Berechenbarkeit und Komplexität: Approximationsalgorithmen für NP-harte Probleme

Prof. Dr. Berthold Vöcking
Lehrstuhl Informatik 1
Algorithmen und Komplexität

8. Februar 2008

Motivation

- Wir nennen ein Optimierungsproblem NP-hart, wenn die Entscheidungsvariante des Problems NP-hart ist.
- Unter der Hypothese $P \neq NP$ gibt es für NP-harte Optimierungsprobleme keinen Polynomialzeitalgorithmus, der eine optimale Lösung berechnet.
- In der Praxis ist es häufig jedoch ausreichend eine Lösung zu berechnen, deren Zielfunktionswert vielleicht nicht optimal ist, aber das Optimum annähernd erreicht.

Def: Approximationsalgorithmus

Sei Π ein Optimierungsproblem. Für eine Instanz I von Π bezeichnen wir den optimalen Zielfunktionswert mit $opt(I)$.

- Ein α-*Approximationsalgorithmus*, $\alpha > 1$, für ein Minimierungsproblem Π berechnet für jede Instanz I von Π eine zulässige Lösung mit Zielfunktionswert höchstens $\alpha \cdot opt(I)$.
- Ein α-*Approximationsalgorithmus*, $\alpha < 1$, für ein Maximierungsproblem Π berechnet für jede Instanz I von Π eine zulässige Lösung mit Zielfunktionswert mindestens $\alpha \cdot opt(I)$.

α wird auch als *Approximationsfaktor* oder *Approximationsgüte* bezeichnet.

Def: Approximationsschema

Ein *Approximationsschema A* für ein Optimierungsproblem ist ein Algorithmus, der es ermöglicht, für jedes vorgegebene $\epsilon > 0$ eine zulässige Lösung mit Approximationsgüte $1 + \epsilon$ bzw. $1 - \epsilon$ zu berechnen.

Def: Approximationsschema

Ein *Approximationsschema A* für ein Optimierungsproblem ist ein Algorithmus, der es ermöglicht, für jedes vorgegebene $\epsilon > 0$ eine zulässige Lösung mit Approximationsgüte $1 + \epsilon$ bzw. $1 - \epsilon$ zu berechnen.

FPTAS (fully polynomial time approximation scheme)

A wird als *FPTAS* bezeichnet, falls die Laufzeit polynomiell sowohl in der Eingabelänge *n* als auch in $\frac{1}{\epsilon}$ beschränkt ist.

Mögliche Laufzeitschranken: z.B. $O(n^2/\epsilon)$ oder $O(n^3 + 1/\epsilon^2)$.

Def: Approximationsschema

Ein *Approximationsschema A* für ein Optimierungsproblem ist ein Algorithmus, der es ermöglicht, für jedes vorgegebene $\epsilon > 0$ eine zulässige Lösung mit Approximationsgüte $1 + \epsilon$ bzw. $1 - \epsilon$ zu berechnen.

FPTAS (fully polynomial time approximation scheme)

A wird als *FPTAS* bezeichnet, falls die Laufzeit polynomiell sowohl in der Eingabelänge n als auch in $\frac{1}{\epsilon}$ beschränkt ist.

Mögliche Laufzeitschranken: z.B. $O(n^2/\epsilon)$ oder $O(n^3 + 1/\epsilon^2)$.

PTAS (polynomial time approximation scheme)

A wird als *PTAS* bezeichnet, falls die Laufzeit für jedes konstante $\epsilon > 0$ polynomiell in n beschränkt ist.

Mögliche Laufzeitschranken: z.B. $O(2^{1/\epsilon} n^2)$ oder $O(n^{1/\epsilon} \log n)$.

- Das Rucksackproblem kann durch einen Algorithmus mit Laufzeit $O(N^2 P)$ gelöst werden, wobei $P = \max_{1 \leq i \leq N} p_i$.
- Warum ist diese Laufzeitschranke nicht polynomiell?

- Das Rucksackproblem kann durch einen Algorithmus mit Laufzeit $O(N^2 P)$ gelöst werden, wobei $P = \max_{1 \leq i \leq N} p_i$.
- Warum ist diese Laufzeitschranke nicht polynomiell?
- Obige Laufzeitschranke ist *pseudopolynomiell*, d.h. polynomiell beschränkt bezogen auf die Eingabelänge bei unärer Kodierung.

- Das Rucksackproblem kann durch einen Algorithmus mit Laufzeit $O(N^2 P)$ gelöst werden, wobei $P = \max_{1 \leq i \leq N} p_i$.
- Warum ist diese Laufzeitschranke nicht polynomiell?
- Obige Laufzeitschranke ist *pseudopolynomiell*, d.h. polynomiell beschränkt bezogen auf die Eingabelänge bei unärer Kodierung.
- Das FPTAS für Rucksack basiert auf diesem pseudopolynomiellen Algorithmus.

