$ oder $\det(\mathbf{A}) = -1$.

Beispiel 1.9

Die Matrix

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

ist unimodular.

Cramersche Regel

Für den Beweis des übernächsten Lemmas benötigen wir die sogenannte Cramersche-Regel.

Lemma 1.10

Es sei $\mathbf{A} \in \mathbb{R}^{n \times n}$ *eine quadratische Matrix mit* $\det(\mathbf{A}) \neq 0$*. Für das LGS* $\mathbf{Ax} = \mathbf{b}$ *sei*

$$\mathbf{A}_j := (\mathbf{a}^1, \ldots, \mathbf{a}^{j-1}, \mathbf{b}, \mathbf{a}^{j+1}, \ldots, \mathbf{a}^n),$$

also die Matrix, die entsteht, wenn in $\mathbf{A}$ *die* j*-te Spalte durch den Vektor* $\mathbf{b}$ *ersetzt wird.*

Dann gilt für die eindeutige Lösung $\mathbf{x} = (x_j) \in \mathbb{R}^n$ *des Gleichungssystems* $\mathbf{Ax} = \mathbf{b}$

$$x_j = \frac{\det(\mathbf{A}_j)}{\det(\mathbf{A})}.$$

Beispiel 1.11

Wir betrachten das LGS

$$\underbrace{\begin{pmatrix} 3 & 0 & 1 \\ 6 & 2 & 1 \\ -3 & -2 & 1 \end{pmatrix}}_{\mathbf{A}} \underbrace{\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}}_{\mathbf{x}} = \underbrace{\begin{pmatrix} 1 \\ 3 \\ -1 \end{pmatrix}}_{\mathbf{b}}.$$

Es gilt

$$\begin{aligned} \det \begin{pmatrix} 3 & 0 & 1 \\ 6 & 2 & 1 \\ -3 & -2 & 1 \end{pmatrix} &= 6 + 0 - 12 + 6 + 6 - 0 \\ &= 6. \end{aligned}$$

Fortsetzung Beispiel.

$$\det(\mathbf{A}_1) = \det \begin{pmatrix} 1 & 0 & 1 \\ 3 & 2 & 1 \\ -1 & -2 & 1 \end{pmatrix} = 2 + 0 - 6 + 2 + 2 - 0 = 0$$

$$\det(\mathbf{A}_2) = \det \begin{pmatrix} 3 & 1 & 1 \\ 6 & 3 & 1 \\ -3 & -1 & 1 \end{pmatrix} = 9 - 3 - 6 + 9 + 3 - 6 = 6$$

$$\det(\mathbf{A}_3) = \det \begin{pmatrix} 3 & 0 & 1 \\ 6 & 2 & 3 \\ -3 & -2 & -1 \end{pmatrix} = -6 + 0 - 12 + 6 + 18 - 0 = 6$$

Daraus folgt

$$x_1 = \frac{0}{6} = 0, \quad x_2 = \frac{6}{6} = 1, \quad x_3 = \frac{6}{6} = 1.$$

Eigenschaft unimodular Matrizen

Lemma 1.12

Es sei $\mathbf{A} \in \mathbb{Z}^{n \times n}$ *eine unimodulare Matrix und* $\mathbf{b} \in \mathbb{Z}^n$ *beliebig.*

Dann hat der Vektor

$$\tilde{\mathbf{b}} = \mathbf{A}^{-1}\mathbf{b}$$

ausschließlich ganzzahlige Komponenten.

Beweis.

- $\tilde{\mathbf{b}}$ ist die eindeutige Lösung des linearen Gleichungssystems $\mathbf{Ax} = \mathbf{b}$.
- Wir nutzen die Cramersche Regel: Es gilt

$$|\tilde{b}_j| = \frac{|\det(\mathbf{A}_j)|}{|\det(\mathbf{A})|} = |\det(\mathbf{A}_j)|,$$

  weil $|\det(\mathbf{A})| = 1$.
- Wegen $\mathbf{A}_j \in \mathbb{Z}^{n \times n}$ ist die Determinante von $\mathbf{A}_j$ ganzzahlig.

Quadratische Untermatrizen

Definition 1.13

Für eine Matrix $\mathbf{A} = (a_{ij}) \in \mathbb{R}^{m \times n}$ sowie

- Zeilenindizes $1 \leq i_1 < i_2 < \cdots < i_k \leq m$ und
- Spaltenindizes $1 \leq j_1 < j_2 < \cdots < j_k \leq n$

heißt die Matrix

$$\begin{pmatrix} a_{i_1,j_1} & a_{i_1,j_2} & \cdots & a_{i_1,j_k} \\ a_{i_2,j_1} & a_{i_2,j_2} & \cdots & a_{i_2,j_k} \\ \vdots & \vdots & & \vdots \\ a_{i_k,j_1} & a_{i_k,j_2} & \cdots & a_{i_k,j_k} \end{pmatrix} \in \mathbb{R}^{k \times k}$$

quadratische Untermatrix von $\mathbf{A}$.

Totale unimodulare Matrix

Definition 1.14

Eine Matrix $\mathbf{A} \in \mathbb{Z}^{m \times n}$ ist total unimodular genau dann, wenn jede quadratische Untermatrix von $\mathbf{A}$ die Determinante $0, 1$ oder -1 hat.

- Wenn $\mathbf{A} = (a_{ij})$ total unimodular ist, dann sind alle Matrixelemente a_{ij} gleich $0, 1$ oder -1.
  ☞ notwendige Bedingung
- Die Umkehrung gilt natürlich nicht.
  ☞ Bedingung ist nicht hinreichend.

Beispiel einer total unimodularen Matrix

Beispiel 1.15

1. Die folgende Matrix ist total unimodular:

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

2. Die Matrix

$$\mathbf{B} = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}$$

ist nicht total unimodular.

Totale Unimodularität und ganzzahlige Ecken

Satz 1.16

Es sei $\mathbf{A} \in \mathbb{Z}^{m \times n}$ *eine total unimodulare Matrix und* $\mathbf{b} \in \mathbb{Z}^m$ *sei ein ganzzahliger Vektor. Dann hat die Menge*

$$\mathcal{X} = \{\mathbf{x} \in \mathbb{R}^n | \mathbf{A}\mathbf{x} = \mathbf{b}, \mathbf{x} \geq \mathbf{0}\} \subseteq \mathbb{R}^n$$

nur ganzzahlige Ecken.

Beweis.

O.B.d.A. gelte $r(\mathbf{A}) = m$.

- $\mathbf{x}$ ist Ecke $\Leftrightarrow$ $\mathbf{x}$ ist zulässige Basislösung (siehe Lineare Optimierung, Satz 2.44)

Fortsetzung Beweis.

- $\mathbf{x}$ ist zulässige Basislösung $\Leftrightarrow \exists\, j_1, \ldots, j_m$ mit:
  - die Spaltenvektoren $\mathbf{a}^{j_1}, \ldots, \mathbf{a}^{j_m}$ sind linear unabhängig,
  - die Komponenten $x_{j_1}, \ldots, x_{j_m}$ von $\mathbf{x}$ sind für $\mathbf{A}' = (\mathbf{a}^{j_1}, \ldots, \mathbf{a}^{j_m})$ (eindeutige) Lösung des LGS

$$\mathbf{A}' \begin{pmatrix} x_{j_1} \\ \vdots \\ x_{j_m} \end{pmatrix} = \mathbf{b},$$

  - $\mathbf{x} \geq \mathbf{0}$.
- Nach der Cramer-Regel gilt

$$x_{j_k} = \frac{\det(\mathbf{A}'_{j_k})}{\det(\mathbf{A}')}.$$

Fortsetzung Beweis.

Weil $\mathbf{A}$ total unimodular ist und die Spaltenvektoren linear unabhängig sind, folgt $\det(\mathbf{A}') = 1$ oder -1.

Weil $\mathbf{b}$ ganzzahlig ist, ist auch $\det(\mathbf{A}'_{j_k})$ ganzzahlig.

Damit sind die x_{j_k} ganzzahlig.

Folgerung 1.17

Sei $\mathbf{A} \in \mathbb{Z}^{m \times n}$ *total unimodular und* $\mathbf{b} \in \mathbb{Z}^m$*.*

Dann haben auch die Mengen

- $\{\mathbf{x} \in \mathbb{R}^n | \mathbf{A}\mathbf{x} \leq \mathbf{b}, \mathbf{x} \geq \mathbf{0}\}$ *und*
- $\{\mathbf{x} \in \mathbb{R}^n | \mathbf{A}\mathbf{x} \leq \mathbf{b}\}$

nur ganzzahlige Ecken.

Varianten total unimodularer Matrizen

Lemma 1.18

Sei $\mathbf{A} \in \mathbb{Z}^{m \times n}$ *total unimodular. Dann gilt:*

1. *Jede Untermatrix von* $\mathbf{A}$ *ist total unimodular.*
2. $\mathbf{A}^T$, $-\mathbf{A}$ *und damit auch* $-\mathbf{A}^T$ *sind total unimodular.*
3. $\mathbf{A}$ *erweitert um einen Spalten- oder Zeileneinheitsvektor ist total unimodular.*

Beweis.

(1) und (2) sind offensichtlich.

(3): Entwicklung nach der Spalte- oder Zeile, die den Einheitsvektor enthält.

Charakterisierungen für total unimodulare Matrizen

Satz 1.19

Es sei $\mathbf{A} \in \{-1, 0, 1\}^{m \times n}$*. Dann sind die folgenden Aussagen äquivalent:*

1. $\mathbf{A}$ *ist total unimodular.*
2. *Für jeden Vektor* $\mathbf{b} \in \mathbb{Z}^m$ *hat das Polyeder* $\{\mathbf{x} \in \mathbb{R}^n | \mathbf{Ax} \leq \mathbf{b}, \mathbf{x} \geq \mathbf{0}\}$ *nur ganzzahlige Ecken.*
3. *Für alle Vektoren* $\mathbf{a}, \mathbf{b} \in \mathbb{Z}^m$ *und* $\mathbf{c}, \mathbf{d} \in \mathbb{Z}^n$ *hat das Polyeder* $\{\mathbf{x} \in \mathbb{R}^n | \mathbf{c} \leq \mathbf{x} \leq \mathbf{d}, \mathbf{a} \leq \mathbf{Ax} \leq \mathbf{b}\}$ *nur ganzzahlige Ecken.*
4. *Jede Spaltenmenge von* $\mathbf{A}$ *kann so in zwei Mengen aufgeteilt werden, dass die Differenz der Spaltensummen der Mengen einen Vektor ergibt, der nur aus den Komponenten* $-1, 0, 1$ *besteht.*
5. *Keine quadratische Untermatrix von* $\mathbf{A}$ *hat die Determinate* $+2$ *oder* -2*.*

Inzidenzmatrix für gerichtete Graphen

Definition 1.20

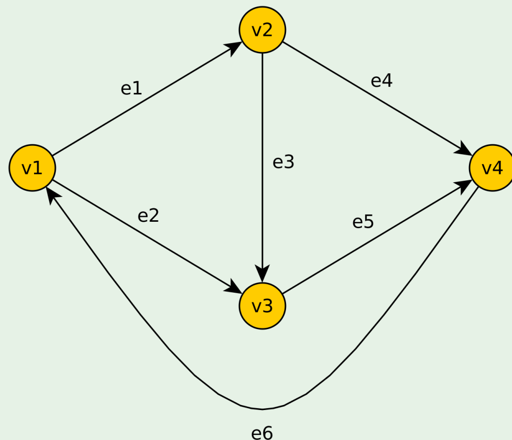
Es sei $G = (V, E)$ ein gerichteter Graph mit Knotenmenge $V = \{v_1, \ldots, v_m\}$ und Kantenmenge $E = \{e_1, \ldots, e_n\}$.
Dann heißt die $m \times n$-Matrix $\mathbf{A} = (a_{ij})$ mit

$$a_{ij} = \begin{cases} -1 & \text{wenn } v_i \text{ Anfangsknoten von } e_j \text{ ist,} \\ 1 & \text{wenn } v_i \text{ Endknoten von } e_j \text{ ist,} \\ 0 & \text{sonst} \end{cases}$$

Inzidenzmatrix von G.

Beispiel 1.21

Die Matrix von Beispiel 1.15 ist Inzidenzmatrix des folgenden Graphen:



Inzidenzmatrix für ungerichtete Graphen

Definition 1.22

Es sei $G = (V, E)$ ein (ungerichteter) Graph mit Knotenmenge $V = \{v_1, \dots, v_m\}$ und Kantenmenge $E = \{e_1, \dots, e_n\}$.
Dann heißt die $m \times n$ Matrix $\mathbf{A} = (a_{ij})$ mit

$$a_{ij} = \begin{cases} 1 & \text{wenn } v_i \text{ inzident mit } e_j \text{ ist} \\ 0 & \text{sonst} \end{cases}$$

Inzidenzmatrix von G.

Eigenschaften einer Inzidenzmatrix (1)

Lemma 1.23

Es sei G ein gerichteter Graph mit m Knoten. Dann hat die Inzidenzmatrix $\mathbf{A}$ von G einen Rang $r(\mathbf{A}) \leq m - 1$.

Beweis.

Die Summe der Zeilenvektoren ergibt den Nullvektor, da in jeder Spalte genau eine 1 und eine -1 existiert. □

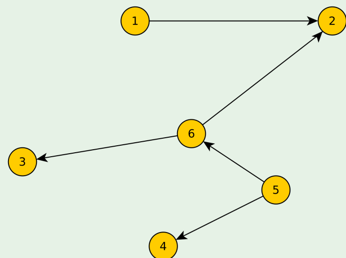
Eigenschaften einer Inzidenzmatrix (2)

Definition 1.24

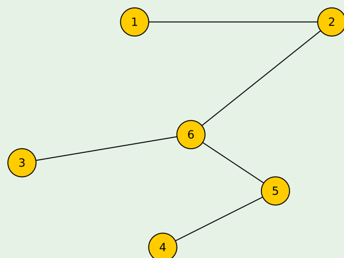
Ein gerichteter Graph G ist ein Wald bzw. ein Baum gdw. der G zugeordnete ungerichtete Graph G' (siehe Graphentheorie, Definition 1.44) ein Wald bzw. ein Baum ist.

Beispiel 1.25

gerichteter Graph G:



Der zugeordnete Graph G' ist ein Baum:



Lemma 1.26

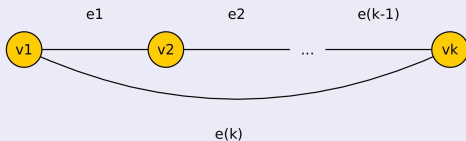
Ein gerichteter Graph G ist genau dann ein Wald, wenn die Spalten der Inzidenzmatrix von G linear unabhängig sind.

Beweis.

Wir zeigen: G enthält einen Kreis gdw. die Spalten der Inzidenzmatrix $\mathbf{A}$ linear abhängig sind.

Fortsetzung Beweis.

"$\Rightarrow$":



- Es sei $C =$ ein Kreis in G' und $j_1, \ldots, j_k$ seien die zugehörigen Spaltenindizes der Inzidenzmatrix.
- Für $l = 1, \ldots, k$ setzen wir:

$$\alpha_l = \begin{cases} 1 & e_l \text{ hat in } G \text{ die Richtung } v_{l-1} \to v_l \\ -1 & \text{sonst} \end{cases}$$

- Damit gilt

$$\alpha_1 \mathbf{a}^{j_1} + \cdots + \alpha_k \mathbf{a}^{j_k} = \mathbf{0}$$

die Spaltenvektoren sind also linear abhängig.

Fortsetzung Beweis.

"$\Leftarrow$":

- Die Spalten von **A** seien linear abhängig.
- Dann existieren Spaltenindizes $j_1, \ldots, j_k$ und Zahlen $\alpha_1, \ldots, \alpha_k \neq 0$ mit

$$\alpha_1 \mathbf{a}^{j_1} + \cdots + \alpha_k \mathbf{a}^{j_k} = \mathbf{0}$$

- Es sei E'' die Menge der Kanten zu den Spaltenindizes $j_1, \ldots, j_k$ und V'' sei die Menge der mit den Kanten aus E'' inzidenten Knoten.
- Wir betrachten jetzt den Graphen $G'' = (V'', E'')$. Weil alle $\alpha_j \neq 0$ muss es für jede Zeile i, in der nicht nur 0en auftreten, mindestens zwei Spalten geben, deren Linearkombination in der i-ten Zeile $= 0$ ist.
- Damit hat jeder Knoten in G'' mindestens den Grad 2 und G'' kann damit nicht kreisfrei sein.

Eigenschaften einer Inzidenzmatrix (3)

Satz 1.27

Es sei $\mathbf{A} \in \mathbb{Z}^{m \times n}$ *die Inzidenzmatriz eines gerichteten Graphen* G.

Dann ist $\mathbf{A}$ *total unimodular.*

Beweis.

Vollständige Induktion über die Größe k einer quadratischen Untermatrix.

$k = 1$: Die Untermatrizen der Größe $k = 1$ sind die Matrixelemente selbst. Per Definition der Inzidenzmatrix sind sie gleich $0, 1$ oder -1.

Fortsetzung Beweis.

$k-1 \to k$: Es sei $\mathbf{A}'$ eine quadratische Untermatrix von $\mathbf{A}$.
Fall 1: $\mathbf{A}'$ hat in jeder Spalte zwei Elemente $\neq 0$.

Dann definieren die Zeilen und Spalten von $\mathbf{A}'$ (als Inzidenzmatrix betrachtet) einen gerichteten Graphen G' mit k Knoten und k Kanten.

Damit kann G' nicht kreisfrei sein. Nach Lemma 1.26 sind die Spaltenvektoren von $\mathbf{A}'$ linear abhängig. Also folgt $\det(\mathbf{A}') = 0$.

Fall 2: $\mathbf{A}'$ enthält eine Spalte j mit höchstens einem Element $a'_{ij} \neq 0$. Zur Berechnung von $\det(\mathbf{A}')$ entwickeln wir nach Spalte j. Es folgt

$$\det(\mathbf{A}') = (-1)^{i+j} \cdot a'_{ij} \cdot \det(\mathbf{A}'_{ij})$$

Nach I.V. gilt $\det(\mathbf{A}'_{ij}) = 0, 1$ oder -1. Also gilt auch

$$\det(\mathbf{A}') = 0, 1 \text{ oder } -1.$$

Eigenschaften einer Inzidenzmatrix (4)

Satz 1.28

Es sei $\mathbf{A} \in \mathbb{Z}^{m \times n}$ *die Inzidenzmatrix eines (ungerichteten) bipartiten Graphen* G.

Dann ist $\mathbf{A}$ *total unimodular.*

Beweis.

Übungsaufgabe

Maximalflussproblem

Aus der Graphentheorie kennen wir das Maximalflussproblem: Gegeben ist ein Flussnetzwerk bestehend aus

- einem gerichteten Graphen $G = (V, E)$,
- einer Kapazitätsfunktion $c : E \longrightarrow \mathbb{R}_{\geq 0}$,
- einer Quelle $s \in V$ und einer Senke $t \in V$ mit $s \neq t$.

Gesucht ist ein Fluss $f : E \longrightarrow \mathbb{R}_{\geq 0}$ mit

$$0 \leq f(e) \leq c(e) \text{ für alle } e \in E$$

und

$$\sum_{(w,v)\in E} f(w, v) = \sum_{(v,w)\in E} f(v, w) \text{ für alle } v \in V \setminus \{s, t\}$$

der den Flusswert $\Phi(f) = \sum_{(s,w)\in E} f(s, w) - \sum_{(w,s)\in E} f(w, s)$ maximiert. Ein Fluss f, der $\Phi(f)$ maximiert, ist ein Maximalfluss.

Maximalflussproblem als LP

Man führe für jede gerichtete Kante $(v, w) \in E$ eine Variable x_{vw} ein. Damit erhält man das LP

$$\max \sum_{(s,w) \in E} x_{sw} - \sum_{(w,s) \in E} x_{ws}$$

unter den Nebenbedingungen

$$x_{vw} \leq c(v, w) \text{ für alle } (v, w) \in E$$

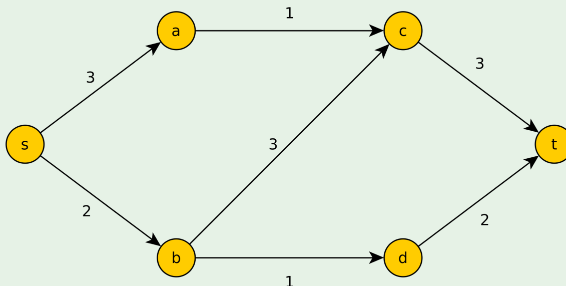
$$\sum_{(w,v) \in E} x_{wv} - \sum_{(v,w) \in E} x_{vw} = 0 \text{ für alle } v \in V \setminus \{s, t\}$$

sowie Vorzeichenbedingungen

$$x_{vw} \geq 0 \text{ für alle } (v, w) \in E$$

Beispiel 1.29

Gegeben sei das Flussnetzwerk



Das LP für das Maximalflussproblem lautet dann

$$\max x_{sa} + x_{sb}$$

Fortsetzung Beispiel.

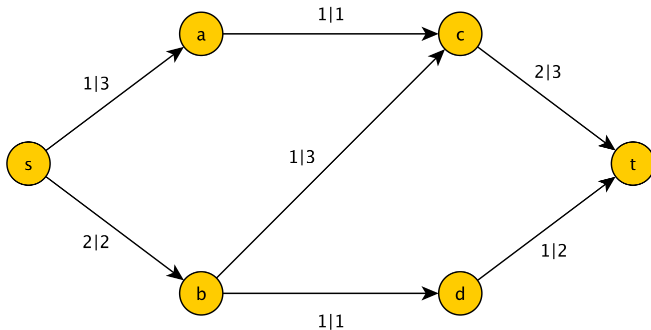
unter den Nebenbedingungen

$$
\begin{aligned}
x_{sa} &\leq 3 \\
x_{sb} &\leq 2 \\
x_{ac} &\leq 1 \\
x_{bc} &\leq 3 \\
x_{bd} &\leq 1 \\
x_{ct} &\leq 3 \\
x_{dt} &\leq 2 \\
x_{sa} - x_{ac} &= 0 \\
x_{sb} - x_{bc} - x_{bd} &= 0 \\
x_{ac} + x_{bc} - x_{ct} &= 0 \\
x_{bd} - x_{dt} &= 0
\end{aligned}
$$

und Vorzeichenbedingungen

$$x_{sa}, x_{sb}, x_{ac}, x_{bc}, x_{bd}, x_{ct}, x_{dt} \geq 0$$

Maximalfluss:



Koeffizientenmatrix in Normalform

Für jede Strukturvariable x_{vw} wird eine Schlupfvariable s_{vw} eingeführt.

$$\mathbf{F} = \left(\begin{array}{ccccccc|ccccccc}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\hline
1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}\right) = \left(\begin{array}{c|c} \mathbf{E} & \mathbf{E} \\ \hline \mathbf{A}' & \mathbf{0} \end{array}\right)$$

Satz 1.30

Die Koeffizientenmatrix $\mathbf{F}$ *eines Maximalflussproblems in Normalform ist total unimodular.*

Beweis.

Folgt induktiv aus Lemma 1.18 (3).

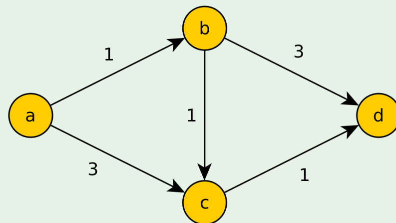
Folgerung 1.31

Für ganzzahlige Kapazitäten liefert der Simplexalgorithmus stets einen ganzzahligen Maximalfluss als optimale Lösung.

Kürzeste Wege

Beispiel 1.32

Für das Netzwerk



soll ein kürzester Weg von a nach d ermittelt werden.

Wir führen für jede gerichtete Kante $e = (v, w)$ eine Variable x_{vw} ein. Dann lautet das LP:

$$\min x_{ab} + 3x_{ac} + x_{bc} + 3x_{bd} + x_{cd}$$

Fortsetzung Beispiel.

unter den Nebenbedingungen

$$
\begin{aligned}
x_{ab} - x_{bc} - x_{bd} &= 0 \\
x_{ac} + x_{bc} - x_{cd} &= 0 \\
-x_{ab} - x_{ac} &= -1 \\
x_{bd} + x_{cd} &= 1
\end{aligned}
$$

und

$$x_{ab}, x_{ac}, x_{bc}, x_{bd}, x_{cd} \in \{0, 1\}$$

bzw. für das relaxierte LP

$$0 \leq x_{ab}, x_{ac}, x_{bc}, x_{bd}, x_{cd} \leq 1$$

Lemma 1.33

Das relaxierte LP zur Ermittlung eines kürzesten Weges hat unabhängig von den Kantengewichten ausschließlich ganzzahlige Ecken.

Transport- und Zuordnungsproblem revisited

Die Matrix $\mathbf{A}'$ der Nebenbedingungen eines relaxierten Zuordnungsproblems entspricht der Inzidenzmatrix eines vollständigen bipartiten Graphen $K_{n,n}$.

Damit ist die Matrix $\mathbf{A}'$ total unimodular.

Für die Beschränkungen $x_{ij} \leq 1$ werden zusätzliche Schlupfvariablen benötigt.

Insgesamt entsteht eine totale unimodulare Matrix in der gleichen Form wie beim Maximalflussproblem:

$$\begin{pmatrix} \mathbf{E} & \mathbf{E} \\ \mathbf{A}' & \mathbf{0} \end{pmatrix}$$

Damit haben wir einen anderen Beweis für Satz 1.6: Beim relaxierten Zuordnungsproblem sind alle Ecken ganzzahlig.

Beim Transportproblem entspricht die Koeffizientenmatrix direkt der Inzidenzmatrix eines vollständigen bipartiten Graphen $K_{m,n}$.

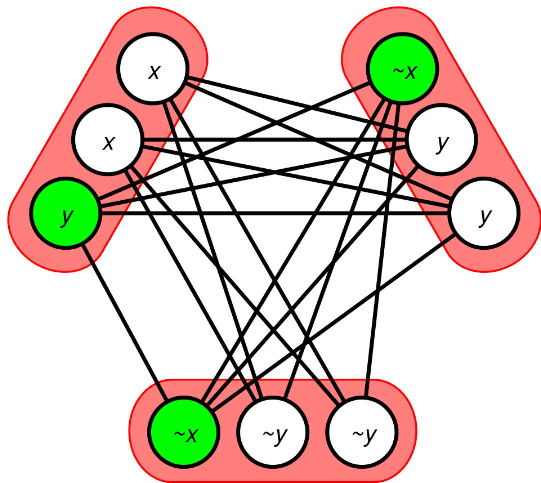
Konsequenz: Wenn die a_i und b_j alle ganzzahlig sind, dann hat das Transportproblem nur ganzzahlige Ecken.

Zusammenfassung

- ganzzahlige Ecken beim Zuordnungsproblem
- totale Unimodularität der Koeffizientenmatrix als wesentlicher Teil einer hinreichende Bedingung für ganzzahlige Ecken
- Inzidenzmatrizen als Beispiele für total unimodulare Matrizen
- Maximalflussproblem und kürzeste Wege als Beispiele für graphentheoretische LPs mit ganzzahligen Ecken

Kapitel 2

Komplexität



Inhalt

2 Komplexität

- Probleme, Komplexitätsmaße, Laufzeiten
- Komplexität der Linearen Programmierung
- Die Klassen $\mathcal{P}$ und $\mathcal{NP}$
- $\mathcal{NP}$-Vollständigkeit
- Komplexität und kombinatorische Optimierung

Probleme

Wir benötigen eine einigermaßen präzise Definition des Begriffs Problem.

Definition 2.1

- Ein Problem ist eine allgemeine Fragestellung, bei der mehrere Parameter offen gelassen sind und für die eine Lösung oder Antwort gesucht wird.
- Ein Problem ist dadurch definiert, dass alle seine Parameter beschrieben werden und dass genau angegeben wird, welche Eigenschaften eine Antwort (Lösung) haben soll.
- Sind alle Parameter eines Problems mit expliziten Daten belegt, dann sprechen wir von einer Probleminstanz.

Beispielproblem

Beispiel 2.2

- Das Travelling-Salesman-Problem ist ein Problem im Sinne von Definition 2.1.
- Seine offenen Parameter sind die Anzahl der Städte und die Entfernungen zwischen diesen Städten.
- Eine Entfernungstabelle definiert eine Probleminstanz für das Travelling-Salesman-Problem.
- Eine kürzeste Rundreise ist eine Lösung.

Bemerkung:

- Der Begriff Problem bezeichnet demnach eine abstrakte Problemstellung.
- Eine konkrete Problemstellung ist eine Probleminstanz.

Lösungsalgorithmus

- Mathematisch betrachten wir ein Problem Π als die Menge aller Probleminstanzen.
- Im mathematischen Sinn ist also das Travelling-Salesman-Problem die Menge aller TSP-Instanzen.

Definition 2.3

Ein Algorithmus löst ein Problem Π, wenn er für jede Probleminstanz $\mathcal{I} \in \Pi$ eine Lösung findet.

Bemerkung: Hier bleibt noch die Frage offen, wie solch eine Lösung aussehen kann.

Kodierungsschemata

- Ziel ist es, möglichst effiziente Verfahren zur Lösung von Problemen zu finden.
- Mittels Komplexitätsmaßen machen wir den Begriff der Effizienz messbar.
- am wichtigsten: Zeit- und Speicherplatzkomplexität
- Die Laufzeit eines Algorithmus hängt in der Regel von der Größe einer Probleminstanz ab, d. h. vom Umfang der Eingabedaten.
- Wir müssen also beschreiben, wie wir Probleminstanzen repräsentieren.
- Kodierungsschemata leisten diese Aufgabe.

Kodierung von Zahlen, Vektoren und Matrizen

- Ganze Zahlen kodieren wir binär.
- Die binäre Darstellung einer nichtnegativen ganzen Zahl n benötigt $\lceil \log_2(n+1) \rceil$ Bits. Hinzu kommt ein Bit für das Vorzeichen.
- Die Kodierungslänge $\langle n \rangle$ einer ganzen Zahl n ist dann

$$\langle n \rangle = \lceil \log_2(|n|+1) \rceil + 1.$$

- Jede rationale Zahl r hat eine Darstellung $r = \frac{p}{q}$ mit $p, q \in \mathbb{Z}$, p, q sind teilerfremd und $q > 0$.
- Die Kodierungslänge von $r = \frac{p}{q}$ ist dann

$$\langle r \rangle = \langle p \rangle + \langle q \rangle.$$

- Die Kodierungslänge eines Vektors $\mathbf{x} \in \mathbb{Q}^n$ ist

$$\langle \mathbf{x} \rangle = \sum_{j=1}^{n} \langle x_j \rangle.$$

- Die Kodierungslänge einer Matrix $\mathbf{A} \in \mathbb{Q}^{m \times n}$ ist

$$\langle \mathbf{A} \rangle = \sum_{i=1}^{m} \sum_{j=1}^{n} \langle a_{ij} \rangle.$$

- Kodierungslängen für andere Datenstrukturen (insbesondere Graphen) leiten wir aus den so gegebenen Kodierungslängen her.

Rechnermodell

- Wir benötigen ein Rechnermodell, um Speicher- und Laufzeitberechnungen durchführen zu können.
- Die Komplexitätstheorie nutzt hierzu Turing- oder RAM-Maschinen. Wir nutzen ein einfacheres Modell.
- Ein Algorithmus A soll Probleminstanz $\mathcal{I}$ des Problems Π lösen. Probleminstanzen liegen immer in kodierter Form vor. $\langle \mathcal{I} \rangle$ bezeichnet die Kodierungslänge von $\mathcal{I}$.
- Der Algorithmus liest diese Daten und beginnt Operationen auszuführen, d. h. Zahlen zu berechnen, zu speichern, zu löschen, usw.
- Die Anzahl der Speicherzellen, die während der Ausführung des Algorithmus A mindestens einmal benutzt werden, nennen wir den Speicherbedarf von A zur Lösung von $\mathcal{I}$.

