Karlsruher Institut für Technologie

Theoretische Grundlagen der Informatik

Komplexitätsklassen - Teil 2

INSTITUT FÜR THEORETISCHE INFORMATIK

Kapitel

- **Komplementsprachen**

Die Klassen NPI, co-P und co-NP

- Die Klasse $\mathcal{NPC}$ ($\mathcal{NP}$-complete) sei die Klasse der $\mathcal{NP}$-vollständigen Sprachen/Probleme.
- Die Klasse $\mathcal{NPI}$ ($\mathcal{NP}$-intermediate) ist definiert durch $\mathcal{NPI} := \mathcal{NP} \backslash (\mathcal{P} \cup \mathcal{NPC})$.

Klasse der Komplementsprachen

- Die Klasse $\mathbf{co} - \mathcal{P}$ ist die Klasse aller Sprachen $\Sigma^* \backslash L$ für $L \subseteq \Sigma^*$ und $L \in \mathcal{P}$.
- Die Klasse $\mathbf{co} - \mathcal{NP}$ ist die Klasse aller Sprachen $\Sigma^* \backslash L$ für $L \subseteq \Sigma^*$ und $L \in \mathcal{NP}$.

Die Klassen NPI, co-P und co-NP

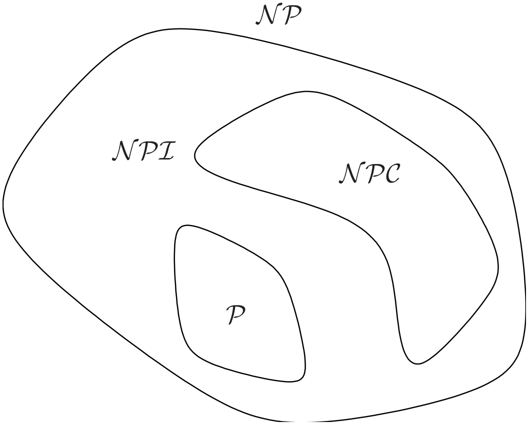
- Die Klasse $\mathcal{NPC}$ ($\mathcal{NP}$-complete) sei die Klasse der $\mathcal{NP}$-vollständigen Sprachen/Probleme.
- Die Klasse $\mathcal{NPI}$ ($\mathcal{NP}$-intermediate) ist definiert durch $\mathcal{NPI} := \mathcal{NP} \backslash (\mathcal{P} \cup \mathcal{NPC})$.

Klasse der Komplementsprachen

- Die Klasse **co** $- \mathcal{P}$ ist die Klasse aller Sprachen $\Sigma^* \backslash L$ für $L \subseteq \Sigma^*$ und $L \in \mathcal{P}$.
- Die Klasse **co** $- \mathcal{NP}$ ist die Klasse aller Sprachen $\Sigma^* \backslash L$ für $L \subseteq \Sigma^*$ und $L \in \mathcal{NP}$.

Satz (Ladner (1975)):
Falls $\mathcal{P} \neq \mathcal{NP}$, so folgt $\mathcal{NPI} \neq \emptyset$.

Vermutete Situation



Offensichtlich: $\mathcal{P} = \text{co} - \mathcal{P}$.

Frage: Gilt auch $\mathcal{NP} = \text{co} - \mathcal{NP}$?

- Natürlich folgt aus $\mathcal{NP} \neq \text{co} - \mathcal{NP}$, dass $\mathcal{P} \neq \mathcal{NP}$ gilt.
- Aber was folgt aus $\mathcal{NP} = \text{co} - \mathcal{NP}$?
- Vermutlich ist $\mathcal{NP} \neq \text{co} - \mathcal{NP}$
  (Verschärfung der $\mathcal{P} \neq \mathcal{NP}$–Vermutung).

Das TSP-Komplement-Problem

Problem co–TSP

Gegeben: Graph $G = (V, E)$, $c\colon E \to \mathbb{Z}^+$ und ein Parameter K.

Aufgabe: Gibt es *keine* Tour der Länge $\leq K$?

- **Bemerkung:** Für ein vernünftiges Kodierungsschema von TSP ist es leicht nachzuweisen, ob ein gegebener String eine gültige TSP-Instanz repräsentiert.
- co–TSP in co $- \mathcal{NP}$, denn TSP in $\mathcal{NP}$.
- Frage: Ist co–TSP in $\mathcal{NP}$?
- Vermutung: Nein.

Lemma

Satz (Lemma):
Falls L $\mathcal{NP}$-vollständig ist und $L \in \text{co} - \mathcal{NP}$, so ist $\mathcal{NP} = \text{co} - \mathcal{NP}$.

Lemma

Satz (Lemma):
Falls L $\mathcal{NP}$-vollständig ist und $L \in \text{co} - \mathcal{NP}$, so ist $\mathcal{NP} = \text{co} - \mathcal{NP}$.

Beweis:

- Sei $L \in \text{co} - \mathcal{NP}$ und L $\mathcal{NP}$-vollständig.
- Dann existiert eine polynomiale nichtdet. Berechnung für L^c.
- Für alle $L' \in \mathcal{NP}$ gilt: $L' \propto L$
- Also existiert eine det. poly. Transformation ${L'}^c \propto L^c$.
- Deshalb existiert eine poly. nichtdet. Berechnung für ${L'}^c$
- Also $L' \in \text{co} - \mathcal{NP}$.

Bemerkung

- Mit der Vermutung $\mathcal{NP} \neq \text{co} - \mathcal{NP}$ folgt auch $\mathcal{NPC} \cap \text{co} - \mathcal{NP} = \emptyset$.
- Unter dieser Annahme: Wenn ein Problem in $\mathcal{NP}$ und $\text{co} - \mathcal{NP}$ ist, aber nicht in $\mathcal{P}$, so ist es in $\mathcal{NPI}$.

Das Problem Subgraphisomorphie

Problem Subgraphisomorphie

Gegeben: Graphen $G = (V, E)$ und $H = (V', E')$ mit $|V'| < |V|$

Frage: Gibt es eine Menge $U \subseteq V$ mit $|U| = |V'|$ und
eine bijektive Abbildung $\mathrm{Iso}\colon V' \to U$,
so dass für alle $x, y \in V'$ gilt:
$\{x, y\} \in E' \iff \{\mathrm{Iso}(x), \mathrm{Iso}(y)\} \in E$

Frage anschaulich: Ist H isomorph zu einem Subgraphen von G?

