Karlsruher Institut für Technologie

Theoretische Grundlagen der Informatik

Vorlesung am 15.12.2022

Torsten Ueckerdt | 15. Dezember 2022

Ein Blick über den Tellerrand

Karlsruher Institut für Technologie

Letzte Vorlesung:

- Entscheidungsprobleme außerhalb von $\mathcal{P}$ und $\mathcal{NP}$

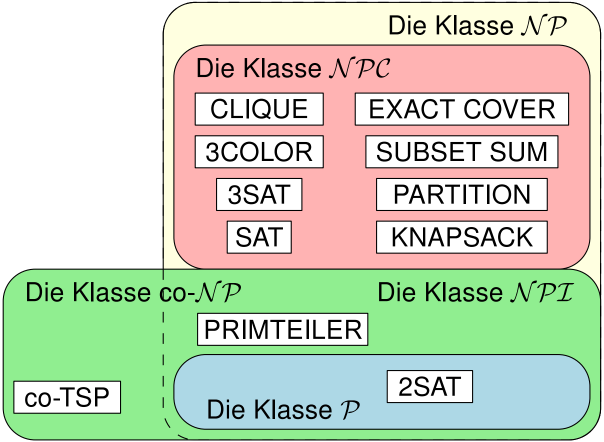
Gleich:

- Probleme die nicht Entscheidungsprobleme sind

Danach:

- Pseudopolynomiale Algorithmen
- Approximationsalgorithmen

Vermutete Situation



Die Klasse PSPACE

Die Klasse **PSPACE**

PSPACE ist die Klasse aller Sprachen / Entscheidungsprobleme die von einer deterministischen Turingmaschine mit polynomiellem Platz entschieden werden können.

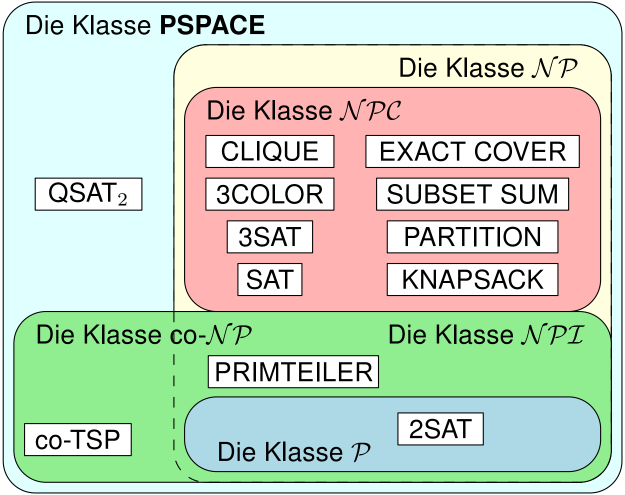
Die Klasse PSPACE

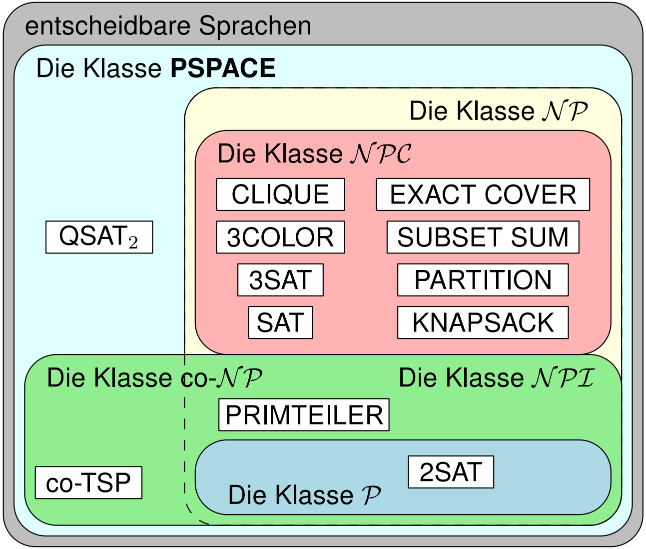
Die Klasse **PSPACE**

PSPACE ist die Klasse aller Sprachen / Entscheidungsprobleme die von einer deterministischen Turingmaschine mit polynomiellem Platz entschieden werden können.

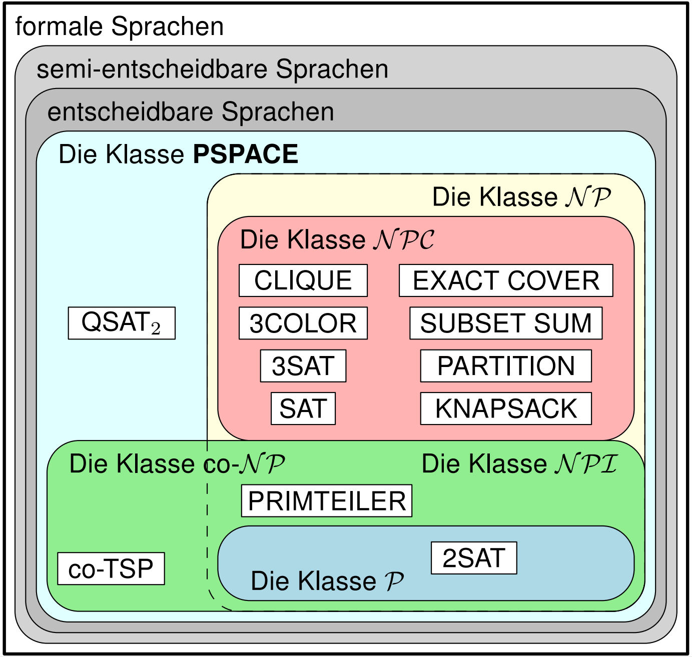
- Betrachte SAT Instanz (U, C) und eine Aufteilung der Variablen in disjunkte Mengen U_1 und U_2.
- Die Instanz heiße *lösbar* wenn
  - es eine Wahrheitsbelegung t_1 von U_1 gibt,
  - so dass für alle Wahrheitsbelegungen t_2 von U_2
  - alle Klauseln in C erfüllt sind.
- $\rightsquigarrow$ bezeichnet als QSAT_2 oder $\exists\forall$SAT

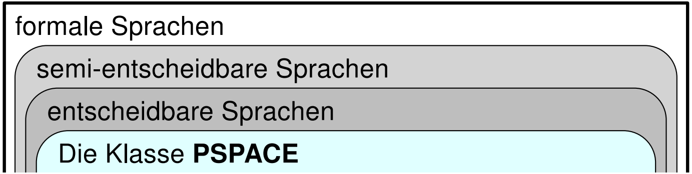
QSAT_2 liegt in **PSPACE** da eine DTM alle Wahrheitsbelegungen t_1 und für jede solche, alle Wahrheitsbelegungen t_2 überprüfen kann.