- Die Laufzeit von A zur Lösung von $\mathcal{I}$ basiert auf der Anzahl der elementaren Operationen, die A bis zur Terminierung ausführt.
- Elementare Operationen sind

  Lesen, Schreiben, Initialisieren, Addieren, Subtrahieren, Multiplizieren, Dividieren und Vergleichen

  von rationalen Zahlen.
- Für die Laufzeitberechnung muss nun jede elementare Operation mit der Kodierungslänge der beteiligten Zahlen multipliziert werden.
- Die Laufzeit von A zur Lösung von $\mathcal{I}$ ist die Anzahl der ausgeführten elementaren Rechenoperationen multipliziert mit der längsten aufgetretenen Kodierungslänge einer Zahl.
- Wir tun also so, als hätten wir alle elementaren Operationen mit der am längsten kodierten Zahl ausgeführt.

Laufzeit- und Speicherplatzbedarf

Definition 2.4

Sei A ein Algorithmus zur Lösung eines Problems Π.

1. Die Funktion $f_A : \mathbb{N} \to \mathbb{N}$, definiert durch

$$f_A(n) = \max\left\{\text{Laufzeit von } A \text{ zur Lösung von } \mathcal{I} | \mathcal{I} \in \Pi \text{ und } \langle \mathcal{I} \rangle \leq n\right\},$$

   heißt Laufzeitfunktion von A.

2. Die Funktion $s_A : \mathbb{N} \to \mathbb{N}$, definiert durch

$$s_A(n) = \max\{\text{Speicherbedarf von } A \text{ zur Lösung von } \mathcal{I} | \\ \mathcal{I} \in \Pi \text{ und } \langle \mathcal{I} \rangle \leq n\},$$

   heißt Speicherplatzfunktion von A.

Fortsetzung Definition.

3. Der Algorithmus A hat polynomielle Laufzeit (kurz: ist polynomiell), wenn es ein Polynom $p : \mathbb{N} \to \mathbb{N}$ gibt mit

$$f_A(n) \leq p(n) \quad \text{für alle } n \in N.$$

   Wir schreiben $f_A = O(n^k)$, falls das Polynom p den Grad k hat.

4. Der Algorithmus A hat polynomiellen Speicherplatzbedarf, falls es ein Polynom $q : \mathbb{N} \to \mathbb{N}$ gibt mit

$$s_A(n) \leq q(n) \quad \text{für alle } n \in N.$$

Bemerkung: Ein Algorithmus, dessen Speicherplatzfunktion nicht durch ein Polynom beschränkt ist, kann nicht polynomiell sein.

Komplexität des Simplexalgorithmus

- Wir betrachten die lineare Programmierung (LP) als Problem.
- Eine Lösung im Sinne von Definition 2.1 ist eine optimale Lösung im Sinne der linearen Programmierung.
- Wie viel Rechenzeit benötigen wir, um mit dem Simplexalgorithmus eine optimale Lösung zu berechnen?
- Wir teilen die Frage auf:
    - Wie aufwendig ist eine Iteration?
    - Wie viele Iterationen sind notwendig?

Aufwand für eine Iteration (1)

- kanonische Normalform, m Nebenbedingungen, n Variablen
- Pivotspalte bestimmen: n Vergleiche
- Pivotzeile bestimmen: m Divisionen und Vergleiche
- Pivotzeile normalisieren: $n + 1$ Divisionen
- Tableau anpassen: $m \cdot (n + 1)$ Divisionen und Subtraktionen

Fazit: $O(mn)$ Operationen für eine Simplexiteration

☞ Dies ist noch kein Beweis für die polynomielle Laufzeit einer Simplexiteration (in der Theorie).

Warum? Kodierungslänge der Werte im Tableau!

Aufwand für eine Iteration (2)

- Kodierungslänge für ein LP: $\langle \mathbf{A} \rangle + \langle \mathbf{b} \rangle + \langle \mathbf{c} \rangle$
- Man kann zeigen, dass die Einträge im Simplextableau im Betrag durch

$$2^{\langle \mathbf{A} \rangle + \langle \mathbf{b} \rangle + \langle \mathbf{c} \rangle - 2n}$$

  beschränkt bleiben.
- Ihre Kodierungslänge bleibt also polynomiell in der Kodierungslänge des LP.
- Einen Beweis hierfür findet man bei Borgwardt (siehe Literaturhinweise).

Fakt 2.5

Eine Iteration des Simplexalgorithmus hat polynomielle Laufzeit.

Klee–Minty-Polytop

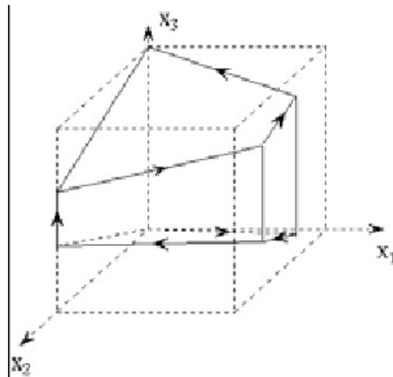
Definition 2.6

Das *d*-dimensionale Klee–Minty-Polytop ist durch folgendes Ungleichungssystem definiert:

$$
\begin{array}{rcrcrcrcrcl}
x_1 & & & & & & & & & \leq & 5 \\
4x_1 & + & x_2 & & & & & & & \leq & 25 \\
8x_1 & + & 4x_2 & + & x_3 & & & & & \leq & 125 \\
\vdots & & \vdots & + & \vdots & & & & & \leq & \vdots \\
2^d x_1 & + & 2^{d-1} x_2 & + & \cdots & & 4x_{d-1} & + & x_d & \leq & 5^d \\
 & & & & & & & & x_1, x_2, \ldots, x_d & \geq & 0
\end{array}
$$

Bemerkungen zum Klee–Minty-Polytop

- Das Klee–Minty-Polytop ist ein verzerrter Würfel.
- Das Klee–Minty-Polytop hat genauso viele d-dimensionale Seiten und Ecken wie der d-dimensionale Würfel,
- also $2d$ viele d-dimensionale Seiten und 2^d Ecken.



Nichtpolynomialität des Simplexalgorithmus (1)

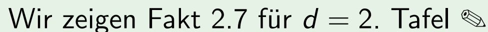
Fakt 2.7

Für die Zielfunktion

$$\max 2^{d-1}x_1 + 2^{d-2}x_2 + \cdots + 2x_{d-1} + x_d$$

durchläuft der Simplexalgorithmus mit der Pivotregel von Dantzig (negatives Element der Zielfunktionszeile mit größtem Betrag) alle 2^d Ecken des d-dimensionalen Klee–Minty-Polytops.

Beispiel 2.8



Wir zeigen Fakt 2.7 für $d = 2$. Tafel ✎

Nichtpolynomialität des Simplexalgorithmus (2)

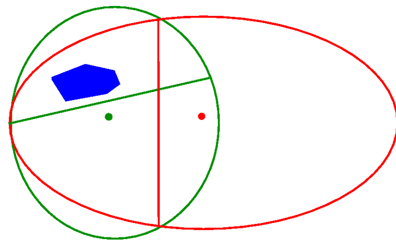
- Folgerung: Der Simplexalgorithmus (mit der Pivotregel von Dantzig) hat im Worst-Case keine polynomielle Laufzeit.
- Auch für alle anderen bekannten Pivot-Regeln gibt es analoge Resultate.
- Noch ungeklärt ist, ob es wirklich keine polynomielle Simplexvariante gibt.

Average-Case:

- Probabilistische Analysen liefern, dass für den Erwartungswert der Anzahl an Pivotschritten $O(m^{\frac{1}{n-1}} n^3)$ gilt (siehe Borgwardt).
- Andere Analysen kommen auf noch bessere Schranken.
- Damit hat der Simplexalgorithmus im Mittel polynomielle Laufzeit.

Ellipsoidalgorithmus

- Leonid Khachiyan (1979)
- polynomieller Algorithmus der entscheidet, ob ein Polyeder $P = \{\mathbf{x} \in \mathbb{R}^n | \mathbf{A}\mathbf{x} \leq \mathbf{b}\}$ leer ist oder nicht
- Idee: man konstruiert immer kleiner werdende Ellipsoide, die das Polyeder umfassen
- Nach einer maximalen Iterationszahl muss das Zentrum des Ellipsoids im Polyeder liegen (wenn nicht leer).
- Die maximale Iterationszahl ist polynomiell in der Kodierungslänge von **A** und **b**.



Äquivalenz zwischen Optimalität und Zulässigkeit

- Die Frage nach einer optimalen Lösung für ein LP ist genauso „schwierig“ wie die Frage nach einer Lösung für ein Ungleichungssystem.
- Das LP (P) $\max \mathbf{c}^T\mathbf{x}$ u.d.N. $\mathbf{A}\mathbf{x} \leq \mathbf{b}, \mathbf{x} \geq \mathbf{0}$ hat genau dann eine optimale Lösung, wenn das folgende Ungleichungssystem (D) eine Lösung hat.

$$\begin{aligned} &\mathbf{A}\mathbf{x} \leq \mathbf{b} \\ &\mathbf{A}^T\mathbf{u} \geq \mathbf{c} \\ &\mathbf{c}^T\mathbf{x} \geq \mathbf{b}^T\mathbf{u} \\ &\mathbf{x} \geq \mathbf{0}, \mathbf{u} \geq \mathbf{0} \end{aligned}$$

- Für jede Lösung $(\tilde{\mathbf{x}}, \tilde{\mathbf{u}})$ von (D) ist $\tilde{\mathbf{x}}$ eine optimale Lösung von (P).
- Begründung: Dualitätssätze

Polynomialität der linearen Programmierung

Folgerung 2.9

Es existiert ein polynomieller Algorithmus für die lineare Programmierung.

- Der Ellipsoidalgorithmus ist von großer theoretischer Bedeutung, für die Praxis aber zu ineffizient.
- Weitere polynomielle Alternativen sind Innere-Punkte-Methoden.
- Ausgehend von einem Punkt im Innern folgt man dem Gradienten der Zielfunktion.
- Probleme:
  - Schrittweite?
  - Am Rand führt die Richtung des Gradienten evtl. in die Unzulässigkeit.
- Hier keine weitere Betrachtung solcher Algorithmen.

Entscheidungsprobleme

Definition 2.10

Ein Entscheidungsproblem ist ein Problem, das nur zwei mögliche Lösungen (Ausgaben) besitzt, nämlich „ja“ oder „nein“.

Beispiel 2.11

Die Fragen

- Enthält ein Graph $G = (V, E)$ einen Kreis?
- Enthält ein Graph $G = (V, E)$ einen hamiltonschen Kreis?
- Ist ein Graph $G = (V, E)$ bipartit?
- Ist die Zahl $n \in \mathbb{N}$ eine Primzahl?

sind Entscheidungsprobleme.

Die Klasse $\mathcal{P}$

Definition 2.12

Die Klasse aller Entscheidungsprobleme, für die ein polynomieller Lösungsalgorithmus existiert, wird mit $\mathcal{P}$ bezeichnet.

Bemerkungen:

- Wir müssten $\mathcal{P}$ eigentlich abhängig von einem Kodierungsschema und einem Rechnermodell definieren.
- Wir beziehen die Definition daher auf das hier definierte Kodierungsschema und Rechenmodell.
- Das Entscheidungsproblem "Enthält ein Graph $G = (V, E)$ einen Kreis?" gehört zur Klasse $\mathcal{P}$.

Verifikation

- Wir wollen herausfinden, ob ein Graph $G = (V, E)$ einen hamiltonschen Kreis enthält.
- Angenommen, G hat einen hamiltonschen Kreis, und ein Orakel nennt uns eine Knotenfolge $K = (v_1, v_2, \ldots, v_n, v_1)$ und behauptet, dies sei ein hamiltonscher Kreis.
- Dann können wir jetzt prüfen, ob K tatsächlich ein Hamiltonkreis von G ist.
- Dazu müssen wir testen, ob
  - jeder Knoten v in V genau einmal in K auftritt und
  - $\{v_i, v_{i+1}\} \in E$ für $i = 1, \ldots, n-1$ und $\{v_n, v_1\} \in E$ gilt.
- Dies ist offensichtlich in polynomieller Zeit möglich.
- Somit können wir die Korrektheit der „ja“-Antwort polynomiell verifizieren.

Die Klasse $\mathcal{NP}$

Definition 2.13

Ein Entscheidungsproblem Π gehört zur Klasse $\mathcal{NP}$, wenn es die folgenden Eigenschaften hat:

1. Für jede Probleminstanz $\mathcal{I} \in \Pi$, für die die Lösung „ja“ lautet, gibt es mindestens ein Objekt Q, mit dessen Hilfe die Korrektheit der „ja“-Antwort überprüft werden kann.
2. Es gibt einen Algorithmus, der
   - Probleminstanzen $\mathcal{I} \in \Pi$ und Zusatzobjekte Q liest und
   - der in einer Laufzeit, die polynomiell in $\langle \mathcal{I} \rangle$ ist, überprüft, ob Q ein Objekt ist, aufgrund dessen Existenz eine „ja“-Antwort gegeben werden muss.

Beispiele für Probleme in $\mathcal{NP}$

Beispiel 2.14

1. Hat $G = (V, E)$ einen Kreis?
2. Hat $G = (V, E)$ einen hamiltonschen Kreis?
3. Ist $n \in \mathbb{N}$ keine Primzahl?
   Hier liefert das Orakel ein Produkt zweier Zahlen $\neq 1$.
4. Gibt es für $G = (V, E)$ eine Färbung mit k Farben?

Bemerkungen zur Klasse $\mathcal{NP}$

- Definition 2.13 sagt nichts darüber aus, wie das Zusatzobjekt Q zu finden ist. Es wird lediglich postuliert, dass es existiert.
- Die Laufzeit des Algorithmus in Definition 2.13 (2) ist polynomiell in $\mathcal{I}$. Da der Algorithmus Q lesen muss, folgt, dass $\langle Q \rangle$ polynomiell in $\mathcal{I}$ sein muss.
- Die Definition der Klasse $\mathcal{NP}$ ist unsymmetrisch: Die Definition impliziert nicht, dass wir auch für Probleminstanzen mit „nein“-Antworten Objekte Q und polynomielle Algorithmen mit den genannten Eigenschaften finden können.

Nichtdeterministischer polynomieller Algorithmus

- $\mathcal{NP}$ ist abgeleitet von „nichtdeterministischer polynomieller Algorithmus“.
- Nichtdeterministische Algorithmen können eine Lösung raten (Orakel).
- Ablauf:
  1. Ein Lösungsvorschlag Q wird geraten.
  2. Gibt es keine Lösung: STOP!
  3. Der Lösungsvorschlag Q wird überprüft.
  4. Ist Q eine Lösung, dann antwortet der Algorithmus mit „ja“.
- Es gilt $\mathcal{P} \subseteq \mathcal{NP}$, denn für Probleme in $\mathcal{P}$ existieren Algorithmen, die ohne Lösungsvorschlag Q in polynomieller Zeit eine Antwort liefern.

Die Klasse co-$\mathcal{NP}$

Definition 2.15

Die Klasse co-$\mathcal{NP}$ besteht aus den Entscheidungsproblemen, die Negationen von Problemen $\Pi \in \mathcal{NP}$ sind.

- Zu co-$\mathcal{NP}$ gehören z. B.:
  1. Hat $G = (V, E)$ keinen Kreis?
  2. Hat $G = (V, E)$ keinen hamiltonschen Kreis?
  3. Ist $n \in \mathbb{N}$ eine Primzahl?
- 1. und 3. sind auch $\in \mathcal{NP}$, sogar $\in \mathcal{P}$.
- Von 2. weiß man dies nicht.
- Es gilt $\mathcal{P} \subseteq$ co-$\mathcal{NP}$.

$\mathcal{P} = \mathcal{NP}$?

- Eigentlich sollte man meinen, dass Algorithmen, die eine Lösung raten können, mächtiger sind als übliche Algorithmen.
- Trotzdem ist die Frage „$\mathcal{P} = \mathcal{NP}$?“ immer noch ungelöst.
- Millennium-Problem, 1 Mio. US$ Preisgeld
- Vermutung: $\mathcal{P} \neq \mathcal{NP}$
- Könnte man dies bestätigen, können wir für viele praxisrelevante Probleme niemals effiziente Algorithmen finden (für große Probleminstanzen).

Weitere offene Fragen

- Wir wissen: $\mathcal{P} \subseteq \mathcal{NP} \cap$ co-$\mathcal{NP}$
- Gilt $\mathcal{P} = \mathcal{NP} \cap$ co-$\mathcal{NP}$?
- Gilt $\mathcal{NP} =$ co-$\mathcal{NP}$?

Polynomielle Transformation

- Wir wollen nun innnerhalb der Klasse $\mathcal{NP}$ eine Klasse von besonders schwierigen Problemen auszeichnen.

Definition 2.16

Gegeben seien die Entscheidungsprobleme Π und Π'.

Eine polynomielle Transformation von Π in Π' ist ein polynomieller Algorithmus, der aus einer Probleminstanz $\mathcal{I} \in \Pi$ eine Probleminstanz $\mathcal{I}' \in \Pi'$ erzeugt, so dass folgendes gilt:

Die Antwort für $\mathcal{I}$ ist genau dann „ja“, wenn die Antwort für $\mathcal{I}'$ „ja“ ist.

Wir sagen dann, dass Π polynomiell transformierbar in Π' ist und schreiben

$$\Pi \leq_p \Pi'.$$

Konsequenz

Damit gilt Folgendes:

- Wenn Π polynomiell transformierbar in Π' ist und für Π' ein polynomieller Lösungsalgorithmus existiert,
- dann können wir auch Π in polynomieller Laufzeit lösen.
- Hierzu transformieren wir einfach eine Instanz $\mathcal{I} \in \Pi$ in eine Instanz $\mathcal{I}' \in \Pi'$ und wenden den Lösungsalgorithmus für Π' an.
- Sowohl die Transformation als auch der Lösungsalgorithmus für Π' sind polynomiell, somit auch die Kombination.

$\mathcal{NP}$-vollständig

Definition 2.17

Ein Entscheidungsproblem Π heißt $\mathcal{NP}$-vollständig, wenn folgendes gilt:

1. $\Pi \in \mathcal{NP}$
2. Für alle Probleme $\Pi' \in \mathcal{NP}$ gilt:
   Π' ist polynomiell transformierbar in Π.

Ein $\mathcal{NP}$-vollständiges Problem hat demnach die folgende Eigenschaft:

- Wenn Π in polynomieller Zeit gelöst werden kann, dann kann auch jedes andere Problem in $\mathcal{NP}$ in polynomieller Zeit gelöst werden.
- Π ist $\mathcal{NP}$-vollständig und $\Pi \in \mathcal{P} \Longrightarrow \mathcal{P} = \mathcal{NP}$.

Satz von Cook

- Konsequenz: Bezüglich polynomieller Lösbarkeit ist kein Problem schwieriger als ein $\mathcal{NP}$-vollständiges.
- Aber gibt es überhaupt $\mathcal{NP}$-vollständige Probleme? Ja!

Definition 2.18

Das Erfüllbarkeitsproblem der Aussagenlogik (SAT) lautet:

- Gegeben ist eine aussagenlogische Formel Φ in konjunktiver Normalform (KNF) mit Variablen $x_1, \ldots, x_n$.
- Existiert eine Belegung der Variablen, so dass Φ wahr wird?

Satz 2.19 (Cook (1971))

SAT ist $\mathcal{NP}$-vollständig.

Beweis für die $\mathcal{NP}$-Vollständigkeit eines Problems

- Wie können wir beweisen, dass ein Entscheidungsproblem Π $\mathcal{NP}$-vollständig ist?
- Hierfür die Definition der $\mathcal{NP}$-Vollständigkeit zu nutzen, ist sehr unhandlich.
- Der Satz von Cook ist überaus hilfreich.

Folgerung 2.20

Es sei Π ein Entscheidungsproblem. Dann ist Π genau dann $\mathcal{NP}$-vollständig, wenn gilt:

1. $\Pi \in \mathcal{NP}$ *und*
2. $SAT \leq_p \Pi$.

3-SAT ist $\mathcal{NP}$-vollständig

Definition 2.21

Das Problem 3-SAT lautet:

- Existiert für eine aussagenlogische Formel Φ in KNF und genau drei Literalen pro Klausel (3KNF) eine Belegung der Variablen, so dass Φ wahr wird?

Lemma 2.22

3-SAT ist $\mathcal{NP}$-vollständig.

Beweis.

1. Wir müssen 3-SAT $\in \mathcal{NP}$ zeigen.
   - Für jede Instanz $\mathcal{I} \in$ 3-SAT gilt $\mathcal{I} \in$ SAT.
   - Ein nichtdeterministischer polynomieller Algorithmus für SAT löst somit auch 3-SAT-Instanzen.
   - Damit folgt 3-SAT $\in \mathcal{NP}$.

Fortsetzung Beweis.

2. Wir müssen SAT $\leq_p$ 3-SAT zeigen.
   - Wir zeigen nun, wie eine beliebige KNF-Formel Φ in eine Formel Ψ mit genau drei Literalen pro Klausel umgewandelt werden kann (3KNF).
   - Es sei C eine Klausel mit nur einem Literal, also $C = x_i$ oder $C = \neg x_i$.

     Dann führen wir zwei neue Variablen y_i, z_i ein und ersetzen C durch die Klauseln

     $$C_1 = x_i \vee y_i \vee z_i, C_2 = x_i \vee y_i \vee \neg z_i, C_3 = x_i \vee \neg y_i \vee z_i, C_4 = x_i \vee \neg y_i \vee \neg z_i.$$

   - Analog für $C = \neg x_i$.
   - Es gilt $C \Leftrightarrow C_1 \wedge C_2 \wedge C_3 \wedge C_4$.
   - Es sei C eine Klausel mit zwei Literalen, o.B.d.A. $C = x_i \vee x_j$.

     Dann führen wir eine neue Variable y_{ij} ein und ersetzen C durch die Klauseln

     $$C_1 = x_i \vee x_j \vee y_{ij}, \quad C_2 = x_i \vee x_j \vee \neg y_{ij}.$$

     Es gilt $C \Leftrightarrow C_1 \wedge C_2$.

Fortsetzung Beweis.

2. Fortsetzung SAT $\leq_p$ 3-SAT.
   - Es sei C eine Klausel von Φ mit vier Literalen, z. B.

$$C = x_1 \vee \neg x_2 \vee \neg x_3 \vee x_4.$$

   - Wir führen eine neue Variable z ein und ersetzen C durch die beiden Klauseln

$$C_1 = x_1 \vee \neg x_2 \vee z \quad \text{und} \quad C_2 = \neg x_3 \vee x_4 \vee \neg z.$$

   - Es gilt: $C \Leftrightarrow C_1 \wedge C_2$.
   - Diese Technik funktioniert auch allgemein, um eine Klausel mit k Literalen durch zwei Klauseln mit 3 und $k-1$ Literalen zu ersetzen.
   - Sukzessive Anwendung liefert einen polynomiellen Transformationsalgorithmus.

Transitivität der polynomiellen Transformierbarkeit

- Statt SAT können wir nun auch 3-SAT in Folgerung 2.20 verwenden.
- Das gilt natürlich nicht nur für 3-SAT, sondern jedes $\mathcal{NP}$-vollständige Problem.

Folgerung 2.23

Es sei Π ein Entscheidungsproblem. Dann ist Π genau dann $\mathcal{NP}$-vollständig, wenn gilt:

1. $\Pi \in \mathcal{NP}$ *und*
2. *es existiert ein $\mathcal{NP}$-vollständiges Problem Π' mit $\Pi' \leq_p \Pi$.*

VC ist $\mathcal{NP}$-vollständig

Definition 2.24

Das Knotenüberdeckungsproblem (vertex cover, VC) lautet:

- Gegeben sei ein Graph $G = (V, E)$ und eine natürliche Zahl k.
- Existiert eine Teilmenge $U \subseteq V$ mit $|U| \leq k$, so dass jede Kante $e \in E$ mit mindestens einem Knoten aus U inzident ist?

Satz 2.25

VC ist $\mathcal{NP}$-vollständig.

Beweis.

1. Für eine Teilmenge $U \subseteq V$ können wir in polynomieller Zeit prüfen, ob die Knotenüberdeckungseigenschaft erfüllt ist.
2. Wir zeigen 3-SAT $\leq_p$ VC.

Fortsetzung Beweis.

- Es sei $\Phi = C_1 \wedge \ldots \wedge C_m$ eine 3KNF Formel, die aus den Klauseln C_i und den Variablen $x_1, \ldots, x_n$ besteht.
- Wir müssen einen Graphen $G = (V, E)$ und ein $k \in \mathbb{N}$ konstruieren, so dass G genau dann eine Knotenüberdeckung der Größe $\leq k$ hat, wenn Φ erfüllbar ist.
- G besteht aus drei Arten von Komponenten.
- Belegungskomponenten:
  - Für jede Variable x_i definieren wir die Komponente $T_i = (V_i, E_i)$ mit $V_i = \{x_i, \neg x_i\}$ und $E_i = \{\{x_i, \neg x_i\}\}$.
  - Jede Knotenüberdeckung enthält somit mindestens einen der beiden Knoten x_i und $\neg x_i$.

Fortsetzung Beweis.

- Testkomponenten:
  - Für jede Klausel $C_i \in \Phi$ definieren wir eine Testkomponente $S_j = (V'_j, E'_j)$, die ein Dreieck bildet:

$$\begin{aligned} V'_j &= \{a_1[j], a_2[j], a_3[j]\}, \\ E'_j &= \{\{a_1[j], a_2[j]\}, \{a_1[j], a_3[j]\}, \{a_2[j], a_3[j]\}\}. \end{aligned}$$

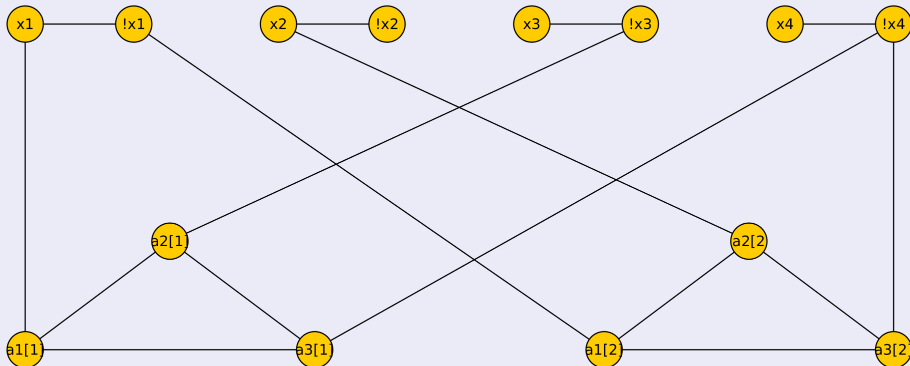
  - Jede Knotenüberdeckung enthält mindestens zwei Knoten aus S_j.
- Kommunikationskomponenten:
  - verbinden die Belegungskomponenten mit den Testkomponenten
  - Für eine Klausel $C_j = u_j \vee v_j \vee w_j$, wobei u_j, v_j, w_j Literale sind, setzen wir

$$E''_j = \{\{a_1[j], u_j\}, \{a_2[j], v_j\}, \{a_3[j], w_j\}\}.$$

  - D. h. wir verbinden das ¡ue$i$-te Literal einer Klausel mit dem i-ten Knoten der zugehörigen Testkomponente.

Fortsetzung Beweis.

- Sei $k = n + 2m$ und $G = (V, E)$ mit
  - $V = \left(\bigcup_{i=1}^{n} V_i\right) \cup \left(\bigcup_{j=1}^{m} V_j'\right)$
  - $E = \left(\bigcup_{i=1}^{n} E_i\right) \cup \left(\bigcup_{j=1}^{m} E_j'\right) \cup \left(\bigcup_{j=1}^{m} E_j''\right)$.
- Beispiel für $\Phi = (x_1 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee \neg x_4)$:



Fortsetzung Beweis.

- „$\Rightarrow$“: Sei $B : \{x_1, \ldots, x_n\} \to \{\texttt{true}, \texttt{false}\}$ eine Belegung, die Φ wahr macht.
- Die Knotenüberdeckung besteht dann aus:
  - x_i, wenn $B(x_i) = \texttt{true}$ und $\neg x_i$, wenn $B(x_i) = \texttt{false}$.
  - Damit sind dann alle Kanten E_i aus den Belegungskomponenten abgedeckt.
  - Außerdem ist jeweils mindestens eine Kante aus den Mengen E_j'' (Kommunikationskomponenten) abgedeckt, denn Φ erfüllt jede Klausel.
  - Um auch die jeweils beiden anderen Kanten der Mengen E_j'' abzudecken, wählen wir die beiden entsprechenden Knoten aus V_j' (Testkomponente).
  - Damit sind dann auch die Kanten E_j' aus den Testkomponenten abgedeckt.
- Die Größe der Knotenüberdeckung ist $n + 2m$.

Fortsetzung Beweis.

- „$\Leftarrow$“: Sei $U \subseteq V$ eine Knotenüberdeckung von G mit $|U| \leq n + 2m$.
- siehe Bemerkungen bei Belegungs- und Testkomponenten: eine Knotenüberdeckung muss mindestens $n + 2m$ Knoten umfassen.
- $\Rightarrow |U| = n + 2m$ und U enhält pro Belegungskomponente genau einen Knoten und pro Testkomponente genau zwei Knoten.
- Gilt $x_i \in U$, dann setzen wir $B(x_i) = \texttt{true}$, ansonsten $B(x_i) = \texttt{false}$.
- Von den drei Kanten in der j-ten Kommunikationskomponente können nur zwei durch $V'_j \cap U$ abgedeckt werden.
- Also wird die dritte Kante durch einen Knoten aus einer Belegungskomponente abgedeckt.
- Die entsprechende Belegung macht damit die j-te Klausel wahr.
- Dies gilt für alle j. Also wird die Formel Φ erfüllt.

Hamiltonkreisproblem

Definition 2.26

Das Hamiltonkreisproblem (HC) lautet:

- Gegeben ist ein Graph $G = (V, E)$.
- Enthält G einen hamiltonschen Kreis?

Das gerichtete Hamiltonkreisproblem (DHC) lautet:

- Gegeben ist ein gerichteter Graph $G = (V, E)$.
- Enthält G einen gerichteten hamiltonschen Kreis?

Das (gerichtete) Hamiltonwegprobem (HP bzw. DHP) lautet:

- Gegeben ist ein (gerichteter) Graph $G = (V, E)$.
- Enthält G einen (gerichteten) hamiltonschen Weg?

DHC ist $\mathcal{NP}$-vollständig

Satz 2.27

DHC ist $\mathcal{NP}$-vollständig.

Beweis

1. Für eine Knotenfolge $(v_1, v_2, \ldots, v_n, v_1)$ können wir in polynomieller Zeit prüfen, ob diese einen gerichteten Hamiltonkreis von G bildet.
2. Wir zeigen 3-SAT $\leq_p$ DHC.
   - Es sei $\Phi = C_1 \wedge \ldots \wedge C_m$ eine 3KNF-Formel, die aus den Klauseln C_i und den Variablen $x_1, \ldots, x_n$ besteht.
   - Wir konstruieren einen gerichteten Graphen $G = (V, E)$.
   - Wir definieren für jede Variable x_i einen Knoten.
   - Wir definieren für jede Klausel eine Komponente, die aus sechs Knoten besteht.
   - Also $|V| = n + 6m$.
   - Jeder Variablenknoten x_i hat genau zwei ausgehende und zwei eingehende Kanten.

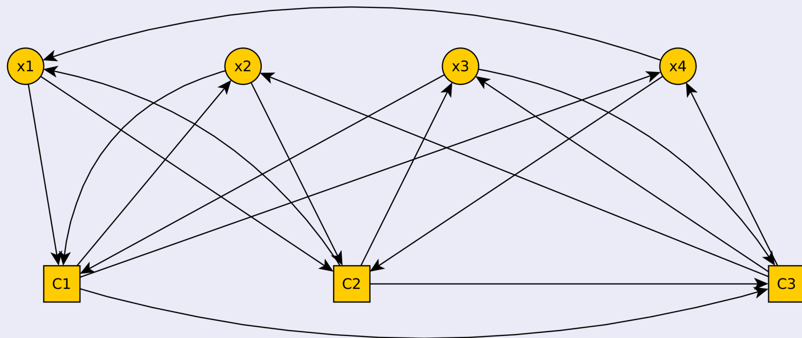
Fortsetzung Beweis.