Das Problem Subgraphisomorphie

Problem Subgraphisomorphie

Gegeben: Graphen $G = (V, E)$ und $H = (V', E')$ mit $|V'| < |V|$

Frage: Gibt es eine Menge $U \subseteq V$ mit $|U| = |V'|$ und eine bijektive Abbildung $\mathrm{Iso}\colon V' \to U$, so dass für alle $x, y \in V'$ gilt: $\{x, y\} \in E' \Longleftrightarrow \{\mathrm{Iso}(x), \mathrm{Iso}(y)\} \in E$

Problem Subgraphisomorphie ist $\mathcal{NP}$-vollständig (ohne Beweis).

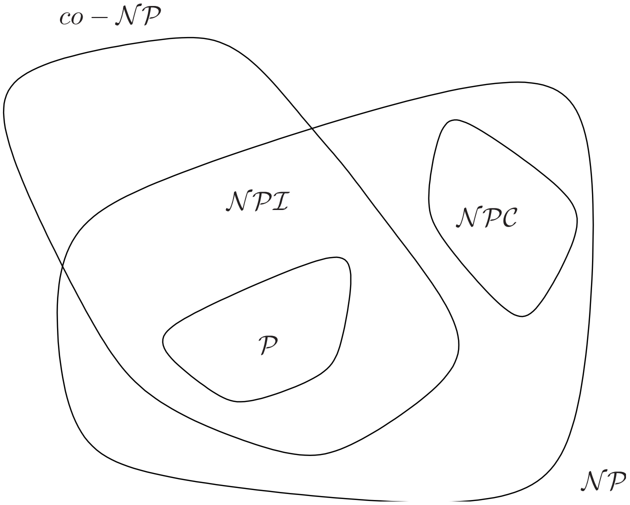
Das Problem Graphisomorphie

Problem Graphisomorphie

Gegeben: Graphen $G = (V, E)$ und $H = (V', E')$ mit $|V| = |V'|$.

Frage: Existiert eine bijektive Abbildung $\mathrm{Iso}\colon V' \to V$ mit
$\{x, y\} \in E' \iff \{\mathrm{Iso}(x), \mathrm{Iso}(y)\} \in E$?

- Frage anschaulich: Sind G und H isomorph?
- Graphisomorphie ist ein Kandidat für ein Problem aus $\mathcal{NPI}$
- Graphisomorphie liegt in $\mathcal{NP}$ und $\mathrm{co} - \mathcal{NP}$.



$co-\mathcal{NP}$
$\mathcal{NPI}$
$\mathcal{NPC}$
$\mathcal{P}$
$\mathcal{NP}$

Kapitel

- **Weitere Komplexitätsklassen über NP hinaus**

Suchprobleme

Ein **Suchproblem** Π wird beschrieben durch

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

Die **Lösung** eines Suchproblems für eine Instanz D_Π 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 $G = (V, E)$ vollständig und gewichtet mit Gewichtsfunktion $c \colon E \to \mathbb{Q}$.

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

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

TSP-Suchproblem (Variante 2)

Gegeben: Graph $G = (V, E)$ vollständig und gewichtet mit Gewichtsfunktion $c \colon E \to \mathbb{Q}$, Parameter $k \in \mathbb{Q}$.

Aufgabe: Gib eine Tour zu G bezüglich c mit Maximallänge k an, falls eine existiert.

Beispiel: Hamilton–Kreis Suchproblem

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

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

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

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.

Aufzählungsprobleme

Ein **Aufzählungsproblem** Π ist gegeben durch

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

Die **Lösung** der Instanz I eines Aufzählungsproblem Π besteht in der 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?

Reduzierbarkeit für Suchprobleme

Zu einem Suchproblem Π sei R_Π folgende Relation:

$$R_\Pi := \{(x, s) \mid x \in D_\Pi, s \in S_\Pi(x)\}$$

Eine Funktion $f\colon \Sigma^* \to \Sigma^*$ **realisiert** eine Relation R, wenn für alle $x \in \Sigma^*$ gilt:

$$f(x) = \begin{cases} \varepsilon & \nexists y \in \Sigma^* \setminus \{\varepsilon\} \;:\; (x, y) \in R \\ y & \text{sonst, mit beliebigem } y : (x, y) \in R \end{cases}$$

Ein Algorithmus **löst** das durch R_Π beschriebene Suchproblem Π, wenn er eine Funktion berechnet, die R_Π realisiert.

Orakel-Turing-Maschine

Eine **Orakel-Turing-Maschine** zum Orakel $G \colon \Sigma^* \to \Sigma^*$ ist eine deterministische Turing-Maschine mit

- einem ausgezeichnetem **Orakelband**
- zwei zusätzlichen Zuständen q_f und q_a.

Dabei ist

- q_f der **Fragezustand**
- q_a der **Antwortzustand**

des Orakels.

- Die Arbeitsweise ist in allen Zuständen $q \neq q_f$ wie bei der normalen Turing-Maschine.

Orakel-TM: Verhalten im Fragezustand

Wenn der

- Zustand q_f angenommen wird,
- Kopf sich auf Position i des Orakelbandes befindet
- Inhalt des Orakelbandes auf Position $1, \ldots, i$ das Wort $y = y_1 \ldots y_i$ ist,

dann verhält sich die Orakel-TM wie folgt:

- falls $y \notin \Sigma^*$: Fehlermeldung und die Orakel-TM hält.
- In einem Schritt wird y auf dem Orakelband gelöscht
- $G(y)$ wird auf Positionen $1, \ldots, |G(y)|$ des Orakelbandes geschrieben
- Der Kopf des Orakelbandes springt auf Position 1
- Folgezustand ist q_a.

Bemerkung

- Orakel-TM und Nichtdeterministische TM sind verschiedene Konzepte.

Turing-Reduktion

Turing-Reduktion
Seien R, R' Relationen über Σ^*. Eine **Turing-Reduktion** $\propto_T$ von R auf R' ($R \propto_T R'$), ist eine Orakel-Turing-Maschine $\mathcal{M}$,

- deren Orakel die Relation R' realisiert
- die selbst in polynomialer Zeit die Funktion f berechnet, die R realisiert.