Die Klasse **PSPACE**
Die Klasse $\mathcal{NP}$
Die Klasse $\mathcal{NPC}$
CLIQUE
EXACT COVER
QSAT$_2$
3COLOR
SUBSET SUM
3SAT
PARTITION
SAT
KNAPSACK
Die Klasse co-$\mathcal{NP}$
Die Klasse $\mathcal{NPI}$
PRIMTEILER
2SAT
co-TSP
Die Klasse $\mathcal{P}$



entscheidbare Sprachen
Die Klasse **PSPACE**
Die Klasse $\mathcal{NP}$
Die Klasse $\mathcal{NPC}$
CLIQUE
EXACT COVER
3COLOR
SUBSET SUM
3SAT
PARTITION
SAT
KNAPSACK
QSAT_2
Die Klasse co-$\mathcal{NP}$
Die Klasse $\mathcal{NPI}$
PRIMTEILER
2SAT
Die Klasse $\mathcal{P}$
co-TSP


formale Sprachen
semi-entscheidbare Sprachen
entscheidbare Sprachen
Die Klasse **PSPACE**
Die Klasse $\mathcal{NP}$
Die Klasse $\mathcal{NPC}$
CLIQUE
EXACT COVER
3COLOR
SUBSET SUM
3SAT
PARTITION
SAT
KNAPSACK
$QSAT_2$
Die Klasse co-$\mathcal{NP}$
Die Klasse $\mathcal{NPI}$
PRIMTEILER
co-TSP
2SAT
Die Klasse $\mathcal{P}$



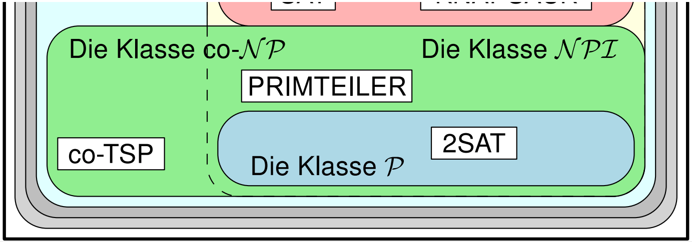
Testen Sie sich:

Lesen Sie den Artikel

A Short Guide to Hard Problems

auf Quanta Magazine

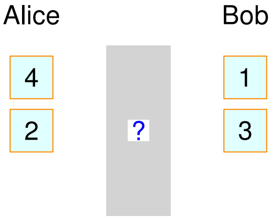
https://www.quantamagazine.org/a-short-guide-to-hard-problems-20180716/



Ein offenes Problem

Spiel **Leben und Tod**:

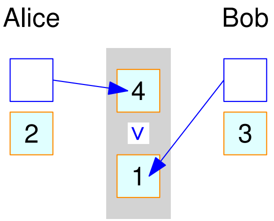
- 2 Spieler haben je einen Stapel von Karten.
- Beide spielen ihre oberste Karte.
- Der Spieler mit der höheren Karte bekommt beide Karten unter seinen / ihrer Stapel.
- Spieler verliert wenn er / sie keine Karten mehr hat.



Ein offenes Problem

Spiel **Leben und Tod**:

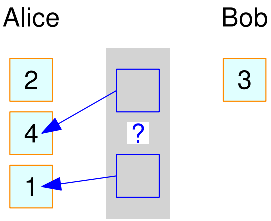
- 2 Spieler haben je einen Stapel von Karten.
- Beide spielen ihre oberste Karte.
- Der Spieler mit der höheren Karte bekommt beide Karten unter seinen / ihrer Stapel.
- Spieler verliert wenn er / sie keine Karten mehr hat.



Ein offenes Problem

Spiel **Leben und Tod**:

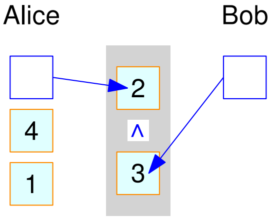
- 2 Spieler haben je einen Stapel von Karten.
- Beide spielen ihre oberste Karte.
- Der Spieler mit der höheren Karte bekommt beide Karten unter seinen / ihrer Stapel.
- Spieler verliert wenn er / sie keine Karten mehr hat.



Ein offenes Problem

Spiel **Leben und Tod**:

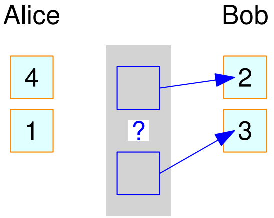
- 2 Spieler haben je einen Stapel von Karten.
- Beide spielen ihre oberste Karte.
- Der Spieler mit der höheren Karte bekommt beide Karten unter seinen / ihrer Stapel.
- Spieler verliert wenn er / sie keine Karten mehr hat.



Ein offenes Problem

Spiel **Leben und Tod**:

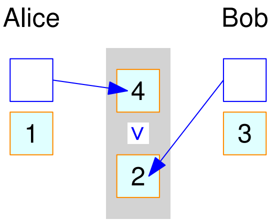
- 2 Spieler haben je einen Stapel von Karten.
- Beide spielen ihre oberste Karte.
- Der Spieler mit der höheren Karte bekommt beide Karten unter seinen / ihrer Stapel.
- Spieler verliert wenn er / sie keine Karten mehr hat.