2. Fortsetzung 3-SAT $\leq_p$ DHC.
   - Jede Klauselkomponente hat genau drei eingehende und drei ausgehende Kanten.
   - Die beiden ausgehenden Kanten aus einem Variablenknoten repräsentieren die Literale x_i und $\neg x_i$.
   - Die erste ausgehende Kante aus dem Knoten x_i ist l-te eingehende Kante der j-ten Klauselkomponente, wenn die j-te Klauselkomponente die erste ist, in der x_i auftritt, wobei x_i das l-te Literal in dieser Klausel ist.
   - Die l-te ausgehende Kante der j-ten Klauselkomponente ist dann die l'-te eingehende Kante in die j'-te Klauselkomponente, wenn die j'-te Klausel die zweite ist, in der x_i auftritt, wobei x_i das l'-te Literal in dieser Klausel ist, usw.
   - Wenn es kein weiteres Vorkommen von x_i gibt, dann ist die aus der Klauselkomponente ausgehende Kante die erste in den Variablenknoten x_{i+1} eingehende Kante, bzw. in den Variablenknoten x_1 für $i = n$.
   - Die zweite aus einem Variablenknoten ausgehende Kante hat die gleiche Funktion für das negative Literal $\neg x_i$.

Fortsetzung Beweis.

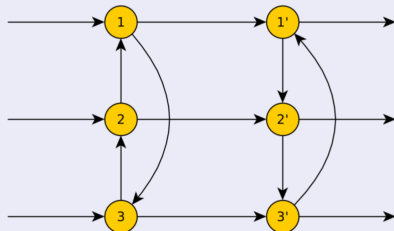
Beispiel für die Formel

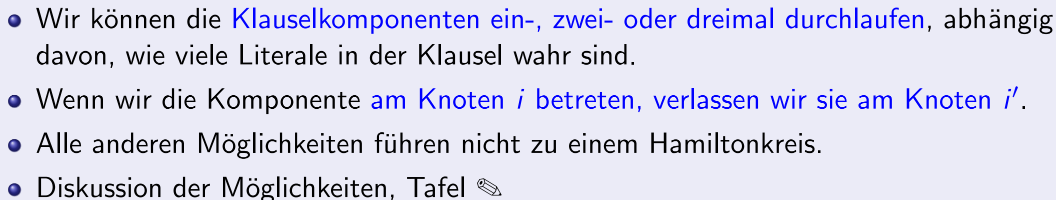
$$\Phi = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_4) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3).$$



Fortsetzung Beweis.

Konstruktion der Klauselkomponenten:





- Wir können die Klauselkomponenten ein-, zwei- oder dreimal durchlaufen, abhängig davon, wie viele Literale in der Klausel wahr sind.
- Wenn wir die Komponente am Knoten i betreten, verlassen wir sie am Knoten i'.
- Alle anderen Möglichkeiten führen nicht zu einem Hamiltonkreis.
- Diskussion der Möglichkeiten, Tafel ✎

Fortsetzung Beweis.

- "$\Rightarrow$": Sei $B : \{x_1, \ldots, x_n\} \rightarrow \{\texttt{true}, \texttt{false}\}$ eine Belegung, die Φ wahr macht.
- Dann starten wir den Hamiltonkreis am Variablenknoten x_1 und
- beginnen mit der Kante, die der Variablenbelegung entspricht.
- Die erreichten Klauselkomponenten werden so durchlaufen wie diskutiert,
- wobei wir berücksichtigen, wie viele Literale pro Klausel wahr sind.
- Wir erreichen Variablenknoten x_2 und fahren entsprechend fort.
- Damit konstruieren wir einen gerichteten Hamiltonkreis.

Fortsetzung Beweis.

- "$\Leftarrow$": Sei ein Hamiltonkreis gegeben.
- Wir durchlaufen ihn beginnend beim Knoten x_1.
- Abhängig von der Kante, welche der Hamiltonkreis wählt, belegen wir x_1.
- Dann durchlaufen wir die erste Klauselkomponente, die das erfüllt x_1-Literal enthält.
- Gemäß unseren Überlegungen wird die Komponente so verlassen, dass wir die zweite Klauselkomponente erreichen, usw.,
- bis wir den Variablenknoten x_2 erreichen.
- Auf diese Weise konstruieren wir eine Belegung der Variablen.
- Wir haben nur Klauselkomponenten durchlaufen, die erfüllte Literale enthalten.
- Da wir einen Hamiltonkreis haben, sind wir durch alle Klauselkomponenten gelaufen, es sind also alle erfüllt.

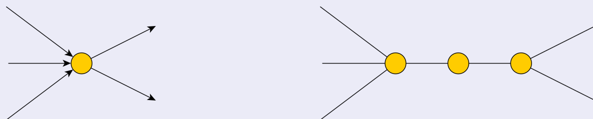
HC ist $\mathcal{NP}$-vollständig

Satz 2.28

HC ist $\mathcal{NP}$-vollständig.

Beweis.

1. Für eine Knotenfolge $(v_1, v_2, \ldots, v_n, v_1)$ können wir in polynomieller Zeit prüfen, ob diese einen Hamiltonkreis von G bildet.
2. Wir zeigen $DHC \leq_p HC$.
   - Hierzu führen wir die folgende Ersetzung für jeden Knoten durch:



   - "$\Rightarrow$": offensichtlich
   - "$\Leftarrow$": Für einen Hamiltonkreis im ungerichteten Graphen legen wir auf Basis des gerichteten Graphen eine Richtung fest.

Traveling-Salesman-Problem

Definition 2.29

Das Traveling-Salesman-Problem (TSP) lautet:

- Gegeben sei ein vollständiger Graph $G = (V, E)$, eine Funktion $c : E \to \mathbb{N}_0$ und ein $k \in \mathbb{N}_0$.
- Enthält G einen Hamiltonkreis mit einer Länge $\leq k$?

Folgerung 2.30

TSP ist $\mathcal{NP}$-vollständig.

Beweis

Übungsaufgabe.

Summenproblem

Definition 2.31

Das Summenproblem (SUM) lautet:

- Gegeben ist eine Menge $A = \{a_1, a_2, \ldots, a_n\} \subseteq \mathbb{N}_0$ und eine Zahl $S \in \mathbb{N}_0$.
- Existiert eine Teilmenge $B \subseteq A$ mit $\sum_{b \in B} b = S$?

Lemma 2.32

SUM ist $\mathcal{NP}$-vollständig.

Beweis.

1. Für eine Menge $B = \{b_1, \ldots, b_k\} \subseteq A$ können wir in polynomieller Zeit prüfen, ob $\sum_{i=1}^{k} b_i = S$ gilt.
2. Wir zeigen 3-SAT $\leq_p$ SUM.

Fortsetzung Beweis.

- Es sei $\Phi = C_1 \land \ldots \land C_m$ eine 3KNF-Formel, die aus den Klauseln C_i und den Variablen $x_1, \ldots, x_n$ besteht.
- Sei $S = \underbrace{44 \cdots 4}_{m \text{ Ziffern}} \underbrace{11 \cdots 1}_{n \text{ Ziffern}}$.
- Die Menge A umfasst $2m + 2n$ viele Zahlen a_i, b_i, c_j, d_j mit $1 \leq i \leq n$ und $1 \leq j \leq m$.
- Alle Zahlen haben wie S auch $m + n$ viele Ziffern.
- Die Zahl a_i repräsentiert das positive Literal x_i.
  - a_i hat genau dann im linken Block bei der j-ten Ziffern eine 1, wenn x_i in der j-ten Klausel auftritt, sonst eine 0.
  - Im rechten Block hat a_i genau an Position i eine 1, sonst 0en.
- Die Zahl b_i ist analog aufgebaut für das Literal $\neg x_i$.
- Die Zahl c_j hat im linken Block an der j-ten Stelle eine 1, sonst überall 0en.
- Die Zahl d_j hat im linken Block an der j-ten Stelle eine 2, sonst überall 0en.

Fortsetzung Beweis.

- Beispiel: $\Phi = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_4) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3)$, also $m = 3, n = 4$.

$$
\begin{array}{lcllcl}
S & = & 444\,1111 & & & \\
\hline
a_1 & = & 100\,1000 & b_1 & = & 011\,1000 \\
a_2 & = & 010\,0100 & b_2 & = & 101\,0100 \\
a_3 & = & 100\,0010 & b_3 & = & 001\,0010 \\
a_4 & = & 000\,0001 & b_4 & = & 010\,0001 \\
\hline
c_1 & = & 100\,0000 & d_1 & = & 200\,0000 \\
c_2 & = & 010\,0000 & d_2 & = & 020\,0000 \\
c_3 & = & 001\,0000 & d_3 & = & 002\,0000
\end{array}
$$

Fortsetzung Beweis.

- "$\Rightarrow$": Sei eine Belegung gegeben, die Φ wahr macht.
- Durch Summenbildung enstehen keine Überträge!
- Dann wählen wir für jedes $1 \leq i \leq n$ von den Zahlen a_i und b_i genau eine aus, abhängig von der Belegung für x_i.
- Damit entsteht in der Summe an allen n Stellen des rechten Blocks eine 1.
- Da die Belegung alle Klauseln erfüllt, haben wir in der Summe an jeder Stelle j des linken Blocks eine 1, 2 oder 3 (abhängig davon, wie viele Literale die j-te Klausel erfüllen).
- Damit können wir für jedes j aus den Zahlen c_j und d_j so auswählen, dass wir in der Summe auf eine 4 kommen.

Fortsetzung Beweis.

- "$\Leftarrow$": Es gebe eine Teilmenge $B \subseteq \{a_1, \dots, a_n, b_1, \dots, b_n, c_1, \dots, c_m, d_1, \dots, d_m\}$ deren Summe gleich S ist.
- Dann muss für jedes i genau eine der beiden Zahlen a_i oder b_i in B enthalten sein (um auf eine 1 an der i-ten Stelle zu kommen).
- Je nachdem welche der beiden Zahlen enthalten ist, belegen wir x_i.
- Da c_j und d_j in der Summe an der j-ten Stelle eine 3 ergeben, muss für jedes j mindestens eine Zahl enthalten sein, die an der j-ten Stelle im linken Block eine 1 hat.
- Damit erfüllt die obige Belegung die j-te Klausel.
- Dies gilt für alle $1 \leq j \leq m$, womit die Formel Φ erfüllt ist.

Rucksackproblem

Definition 2.33

Das Rucksackproblem (KP) lautet:

- Gegeben sind Zahlen $w_1, \ldots, w_n \in \mathbb{N}$ und $p_1, \ldots, p_n \in \mathbb{N}$ sowie Zahlen $W, P \in \mathbb{N}$.
- Existiert eine Teilmenge $I \subseteq \{1, \ldots, n\}$ für die gilt $\sum_{i \in I} w_i \leq W$ und $\sum_{i \in I} p_i \geq P$?

Satz 2.34

KP ist $\mathcal{NP}$-vollständig.

Beweis.

1. Offensichtlich.
2. Mit $w_i = p_i = a_i$ und $P = W = S$ ergibt sich SUM $\leq_p$ KP.

Beweistechniken für $\mathcal{NP}$-Vollständigkeit

Wir wollen zeigen, dass ein Problem Π $\mathcal{NP}$-vollständig ist.

1. Restriktion
   Wir zeigen, dass Π ein $\mathcal{NP}$-vollständiges Problem Π' als Spezialfall enthält.
2. Lokale Ersetzung
   Wir ersetzen einen bestimmten Aspekt eines $\mathcal{NP}$-vollständigen Problems Π', um damit Π zu modellieren.
3. Komponentenentwurf
   Wir konstruieren Komponenten für Π, um damit ein $\mathcal{NP}$-vollständiges Problem Π' lösen zu können.

Restriktion

- Der Beweis für KP ist Restriktion.
- KP mit $p_i = w_i$ und $P = W$ ist SUM.

Beispiel 2.35

Problem Isomorphie von Untergraphen (subgraph isomophism):

- Gegeben seien zwei Graphen $G = (V_1, E_1)$ und $H = (V_2, E_2)$.
- Enthält G einen Untergraphen, der isomorph zu H ist?

Dieses Problem enthält CLIQUE als Spezialfall.

Hierzu wählt man für H einen vollständigen Graphen mit k Knoten.

Subgraph Isomorphism für einen vollständigen Graphen H ist CLIQUE.

Lokale Ersetzung

Wir ersetzen einen bestimmten Aspekt eines $\mathcal{NP}$-vollständigen Problems Π', um damit Π zu modellieren.

Beispiele:

- SAT $\leq_p$ 3-SAT
  Wir haben beliebige Klauseln durch Klauseln mit genau drei Literalen ersetzt.
- DHC $\leq_p$ HC
  Wir haben einzelne Knoten durch Knotentripel ersetzt.

Komponentenentwurf

Beispiele:

- Knotenüberdeckung (VC)
  3-SAT $\leq$ VC
- gerichtetes Hamiltonkreisproblem (DHC)
  3-SAT $\leq$ DHC
- Summenproblem (SUM)
  3-SAT $\leq$ SUM

Optimierungsprobleme und $\mathcal{NP}$ (1)

Die Definitionen 2.29 und 2.33 zeigen beispielhaft, wie wir aus einem Optimierungsproblem ein Entscheidungsproblem machen können.

- Ist Π ein Maximierungsproblem (Minimierungsproblem), so legen wir zusätzlich zu jeder Instanz $\mathcal{I} \in \Pi$ noch eine Schranke B fest und fragen:
  - Gibt es für $\mathcal{I}$ eine Lösung, deren Wert nicht kleiner (nicht größer) als B ist?

Definition 2.36

Wir nennen ein Optimierungsproblem $\mathcal{NP}$-schwer, wenn das zugeordnete Entscheidungsproblem $\mathcal{NP}$-vollständig ist.

Optimierungsprobleme und $\mathcal{NP}$ (2)

- Alle $\mathcal{NP}$-schweren Optimierungsprobleme sind mindestens so schwierig wie die $\mathcal{NP}$-vollständigen Probleme.
- Begründung: Könnten wir ein $\mathcal{NP}$-schweres Optimierungsproblem in polynomieller Zeit lösen, dann könnten wir auch das zugehörige Entscheidungsproblem in polynomieller Zeit lösen.
  - Wir berechnen den Wert w einer Optimallösung und vergleichen ihn mit B.
  - Ist bei einem Maximierungsproblem $w \geq B$, so antworten wir „ja“, ansonsten „nein“.

Optimierungsprobleme und $\mathcal{NP}$ (3)

- Häufig kann man Entscheidungsprobleme dazu benutzen, um Optimierungsprobleme zu lösen.
- Wir betrachten als Beispiel das TSP-Entscheidungsproblem mit n Städten.
- Ist s die kleinste vorkommende Entfernung, so ziehen wir von allen Entfernungen s ab.
- Damit hat dann die kürzeste Entfernung den Wert 0.
- Ist t die größte der (modifizierten Entfernungen), so folgt, dass keine Tour länger als $n \cdot t$ ist.
- Wir fragen nun den Algorithmus zur Lösung des TSP-Entscheidungsproblems, ob es eine Rundreise gibt, deren Länge nicht größer als $\frac{nt}{2}$ ist.
- Ist das so, fragen wir, ob es eine Rundreise gibt, deren Länge höchstens $\frac{nt}{4}$ ist, andernfalls fragen wir, ob es eine Rundreise gibt mit Länge höchstens $\frac{3nt}{4}$.

- Wir fahren auf diese Weise fort, bis wir das Interval für die Tourlänge einer optimalen Lösung auf eine einzelne mögliche Zahl reduziert haben.
- Diese Zahl ist dann die Länge einer kürzesten Rundreise.
- Insgesamt müssen wir zur Lösung des Optimierungsproblems das TSP-Entscheidungsproblem $(\lceil \log_2(nt) \rceil + 1)$-mal aufrufen.
- binäre Suche

$\mathcal{NP}$-Äquivalenz

Definition 2.37

Ein Optimierungsproblem Π heißt $\mathcal{NP}$-leicht, wenn ein Entscheidungsproblem $\Pi' \in \mathcal{NP}$ existiert, so dass Π durch polynomial viele Aufrufe eines Algorithmus zur Lösung von Π' gelöst werden kann.

Ein Optimierungsproblem Π heißt $\mathcal{NP}$-äquivalent, wenn Π sowohl $\mathcal{NP}$-leicht als auch $\mathcal{NP}$-schwer ist.

- $\mathcal{NP}$-leichte Probleme sind also nicht schwerer als die Probleme in $\mathcal{NP}$.
- Ein $\mathcal{NP}$-äquivalentes Problem ist genau dann in polynomieller Zeit lösbar, wenn $\mathcal{P} = \mathcal{NP}$ gilt.

Kombinatorische Optimierung

- Wir könnten ein allgemeines kombinatorisches Optimierungsproblem wie folgt definieren:
  - Gegeben seien eine endliche Menge $\mathcal{I}$ und eine Funktion $f : \mathcal{I} \to \mathbb{R}$, die jedem Element von $\mathcal{I}$ einen Wert zuordnet.
  - Gesucht als Lösung ist ein Element $I^* \in \mathcal{I}$, so dass $f(I^*)$ möglicht groß (oder klein) ist.
- Eine Problemformulierung dieser Art ist aber relativ sinnlos, denn es können kaum vernünftige mathematische Aussagen über das Problem getroffen werden.
- Algorithmisch ist dieses Problem trivial in linearer Laufzeit (in $\mathcal{I}$) lösbar: man durchlaufe alle Elemente $I \in \mathcal{I}$ und werte sie aus.

- In der Regel betrachten wir kombinatorische Optimierungsprobleme mit linearer Zielfunktion.
  - Gegeben sei eine endliche Grundmenge E, eine Menge $\mathcal{I} \subseteq \mathcal{P}(E)$ von zulässigen Lösungen und eine Funktion $c : E \to \mathbb{R}$.
  - Für jede Menge $F \subseteq E$ definieren wir ihren Wert

$$c(F) = \sum_{e \in F} c(e).$$

  - Wir suchen eine Menge $I^* \in \mathcal{I}$, so dass $c(I^*)$ minimal (oder maximal) wird.
- Die Menge $\mathcal{I}$ der zulässigen Lösungen wird dabei üblicherweise implizit angegeben.

$$\mathcal{I} = \{I \subseteq E | I \text{ hat Eigenschaft } \Pi\}$$

- Typischerweise wächst die Anzahl der Elemente von $\mathcal{I}$ dabei exponentiell in der Größe von E.

Gewichtetes VC als kombinatorisches Optimierungsproblem

Beispiel 2.38

- Gegeben sei ein Graph $G = (V, E)$ und eine Kostenfunktion $c : V \to \mathbb{R}$, die jedem Knoten $v \in V$ ein Gewicht zuordnet.
- Dann lautet die Menge $\mathcal{I}$ der zulässigen Lösungen:

$$\mathcal{I} = \{U \subseteq V | \forall e = \{v, w\} \in E : v \in U \vee w \in U\}.$$

- Zielfunktion für ein $I \in \mathcal{I}$:

$$\sum_{v \in I} c(v).$$

Binary Programming

In den meisten Fällen, die wir betrachten, können wir $\mathcal{I}$ durch eine Menge von m linearen Gleichungen oder Ungleichungen ausdrücken.

Definition 2.39

Ein Optimierungsproblem mit

- n Variablen $x_1, \ldots, x_n$,
- Zielfunktion $\sum_{j=1}^{n} c_j x_j$
- Nebenbedingungen $\sum_{j=1}^{n} a_{ij} x_j \theta b_i$ mit $\theta \in \{\leq, \geq, =\}$ und $i = 1, \ldots, m$
- und $x_j \in \{0, 1\}$.

heißt 0-1-Optimierungsproblem (BP).

- engl.: binary programming oder 0-1-programming
- Typischerweise sind dabei c_j, a_{ij}, b_j ganzzahlig.
- kurz: $\min / \max \mathbf{c}^T \mathbf{x}$ u.d.N. $\mathbf{A}\mathbf{x} \leq \mathbf{b}, \mathbf{x} \in \{0, 1\}^n$.

Gewichtetes VC als 0-1-Programm

Beispiel 2.40

Das Problem aus Beispiel 2.38 können wir wie folgt definieren:

- Sei $V = \{v_1, \ldots, v_n\}$. Für jeden Knoten $v_i \in V$ definieren wir eine Variable $x_i \in \{0, 1\}$.
- $c_j := c(v_j)$
- Zielfunktion:

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

- Nebenbedingungen: Für jede Kante $e = \{v_i, v_j\} \in E$ ensteht die Ungleichung

$$x_i + x_j \geq 1.$$

BP ist $\mathcal{NP}$-äquivalent

Folgerung 2.41

BP ist $\mathcal{NP}$-äquivalent.

Beweis.

- VC ist $\mathcal{NP}$-vollständig.
- Beispiel 2.40 zeigt, wie VC polynomiell in BP transformiert werden kann (setze $c_j = 1$).
- Für die Entscheidungsvariante von BP kann in polynomieller Zeit geprüft werden, ob ein Lösungsvorschlag $\mathbf{x} \in \{0,1\}^n$ die Bedingungen
  - $\mathbf{Ax} \leq \mathbf{b}$ und
  - $\mathbf{c}^T\mathbf{x} \leq k$ oder $\geq k$ erfüllt.

Folgerung 2.42

ILP (siehe Lineare Optimierung, Definition 1.8) ist $\mathcal{NP}$-äquivalent.

SAT und BP

Beispiel 2.43

Gegeben sei eine aussagenlogische Formel $\Phi = C_1 \wedge \ldots \wedge C_m$ in KNF.

Dann können wir die Frage, ob Φ erfüllbar ist, mit einem 0-1-Programm entscheiden.

- Wir führen für jede aussagenlogische Variable eine algebraische Variable $x_i \in \{0, 1\}$ ein.
- Aus jeder Klausel machen wir folgendermaßen eine Ungleichung:
  - positives Literal wird algebraisch zu x_i
  - negatives Literal wird algebraisch zu $(1 - x_i)$
  - Wir bilden die Summe dieser Terme.
  - Ungleichung: immer ≥ 1
- Beispiel: Aus $x_1 \vee \neg x_2 \vee \neg x_3$ wird

$$x_1 + (1 - x_2) + (1 - x_3) \geq 1 \quad \Leftrightarrow \quad -x_1 + x_2 + x_3 \leq 1.$$

Worum geht's?

Eine Analogie von Vašek Chvátal

"In den kommunistischen Ländern des Ostblocks in den 60'er und 70'er Jahren war es möglich, intelligent, ehrenhaft und ein Mitglied der kommunistischen Partei zu sein, aber es war nicht möglich, alle drei Eigenschaften gleichzeitig zu verkörpern."

- tschechisch-kanadischer Mathematiker
- arbeitet vor allem auf den Gebieten lineare und ganzzahlige Programmierung, Graphentheorie und Kombinatorik

Algorithmischer Wunschzettel

Für die Problemlösung wünschen wir uns Algorithmen, die

1. optimale Lösungen berechnen,
2. für jede mögliche Instanz und
3. in polynomieller Zeit.

Für $\mathcal{NP}$-schwere Probleme können wir dies nicht:

- aktuell nicht, weil niemand solch einen Algorithmus kennt,
- prinzipiell nicht, falls $\mathcal{P} \neq \mathcal{NP}$ gilt.

Was tun?

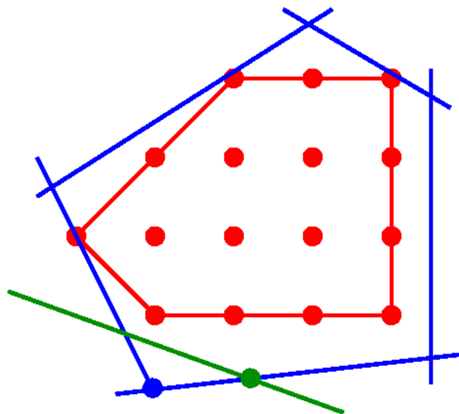
- Wir verzichten auf die polynomielle Zeit im Worst-Case.
  - klassische kombinatorische Optimierung bzw. ganzzahlige Programmierung.
  - Ziel: Für möglichst große Probleme in akzeptabler Rechenzeit eine optimale Lösung finden.
  - ☞ Kapitel 3 bis 5
- Wir betrachten nicht alle Instanzen.
  - Wir versuchen Spezialfälle zu finden, die in polynomieller Zeit optimal gelöst werden können.
  - problem- statt methodenorientiert, theoretisch interessant, in der Praxis wenig brauchbar
  - ☞ Diesen Ansatz verfolgen wir nicht weiter.
- Wir verzichten auf die Optimalität.
  - Es genügt uns, wenn eine Lösung fast optimal ist.
  - Verwendung von Heuristiken
  - Wir versuchen auch Aussagen über die Güte einer Lösung herzuleiten.
  - ☞ Kapitel 6 und 7

Zusammenfassung

- Simplexalgorithmus ist im Worst-Case nicht polynomiell, aber im Average-Case.
- Mit dem Ellipsoidalgorithmus existiert ein polynomieller Algorithmus für die lineare Programmierung.
- Klasse $\mathcal{NP}$: Polynomiell Lösungsvorschläge überprüfen können.
- polynomielle Transformation
- Kern für den Beweis der $\mathcal{NP}$-Vollständigkeit von Π: Ein $\mathcal{NP}$-vollständiges Π' polynomiell in Π transformieren.
- BP und ILP sind $\mathcal{NP}$-vollständig.

Kapitel 3

Schnittebenenverfahren



Inhalt

3 Schnittebenenverfahren

- Ganzzahlige lineare Programmierung
- Schnittebenenverfahren
- Konstruktion von Schnittebenen
- Auswahlkriterium für Schnittrestriktionen
- Schnittebenenverfahren mit Separation

Ganzzahliges lineares Programm

Definition 3.1

Ein lineares Programm mit zusätzlichen Bedingungen $x_i \in \mathbb{Z}$ für alle Variablen x_i heißt ganzzahliges lineares Programm (integer linear program, ILP).

Gilt $x_i \in \mathbb{Z}$ nicht für alle sondern nur für einige der Variablen, so spricht man von einem gemischt-ganzzahligen linearen Programm (mixed integer program, MIP).

Das lineare Programm, das entsteht, wenn wir in einem ILP bzw. MIP die Bedingungen für die Ganzzahligkeit weglassen, heißt LP-Relaxation.

Beispiele

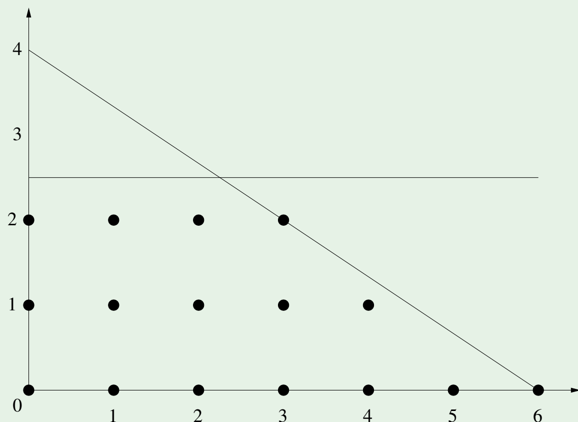
Beispiel 3.2

Wir betrachten das ILP

$$\max x_1 + 2x_2$$

unter den Neben- und Vorzeichenbedingungen

$$\begin{array}{rcrcr} & & 2x_2 & \leq & 5 \\ 2x_1 & + & 3x_2 & \leq & 12 \\ & & x_1, x_2 & \geq & 0 \\ & & x_1, x_2 & \in & \mathbb{Z} \end{array}$$



Fortsetzung Beispiel.

Die LP-Relaxation hat die optimale Lösung

$$\mathbf{x}' = (\frac{9}{4}, \frac{5}{2})$$

mit einem Zielfunktionswert $z' = \frac{29}{4}$.

Das ILP hat dagegen als optimale Lösung

$$\mathbf{x} = (3, 2)$$

mit Zielfunktionswert $z = 7$.

Man beachte: Die ganzzahlige Lösung, die $\mathbf{x}'$ am nächsten liegt, ist nicht optimal.

Beispiel 3.3

Für das ILP

$$\max x_1 + 2x_2$$

unter den Neben- und Vorzeichenbedingungen

$$\begin{array}{rcrcr} & & x_2 & \leq & 3 \\ 2x_1 & + & 3x_2 & \leq & 12 \\ & & x_1, x_2 & \geq & 0 \\ & & x_1, x_2 & \in & \mathbb{Z} \end{array}$$

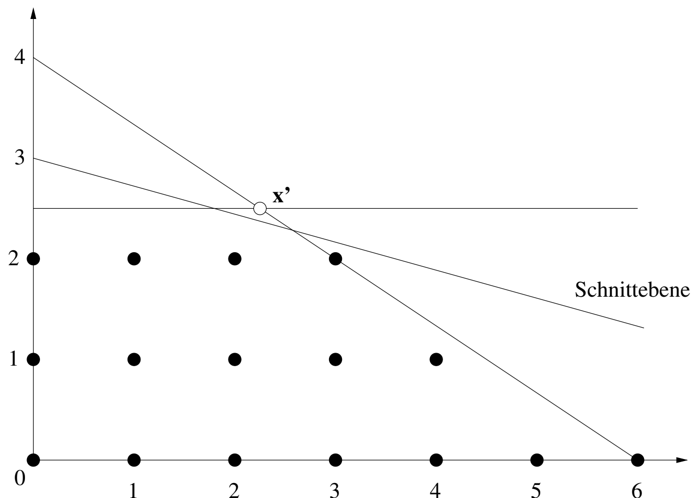
hat die LP-Relaxation die eindeutige optimale Lösung $\mathbf{x}' = (\frac{3}{2}, 3)$.
Das ILP hat dagegen zwei unterschiedliche optimale Lösungen: $\mathbf{x} = (3, 2)$ und $\mathbf{x} = (1, 3)$.

Grundidee von Schnittebenenverfahren

1. Bestimme die optimale Lösung $\mathbf{x}'$ der LP-Relaxation.
2. Gilt $\mathbf{x}' \in \mathbb{Z}^n$, dann ist $\mathbf{x} := \mathbf{x}'$ eine optimale Lösung des ILP.
3. Ansonsten finde eine lineare Nebenbedingung (Schnittrestriktion), die
   - von allen zulässigen Lösungen des ILP erfüllt wird aber
   - von $\mathbf{x}'$ nicht erfüllt wird.
4. Füge diese Nebenbedingung dem ILP hinzu und gehe zu Schritt 1.

☞ $\mathbf{x}'$ wird aus dem Zulässigkeitsbereich der LP-Relaxation geschnitten.

Veranschaulichung



Begriffe

Schnittebenenverfahren werden auch als Cutting-Plane-Verfahren bezeichnet.

Die zur Schnittrestriktion gehörende Ebene ist die Schnittebene bzw. Cutting-Plane.

Das Verfahren wurde 1958 von dem amerikanischen Informatiker Ralph E. Gomory veröffentlicht.

Deshalb werden die Schnittrestriktionen auch als Gomory-Cuts bezeichnet, bzw. das Verfahren als Verfahren von Gomory.

Voraussetzungen

Gegeben sei ein Maximumproblem mit folgenden zusätzlichen Einschränkungen:

- der Begrenzungsvektor $\mathbf{b}$ ist ganzzahlig,
- die Koeffizienten des Zielfunktionsvektors $\mathbf{c}$ sind ganzzahlig und
- die Koeffizienten der Matrix $\mathbf{A}$ der Nebenbedingungen sind ganzzahlig.

Konsequenz: Schlupfvariablen und Zielfunktionswert sind ebenfalls ganzzahlig.

