Näherungsalgorithmen

(Approximationsalgorithmen)

WiSe 2006/07 in Trier

Henning Fernau
Universität Trier
fernau@informatik.uni-trier.de

Näherungsalgorithmen Gesamtübersicht

- Organisatorisches
- Einführung / Motivation
- Grundtechniken für Näherungsalgorithmen
- Approximationsklassen (Approximationstheorie)

Grundtechniken

- Greedy-Verfahren
- Partitionsprobleme
- Lokale Suche
- Lineares Programmieren
- Dynamisches Programmieren

(Greedy-)Heuristik für Partitionsprobleme allgemein

1. Sortiere L (gemäß eines problemspezifischen Kriteriums)
   Sei $(x_1, \ldots, x_n)$ die sortierte Liste.

2. $P := \{\{x_1\}\}$;

3. Für $i = 2$ bis n tue
   Falls x_i zu einer Menge p aus P hinzugefügt werden kann
   (gemäß einem problemabhängigen Kriterium),
   so füge x_i zu p hinzu
   sonst $P := P \cup \{\{x_i\}\}$.

4. Liefere P zurück.

Scheduling auf identischen Maschinen (MSIM)

I : Menge von Tasks $T = \{t_1, \ldots, t_n\}$, Anzahl p der identischen Maschinen, $l = (l_1, \ldots, l_n)$; Dauer l_j der Ausführung von Task $t_j \in T$
(Tasks sind nicht unterbrechbar!)

S : $\{f : T \to \{1, \ldots, p\}\}$

(oder: Partition $P = \{m_1, \ldots, m_p\}, \bigcup_{i=1}^{p} m_i = T$)

m : Gesamtausführungszeit von T:

$m(f) = max\{\sum_{t_j \in T, f(t_j)=i} l_j \mid 1 \leq i \leq p\}$

opt : min.

Bem.: Historischer Ausgangspunkt für Näherungsalgorithmen: Graham 1969, vgl. das 1. und 2. Kapitel im Buch von Hochbaum!

List-Scheduling:

Heuristische Idee: Gib den nächsten Task an die Maschine, die mit ihrer bisherigen Arbeit am schnellsten fertig ist.

Satz 1: Ist $x = (T, p, l)$ eine Instanz von MSIM, so findet LS eine Näherungslösung vom Wert $m_{LS}(x)$ (unabhängig von der Ordnung von T), welche

$$m_{LS}(x)/m^*(x) \leq (2 - \frac{1}{p})$$

erfüllt.

Beweis: $T = \{t_1, \ldots, t_n\}$. Setze $W := \sum_{k=1}^{n} l_k$. Klar: $m^*(x) \geq W/p$.
Wir nehmen an, LS habe bereits j Tasks zugewiesen. Bezeichne jetzt $A_i(j)$ die Zeit, die notwendig ist, um die davor der Maschine i zugewiesenen Tasks zu beenden, d.h.

$$A_i(j) = \sum_{1 \leq k < j, f(t_k) = i} l_k \quad \text{für das von LS konstruierte Schedule } f.$$

Sei h eine Maschine mit $A_h(n+1) = m_{LS}(x)$ und sei j der Index des Tasks, der als letztes h zugewiesen wurde. t_j wurde von LS einer Maschine mit minimaler Auslastung zugewiesen, d.h.

$$\forall 1 \leq i \leq p : A_i(n+1) \geq A_i(j) \geq A_h(j) = A_h(n+1) - l_j.$$

$$\begin{aligned}
\rightsquigarrow W &= \sum_{i=1}^{p} A_i(n+1) \geq (p-1) \cdot (A_h(n+1) - l_j) + A_h(n+1) \\
&= p \cdot (A_h(n+1) - l_j) + l_j. \\
\rightsquigarrow m_{LS}(x) &= A_h(n+1) \leq \frac{W}{p} + \frac{(p-1)l_j}{p} \\
\rightsquigarrow m_{LS}(x) &\leq m^*(x) + \frac{p-1}{p} m^*(x) = \left(2 - \frac{1}{p}\right) m^*(x), \quad \text{denn } m^*(x) \geq \max\{W/p, l_j\}
\end{aligned}$$

Die in Satz 1 angegebene Schranke ist scharf:

Beispiel: Ggb: p (Maschinenanzahl).
Betrachte die Instanz von MSIM mit $p^2 - p$ Tasks der Länge 1 und einem Task der Länge p.
Optimal wäre es, die $p(p-1)$ kurzen Tasks auf $p-1$ Maschinen zu verteilen und den großen Task auf die verbleibende.
Der optimale Wert wäre also gleich p, denn ein Gesamtarbeitsvolumen von p^2 benötigt auf p Maschinen wenigstens p Schritte.

Wenn der große Task jedoch der Letzte der Liste ist, werden zunächst die $p(p-1)$ kleinen Tasks gleichmäßig auf die p Maschinen verteilt, und dann bekommt irgendeine der Maschinen noch weitere p Schritte Arbeit, nämlich durch den großen Task. Daher liefert List Scheduling dann eine Lösung vom Wert $2p-1$.

Hinweis: online-Algorithmen

Neue Heuristische Idee: LPT (Largest Processing Task)

Sortiere eingangs die Tasks in absteigender Folge gemäß ihrer Laufzeit, d.h. es gilt dann: $l_1 \geq l_2 \geq \ldots \geq l_n$. Danach wird der LIST Scheduling Algorithmus auf die sortierte Liste angewendet. $\rightsquigarrow$ Algorithmus LPTLS

Satz 2: Ist $x = (T = \{t_1, \ldots t_n\}, p, l)$ eine Instanz von MSIM, so findet LPTLS eine Näherungslösung vom Wert $m_{LPTLS}(x)$, welche

$$\frac{m_{LPTLS}(x)}{m^*(x)} \leq \left(\frac{4}{3} - \frac{1}{3p}\right)$$

erfüllt.