Ein offenes Problem

Spiel **Leben und Tod**:

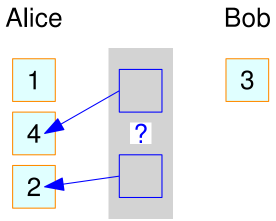
- 2 Spieler haben je einen Stapel von Karten.
- Beide spielen ihre oberste Karte.
- Der Spieler mit der höheren Karte bekommt beide Karten unter seinen / ihrer Stapel.
- Spieler verliert wenn er / sie keine Karten mehr hat.



Ein offenes Problem

Spiel **Leben und Tod**:

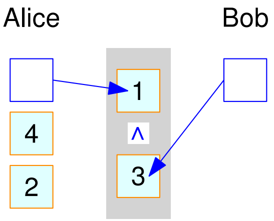
- 2 Spieler haben je einen Stapel von Karten.
- Beide spielen ihre oberste Karte.
- Der Spieler mit der höheren Karte bekommt beide Karten unter seinen / ihrer Stapel.
- Spieler verliert wenn er / sie keine Karten mehr hat.



Ein offenes Problem

Spiel **Leben und Tod**:

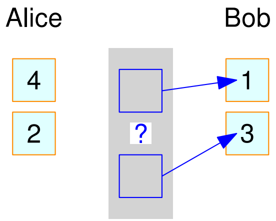
- 2 Spieler haben je einen Stapel von Karten.
- Beide spielen ihre oberste Karte.
- Der Spieler mit der höheren Karte bekommt beide Karten unter seinen / ihrer Stapel.
- Spieler verliert wenn er / sie keine Karten mehr hat.



Ein offenes Problem

Spiel **Leben und Tod**:

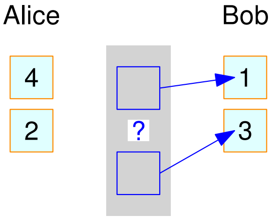
- 2 Spieler haben je einen Stapel von Karten.
- Beide spielen ihre oberste Karte.
- Der Spieler mit der höheren Karte bekommt beide Karten unter seinen / ihrer Stapel.
- Spieler verliert wenn er / sie keine Karten mehr hat.



Ein offenes Problem

Spiel **Leben und Tod**:

- 2 Spieler haben je einen Stapel von Karten.
- Beide spielen ihre oberste Karte.
- Der Spieler mit der höheren Karte bekommt beide Karten unter seinen / ihrer Stapel.
- Spieler verliert wenn er / sie keine Karten mehr hat.

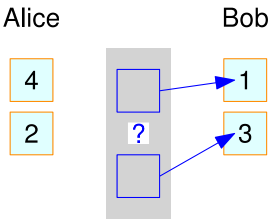


Entweder: Spieler mit höchster Karte gewinnt
Oder: Das Spiel geht unendlich lange

Ein offenes Problem

Spiel **Leben und Tod**:

- 2 Spieler haben je einen Stapel von Karten.
- Beide spielen ihre oberste Karte.
- Der Spieler mit der höheren Karte bekommt beide Karten unter seinen / ihrer Stapel.
- Spieler verliert wenn er / sie keine Karten mehr hat.



Entweder: Spieler mit höchster Karte gewinnt
Oder: Das Spiel geht unendlich lange

Offenes Problem

Kann man in Polynomialzeit entscheiden, ob das Spiel unendlich geht?

Wir wissen bisher nur, dass es in **PSPACE** ist.

Ein Blick über den Tellerrand

Bisher:

- Entscheidungsprobleme außerhalb von $\mathcal{P}$ und $\mathcal{NP}$

Jetzt:

- Probleme die nicht Entscheidungsprobleme sind

Danach:

- Pseudopolynomiale Algorithmen
- Approximationsalgorithmen

Suchprobleme

Ein **Suchproblem** Π wird beschrieben durch

- die Menge D_Π der Instanzen und
- für $I \in D_\Pi$ die Menge $S_\Pi(I)$ *aller* Lösungen von I.

Die **Lösung** eines Suchproblems für eine Instanz $I \in D_\Pi$ ist

- ein beliebiges Element aus $S_\Pi(I)$ falls $S_\Pi(I) \neq \emptyset$
- $\emptyset$ sonst

Beispiel: TSP-Suchproblem

TSP-Suchproblem (Variante 1)

Gegeben: Graph $G = (V, E)$,
Gewichtsfunktion $c\colon E \longrightarrow \mathbb{Q}$

Aufgabe: Gib eine optimale Tour zu G bezüglich c an.

TSP-Suchproblem (Variante 2)

Gegeben: Graph $G = (V, E)$,
Gewichtsfunktion $c\colon E \longrightarrow \mathbb{Q}$,
Parameter $k \in \mathbb{Q}$

Aufgabe: Gib eine Tour zu G bzgl. c mit Länge höchstens k an.

Variante 1: $S_\Pi(G)$ ist die Menge aller optimalen Touren zu G.

Variante 2: $S_\Pi(G)$ ist die Menge aller Touren der Länge höchstens k.

Beispiel: Hamilton-Kreis Suchproblem

Gegeben ist ein Graph $G = (V, E)$.

Ein Hamilton-Kreis in G ist eine zyklische Permutation π auf V, so dass

$$\{\pi(i), \pi(i+1)\} \in E \text{ für } 1 \leq i \leq n \text{ ist.}$$

Hamilton-Kreis Suchproblem

Gegeben: Ein ungerichteter, ungewichteter Graph $G = (V, E)$

Aufgabe: Gib einen Hamilton-Kreis in G an, falls einer existiert.

Bemerkung: $S_\Pi(G)$ ist die Menge aller Hamilton-Kreise in G.

$\mathcal{NP}$-Schwere bei Suchproblemen

Für Suchprobleme gibt es (ähnlich wie zu Entscheidungsproblemen):

- Eine Variante der Orakel-Turing-Maschine.
- Eine Klasse $\mathcal{NP}$ der Suchprobleme, die in polynomieller Zeit von einer OTM gelöst werden können.
- Eine Klasse $\mathcal{P}$ der Suchprobleme, die in polynomieller Zeit von einer deterministischen TM gelöst werden können.
- Den Begriff der $\mathcal{NP}$-Schwere, und sogenannte Turingreduktionen $\propto_T$.
- Die Frage ob $\mathcal{P} = \mathcal{NP}$?