Beispielhafte Darstellung von Schnittebenenverfahren

Beispiel 3.4

Wir lösen beispielhaft das ILP von Beispiel 3.3. Zunächst lösen wir die LP-Relaxation.
Starttableau:

	x_1	x_2	x_3	x_4	**b**
x_3	0	1	1	0	3
x_4	2	3	0	1	12
z	-1	-2	0	0	0

x_2 ist Pivotspalte, x_3 Pivotzeile

	x_1	x_2	x_3	x_4	**b**
x_2	0	1	1	0	3
x_4	2	0	-3	1	3
z	-1	0	2	0	6

Fortsetzung Beispiel.

x_1 ist Pivotspalte, x_4 Pivotzeile

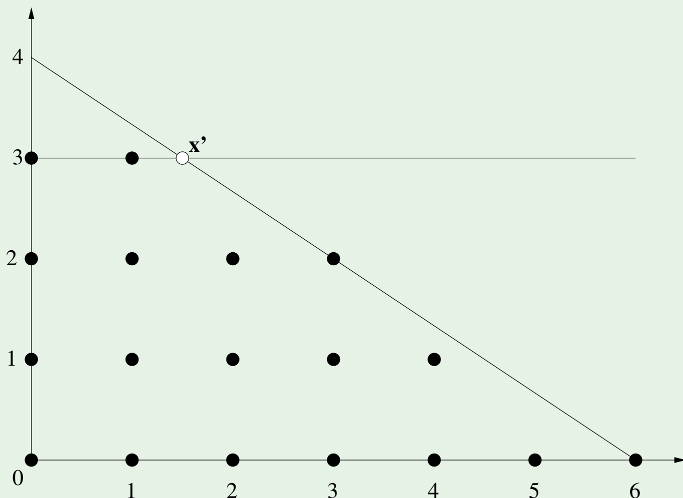
	x_1	x_2	x_3	x_4	$\mathbf{b}$
x_2	0	1	1	0	3
x_1	1	0	$-\frac{3}{2}$	$\frac{1}{2}$	$\frac{3}{2}$
z	0	0	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{15}{2}$

Damit haben wir die optimale Lösung

$$\mathbf{x}' = \begin{pmatrix} \frac{3}{2} \\ 3 \end{pmatrix}$$

der LP-Relaxation ermittelt.

Fortsetzung Beispiel.



Fortsetzung Beispiel.

Wir betrachten die Zeile der nicht-ganzzahligen BV x_1:

$$x_1 - \frac{3}{2}x_3 + \frac{1}{2}x_4 = \frac{3}{2}$$

Wir formen die Gleichung um, indem wir alle Brüche aufspalten in einen ganzzahligen Anteil und einen Bruchanteil aus dem Intervall $(0, 1)$.

$$x_1 + \left(-2 + \frac{1}{2}\right) x_3 + \left(0 + \frac{1}{2}\right) x_4 = \left(1 + \frac{1}{2}\right)$$

Jetzt bringen wir alle ganzzahligen Teile auf die linke und die Bruchteile auf die rechte Seite:

$$x_1 - 2x_3 - 1 = -\frac{1}{2}x_3 - \frac{1}{2}x_4 + \frac{1}{2} \quad (*)$$

Fortsetzung Beispiel.

Wegen $x_3, x_4 \geq 0$ folgt $-\frac{1}{2}x_3 - \frac{1}{2}x_4 \leq 0$ und damit die Ungleichung

$$-\frac{1}{2}x_3 - \frac{1}{2}x_4 + \frac{1}{2} \leq \frac{1}{2} < 1$$

Andererseits muss wegen $(*)$ die linke Seite der Ungleichung ganzzahlig sein. Daher können wir die Ungleichung verschärfen zu

$$-\frac{1}{2}x_3 - \frac{1}{2}x_4 + \frac{1}{2} \leq 0$$

bzw.

$$-\frac{1}{2}x_3 - \frac{1}{2}x_4 \leq -\frac{1}{2} \quad (**)$$

Diese Ungleichung wird von $\mathbf{x}'$ nicht erfüllt, aber von jeder ganzzahligen zulässigen Lösung.

Fortsetzung Beispiel.

Mit Hilfe der Schlupfvariablen x_5 fügen wir die Ungleichung $(**)$ dem ILP hinzu. Das neue Tableau lautet:

	x_1	x_2	x_3	x_4	x_5	**b**
x_2	0	1	1	0	0	3
x_1	1	0	$-\frac{3}{2}$	$\frac{1}{2}$	0	$\frac{3}{2}$
x_5	0	0	$-\frac{1}{2}$	$-\frac{1}{2}$	1	$-\frac{1}{2}$
z	0	0	$\frac{1}{2}$	$\frac{1}{2}$	0	$\frac{15}{2}$

Dieses Tableau ist nicht primal zulässig, aber dual: Pivotzeile x_5, Pivotspalte x_3

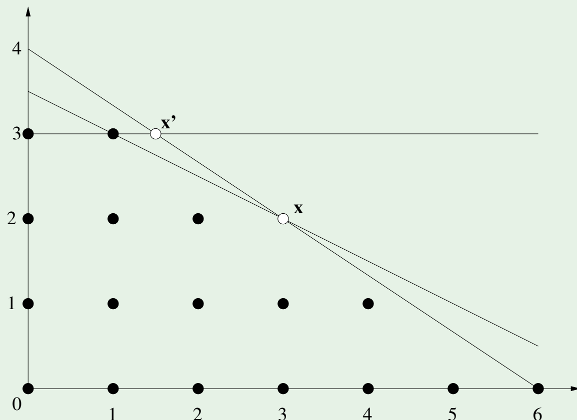
	x_1	x_2	x_3	x_4	x_5	**b**
x_2	0	1	0	-1	2	2
x_1	1	0	0	2	-3	3
x_3	0	0	1	1	-2	1
z	0	0	0	0	1	7

Fortsetzung Beispiel.

Damit haben wir eine optimale Lösung **x** für das ILP.

In Strukturvariablen ausgedrückt lautet die Schnittrestriktion (Herleitung ✎) $x_1 + 2x_2 \leq 7$.

Visualisierung:



Ausgangssituation

Ausgangslage: Wir haben die LP-Relaxation mit dem primalen Simplexverfahren gelöst, das zugehörige Tableau liegt vor.

Bezeichungen:

- a_{ij}^r: Koeffizienten im Tableau
- b_i^r: Komponenten des Begrenzungsvektors im Tableau
- x_i^r: Variablenwerte

Für jede BV x_i^r gilt

$$x_i^r + \sum_{j \in I_{NBV}} a_{ij}^r x_j^r = b_i^r$$

Man beachte:

- I_{NBV} ist die Menge der Indizes der NBVs,
- $a^r_{ii} = 1$,
- $a^r_{ij} = 0$, falls x^r_j BV und
- $x^r_j = 0$, falls x^r_j NBV.

Daher insgesamt (wie bekannt)

$$x^r_i = b^r_i$$

Konstruktion von Schnittebenen

Definition 3.5

Für $a \in \mathbb{R}$ bezeichne $\lfloor a \rfloor \in \mathbb{Z}$ die ganze Zahl, für die

$$a - 1 < \lfloor a \rfloor \leq a$$

gilt. $\lfloor a \rfloor$ heißt das größte Ganze von a.

Wir zerlegen nun die Koeffizienten b_i^r bzw. a_{ij}^r in

- ganzzahlige Anteile $\lfloor b_i^r \rfloor$ bzw. $\lfloor a_{ij}^r \rfloor$ und
- positive Bruchanteile $\beta_i^r \in (0,1)$ bzw. $\alpha_{ij}^r \in [0,1)$.

Damit gilt

$$\begin{aligned} b_i^r &= \lfloor b_i^r \rfloor + \beta_i^r \\ a_{ij}^r &= \lfloor a_{ij}^r \rfloor + \alpha_{ij}^r \end{aligned}$$

und es folgt mit der Gleichung von Folie 164

$$x_i^r + \sum_{j \in I_{NBV}} \lfloor a_{ij}^r \rfloor x_j^r + \sum_{j \in I_{NBV}} \alpha_{ij}^r x_j^r = \lfloor b_i^r \rfloor + \beta_i^r$$

Fassen wir die ganzzahligen Teile auf der linken und die Brüche auf der rechten Seite zusammen, ergibt sich

$$x_i^r + \sum_{j \in I_{NBV}} \lfloor a_{ij}^r \rfloor x_j^r - \lfloor b_i^r \rfloor = - \sum_{j \in I_{NBV}} \alpha_{ij}^r x_j^r + \beta_i^r$$

Wegen $x_j^r \geq 0$ und $\alpha_{ij}^r \geq 0$ folgt $-\sum_{j \in I_{NBV}} \alpha_{ij}^r x_j^r \leq 0$ und damit

$$-\sum_{j \in I_{NBV}} \alpha_{ij}^r x_j^r + \beta_i^r \leq \beta_i^r < 1$$

Andererseits muss die linke Seite dieser Ungleichung ganzzahlig sein, daher kann die Ungleichung zu ≤ 0 verschärft werden und es folgt die Schnittrestriktion:

$$-\sum_{j \in I_{NBV}} \alpha_{ij}^r x_j^r \leq -\beta_i^r$$

Als Gleichung mit zusätzlicher Schlupfvariable $r_i \geq 0$ ergibt sich für die Schnittrestriktion:

$$-\sum_{j \in I_{NBV}} \alpha_{ij}^r x_j^r + r_i = -\beta_i^r$$

Beispiel: Herleitung von Schnittebenen

Beispiel 3.6

Wir wollen das folgende ILP lösen:

$$\max x_1 + 2x_2$$

unter den Neben- und Vorzeichenbedingungen:

$$\begin{array}{rcrcr} 6x_1 & + & 5x_2 & \leq & 30 \\ 4x_1 & + & 9x_2 & \leq & 36 \\ & & x_1, x_2 & \geq & 0 \\ & & x_1, x_2 & \in & \mathbb{Z} \end{array}$$

Fortsetzung Beispiel.

Lösen der LP-Relaxation:

	x_1	x_2	x_3	x_4	**b**
x_3	6	5	1	0	30
x_4	4	9	0	1	36
z	-1	-2	0	0	0

	x_1	x_2	x_3	x_4	**b**
x_3	$34/9$	0	1	$-5/9$	10
x_2	$4/9$	1	0	$1/9$	4
z	$-1/9$	0	0	$2/9$	8

	x_1	x_2	x_3	x_4	**b**
x_1	1	0	$9/34$	$-5/34$	$45/17$
x_2	0	1	$-2/17$	$3/17$	$48/17$
z	0	0	$1/34$	$7/34$	$141/17$

Fortsetzung Beispiel.

Lösung ist nicht ganzzahlig. Mögliche Schnittrestriktionen:

1.

$$x_1 + \frac{9}{34}x_3 - \frac{5}{34}x_4 = \frac{45}{17}$$

$$\Rightarrow \quad x_1 + \left(0 + \frac{9}{34}\right)x_3 + \left(-1 + \frac{29}{34}\right)x_4 = \left(2 + \frac{11}{17}\right)$$

Daraus ergibt sich die Schnittrestriktion

$$-\frac{9}{34}x_3 - \frac{29}{34}x_4 \leq -\frac{11}{17}$$

bzw. als Gleichung mit zusätzlicher Schlupfvariable r_1

$$-\frac{9}{34}x_3 - \frac{29}{34}x_4 + r_1 = -\frac{11}{17}$$

Fortsetzung Beispiel.

2. Für die x_2-Zeile ergibt sich analog

$$-\frac{15}{17}x_3 - \frac{3}{17}x_4 \leq -\frac{14}{17} \quad \text{bzw.} \quad -\frac{15}{17}x_3 - \frac{3}{17}x_4 + r_2 = -\frac{14}{17}$$

3. Auch aus der Zielfunktionszeile lässt sich eine Schnittrestriktion herleiten:

$$-\frac{1}{34}x_3 - \frac{7}{34}x_4 \leq -\frac{5}{17} \quad \text{bzw.} \quad -\frac{1}{34}x_3 - \frac{7}{34}x_4 + r_3 = -\frac{5}{17}$$

Eigenschaften

- Die bisher optimale Lösung ist wegen

$$r_i = \lfloor b_i^r \rfloor - b_i^r < 0$$

  nicht mehr zulässig.
- Alle anderen ganzzahligen zulässigen Punkte erfüllen dagegen die zusätzliche Ungleichung.
- Wir nehmen die zusätzliche Ungleichung zum Tableau hinzu. Dadurch entsteht auch eine neue Schlupfvariable.
- Wegen der negativen rechten Seite ist das entstehende Tableau nicht mehr primal, aber dual zulässig.
- Durch Anwendung des dualen Simplexalgorithmus erhalten wir eine optimale Lösung für die Relaxation mit zusätzlicher Ungleichung.

Auswahl einer Schnittrestriktion

- Prinzipiell könnten wir alle in Beispiel 3.6 hergeleiteten Schnittrestriktionen dem Tableau hinzufügen.
- Nachteil: Jeweils auch eine zusätzliche Schlupfvariable. Tableau wird größer, je mehr Schnittrestriktionen hinzugenommen werden.
- Deshalb: Auswahl genau einer Schnittrestriktion
- Welche?
- Auswahl kann entscheidend für die Effizienz sein.

Effizienz für verschiedene Schnittrestriktionen

Beispiel 3.7

Wir setzen Beispiel 3.6 fort.

Wir wählen die dritte Schnittrestriktion von Beispiel 3.6. Damit entsteht:

	x_1	x_2	x_3	x_4	r_3	**b**
x_1	1	0	$9/34$	$-5/34$	0	$45/17$
x_2	0	1	$-2/17$	$3/17$	0	$48/17$
r_3	0	0	$-1/34$	$-7/34$	1	$-5/17$
z	0	0	$1/34$	$7/34$	0	$141/17$

Dual zulässig, r_3 ist Pivotzeile, wähle x_3 als Pivotspalte. Es entsteht:

Fortsetzung Beispiel.

	x_1	x_2	x_3	x_4	r_3	**b**
x_1	1	0	0	-1	9	0
x_2	0	1	0	1	-4	4
x_3	0	0	1	7	-34	10
z	0	0	0	0	1	8

Damit haben wir eine optimale Lösung für das ILP.

Wir nehmen stattdessen die zweite Schnittrestriktion von Beispiel 3.6. Damit entsteht:

	x_1	x_2	x_3	x_4	r_2	**b**
x_1	1	0	$9/34$	$-5/34$	0	$45/17$
x_2	0	1	$-2/17$	$3/17$	0	$48/17$
r_2	0	0	$-15/17$	$-3/17$	1	$-14/17$
z	0	0	$1/34$	$7/34$	0	$141/17$

Dual zulässig, r_2 ist Pivotzeile, x_3 Pivotspalte. Es entsteht:

Fortsetzung Beispiel.

	x_1	x_2	x_3	x_4	r_2	**b**
x_1	1	0	0	$-1/5$	$3/10$	$12/5$
x_2	0	1	0	$1/5$	$-2/15$	$44/15$
x_3	0	0	1	$1/5$	$-17/15$	$14/15$
z	0	0	0	$1/5$	$1/30$	$124/15$

Nicht ganzzahlig. Wir müssten jetzt weitere Schnittrestriktionen herleiten.

Auswahlkriterium

☞ Wähle die Schnittrestriktion, deren Schnittebene den größten Abstand vom nicht-ganzzahligen Optimum hat.

Ebenendarstellung: Für $\mathbf{d} \in \mathbb{R}^n, \mathbf{d} \neq \mathbf{0}$ und $e \in \mathbb{R}$ beschreibt die Menge

$$E = \left\{ \mathbf{x} \in \mathbb{R}^n \,|\, \mathbf{d}^T \mathbf{x} + e = 0 \right\}$$

eine Ebene E des $\mathbb{R}^n$.

Der Vektor $\mathbf{d}$ ist ein Normalenvektor für die Ebene E, die Gleichung ist eine Normalengleichung für E.

Normierte Ebenendarstellung: Indem wir die Normalengleichung durch $\|d\|$ teilen, erhalten wir die Hessesche Normalform für eine Ebene:

$$E = \left\{ \mathbf{x} \in \mathbb{R}^n \mid \frac{1}{\|d\|} \cdot \mathbf{d}^T \mathbf{x} + \frac{e}{\|d\|} = 0 \right\}$$

Der Vektor $\frac{1}{\|d\|} \cdot \mathbf{d}$ heißt Normaleneinheitsvektor.

Abstand Punkt zu Ebene: Der Abstand eines Punktes $\mathbf{p} \in \mathbb{R}^n$ zu einer Ebene E ergibt sich, indem man $\mathbf{p}$ in die Gleichung der Hesseschen Normalenform für E einsetzt:

$$\text{distance}(p, E) = \frac{1}{\|d\|} \cdot \left| \mathbf{d}^T \mathbf{p} + e \right|$$

Beispiel: Auswahlkriterium

Beispiel 3.8

Wir führen die Berechnung für die drei Schnittebenen von Beispiel 3.6 durch. Wir haben

$$\mathbf{p} = \frac{1}{17} \begin{pmatrix} 45 \\ 48 \\ 0 \\ 0 \end{pmatrix}$$

1. Ebenengleichung:

$$-\frac{9}{34}x_3 - \frac{29}{34}x_4 + \frac{11}{17} = 0$$

   und damit

$$\text{distance}(p, E) = \frac{34}{\sqrt{9^2 + 29^2}} \cdot \frac{11}{17} = 0.7245\ldots$$

Fortsetzung Beispiel.

2. Ebenengleichung:

$$-\frac{15}{17}x_3 - \frac{3}{17}x_4 + \frac{14}{17} = 0$$

und damit

$$\text{distance}(p, E) = \frac{17}{\sqrt{15^2 + 3^2}} \cdot \frac{14}{17} = 0.9152\ldots$$

3. Ebenengleichung:

$$-\frac{1}{34}x_3 - \frac{7}{34}x_4 + \frac{5}{17} = 0$$

und damit

$$\text{distance}(p, E) = \frac{34}{\sqrt{1^2 + 7^2}} \cdot \frac{5}{17} = \sqrt{2} = 1.4142\ldots$$

Also wählen wir die 3. Schnittrestriktion wie in Beispiel 3.7.

Spezielle Schnittrestriktionen

- Das bisher betrachtete Verfahren zur Erzeugung von Schnittebenen ist allgemeiner Natur: Es kann für jedes ganzzahlige Problem eingesetzt werden.
- Solche problemunabhängigen Schnittebenen sind häufig nicht besonders leistungsfähig.
- Wir leiten in diesem Abschnitt mehrere Klassen von Schnittebenen her, die für bestimmte Probleme besonders geeignet sind.

Prinzipieller Ansatz

- Wir betrachten ein ILP mit ganzzahliger Matrix $\mathbf{A} \in \mathbb{Z}^{m \times n}$ und ganzzahligem Begrenzungsvektor $\mathbf{b} \in \mathbb{Z}^m$.
  Nebenbedingungen: $\mathbf{Ax} \leq \mathbf{b}$
- Es seien $\mu_1, \ldots, \mu_m \in \mathbb{Q}$.
- Wir bilden mit den μ_i eine Linearkombination der Zeilen von $\mathbf{Ax} \leq \mathbf{b}$. Es entsteht die Ungleichung

$$\langle \mu, \mathbf{a}^{(1)} \rangle x_1 + \cdots + \langle \mu, \mathbf{a}^{(n)} \rangle x_n \leq \langle \mu, \mathbf{b} \rangle.$$

- Gilt nun
  - $\langle \mu, \mathbf{a}^{(i)} \rangle \in \mathbb{Z}$ für $i = 1, \ldots, n$, aber
  - $\langle \mu, \mathbf{b} \rangle \notin \mathbb{Z}$,

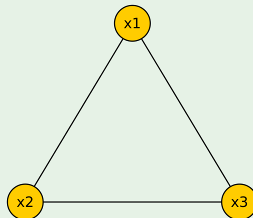
  dann können wir die Ungleichung verschärfen auf

$$\langle \mu, \mathbf{a}^{(1)} \rangle x_1 + \cdots + \langle \mu, \mathbf{a}^{(n)} \rangle x_n \leq \lfloor \langle \mu, \mathbf{b} \rangle \rfloor.$$

Anwendung

Beispiel 3.9

Wir betrachten Independent Set als Optimierungsproblem für den folgenden Graphen:



Damit haben wir die Zielfunktion $\max x_1 + x_2 + x_3$ u. d. N.:

$$
\begin{array}{ccccccc}
x_1 & + & x_2 & & & \leq & 1 \\
x_1 & & & + & x_3 & \leq & 1 \\
 & & x_2 & + & x_3 & \leq & 1
\end{array}
$$

Welche Lösung ist optimal für die LP-Relaxation $(0 \leq x_i \leq 1)$?

Fortsetzung Beispiel.

Optimal für die Relaxation ist $x_1 = x_2 = x_3 = \frac{1}{2}$ mit Zielfunktionswert $\frac{3}{2}$.

Mit $\mu_1 = \mu_2 = \mu_3 = \frac{1}{2}$ erhalten wir die Ungleichung

$$x_1 + x_2 + x_3 \leq \frac{3}{2}.$$

Diese können wir verschärfen zu

$$x_1 + x_2 + x_3 \leq 1.$$

Die optimale Lösung der Relaxation erfüllt diese Ungleichung nicht.

Nehmen wir diese Ungleichung zur LP-Relaxation hinzu, erhalten wir eine optimale Lösung für das ganzzahlige Problem.

Gültige Ungleichung

Definition 3.10

Eine lineare Ungleichung (bzw. Gleichung), die von allen zulässigen Lösungen eines kombinatorischen Optimierungsproblems erfüllt wird, heißt gültige Ungleichung (bzw. gültige Gleichung) für dieses Problem.

Bemerkung:

- Die Identifikation von gültigen Ungleichungen für ein Problem kann sehr hilfeich für eine effizientere Behandlung sein.
- Für eine Reihe von Packungs- und Überdeckungsprobleme liefert der Ansatz aus Beispiel 3.9 eine wichtige Klasse von gültigen Ungleichungen.

Odd-Cycle-Ungleichungen (1)

- Wir verallgemeinern den Ansatz aus Beispiel 3.9.
- Wir betrachten Independent Set auf einem Graphen $G = (V, E)$ mit $V = \{x_1, \ldots, x_n\}$.
- Es sei $C = (x_{i_1}, x_{i_2}, \ldots, x_{i_k}, x_{i_1})$ ein Kreis ungerader Länge $|C|$ in G.
- Wir setzen

$$\mu_i = \begin{cases} \frac{1}{2} & \text{für } i \in \{i_1, \ldots, i_k\} \\ 0 & \text{sonst} \end{cases}$$

  und erhalten damit die gültige Ungleichung

$$\sum_{j=1}^{k} x_{i_j} =: x(C) \leq \frac{|C| - 1}{2}.$$

- Hierbei bezeichnet $x(C)$ die Summe aller Variablen/Knoten, die an C beteiligt sind.

Odd-Cycle-Ungleichungen (2)

Folgerung 3.11

Es sei $G = (V, E)$ ein Graph und C ein Kreis ungerader Länge in G

Die Ungleichung

$$x(C) \leq \frac{|C| - 1}{2}$$

ist eine gültige Ungleichung für Independent Set.

Die Ungleichung

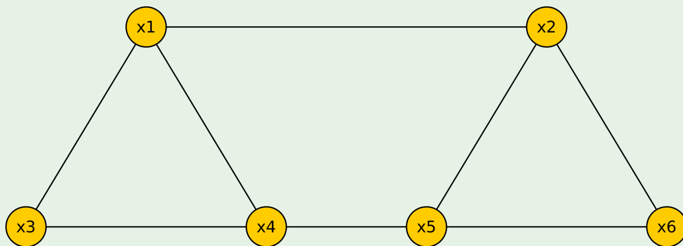
$$x(C) \geq \frac{|C| + 1}{2}$$

ist eine gültige Ungleichung für das Knotenüberdeckungsproblem (Vertex Cover).

VC mit Odd-Cycle-Inequality

Beispiel 3.12

Wir betrachten das Knotenüberdeckungsproblem für den folgenden Graphen.



Wir Lösen dieses Problem mithilfe von Odd-Cycle-Ungleichungen. ✎

Separationsproblem

Definition 3.13

Gegeben sei

- eine optimale Lösung $\mathbf{x}'$ einer LP-Relaxation für ein kombinatorisches Optimierungsproblem Π,
- aber $\mathbf{x}'$ sei nicht zulässig für Π.

Das Separationsproblem für eine Klasse K von gültigen Ungleichungen lautet dann:

- Man finde eine Ungleichung aus K, die von $\mathbf{x}'$ verletzt wird oder
- man zeige, dass keine solche Ungleichung existiert.

Bemerkung: Die gesuchte Ungleichung soll $\mathbf{x}'$ von der konvexen Hülle der zulässigen Lösungen separieren.

Diskussion Separationsproblem

- Häufig kann man das Separationsproblem nicht lösen.
- Dann finden wir innerhalb der Klasse K keine gültige Ungleichung, die verletzt ist.
- Dann müssen wir andere Klassen von Ungleichungen betrachten (falls bekannt) oder anders verfahren (siehe folgendes Kapitel).
- Außerdem können wir nicht erwarten, dass für alle Klassen K das Separationsproblem in polynomieller Zeit gelöst werden kann.
- In diesem Fall können auch Heuristiken für die Separation zum Einsatz kommen.

Schnittebenenverfahren mit Separation

Algorithmus 3.14

1. *Man löse die LP-Relaxation eines kombinatorischen Optimierungsproblems.*

   Sei $\mathbf{x}'$ *die optimale Lösung der Relaxation.*
2. *Separation: Man finde eine Klasse* K *von gültigen Ungleichungen und eine Ungleichung* U *aus* K*, die von* $\mathbf{x}'$ *verletzt wird.*
3. *Findet man in Schritt 2 keine verletzte Ungleichung, kann man eine Ungleichung* U *wie auf Folie 166 ff. beschrieben konstruieren.*
4. *Man erweitere die LP-Relaxation um die Ungleichung* U*.*

   Gehe zu Schritt 1.

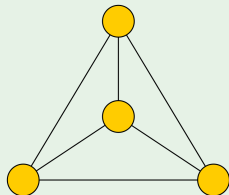
Wheel

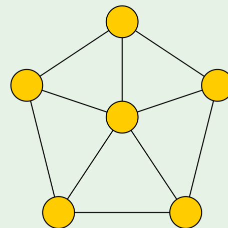
Definition 3.15

Ein Graph, der aus einem Kreis C und einem Knoten $v \notin C$ besteht, so dass v mit allen Knoten aus C adjazent ist, heißt Wheel-Graph.

Beispiel 3.16

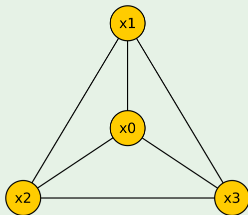
Wheel-Graph mit vier bzw. sechs Knoten.





Beispiel 3.17

Wir betrachten Independent Set auf dem linken Graphen von Beispiel 3.16.



Welche Lösung ist optimal für die LP-Relaxation?

Helfen hier Odd-Cycle-Ungleichungen weiter?

Wenn x_0, dann kein Knoten aus dem Kreis (x_1, x_2, x_3) und wenn nicht x_0, dann aus dem Kreis maximal ein Knoten. Also ist

$$x_0 + x_1 + x_2 + x_3 \leq 1.$$

eine gültige Ungleichung.

Wheel-Ungleichungen

Folgerung 3.18

Es sei $G = (V, E)$ ein Graph, W ein Wheel-Graph in G, der aus einem Kreis C ungerader Länge und einem Knoten x_0 besteht.
Dann ist die Ungleichung

$$\frac{|C|-1}{2} x_0 + x(C) \leq \frac{|C|-1}{2}$$

eine gültige Ungleichung für Independent Set.

Die Ungleichung

$$\frac{|C|-1}{2} x_0 + x(C) \geq |C|$$

ist eine gültige Ungleichung für das Knotenüberdeckungsproblem.

Rucksackproblem

Wir betrachten das Rucksackproblem in der Optimierungsvariante.

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

u. d. N.

$$\sum_{j=1}^{n} w_j x_j \leq C$$

und $x_j \in \{0, 1\}$.

Lösung einer Rucksack-Relaxation

Beispiel 3.19

$$\begin{array}{rcl} n & = & 7, \\ (p_j) & = & (70, \quad 20, \quad 39, \quad 37, \quad 7, \quad 5, \quad 10), \\ (w_j) & = & (31, \quad 10, \quad 20, \quad 19, \quad 4, \quad 3, \quad 6), \\ C & = & 50 \end{array}$$

Optimale Lösung der Relaxation:

$$x_1 = x_2 = 1, x_3 = \frac{9}{20}, x_4 = x_5 = x_6 = x_7 = 0$$

Knapsack-Ungleichungen

- Gegeben sei ein Rucksackproblem mit n Gegenständen.
- Als Knapsack-Cover bezeichnen wir eine Indexmenge $O \subseteq \{1, \ldots, n\}$ für die gilt:

$$\sum_{j \in O} w_j > C$$

- Ein Knapsack-Cover O heißt minimal, wenn

$$\sum_{j \in O \setminus \{j'\}} w_j \leq C$$

  für alle $j' \in O$ gilt.
- Wenn O ein Knapsack-Cover ist, dann ist

$$\sum_{j \in O} x_i \leq |O| - 1$$

  eine gültige Ungleichung.

Beispiel 3.20

Wir konstruieren für Beispiel 3.19 einige Knapsack-Cover und die zugehörigen Ungleichungen:

$$\begin{aligned} x_1 + x_2 + x_3 &\leq 2 \\ x_1 + x_2 + x_4 &\leq 2 \\ x_1 + x_2 + x_4 + x_6 &\leq 3 \\ x_1 + x_4 + x_5 &\leq 2 \end{aligned}$$

Die optimale Lösung der Relaxation verletzt die erste Ungleichung.

Wir lösen das Rucksackproblem mittels Separation, also indem wir schrittweise weitere Knapsack-Ungleichungen hinzufügen. ✎

Die optimale Lösung ist

$$x_1 = x_4 = 1, \quad x_2 = x_3 = x_5 = x_6 = x_7 = 0$$

mit einem Zielfunktionswert von 107.

Erweiterte Knapsack-Ungleichungen (1)

- Es sei O ein Knapsack-Cover.
- Wir definieren das erweiterte Knapsack-Cover $E(O)$ durch

$$E(O) = O \cup \{1 \leq j \leq n | w_j \geq \max_{i \in O} w_i\}$$

- Wenn $E(O)$ ein erweitertes Knapsack-Cover ist, dann ist

$$\sum_{j \in E(O)} x_j \leq |O| - 1$$

  eine gültige Ungleichung.
- Wenn $E(O) \neq O$ gilt, dann ist diese Ungleichung strenger als die ursprüngliche.

Erweiterte Knapsack-Ungleichungen (2)

Beispiel 3.21

Für das Problem aus Beispiel 3.19 ist

$$x_2 + x_3 + x_4 + x_5 \leq 3$$

die Ungleichung zum Knapsack-Cover $O = \{2, 3, 4, 5\}$.

Es gilt $E(O) = \{1, 2, 3, 4, 5\}$. Damit kann die Ungleichung zu

$$x_1 + x_2 + x_3 + x_4 + x_5 \leq 3$$

verschärft werden.

Knapsack-Ungleichungen verwenden

- Wir betrachten ein Binary Program

$$\max \mathbf{c}^T\mathbf{x}, \text{ u. d. N. } \mathbf{A}\mathbf{x} \leq \mathbf{b}, \mathbf{x} \in \{0,1\}^n,$$

  wobei alle Elemente von **A** nicht negativ sind.
- Jede Ungleichung des Problems entspricht einzeln betrachtet einer Einschränkung beim Rucksackproblem.
- Wir können also für jede Ungleichung die Knapsack-Cover-Ungleichungen hinzufügen.

