Karlsruher Institut für Technologie

Theoretische Grundlagen der Informatik

Vorlesung am 16. Dezember 2014

INSTITUT FÜR THEORETISCHE INFORMATIK

Organisatorisches

- Keine Vorlesung am 23.12.2014.
- Studierende, die die Tutorien von Tobias Stolzmann (Tutorium 18 und 20) besuchen, werden gebeten sich für das restliche Semester ein anderes Tutorium zu suchen. Die Termine der Tutorien können auf der Vorlesungshomepage abgerufen werden.

  Abgabe der Übungsblätter: Die Abgabe der Übungsblätter erfolgt jedoch weiterhin an das jeweilige Tutorium 18 oder 20. Die korrigierten Übungsblätter können in etwa 2,5 Wochen nach der Abgabe bei den Übungsleitern im Büro abgeholt werden.
- Tutorium von Jérome Urhausen wird vom 23.12. auf den 22.12. vorverlegt (Raum 301 und Uhrzeit 17.30 Uhr, einmalige Änderung)

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

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

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

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

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

- 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. $M = \{1, \ldots, n\}$. Für jedes $w \in N_0, w \leq W$ und $i \in M$ definiere

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

- 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 $c_n^W \geq C$.

Beispiel: Problem KNAPSACK

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

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

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

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.

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.

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

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

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

heißt Approximationsalgorithmus mit relativer Gütegarantie. $\mathcal{A}$ 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 $i = 1, \ldots, n$
- 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 $W := W - \left\lfloor \frac{W}{w_i} \right\rfloor \cdot w_i$.

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 $i = 1, \ldots, n$
- 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 $W := W - \left\lfloor \frac{W}{w_i} \right\rfloor \cdot w_i$.

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 $i = 1, \ldots, n$
- 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 $W := W - \left\lfloor \frac{W}{w_i} \right\rfloor \cdot w_i$.

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

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