Der pseudopolynomielle Algorithmus

Wir zeigen

Lemma

Für das Rucksackproblem gibt es einen Algorithmus, der eine optimale Lösung in $O(N^2 P)$ uniformen Rechenschritten berechnet.

Da jeder uniforme Rechenschritt dieses Algorithmus nur polynomiell viele Bits anspricht, hat der Algorithmus auch im logarithmischen Kostenmaß eine pseudopolynomielle Laufzeitschranke.

Der im Lemma beschriebene Algorithmus basiert auf dem Prinzip der dynamischen Programmierung.

Der pseudopolynomielle Algorithmus - Beweis des Lemmas

- Der optimale Nutzenwert liegt offensichtlich zwischen 0 und $N \cdot P$.
- Für $i \in \{0, \ldots, N\}$ und $p \in \{0, \ldots, N \cdot P\}$, sei $A_{i,p}$ das kleinstmögliche Gewicht, mit dem man den Nutzenwert p exakt erreichen kann, wenn man nur Objekte aus der Menge $\{1, \ldots, i\}$ verwenden darf.
- Wir setzen $A_{i,p} = \infty$, falls der Nutzen p nicht durch eine Teilmenge von $\{1, \ldots, i\}$ erreicht werden kann.
- Für alle $i \in \{1, \ldots, N\}$ gilt $A_{i,0} = 0$.
- Außerdem gilt $A_{0,p} = \infty$ für alle $p > 0$ und $A_{i,p} = \infty$ für alle $p < 0$.
- Für $i \in \{1, \ldots, N\}$ und $p \in \{1, \ldots, N \cdot P\}$ gilt nun die folgende Rekursionsgleichung:

$$A_{i+1,p} = \min(A_{i,p}, A_{i,p-p_{i+1}} + w_{i+1}) .$$

Der pseudopolynomielle Algorithmus - Beweis des Lemmas

- Wir verwenden den Ansatz der dynamischen Programmierung und berechnen Zeile für Zeile alle Einträge in der Tabelle bzw. Matrix $(A_{i,p})_{i\in\{0,\ldots,N\},p\in\{0,\ldots,NP\}}$.
- Der gesuchte, maximal erreichbare Nutzenwert ist

$$\max\{p \mid A_{N,p} \leq b\} \ .$$

- Die zugehörige Rucksackbepackung lässt sich durch Abspeichern geeigneter Zusatzinformationen zu den einzelnen Tabelleneinträgen rekonstruieren.
- Laufzeit?

Der pseudopolynomielle Algorithmus - Beweis des Lemmas

- Wir verwenden den Ansatz der dynamischen Programmierung und berechnen Zeile für Zeile alle Einträge in der Tabelle bzw. Matrix $(A_{i,p})_{i\in\{0,\ldots,N\},p\in\{0,\ldots,NP\}}$.
- Der gesuchte, maximal erreichbare Nutzenwert ist

$$\max\{p \mid A_{N,p} \leq b\} \ .$$

- Die zugehörige Rucksackbepackung lässt sich durch Abspeichern geeigneter Zusatzinformationen zu den einzelnen Tabelleneinträgen rekonstruieren.
- Laufzeit? $O(N^2P)$ (= Größe der Tabelle). □

Vom dynamischen Programm zum FPTAS

- Die Laufzeitschranke des dynamichen Programms hängt linear von der Größe der Nutzenwerte ab.
- Zu beachten ist, dass der Algorithmus davon ausgeht, dass die Nutzenwerte – wie in der Problemspezifikation festgelegt – natürliche Zahlen sind.
- Wir können die Größe der Eingabezahlen verringern, indem wir alle Nutzenwerte mit demselben Skalierungsfaktor α multiplizieren (z.B. $\alpha = 0.01$) und die dabei entstehenden Nachkommastellen streichen.
- Was passiert nun, wenn wir das dynamische Programm mit derart skalierten und gerundeten Nutzenwerten aufrufen? ... Diskussion ...

Vom dynamischen Programm zum FPTAS

Für das FPTAS müssen wir "nur" den Rundungsfaktor geeignet wählen ...

Das FPTAS für das Rucksackproblem

1. Skaliere die Nutzenwerte mit dem Faktor $\alpha = \frac{N}{\epsilon P}$ und runde ab, d.h. für $i \in \{1, \ldots, N\}$ setze $p'_i = \lfloor \alpha p_i \rfloor$.
2. Berechne eine optimale Rucksackbepackung $K \subseteq \{1, \ldots, N\}$ für die Nutzenwerte $p'_1, \ldots, p'_N$ mit dem dynamischen Programm aus dem Lemma.

