

# Theoretische Grundlagen der Informatik

Vorlesung am 13. Dezember 2011

INSTITUT FÜR THEORETISCHE INFORMATIK



- 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  $\Pi$ , für die gilt, dass für alle Probleme  $\Pi' \in \mathcal{NP}$  gilt  $\Pi' \leq \Pi$ , aber für die nicht klar ist, ob  $\Pi \in \mathcal{NP}$ .

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

## 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  $x_1, \dots, x_n \in \mathbb{N}_0$ , so dass

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

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

$$\underbrace{\hspace{10em}}_{A \cdot \bar{x} \leq \bar{b}}$$

## 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  $x_1, \dots, x_n \in \mathbb{N}_0$ , so dass

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

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

$$\underbrace{\hspace{10em}}_{A \cdot \bar{x} \leq \bar{b}}$$

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

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

## Beweis:

Zeigen: SUBSET SUM  $\propto$  INTEGER PROGRAMMING.

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

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



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

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

- 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.

## ■ 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 polynomielle Algorithmen besitzen.
- Solche Algorithmen nennt man **pseudopolynomielle Algorithmen**

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

- Eingabegröße und
- Größe der größten in der Eingabe vorkommenden Zahl beschränkt ist.

## Problem KNAPSACK

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

**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.

## 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

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

- $c_{i+1}^w$  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  $c_n^W \geq C$ .

## Satz:

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

**Beweis:** Berechne  $c_n^W$  wie folgt:

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

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

## Satz:

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

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

## ■ Approximationsalgorithmen für Optimierungsprobleme

## Absoluter Approximationsalgorithmus

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

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

und  $K \in \mathbb{N}_0$  konstant, heißt Approximationsalgorithmus mit Differenzgarantie 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

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

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

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

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

- Vorsicht: Dies ist nicht exakt das Optimierungsproblem zum KNAPSACK-Entscheidungsproblem aus der Vorlesung!

**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  $|\text{OPT}(I) - \mathcal{A}(I)| \leq K$  für alle  $I$ .

Sei  $I = (M, w_j, c_j, W)$  eine KNAPSACK-Instanz.

Betrachte KNAPSACK-Instanz

$$I' = (M' := M, w'_j := w_j, W' := W, c'_j := c_j \cdot (K + 1))$$

Damit ist

$$\text{OPT}(I') = (K + 1) \text{OPT}(I)$$

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

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

## (Widerspruchs-)Beweis

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

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

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

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

für den gilt:

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

Also ist

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

## (Widerspruchs-)Beweis

Also ist

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

Da

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

ist also

$$\text{OPT}(I) = \mathcal{L}(I) .$$

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  $\Pi$  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  $\varepsilon$ -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 Gewichtsichte in die Lösung aufgenommen.

- Berechne die Gewichtsichten  $p_i := \frac{c_i}{w_i}$  für  $i = 1, \dots, n$
- Sortiere nach Gewichtsichtenindiziere:  $p_1 \geq p_2 \geq \dots \geq p_n$
- 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 Gewichtsichten  $p_i := \frac{c_i}{w_i}$  für  $i = 1, \dots, n$
- Sortiere nach Gewichtsichtenindiziere:  $p_1 \geq p_2 \geq \dots \geq p_n$
- 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 Gewichtsichten  $\rho_i := \frac{c_i}{w_i}$  für  $i = 1, \dots, n$
- Sortiere nach Gewichtsichtenindiziere:  $\rho_1 \geq \rho_2 \geq \dots \geq \rho_n$
- 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

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

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

# Beispiel: Greedy-Algorithmus für KNAPSACK

- Berechne die Gewichtsichten  $p_i := \frac{c_i}{w_i}$  für  $i = 1, \dots, n$
- Sortiere nach Gewichtsichtenindiziere:  $p_1 \geq p_2 \geq \dots \geq p_n$
- 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  $W := W - \left\lfloor \frac{W}{w_i} \right\rfloor \cdot w_i$ .

**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  $\text{OPT}(I) = 4w_2 - 2$ , also

$$\frac{\text{OPT}(I)}{\mathcal{A}(I)} = \frac{4w_2 - 2}{2w_1} = \frac{2w_1 - 3}{w_1} \rightarrow 2 \quad \text{für } w_1 \rightarrow \infty$$