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

Hier das Beispiel aus den Vorlesungsfolien, das wir noch genau betrachten werden:
„m Maschinen, $n = 2m + 1$ Jobs. Je zwei Jobs der Längen $m, m + 1, m + 2, \ldots, 2m - 1$ und zusätzlich ein weiterer Job der Länge m."

Anhand dieses Beispiels sollen wir nun erläutern, dass $\frac{3}{4}$ der bestmögliche konstante Gütegarantiefaktor für Greedy-Balance mit LIF-Regel ist.

Wir wollen dazu die folgenden Dinge tun:

- zeigen, wie Greedy-Balance mit LIF-Regel mit Eingaben nach Form des oben angegebenen Beispiels umgeht,
- das Resultat des Algorithmus in Abhängigkeit von m ermitteln und
- diese Approximation mit dem theoretischen bestmöglichen Ergebnis in Bezug setzen.

Wendet man Greedy-Balance mit LIF-Regel auf das Beispiel an, so wird dieses nach einem festen Schema abgearbeitet. Zuerst werden zwei Maschinen die zwei größten Jobs mit den Längen $2m - 1$ zugeordnet, dann zwei weiteren Maschinen die nächstgrößeren Jobs mit den Längen $2m-2$ zugeordnet etc., bis nach m Zuordnungen jeder Maschine ein Job zugeordnet wurde. Da die Jobs der Längen nach absteigend behandelt wurden, werden die Maschinen jetzt in umgekehrter Reihenfolge abgearbeitet. Nach $2m - 2$ Schritten bleiben für die ersten beiden Maschinen noch zwei Jobs der Länge m übrig. Schlussendlich wird der ersten Maschine noch der eine zusätzliche Job der Länge m zugeordnet.
Zur Verdeutlichung hier noch mal als Tabelle:

Für gerade m:

Maschine	1. Job	2. Job	3. Job
1	$2m-1$	m	m
2	$2m-1$	m	—
3	$2m-2$	$m+1$	—
...	...	...	...
m	$\frac{3}{2}m$	$\frac{3}{2}m-1$	—

Für ungerade m:

Maschine	1. Job	2. Job	3. Job
1	$2m-1$	m	m
2	$2m-1$	m	—
3	$2m-2$	$m+1$	—
...	...	...	...
m	$\lfloor\frac{3}{2}m\rfloor$	$\lfloor\frac{3}{2}m\rfloor$	—

Jetzt sind mehrere Dinge leicht zu sehen:

- Die vom Algorithmus für das Beispiel gelieferte Gesamtlänge ist immer gleich, sie steht nämlich in der ersten Zeile, welche sich immer aus dem größten Zeitwert $2m - 1$ und dem kleinsten Zeitwert m, sowie dem einen zusätzlichen m zusammensetzt. Damit liegt das Ergebnis immer bei $4m - 1$.
- Die Summe der Längen aller zu verteilenden Jobs kann man darstellen, indem man sie so aufschlüsselt, wie sie vom Algorithmus auf die Maschinen verteilt werden. Man erhält dann $m \cdot ((2m - (1 + x)) + (m + x)) + m = m \cdot 3m - 1 + m = m \cdot 3m$.
- Die beste vorstellbare Verteilung dieser Gesamtlänge auf die m Maschinen wäre eine Gleichverteilung von $3m$ pro Maschine. In diesem Fall läge das Verhältnis von Approximation zu optimalem Ergebnis bei $\frac{4m-1}{3m}$.
- Ohne zu wissen, ob so eine Gleichverteilung möglich ist, liegt die Lösung zu unserem Beispiel auch immer unter dieser Schranke, denn falls es diese Gleichverteilung nicht gibt, muss die optimale Lösung größer als $3m$ sein, womit die Differenz aus Approximation und optimaler Lösung noch kleiner wäre und somit sicher auch unter $\frac{4}{3}$ liegen würde.
- Für $m \to \infty$ ergibt sich dann $\lim_{m\to\infty} = \frac{4}{3}$. D.h. für dieses Beispiel liegt die Approximation schon um den Faktor $\frac{4}{3}$ vom optimalen Ergebnis entfernt, weshalb es auch keinen besseren konstanten Gütegarantiefaktor geben kann.