Lösungsstrategien für NP-schwere Probleme der Kombinatorischen Optimierung

— Übungsblatt 6 —

Walter Stieben (4stieben@inf) Tim Reipschläger (4reipsch@inf) Louis Kobras (4kobras@inf) Hauke Stieler (4stieler@inf)

Abgabe am: 30. Mai 2016

Aufgabe 6.1

a)

Zu zeigen ist, dass der angegebene Algorithmus kein 2-Approximationsalgorithmus ist. Zeigen kann man das mit einem Gegenbeispiel:

Sei $A = \{1, 2, 8\}$ und $B = 10$. Der Algorithmus findet nun folgende Mengen:

Index i	Gefundene Menge S
1	$\{1\}$
2	$\{1, 2\}$
3	$\{1, 2\}$

Der Algorithmus nimmt keine Zahlen mehr ab dem Index auf, da dann die Bedingung $\sum_{a_i \in S} a_i \leq B$ nicht mehr gelten würde, da $1 + 2 + 8 = 11 > 10$ gilt.

Das Ergebnis erfüllt somit nicht die Bedingung eines ρ-Approqimationsalgorithmus für Maximierungsprobleme $L^*/L_A \leq \rho$. Stattdessen gilt für das Ergebnis $L_A = 3$, die totale Summe $L^* = B = 10$ und $\rho = 2$ die Gleichung $L^*/L_A = 10/3 = \overline{3,3} \not\leq \rho$.

Damit ist der angegebene Algorithmus kein 2-Approximationsalgorithmus.

□

b)

Algorithm 1 FindTotalSum

```
1:  procedure FindTotalSum(A, B, ρ)
2:      A ← ConvertToList(A)
3:      A ← MergeSort(A)
4:      T := 0
5:      S := ∅
6:      for i ∈ {n, ..., 1} do
7:          if T + a_i ≤ B then
8:              T ← T + a_i
9:              S ← S ∪ {a_i}
10:         end if
```

11: **end for**
12: **end procedure**

Laufzeitbeweis

Der Algorithmus soll die Laufzeitschranke von $\mathcal{O}(n \cdot \log n)$ nicht überschreiten, was zu beweisen gilt:

Eine Menge in eine Liste zu konvertieren ist bei der Erzeugung einer verketteten Liste in linearer Laufzeit möglich.
Die Liste wird nun mittels MergeSort sortiert. Die worst-case-Laufzeit von MergeSort liegt dabei in $\mathcal{O}(n \cdot \log n)$.
Die Schleife (Zeile 6 bis 11) wird genau n mal ausgeführt. Alle Operationen in der Schleife lassen sich in konstanter Zeit bewerkstelligen, sofern man die Menge genügend schlau implementiert (z.B. als verkettete Liste).

Somit liegt die Gesamtlaufzeit auch in $\mathcal{O}(n \cdot \log n)$.

□

Korrektheitsbeweis

Zunächst kann man allgemein sagen, dass $a_i > a_{i+1}$ mit $1 \leq i \leq n$ gilt und das alle Elemente übersprungen werden für die $a_i > B$ gilt.

Man kann nun zwischen zwei Fällen unterscheiden: Entweder es gibt Elemente a_i für die gilt $a_i > \frac{B}{2}$ mit $1 \leq i \leq n$ oder es gibt sie nicht.

(a) Für den Fall, dass es solche Elemente gibt, nehmen wir eines der Elemente auf und es gilt sofort $T \geq \frac{B}{2}$.

(b) Für den Fall, dass es solche Elemente nicht gibt und für alle Elemente a_i mit $1 \leq i \leq n$ die Ungleichung $a_i \leq \frac{B}{2}$ gilt, kann man wieder zwei Fälle unterscheiden: Entweder es wurden alle Elemente aufgenommen und es gilt $T = L^*$ oder eben nicht.

(b.a) Für den Fall, dass alle Elemente aufgenommen wurden und somit $T = L^*$ gilt, haben wir direkt L^* als Ergebnis und wir brauchen nichts mehr zeigen, weil es das Optimum ist.

(b.b) Für den Fall $T \neq L^*$, so gibt es ein a_i mit $1 \leq i \leq n$, welches nicht aufgenommen wurde (weil sonst Fall b.a gelten würde). Es gilt trotzdem $T > \frac{B}{2}$.
Da $a_i \leq \frac{B}{2}$ ist (Fall b) muss $T \leq B - a_i \leq \frac{B}{2}$ gelten.

Damit gilt $T \geq \frac{B}{2}$ in jedem Fall für den angegebenen Algorithmus.

□

Aufgabe 6.2