Laufzeitanalyse

- Durch die Skalierung sind die Nutzenwerte nach oben beschränkt durch $P' = \lfloor \alpha P \rfloor = \lfloor \frac{N}{\epsilon} \rfloor$.
- Aus dem Lemma 1 ergibt sich somit unmittelbar eine Laufzeitschranke von $O(N^2 P') = O(N^3/\epsilon)$ uniformen Rechenschritten.

Analyse des Approximationsfaktors

- Sei $K^* \subseteq \{1, \ldots, N\}$ eine optimale Rucksackbepackung.
- Die durch den Algorithmus berechnete Rucksackbepackung bezeichnen wir mit K.
- Es gilt zu zeigen

$$p(K) \geq (1 - \epsilon)\, p(K^*) \ .$$

- Dazu skalieren wir die gerundeten Nutzenwerte virtuell wieder herauf, allerdings ohne dabei den Rundungsfehler rückgängig zu machen, d.h. wir setzen $p_i'' = p_i'/\alpha$.
- Die Rucksackbepackung K ist optimal für die Nutzenwerte p_i' und somit auch optimal für die Nutzenwerte p_i''.

Analyse des Approximationsfaktors

- Bezogen auf die ursprünglichen Nutzenwerte p_i macht der Algorithmus allerdings möglicherweise für jedes Objekt einen Rundungsfehler: Objekte sehen weniger profitabel aus als sie eigentlich sind.
- Der Rundungsfehler für Objekt i lässt sich abschätzen durch

$$p_i - p_i'' = p_i - \frac{\lfloor \alpha p_i \rfloor}{\alpha} \leq p_i - \frac{\alpha p_i - 1}{\alpha} = \frac{1}{\alpha} .$$

- Für eine Teilmenge $S \subseteq \{1, \ldots, N\}$ sei $p(S) = \sum_{i \in S} p_i$ und $p''(S) = \sum_{i \in S} p_i''$.

Analyse des Approximationsfaktors

- Der Rundungsfehler für eine optimale Lösung K^* lässt sich entsprechend abschätzen durch

$$p(K^*) - p''(K^*) \;\leq\; \sum_{i \in K^*} \frac{1}{\alpha} \;\leq\; \frac{N}{\alpha} \;=\; \epsilon\, P \;\leq\; \epsilon\, p(K^*) \;,$$

  wobei die Ungleichung $p(K^*) \geq P$ daraus folgt, dass der optimale Nutzen $p(K^*)$ mindestens so groß ist wie der Nutzen des Objektes mit dem maximalen Nutzenwert P.
- Aus der obigen Ungleichung folgt $p''(K^*) \geq (1-\epsilon)\, p(K^*)$.
- Nun ergibt sich

$$p(K) \;\geq\; p''(K) \;\geq\; p''(K^*) \geq (1-\epsilon)\, p(K^*) \;.$$

Diskussion

- Eine wichtige Grundlage für das FPTAS des Rucksackproblems ist das dynamische Programm mit Laufzeit $O(N^2 P)$.
- Ein anderes dynamisches Programm hat Laufzeit $O(N^2 W)$, wobei W das maximale Gewicht ist.
- Dieser Algorithmus ist also pseudopolynomiell in den Gewichten statt in den Nutzenwerten.
- Um aus diesem Algorithmus ein FPTAS zu gewinnen, müsste man die Gewichte runden. Welche Auswirkungen hätte das?

Diskussion

- Eine wichtige Grundlage für das FPTAS des Rucksackproblems ist das dynamische Programm mit Laufzeit $O(N^2 P)$.
- Ein anderes dynamisches Programm hat Laufzeit $O(N^2 W)$, wobei W das maximale Gewicht ist.
- Dieser Algorithmus ist also pseudopolynomiell in den Gewichten statt in den Nutzenwerten.
- Um aus diesem Algorithmus ein FPTAS zu gewinnen, müsste man die Gewichte runden. Welche Auswirkungen hätte das?
- *Fazit:* Nicht jeder pseudopolynomielle Algorithmus liefert auch ein FPTAS.

Binäre versus unäre Kodierung der Eingabe

- Bisher sind wir durchgängig davon ausgegangen, dass Zahlen in der Eingabe eines Problems binär kodiert werden.
- Wenn wir das Eingabeformat ändern, so erhalten wir ein neues Problem.
- Das Rucksackproblem mit binärer Kodierung haben wir mit KP bezeichnet.
- Sei u-KP die Variante des Rucksackproblems, bei der wir annehmen, dass die Eingabezahlen unär kodiert vorliegen, d.h. wir kodieren eine Zahl $k \in \mathbb{N}$ durch k aufeinander folgende Einsen.