Bemerkung:

- Falls R' in polynomialer Zeit realisierbar ist und $R \propto_T R'$, so ist auch R in polynomialer Zeit realisierbar.
- Falls $R \propto_T R'$ und $R' \propto_T R''$ so auch $R \propto_T R''$.

NP-schwer

Ein Suchproblem Π heißt $\mathcal{NP}$–schwer, falls es eine $\mathcal{NP}$-vollständige Sprache L gibt mit $L \propto_T \Pi$.

Bemerkung

- Ein Problem das $\mathcal{NP}$–schwer ist, muss nicht notwendigerweise in $\mathcal{NP}$ sein.

Das TSP-Suchproblem ist NP-schwer

Karlsruher Institut für Technologie

TSP-Suchproblem (Variante 1)

Gegeben: Graph $G = (V, E)$ vollständig und gewichtet mit Gewichtsfunktion $c\colon E \to \mathbb{Q}$.

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

TSP-Entscheidungsproblem

Gegeben: Graph $G = (V, E)$ vollständig und gewichtet mit Gewichtsfunktion $c\colon E \to \mathbb{Q}$, Parameter $k \in \mathbb{Q}$.

Aufgabe: Gibt es eine Tour der Länge höchstens k?

Satz:
Das TSP-Suchproblem ist NP-schwer.

Beweisskizze

- Bezeichne TSP_E das Entscheidungsproblem.
- Bezeichne TSP_S das Suchproblem.

Die zu TSP_E bzw, TSP_S gehörenden Relationen R_E und R_S sind gegeben durch

$$\begin{aligned} R_E &:= \{(x, J) \mid x \in J_{TSP_E}\} \\ R_S &:= \{(x, y) \mid x \in D_{TSP_O}, y \in S_{TSP_O}(x)\} . \end{aligned}$$

Beweisskizze

$$R_E := \{(x, J) \mid x \in J_{TSP_E}\}$$
$$R_S := \{(x, y) \mid x \in D_{TSP_O}, y \in S_{TSP_O}(x)\} .$$

Wir zeigen $R_E \propto_T R_S$:

Dazu geben wir eine OTM (Orakel-Turing-Maschine) mit Orakel $\Omega : \Sigma^* \to \Sigma^*$ an. Ω realisiert R_S.

Die OTM arbeitet wie folgt für eine Eingabe w:

- Schreibe die Eingabe auf das Orakelband und gehe in Zustand q_f.
- Weise das Orakel an, in einem Schritt $\Omega(w)$ auf das Orakelband zu schreiben und anschließend in den Zustand q_a zu wechseln.
- Prüfe, ob $\Omega(w)$ eine Tour der Länge $\leq k$ kodiert. Falls ja, lösche das Band und schreibe J, andernfalls lösche das Band.

Die gegebene OTM realisiert R_E und hat polynomial beschränkte Laufzeit.

Verallgemeinerte 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: $a_{ij} \in \mathbb{Z}_0$, $b_i, c_j \in \mathbb{Z}_0$, $1 \leq i \leq m$, $1 \leq j \leq n$, $B \in \mathbb{Z}_0$.

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

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

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

Das Problem INTEGER PROGRAMMING

Problem INTEGER PROGRAMMING

Gegeben: $a_{ij} \in \mathbb{Z}_0$, $b_i, c_j \in \mathbb{Z}_0$, $1 \leq i \leq m$, $1 \leq j \leq n$, $B \in \mathbb{Z}_0$.

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

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

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

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

Beweis

$$\exists\, x_1, \ldots, x_n \in \mathbb{N}_0, \text{ dass } \sum_{j=1}^{n} c_j \cdot x_j = B \text{ und } \underbrace{\sum_{j=1}^{n} a_{ij} \cdot x_j \leq b_i}_{A \cdot \bar{x} \leq \bar{b}} \text{ für } 1 \leq i \leq m?$$

Beweis:
Zeigen: SUBSET SUM $\propto$ INTEGER PROGRAMMING.
Zu M, $w : M \to \mathbb{N}_0$ und $K \in \mathbb{N}_0$ Beispiel für SUBSET SUM wähle $m = n := |M|$, o.B.d.A. $M = \{1, \ldots, n\}$, $c_j := w(j)$, $B := K$, $b_i = 1$ und $A = (a_{ij})$ Einheitsmatrix. Dann gilt:

$$\exists M' \subseteq M \text{ mit } \sum_{j \in M'} w(j) = K$$

$$\Updownarrow$$

$$\exists x_1, \ldots, x_n \in \mathbb{N}_0 \text{ mit } \sum_{j \in M} w(j) \cdot x_j = B \text{ und } x_j \leq 1 \text{ für } 1 \leq j \leq n.$$

$M' = \{j \in M : x_j = 1\}$

Bemerkungen

- INTEGER PROGRAMMING $\in \mathcal{NP}$ ist nicht so leicht zu zeigen. 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 ($a_{ij}, c_j, b_i \in \mathbb{Q};\ x_i \in \mathbb{R}$) existieren polynomiale Algorithmen.

Kapitel

- **Pseudopolynomiale Algorithmen**

Pseudopolynomielle Algorithmen

Karlsruher Institut für Technologie

- 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 **pseudopolynomielle Algorithmen**

Sei Π ein Optimierungsproblem. Ein Algorithmus, der Problem Π löst, heißt pseudopolynomiell, falls seine Laufzeit durch ein Polynom der beiden Variablen

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

beschränkt ist.

Beispiel: Problem KNAPSACK

Problem KNAPSACK

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

Frage: Existiert eine Teilmenge $M^{'} \subseteq M$ mit $\sum_{a \in M'} w(a) \leq W$ und $\sum_{a \in M'} c(a) \geq C$?

Satz:
Ein beliebiges Beispiel (M, w, c, W, C) für KNAPSACK kann in $\mathcal{O}(|M| \cdot W)$ entschieden werden.

Beispiel: Problem KNAPSACK

Satz:
Ein beliebiges Beispiel (M, w, c, W, C) für KNAPSACK kann in $\mathcal{O}(|M| \cdot W)$ entschieden werden.