$\mathcal{NP}$-Schwere bei Suchproblemen

Für Suchprobleme gibt es (ähnlich wie zu Entscheidungsproblemen):

- Eine Variante der Orakel-Turing-Maschine.
- Eine Klasse $\mathcal{NP}$ der Suchprobleme, die in polynomieller Zeit von einer OTM gelöst werden können.
- Eine Klasse $\mathcal{P}$ der Suchprobleme, die in polynomieller Zeit von einer deterministischen TM gelöst werden können.
- Den Begriff der $\mathcal{NP}$-Schwere, und sogenannte Turingreduktionen $\propto_T$.
- Die Frage ob $\mathcal{P} = \mathcal{NP}$?

Bemerkung:

- Die Suchprobleme für Hamilton-Kreis und TSP sind $\mathcal{NP}$-schwer.

Aufzählungsprobleme

Ein **Aufzählungsproblem** Π ist gegeben durch

- die Menge D_Π der Problembeispiele und
- für $I \in D_\Pi$ die Menge $S_\Pi(I)$ aller Lösungen von I.

Die **Lösung** eines Aufzählungsproblems für eine Instanz $I \in D_\Pi$ ist

- die Angabe der Kardinalität von $S_\Pi(I)$, d.h. von $|S_\Pi(I)|$.

Beispiel: Hamilton-Kreis Aufzählungsproblem

Karlsruher Institut für Technologie

Hamilton-Kreis Aufzählungsproblem

Gegeben: Ein ungerichteter, ungewichteter Graph $G = (V, E)$

Aufgabe: Wieviele Hamilton-Kreise gibt es in G?

Beispiel: Hamilton-Kreis Aufzählungsproblem

Hamilton-Kreis Aufzählungsproblem

Gegeben: Ein ungerichteter, ungewichteter Graph $G = (V, E)$

Aufgabe: Wieviele Hamilton-Kreise gibt es in G?

Bemerkung:

- Aufzählprobleme sind sehr schwierig!
- Beispiel: Permanente einer Matrix (Anzahl der perfekten Matchings in einem bipartiten Graphen) ist $\#P$-vollständig.
- **Satz von Toda.**
  Jedes Problem in der **Polynomiellen Hierarchie** (z.B. QSAT_2) kann durch einen Aufruf eines $\#P$-vollständigen Problems gelöst werden.

Optimierungsprobleme

Ein **Optimierungsproblem** Π ist gegeben durch

- die Menge D_Π der Problembeispiele,
- für $I \in D_\Pi$ die Menge $S_\Pi(I)$ aller Lösungen von I und
- für $L \in S_\Pi(I)$ der Wert $w(L) \in \mathbb{R}$ der Lösung L.

Die **Lösung** eines Optimierungsproblems für eine Instanz $I \in D_\Pi$ ist

Minimierungsproblem: die Angabe einer Lösung minimalen Werts

Maximierungsproblem: die Angabe einer Lösung maximalen Werts

Verallgemeinerte $\mathcal{NP}$-Schwere

- Wir nennen ein Problem $\mathcal{NP}$-schwer, wenn es mindestens so schwer ist, wie alle $\mathcal{NP}$-vollständigen Probleme.

Darunter fallen auch

- Optimierungsprobleme, für die das zugehörige Entscheidungsproblem $\mathcal{NP}$-vollständig ist.
- Entscheidungsprobleme Π, für die gilt, dass für alle Probleme $\Pi' \in \mathcal{NP}$ gilt $\Pi' \propto \Pi$, aber für die nicht klar ist, ob $\Pi \in \mathcal{NP}$.

Klar ist, dass ein $\mathcal{NP}$-vollständiges Problem auch $\mathcal{NP}$-schwer ist.

Das Problem INTEGER PROGRAMMING

Problem INTEGER PROGRAMMING

Gegeben: $a_{ij} \in \mathbb{Z}$, $b_i, c_j \in \mathbb{Z}, 1 \leq i \leq m$, $1 \leq j \leq n$, $B \in \mathbb{Z}$.

Frage: Existieren $x_1, \ldots, x_n \in \mathbb{N}_0$, so dass

$$\sum_{j=1}^{n} c_j \cdot x_j = B \text{ und}$$

$$\underbrace{\sum_{j=1}^{n} a_{ij} \cdot x_j \leq b_i \text{ für } 1 \leq i \leq m}_{A \cdot \bar{x} \leq \bar{b}}?$$

Das Problem INTEGER PROGRAMMING

Karlsruher Institut für Technologie

Problem INTEGER PROGRAMMING

Gegeben: $a_{ij} \in \mathbb{Z}$, $b_i, c_j \in \mathbb{Z}$,$1 \leq i \leq m$, $1 \leq j \leq n$, $B \in \mathbb{Z}$.

Frage: Existieren $x_1, \ldots, x_n \in \mathbb{N}_0$, so dass

$$\sum_{j=1}^{n} c_j \cdot x_j = B \text{ und}$$

$$\underbrace{\sum_{j=1}^{n} a_{ij} \cdot x_j \leq b_i \text{ für } 1 \leq i \leq m}_{A \cdot \bar{x} \leq \bar{b}}?$$

Satz.

Das Problem INTEGER PROGRAMMING ist $\mathcal{NP}$-schwer.

INTEGER PROGRAMMING ist $\mathcal{NP}$-schwer

Beweis:

Zeige Reduktion SUBSET SUM $\propto$ INTEGER PROGRAMMING.

- Sei $(M, w\colon M \to \mathbb{N}_0, K \in \mathbb{N}_0)$ beliebige Instanz von SUBSET SUM. Sei $n = |M|$ und o.B.d.A. $M = \{1, \dots, n\}$.

Integer Programming:
$\exists\, x_1, \dots, x_n \in \mathbb{N}_0$, dass
- $\sum\limits_{j=1}^{n} c_j \cdot x_j = B$ und
- $\sum_{j=1}^{n} a_{ij} \cdot x_j \leq b_i$ für $1 \leq i \leq m$?

