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 sei das triviale ausgesprochen: Da A aufsteigend sortiert ist gilt die Ungleichung $a_i < a_{i+1}$, es gibt zudem kein Element doppelt (deswegen auch keine $\leq$-Relation).

Der Algorithmus überspringt zudem alle Elemente die größer als die Schranke B sind. Da diese auch nicht in L^* auftauchen können (weil $L^* \leq B$ gilt), braucht man diese auch nicht gesondert zu betrachten. Relevant wird es ab dem Element $a_k \leq B$ mit $1 \leq k \leq n$. Gibt es kein k für das die Ungleichung gilt (sprich sind alle Elemente größer als B), so ist $S = L^* = 0$.

Für die Hauptschleife (Zeile 6 bis 11) gibt es eine Schleifeninvariante: *Ist $T + a_i > B$, so wird a_i nicht aufgenommen. Findet sich kein a_j mit $0 \leq j \leq i$, für das $T + a_j \leq B$ gilt, so ist $T \geq \frac{L^*}{2}$*. Lässt sich also kein j finden ist der Algorithmus entweder zu Ende oder hat ein genügend genaues Ergebnis geliefert für das gilt $T \geq \frac{L^*}{2}$. Daraus folgt, dass der angegebene Algorithmus ein 2-Approximationsalgorithmus ist.

Beweis der Invariante mittels Widerspruch für *nicht* beendeten Algorithmus

> Lässt sich ein a_j finden ist nichts zu zeigen. Es wird also nur die Situation betrachtet in der sich kein a_j finden lässt und in der der Algorithmus noch nicht zu Ende ist (also wenn $j \neq 1$). Der aktuelle Laufindex der Schleife ist dabei i.

Angenommen es lässt sich kein a_j finden, dann ist $T < \frac{L^*}{2}$.
Da sich kein a_j finden lässt gilt $T + a_j > B$ für jedes a_j mit $1 \leq j < i$. Für diese gilt dadurch $a_j > B - T \geq L^* - T > \frac{L^*}{2}$, was direkt aus $T + a_j > B$ und $T < \frac{L^*}{2}$ folgt. Somit gilt auch, dass jedes $a_k < \frac{L^*}{2}$ mit $i \leq k \leq n$ ist, da sich die bisherige Summe T aus mindestens einem a_k zusammensetzt. Es muss also $a_j > a_k$ gelten.

Die Liste aller Zahlen A ist jedoch aufsteigend sortiert, wodurch $a_j > a_k$ einen Widerspruch darstellt. Daraus folgt, dass $T \geq \frac{L^*}{2}$ gelten muss wenn sich kein a_j finden lässt.

Es fehlt nun noch der Beweis für $T \geq \frac{L^*}{2}$ wenn der Algorithmus zu Ende gekommen ist.

Beweis von $T \geq \frac{L^*}{2}$ für beendeten Algorithmus

Man kann zwei Fälle unterscheiden: Entweder das letzte Element a_1 wurde aufgenommen oder nicht aufgenommen. Wurde es nicht aufgenommen, so folgt aus obigem Beweis, dass $T \geq \frac{L^*}{2}$ gilt und es ist nichts zu zeigen.

Wenn a_1 aufgenommen wurde, ...

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,
- erläutern, warum dieses Beispiel tatsächlich ein Grenzfall ist und warum alle anderen Eingaben deshalb nicht schlechter approximiert werden und im Zuge dessen
- dass Verschiebungen am Beispiel die Gültigkeit des Gütegarantiefaktors nicht beeinträchtigen.

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 $2m-2$ usw. bis nach m Zuordnungen jeder Maschine ein Job zugeordnet wurde. Da die Längen der Jobs 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 $\frac{4m-1}{3m} < \frac{4}{3}$.

- 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 muss. Das gilt insbesondere auch für alle Beispiele, in denen die zu verteilende Gesamtlänge nicht genau durch m teilbar ist.

Rotation durch das Beispiel