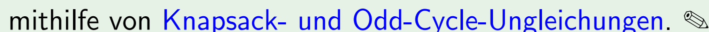
Beispiel 3.22

Wir lösen

$$\max 8x_1 + 6x_2 + 7x_3 + 4x_4$$

u. d. N.

$$\begin{aligned} 11x_1 + 6x_2 + 6x_3 + 5x_4 &\leq 13 \\ 4x_1 + 7x_2 + 6x_3 + 3x_4 &\leq 8 \\ x_1, x_2, x_3, x_4 &\in \{0, 1\} \end{aligned}$$



mithilfe von Knapsack- und Odd-Cycle-Ungleichungen. ✎

Optimale Lösung:

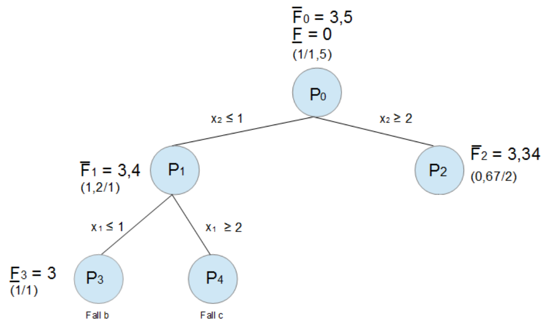
$$x_1 = 1, x_2 = x_3 = x_4 = 0$$

Zusammenfassung

- Mit Schnittebenenverfahren können wir prinzipiell ILPs lösen.
- **Problem:** Tableau und Rechenaufwand wird mit jeder zusätzlichen Schnittebene größer.
- Weitere Probleme: numerische Instabilität, Terminierung
- Spezielle Klassen von Ungleichungen:
  - Odd-Cycle-Ungleichungen
  - Wheel-Ungleichungen
  - Knapsack-Ungleichungen
- **Ausblick:** Kombination solcher Schnittebenen mit Verfahren des nächsten Kapitels

Kapitel 4

Branch-and-Bound



Inhalt

4 Branch-and-Bound

- Branch-and-Bound
- Anwendungsbeispiele

Schranken (1)

Definition 4.1

Gegeben sei ein Maximierungsproblem der Art

$$\max F(\mathbf{x}), \quad \text{u.d.N. } \mathbf{x} \in \mathcal{X}$$

Gilt

$$F(\mathbf{x}) \leq B_{up} \text{ für alle } \mathbf{x} \in \mathcal{X}$$

dann ist B_{up} eine obere Schranke für den optimalen Zielfunktionswert.

Gilt

$$F(\mathbf{x}) \geq B_{low} \text{ für ein } \mathbf{x} \in \mathcal{X}$$

dann ist B_{low} eine untere Schranke für den optimalen Zielfunktionswert.

Schranken (2)

Definition 4.2

Gegeben sein ein Minimierungsproblem der Art

$$\min F(\mathbf{x}), \quad \text{u.d.N. } \mathbf{x} \in \mathcal{X}$$

Gilt

$$F(\mathbf{x}) \geq B_{low} \text{ für alle } \mathbf{x} \in \mathcal{X}$$

dann ist B_{low} eine untere Schranke für den optimalen Zielfunktionswert.

Gilt

$$F(\mathbf{x}) \leq B_{up} \text{ für ein } \mathbf{x} \in \mathcal{X}$$

dann ist B_{up} eine obere Schranke für den optimalen Zielfunktionswert.

Herleitung von Schranken

Für Maximierungsprobleme:

- obere Schranken

  typischerweise durch Relaxationen, z.B. LP-Relaxation bei einem ILP
- untere Schranken

  durch zulässige i.d.R. aber nicht optimale Lösungen, z.B. auf der Basis von Heuristiken

Für Minimierungsprobleme: genau umgekehrt

Wenn nicht anders erwähnt betrachten wir im Folgenden stets Maximierungsprobleme.

Grundprinzip von Branch-and-Bound

Suchverfahren, das die Menge $\mathcal{X}$ der zulässigen Lösungen systematisch durchsucht.

Wesentliche Operationen bei der Suche:

- Verzweigung (Branch)

  Teile das Ausgangsproblem P_0 (bzw. die Menge $\mathcal{X}(P_0)$ der zulässigen Lösungen) in zwei Teilprobleme P_1 und P_2 (bzw. Teilmengen $\mathcal{X}(P_1)$ und $\mathcal{X}(P_2)$) auf.

- Beschränkung (Bound)

  Berechne für die Teilprobleme P_1 und P_2 obere Schranken (upper bound) B_{up}^1 und B_{up}^2 für die optimale Lösung in $\mathcal{X}(P_1)$ und in $\mathcal{X}(P_2)$.

Nutzung von Schranken für die Suche (1)

Es sei B_{low} eine bekannte untere Schranke für $\mathcal{X}(P_0)$, z.B. der Zielfunktionswert einer bekannten zulässigen Lösung **x**.

Dann gilt:

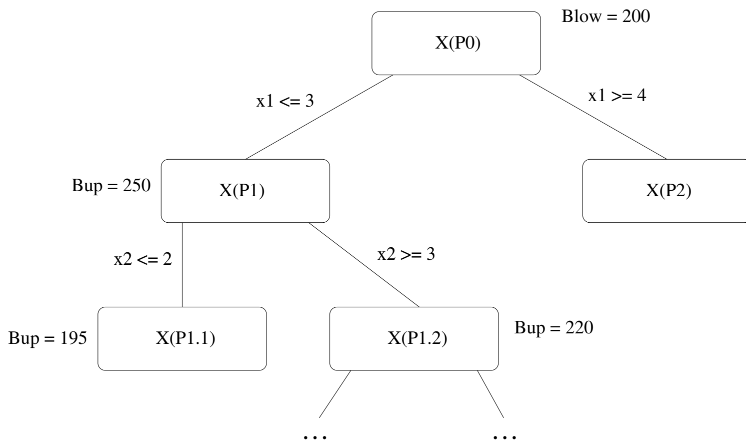
- Aus $B_{up}^1 \leq B_{low}$ folgt, dass $\mathcal{X}(P_1)$ keine bessere Lösung als **x** enthalten kann.
- Analog für $B_{up}^2 \leq B_{low}$

☞ Im Fall von $B_{up}^1 \leq B_{low}$ bzw. $B_{up}^2 \leq B_{low}$ müssen wir in $\mathcal{X}(P_1)$ bzw. $\mathcal{X}(P_2)$ nicht mehr nach einer optimalen Lösung suchen.

Nutzung von Schranken für die Suche (2)

- Es sei B_{up} eine bekannte obere Schranke für $\mathcal{X}$.
- Findet man ein $\mathbf{x} \in \mathcal{X}$ mit $F(\mathbf{x}) = B_{up}$, dann ist $\mathbf{x}$ eine optimale Lösung.

Branch-and-Bound: Prinzipieller Ablauf für ein ILP



Beispiele: Branch-and-Bound für ILP

Beispiel 4.3

Wir wollen das ILP von Beispiel 3.2 lösen. Es sei dies das Problem P_0.

Die LP-Relaxation hat die optimale Lösung $(\frac{9}{4}, \frac{5}{2})$ mit Zielfunktionswert $\frac{29}{4}$. Dies liefert uns eine allgemeine obere Schranke $B_{up} = \lfloor \frac{29}{4} \rfloor = 7$.

Wir unterteilen das Problem P_0 in zwei disjunkte Teilprobleme:

- Für P_1 gelte die zusätzliche Bedingung $x_1 \geq 3$.
- Für P_2 gelte die zusätzliche Bedingung $x_1 \leq 2$.

Für die LP-Relaxation von P_1 erhalten wir die optimale Lösung $\mathbf{x}^1 = (3, 2)$ mit Zielfunktionswert 7.

- Damit haben wir eine zulässige Lösung gefunden, deren Zielfunktionswert mit $B_{up} = 7$ übereinstimmt.
- Somit ist $\mathbf{x}^1$ eine optimale Lösung für P_0.

Fortsetzung Beispiel.

Wir brauchen P_2 nicht mehr zu lösen. Wenn wir trotzdem die zugehörige LP-Relaxation lösen,

- können wir $B_{low} = 7$ setzen, da $\mathbf{x}^1$ eine zulässige Lösung ist und
- wir erhalten $\mathbf{x}^2 = (2, \frac{5}{2})$ mit Zielfunktionswert $B_{up}^2 = 7 \leq B_{low}$ als optimale Lösung für die LP-Relaxation von P_2.
- Hieraus folgt, dass in $\mathcal{X}(P_2)$ keine bessere Lösung als $\mathbf{x}^1$ enthalten sein kann.

Beispiel 4.4

Wir betrachten das ILP

$$\max x_1 + x_2$$

unter den Neben- und Vorzeichenbedingungen

$$\begin{array}{rcrcl} 4x_1 & + & x_2 & \leq & 20 \\ & & 4x_2 & \leq & 10 \\ 2x_1 & + & 3x_2 & \leq & 12 \\ & & x_1, x_2 & \geq & 0 \\ & & x_1, x_2 & \in & \mathbb{Z} \end{array}$$

Tafel

Beispiel 4.5 (Winston, Kapitel 9.3)

Die Telfa Corporation produziert Tische und Stühle.

Für die Produktion eines Tisches werden eine Arbeitsstunde und 9 Quadratmeter Holz benötigt, ein Stuhl erfordert eine Arbeitsstunde und 5 Quadratmeter Holz. Es stehen 6 Arbeitsstunden und 45 Quadratmeter Holz zur Verfügung.

Der Gewinn für einen Tisch beträgt 8 €, für einen Stuhl 5 €.

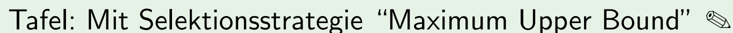
Beispiel 4.6 (Nickel et al., Kapitel 5.4.1)

Gegeben sei das ILP

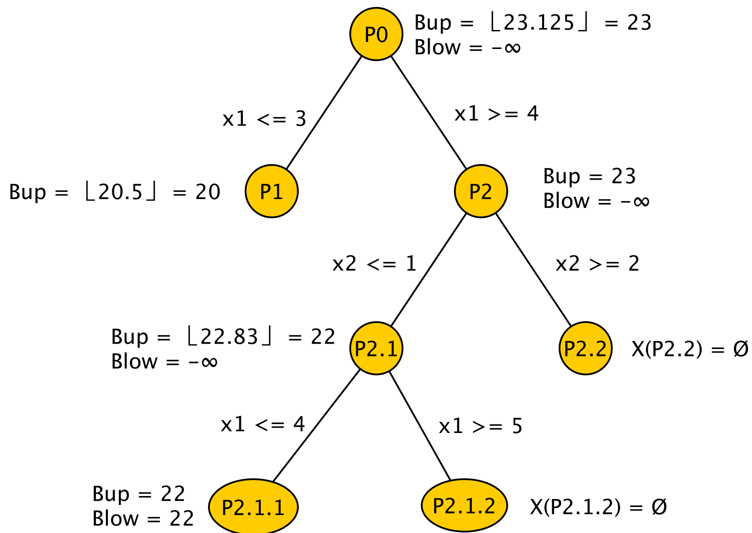
$$\max 5x_1 + 2x_2$$

unter den Neben- und Vorzeichenbedingungen

$$\begin{array}{rcrcl} 6x_1 & + & 2x_2 & \leq & 27 \\ 4x_1 & + & 4x_2 & \leq & 23 \\ & & x_1, x_2 & \geq & 0 \\ & & x_1, x_2 & \in & \mathbb{Z} \end{array}$$



Tafel: Mit Selektionsstrategie "Maximum Upper Bound" ✎


P0
Bup = ⌊23.125⌋ = 23
Blow = −∞
x1 <= 3
x1 >= 4
P1
Bup = ⌊20.5⌋ = 20
P2
Bup = 23
Blow = −∞
x2 <= 1
x2 >= 2
P2.1
Bup = ⌊22.83⌋ = 22
Blow = −∞
P2.2
X(P2.2) = Ø
x1 <= 4
x1 >= 5
P2.1.1
Bup = 22
Blow = 22
P2.1.2
X(P2.1.2) = Ø

Was brauchen wir für Branch-and-Bound?

- Verzweigungsregel: Wie teilen wir die Menge der zulässigen Lösungen auf?
- Obere Schranke: Effizientes Verfahren zur Berechnung einer oberen Schranke für ein Teilproblem, z.B. eine geeignete LP-Relaxation.
- Optionale untere Schranke: Eine Heuristik zur Ermittlung einer guten zulässigen Lösung für ein Teilproblem (üblicherweise auf Basis der optimalen Lösung des relaxierten Problems).
- Selektionsstrategie: In welcher Reihenfolge werden Teilprobleme abgearbeitet? Tiefensuche? Breitensuche? Gesteuert durch die oberen Schranken?

Verzweigung (Branch)

- Ein Problem P_0 wird in k Teilprobleme $P_1, \ldots, P_k$ unterteilt, so dass für die Mengen $\mathcal{X}(P_i)$ der zulässigen Lösungen gilt:

$$\mathcal{X}(P_0) = \bigcup_{i=1}^{k} \mathcal{X}(P_i)$$

  und

$$\mathcal{X}(P_i) \cap \mathcal{X}(P_j) = \emptyset \text{ für } i \neq j$$

- Die Probleme $P_1, \ldots, P_k$ werden falls notwendig weiter unterteilt. Dadurch entsteht ein Baum von Problemen mit P_0 als Wurzel.

Übliche Verzweigungsstrategien

Es sei **x** die optimale Lösung der LP-Relaxation von P_0 und die Komponente $x_i = \alpha$ sei nicht ganzzahlig.

- ILP:
  - P_1 erhält die zusätzliche Nebenbedingung $x_i \leq \lfloor \alpha \rfloor$
  - P_2 erhält die zusätzliche Nebenbedingung $x_i \geq \lceil \alpha \rceil$
- kombinatorische Probleme:
  Setze $x_i = 0$ für P_1 bzw. $x_i = 1$ für P_2.

Welches x_i auswählen? Z.B. das am wenigsten bestimmte.

Obere Schranken

- Für jedes Teilproblem P_i bestimmen wir eine obere Schranke (upper bound) B^i_{up}.
- Hierfür lösen wir eine Relaxation P_i^{relax} von P_i, also ein gegenüber P_i vereinfachtes (relaxiertes) Problem (weniger Nebenbedingungen).
- Es muss gelten
$$\mathcal{X}(P_i) \subseteq \mathcal{X}(P_i^{relax})$$
- Wichtig ist, dass die Relaxationen effizient gelöst werden können.
- Für gewöhnliche ILPs benutzt man üblicherweise LP-Relaxationen.
- Für spezielle kombinatorische Probleme können die Relaxationen vereinfachte und effizient lösbare Probleme sein.

  Beispiel: Minimalgerüst ist Relaxation für kürzesten Hamiltonschen Weg

Untere Schranken

- Durch die heuristische Bestimmung einer zulässigen Lösung erhält man eine untere Schranke B_{low}.
- Prinzipiell nicht notwendig. Schlimmstenfalls starten wir mit $B_{low} = -\infty$.
- Ist die optimale Lösung einer Relaxation auch zulässig für das eigentliche Problem P_0, stellt der zugehörige Zielfunktionswert eine untere Schranke dar.

  Beispiel: Bei einem ILP ist die optimale Lösung der LP-Relaxation ganzzahlig.
- B_{low} ist dann im Laufe des Verfahrens gleich dem Zielfunktionswert der besten bekannten zulässigen Lösung von P_0.

Auslotung eines Problems

Definition 4.7

Ein Problem P_i heißt ausgelotet, wenn einer der folgenden Fälle auftritt:

(a) $B^i_{up} \leq B_{low}$

In $\mathcal{X}(P_i)$ kann es keine bessere Lösung geben als die beste bisher bekannte.

(b) $B^i_{up} > B_{low}$ und die optimale Lösung von P_i^{relax} ist zulässig für P_i.

Dann hat man eine neue bisher beste zulässige Lösung für P_0 gefunden. Man setzt nun $B_{low} := B^i_{up}$.

(c) $\mathcal{X}(P_i^{relax}) = \emptyset$

Dann hat P_i^{relax} und damit auch P_i keine zulässige Lösung.

Selektionsstrategie

- Tiefensuche: Durchsuche für ein noch offenes Teilproblem (Knoten im Suchbaum) zuerst den "linken" Teilbaum, dann den "rechten" Teilbaum

  Vorteile:

  - Man erhält i.d.R. schnell eine zulässige Lösung (und damit eine untere Schranke)
  - geringer Speicherplatzverbrauch (kleine Agenda)

  Nachteile:

  - i.d.R. größerer Suchbaum

- Maximum Upper Bound: Untersuche als nächstes das noch offene Teilproblem P_i mit der größten oberen Schranke B^i_{up}.

  Vorteil:

  - Suchbaum i.d.R. kleiner als bei der Tiefensuche

  Nachteil:

  - Größerer Speicherplatzverbrauch für Agenda

Rucksackproblem

Definition 4.8

Das Optimierungsproblem

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

unter den Neben- und Vorzeichenbedingungen

$$\begin{aligned} \sum_{j=1}^{n} w_j x_j &\leq C \\ x_j &\in \{0,1\} \quad \text{für } j = 1, \ldots, n \end{aligned}$$

heißt Rucksackproblem (knapsack problem, KP).

Bemerkung:

$$
\begin{aligned}
p_j &= \text{Nutzen von Gegenstand } j \\
w_j &= \text{Gewicht von Gegenstand } j \\
C &= \text{Kapazität des Rucksacks} \\
x_j &= \begin{cases} 1 & \text{wenn Gegenstand } j \text{ ausgewählt wird} \\ 0 & \text{sonst} \end{cases}
\end{aligned}
$$

Voraussetzungen (O.B.d.A.):

- p_j, w_j und C sind natürliche Zahlen,
- $\sum_{j=1}^{n} w_j > C$,
- $w_j \leq C$ für alle $j = 1, \ldots, n$.

Wenn nicht anders angegeben gelte außerdem

$$\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq \cdots \geq \frac{p_n}{w_n},$$

d.h., die Gegenstände sind absteigend sortiert nach spezifischem Nutzen.

Relaxation für KP

Die LP-Relaxation von KP ist das stetige Rucksackproblem (continious knapsack problem, CKP):

$$
\begin{aligned}
\max \quad & \sum_{j=1}^{n} p_j x_j \\
\text{u. d. N.} \quad & \sum_{j=1}^{n} w_j x_j \leq C \\
& 0 \leq x_j \leq 1, \quad j = 1, \ldots, n
\end{aligned}
$$

CKP kann effizient gelöst werden, in dem der kritische Gegenstand s ermittelt wird:

$$s = \min\{j : \sum_{i=1}^{j} w_i > C\}$$

Satz 4.9

Die optimale Lösung $\mathbf{x}$ *von CKP ist*

$$\begin{aligned} x_j &= 1 \quad \text{für } j = 1, \ldots, s-1, \\ x_j &= 0 \quad \text{für } j = s+1, \ldots, n, \\ x_s &= \frac{C - \sum_{j=1}^{s-1} w_j}{w_s} \end{aligned}$$

Folgerung 4.10

Der optimale Zielfunktionswert $z(CKP)$ lautet:

$$z(CKP) = \sum_{j=1}^{s-1} p_j + p_s \frac{C - \sum_{j=1}^{s-1} w_j}{w_s}$$

Weiterhin ist

$$U = \lfloor z(CKP) \rfloor = \left\lfloor \sum_{j=1}^{s-1} p_j + p_s \frac{C - \sum_{j=1}^{s-1} w_j}{w_s} \right\rfloor$$

eine obere Schranke für den Zielfunktionswert $z(KP)$ des Rucksackproblems.

Greedy-Algorithmus

Algorithmus 4.11 (GreedyKP)

$z := 0$
for $j := 1$ **to** n **do**
  if $C - w_j \geq 0$ **then**
    $x_j := 1$
    $z := z + p_j$
    $C := C - w_j$
  else
    $x_j := 0$
  end
end

Eigenschaften des Greedy-Algorithmus

- Es sei $z(GKP)$ der Zielfunktionswert der Lösung, die durch GreedyKP ermittelt wird. Dann gilt:

$$z(GKP) \leq z(KP) \leq U \leq z(GKP) + p_s$$

  D.h., der absolute Fehler von GreedyKP ist $\leq p_s$.

- Man betrachte die Folge der KPs mit

$$n = 2, p_1 = w_1 = 1, p_2 = w_2 = k, C = k.$$

  Für $k \to \infty$ gilt $\frac{z(GKP)}{z(KP)} \to 0$, d.h. die relative Güte (worst-case performance ratio) der berechneten Lösung wird beliebig schlecht.

- Verbesserung: Wir vergleichen $z(GKP)$ mit p_s. Es sei

$$z(GKP') = \max\{z(GKP), p_s\}$$

Dann gilt

$$\frac{z(GKP')}{z(KP)} \geq \frac{1}{2}$$

Begründung: Aus $z(KP) \leq z(GKP) + p_s$ folgt $z(KP) \leq 2z(GKP')$.
Damit ist die relative Güte des verbesserten Greedy-Algorithmus nicht mehr beliebig schlecht.

Branch-and-Bound-Algorithmus von Horowitz und Sahni

Algorithmus 4.12 (Horowitz-Sahni (1974))

Allgemeines Vorgehen: Entscheidungen für $j = 1, \ldots, n$ sequentiell treffen.

- Verzweigungsregel: $x_j = 1$ bzw. $x_j = 0$. Wenn nur eine Entscheidung möglich ist, wird nicht verzweigt.
- Obere Schranke: U, unter Berücksichtigung der getroffenen Entscheidungen. Wird nur unmittelbar vor einer Verzweigung berechnet.
- untere Schranke: keine.
- Selektionsstrategie: Tiefensuche, Zweig $x_j = 1$ wird zuerst untersucht.

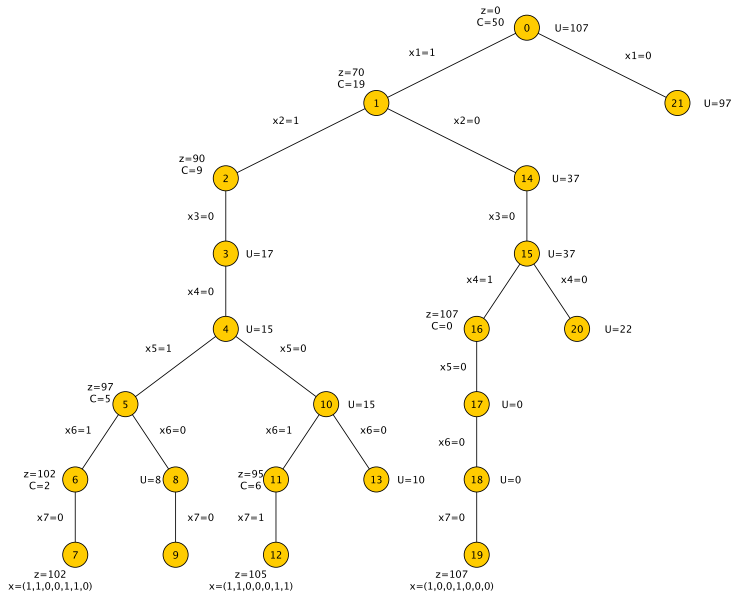
Beispiel zum Algorithmus von Horowitz-Sahni

Beispiel 4.13

$$\begin{array}{rcl} n & = & 7, \\ (p_j) & = & (70, \; 20, \; 39, \; 37, \; 7, \; 5, \; 10), \\ (w_j) & = & (31, \; 10, \; 20, \; 19, \; 4, \; 3, \; 6), \\ C & = & 50 \end{array}$$

In der folgenden Darstellung sind:

- z der aktuelle Zielfunktionswert,
- C die verbleibende Restkapazität,
- U obere Schranke unter Berücksichtigung der bisherigen Entscheidungen,
- $x_j = 0, 1$ Entscheidungen.


z=0
C=50
0
U=107
x1=1
x1=0
z=70
C=19
1
21
U=97
x2=1
x2=0
z=90
C=9
2
14
U=37
x3=0
x3=0
3
U=17
15
U=37
x4=0
x4=1
x4=0
4
U=15
z=107
C=0
16
20
U=22
x5=1
x5=0
x5=0
z=97
C=5
5
10
U=15
17
U=0
x6=1
x6=0
x6=1
x6=0
x6=0
z=102
C=2
6
U=8
8
z=95
C=6
11
13
U=10
18
U=0
x7=0
x7=0
x7=1
x7=0
7
9
12
19
z=102
x=(1,1,0,0,1,1,0)
z=105
x=(1,1,0,0,0,1,1)
z=107
x=(1,0,0,1,0,0,0)

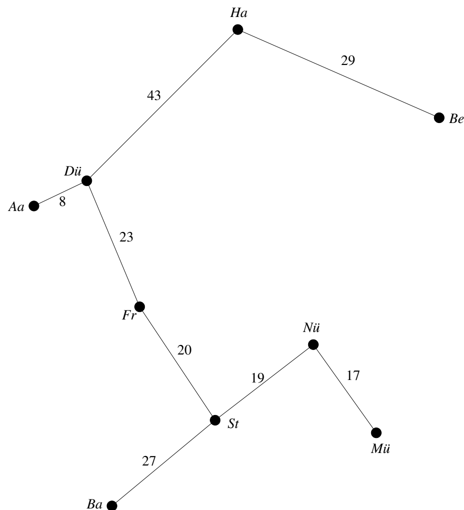
Branch-and-Bound für TSP

Man bestimme eine optimale Lösung für das TSP mit folgender Entfernungsmatrix:

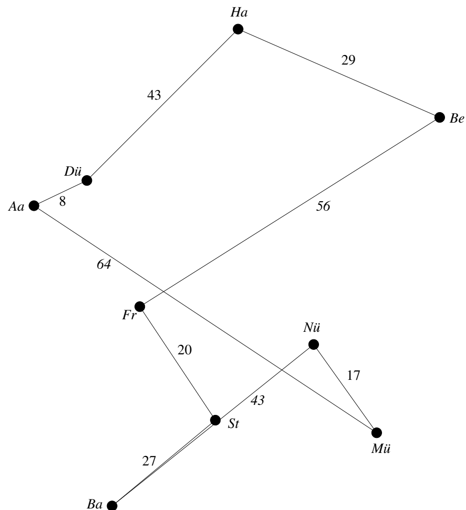
	Aa	Ba	Be	Dü	Fr	Ha	Mü	Nü	St
Aa	0	57	64	8	26	49	64	47	46
Ba	57	0	88	54	34	83	37	43	27
Be	64	88	0	57	56	29	60	44	63
Dü	8	54	57	0	23	43	63	44	41
Fr	26	34	56	23	0	50	40	22	20
Ha	49	83	29	43	50	0	80	63	70
Mü	64	37	60	63	40	80	0	17	22
Nü	47	43	44	44	22	63	17	0	19
St	46	27	63	41	20	70	22	19	0

Obere Schranke: Heuristik

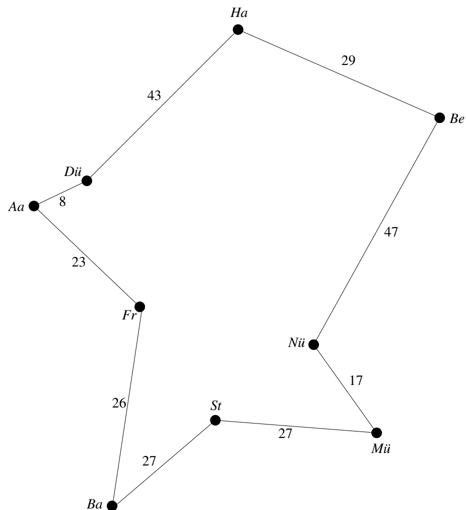
- Berechne Minimalgerüst (MST) mit Länge $z(MST) = 186$.
- $z(TSP) \leq 2 * z(MST) = 372$
- Besser: Aus MST eine möglichst gute zulässige Lösung konstruieren.
- Zur Erinnerung: Tiefensuche auf Minimalgerüst



- Tiefensuche starten in Aachen
- Backtracking
- In der Tour Abkürzungen bei Backtracking
- Erlaubt wegen Dreiecksungleichung
- Länge der Tour (erste obere Schranke): 307



- Weitere Verbesseung durch Kantenaustausch
- 2-opt und 3-opt
- Eliminierung von Kreuzungen
- Länge der Tour (verbesserte obere Schranke): $B_{up} = 250$



Untere Schranke

- Möglichkeit Minimalgerüst, wird hier aber nicht genutzt.
- LP-Relaxation wäre auch möglich, betrachten wir im nächsten Abschnitt.

Bezeichungen:

- Ab jetzt setzen wir in Entfernungsmatrizen für TSP die Diagonalelemente auf ∞.
- Für solch eine Entfernungsmatrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ bezeichne $z(TSP_A)$ die Länge einer optimalen TSP-Tour.
- Es sei $\mathbf{A} \in \mathbb{R}^{n \times n}$ eine Entfernungsmatrix für ein TSP sowie $u_1, \ldots, u_n \in \mathbb{R}$ und $v_1, \ldots, v_n \in \mathbb{R}$ reelle Zahlen. Dann definieren wir die Entfernungsmatrix $\mathbf{B} = (b_{ij})$ durch $b_{ij} = a_{ij} - u_i - v_j$.

Lemma 4.14

$$z(TSP_A) = z(TSP_B) + \sum_{i=1}^{n} u_i + \sum_{j=1}^{n} v_j$$

Beweis.

Jede Stadt wird in einer TSP-Tour genau einmal betreten und wieder verlassen. □

Folgerung 4.15

Gilt $\mathbf{B} \geq \mathbf{0}$, *dann ist*

$$z(TSP_A) \geq \sum_{i=1}^{n} u_i + \sum_{j=1}^{n} v_j$$

Beweis.

Aus $\mathbf{B} \geq \mathbf{0}$ folgt $z(TSP_B) \geq 0$ und mit Lemma 4.14

$$z(TSP_A) \geq \sum_{i=1}^{n} u_i + \sum_{j=1}^{n} v_j.$$

Damit ist

$$B_{low} = \sum_{i=1}^{n} u_i + \sum_{j=1}^{n} v_j$$

eine untere Schranke für $z(TSP_A)$. □

Berechnung einer unteren Schranke:

- Ziehe von jeder Zeile i den Maximalwert u_i ab.
- Ziehe anschließend von jeder Spalte j den Maximalwert v_j ab.

$$B_{low} = 8 + 27 + 29 + 8 + 20 + 29 + 17 + 17 + 19 + 8 + 1 = 183$$

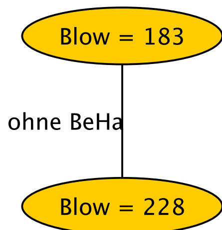
Es entsteht eine neue (jetzt asymmetrische Matrix).

	Aa	Ba	Be	Dü	Fr	Ha	Mü	Nü	St
Aa	∞	41	56	0	17	41	56	39	38
Ba	30	∞	61	27	6	56	10	16	0
Be	35	51	∞	28	26	0	31	15	34
Dü	0	38	49	∞	14	35	55	36	33
Fr	6	6	36	3	∞	30	20	2	0
Ha	20	46	0	14	20	∞	51	34	41
Mü	47	12	43	46	22	63	∞	0	5
Nü	30	18	27	27	4	46	0	∞	2
St	27	0	44	22	0	51	3	0	∞

Verzweigung

- Wir unterteilen eine Lösungsmenge in zwei disjunkte Teilmengen.
- Die erste Teilmenge enthält eine Kante, z.B. Berlin-Hamburg (BeHa).
- Die zweite enthält diese Kante nicht.
- Für die zweite Teilmenge können wir die untere Schranke auf $183 + 30 + 15 = 228$ erhöhen.

Da wir die Kanten als gerichtet betrachten, genügt es hier (aber nur hier), ausschließlich die Teilmenge mit der höheren unteren Schranke zu durchsuchen.

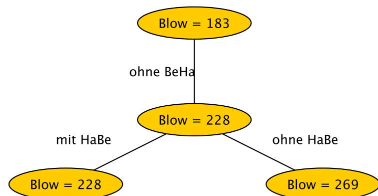


	Aa	Ba	Be	Dü	Fr	Ha	Mü	Nü	St
Aa	∞	41	56	0	17	11	56	39	38
Ba	30	∞	61	27	6	26	10	16	0
Be	20	36	∞	13	11	∞	16	0	19
Dü	0	38	49	∞	14	5	55	36	33
Fr	6	6	36	3	∞	0	20	2	0
Ha	20	46	0	14	20	∞	51	34	41
Mü	47	12	43	46	22	33	∞	0	5
Nü	30	18	27	27	4	16	0	∞	2
St	27	0	44	22	0	21	3	0	∞

Wir unterteilen nun nach HaBe. Für die Lösungsmenge ohne HaBe ergibt sich $B_{low} = 228 + 14 + 27 = 269 \geq B_{up} = 250$.

$\Rightarrow$ Optimale Lösung muss HaBe enthalten.