Beweis:
Sei o.B.d.A. $M = \{1, \dots, n\}$. Für jedes $w \in N_0$, $w \leq W$ und $i \in M$ definiere

$$c_i^w := \max_{M' \subseteq \{1,\dots,i\}} \left\{ \sum_{j \in M'} c(j) : \sum_{j \in M'} w(j) \leq w \right\}.$$

Dann kann c_{i+1}^w für $0 \leq i < n$ leicht berechnet werden als

$$c_{i+1}^w = \max \left\{ c_i^w, c(i+1) + c_i^{w-w(i+1)} \right\}.$$

Starke NP-Vollständigkeit

- 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 Eingaben I mit $\max(I) \leq p(|I|)$ vorkommen.
- Ein Entscheidungsproblem Π heißt **stark $\mathcal{NP}$-vollständig**, wenn Π_p für ein Polynom p $\mathcal{NP}$-vollständig ist.

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

- Problem TSP ist **stark $\mathcal{NP}$–vollständig**.

Kapitel

- **Approximationsalgorithmen für Optimierungsprobleme**

Absolute Approximationsalgorithmen

Absoluter Approximationsalgorithmus
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 Differenzengarantie oder absoluter Approximationsalgorithmus.

- Es gibt nur wenige $\mathcal{NP}$–schwere Optimierungsprobleme, für die ein absoluter Approximationsalgorithmus existiert
- Es gibt viele Negativ–Resultate.

Das allgemeine KNAPSACK-Suchproblem

Das allgemeine KNAPSACK-Suchproblem

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 $\sum_{i=0}^{n} x_i w_i \leq W$ und $\sum_{i=1}^{n} x_i c_i$ maximal ist.

Das allgemeine KNAPSACK-Suchproblem

Das allgemeine KNAPSACK-Suchproblem

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 $\sum_{i=0}^{n} x_i w_i \leq W$ und $\sum_{i=1}^{n} x_i c_i$ maximal ist.

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

Satz

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

(Widerspruchs-)Beweis

Sei $\mathcal{A}$ ein abs. Approximationsalgo mit $|\,\mathrm{OPT}(I) - \mathcal{A}(I)| \leq K$ für alle I.
Sei $I = (M, w_i, c_i, W)$ eine KNAPSACK-Instanz.
Betrachte KNAPSACK-Instanz

$$I' = (M' := M, w_i' := w_i, W' := W, c_i' := c_i \cdot (K+1))$$

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 $\sum_{i=1}^{n} x_i c_i' = \mathcal{A}(I')$, für den gilt:

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

(Widerspruchs-)Beweis

Dann liefert $\mathcal{A}$ zu I' eine Lösung $x_1, \ldots, x_n$ mit Wert $\sum_{i=1}^{n} x_i c_i' = \mathcal{A}(I')$, für den gilt:

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

$\mathcal{A}(I')$ induziert damit eine Lösung $x_1, \ldots, x_n$ für I mit dem Wert

$$\mathcal{L}(I) := \sum_{i=1}^{n} x_i c_i,$$

für den gilt:

$$|(K+1)\,\mathrm{OPT}(I) - (K+1)\mathcal{L}(I)| \leq K$$

Also ist

$$|\,\mathrm{OPT}(I) - \mathcal{L}(I)| \leq \frac{K}{K+1} < 1\,.$$

(Widerspruchs-)Beweis

Also ist

$$|\,\mathrm{OPT}(I) - \mathcal{L}(I)| \leq \frac{K}{K+1} < 1\ .$$

Da

$$\mathrm{OPT}(I) \text{ und } \mathcal{L}(I) \in \mathbb{N}_0 \text{ für alle } I,$$

ist also

$$\mathrm{OPT}(I) = L(I)\ .$$

Der entsprechende Algorithmus ist natürlich polynomial und liefert einen Optimalwert für das KNAPSACK–Problem. Dies steht im Widerspruch zur Annahme, dass $\mathcal{P} \neq \mathcal{NP}$.

Approximation mit relativer Gütegarantie

Sei Π ein Optimierungsproblem. Ein polynomialer Algorithmus $\mathcal{A}$, der für jedes $I \in D_\Pi$ einen Wert $\mathcal{A}(I)$ liefert mit $R_\mathcal{A}(I) \leq K$, wobei $K \geq 1$ eine Konstante, und

$$\mathcal{R}_\mathcal{A}(I) := \begin{cases} \frac{\mathcal{A}(I)}{\text{OPT}(I)} & \text{falls } \Pi \text{ Minimierungsproblem} \\ \frac{\text{OPT}(I)}{\mathcal{A}(I)} & \text{falls } \Pi \text{ Maximierungsproblem} \end{cases}$$

heißt Approximationsalgorithmus mit relativer Gütegarantie. $\mathcal{A}$ heißt ε–approximativ, falls $\mathcal{R}_\mathcal{A}(I) \leq 1 + \varepsilon$ für alle $I \in D_\Pi$.

Beispiel: Greedy-Algorithmus für KNAPSACK

Idee: Es werden der Reihe nach so viele Elemente wie möglich mit absteigender Gewichtsdichte in die Lösung aufgenommen.

- Berechne die Gewichtsdichten $p_i := \frac{c_i}{w_i}$ für $i = 1, \dots, n$
- Sortiere nach Gewichtsdichtenindiziere: $p_1 \geq p_2 \geq \dots \geq p_n$
- Dies kann in Zeit $\mathcal{O}(n \log n)$ geschehen.
- Für $i = 1$ bis n setze $x_i := \left\lfloor \frac{W}{w_i} \right\rfloor$ und $W := W - \left\lfloor \frac{W}{w_i} \right\rfloor \cdot w_i$.

Die Laufzeit dieses Algorithmus ist in $\mathcal{O}(n \log n)$.

Beispiel: Greedy-Algorithmus für KNAPSACK

- Berechne die Gewichtsdichten $p_i := \frac{c_i}{w_i}$ für $i = 1, \dots, n$
- Sortiere nach Gewichtsdichtenindiziere: $p_1 \geq p_2 \geq \dots \geq p_n$
- Dies kann in Zeit $\mathcal{O}(n \log n)$ geschehen.
- Für $i = 1$ bis n setze $x_i := \left\lfloor \frac{W}{w_i} \right\rfloor$ und $W := W - \left\lfloor \frac{W}{w_i} \right\rfloor \cdot w_i$.