INTEGER PROGRAMMING ist $\mathcal{NP}$-schwer

Karlsruher Institut für Technologie

Beweis:

Zeige Reduktion SUBSET SUM $\propto$ INTEGER PROGRAMMING.

- Sei $(M, w\colon M \to \mathbb{N}_0, K \in \mathbb{N}_0)$ beliebige Instanz von SUBSET SUM. Sei $n = |M|$ und o.B.d.A. $M = \{1, \ldots, n\}$.
- **Idee:** Variable $x_j = 1 \leftrightarrow j \in M'$ und Variable $x_j = 0 \leftrightarrow j \notin M'$

Wähle $c_j := w(j)$. Dann $\sum_{j=1}^{n} c_j \cdot x_j = \sum_{j \in M'} w(j) = w(M')$.

Wähle $B := K$. Dann $\sum_{j=1}^{n} c_j \cdot x_j = B \Longleftrightarrow \sum_{j \in M'} w(j) = w(M') = K$

Integer Programming:
$\exists\, x_1, \ldots, x_n \in \mathbb{N}_0$, dass
- $\sum_{j=1}^{n} c_j \cdot x_j = B$ und
- $\sum_{j=1}^{n} a_{ij} \cdot x_j \leq b_i$ für $1 \leq i \leq m$?

INTEGER PROGRAMMING ist $\mathcal{NP}$-schwer

Beweis:

Zeige Reduktion SUBSET SUM $\propto$ INTEGER PROGRAMMING.

- Sei $(M, w\colon M \to \mathbb{N}_0, K \in \mathbb{N}_0)$ beliebige Instanz von SUBSET SUM. Sei $n = |M|$ und o.B.d.A. $M = \{1, \ldots, n\}$.
- **Idee:** Variable $x_j = 1 \leftrightarrow j \in M'$ und Variable $x_j = 0 \leftrightarrow j \notin M'$
- **Idee:** Bedingung $\sum_{j=1}^{n} a_{ij} \cdot x_j \leq b_i \leftrightarrow$ Element i höchstens 1x gewählt

Wähle $m = n, (a_{ij}) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & 1 \end{pmatrix}$ Dann $\sum_{j=1}^{n} a_{ij} \cdot x_j = x_i$.

Wähle $b_i = 1$. Dann $\sum_{j=1}^{n} a_{ij} \cdot x_j \leq b_i \iff x_i \leq 1$

Integer Programming:

$\exists\, x_1, \ldots, x_n \in \mathbb{N}_0$, dass

- $\sum_{j=1}^{n} c_j \cdot x_j = B$ und
- $\sum_{j=1}^{n} a_{ij} \cdot x_j \leq b_i$ für $1 \leq i \leq m$?

INTEGER PROGRAMMING ist $\mathcal{NP}$-schwer

Beweis:

Zeige Reduktion SUBSET SUM $\propto$ INTEGER PROGRAMMING.

- Sei $(M, w\colon M \to \mathbb{N}_0, K \in \mathbb{N}_0)$ beliebige Instanz von SUBSET SUM. Sei $n = |M|$ und o.B.d.A. $M = \{1, \ldots, n\}$.
- **Idee:** Variable $x_j = 1 \leftrightarrow j \in M'$ und Variable $x_j = 0 \leftrightarrow j \notin M'$

  Wähle $B := K$. Dann $\sum_{j=1}^{n} c_j \cdot x_j = B \iff \sum_{j \in M'} w(j) = w(M') = K$
- **Idee:** Bedingung $\sum_{j=1}^{n} a_{ij} \cdot x_j \leq b_i \leftrightarrow$ Element i höchstens 1x gewählt

  Wähle $b_i = 1$. Dann $\sum_{j=1}^{n} a_{ij} \cdot x_j \leq b_i \iff x_i \leq 1$

$\boxed{\exists \bar{x} \in \mathbb{N}_0^n \text{ mit } \bar{c}^T \bar{x} = B,\ A\bar{x} \leq \bar{b}} \iff \boxed{\exists M' \subseteq M \text{ mit } w(M') = K}$

Integer Programming:
$\exists\, x_1, \ldots, x_n \in \mathbb{N}_0$, dass
- ▹ $\sum_{j=1}^{n} c_j \cdot x_j = B$ und
- ▹ $\sum_{j=1}^{n} a_{ij} \cdot x_j \leq b_i$ für $1 \leq i \leq m$?

Bemerkungen

- INTEGER PROGRAMMING $\in \mathcal{NP}$ ist nicht so leicht zu zeigen.
  $\longrightarrow$ Siehe: Papadimitriou “On the complexity of integer programming”, J.ACM, 28, 2, pp. 765-769, 1981.
- Wie der vorherige Beweis zeigt, ist INTEGER PROGRAMMING sogar schon $\mathcal{NP}$-schwer, falls $a_{ij}, b_i \in \{0, 1\}$ und $x_i \in \{0, 1\}$.
- Es kann sogar unter der Zusatzbedingung $c_{ij} \in \{0, 1\}$ $\mathcal{NP}$-Vollständigkeit gezeigt werden (ZERO-ONE PROGRAMMING)

Bemerkungen

- INTEGER PROGRAMMING $\in \mathcal{NP}$ ist nicht so leicht zu zeigen.
  $\longrightarrow$ Siehe: Papadimitriou “On the complexity of integer programming”, J.ACM, 28, 2, pp. 765-769, 1981.
- Wie der vorherige Beweis zeigt, ist INTEGER PROGRAMMING sogar schon $\mathcal{NP}$-schwer, falls $a_{ij}, b_i \in \{0, 1\}$ und $x_i \in \{0, 1\}$.
- Es kann sogar unter der Zusatzbedingung $c_{ij} \in \{0, 1\}$ $\mathcal{NP}$-Vollständigkeit gezeigt werden (ZERO-ONE PROGRAMMING)
- Für beliebige lineare Programme

  Finde $x_1, \ldots, x_n \in \mathbb{R}$

  $$\text{mit} \quad \sum_{i=1}^{n} c_i \cdot x_i = B \quad \text{und} \quad \sum_{j=1}^{n} a_{ij} \cdot x_i \leq b_i \qquad i = 1, \ldots, m$$

  existieren polynomiale Algorithmen.

