ETH zürich
Departement Informatik

Approximations- und Online-Algorithmen

Dr. Hans-Joachim Böckenhauer
Prof. Dr. Dennis Komm
Dr. Manuel Wettstein

courses.csedu.inf.ethz.ch/approx_online_alg_24

Lösungsvorschläge – Blatt 5

Zürich, 28. März 2024

Lösung zu Aufgabe 6

Sei $U \in \text{NPO}$ ein Optimierungsproblem, bei dem für alle Probleminstanzen I alle möglichen Lösungswerte nur positive ganze Zahlen sein können und die Kosten $\text{cost}(\text{Opt}(I))$ der optimalen Lösung durch ein Polynom p über den beiden Variablen $|I|$ und $\text{Max-Int}(I)$ beschränkt sind. Es gelte für die Schwellenwertsprache $Lang_U$, dass sie stark NP-schwer ist.

Wir wollen zeigen, dass kein FPTAS für U existieren kann, wenn P $\neq$ NP. Aus der Vorlesung wissen wir, dass es für kein Optimierungsproblem U mit einer stark NP-schweren Schwellenwertsprache $Lang_U$ einen Pseudo-Polynomzeit-Algorithmus geben kann.

Beweis durch Widerspruch: Wir wollen zeigen, dass, wenn für das Problem U ein FPTAS A existiert, auch ein Pseudo-Polynomzeit-Algorithmus A' für U existiert. Sei A ein FPTAS für U. Es existiert der folgende Algorithmus A'.

Gegeben ist die Eingabeinstanz I, sei $\varepsilon = \dfrac{1}{p(|I|, \text{Max-Int}(I)) + 1}$.

Man wendet nun A auf I und ε an, um eine Lösung für I zu bekommen in einer Zeit, die polynomiell in $|I|$ und $1/\varepsilon = p(|I|, \text{Max-Int}(I)) + 1$ ist. Falls U ein Maximierungsproblem ist, gilt für die berechnete Lösung $A'(I)$:

$$\begin{aligned}
|\text{cost}(A'(I)) - \text{cost}(\text{Opt}(I))| &= \text{cost}(\text{Opt}(I)) - \text{cost}(A'(I)) \\
&\leq (1+\varepsilon)\,\text{cost}(A'(I)) - \text{cost}(A'(I)) \\
&\leq \varepsilon\,\text{cost}(\text{Opt}(I)) \\
&= \frac{\text{cost}(\text{Opt}(I))}{p(|I|, \text{Max-Int}(I)) + 1}\,.
\end{aligned}$$

Dabei haben wir verwendet, dass A ein Approximationsschema ist und daher $\text{cost}(\text{Opt}(I)) \leq (1+\varepsilon)\,\text{cost}(A'(I))$ gilt. Da ausserdem $\text{cost}(\text{Opt}(I)) \leq p(|I|, \text{Max-Int}(I))$ ist, können wir nun folgern, dass $|\text{cost}(A'(I)) - \text{cost}(\text{Opt}(I))| < 1$. Falls U ein Minimierungsproblem ist, können wir analog zeigen, dass $|\text{cost}(A'(I)) - \text{cost}(\text{Opt}(I))| = \text{cost}(A'(I)) - \text{cost}(\text{Opt}(I)) < 1$. Da aber alle möglichen Lösungswerte positive ganze Zahlen sind, muss in beiden Fällen die approximative Lösung optimal sein. Es folgt $\text{cost}(A'(I)) = \text{cost}(\text{Opt}(I))$. Damit ist $A'(I)$ ein Pseudo-Polynomzeit-Algorithmus für U.

Das ist ein direkter Widerspruch zur Annahme, dass $Lang_U$ stark NP-schwer ist.

Lösung zu Aufgabe 7

(a) **Idee:** Sei $I = (a_1, \ldots, a_n)$ eine Eingabe für Exact Partition. Wir werden daraus eine Eingabe I' für 2D-Knapsack konstruieren. Wir wollen mit den Kosten genau die Anzahl Elemente in der Lösung zählen und setzen also Kosten $c_i = 1$ für $i = 1, \ldots, n$.
Die Bedingung $\sum_{i \in T} a_i = \sum_{i \notin T} a_i$ ist äquivalent zu $\sum_{i \in T} = \frac{1}{2} \sum_{1 \leq i \leq n} a_i$.
Diese Bedingung schreiben wir wiederum als zwei Ungleichungen

$$\sum_{i \in T} a_i \leq \frac{1}{2} \sum_{1 \leq i \leq n} a_i, \quad \sum_{i \in T} a_i \geq \frac{1}{2} \sum_{1 \leq i \leq n} a_i \ .$$

Die erste dieser Ungleichungen können wir durch die Grössen s_i ausdrücken. Die zweite schreiben wir als

$$\sum_{i \in T} -a_i \leq \frac{1}{2} \sum_{1 \leq i \leq n} -a_i \, .$$

Diese Gewichte sind jedoch negativ. Da ohnehin $|T| = n/2$ gelten soll, ersetzen wir die Bedingung durch

$$\begin{aligned} & a_{\max} \cdot |T| + \sum_{i \in T} -a_i \leq a_{\max} \cdot \frac{n}{2} + \frac{1}{2} \sum_{1 \leq i \leq n} -a_i \\ \iff & \sum_{i \in T} (a_{\max} - a_i) \leq \frac{1}{2} \sum_{1 \leq i \leq n} (a_{\max} - a_i) \, . \end{aligned}$$