Binäre versus unäre Kodierung der Eingabe

- Wir wissen KP ist NP-hart, hat aber einen Algorithmus mit pseudopolynomieller Laufzeitschranke.
- Die Laufzeit dieses Algorithmus ist polynomiell in der Eingabelänge von u-KP.

Das Rucksackproblem bei unärer Kodierung ist in P, obwohl es in der üblichen binären Kodierung NP-hart ist.

Wie lässt sich dieses Paradoxon erklären?

Definition

- Ein NP-hartes Problem, das bei unärer Kodierung einen polynomiellen Algorithmus hat, wird als *schwach* NP-*hart* bezeichnet.
- Ein Problem, das auch bei unärer Kodierung der Eingabezahlen NP-hart bleibt, wird als *stark* NP-*hart* bezeichnet.

Definition

- Ein NP-hartes Problem, das bei unärer Kodierung einen polynomiellen Algorithmus hat, wird als *schwach* NP-*hart* bezeichnet.
- Ein Problem, das auch bei unärer Kodierung der Eingabezahlen NP-hart bleibt, wird als *stark* NP-*hart* bezeichnet.

schwach NP-hart sind z.B. KP, SUBSET-SUM, PARTITION

stark NP-hart sind z.B. CLIQUE, TSP, BPP

Frage: Warum sind CLIQUE und TSP stark NP-hart?

Stark NP-harte Probleme haben kein FPTAS

Satz

Sei Π ein Optimierungsproblem mit ganzzahliger, nicht-negativer Zielfunktion. Sei p ein geeignetes Polynom. Für eine Eingabe I bezeichne $opt(I) \in \mathbb{N}$ den Wert einer optimalen Lösung und $n_u(I)$ die unäre Eingabelänge. Es gelte $opt(I) < p(n_u(I))$ für jede Eingabe I. Falls Π stark NP-hart ist, so hat Π kein FPTAS, es sei denn $P = NP$.

Stark NP-harte Probleme haben kein FPTAS – Beweis

- Zum Zwecke des Widerspruchs nehmen wir an, Π ist stark NP-hart und hat ein FPTAS.
- Aus dem FPTAS werden wir einen pseudopolynomiellen Algorithmus für Π konstruieren.
- Einen derartigen Algorithmus kann es unter der Hypothese $P \neq NP$ nicht geben. Also hat Π kein FPTAS oder $P = NP$.

(Im Folgenden nehmen wir der Einfachheit halber an, Π ist ein Minimierungsproblem.)

Stark NP-harte Probleme haben kein FPTAS – Beweis

- Sei A ein FPTAS für Π.
- Bei Eingabe I mit unärere Länge $n_u(I)$ setzen wir $\epsilon = 1/p(n_u(I))$.
- Die von A berechnete Lösung hat damit den Wert höchstens

$$(1+\epsilon)\,opt(I) \;<\; opt(I) + \epsilon\, p(n_u(I)) \;=\; opt(I) + 1 \;.$$

Stark NP-harte Probleme haben kein FPTAS – Beweis

- Sei A ein FPTAS für Π.
- Bei Eingabe I mit unärere Länge $n_u(I)$ setzen wir $\epsilon = 1/p(n_u(I))$.
- Die von A berechnete Lösung hat damit den Wert höchstens

$$(1+\epsilon)\,opt(I) \;<\; opt(I) + \epsilon\, p(n_u(I)) \;=\; opt(I) + 1 \;.$$

- Somit berechnet A eine Lösung mit Zielfunktionswert $z \in [opt(I), opt(I)+1)$.
- Aus der Ganzzahligkeit der Zielfunktion folgt, $z = opt(I)$.
- A berechnet also eine optimale Lösung.
- Die Laufzeit von A ist polynomiell in $\frac{1}{\epsilon} = p(n_u(I))$ beschränkt, also polynomiell in $n_u(I)$, und somit pseudopolynomiell.

Diskussion

- Aus dem obigen Satz folgt beispielsweise, dass TSP kein FPTAS hat, da es stark NP-hart ist.
- Die Umkehrung des Satzes gilt nicht: Es gibt Probleme mit pseudopolynomiellen Algorithmen, für die es unter der Hypothese P $\neq$ NP beweisbar kein FPTAS gibt.

Ausblick

- In der Vorlesung *Effiziente Algorithmen* wird das Thema Approximationsalgorithmen intensiver vorgestellt.
- Es wird z.B. gezeigt, dass TSP in seiner allgemeinen Form überhaupt keinen Approximationsalgorithmus mit polynomiell beschränkter Laufzeit hat, es sei denn $P = NP$.
- Nur wenn man bestimmte Forderungen an die Distanzmatrix stellt, also die Eingabe einschränkt, kann man das TSP-Problem bis auf einen konstanten Faktor approximieren.