Kapitel

- **Pseudopolynomiale Algorithmen**

Kapitel

- **Pseudopolynomiale Algorithmen**

SUBSET SUM, PARTITION, KNAPSACK, ... “Komplexität in den Zahlen”	**vs.**	3SAT, HAMILTONKREIS, EXACT COVER, ... “Komplexität in der Struktur”

Kapitel

- **Pseudopolynomiale Algorithmen**

SUBSET SUM, PARTITION, KNAPSACK, ... "Komplexität in den Zahlen"	**vs.**	3SAT, HAMILTONKREIS, EXACT COVER, ... "Komplexität in der Struktur"
$\Uparrow$ pseudopolynomiale Algorithmen		$\Uparrow$ sehr schwierige Probleme

Pseudopolynomiale Algorithmen

- Kodiert man vorkommende Zahlen nicht binär sondern unär, gehen diese nicht logarithmisch, sondern linear in die Inputlänge ein.
- Es gibt $\mathcal{NP}$-vollständige Probleme, die für solche Kodierungen polynomiale Algorithmen besitzen.
- Solche Algorithmen nennt man **pseudopolynomiale Algorithmen**

Definition.

Sei Π ein Optimierungs- oder Entscheidungsproblem. Ein Algorithmus, der Problem Π löst, heißt pseudopolynomial, falls seine Laufzeit durch ein Polynom der beiden Variablen

- Eingabegröße und
- Größe der größten in der Eingabe vorkommenden Zahl

beschränkt ist.

Beispiel: Problem KNAPSACK

Karlsruher Institut für Technologie

Problem KNAPSACK

Gegeben: Eine endliche Menge M,
eine Gewichtsfunktion $w\colon M \longrightarrow \mathbb{N}_0$,
eine Kostenfunktion $c\colon M \longrightarrow \mathbb{N}_0$,
$W, C \in \mathbb{N}_0$

Frage: Existiert eine Teilmenge $M' \subseteq M$ mit $w(M') \leq W$ und $c(M') \geq C$?

Beispiel: Problem KNAPSACK

Problem KNAPSACK

Gegeben: Eine endliche Menge M,
eine Gewichtsfunktion $w\colon M \longrightarrow \mathbb{N}_0$,
eine Kostenfunktion $c\colon M \longrightarrow \mathbb{N}_0$,
$W, C \in \mathbb{N}_0$

Frage: Existiert eine Teilmenge $M' \subseteq M$ mit $w(M') \leq W$ und $c(M') \geq C$?

Notation:

$w(M') = \sum_{a \in M'} w(a)$
$c(M') = \sum_{a \in M'} c(a)$

Satz.

Eine beliebige Instanz (M, w, c, W, C) für KNAPSACK kann in $\mathcal{O}(|M| \cdot W)$ entschieden werden.

Ein Pseudopolynomialer Algorithmus für KNAPSACK

Karlsruher Institut für Technologie

Sei o.B.d.A. $M = \{1, \ldots, n\}$. Für jedes $w \in \mathbb{N}_0$, $w \leq W$ und $i \in M$ definiere

$$c_i^w := \max\{c(M') \mid M' \subseteq \{1, \ldots, i\}, w(M') \leq w\}.$$

- c_{i+1}^w kann für $0 \leq i < n$ leicht berechnet werden als
$$c_{i+1}^w = \max\left\{c_i^w, c(i+1) + c_i^{w-w(i+1)}\right\}.$$
- Die Instanz ist genau dann lösbar, wenn $c_n^W \geq C$.

Ein Pseudopolynomialer Algorithmus für KNAPSACK

Sei o.B.d.A. $M = \{1, \ldots, n\}$. Für jedes $w \in \mathbb{N}_0$, $w \leq W$ und $i \in M$ definiere

$$c_i^w := \max \{c(M') \mid M' \subseteq \{1, \ldots, i\}, w(M') \leq w\} .$$

- c_{i+1}^w kann für $0 \leq i < n$ leicht berechnet werden als
$$c_{i+1}^w = \max \left\{c_i^w, c(i+1) + c_i^{w-w(i+1)}\right\} .$$
- Die Instanz ist genau dann lösbar, wenn $c_n^W \geq C$.

Berechne c_n^W wie folgt:

- Für $w = 1, \ldots, W$
  - $c_0^w := 0$
- Für $i = 1, \ldots, n$
  - Für $w = 1, \ldots, W$ setze $c_i^w := \max \left\{c_{i-1}^w, c(i) + c_{i-1}^{w-w(i)}\right\}$

Laufzeit $= \mathcal{O}(W + n \cdot W) = \mathcal{O}(|M| \cdot W)$.

Starke $\mathcal{NP}$-Vollständigkeit

Karlsruher Institut für Technologie

- Für ein Problem Π und eine Instanz I von Π bezeichne $|I|$ die Länge der Instanz I und $\max(I)$ die größte in I vorkommende Zahl.
- Für ein Problem Π und ein Polynom p sei Π_p das Teilproblem von Π, in dem nur die Instanzen I mit $\max(I) \leq p(|I|)$ vorkommen.
- Ein Entscheidungsproblem Π heißt **stark $\mathcal{NP}$-vollständig**, wenn es ein Polynom p gibt für das Π_p schon $\mathcal{NP}$-vollständig ist.

Satz.

Ist Π stark $\mathcal{NP}$-vollständig und $\mathcal{NP} \neq \mathcal{P}$, dann gibt es keinen pseudopolynomialen Algorithmus für Π.

- Zum Beispiel ist das Problem TSP stark $\mathcal{NP}$-vollständig.