Wir setzen Kosten, Gewichte, Grössen und Grenzwerte also wie folgt:

$$\begin{aligned} & c_i = 1 \\ & w_i = a_i, \quad B_w = \frac{1}{2} \sum_{1 \leq i \leq n} a_i \\ & s_i = a_{\max} - a_i, \quad B_s = \frac{1}{2} \sum_{1 \leq i \leq n} (a_{\max} - a_i) \end{aligned}$$

Beweis: Mit den Argumenten oben können wir leicht sehen, dass eine Lösung für I' zu einer Lösung für I mit Kosten $n/2$ führt. Nehmen wir nun an, dass wir eine Lösung für I mit Kosten mindestens $n/2$ haben. Das entspricht $T \subseteq \{1, \ldots, n\}$ mit $|T| \geq n/2$. Aufgrund der Bedingung an die Gewichte gilt

$$\sum_{i \in T} a_i \leq \frac{1}{2} \sum_{1 \leq i \leq n} a_i \, . \tag{1}$$

Aufgrund der Bedingung an die Grössen gilt

$$\sum_{i \in T} (a_{\max} - a_i) \leq \frac{1}{2} \sum_{1 \leq i \leq n} (a_{\max} - a_i) \tag{2}$$

$$\implies a_{\max} \cdot |T| + \frac{1}{2} \sum_{i \in T} a_i \leq a_{\max} \cdot \frac{n}{2} + \sum_{i \in T} a_i \tag{3}$$

und wegen (1) muss damit auch $a_{\max} \cdot |T| \leq a_{\max} \cdot n/2$ gelten, also $|T| \leq n/2$ und damit $|T| = n/2$. Wenn wir das wieder in (3) einsetzen, erhalten wir auch $\sum_{i \in T} a_i \geq \frac{1}{2} \sum_{1 \leq i \leq n} a_i$ und wegen (1) sogar $\sum_{i \in T} a_i = \frac{1}{2} \sum_{1 \leq i \leq n} a_i$. Das heisst T ist eine Lösung für I und I ist eine Ja-Instanz.

Insgesamt hat die optimale Lösung von I' also Kosten von mindestens (sogar genau) $n/2$, genau dann wenn I eine Ja-Instanz ist.

(b) Nehmen wir an, dass es einen FPTAS für 2D-Knapsack gibt und setzen wir $\varepsilon = \frac{1}{n-1}$. Damit können wir in Zeit $\mathcal{O}(p(\varepsilon^{-1} \cdot n)) = \mathcal{O}(p(n^2))$, also in polynomieller Zeit, eine ε-Approximation für die Instanz aus (a) berechnen. Wenn die Instanz also eine Lösung mit Kosten mindestens $n/2$ hat, erhalten wir eine Lösung der Grösse mindestens

$$\frac{1}{1+\varepsilon} \cdot \frac{n}{2} = \frac{1}{1+\frac{1}{n-1}} \cdot \frac{n}{2} = \frac{n-1}{n} \cdot \frac{n}{2} = \frac{n-1}{2} > \frac{n}{2} - 1\,.$$

Das heisst, wir erhalten tatsächlich eine Lösung mit Kosten mindestens $n/2$. Wenn es keine solche Lösung gibt, finden wir sie natürlich auch nicht. Das heisst, mithilfe von Teilaufgabe (a) können wir in polynomieller Zeit Exact Partition lösen. Weil Exact Partition NP-schwer ist, ist das nur möglich, wenn P = NP.

(c) Wir können wie beim normalen Rucksackproblem mit dynamischer Programmierung arbeiten:

Eingabe : Eine Instanz $(B_w, B_s, c_1, \ldots, c_n, w_1, \ldots, w_n, s_1, \ldots, s_n)$
Ausgabe : Eine Indexmenge $T \subseteq \{1, \ldots, n\}$, die eine optimale Lösung für die Instanz I beschreibt.

$\text{TUPLE}'(1) := \{(0,0,0,\emptyset)\} \cup \{(c_1, w_1, s_1, \{1\}) \mid \text{if } w_1 \leq B_w \wedge s_1 \leq B_S\}$;
for $i = 1$ **to** $n-1$ **do**
  $\text{SET}(i+1) := \text{TUPLE}'(i)$;
  foreach $(k, w, s, T) \in \text{TUPLE}'(i)$ **do**
    if $w + w_{i+1} \leq B_w \wedge s + s_{i+1} \leq B_s$ **then**
      $\text{SET}(i+1) := \text{SET}(i+1) \cup \{(k + c_{i+1}, w + w_{i+1}, s + s_{i+1}, T \cup \{i+1\})\}$;
  end
  Bestimme $\text{TUPLE}'(i+1)$ als eine Teilmenge von $\text{SET}(i+1)$, die für jedes erreichbare Paar (w, s) von Gewicht und Grösse in $\text{SET}(i+1)$ genau ein Tripel (m, w, s, T) enthält, mit maximalen Kosten für das gegebene Paar (w, s);
end
$c := \max\{k \in \{1, \ldots, \sum_{i=1}^{n} c_i\} \mid (k, w, s, T) \in \text{TUPLE}'(n) \text{ für ein } w,\ s \text{ und } T\}$;
$T \leftarrow \{T \mid (c, w, s, T) \in \text{TUPLE}'(n) \text{ für ein } w \text{ und } s\}$;

Wo der Algorithmus für das normale Rucksackproblem Laufzeit in $\mathcal{O}(|I| \cdot \text{Max-Int(I)})$ hatte, können wir leicht überprüfen, dass dieser Algorithmus in $\mathcal{O}(|I| \cdot \text{Max-Int(I)}^2)$ läuft.