Karlsruher Institut für Technologie

Theoretische Grundlagen der Informatik
Vorlesung am 11. Dezember 2018

INSTITUT FÜR THEORETISCHE INFORMATIK

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

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

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

Das Problem INTEGER PROGRAMMING

Problem INTEGER PROGRAMMING

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

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

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

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

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

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

$\exists\, x_1, \ldots, x_n \in \mathbb{N}_0$, 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 $1 \leq i \leq m$?

Beweis:
Wir zeigen die 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. $M = \{1, \ldots, n\}$.

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

$\exists\ x_1, \ldots, x_n \in \mathbb{N}_0$, 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 $1 \leq i \leq m$?

Beweis:
Wir zeigen die 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 $\sum_{j=1}^{n} c_j \cdot x_j = B \iff \sum_{j \in M'} w(j) = w(M') = K$

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

$\exists\, x_1, \dots, x_n \in \mathbb{N}_0$, 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 $1 \leq i \leq m$?

Beweis:
Wir zeigen die 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. $M = \{1, \dots, n\}$.
- **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 $\sum\limits_{j=1}^{n} a_{ij} \cdot x_j = x_i$.

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

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

$\exists\ x_1, \ldots, x_n \in \mathbb{N}_0$, 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 $1 \leq i \leq m$?

Beweis:

Wir zeigen die 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.
- **Idee:** Variable $x_j = 1 \leftrightarrow j \in M'$ und Variable $x_j = 0 \leftrightarrow j \notin M'$

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

- **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 $\sum\limits_{j=1}^{n} a_{ij} \cdot x_j \leq b_i \iff x_i \leq 1$

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

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).

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

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

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

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**

Sei Π ein Optimierungsproblem. 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

Problem KNAPSACK

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

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

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

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

Beispiel: Problem KNAPSACK

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

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

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

- 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 $c_n^W \geq C$.

Beispiel: Problem KNAPSACK

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

Beweis:

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

- 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**

Optimierungsprobleme Π (bzw. Optimalwertproblem)

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

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

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

(hier für Minimierungsproblem Π)

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 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-Suchproblem

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-Suchproblem

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.

- Lösung ist gegeben durch $x_1, \ldots, x_n \in \mathbb{N}_0$ mit $\sum_{i=0}^{n} w_i \cdot x_i \leq W$.
- 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-Suchproblem

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.

- Lösung ist gegeben durch $x_1, \ldots, x_n \in \mathbb{N}_0$ mit $\sum_{i=0}^{n} w_i \cdot x_i \leq W$.
- 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-Suchproblem

Karlsruher Institut für Technologie

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.

- Lösung ist gegeben durch $x_1, \dots, x_n \in \mathbb{N}_0$ mit $\sum_{i=0}^{n} w_i \cdot x_i \leq W$.
- 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.

Testen Sie sich:

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

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 $\mathrm{OPT}(I') = (K+1)\,\mathrm{OPT}(I)$.
- 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

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

- 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 $\mathcal{P} = \mathcal{NP}$.

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 Minimierungsproblem Π)

Approximation mit relativer Gütegarantie

Karlsruher Institut für Technologie

Relativer Approximationsalgorithmus
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

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

heißt Approximationsalgorithmus mit relativer Gütegarantie oder relativer Approximationsalgorithmus.

- $\mathcal{A}$ heißt ε-approximativ, falls $\mathcal{R}_\mathcal{A}(I) \leq 1 + \varepsilon$ für alle $I \in D_\Pi$.
- Einige, aber nicht alle $\mathcal{NP}$-schweren Optimierungsprobleme erlauben einen relativen Approximationsalgorithmus.

Genereller Ansatz

Bei Minimierungsproblemen

- Wir wollen $\mathcal{R}_{\mathcal{A}}(I) = \frac{\mathcal{A}(I)}{\mathrm{OPT}(I)} \leq K$, also $\mathcal{A}(I) \leq K \cdot \mathrm{OPT}(I)$.
- Wir brauchen:
  - eine **obere** Schranke für $\mathcal{A}(I)$ — "$\mathcal{A}$ ist gut"
  - eine **untere** Schranke für $\mathrm{OPT}(I)$ — "viel besser geht es nicht"

$$\mathcal{A}(I) \leq X \text{ und } \mathrm{OPT}(I) \geq Y \quad \Longrightarrow \quad \mathcal{A}(I) \leq X = \frac{X \cdot Y}{Y} \leq \frac{X}{Y} \cdot \mathrm{OPT}(I)$$

Genereller Ansatz

Bei Minimierungsproblemen

- Wir wollen $\mathcal{R}_\mathcal{A}(I) = \frac{\mathcal{A}(I)}{\mathrm{OPT}(I)} \leq K$, also $\mathcal{A}(I) \leq K \cdot \mathrm{OPT}(I)$.
- Wir brauchen:
  - eine **obere** Schranke für $\mathcal{A}(I)$ — "$\mathcal{A}$ ist gut"
  - eine **untere** Schranke für $\mathrm{OPT}(I)$ — "viel besser geht es nicht"

$$\mathcal{A}(I) \leq X \text{ und } \mathrm{OPT}(I) \geq Y \quad \Longrightarrow \quad \mathcal{A}(I) \leq X = \frac{X \cdot Y}{Y} \leq \frac{X}{Y} \cdot \mathrm{OPT}(I)$$