Kapitel

- **Approximationsalgorithmen für Optimierungsprobleme**

Kapitel

- **Approximationsalgorithmen für Optimierungsprobleme**

  Optimierungsproblem Π (bzw. Optimalwertproblem)

  Instanz I $\rightsquigarrow$ optimale Lösungen haben Wert $\mathrm{OPT}(I)$

  Algorithmus $\mathcal{A}$ $\rightsquigarrow$ liefert Lösung mit Wert $\mathcal{A}(I)$

Kapitel

- **Approximationsalgorithmen für Optimierungsprobleme**

Optimierungsproblem Π (bzw. Optimalwertproblem)

Instanz I $\rightsquigarrow$ optimale Lösungen haben Wert $\mathrm{OPT}(I)$

Algorithmus $\mathcal{A}$ $\rightsquigarrow$ liefert Lösung mit Wert $\mathcal{A}(I)$

Absolute Approximation		**Relative Approximation**
“additiver Fehler”	**vs.**	“multiplikativer Fehler”
$\mathcal{A}(I) \leq \mathrm{OPT}(I) + K$		$\mathcal{A}(I) \leq \mathrm{OPT}(I) \cdot K$

(hier für <u>Minimierungsproblem</u> Π)

Absolute Approximationsalgorithmen

Karlsruher Institut für Technologie

Definition

Sei Π ein Optimierungsproblem. Ein polynomialer Algorithmus $\mathcal{A}$, der für jedes $I \in D_\Pi$ einen Wert $\mathcal{A}(I)$ liefert, mit

$$|\,\mathrm{OPT}(I) - \mathcal{A}(I)| \leq K$$

und $K \in \mathbb{N}_0$ konstant, heißt Approximationsalgorithmus mit absoluter Gütegarantie oder absoluter Approximationsalgorithmus.

- Π Minimierungsproblem: $\mathcal{A}(I) \leq \mathrm{OPT}(I) + K$
  Π Maximierungsproblem: $\mathcal{A}(I) \geq \mathrm{OPT}(I) - K$
- Es gibt nur wenige $\mathcal{NP}$-schwere Optimierungsprobleme, für die ein absoluter Approximationsalgorithmus existiert
- Es gibt viele Negativ-Resultate.

Das allgemeine KNAPSACK-Optimierungsproblem

Karlsruher Institut für Technologie

Das allgemeine KNAPSACK-Optimierungsproblem

Gegeben: Menge $M = \{1, \ldots, n\}$,
Kosten $c_1, \ldots, c_n \in \mathbb{N}_0$,
Gewichte $w_1, \ldots, w_n \in \mathbb{N}$,
Gesamtgewicht $W \in \mathbb{N}$

Aufgabe: Gib $x_1, \ldots, x_n \in \mathbb{N}_0$ an, so dass $\sum_{i=0}^{n} w_i \cdot x_i \leq W$
und $\sum_{i=1}^{n} c_i \cdot x_i$ maximal ist.

Das allgemeine KNAPSACK-Optimierungsproblem

Das allgemeine KNAPSACK-Optimierungsproblem

Gegeben: Menge $M = \{1, \dots, n\}$,
Kosten $c_1, \dots, c_n \in \mathbb{N}_0$,
Gewichte $w_1, \dots, w_n \in \mathbb{N}$,
Gesamtgewicht $W \in \mathbb{N}$

Aufgabe: Gib $x_1, \dots, x_n \in \mathbb{N}_0$ an, so dass $\sum_{i=0}^{n} w_i \cdot x_i \leq W$ und $\sum_{i=1}^{n} c_i \cdot x_i$ maximal ist.

- Die Zahl x_i gibt an wie oft Element $i \in M$ genommen wird.
- Wert einer Lösung ist $\sum_{i=1}^{n} c_i \cdot x_i$.
- Dies ist ein Maximierungsproblem.

Das allgemeine KNAPSACK-Optimierungsproblem

Das allgemeine KNAPSACK-Optimierungsproblem

Gegeben: Menge $M = \{1, \ldots, n\}$,
Kosten $c_1, \ldots, c_n \in \mathbb{N}_0$,
Gewichte $w_1, \ldots, w_n \in \mathbb{N}$,
Gesamtgewicht $W \in \mathbb{N}$

Aufgabe: Gib $x_1, \ldots, x_n \in \mathbb{N}_0$ an, so dass $\sum_{i=0}^{n} w_i \cdot x_i \leq W$
und $\sum_{i=1}^{n} c_i \cdot x_i$ maximal ist.

- Die Zahl x_i gibt an wie oft Element $i \in M$ genommen wird.
- Wert einer Lösung ist $\sum_{i=1}^{n} c_i \cdot x_i$.
- Dies ist ein Maximierungsproblem.
- Das allgemeine KNAPSACK-Optimierungsproblem ist $\mathcal{NP}$-schwer.

Das allgemeine KNAPSACK-Optimierungsproblem

Karlsruher Institut für Technologie

Das allgemeine KNAPSACK-Optimierungsproblem

Gegeben: Menge $M = \{1, \ldots, n\}$,
Kosten $c_1, \ldots, c_n \in \mathbb{N}_0$,
Gewichte $w_1, \ldots, w_n \in \mathbb{N}$,
Gesamtgewicht $W \in \mathbb{N}$

Aufgabe: Gib $x_1, \ldots, x_n \in \mathbb{N}_0$ an, so dass $\sum_{i=0}^{n} w_i \cdot x_i \leq W$ und $\sum_{i=1}^{n} c_i \cdot x_i$ maximal ist.

Testen Sie sich:

Passen Sie heutige Betrachtungen für das (spezielle) KNAPSACK an.

- Die Zahl x_i gibt an wie oft Element $i \in M$ genommen wird.
- Wert einer Lösung ist $\sum_{i=1}^{n} c_i \cdot x_i$.
- Dies ist ein Maximierungsproblem.

- Das allgemeine KNAPSACK-Optimierungsproblem ist $\mathcal{NP}$-schwer.

Negativ-Resultat

Satz.

Falls $\mathcal{P} \neq \mathcal{NP}$, so gibt es keinen absoluten Approximationsalgorithmus $\mathcal{A}$ für das allgemeine KNAPSACK-Optimierungsproblem.

Negativ-Resultat

Satz.