$\Rightarrow$ Zeile Ha und Spalte Be kann aus der Matrix entfernt werden.

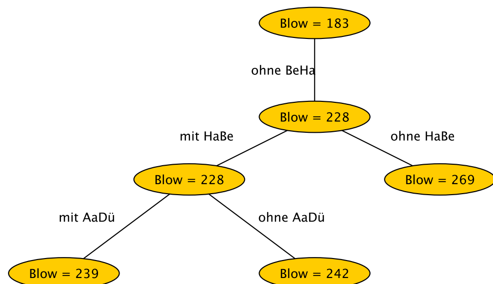


	Aa	Ba	Dü	Fr	Ha	Mü	Nü	St
Aa	∞	41	0	17	11	56	39	38
Ba	30	∞	27	6	26	10	16	0
Be	20	36	13	11	∞	16	0	19
Dü	0	38	∞	14	5	55	36	33
Fr	6	6	3	∞	0	20	2	0
Mü	47	12	46	22	33	∞	0	5
Nü	30	18	27	4	16	0	∞	2
St	27	0	22	0	21	3	0	∞

Als nächstes bietet sich eine Unterteilung auf der Basis von AaDü an.

Ohne AaDü: $B_{low} = 228 + 11 + 3 = 242$

Mit AaDü bedeutet ohne DüAa. Dadurch kann LB für mit AaDü erhöht werden:
$B_{low} = 228 + 5 + 6 = 239$

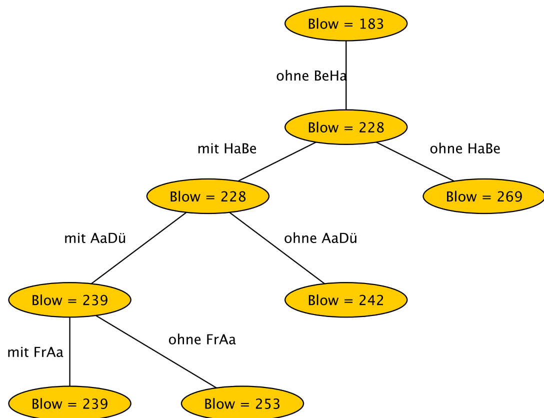


	Aa	Ba	Fr	Ha	Mü	Nü	St
Ba	24	∞	6	26	10	16	0
Be	14	36	11	∞	16	0	19
Dü	∞	33	9	0	50	31	28
Fr	0	6	∞	0	20	2	0
Mü	41	12	22	33	∞	0	5
Nü	24	18	4	16	0	∞	2
St	21	0	0	21	3	0	∞

Als nächstes betrachten wir die Kante FrAa.

Ohne FrAa: $B_{low} = 239 + 14 = 253 \geq 250$. Ausgelotet!

Also mit FrAa, $\Rightarrow$ ohne DüFr.

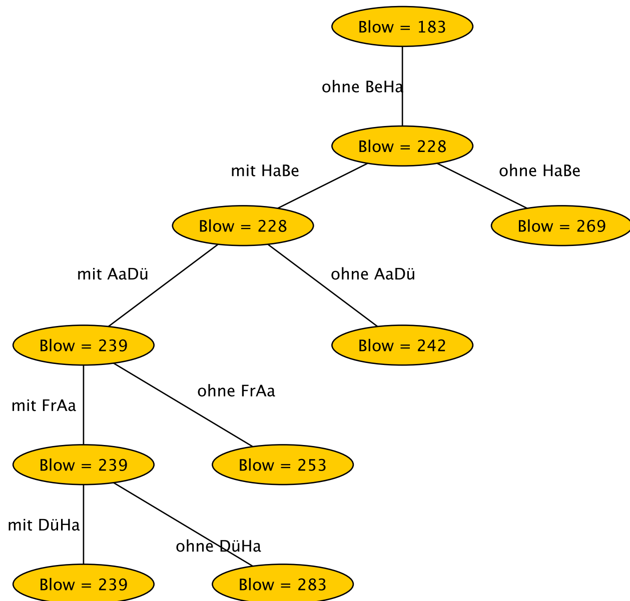


	Ba	Fr	Ha	Mü	Nü	St
Ba	∞	6	26	10	16	0
Be	36	11	∞	16	0	19
Dü	33	∞	0	50	31	28
Mü	12	22	33	∞	0	5
Nü	18	4	16	0	∞	2
St	0	0	21	3	0	∞

Nun die Kante DüHa.

Ohne DüHa: $B_{low} = 239 + 28 + 16 = 283 \geq 250$. Ausgelotet!

Also mit DüHa. Wegen Weg (Fr,Aa,Dü,Ha,Be) folgt ohne BeFr.



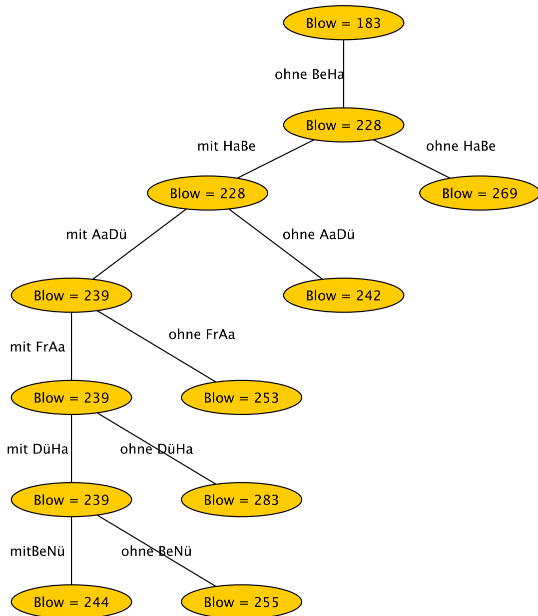
	Ba	Fr	Mü	Nü	St
Ba	∞	6	10	16	0
Be	36	∞	16	0	19
Mü	12	22	∞	0	5
Nü	18	4	0	∞	2
St	0	0	3	0	∞

Nun die Kante BeNü.

Ohne BeNü:
$B_{low} = 239 + 16 = 255 \geq 250$.
Ausgelotet!

Mit BeNü: $B_{low} = 239 + 5 = 244$.

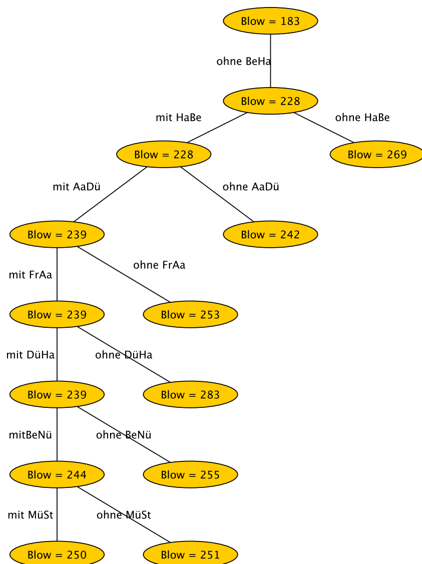
$\Rightarrow$ ohne NüFr.



	Ba	Fr	Mü	St
Ba	∞	6	10	0
Mü	7	17	∞	0
Nü	18	∞	0	2
St	0	0	3	∞

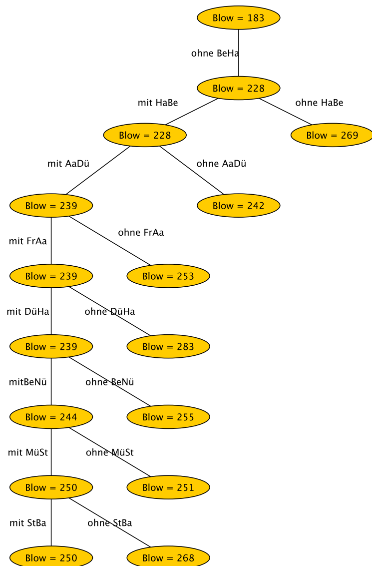
Ohne MüSt:
$B_{low} = 244 + 7 = 251 \geq 250$.
Ausgelotet!

Mit MüSt: $B_{low} = 244 + 6 = 250$.



	Ba	Fr	Mü
Ba	∞	0	4
Nü	18	∞	0
St	0	0	3

Ohne StBa:
$B_{low} = 250 + 18 = 269 \geq 250.$
Ausgelotet!



	Fr	Mü
Ba	0	4
Nü	∞	0

Wähle BaFr und NüMü.

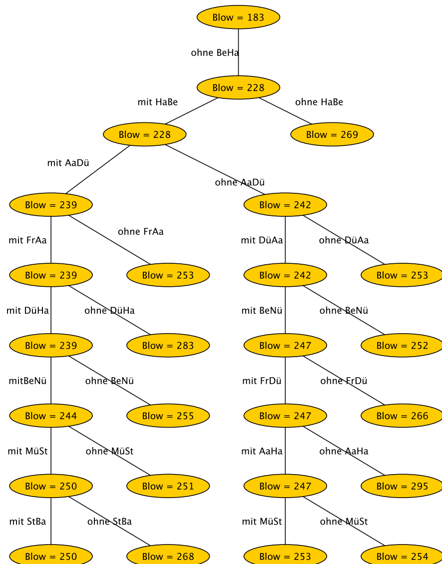
$\Rightarrow$ bekannte Tour mit $B_{up} = 250$.

Wir müssen noch den Zweig “Mit HaBe, ohne AaDü” untersuchens mit $B_{low} = 242$.

Blow = 183
ohne BeHa
Blow = 228
mit HaBe
ohne HaBe
Blow = 228
Blow = 269
mit AaDü
ohne AaDü
Blow = 239
Blow = 242
mit FrAa
ohne FrAa
Blow = 239
Blow = 253
mit DüHa
ohne DüHa
Blow = 239
Blow = 283
mitBeNü
ohne BeNü
Blow = 244
Blow = 255
mit MüSt
ohne MüSt
Blow = 250
Blow = 251
mit StBa
ohne StBa
Blow = 250
Blow = 268

	Aa	Ba	Dü	Fr	Ha	Mü	Nü	St
Aa	∞	30	∞	6	0	45	28	27
Ba	30	∞	24	6	26	10	16	0
Be	20	36	10	11	∞	16	0	19
Dü	0	38	∞	14	5	55	36	33
Fr	6	6	0	∞	0	20	2	0
Mü	47	12	43	22	33	∞	0	5
Nü	30	18	24	4	16	0	∞	2
St	27	0	19	0	21	3	0	∞

Tafel ✎

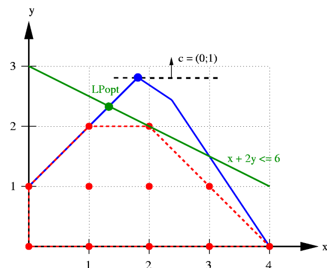

Blow = 183
ohne BeHa
Blow = 228
mit HaBe
ohne HaBe
Blow = 228
Blow = 269
mit AaDü
ohne AaDü
Blow = 239
Blow = 242
mit FrAa
ohne FrAa
mit DüAa
ohne DüAa
Blow = 239
Blow = 253
Blow = 242
Blow = 253
mit DüHa
ohne DüHa
mit BeNü
ohne BeNü
Blow = 239
Blow = 283
Blow = 247
Blow = 252
mitBeNü
ohne BeNü
mit FrDü
ohne FrDü
Blow = 244
Blow = 255
Blow = 247
Blow = 266
mit MüSt
ohne MüSt
mit AaHa
ohne AaHa
Blow = 250
Blow = 251
Blow = 247
Blow = 295
mit StBa
ohne StBa
mit MüSt
ohne MüSt
Blow = 250
Blow = 268
Blow = 253
Blow = 254

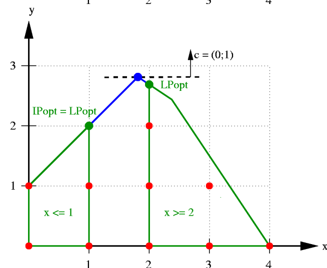
Zusammenfassung

- Branch-and-Bound: Problem aufteilen in Teilprobleme und Teilprobleme lösen
- Bildung der Teilprobleme durch Fixierung von Variablen
- Verwendung von oberen und unteren Schranken, um Teilprobleme möglichst schnell ausloten zu können
- Maximierung: obere Schranken durch Relaxationen, untere durch Heuristiken

Kapitel 5

Branch-and-Cut





Inhalt

5 Branch-and-Cut

- Branch-and-Cut
- Separation für TSP
- Praktischer Einsatz von Branch-and-Cut

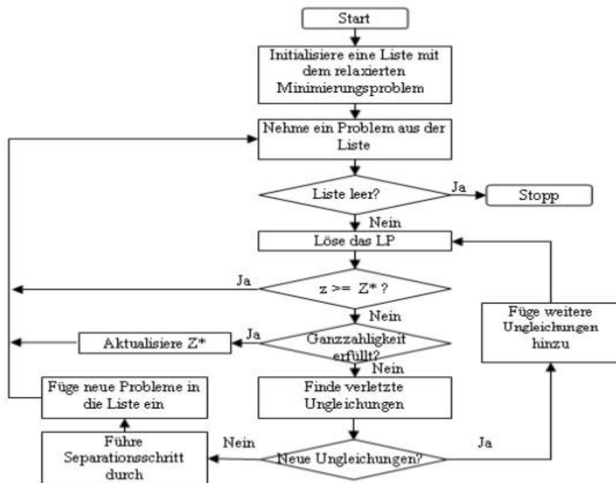
Branch-and-Cut

- Verwende Branch-and-Bound als übergeordneten Algorithmus.
- Nutze LP-Relaxationen zur Bestimmung oberer Schranken für die Teilprobleme P_i.
- Falls Lösung der LP-Relaxation nicht zu einer Auslotung führt:
  - ☞ Versuche durch Schnittebenen die obere Schranke zu verringern.
  - ☞ Im Idealfall führt dies zu einem ausgeloteten Teilproblem P_i.
- Branching erst dann, wenn weitere Schnittebenen "nichts mehr bringen".
- Als Schnittebenen nutzen wir i. d. R. problemspezifische gültige Ungleichungen.

Branch-and-Cut-Algorithmus

Algorithmus 5.1 (für Maximierungsprobleme)

1. $\mathcal{P} := \{$*LP-Relaxation von* $P\}$
   $z^* :=$ *Wert einer heuristischen Lösung (oder* $-\infty$*)*
2. $\mathcal{P} = \emptyset$*? Wenn ja, dann STOP!*
   Wähle $P \in \mathcal{P}$. $\mathcal{P} := \mathcal{P} \setminus \{P\}$
3. *Löse* P*. Sei* z *der zugehörige Zielfunktionswert.*
4. $z \leq z^*$*? Wenn ja, dann gehe zu 2.*
5. *Ist die Lösung von* P *ganzzahlig? Wenn, ja dann aktualisiere eventuell* z^* *und gehe zu 2.*
6. *Suche eine verletzte gültige Ungleichung.*
   Wenn Ungleichung gefunden, dann füge diese P *hinzu und gehe zu 3.*
7. *Wähle aus der Lösung von* P *eine gebrochene Variable* x_i *und erzeuge damit zwei neue Probleme* P_1 *und* P_2.
   Setze $\mathcal{P} := \mathcal{P} \cup \{P_1, P_2\}$ *und gehe zu 2.*


Start
Initialisiere eine Liste mit dem relaxierten Minimierungsproblem
Nehme ein Problem aus der Liste
Liste leer?
Ja
Stopp
Nein
Löse das LP
Ja
z >= Z* ?
Nein
Ja
Aktualisiere Z*
Ganzzahligkeit erfüllt?
Nein
Füge weitere Ungleichungen hinzu
Füge neue Probleme in die Liste ein
Finde verletzte Ungleichungen
Führe Separationsschritt durch
Nein
Neue Ungleichungen?
Ja

Diskussion Branch-and-Cut

- Der Unterschied zum reinen Branch-and-Bound besteht im Schritt 6.
- Hier werden üblicherweise Ungleichungen verwendet, die speziell für eine Problemklasse sind.
- Separationsproblem: Welche gültigen Ungleichungen werden von einer optimalen Lösung einer LP-Relaxation verletzt?
- Die Effizienz eines Branch-and-Cut-Algorithmus wird maßgeblich von der Güte der erkannten verletzten Ungleichungen und der Effizienz der Separation beeinflusst.

Wir untersuchen den Branch-and-Cut-Ansatz am Beispiel des Traveling-Salesman-Problems.

TSP als lineares Programm: Variablen und Zielfunktion

Vollständiger Graph (V, E) mit Knotenmenge $V = \{v_1, \ldots, v_n\}$.

c_{ij} bzw. c_e sei die Länge der Kante $e = \{v_i, v_j\}$.

Wegen Symmetrie gelte stets $i < j$.

$$x_{ij} = x_e = \begin{cases} 1 & \text{Kante } e = \{v_i, v_j\} \text{ ist in Tour enthalten} \\ 0 & \text{sonst} \end{cases}$$

Zielfunktion:

$$\min \sum_{i=1}^{n} \sum_{j=i+1}^{n} c_{ij} x_{ij} \quad \text{bzw.} \quad \min \sum_{e \in E} c_e x_e$$

Menge der Schnittkanten

Definition 5.2

Es sei $G = (V, E)$ ein Graph und $S \subseteq V$. Die Menge

$$\delta(S) = \{e = \{v, w\} \in E | v \in S, w \in V \setminus S\}.$$

heißt Menge der Schnittkanten von S.

Für einen einzelnen Knoten $v \in V$ schreiben wir kurz $\delta(v) := \delta(\{v\})$.

Bemerkung: $\delta(v)$ besteht aus allen Kanten, die inzident zu v sind.

Beispiel 5.3

Beispiele für die Menge der Schnittkanten. ✎

Bezeichnungen für TSP

Definition 5.4

Es sei $G = (V, E)$ ein vollständiger Graph.

Für eine Kantenteilmenge $F \subseteq E$ bezeichne

$$x(F) = \sum_{e \in F} x_e .$$

Für eine Knotenteilmenge $S \subseteq V$ bezeichne

$$x(S) = x(E(S)) \text{ mit } E(S) = \{e = \{v, w\} | v, w \in S\}.$$

Gültige Ungleichungen für das TSP (1)

In einer TSP-Tour ist jeder Knoten inzident mit genau zwei Kanten.

Für $i = 1, \ldots, n$:

$$\sum_{j=1}^{i-1} x_{ji} + \sum_{j=i+1}^{n} x_{ij} = 2,$$

oder kurz: für alle $v \in V$ gilt

$$x(\delta(v)) = 2.$$

Grad-Gleichungen

Gültige Ungleichungen für das TSP (2)

Jede Kante ist an einer TSP-Tour mit einem Gewicht von höchsten 1 beteiligt.

Für $1 \leq i < j \leq n$:

$$0 \leq x_{ij} \leq 1,$$

bzw. für alle $e \in E$ gilt:

$$0 \leq x_e \leq 1.$$

Variablenbeschränkungen (triviale Ungleichungen)

Gültige Ungleichungen für das TSP (3)

Für jede echte Teilmenge $S \subsetneq V$ ist der induzierte Untergraph $T(S)$ einer TSP-Tour T kreisfrei, enthält also höchstens $|S| - 1$ Kanten.

Für $3 \leq |S| \leq n - 1$:

$$\sum_{v_i, v_j \in S, i<j} x_{ij} \leq |S| - 1,$$

oder kurz

$$x(S) \leq |S| - 1.$$

Subtour Elimination Constraints (SEC)

Diskussion Subtour Elimination Constraints

- Alternative Formulierung:

$$x(\delta(S)) \geq 2.$$

- Erste Formulierung besser für „kleine“ S, zweite für „große“ S.
- Es genügt, SECs für $3 \leq |S| \leq \frac{n}{2}$ zu formulieren.
- Problem: Die Anzahl der Subtour Eliminitation Constraints wächst exponentiell in n.
- Lösungsansatz:
  - Beginne nur mit den Grad-Gleichungen als LP-Relaxation.
  - Falls die Lösung der LP-Relaxation eine der gültigen Ungleichungen verletzt, nehme diese zur Relaxation hinzu.
- Mit diesem Ansatz lösen wir nun einige TSP-Probleme.

Beispiele für TSP mit Schnittebenen (1)

Beispiel 5.5

TSP von Folie 238.

LP: Grad-Ungleichungen für alle Knoten

$z = 189$

Verletzte Ungleichungen:

- Variablenbeschränkungen für: AaDü, BeHa, MüNü
- Subtour für: Ba – Fr – St

Fortsetzung Beispiel.

Nehme entsprechende Ungleichungen hinzu.

$z = 250$

- ☞ Optimale Lösung der LP-Relaxation ist (die bekannte) zulässige TSP-Tour!
- ☞ Branching war nicht erforderlich!

Beispiele für TSP mit Schnittebenen (2)

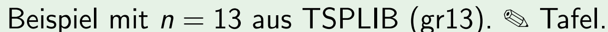
Beispiel 5.6

Beispiel mit $n = 17$ aus TSPLIB (gr17). ✎ Tafel.

LP-Relaxationen siehe Homepage.

Wiederum ist kein Branching notwendig.

Beispiel 5.7



Beispiel mit $n = 13$ aus TSPLIB (gr13). ✎ Tafel.

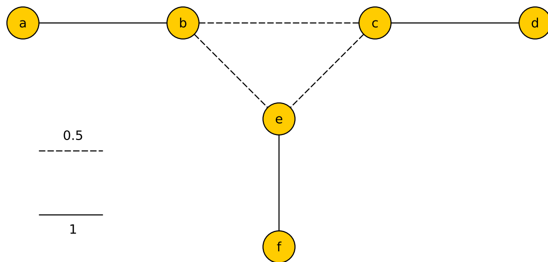
Im Gegensatz zu den beiden vorangegangenen Beispielen treten hier Brüche bei den optimalen Lösungen der Relaxation auf.

Trotzdem genügen die bekannten Ungleichungen, um das das Problem zu lösen.

LP-Relaxationen siehe Homepage.

2-Matching-Ungleichungen (1)

Bei der Lösung einer TSP-Relaxation kann die folgende Situation (als Teil der gesamten Lösung) auftreten.



Hier ist keine der uns bekannten gültigen Ungleichungen verletzt. Was tun?

2-Matching-Ungleichungen (2)

Von den sechs Kanten können höchstens vier an einer TSP-Tour beteiligt sein.

- Es können maximal zwei der Kanten $\{b, c\}, \{b, e\}, \{c, e\}$ enthalten sein, sonst wäre die SEC für $S = \{b, c, e\}$ verletzt.
- Wenn zwei Kanten von $\{b, c\}, \{b, e\}, \{c, e\}$ enthalten sind, z. B. $\{b, c\}$ und $\{c, e\}$, dann ist eine äußere Kante nicht enthalten, hier $\{c, d\}$.
- Damit folgt:

$$x_{a,b} + x_{b,c} + x_{b,e} + x_{c,e} + x_{c,d} + x_{e,f} \leq 4$$

ist eine gültige Ungleichung, die die dargestelle Situation separiert.

2-Matching-Ungleichungen (3)

Zur Definition einer 2-Matching-Ungleichung benötigen wir:

1. eine Knotenmenge (handle) $H \subseteq V$ mit $3 \leq |H|$,
2. eine Kantenmenge (teeth) $T \subseteq \delta(H)$ mit
   - einer ungeraden Anzahl an Kanten,
   - die keinen Knoten gemeinsam haben, also $e \cap e' = \emptyset$ für alle $e, e' \in E$ mit $e \neq e'$.

Die 2-Matching-Ungleichung lautet dann:

$$x(H) + x(T) \leq |H| + \left\lfloor \frac{|T|}{2} \right\rfloor .$$

Herleitung der 2-Matching-Ungleichungen

- Vorzeichenbedingungen für alle $e \in \delta(H) \setminus T$:

$$-x_e \leq 0.$$

- Variablenbeschränkungen für alle $e \in T$:

$$x_e \leq 1.$$

- Gradgleichungen für alle $v \in H$ als $\leq$-Ungleichung:

$$x(\delta(v)) \leq 2.$$

Wir multiplizieren alle Gleichungen mit $\frac{1}{2}$ und summieren auf.

Damit erhalten wir:

$$\begin{array}{rcrcrcr} & & -\frac{1}{2}x(\delta(H) \setminus T) & & & \leq & 0 \\ & & & & \frac{1}{2}x(T) & \leq & \frac{1}{2}|T| \\ x(H) & + & \frac{1}{2}x(\delta(H) \setminus T) & + & \frac{1}{2}x(T) & \leq & |H| \end{array}$$

Die Summation dieser drei Ungleichungen liefert:

$$x(H) + x(T) \leq |H| + \frac{|T|}{2}.$$

Jetzt können wir die rechte Seite abrunden.

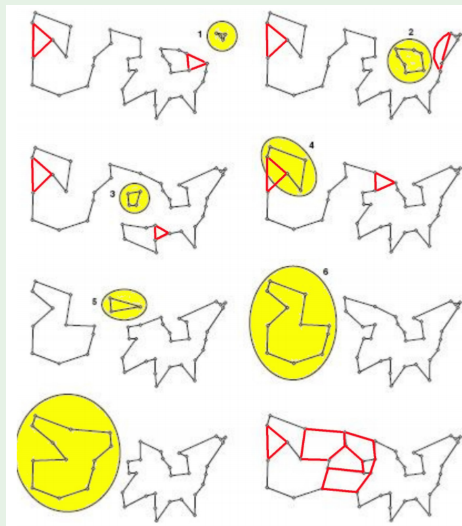
Beispiel 5.8 (Dantzig, Fulkerson, Johnson (1954))

Problem mit 42 US-Städten aus dem Jahr 1954.

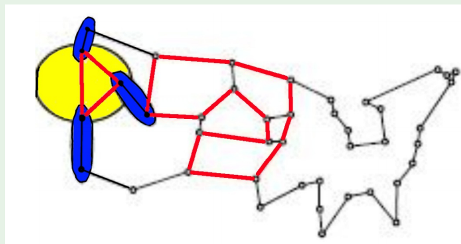
schwarz $= 1$, rot $= \frac{1}{2}$

Sieben SECs liefern die Tour unten rechts.

Hier ist keine SEC mehr verletzt, aber eine 2-Matching-Ungleichung.



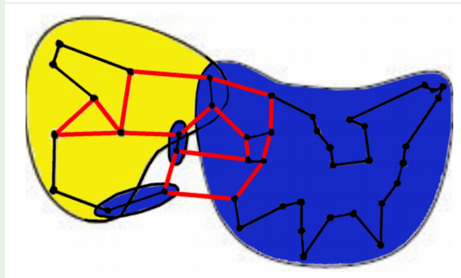
Fortsetzung Beispiel.



Die Grafik oben zeigt die verletzte 2-Matching Ungleichung.

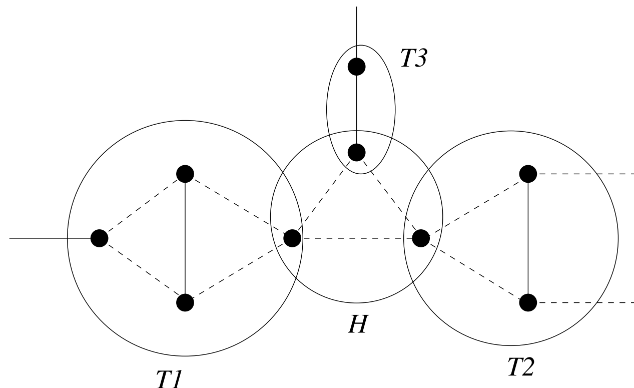
Nach Separation erhalten wir die Lösung unten.

Die hier verletzte Ungleichung ist eine Verallgemeinerung einer 2-Matching-Ungleichung.



Comb-Ungleichungen (1)

2-Matching-Ungleichungen sind ein Spezialfall von Comb-Ungleichungen.



Comb-Ungleichungen (2)

Für eine Comb-Ungleichung benötigen wir:

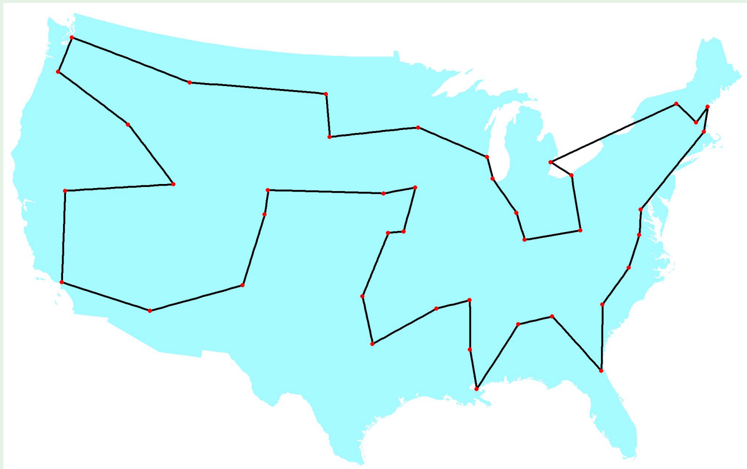
1. eine Knotenmenge (handle) $H \subseteq V$
2. eine ungerade Anzahl an Knotenmengen (teeth) $T_1, \ldots, T_k \subseteq V$ für die gilt:
   - $T_i \cap T_j = \emptyset$ für $1 \leq i < j \leq k$,
   - $H \cap T_i \neq \emptyset$ für $1 \leq i \leq k$,
   - $T_i \setminus H \neq \emptyset \; 1 \leq i \leq k$.

Die Comb-Ungleichung lautet dann:

$$x(H) + \sum_{i=1}^{k} x(T_i) \leq |H| + \sum_{i=1}^{k} |T_i| - \left\lceil \frac{3k}{2} \right\rceil .$$

Beispiel 5.9

Separation der Comb-Ungleichung von Beispiel 5.8 führt zur optimalen Lösung.



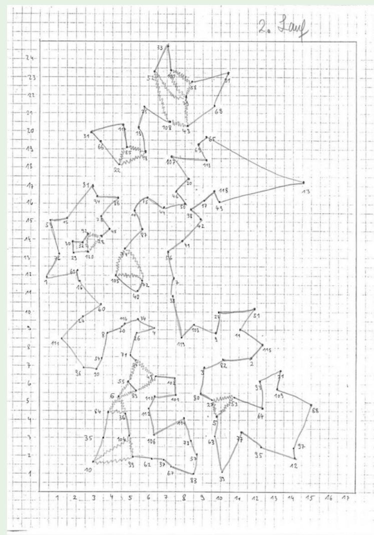
Beispiel 5.10 (Grötschel (1977))

Grötschel 1977, 120 Städte in Deutschland, damals Weltrekord

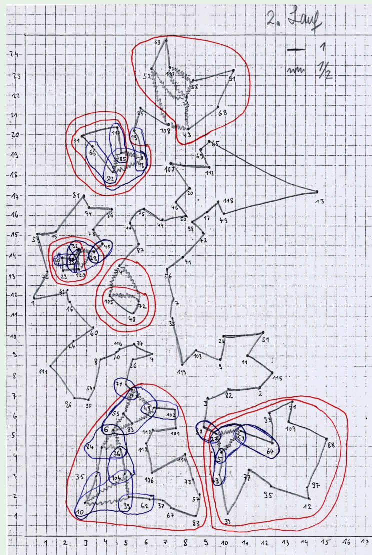
LP-Solver löst die Relaxationen, die dann gezeichnet werden.

Anhand der Zeichnung werden neue verletzte Ungleichungen identifiziert.

Welche SECs und 2-Matching-Ungleichung können Sie erkennen?



Fortsetzung Beispiel.



Separationsverfahren

- Für die effiziente Lösung eines TSP müssen wir eine verletzte gültige Ungleichung algorithmisch effizient erkennen können.
- Hierzu können Heuristiken und exakte Verfahren eingesetzt werden.
- Grundlage für diese Verfahren ist der stützende Graph für die Lösung einer Relaxation.
- Die Separation von verletzten Variablenbeschränkungen ist offensichtlich trivial.
- Im Folgenden betrachten wir die Separation von SECs.

Support Graph

Definition 5.11

Es sei der vollständige Graph $G = (V, E)$ für ein TSP und $\mathbf{x}$ die optimale Lösung einer LP-Relaxation.