Bei Maximierungsproblemen

- Wir wollen $\mathcal{R}_\mathcal{A}(I) = \frac{\mathrm{OPT}(I)}{\mathcal{A}(I)} \leq K$, also $\mathcal{A}(I) \geq \frac{1}{K} \cdot \mathrm{OPT}(I)$.
- Wir brauchen:
  - eine **untere** Schranke für $\mathcal{A}(I)$ — "$\mathcal{A}$ ist gut"
  - eine **obere** Schranke für $\mathrm{OPT}(I)$ — "viel besser geht es nicht"

$$\mathcal{A}(I) \geq X \text{ und } \mathrm{OPT}(I) \leq Y \quad \Longrightarrow \quad \mathcal{A}(I) \geq X = \frac{X \cdot Y}{Y} \geq \frac{X}{Y}\,\mathrm{OPT}(I)$$

Beispiel: Greedy-Algorithmus für KNAPSACK

KNAPSACK-Instanz

$M = \{1, \ldots, n\}$, Kosten $c_1, \ldots, c_n$, Gewichte $w_1, \ldots, w_n$, Gesamtgew. W.

Idee:

Bevorzuge Elemente mit günstigem Kosten-pro-Gewicht Verhältnis, also hoher Kostendichte.

$\rightsquigarrow$ Es werden der Reihe nach so viele Elemente wie möglich mit absteigender Kostendichte in die Lösung aufgenommen.

- Berechne die Gewichtsdichten $p_i := \frac{c_i}{w_i}$ für $i = 1, \ldots, n$
- Sortiere nach Kostendichten und indiziere: $p_1 \geq p_2 \geq \cdots \geq p_n$
- 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

Karlsruher Institut für Technologie

- Berechne die Gewichtsdichten $p_i := \frac{c_i}{w_i}$ für $i = 1, \ldots, n$
- Sortiere nach Kostendichten und indiziere: $p_1 \geq p_2 \geq \cdots \geq p_n$
- 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.

Bei Maximierungsproblemen

- Wir wollen $\mathcal{R}_{\mathcal{A}}(I) = \frac{\mathrm{OPT}(I)}{\mathcal{A}(I)} \leq K$, also $\mathcal{A}(I) \geq \frac{1}{K} \cdot \mathrm{OPT}(I)$.
- Wir brauchen:
  - eine **untere** Schranke für $\mathcal{A}(I)$ — “$\mathcal{A}$ ist gut”
  - eine **obere** Schranke für $\mathrm{OPT}(I)$ — “viel besser geht es nicht”

Beispiel: Greedy-Algorithmus für KNAPSACK

- Berechne die Gewichtsdichten $p_i := \frac{c_i}{w_i}$ für $i = 1, \ldots, n$
- Sortiere nach Kostendichten und indiziere: $p_1 \geq p_2 \geq \cdots \geq p_n$
- 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 $w_1 \leq W$.)

Eine untere Schranke für $\mathcal{A}(I)$: $\mathcal{A}(I) \geq c_1 \cdot x_1 = c_1 \cdot \left\lfloor \frac{W}{w_1} \right\rfloor$

Eine obere Schranke für $\mathrm{OPT}(I)$: $\mathrm{OPT}(I) \leq p_1 \cdot W = \frac{c_1}{w_1} \cdot W$

Beispiel: Greedy-Algorithmus für KNAPSACK

- Berechne die Gewichtsdichten $p_i := \frac{c_i}{w_i}$ für $i = 1, \ldots, n$
- Sortiere nach Kostendichten und indiziere: $p_1 \geq p_2 \geq \cdots \geq p_n$
- 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 $w_1 \leq W$.)

Eine untere Schranke für $\mathcal{A}(I)$: $\mathcal{A}(I) \geq c_1 \cdot x_1 = c_1 \cdot \left\lfloor \frac{W}{w_1} \right\rfloor$

Eine obere Schranke für $\text{OPT}(I)$: $\text{OPT}(I) \leq p_1 \cdot W = \frac{c_1}{w_1} \cdot W$

Dann gilt

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

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

Grenzen für den Greedy-Algorithmus

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 $\frac{c_1}{w_1} = 2$ aber $\frac{c_2}{w_2} < 2$.
- Es gilt $\mathcal{A}(I) = c_1$ und $\mathrm{OPT}(I) = 2c_2$, also

$$\frac{\mathrm{OPT}(I)}{\mathcal{A}(I)} = \frac{2c_2}{c_1} = \frac{4w_1 - 6}{2w_1} \longrightarrow 2 \qquad \text{für } w_1 \to \infty$$