Falls $\mathcal{P} \neq \mathcal{NP}$, so gibt es keinen absoluten Approximationsalgorithmus $\mathcal{A}$ für das allgemeine KNAPSACK-Optimierungsproblem.

typischer Beweis mittels Kontraposition

- Angenommen, $\mathcal{A}$ sei absoluter Approximationsalgorithmus für das allgemeine KNAPSACK-Optimierungsproblem.
- Dann entwerfen wir Algorithmus $\tilde{\mathcal{A}}$ der KNAPSACK optimal löst.
- $\tilde{\mathcal{A}}$ ist polynomial, wenn $\mathcal{A}$ polynomial.
- Da das allgemeine KNAPSACK-Optimierungsproblem $\mathcal{NP}$-schwer ist, folgt dann $\mathcal{P} = \mathcal{NP}$.

Negativ-Resultat

Satz.

Falls $\mathcal{P} \neq \mathcal{NP}$, so gibt es keinen absoluten Approximationsalgorithmus $\mathcal{A}$ für das allgemeine KNAPSACK-Optimierungsproblem.

typischer Beweis mittels Kontraposition

- Angenommen, $\mathcal{A}$ sei absoluter Approximationsalgorithmus für das allgemeine KNAPSACK-Optimierungsproblem.
- Dann entwerfen wir Algorithmus $\tilde{\mathcal{A}}$ der KNAPSACK optimal löst.
- $\tilde{\mathcal{A}}$ ist polynomial, wenn $\mathcal{A}$ polynomial.
- Da das allgemeine KNAPSACK-Optimierungsproblem $\mathcal{NP}$-schwer ist, folgt dann $\mathcal{P} = \mathcal{NP}$.

- Dieses Vorgehen kann auch als eine Art Reduktion / Transformation gesehen werden.

(Kontrapositions-)Beweis

- Angenommen, $\mathcal{A}$ sei ein absoluter Approximationsalgorithmus mit $|\operatorname{OPT}(I) - \mathcal{A}(I)| \leq K$ für alle KNAPSACK-Instanzen I.
- Sei $I = (M, w_i, c_i, W)$ eine beliebige KNAPSACK-Instanz.
- Betrachte die modifizierte KNAPSACK-Instanz
$$I' = (M' := M, w_i' := w_i, W' := W, c_i' := c_i \cdot (K+1))$$
- Damit ist $\operatorname{OPT}(I') = (K+1)\operatorname{OPT}(I)$.
- Dann liefert $\mathcal{A}$ zu I' eine Lösung $x_1, \ldots, x_n$ mit Wert $\sum_{i=1}^{n} c_i' \cdot x_i = \mathcal{A}(I')$, für den gilt: $\mathcal{A}(I') \geq \operatorname{OPT}(I') - K$.

(Kontrapositions-)Beweis

- Damit ist $\mathrm{OPT}(I') = (K+1)\,\mathrm{OPT}(I)$.
- Dann liefert $\mathcal{A}$ zu I' eine Lösung $x_1, \ldots, x_n$ mit Wert $\sum_{i=1}^{n} c_i' \cdot x_i = \mathcal{A}(I')$, für den gilt: $\mathcal{A}(I') \geq \mathrm{OPT}(I') - K$.
- $\mathcal{A}(I')$ induziert damit eine Lösung $x_1, \ldots, x_n$ für I mit dem Wert $\tilde{\mathcal{A}}(I) := \sum_{i=1}^{n} c_i \cdot x_i$, für den gilt: $\tilde{\mathcal{A}}(I) = \frac{1}{K+1}\mathcal{A}(I')$.
- Also ist

$$\tilde{\mathcal{A}}(I) = \frac{1}{K+1}\mathcal{A}(I') \geq \frac{1}{K+1}\left(\mathrm{OPT}(I') - K\right) = \mathrm{OPT}(I) - \frac{K}{K+1}.$$

(Kontrapositions-)Beweis

- Damit ist $\quad \mathrm{OPT}(I') = (K+1)\,\mathrm{OPT}(I)$.
- Dann liefert $\mathcal{A}$ zu I' eine Lösung $x_1, \ldots, x_n$ mit Wert $\sum_{i=1}^{n} c_i' \cdot x_i = \mathcal{A}(I')$, für den gilt: $\quad \mathcal{A}(I') \geq \mathrm{OPT}(I') - K$.
- $\mathcal{A}(I')$ induziert damit eine Lösung $x_1, \ldots, x_n$ für I mit dem Wert $\tilde{\mathcal{A}}(I) := \sum_{i=1}^{n} c_i \cdot x_i$, für den gilt: $\quad \tilde{\mathcal{A}}(I) = \frac{1}{K+1}\mathcal{A}(I')$.
- Also ist

$$\tilde{\mathcal{A}}(I) = \frac{1}{K+1}\mathcal{A}(I') \geq \frac{1}{K+1}\left(\mathrm{OPT}(I') - K\right) = \mathrm{OPT}(I) - \frac{K}{K+1}.$$

- Da $\mathrm{OPT}(I)$ und $\tilde{\mathcal{A}}(I) \in \mathbb{N}_0$, und $0 < \frac{K}{K+1} < 1$, ist also

$$\tilde{\mathcal{A}}(I) = \mathrm{OPT}(I).$$

- Algorithmus $\tilde{\mathcal{A}}$ ist polynomial (da $\mathcal{A}$ polynomial ist) und liefert eine optimale Lösung für das KNAPSACK-Optimierungsproblem.

- Da KNAPSACK $\mathcal{NP}$-schwer ist, folgt $\mathcal{P} = \mathcal{NP}$.

Übersicht Approximationsalgorithmen

Optimierungsproblem Π (bzw. Optimalwertproblem)

Instanz I $\rightsquigarrow$ optimale Lösungen haben Wert $\mathrm{OPT}(I)$

Algorithmus $\mathcal{A}$ $\rightsquigarrow$ liefert Lösung mit Wert $\mathcal{A}(I)$

Absolute Approximation		**Relative Approximation**
"additiver Fehler"	**vs.**	"multiplikativer Fehler"
$\mathcal{A}(I) \leq \mathrm{OPT}(I) + K$		$\mathcal{A}(I) \leq \mathrm{OPT}(I) \cdot K$

(hier für Minimierungsproblem Π)