Dann heißt der Graph $G(\mathbf{x}) = (V, E')$ mit

$$E' = \{e \in E | x_e > 0\}$$

stützender Graph (support Graph) von $\mathbf{x}$.

Wir ordnen dabei der Kante $e \in E'$ das Gewicht x_e zu.

- Der Stützgraph enthält also genau die Kanten, die zu Entscheidungsvariablen mit einem positiven Wert gehören.
- Wir haben in unseren Beispielen schon implizit mit stützenden Graphen gearbeitet.

Separation von SECs: Zusammenhang

Wir beginnen mit einer sehr einfachen Heuristik.

Lemma 5.12

Es sei $G(\mathbf{x})$ ein stützender Graph.

Dann gilt: Wenn $G(\mathbf{x})$ nicht zusammenhängend ist, dann bildet jede ZHK eine verletzte SEC.

Folgerung:

- Mittels Tiefen- oder Breitensuche können wir effizient verletzte SECs erkennen.
- In Beispiel 5.8 trifft dies auf 6 von 7 SECs zu.

Separation von SECs: Schnitt

Grundlage für alle exakten Separationsverfahren für SECs ist das folgende Lemma.

Lemma 5.13

Es sei $G(\mathbf{x})$ ein stützender Graph.

Die SEC für eine Knotenmenge S ist genau dann verletzt, wenn

$$x(\delta(S)) < 2$$

gilt, d. h. die Schnittkanten zu S haben in $G(\mathbf{x})$ eine Kapazität (Gewicht) < 2.

Folgerung:

- Wir berechnen einen minimalen Schnitt für $G(\mathbf{x})$.
- Wenn dieser Schnitt eine Kapazität < 2 hat, haben wir eine verletzte SEC gefunden.

Schnitte berechnen

- gerichteter Graph mit ausgezeichneter Quelle s und Senke t: Ford-Fulkerson- bzw. Edmonds-Karp-Algorithmus zur Berechnung eines Maximalflusses bzw. eines minimalen Schnitts.
- prinzipiell möglich: Wir ersetzen im Stützgraph $G(\mathbf{x})$ jede ungerichtete Kante durch zwei gerichtete Kanten und
- lösen das Maximalflussproblem für jede mögliche s, t-Menge.
- also: $\frac{n(n-1)}{2}$ Fluss-/Schnittprobleme
- ist polynomiell!
- Verbesserung mit dem Gomory-Hu-Algorithmus auf $n - 1$ Fluss-/Schnittprobleme.

Reduktionsverfahren für SECs (1)

Insbesondere für eine exakte Separation von SECs bietet es sich an, den Stützgraph vor Berechnung der Schnitte zu „schrumpfen“.

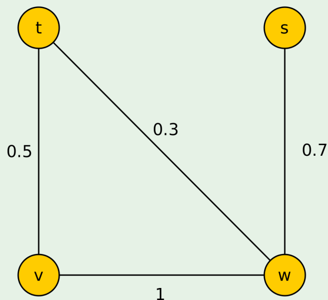
Algorithmus 5.14 (Padberg/Crowder)

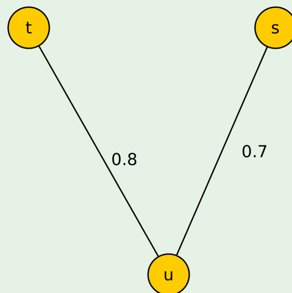
Solange es Kanten $e = \{v, w\}$ *mit Gewicht* $x_e = 1$ *gibt, wiederhole:*

1. *Wähle eine Kante* $e = \{v, w\}$ *mit Gewicht* 1.
2. *Ersetze die Kante* e *durch einen neuen Knoten* u.
3. *Wenn es einen oder mehrere Knoten* t *gibt, die zu beiden Knoten* v *und* w *adjazent sind, dann ersetze die Kanten* $\{v, t\}$ *und* $\{w, t\}$ *durch eine Kante* $\{u, t\}$ *mit Gewicht* $x_{\{u,t\}} = x_{\{v,t\}} + x_{\{w,t\}}$.
4. *Wenn es einen oder mehrer Knoten* t *gibt, die entweder nur zu* v *oder nur zu* w *adjazent sind, so ersetze die jeweilige Kante durch* $\{u, t\}$ *mit dem gleichen Gewicht wie zuvor.*

Reduktionsverfahren für SECs (2)

Beispiel 5.15





Vor-und Nachteile von Branch-and-Cut

Vorteile:

- Für viele Probleme führt der Branch-and-Cut-Ansatz zu vergleichsweise effizienten Lösungsalgorithmen.
- Für die Lösung der LP-Relaxationen stehen leistungsfähige LP-Solver zur Verfügung.

Nachteile:

- hoch spezialisierter Ansatz: Schnittebenen und Separation sind problemspezifisch.
- theoretische Hürde: Es müssen leistungsfähige Schnittebenen bekannt sein, in Verbidung mit effizienten Separationsalgorithmen.
- hoher Implementierungsaufwand: Die Separationsalgorithmen selbst können sehr komplex sein.

Branch-and-Cut mit rein ganzzahligen Lösungen (1)

Problem: Wir wären gerne in der Lage, schnell einen Prototyp zu implementieren, um die Leistungsfähigkeit eines Branch-and-Cut-Ansatzes für ein Optimierungsproblem einschätzen zu können.

Idee:

- Einsatz moderner leistungsfähiger Solver für die ganzzahlige Programmierung.
- Wir nutzen die eingebauten Techniken der Solver, um die LP-Relaxationen optimal ganzzahlig zu lösen.
- Separation nur auf den ganzzahligen Lösungen

Branch-and-Cut mit rein ganzzahligen Lösungen (2)

Das beschriebene Verfahren eignet sich für kombinatorische Optimierungsprobleme mit einer großen Anzahl an Nebenbedingungen in der LP-Formulierung (Beispiel: TSP).

Algorithmus 5.16

Es sei I eine Probleminstanz eines kombinatorischen Optimierungsproblems.

1. *Lege eine LP-Relaxation P für I fest.*
2. *Bestimme eine ganzzahlige Lösung* $\mathbf{x}$ *für P.*
3. *Ist* $\mathbf{x}$ *zulässig für I? Wenn ja, dann STOP!*
4. *Bestimme für* $\mathbf{x}$ *eine verletzte gültige Ungleichung und füge diese P hinzu.*

   Gehe zu 2.

Beispiel TSP

Wenn wir nur ganzzahlige Lösungen der LP-Relaxationen betrachten,

- dann ist durch Gradgleichungen und SECs das Problem vollständig beschrieben.
- Die Variablenbeschränkungen entsprechen SECs auf zwei Knoten.
- Die Separation für SECs ist sehr einfach zu implementieren.

Fazit: Geringer Aufwand zur Implementierung eines TSP-Solvers.

Mit diesem einfachen Ansatz kann man schon erstaunliches erreichen.

Anwendung TSP

Beispiel 5.17

Lösungszeiten (real time) für TSP-Probleme auf i5-8250U CPU (4 Kerne/8 Threads) Notebook mit Gurobi 9 als ILP-Solver:

Problem	#Städte	Zeit
gr21	21	0,016s
fri26	26	0,032s
gr48	48	0,312s
gr96	96	1,636s
rd100	100	0,558s
gr120	120	2,079s

Problem	#Städte	Zeit
bier127	127	0,679s
gr202	202	5,367s
gr229	229	19,521s
rd400	400	1m30,070s
gr431	431	7m37,847s
gr666	666	3m45,436s

Alle Probleme stammen aus der TSPLIB.

Linear Ordering Problem

Gegeben seien n Objekte $o_1, \ldots, o_n$, die in eine Reihenfolge gebracht werden sollen.

Gesucht ist also eine Permutation der n Objekte.

Wenn in der Permutation Objekt o_i (irgendwo) vor Objekt o_j steht, dann fällt Nutzen/Kosten c_{ij} an.

Welche Reihenfolge ist optimal?

Linear Ordering Problem als LP

Variablen

$$x_{ij} = \begin{cases} 1 & o_i \text{ ist vor } o_j \\ 0 & \text{sonst} \end{cases}$$

Maximiere

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

unter den Nebenbedingungen

$$\begin{aligned} x_{ij} + x_{ji} &= 1 \text{ für alle } i < j \\ x_{ij} + x_{jk} + x_{ki} &\leq 2 \text{ für alle } i < j, i < k, j \neq k \\ x_{ij} &\in \{0, 1\}. \end{aligned}$$

Die Ungleichungen heißen Dicycle-Ungleichungen.

Diskussion Linear Ordering Problem

- LOP ist (als Entscheidungsproblem) NP-vollständig.
- $O(n^3)$ viele Dicycle-Ungleichungen, problematisch für größere n
- einfacher Branch-and-Cut-Ansatz:
  - beginne nur mit den Gleichungen $x_{ij} + x_{ji} = 1$,
  - prüfe in einer ganzzahligen Lösung, ab alle Dicycle-Ungleichungen erfüllt sind und
  - füge verletzte Dicycle-Ungleichungen der Relaxation hinzu.

LP-Solver-Frameworks

- Die modernen LP-Solver bieten ein Framework für die Entwicklung von Algorithmen an.
- Man kann nicht nur einen LP-Solver nutzen, sondern über das Framework selbst in die Lösung eingreifen.
- Hierzu registriert man beim Solver eine Callback-Funktion, die bei bestimmten Ereignissen aufgerufen wird.
- Die Callback-Funktion kann dazu genutzt werden
  - Schnittebenen zu erzeugen und
  - heuristische Lösungen zu generieren.

Beispiel Gurobi

Reference Manual

- Monitoring Progress - Logging and Callbacks
  - `GRBsetcallbackfunc`
  - `GRBcbget`
- Modifying Solver Behavior - Callbacks
  - `GRBcbcut`
  - `GRBcblazy`
  - `GRBcbsolution`
- Callback Codes
  - `MIPNODE`
  - `MIPSOL`

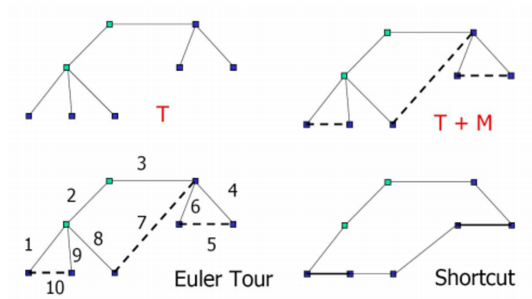
Examples: Gurobi TSP Beispiel

Zusammenfassung

- Branch-and-Cut: Branch-and-Bound in Verbindung mit problemspezifischen Schnittebenen und effizienter Separation
- sehr erfolgreich bei der Suche nach optimalen Lösungen
- Schnittebenen für TSP: SECs, 2-Matching- und Comb-Ungleichungen
- hoher Implementierungsaufwand
- praktischer Ansatz:
  - Separation rein ganzzahliger Lösungen
  - Nutzung leistungsfähiger LP-Solver als Branch-and-Cut-Framework

Kapitel 6

Heuristiken und Approximation



Inhalt

6 Heuristiken und Approximation

- Approximationsalgorithmen
- Approximation mithilfe von Greedy-Algorithmen
- Dynamische Programmierung und Approximationsschemata
- LP und randomisiertes Runden

Lösungsansatz und Fragen

Ansatz:

- Wir verzichten auf die Forderung nach Optimalität.
- Es genügt uns, wenn eine Lösung fast optimal ist.
- Wir versuchen, auch Aussagen über die Güte einer Lösung herzuleiten.

Fragen:

- Wie können wir zu einem Problem einen Algorithmus finden, der für jede beliebige Instanz eine nahezu optimale Lösung liefert?
- Wie können wir "nahezu optimal" quantifizieren? Können wir eine Schranke für die Güte einer Lösung beweisen?
- Ist solch eine ermittelte Schranke scharf?
- Gibt es Grenzen für diese Güteschranken?

Methoden für den Entwurf von Approximationsalgorithmen

- Greedy-Algorithmen
- Dynamische Programmierung
- LP-Relaxationen
- Lokale Suche und Metaheuristiken

Optimierungsproblem aus Sicht der Approximation

Definition 6.1

Ein Optimierungsproblem Π besteht aus

- einer Menge $\mathcal{I}_\Pi$ von Instanzen oder Eingaben.
- Zu jeder Instanz $I \in \mathcal{I}_\Pi$ gehört eine Menge $\mathcal{S}_I$ von (zulässigen) Lösungen und
- eine Zielfunktion $f_I : \mathcal{S}_I \to \mathbb{R}$, die jeder (zulässigen) Lösung einen reellen Wert zuordnet.
- Zusätzlich ist vorgegeben, ob wir eine Lösung mit minimalem oder maximalen Wert $f_I(x)$ suchen.

Für eine Instanz $I \in \mathcal{I}_\Pi$ bezeichnet $\text{OPT}(I)$ den Wert einer optimalen Lösung.

Geht die Instanz I aus dem Kontext hervor, schreiben wir auch kurz OPT statt $\text{OPT}(I)$.

Beispiel 6.2

- Eine Instanz I von MST (minimum spanning tree problem, Minimalgerüst) wird durch einen ungerichteten Graphen $G = (V, E)$ und Kantengewichte $c : E \to \mathbb{N}$ beschrieben.
- Die Menge $\mathcal{S}_I$ der Lösungen ist die Menge aller Gerüste (aufspannender, zusammenhängender, kreisfreier Untergraph) für den Graphen G.
- Sei $T = (V, F) \in \mathcal{S}_I$ ein Gerüst von G. Dann lautet die Zielfunktion:

$$f_I(T) = \sum_{e \in F} c(e).$$

Sie weist jedem Gerüst $T \in \mathcal{S}_I$ ein Gewicht zu.

- Wir möchten minimieren. Also:

$$\mathrm{OPT}(I) = \min_{T \in \mathcal{S}_I} f_I(T).$$

Approximationsalgorithmus

Definition 6.3

Ein Approximationsalgorithmus A für ein Optimierungsproblem Π ist ein Polynomialzeitalgorithmus, der zu jeder Instanz I eine Lösung aus $\mathcal{S}_I$ ausgibt.

Wir bezeichnen mit $A(I)$ die ausgegebene Lösung für die Instanz I und mit $w_A(I) = f_I(A(I))$ ihren Wert.

Bemerkung:

- Je näher $w_A(I)$ dem optimalen Wert $\text{OPT}(I)$ ist, desto besser ist der Algorithmus.

Approximationsgüte

Definition 6.4

Ein Approximationsalgorithmus A für ein Minimierungs- bzw. Maximierungsproblem Π hat eine Approximationsgüte von $r \geq 1$ bzw. $r \leq 1$, wenn

$$w_A(I) \leq r \cdot \mathrm{OPT}(I) \quad \text{bzw.} \quad w_A(I) \geq r \cdot \mathrm{OPT}(I)$$

für alle Instanzen $I \in \mathcal{S}_I$ gilt. Wir sagen dann, dass A ein r-Approximationsalgorithmus ist.

Fortsetzung Definition.

Hängt der Approximationsfaktor von der Länge der Eingabe ab, dann ist

$$r : \mathbb{N} \to [1, \infty) \quad \text{bzw.} \quad r : \mathbb{N} \to [0, 1]$$

eine Funktion und es muss

$$w_A(I) \leq r(|I|) \cdot \mathrm{OPT}(I) \quad \text{bzw.} \quad w_A(I) \geq r(|I|) \cdot \mathrm{OPT}(I)$$

für alle Instanzen $I \in \mathcal{S}_I$ gelten. Hierbei bezeichnet $|I|$ die Länge der Eingabe.

Diskussion Approximationsgüte

- Haben wir für ein Minimierungsproblem einen 2-Approximationsalgorithmus, so bedeutet dies, dass der Algorithmus für jede Instanz eine Lösung berechnet, deren Wert höchstens doppelt so groß ist wie der optimale Wert.
- Haben wir einen $\frac{1}{2}$-Approximationsalgorithmus für ein Maximierungsproblem, so berechnet der Algorithmus für jede Instanz eine Lösung, deren Wert mindestens halb so groß ist wie der optimale Wert.

- Ein Polynomialzeitalgorithmus für ein Optimierungsproblem, der stets eine optimale Lösung berechnet, ist ein 1-Approximationsalgorithmus.
- Ist ein Optimierungsproblem NP-schwer, so kann, unter der Voraussetzung $P \neq NP$, kein 1-Approximationsalgorithmus existieren.
- Es wäre aber durchaus möglich, dass ein 1.01-Approximationsalgorithmus existiert, also ein Algorithmus mit polynomieller Laufzeit, der stets eine Lösung berechnet, deren Wert höchstens um 1% größer als der optimale Wert ist.

Beispiel Approximationsgüte: TSP

Beispiel 6.5

Aus der Graphentheorie kennen sie den folgenden 2-Approximationsalgorithmus für das metrische TSP (Traveling Salesman Problem).

Eine Instanz wird beschreiben durch eine Knotenmenge V und eine Metrik d auf V.

1. Sei $G = (V, E)$ der vollständiger Graph mit Knotenmenge V. Berechne ein Minimalgerüst T von G bezüglich der Metrik d.
2. Führe auf T eine Tiefensuche aus und vergebe dabei die Tiefensuchnummern in der Reihenfolge der Knotenbesuche.
3. Gebe die Knoten in der Reihenfolge der Tiefensuchnummern aus, zusätzlich den Startknoten am Ende nochmals. Dies beschreibt einen Hamiltonkreis C als Lösung.

Beispiel Approximationsgüte: Rucksackproblem

Beispiel 6.6

- Der Algorithmus GreedyKP kann beliebig schlechte Lösungen liefern.
- Durch eine einfache Anpassung erreichen wir einen $\frac{1}{2}$-Approximationsalgorithmus.
- siehe Folie 232 ff.

Greedy

- Paradigma für Optimierungsprobleme
- greedy = gierig, gefräßig
- Man versucht ein Optimierungsproblem durch lokale Auswahl eines jeweils besten Elements unter Beachtung der Nebenbedingungen zu lösen.
- Allgemein liefert dies keine optimale Lösung sondern nur irgendeine Lösung.

Teilmengensystem

Wir betrachten Greedy-Algorithmen formal auf der Basis von Teilmengensystemen.

Definition 6.7

Es sei E eine endliche Menge und $\mathcal{T}$ eine nichtleere Menge von Teilmengen von E.

Das Mengensystem $(E, \mathcal{T})$ heißt Teilmengensystem, wenn für alle $A, B \subseteq E$ gilt:

$$A \in \mathcal{T} \text{ und } B \subseteq A \Longrightarrow B \in \mathcal{T}$$

d. h., jede der Teilmengen in $\mathcal{T}$ ist bezüglich $\subseteq$ abgeschlossen.

Beispiele für Teilmengensysteme

Beispiel 6.8

1. Es sei $E = \{e_1, \ldots, e_n\}$ eine beliebige endliche Menge und $k \leq n$. Dann ist $(E, \mathcal{T})$ mit
$$\mathcal{T} = \{A \subseteq E \,|\, |A| \leq k\}$$
ein Teilmengensystem.
2. Es sei $G = (V, E)$ ein zusammenhängender Graph. Dann ist $(E, \mathcal{T})$ mit
$$\mathcal{T} = \{F \subseteq E \,|\, (V, F) \text{ ist ein kreisfreier Untergraph von } G\}$$
ein Teilmengensystem.

Fortsetzung Beispiel.

3. Es sei $G = (V, E)$ ein vollständiger Graph. Dann ist $(E, \mathcal{T})$ mit

$$\mathcal{T} = \{F \subseteq E \mid F \text{ ist Kantenteilmenge eines Hamiltonkreis}\}$$

ein Teilmengensystem.

4. Es sei $E = \{e_1, \ldots, e_n\}$ eine beliebige endliche Menge und $c : E \to \mathbb{R}_+$ eine Gewichtung der Elemente von E. Weiterhin sei $K \in \mathbb{R}_+$. Dann ist $(E, \mathcal{T})$ mit

$$\mathcal{T} = \left\{ F \subseteq E \,\middle|\, \sum_{e \in F} c(e) \leq K \right\}$$

ein Teilmengensystem.

Teilmengensystem und Optimierungsproblem

Definition 6.9

Das zu einem Teilmengensystem $(E, \mathcal{T})$ gehörige Optimierungsproblem besteht darin, für eine beliebige Gewichtsfunktion $w : E \to \mathbb{R}$ eine in $\mathcal{T}$ maximale Menge T zu finden, deren Gesamtgewicht

$$w(T) := \sum_{e \in T} w(e)$$

maximal (bzw. minimal) ist.

Bemerkung: Eine Menge $T \in \mathcal{T}$ ist maximal, wenn es in $\mathcal{T}$ keine echte Obermenge von T gibt.

Beispiele: Teilmengensystem und Optimierungsproblem

Beispiel 6.10

1. MST entspricht dem Optimierungsproblem zum Teilmengensystem (2) aus Beispiel 6.8. Hierbei entspricht $w(e)$ dem Gewicht einer Kante.
2. TSP entspricht dem Optimierungsproblem zum Teilmengensystem (3) aus Beispiel 6.8. Hierbei entspricht $w(e)$ der Länge einer Kante.

Greedy-Algorithmus

Algorithmus 6.11 (Maximierung)

Gegeben: Teilmengensystem $(E, \mathcal{T})$ *mit* $E = \{e_1, \ldots, e_n\}$

Ordne alle Elemente $e \in E$ *absteigend nach Gewicht:*
$w(e_1) \geq w(e_2) \geq \cdots \geq w(e_n)$
$T = \emptyset$
for $i := 1$ **to** n **do**
    if $T \cup \{e_i\} \in \mathcal{T}$ **then**
        $T := T \cup \{e_i\}$
    end
end
gebe T *aus*

Bemerkung: Für eine Minimierung werden die Elemente von E aufsteigend sortiert, also $w(e_1) \leq w(e_2) \leq \cdots \leq w(e_n)$.

Anwendung von Greedy: Algorithmus von Kruskal

Optimierungsproblem: MST, siehe Beispiel 6.2.

Algorithmus 6.12 (Kruskal)

$F := \emptyset$; $ZHK := \emptyset$; $i := 0$;
$L :=$ *Liste der Kanten aufsteigend sortiert nach ihrer Länge;*
for all $v \in V$ **do** $ZHK := ZHK \cup \{\{v\}\}$; **end**
while $|ZHK| > 1$ **do**
    $i := i + 1$;
    Es sei $\{v, w\}$ *das* i*-te Element von* L*;*
    if v *und* w *liegen in verschiedenen ZHKs* K_1 *und* K_2 **then**
        $ZHK := (ZHK \setminus \{K_1, K_2\}) \cup \{\{K_1 \cup K_2\}\}$;
        $F := F \cup \{\{v, w\}\}$;
    end
end
Ausgabe von F*;*

Diskussion Greedy-Algorithmus

Der Greedy-Algorithmus liefert nicht immer eine Menge T mit maximalem Gewicht.

Beispiel 6.13

1. Sei $E = \{a, b, c\}$ und $\mathcal{T} = \{\emptyset, \{a\}, \{b\}, \{c\}, \{a, b\}\}$ mit $w(a) = w(b) = 2, w(c) = 3$.

   Dann berechnet der Greedy-Algorithmus $T = \{c\}$ mit $w(T) = 3$, obwohl $T' = \{a, b\}$ mit $w(T') = 4$ optimal ist.
2. Für TSP (Teilmengensystem (3) aus Beispiel 6.8) liefert der Greedy-Algorithmus i. A. keine optimale Lösung (siehe Beispiel 6.19).

Frage: Für welche Teilmengensysteme liefert der Greedy-Algorithmus eine optimale Lösung?

Matroid

Definition 6.14

Ein Teilmengensystem $(E, \mathcal{T})$ heißt Matroid, wenn für alle $A, B \in \mathcal{T}$ das folgende Austauschaxiom gilt:

$$|A| = |B| + 1 \quad \Longrightarrow \quad \exists a \in A \setminus B : B \cup \{a\} \in \mathcal{T}.$$

Die Mengen in $\mathcal{T}$ werden unabhängige Mengen genannt. Jede maximale unabhängige Menge heißt Basis.

Kardinalität der Basen

Lemma 6.15

Also Basen eines Matroids haben die gleiche Kardinalität.

Beweis.

Ann.: Es existieren zwei Basen A, B mit $|A| \neq |B|$.

- O.B.d.A. gelte $|A| > |B|$.
- Wegen der Teilmengensystemeigenschaft folgt, dass eine Menge $A' \subseteq A$ existiert mit $|A'| = |B| + 1$.
- Mit dem Austauschaxiom folgt, dass $a \in A'$ existiert mit $B \cup \{a\} \in \mathcal{T}$.
- Wegen $B \subseteq B \cup \{a\}$ ist dies ein Widerspruch zur Basiseigenschaft von B.

Optimalität des Greedy-Algorithmus für Matroide

Satz 6.16

Der Greedy-Algorithmus löst das zu einem Teilmengensystem $(E, \mathcal{T})$ gehörige Optimierungsproblem für jede Gewichtsfunktion $w : E \to \mathbb{R}$ genau dann, wenn $(E, \mathcal{T})$ ein Matroid ist.

Beweis

"$\Leftarrow$":

- Nach Lemma 6.15 haben alle Basen von $(E, \mathcal{T})$ die gleiche Kardinalität.
- Es sei k diese Kardinalität und es sei $T = \{e_1, \ldots, e_k\}$ die vom Greedy-Algorithmus berechnete Menge $T \in \mathcal{T}$.
  Es gelte $w(e_1) \geq \cdots \geq w(e_k)$.
- Es sei $U = \{g_1, \ldots, g_k\} \in \mathcal{T}$ eine andere Basis aus $\mathcal{T}$.
  Es gelte $w(g_1) \geq \cdots \geq w(g_k)$.

Fortsetzung Beweis.

- Annahme: Es existiert ein i mit $w(g_i) > w(e_i)$.
- Es sei i der kleinste solche Index, also $w(g_1) \leq w(e_1), \ldots, w(g_{i-1}) \leq w(e_{i-1})$.
- Sei $A = \{g_1, \ldots, g_i\}$ und $B = \{e_1, \ldots, e_{i-1}\}$. Mit dem Austauschaxiom folgt, dass $g_j \in A \setminus B$ existiert, so dass $B \cup \{g_j\} \in \mathcal{T}$ gilt.
- Wegen $w(g_j) \geq w(g_i) > w(e_i)$ hätte der Greedy-Algorithmus vor e_i schon g_j genommen. Widerspruch!
- Also gilt $w(g_i) \leq w(e_i)$ für $1 \leq i \leq k$ und somit $w(U) \leq w(T)$.

Fortsetzung Beweis.

"$\Rightarrow$": Das Austauschaxiom gelte nicht.

- Dann existieren $A, B \in \mathcal{T}$ mit $|A| = |B| + 1$ und $B \cup \{a\} \notin \mathcal{T}$ für alle $A \setminus B$.
- Es sei $r := |A|$ und $w : E \to \mathbb{R}$ sei gegeben durch:

$$w(e) := \begin{cases} r+1 & \text{für } x \in B, \\ r & \text{für } x \in A \setminus B, \\ 0 & \text{sonst.} \end{cases}$$

- Der Greedy-Algorithmus wählt dann eine Menge T mit $T \supseteq B$ und $T \cap (A \setminus B) = \emptyset$ und Gewicht $w(T) = |B| \cdot (r+1) = r^2 - 1$.
- Wählt man stattdessen ein $U \in \mathcal{T}$ mit $U \supseteq A$, dann ergibt sich $w(U) \geq |A| \cdot r = r^2 > w(T)$.

Graphisches Matroid

Folgerung 6.17

Das Teilmengensystem von Beispiel 6.8 (2) ist ein Matroid.

Beweis.

- Der Kruskal-Algorithmus entspricht dem Greedy-Algorithmus für dieses Teilmengensystem und er liefert für jede Kantengewichtung eine optimale Lösung (siehe Graphentheorie).
- Mit Satz 6.16 folgt, dass das Teilmengensystem ein Matroid ist.

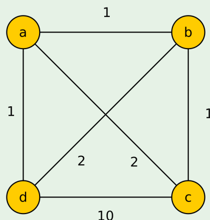
Definition 6.18

Das Teilmengensystem von Beispiel 6.8 (2) heißt graphisches Matroid.

Weitere Teilmengensysteme

Beispiel 6.19

- Das Teilmengensystem von Beispiel 6.8 (1) ist ein Matroid. Dies lässt sich einfach direkt nachweisen.
- Das Teilmengensystem von Beispiel 6.8 (3) ist kein Matroid.
  Begründung: Der Greedy-Algorithmus liefert für den folgenden Graphen den Hamiltonkreis (a, b, c, d, a) mit Gesamtlänge 13.



Optimal wäre (a, c, b, d, a) mit Gesamtlänge 6.

Anwendung Scheduling

Beispiel 6.20

Gegeben Sei eine Menge $J = \{1, 2, \ldots, n\}$ von Jobs.
Jeder Job hat

- eine Deadline $d(i)$ und
- eine Strafe $p(i)$, die bei nicht rechtzeitiger Fertigstellung gezahlt werden muss.

Pro Tag kann nur ein Job abgearbeitet werden.
Wie müssen wir die Jobs einplanen, so dass die Gesamtstrafe minimiert wird?

Job i	1	2	3	4	5	6
$d(i)$	1	1	2	3	3	6
$p(i)$	10	9	6	7	4	2

Fortsetzung Beispiel.

$(J, \mathcal{T})$ mit

$$\begin{aligned}\mathcal{T} &:= \{P \subseteq J | \forall j \in P : j \text{ ist pünktlich planbar}\} \\ &= \{P \subseteq J | \forall n \in \mathbb{N} : |\{j \in P | d(j) \leq n\}| \leq n\}\end{aligned}$$

ist ein Matroid. Also nehmen wir $p(i)$ als Gewichtsfunktion und wenden den Greedy-Algorithmus an.
Terminplan:

Job i	1	2	3	4	5	6
$d(i)$	1	1	2	3	3	6
$p(i)$	10	9	6	7	4	2
$t(i)$	1	5	2	3	6	4

Strafe: 13. Die Jobs 2 und 5 werden nicht rechtzeitig fertig.

Matching

Definition 6.21

Es sei $G = (V, E)$ ein Graph.

Eine Kantenteilmenge $M \subseteq E$ heißt Matching, wenn es keinen Knoten $v \in V$ gibt, der zu mehr als einer Kante aus M inzident ist.

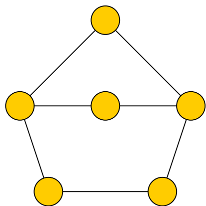
Ein Matching $M \subseteq E$ heißt (inklusions-)maximal, wenn für alle $e \in E \setminus M$ gilt, dass $M \cup \{e\}$ kein Matching ist.

Greedy-Algorithmus für Vertex Cover

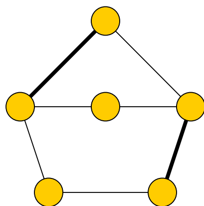
Algorithmus 6.22 (Matching-VC)

1. *Berechne wie folgt ein maximales Matching $M \subseteq E$:*
   1. *Setze $M = \emptyset$.*
   2. *Gehe die Kantenmenge E einmal durch. Füge dabei $e = \{v, w\} \in E$ zu M hinzu, wenn weder v noch w inzident zu einer Kante aus M ist.*
2. *Sei $V(M)$ die Menge aller Knoten, die zu einer Kante in M inzident sind. Gib $V(M)$ aus.*

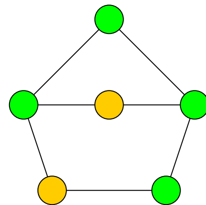
Veranschaulichung Matching-VC



$G = (V, E)$



maximales Matching



zugehörige Knotenüberdeckung

Eigenschaften von Matching-VC

Satz 6.23

Für einen Graphen $G = (V, E)$ hat der Algorithmus Matching-VC die folgenden Eigenschaften:

1. *Seine Laufzeit beträgt $O(|V| + |E|)$.*
2. *Er berechnet stets eine Knotenüberdeckung G.*
3. *Die Approximationsgüte beträgt 2.*

Matching-VC ist also ein 2-Approximationsalgorithmus mit linearer Laufzeit.

Beweis der Eigenschaften

Beweis.