Satz:
Der Greedy–Algorithmus $\mathcal{A}$ für KNAPSACK erfüllt $\mathcal{R}_{\mathcal{A}}(I) \leq 2$ für alle Instanzen I.

Beispiel: Greedy-Algorithmus für KNAPSACK

- Berechne die Gewichtsdichten $p_i := \frac{c_i}{w_i}$ für $i = 1, \dots, n$
- Sortiere nach Gewichtsdichtenindiziere: $p_1 \geq p_2 \geq \dots \geq p_n$
- Dies kann in Zeit $\mathcal{O}(n \log n)$ geschehen.
- Für $i = 1$ bis n setze $x_i := \left\lfloor \frac{W}{w_i} \right\rfloor$ und $W := W - \left\lfloor \frac{W}{w_i} \right\rfloor \cdot w_i$.

Beweis:
O.B.d.A. sei $w_1 \leq W$. Offensichtlich gilt:

$$\mathcal{A}(I) \geq c_1 \cdot x_1 = c_1 \cdot \left\lfloor \frac{W}{w_1} \right\rfloor \text{ für alle } I$$

und

$$\mathrm{OPT}(I) \leq c_1 \cdot \frac{W}{w_1} \leq c_1 \cdot \left(\left\lfloor \frac{W}{w_1} \right\rfloor + 1 \right) \leq 2 \cdot c_1 \cdot \left\lfloor \frac{W}{w_1} \right\rfloor \leq 2 \cdot \mathcal{A}(I) \; .$$

Also $\mathcal{R}_{\mathcal{A}}(I) \leq 2$.

Beispiel: Greedy-Algorithmus für KNAPSACK

- Berechne die Gewichtsdichten $p_i := \frac{c_i}{w_i}$ für $i = 1, \ldots, n$
- Sortiere nach Gewichtsdichtenindiziere: $p_1 \geq p_2 \geq \ldots \geq p_n$
- Dies kann in Zeit $\mathcal{O}(n \log n)$ geschehen.
- Für $i = 1$ bis n setze $x_i := \left\lfloor \frac{W}{w_i} \right\rfloor$ und $W := W - \left\lfloor \frac{W}{w_i} \right\rfloor \cdot w_i$.

Bemerkung: Die Schranke $\mathcal{R}_{\mathcal{A}}(I)$ ist in gewissem Sinne scharf.

Sei $n = 2,\ w_2 = w_1 - 1,\ c_1 = 2 \cdot w_1,\ c_2 = 2 \cdot w_2 - 1,\ W = 2 \cdot w_2$.
Dann ist

$$\frac{c_1}{w_1} = 2 > \frac{c_2}{w_2} = 2 - \frac{1}{w_2}$$

und $\mathcal{A}(I) = 2w_1$ und $\mathrm{OPT}(I) = 4w_2 - 2$, also

$$\frac{\mathrm{OPT}(I)}{\mathcal{A}(I)} = \frac{4w_2 - 2}{2w_1} = \frac{2w_1 - 3}{w_1} \longrightarrow 2 \qquad \text{für } w_1 \to \infty$$

Definition

Zu einem polynomialen Approximationsalgorithmus $\mathcal{A}$ sei

$$\mathcal{R}_{\mathcal{A}}^{\infty} := \inf \left\{ r \geq 1 \;\middle|\; \begin{array}{l} \text{es gibt ein } N_0 > 0 \text{, so dass } \mathcal{R}_{\mathcal{A}}(I) \leq r \\ \text{für alle } I \text{ mit } \mathrm{OPT}(I) \geq N_0 \end{array} \right\}$$

Approximierbarkeit von COLOR

$$\mathcal{R}_{\mathcal{A}}^{\infty} := \inf \left\{ r \geq 1 \;\middle|\; \begin{array}{l} \text{es gibt ein } N_0 > 0 \text{, so dass } \mathcal{R}_{\mathcal{A}}(I) \leq r \\ \text{für alle } I \text{ mit } \mathrm{OPT}(I) \geq N_0 \end{array} \right\}$$

Problem COLOR (Optimalwertfassung)

Gegeben: Graph $G = (V, E)$

Frage: Wieviele Farben benötigt man um V zu färben, so dass je zwei adjazente Knoten verschiedene Farben besitzen?

Satz:
Falls $\mathcal{P} \neq \mathcal{NP}$, dann existiert kein relativer Approximationsalgorithmus $\mathcal{A}$ für COLOR mit $\mathcal{R}_{\mathcal{A}}^{\infty} < \frac{4}{3}$.

Approximierbarkeit von COLOR

$$\mathcal{R}_{\mathcal{A}}^{\infty} := \inf \left\{ r \geq 1 \;\middle|\; \begin{array}{l} \text{es gibt ein } N_0 > 0 \text{, so dass } \mathcal{R}_{\mathcal{A}}(I) \leq r \\ \text{für alle } I \text{ mit } \mathrm{OPT}(I) \geq N_0 \end{array} \right\}$$

Satz:
Falls $\mathcal{P} \neq \mathcal{NP}$, dann existiert kein relativer Approximationsalgorithmus $\mathcal{A}$ für COLOR mit $\mathcal{R}_{\mathcal{A}}^{\infty} < \frac{4}{3}$.

Beweis:

- Angenommen es gibt einen relativen Approximationsalgorithmus $\mathcal{A}$ für COLOR mit $\mathcal{R}_{\mathcal{A}}^{\infty} < \frac{4}{3}$.
- Wir benutzen $\mathcal{A}$ um 3COLOR zu lösen.
- Dies ist ein Widerspruch zu $\mathcal{P} \neq \mathcal{NP}$

Approximierbarkeit von COLOR

Zu zwei Graphen

$$G_1 = (V_1, E_1) \text{ und } G_2 = (V_2, E_2)$$

sei

$$G := (V, E) := G_1[G_2]$$

definiert durch

$$V := V_1 \times V_2$$

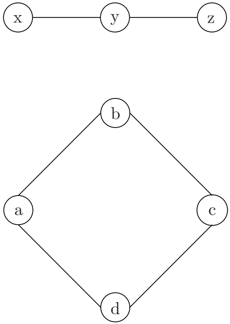
und

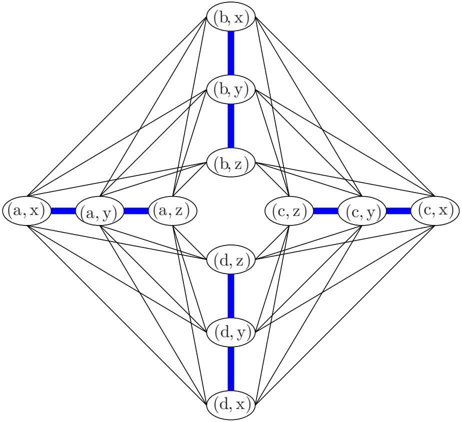
$$E := \left\{ \{(u_1, u_2), (v_1, v_2)\} \;\middle|\; \begin{array}{l} \text{entweder } \{u_1, v_1\} \in E_1 \text{, oder} \\ u_1 = v_1 \text{ und } \{u_2, v_2\} \in E_2 \end{array} \right\}$$

Anschaulich

- Jeder Knoten aus G_1 wird durch eine Kopie von G_2 ersetzt
- Jede Kante aus E_1 durch einen vollständig bipartiten Graphen zwischen den entsprechenden Kopien.

Approximierbarkeit von COLOR





Approximierbarkeit von COLOR

$$\mathcal{R}_{\mathcal{A}}^{\infty} := \inf \left\{ r \geq 1 \;\middle|\; \begin{array}{l} \text{es gibt ein } N_0 > 0 \text{, so dass } \mathcal{R}_{\mathcal{A}}(I) \leq r \\ \text{für alle } I \text{ mit } \mathrm{OPT}(I) \geq N_0 \end{array} \right\}$$

- Angenommen es gibt einen relativen Approximationsalgorithmus $\mathcal{A}$ für COLOR mit $\mathcal{R}_{\mathcal{A}}^{\infty} < \frac{4}{3}$.
- Dann existiert ein $N \in \mathbb{N}$ so, dass $\mathcal{A}(G) < \frac{4}{3}\,\mathrm{OPT}(G)$ für alle Graphen G mit $\mathrm{OPT}(G) \geq N$.

Approximierbarkeit von COLOR

- Dann existiert ein $N \in \mathbb{N}$ so, dass $\mathcal{A}(G) < \frac{4}{3}\,\mathrm{OPT}(G)$ für alle Graphen G mit $\mathrm{OPT}(G) \geq N$.
- Sei also $G = (V, E)$ ein beliebiges Beispiel für 3COLOR.
- Dann definiere $G^* := K_N[G]$, wobei K_N der vollständige Graph über N Knoten ist.
- Dann gilt: $\mathrm{OPT}(G^*) = N \cdot \mathrm{OPT}(G) \geq N$.

Fallunterscheidung:

- Falls G dreifärbbar ist, gilt:

$$\mathcal{A}(G^*) < \frac{4}{3}\,\mathrm{OPT}(G^*) = \frac{4}{3} \cdot N \cdot \mathrm{OPT}(G) \leq \frac{4}{3} \cdot N \cdot 3 = 4N.$$

- Andererseits, falls G nicht dreifärbbar ist, gilt

$$\mathcal{A}(G^*) \geq \mathrm{OPT}(G^*) = N \cdot \mathrm{OPT}(G) \geq 4N.$$

Fazit: G ist dreifärbbar genau dann, wenn $\mathcal{A}(G^*) < 4N$.

Approximierbarkeit von COLOR

- Die Größe von G^* ist polynomial in der Größe von G.
- Also kann G^* in polynomialer Zeit konstruiert werden.
- Damit ist die Anwendung von $\mathcal{A}$ auf G^* polynomial in der Größe von G.
- Also haben wir einen polynomialen Algorithmus zur Lösung von 3COLOR konstruiert.
- Dies ist ein Widerspruch zur Annahme, dass $\mathcal{P} \neq \mathcal{NP}$.

Approximierbarkeit von TSP

Karlsruher Institut für Technologie

TSP-Optimalwertproblem mit Dreiecksungleichung

Gegeben: Graph $G = (V, E)$ vollständig und gewichtet mit Gewichtsfunktion $c \colon E \to \mathbb{Q}$.
Es gilt $c(u, w) \leq c(u, v) + c(v, w)$ für alle $u, v, w \in V$

Frage: Wie lange ist eine optimale Tour zu G bezüglich c?

Satz:
Für das TSP-Optimalwertproblem mit Dreiecksungleichung existiert ein Approximationsalgorithmus $\mathcal{A}$ mit $\mathcal{R}_{\mathcal{A}} \leq 2$ für alle Instanzen I.

Approximierbarkeit von TSP

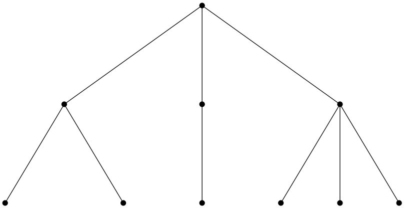
Beweis.

- Sei $(G = (V, E), c)$ eine Instanz des TSP-Optimalwertproblems mit Dreiecksungleichung.

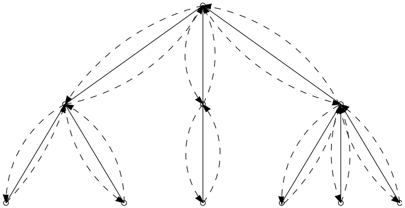
Betrachte folgenden Algorithmus:

- Berechne einen *MST* (Minimum Spanning Tree) von G.
- Wähle einen beliebigen Knoten w als Wurzel.
- Durchlaufe den *MST* in einer Tiefensuche mit Startpunkt w
- **Ergebnis:** Tour T mit Start- und Endpunkt w, die jede Kante zweimal durchläuft.
- Konstruiere aus T eine Tour $T^{'}$, indem bereits besuchte Knoten übersprungen werden und die Tour beim nächsten unbesuchten Knoten fortgesetzt wird.