1. - Wir numerieren die Knoten mit Zahlen $0, \ldots, |V| - 1$.
   - Wir speichern in einem Feld für die Knoten (Knotennummer als Index), welche Knoten zu einer Kante aus M inzident sind.
   - Wir iterieren über die Kanten und testen dabei in jeder Iteration, ob wir die aktuelle Kante $e = \{v, w\}$ zu M hinzufügen können. Laufzeit für Test und Hinzufügen ist $O(1)$, insgesamt also $O(|E|)$.
   - Wir laufen über das Knotenfeld und geben die Knoten, die als inzident markiert sind, aus. Laufzeit: $O(|V|)$.
2. Annahme: $V(M)$ ist keine Knotenüberdeckung.
   - Dann gibt es eine Kante $e = \{v, w\}$ mit $v, w \notin V(M)$.
   - Damit wäre $M \cup \{e\}$ ein Matching.
   - Somit hätte Matching-VC e zu M hinzufügen müssen. Widerspruch!

Fortsetzung Beweis.

3.
   - Sei V^* eine minimale Knotenüberdeckung.
   - V^* deckt jede Kante aus M ab. Für jede Kante aus M muss also mindestens einer der beiden Knoten in V^* sein.
   - Kein Knoten $v \in V^*$ kann mehr als eine Kante von M abdecken (sonst kein Matching).
   - Damit folgt: $\mathrm{OPT}(G) = |V^*| \geq |M|$.
   - Für unsere konstruierte Lösung $V(M)$ gilt $|V(M)| = 2|M|$.

Insgesamt folgt damit:

$$\frac{|V(M)|}{\mathrm{OPT}(G)} \leq \frac{2|M|}{|M|} = 2.$$

Beweis der Approximationsgüte

Zum Beweis einer Approximationsgüte bei einem Minimierungsproblem müssen wir zwei Dinge tun:

1. Herleitung einer oberen Schranke für die Lösung des Approximationsalgorithmus.
   - Bei Matching-VC war dies $2|M|$.
2. Herleitung einer unteren Schranke für den Wert der optimalen Lösung.
   - Bei Matching-VC war dies $|M|$.

Der Quotient dieser Schranken ist ein Abschätzung für die Approximationsgüte.

Zum Beweis einer Approximationsgüte bei einem Maximierungsproblem:

1. Herleitung einer unteren Schranke für die Lösung des Approximationsalgorithmus.
2. Herleitung einer oberen Schranke für den Wert der optimalen Lösung.

Die Schwierigkeit besteht oft darin, eine Schranke für den Wert einer optimalen Lösung zu finden.

Set Cover

Definition 6.24

Das Problem Set Cover (SC) wird wie folgt definiert:

Eingabe: Grundmenge S mit n Elementen,
m Teilmengen $S_1, \ldots, S_m \subseteq S$ mit $\cup_{i=1}^{m} S_i = S$,
Kosten $c_1, \ldots, c_m \in \mathbb{N}$.

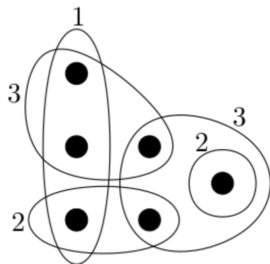
Lösungen: alle Teilmengen $A \subseteq \{1, \ldots, m\}$ mit:

$$\cup_{i \in A} S_i = S$$

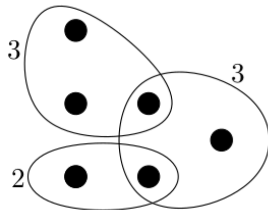
Eine Menge A mit dieser Eigenschaft nennen wir Set Cover von S.

Zielfunktion: minimiere $c(A) = \sum_{i \in A} c_i$

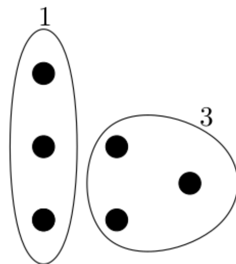
Veranschaulichung Set Cover



(a) Instanz



(b) Set Cover mit Wert 8



(c) Set Cover mit Wert 4

Fakt 6.25

Das Problem Set Cover ist NP-schwer.

Greedy-Algorithmus für Set Cover

Algorithmus 6.26 (Greedy-SC)

$A := \emptyset$; $C := \emptyset$;

while $C \neq S$ **do**

  Wähle i bzw. S_i mit minimalen relativen Kosten $r_i(C) = \frac{c_i}{|S_i \setminus C|}$.

  Setze $\text{price}(x) := \frac{c_i}{|S_i \setminus C|}$ *für alle* $x \in S_i \setminus C$.

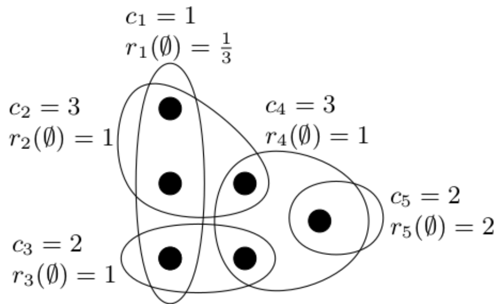
  $A := A \cup \{i\}$; $C := C \cup S_i$;

end

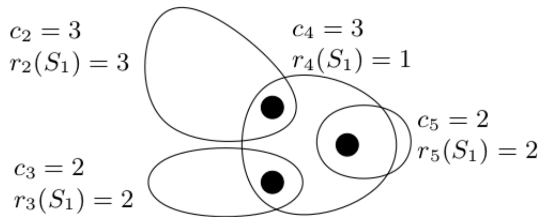
return A

Bemerkung: Die mittlere Anweisung in der Schleife dient nur der nachfolgenden Analyse.

Veranschaulichung Greedy-SC



(a) S_1 hat minimale relative Kosten



(b) S_4 hat minimale relative Kosten

Eigenschaften von Greedy-SC

Definition 6.27

Mit $H_n = \sum_{i=1}^{n} \frac{1}{i}$ bezeichnen wir die n-te harmonische Zahl.

Satz 6.28

Der Algorithmus Greedy-SC ist ein H_n-Approximationsalgorithmus für das Problem Set Cover.

Beweis.

- vergleiche $c(A)$ mit Kosten einer optimalen Lösung
- Wir verteilen $c(A)$ auf die Elemente der Grundmenge S.
- Hierzu dient price : $S \to \mathbb{R}$. Es gilt

$$c(A) = \sum_{x \in S} \text{price}(x).$$

- Was können wir über die Kosten der optimalen Lösung aussagen?
- Wir ordnen die Elemente von S in der Reihenfolge, in der sie zu C hinzugefügt werden.
- Sei $x_1, \ldots, x_n$ die Reihenfolge, die sich ergibt.

Lemma 6.29

Für jedes $k \in \{1, \ldots, n\}$ *gilt* $\text{price}(x_k) \leq \text{OPT}/(n - k + 1)$.

Beweis von Lemma 6.29.

- Wir betrachten ein beliebiges $k \in \{1, \ldots, n\}$. Sei $i \in \{1, \ldots, m\}$ der Index von S_i, durch deren Hinzunahme x_k abgedeckt wird. Sei A die Auswahl unmittelbar bevor i zu A hinzugefügt wird.
- Nicht abgedeckte Elemente $\overline{C} = S \setminus C$ mit $C = \bigcup_{j \in A} S_j$.
- $|\overline{C}| \geq n - k + 1$, da höchstens die Elemente $x_1, \ldots, x_{k-1}$ abgedeckt sind.
- Wir betrachten nun die Set-Cover-Instanz I' eingeschränkt auf die Elemente von $\overline{C}$, also bereits abgedeckten Elemente werden aus S und jeder Menge S_j entfernt.
- Sei dann A^* die optimale Lösung für I' mit $\text{OPT}' = c(A^*)$. Damit gilt:

$$\text{OPT}' = \sum_{j \in A^*} c_j = \sum_{j \in A^*} \left(|S_j \setminus C| \cdot \frac{c_j}{|S_j \setminus C|} \right) = \sum_{j \in A^*} \left(\sum_{x_j \in S_j \setminus C} r_j(C) \right).$$

Beweis von Lemma 6.29.

Aus der Wahl von S_i als nächste Menge folgt, dass es in der aktuellen Situation keine Menge gibt, deren relative Kosten kleiner als $r_i(C)$ sind. Damit folgt:

$$\begin{aligned}
\mathrm{OPT} \geq \mathrm{OPT}' &= \sum_{j \in A^*} \left(\sum_{x_j \in S_j \setminus C} r_j(C) \right) \\
&\geq \sum_{j \in A^*} \left(\sum_{x_j \in S_j \setminus C} r_i(C) \right) \geq |S \setminus C| \cdot r_i(C) \\
&\geq (n - k + 1) \cdot r_i(C).
\end{aligned}$$

Wegen $r_i(C) = \mathrm{price}(x_k)$ ist damit das Lemma beweisen.

Fortsetzung Beweis von Satz 6.28.

Aus Lemma 6.29 folgt nun der Beweis von Satz 6.28:

$$
\begin{aligned}
c(A) = \sum_{x \in S} \text{price}(S) &= \sum_{k=1}^{n} \text{price}(x_k) \\
&\leq \sum_{k=1}^{n} \frac{\text{OPT}}{n-k+1} = \text{OPT} \cdot \sum_{k=1}^{n} \frac{1}{k} \\
&= \text{OPT} \cdot H_n.
\end{aligned}
$$

Schranken für harmonische Zahlen

Lemma 6.30

Für alle $n \in \mathbb{N}$ gilt:

$$\log(n+1) \leq H_n \leq 1 + \log(n).$$

Der Beweis ist Übungsaufgabe.

Folgerung 6.31

Der Algorithmus Greedy-SC hat einen Approximationsfaktor von $O(\log(n))$.

Rucksackproblem und dynamische Programmierung

Algorithmus 6.32

// *Backward Computation*

$V_{n+1,c} := 0$ *für* $c = 0, \ldots, C$
for $i := n$ **downto** 1 **do**
  for $s := 0$ **to** C **do**
    $V_{i,s} := V_{i+1,s}$
    $a_{i,s} := 0$
    if $s \geq w_i \wedge V_{i,s} < p_i + V_{i+1,s-w_i}$ **then**
      $V_{i,s} := p_i + V_{i+1,s-w_i}$
      $a_{i,s} := 1$
    end
  end
end

Fortsetzung Algorithmus.

```
// Forward Computation
z := V_{1,C}
s := C
for i := 1 to n do
    x_i := a_{i,s}
    if x_i = 1 then
        s := s − w_i
    end
end
return x, z
```

Satz 6.33

Algorithmus 6.32 berechnet in Zeit $O(Cn)$ eine optimale Lösung für eine Instanz des Rucksackproblems.

Diskussion Algorithmus

- Dynamische Programmierung: Zustände, Aktionen, Zustandsübergänge
- bei diskreten Zustands- und Aktionenmengen immer als Wegeproblem in einem Graphen modellierbar
- Laufzeit gut einschätzbar, im Gegensatz zu Branch-and-Bound
- Laufzeit $O(Cn)$ ist nicht linear sondern exponentiell!
- Begründung: C ist ein einzelner Wert, die Laufzeit wächst exponentiell in der Kodierungslänge von C.
- praktischer Aspekt: für beschränktes C ein linearer Algorithmus

Pseudopolynomialität

Definition 6.34

Es sei Π ein Optimierungsproblem, in dessen Instanzen ganze Zahlen enthalten sind.

Ein Algorithmus A zur Lösung von Π heißt pseudopolynomieller Algorithmus, wenn seine Laufzeit durch ein Polynom in der Eingabelänge und dem größten Absolutwert der Zahlen der Eingabe nach oben beschränkt ist.

Bemerkungen:

- Algorithmus 6.32 ist ein pseudopolynomieller Algorithmus für das Rucksackproblem.
- Wir können damit Instanzen, für die C polynomiell in der Anzahl der Objekte beschränkt ist, effizient lösen.
- Gilt bspw. $C = O(n^2)$, haben wir einen $O(n^3)$ Algorithmus.

Approximationsschema

Definition 6.35

Ein Approximationsschema A für ein Optimierungsproblem Π ist ein Algorithmus,

- der zu jeder Eingabe der Form (I, ϵ) mit $I \in \mathcal{I}_\Pi$ und $\epsilon > 0$ eine Lösung $A(I, \epsilon) \in \mathcal{S}_I$ berechnet.
- Dabei muss für den Wert $w_A(I, \epsilon) = f_I(A(I, \epsilon))$ dieser Lösung für jede Eingabe (I, ϵ) bei einem Minimierungs- oder Maximierungsproblem Π die Ungleichung

$$w_A(I, \epsilon) \leq (1 + \epsilon) \cdot \text{OPT}(I) \quad \text{bzw.} \quad w_A(I, \epsilon) \geq (1 - \epsilon) \cdot \text{OPT}(I)$$

gelten.

- Wie konstruiert man ein Approximationsschema?
- Wie hängt dabei die Laufzeit von ϵ ab?

PTAS und FPTAS

Definition 6.36

Ein Approximationsschema A heißt polynomielles Approximationsschema (PTAS), wenn die Laufzeit von A für jede feste Wahl von $\epsilon > 0$ durch ein Polynom in $|I|$ nach oben beschränkt ist.

Wir nennen ein Approximationsschema A voll-polynomielles Approximationsschema (FPTAS), wenn die Laufzeit von A durch ein bivariates Polynom in $|I|$ und $1/\epsilon$ nach oben beschränkt ist.

Diskussion PTAS und FPTAS

- Jedes FPTAS ist auch ein PTAS. Umgekehrt gilt dies jedoch nicht.
- Ein PTAS könnte z. B. eine Laufzeit von $O(|I|^{1/\epsilon})$ haben.
- Diese ist für jedes feste $\epsilon > 0$ polynomiell in $|I|$.
- Diese Laufzeit ist bei einem FPTAS jedoch nicht erlaubt, weil sie nicht polynomiell in $1/\epsilon$ beschränkt ist.
- Eine erlaubte Laufzeit eines FPTAS ist z. B. $O(|I|^3/\epsilon^2)$.

- Ein PTAS impliziert, dass es für das entsprechende Problem für jede Konstante $\epsilon > 0$ einen $(1+\epsilon)$- bzw. $(1-\epsilon)$-Approximationsalgorithmus gibt.
- Der Grad des Polynoms hängt aber im Allgemeinen vom gewählten ϵ ab.
- Ist die Laufzeit zum Beispiel $O(|I|^{1/\epsilon})$, so ergibt sich für $\epsilon = 0.01$ eine Laufzeit von $O(|I|^{100})$.
- Eine solche Laufzeit ist nur von theoretischem Interesse.
- Bei einem FPTAS ist es hingegen oft so, dass man auch für kleine ϵ eine passable Laufzeit erhält.

Alternativer DP-Ansatz für Rucksackproblem

- Für die Konstruktion eines FPTAS benötigen wir zunächst einen anderen DP-Ansatz.
- Wir betrachten hier (o.B.d.A.) nur den optimalen Zielfunktionswert.
- Teilproblem: Finde unter allen Teilmengen der Objekte $1, \ldots, i$ mit Gesamtnutzen $\geq p$ eine mit dem kleinsten Gewicht.
- Es sei $W(i, p)$ das Gewicht solch einer Teilmenge. Existiert keine solche Teilmenge, so setzen wir $W(i, p) = \infty$.
- Wir lösen das Rucksackproblem, indem wir eine Tabelle konstruieren, die die $W(i, p)$-Werte enthält.
- Obere Schranke für den erreichbaren Nutzen: nP mit $P := \max_{i \in \{1,\ldots,n\}} p_i$.

- Seien die Werte $W(i-1,p)$ für ein i und alle $p \in \{0,\dots,nP\}$ bekannt. Wie können wir $W(i,p)$ aus den bekannten Werten berechnen?
- Gesucht ist eine Teilmenge $I \subseteq \{1,\dots,i\}$ mit $\sum_{i \in I} p_i \geq p$ und kleinstmöglichem Gewicht. Wir unterscheiden nun zwei Fälle.
- $i \notin I$
  - Dann gilt $I \subseteq \{1,\dots,i-1\}$ mit $\sum_{i \in I} p_i \geq p$.
  - Unter allen diesen Teilmengen besitzt I minimales Gewicht.
  - Damit gilt $W(i,p) = W(i-1,p)$.
- $i \in I$
  - Dann ist $I \setminus \{i\}$ eine Teilmenge von $\{1,\dots,i-1\}$ mit Nutzen von mindestens $p - p_i$.
  - Unter allen Teilmengen, die diese Bedingung erfüllen, hat $I \setminus \{i\}$ minimales Gewicht.
  - Es gilt also $w(I \setminus \{i\}) = W(i-1,p-p_i)$ und somit $W(i,p) = W(i-1,p-p_i) + w_i$.

Algorithmus 6.37

1. $W(i,p) := 0$ *für* $i \in \{1,\ldots,n\}$ *und* $p \leq 0$.
2. $P := \max_{i \in \{1,\ldots,n\}} p_i$
3. $W(1,p) := w_1$ *für* $p = 1,\ldots,p_1$
4. $W(1,p) := \infty$ *für* $p = p_1 + 1,\ldots,nP$
5. $W(i,p) = \min\{W(i-1,p), W(i-1,p-p_i) + w_i\}$ *für* $i = 2,\ldots,n$ *und* $p = 1,\ldots,nP$
6. **return** $\max p \in \{1,\ldots,nP\}$ *mit* $W(n,p) \leq C$.

Lemma 6.38

Algorithmus 6.37 bestimmt in Zeit $O(n^2 P)$ den maximalen Nutzen für eine Instanz des Rucksackproblems.

Idee: FPTAS

- Algorithmus 6.37 ist effizient für kleine Nutzenwerte (polynomiell in der Eingabelänge beschränkt).
- Idee: große Nutzenwerten skalieren, d. h. alle durch dieselbe Zahl K teilen.
- Der maximal erreichbare Nutzen wird dann ebenfalls um den Faktor K kleiner.
- optimale Lösung ändert sich nicht
- Trick: Schneide die Nachkommastellen der skalierten Nutzenwerte ab!
- Wähle dabei K so, dass Algorithmus 6.37 polynomielle Laufzeit hat.
- Abhängig von K ergibt sich dann auch eine Approximationsgüte.

FPTAS für Rucksackproblem

Algorithmus 6.39

Eingabe sei $(\mathcal{I}, \epsilon)$ *mit* $\mathcal{I} = (p_1, \dots, p_n, w_1, \dots, w_n, C)$.

1. $P := \max_{i \in \{1,\dots,n\}} p_i$
2. $K := \frac{\epsilon P}{n}$
3. $p'_i := \left\lfloor \frac{p_i}{K} \right\rfloor$ *für* $i = 1, \dots, n$
4. *Berechne mit Algorithmus 6.37 eine optimale Lösung für die Instanz* $\mathcal{I} = (p'_1, \dots, p'_n, w_1, \dots, w_n, C)$.

Satz 6.40

Algorithmus 6.39 ist ein FPTAS für das Rucksackproblem mit einer Laufzeit von $O(n^3/\epsilon)$.

Beweis für FPTAS

Beweis.

Laufzeit:

- Wird durch den Aufruf von Algorithmus 6.37 dominiert.
- Sei $P' = \max_{i \in \{1,\ldots,n\}} p'_i$.
- Es gilt:

$$P' = \max_{i \in \{1,\ldots,n\}} \left\lfloor \frac{p_i}{K} \right\rfloor = \left\lfloor \frac{P}{K} \right\rfloor = \left\lfloor \frac{n}{\epsilon} \right\rfloor \leq \frac{n}{\epsilon}.$$

- Somit beträgt die Laufzeit $O(n^2 P') = O(n^3/\epsilon)$.

Approximation:

- Sei $I' \subseteq \{1, \ldots, n\}$ eine optimale Lösung für das skalierte Problem mit Nutzenwerten $p'_1, \ldots, p'_n$.
- Sei $I \subseteq \{1, \ldots, n\}$ eine optimale Lösung für das originäre Problem mit Nutzenwerten $p_1, \ldots, p_n$.

Fortsetzung Beweis.

- I' ist auch optimal für die Nutzenwerte $p_1^*, \ldots, p_n^*$ mit $p_i^* = K \cdot p_i'$.
- Für $J \subseteq \{1, \ldots, n\}$ sei

$$p(J) = \sum_{i \in J} p_i \quad \text{und} \quad p^*(J) = \sum_{i \in J} p_i^*.$$

- Dann ist $p(I')$ der Wert der Lösung von Algorithmus 6.39 und $p(I)$ der Wert OPT einer optimalen Lösung.
- Die Nutzenwerte p_i und p_i^* weichen nicht stark voneinander ab. Es gilt

$$p_i^* = K \left\lfloor \frac{p_i}{K} \right\rfloor \geq K \left(\frac{p_i}{K} - 1 \right) = p_i - K$$

und

$$p_i^* = K \left\lfloor \frac{p_i}{K} \right\rfloor \leq p_i.$$

Fortsetzung Beweis.

- Dementsprechend gilt für jede Teilmenge $J \subseteq \{1, \ldots, n\}$

$$p(J) - nK \leq p^*(J) \leq p(J).$$

- I' ist eine optimale Lösung für die Nutzenwerte $p_1^*, \ldots, p_n^*$ (siehe oben).
- Es gilt also $p^*(I') \geq p^*(I)$ und somit

$$p(I') \geq p^*(I') \geq p^*(I) \geq p(I) - nK.$$

- Da jedes Objekt alleine in den Rucksack passt, gilt $p(I) \geq P$ und damit

$$\frac{p(I')}{\mathrm{OPT}} = \frac{p(I')}{p(I)} \geq \frac{p(I) - nK}{p(I)} = 1 - \frac{\epsilon P}{p(I)} \geq 1 - \epsilon.$$

Diskussion

- Beim FPTAS Abrundung der Nutzenwerte so, dass der pseudopolynomielle Algorithmus die abgerundete Instanz in polynomieller Zeit lösen kann.
- gute Approximation der optimalen Lösung bezüglich der eigentlichen Nutzenwerte
- Technik zum Entwurf eines FPTAS lässt sich für andere Probleme anwenden, für die es einen pseudopolynomiellen Algorithmus gibt.
- Geht das auch mit Runden der Gewichtswerte?
- eher nicht, optimale Lösung dann u. U. nicht mehr zulässig, evtl. sehr schlechte Approximation
- Faustregel: guter Ansatz, wenn pseudopolynomial bzgl. Koeffizienten in der Zielfunktion.
- Daher auch der Ansatz mit Algorithmus 6.37 statt 6.32.

Max-SAT

Definition 6.41

Das Optimierungsproblem MAX-SAT lautet:

- Instanzen: Gegeben ist eine aussagenlogische Formel

$$\Phi = C_1 \wedge C_2 \wedge \dots C_m$$

  in KNF
  - mit den Klauseln $C_1, \dots, C_m$,
  - in den Variablen $x_1, \dots, x_n$,
  - sowie Gewichten $w_1, \dots, w_m \in \mathbb{R}_+$.
- Lösungen: alle Belegungen der Variablen $x_1, \dots, x_n$.
- Zielfunktion: maximiere das Gesamtgewicht der durch die Beleguung erfüllten Klauseln.

Randomisierter Approximationsalgorithmus

Definition 6.42

- Ein randomisierter Approximationsalgorithmus A für ein Minimierungs- oder Maximierungsproblem Π erreicht einen Approximationsfaktor von $r \geq 1$ bzw. $r \leq 1$, wenn

$$E(w_A(I)) \leq r \cdot \mathrm{OPT}(I) \quad \text{bzw.} \quad E(w_A(I)) \geq r \cdot \mathrm{OPT}(I)$$

  für alle Instanzen $\mathcal{I}_\Pi$ gilt. Wir sagen dann, dass A ein randomisierter r-Approximationsalgorithmus ist.

- In vielen Fällen hängt der Approximationsfaktor von der Eingabelänge ab. Dann ist $r : \mathbb{N} \to [0, \infty)$ eine Funktion und es muss

$$E(w_A(I)) \leq r(|I|) \cdot \mathrm{OPT}(I) \quad \text{bzw.} \quad E(w_A(I)) \geq r(|I|) \cdot \mathrm{OPT}(I)$$

  für alle Instanzen I gelten.

Randomisierter Approximationsalgorithmus für Max-SAT

Algorithmus 6.43 (RandMaxSat)

Erzeuge eine Belegung $B : \{x_1, \ldots, x_n\} \to \{0, 1\}$ *wie folgt:*

Setze jede Variable x_i *unabhängig von den anderen Variablen mit einer Wahrscheinlichkeit von* $\frac{1}{2}$ *auf* 0 *bzw.* 1.

Satz 6.44

Algorithmus 6.43 ist ein randomisierter $\frac{1}{2}$*-Approximationsalgorithmus für Max-SAT.*

Beweis.

- Wir definieren für jede Klausel C_j eine Zufallsvariable

$$Y_j = \begin{cases} w_j & \text{falls } B \text{ Klausel } C_j \text{ erfüllt,} \\ 0 & \text{sonst.} \end{cases}$$

- Dann gilt für das Gesamtgewicht Y der erfüllten Klauseln

$$E(Y) = E\left(\sum_{j=1}^{m} Y_j\right) = \sum_{j=1}^{m} E(Y_j) = \sum_{j=1}^{m} w_j \cdot P(B \text{ erfüllt } C_j).$$

- Wann wird C_j nicht erfüllt?
  - Wenn alle positiven Literale in C_j auf 0 und
  - alle negativen Literale auf 1 gesetzt sind.

  Bezeichne l_j die Anzahl an Literalen in C_j.

Fortsetzung Beweis.

- Damit folgt

$$P(B \text{ erfüllt } C_j \text{ nicht}) = \left(\frac{1}{2}\right)^{l_j} \leq \frac{1}{2}.$$

- Das Gesamtgewicht aller Klauseln ist eine obere Schranke für OPT. Damit folgt

$$E(Y) = \sum_{j=1}^{m} w_j \cdot P(B \text{ erfüllt } C_j) \geq \sum_{j=1}^{m} w_j \cdot \frac{1}{2} \geq \frac{1}{2} \cdot \text{OPT}.$$

Bemerkungen:

- Aus Beweis folgt: Der Approximationsfaktor wird besser, je länger die Klauseln sind.
- Für MAX-3-SAT erhalten wir einen Approximationsfaktor von $\frac{7}{8}$.

Max-SAT als ILP

Maximiere

$$\sum_{j=1}^{m} w_j z_j$$

unter den Nebenbedingungen

$$\begin{aligned} \sum_{i \in P_j} y_i + \sum_{i \in N_j} (1 - y_i) &\geq z_j \quad \text{für } j = 1, \ldots, m \\ y_i &\in \{0, 1\} \quad \text{für } i = 1, \ldots, n \\ z_j &\in \{0, 1\} \quad \text{für } j = 1, \ldots, m \end{aligned}$$

Randomisiertes Runden für Max-SAT

Algorithmus 6.45

Erzeuge eine Belegung $B : \{x_1, \ldots, x_n\} \to \{0, 1\}$ *wie folgt:*

1. *Berechne eine Lösung für die LP-Relaxation (also* $y_j, z_j \in [0, 1]$*) von Max-SAT.*
   Es sei $(\mathbf{y}, \mathbf{z})$ *eine optimale Lösung der LP-Relaxation.*
2. *Setze jede Variable* x_i *unabhängig von den anderen Variablen mit einer Wahrscheinlichkeit von* y_i *auf* 1 *bzw. mit einer Wahrscheinlichkeit von* $1 - y_i$ *auf* 0*.*

Satz 6.46

Algorithmus 6.45 ist ein randomisierter $\left(1 - \frac{1}{e}\right)$*-Approximationsalgorithmus für Max-SAT.*

Beweis.

- Sei P_j die Menge der positiven Literale von C_j und N_j die Menge der negativen Literale.
- Damit gilt

$$P(B \text{ erfüllt } C_j \text{ nicht}) = \prod_{i \in P_j} (1 - y_i) \cdot \prod_{i \in N_j} y_i .$$

- Wir nutzen die Ungleichung vom arithmetischen und geometrischen Mittel

$$\sqrt[n]{\prod_{i=1}^{n} a_i} \leq \frac{1}{n} \sum_{i=1}^{n} a_i \quad \text{bzw.} \quad \prod_{i=1}^{n} a_i \leq \left(\frac{1}{n} \sum_{i=1}^{n} a_i \right)^n .$$

Fortsetzung Beweis.

- Angewendet mit $|C_j| = |P_j| + |N_j|$ ergibt sich

$$
\begin{aligned}
\prod_{i\in P_j}(1-y_i)\cdot\prod_{i\in N_j} y_i &\le \left(\frac{1}{|C_j|}\left(\sum_{i\in P_j}(1-y_i)+\sum_{i\in N_j} y_i\right)\right)^{|C_j|} \\
&= \left(\frac{1}{|C_j|}\left(|C_j|+\sum_{i\in P_j}(-y_i)+\sum_{i\in N_j}(y_i-1)\right)\right)^{|C_j|} \\
&= \left(1-\frac{1}{|C_j|}\left(\sum_{i\in P_j} y_i+\sum_{i\in N_j}(1-y_i)\right)\right)^{|C_j|}.
\end{aligned}
$$

Fortsetzung Beweis.

- Das LP erfüllt die Nebenbedingung $\sum_{i \in P_j} y_i + \sum_{i \in N_j}(1 - y_i) \geq z_j$. Damit folgt

$$P(C_j \text{ nicht erfüllt}) \leq \left(1 - \frac{z_j}{|C_j|}\right)^{|C_j|}$$

bzw.

$$P(C_j \text{ erfüllt}) \geq 1 - \left(1 - \frac{z_j}{|C_j|}\right)^{|C_j|}.$$

- Die Funktion $f(x) = 1 - \left(1 - \frac{x}{|C_j|}\right)^{|C_j|}$ ist konkav. Damit liegt die Gerade g, die durch die Punkte $(0, f(0))$ und $(1, f(1))$ geht, nirgendwo überhalb von f.
- Wegen $f(0) = 0$ gilt

$$g(x) = f(1) \cdot x = \left(1 - \left(1 - \frac{1}{|C_j|}\right)^{|C_j|}\right) \cdot x.$$

Fortsetzung Beweis.

- Damit folgt:

$$P(C_j \text{ erfüllt}) \geq f(z_j) \geq g(z_j) = \left(1 - \left(1 - \frac{1}{|C_j|}\right)^{|C_j|}\right) \cdot z_j.$$

- Sei Y wieder das Gesamtgewicht der erfüllten Klauseln.

$$\begin{aligned} E(Y) &= \sum_{j=1}^{m} w_j \cdot P(C_j \text{ erfüllt}) \\ &\geq \sum_{j=1}^{m} w_j \left(1 - \left(1 - \frac{1}{|C_j|}\right)^{|C_j|}\right) \cdot z_j \\ &\geq \min_{k \geq 1} \left(1 - \left(1 - \frac{1}{k}\right)^{k}\right) \cdot \sum_{j=1}^{m} w_j z_j \end{aligned}$$

Fortsetzung Beweis.

$$\begin{aligned} &= \min_{k\geq 1}\left(1-\left(1-\frac{1}{k}\right)^k\right) z_{LP} \\ &\geq \min_{k\geq 1}\left(1-\left(1-\frac{1}{k}\right)^k\right) \mathrm{OPT}. \end{aligned}$$

- Die Funktion $1-(1-\frac{1}{k})^k$ ist streng monoton fallend und geht von oben gegen $1-\frac{1}{\mathrm{e}} \approx 0.632$. Damit erhalten wir

$$E(Y) \geq \left(1-\frac{1}{\mathrm{e}}\right) \cdot \mathrm{OPT}.$$

Zusammenfassung

- Greedy-Algorithmen als Ansatz für Approximationsalgorithmen
- Auf Matroiden liefern Greedy-Algorithmen optimale Lösungen.
- Beweis der Approximationsgüte: Schranken für optimale Lösung, approximative Lösung und diese in Beziehung setzen.
- Dynamische Programmierung und Pseudopolynomialität
- FPTAS durch geeignetes Runden und Nutzung eines pseudopolynomialen Algorithmus
- Randomisierte Approximation: Runden auf Basis der Lösung einer LP-Relaxation