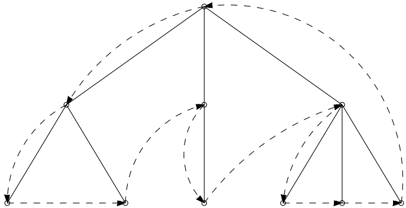
Approximierbarkeit von TSP



(a) MST eines Graphen



(b) Tiefensuch-Tour durch den MST



(c) TSP-Tour als abgekürzte Tiefensuch-Tour

Approximierbarkeit von TSP

- Bezeichne $c(G')$ die Summe der Kantengewichte in Subgraph G'

Es gilt

$$c(T') \leq c(T) = 2 \cdot c(MST) \; .$$

Eine TSP-Tour kann als ein aufspannender Baum plus eine zusätzliche Kante betrachtet werden. Also gilt

$$c(MST) \leq c(\mathrm{OPT}).$$

Insgesamt erhält man

$$c(T') \leq c(T) = 2 \cdot c(MST) \leq 2 \cdot c(\mathrm{OPT}),$$

also

$$\mathcal{R}_{\mathcal{A}} = \frac{c(T')}{c(\mathrm{OPT})} \leq 2 \; .$$

Approximationsschemata

Ein (polynomiales) **Approximationsschema** (**PAS**) für ein Optimierungsproblem Π ist eine Familie von Algorithmen $\{\mathcal{A}_\varepsilon \mid \varepsilon > 0\}$, so dass für alle $\varepsilon > 0$

- $\mathcal{R}_{\mathcal{A}_\varepsilon} \leq 1 + \varepsilon$ ist (d.h. $\mathcal{A}_\varepsilon$ ist ein ε–approximierender Algorithmus).
- $\mathcal{A}_\varepsilon$ polynomial in der Größe des Inputs ist.

Ein Approximationsschema $\{\mathcal{A}_\varepsilon \mid \varepsilon > 0\}$ heißt **vollpolynomial** (**FPAS**) falls seine Laufzeit zudem polynomial in $\frac{1}{\varepsilon}$ ist.

Approximationsschemata

Bezeichne $\langle I \rangle$ die Kodierungslänge der Eingabe-Instanz I.

Satz:
Sei Π ein $\mathcal{NP}$–schweres Optimierungsproblem mit

- $\text{OPT}(I) \in \mathbb{N}$ für alle $I \in D_\Pi$, und
- es existiert ein Polynom q mit $\text{OPT}(I) < q(\langle I \rangle)$ für alle $I \in D_\Pi$.

Falls $\mathcal{P} \neq \mathcal{NP}$, so gibt es kein FPAS $\{\mathcal{A}_\varepsilon \mid \varepsilon > 0\}$ für Π.

Approximationsschemata

Satz:
Sei Π ein $\mathcal{NP}$–schweres Optimierungsproblem mit

- $\mathrm{OPT}(I) \in \mathbb{N}$ für alle $I \in D_\Pi$, und
- es existiert ein Polynom q mit $\mathrm{OPT}(I) < q(\langle I \rangle)$ für alle $I \in D_\Pi$.

Falls $\mathcal{P} \neq \mathcal{NP}$, so gibt es kein FPAS $\{\mathcal{A}_\varepsilon \mid \varepsilon > 0\}$ für Π.

Beweis:

- O.B.d.A. sei Π ein Maximierungsproblem.
- Sei $\{\mathcal{A}_\varepsilon \mid \varepsilon > 0\}$ ein FPAS für Π
- Zu $I \in D_\Pi$ sei $\varepsilon_0 := \frac{1}{q(\langle I \rangle)}$
- Dann ist $\mathcal{A}_{\varepsilon_0}$ polynomial in $\langle I \rangle$ und in $\frac{1}{\varepsilon_0} = q(\langle I \rangle)$

Approximationsschemata

Satz:
Sei Π ein $\mathcal{NP}$–schweres Optimierungsproblem mit

- $\text{OPT}(I) \in \mathbb{N}$ für alle $I \in D_\Pi$, und
- es existiert ein Polynom q mit $\text{OPT}(I) < q(\langle I \rangle)$ für alle $I \in D_\Pi$.

Falls $\mathcal{P} \neq \mathcal{NP}$, so gibt es kein FPAS $\{\mathcal{A}_\varepsilon \mid \varepsilon > 0\}$ für Π.

Es gilt:

$$\begin{aligned} OPT(I) &\leq (1+\varepsilon_0)\mathcal{A}_{\varepsilon_0}(I) \text{ und} \\ OPT(I) &< q(\langle I \rangle) = \frac{1}{\varepsilon_0} \end{aligned}$$

Also auch

$$\text{OPT}(I) - \mathcal{A}_{\varepsilon_0}(I) \leq \varepsilon_0 \cdot \mathcal{A}_{\varepsilon_0}(I) \leq \varepsilon_0 \cdot \text{OPT}(I) < 1$$

- Da $\text{OPT}(I) \in \mathbb{N}$, ist $\text{OPT}(I) = \mathcal{A}_{\varepsilon_0}(I)$
- Widerspruch zur Annahme, dass $\mathcal{P} \neq \mathcal{NP}$.

Ein FPAS für KNAPSACK

Problem KNAPSACK

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

Aufgabe: Gib eine Teilmenge M' von M an, so dass
$\sum_{i \in M'} w_i \leq W$ und $\sum_{i \in M'} c_i$ maximal ist.

Ein pseudopolynomialer, optimaler Algorithmus für KNAPSACK

Bezeichne, für $r \in \mathbb{N}_0$ $\quad w_r^j := \min\limits_{M' \subseteq \{1,\dots,j\}} \left\{ \sum\limits_{i \in M'} w_i \;\middle|\; \sum\limits_{i \in M'} c_i = r \right\}$

- **Initialisierung**
  Für $1 \leq j \leq n$ setze $w_0^j := 0$ ansonsten setze $c := \sum_{i=1}^n c_i$
- **Berechung**
  Solange $w_r^j \leq W$ berechne für $2 \leq j \leq n$ und $1 \leq r \leq c$ den Wert

$$w_r^j = \min \left\{ w_{r-c_j}^{j-1} + w^j, w_r^{j-1} \right. .$$

- **Ausgabe**

$$c^* := \max_{1 \leq i \leq n} \left\{ r \mid w_r^j \leq W \right\}$$

und die entsprechende Menge $M' \subseteq M$ mit $c^* = \sum_{i \in M'} c_i$.

Ein pseudopolynomialer, optimaler Algorithmus für KNAPSACK

Bezeichne, für $r \in \mathbb{N}_0$ $\quad w_r^j := \min_{M' \subseteq \{1,\dots,j\}} \left\{ \sum_{i \in M'} w_i \;\middle|\; \sum_{i \in M'} c_i = r \right\}$

- **Initialisierung**
  Für $1 \leq j \leq n$ setze $w_0^j := 0$ ansonsten setze $c := \sum_{i=1}^n c_i$
- **Berechung**
  Solange $w_r^j \leq W$ berechne für $2 \leq j \leq n$ und $1 \leq r \leq c$ den Wert

$$w_r^j = \min \left\{ w_{r-c_j}^{j-1} + w^j, w_r^{j-1} \right. .$$

- **Ausgabe**

$$c^* := \max_{1 \leq i \leq n} \left\{ r \mid w_r^j \leq W \right\}$$

  und die entsprechende Menge $M' \subseteq M$ mit $c^* = \sum_{i \in M'} c_i$.

Laufzeit: in $\mathcal{O}(n \cdot c)$. **Lösung:** optimal.
$\Rightarrow$Optimaler pseudopolynomialer Algorithmus.

Ein FPAS für KNAPSACK

- Bezeichne $\mathcal{A}$ obigen pseudopolynomialen Algorithmus mit Laufzeit $\mathcal{O}(n \cdot c)$ für KNAPSACK.
- Sei k beliebig aber fest.
- **Betrachte das skalierte Problem Π_k zu mit $c_i' := \left\lfloor \frac{c_i}{k} \right\rfloor$ für alle $i \in M$.**

- Dann liefert $\mathcal{A}$ für jedes $I_k \in \Pi_k$ eine Menge $M' \subseteq M$ mit $\sum_{i \in M'} c_i' = \mathrm{OPT}(I_k)$.
- Setze nun $c_{\max} := \max_{i \in M} c_i$.
- **Zu $\varepsilon > 0$ sei $\mathcal{A}_\varepsilon$ Algorithmus $\mathcal{A}$ angewendet auf I_k, wobei**

$$k := \frac{c_{\max}}{\left(\frac{1}{\varepsilon} + 1\right) \cdot n}$$

Ein FPAS für KNAPSACK

Satz:
$\mathcal{R}_{\mathcal{A}_\varepsilon}(I) \leq 1 + \varepsilon$ für alle $I \in D_\Pi$ und die Laufzeit von $\mathcal{A}_\varepsilon$ ist in $\mathcal{O}(n^3 \cdot \frac{1}{\varepsilon})$ für alle $\varepsilon > 0$, d.h. $\{\mathcal{A}_\varepsilon \mid \varepsilon > 0\}$ ist ein FPAS für KNAPSACK.

Ein FPAS für KNAPSACK

Satz:
$\mathcal{R}_{\mathcal{A}_\varepsilon}(I) \leq 1 + \varepsilon$ für alle $I \in D_\Pi$ und die Laufzeit von $\mathcal{A}_\varepsilon$ ist in $\mathcal{O}(n^3 \cdot \frac{1}{\varepsilon})$ für alle $\varepsilon > 0$, d.h. $\{\mathcal{A}_\varepsilon \mid \varepsilon > 0\}$ ist ein FPAS für KNAPSACK.

Beweis:

Die Laufzeit von $\mathcal{A}_\varepsilon$ ist in $\mathcal{O}(n \cdot \sum_{i=1}^{n} c_i')$ und

$$\sum_{i=1}^{n} c_i' < \sum_{i=1}^{n} \frac{c_i}{k} \leq n \cdot \frac{c_{\max}}{k} = \left(\frac{1}{\varepsilon} + 1\right) n^2.$$

Also ist die Laufzeit von $\mathcal{A}_\varepsilon$ in $\mathcal{O}(n^3 \cdot \frac{1}{\varepsilon})$ für alle $\varepsilon > 0$.

Ein FPAS für KNAPSACK

Für die Abschätzung von $\mathcal{R}_{\mathcal{A}_\varepsilon}$ betrachte M' mit $\text{OPT}(I) = \sum_{i \in M'} c_i$. Es gilt

$$\text{OPT}(I_k) \geq \sum_{i \in M'} \left\lfloor \frac{c_i}{k} \right\rfloor \geq \sum_{i \in M'} \left(\frac{c_i}{k} - 1 \right).$$

Also ist

$$\text{OPT}(I) - k \cdot \text{OPT}(I_k) \leq k \cdot n.$$

Da $\frac{1}{k}\mathcal{A}_\varepsilon(I) \geq \text{OPT}(I_k)$ ist, folgt

$$\text{OPT}(I) - \mathcal{A}_\varepsilon(I) \leq k \cdot n$$

und wegen $\text{OPT}(I) \geq c_{\max}$ (wir setzen wieder o.B.d.A. $W \geq w_i$ für alle $i \in M$ voraus) folgt

$$\begin{aligned}
\mathcal{R}_{\mathcal{A}_\varepsilon}(I) &= \frac{\text{OPT}(I)}{\mathcal{A}_\varepsilon(I)} \leq \frac{\mathcal{A}_\varepsilon(I) + kn}{\mathcal{A}_\varepsilon(I)} &&= 1 + \frac{kn}{\mathcal{A}_\varepsilon(I)} \leq 1 + \frac{kn}{\text{OPT}(I) - kn} \\
&\leq 1 + \frac{kn}{c_{\max} - kn} &&= 1 + \frac{1}{\frac{1}{\varepsilon} + 1 - 1} = 1 + \varepsilon\,.
\end{aligned}$$

Ein allgemeineres Resultat

Mit einem ähnlichen Beweis kann man zeigen:

Satz:
Sei Π ein Optimierungsproblem für das gilt:

- $\mathrm{OPT}(I) \in \mathbb{N}$ für alle $I \in D_\Pi$
- es existiert ein Polynom q mit $\mathrm{OPT}(I) \leq q(\langle I \rangle + \max \#(I))$
  ($\max \#(I)$ ist die größte in I vorkommende Zahl)

Falls Π ein FPAS hat, so hat es einen pseudopolynomialen optimalen Algorithmus.