Beweis: Nach der Sortierung ist l_n eine der kürzesten Tasklängen.
Wir unterscheiden zwei Fälle:

1. Fall: $l_n > m^*(x)/3$. Dann können höchstens 2 Tasks auf eine Maschine gelegt werden. LPTLS liefert hier sogar eine optimale Lösung, wie man wie folgt einsieht.

   a) Zunächst kommen $\tilde{p} \leq p$ Tasks der Länge $> m^*(x)/2$. LPTLS macht „nichts falsch“.

   b) Dann kommen Tasks der Länge l, $m^*(x)/2 \geq l > m^*(x)/3$.
   Die ersten $\min(2(p-\tilde{p}), n-\tilde{p})$ dieser Tasks werden von LPTLS auf die $p-\tilde{p}$ Maschinen verteilt, die unter a) noch keine Arbeit bekamen. Wegen $l_n > m^*(x)/3$ sind diese $\tilde{p}$ Maschinen somit soweit ausgefüllt, dass sie keine weiteren Tasks mehr aufnehmen können.
   Wenn $n - \tilde{p} = \min(2(p - \tilde{p}), n - \tilde{p})$, so ist die Zuweisung optimal.
   Dann „Tauschargument“!

Für das abschließende Argument übernehmen wir die Bezeichnungsweise vom Beweis von Satz 1, insbesondere betreffend h und j.

2. Fall: $l_n \leq m^*(x)/3$.

Fall 1a: $l_j > m^*/3$:

Gilt $l_j > l_n$, so betrachte die „abgeschnittene“ Instanz

$$x' = (T' = \{x_1, \ldots, x_j\}, \quad p, l' = (l_1, \ldots, l_j)),$$

wobei wir die $\{x_1, \ldots, x_n\}$ als absteigend sortiert annehmen. Es gilt:

$$\frac{m_{LPTLS}(x)}{m^*(x)} = \frac{m_{LPTLS}(x')}{m^*(x)} \leq \frac{m_{LPTLS}(x')}{m^*(x')} = 1$$

nach Fall 1.

2. Fall: $l_j \leq m^*/3$. Das Argument vom Beweis von Satz 1 zeigt:

$$\begin{aligned} m_{LPTLS}(x) &\leq \frac{W}{p} + \frac{(p-1)}{p} \cdot l_j \leq m^*(x) + \frac{p-1}{p} \cdot \frac{1}{3} m^*(x) \\ &\leq \left(\frac{3p + p - 1}{3p}\right) m^*(x) \\ &\leq \left(\frac{4}{3} - \frac{1}{3p}\right) m^*(x) \quad \square \end{aligned}$$

Minimum Bin Packing (MBP)

I : $I = \{a_1, \ldots, a_n\}, a_i \in (0, 1] \cap \mathbb{Q}, 1 \leq i \leq n$

S : Partition $P = \{B_1, \ldots, B_k\}$ von I mit
$\forall 1 \leq j \leq k : \sum_{a_i \in B_j} a_i \leq 1$

m : $k = |P|$

opt : min.

Einfachste Heuristik: **NextFit**.

I wird in der vorgegebenen Reihenfolge abgearbeitet.

a_1 wird in B_1 platziert.

Ist im Verlauf des Algorithmus B_j der zuletzt benutzte Behälter und wird a_i betrachtet, so wird a_i an B_j zugewiesen, falls es noch geht, und sonst an B_{j+1}.

Satz 3: Ist $I = \{a_1, \ldots, a_n\}$ eine Instanz von MBP, so findet NextFit eine Lösung vom Wert $m_{NF}(I)$ mit $m_{NF}(I)/m^*(I) \leq 2$.

Beweis: Sei $A = \sum_{i=1}^{n} a_i$. Da für je zwei aufeinanderfolgende Behälter B_j, B_{j+1} nach Ablauf von NextFit gilt, dass die Summe der Größen $a(B_j)$ und $a(B_{j+1})$ der sich in B_j und B_{j+1} befindenen Gegenstände größer als Eins ist (sonst wären sie ja alle in B_j gelandet), gilt (mit $k = m_{NF}(I)$):

$$2A = a(B_1) + \sum_{l=1}^{k-1} (a(B_l) + a(B_{l+1})) + a(B_k) > a(B_1) + a(B_k) + k - 1$$

$$\rightsquigarrow 2A > k - 1 = m_{NF}(I) - 1$$

Andererseits ist $m^*(I) \geq \lceil A \rceil$. (Einheitsvolumen der Behälter!)

$$\rightsquigarrow m_{NF}(I) < 2m^*(I) + 1.$$

Da die Werte ganzzahlig sind, folgt $m_{NF}(I) \leq 2m^*(I)$. □

Die für NextFit angegebene Schranke ist asymptotisch nicht verbesserbar.

Beispiel: Betrachte dazu $I = \{a_1, \ldots, a_{4n}\}$ mit

$$a_i = \begin{cases} \frac{1}{2n}, & i \text{ ungerade} \\ \frac{1}{2}, & i \text{ gerade} \end{cases}$$

Offenbar ist

$$m_{NF}(I) = 2n$$

und

$$m^*(I) = n + 1.$$

NextFit ist in doppeltem Sinne ein *Online-Algorithmus* (und lässt sich daher auch in Linearzeit mit konstantem Speicher implementieren):

1. Die Gegenstände werden „der Reihe nach“ (so wie sie in der Eingabe stehen) betrachtet. Insbesondere werden sie nicht sortiert.

2. Ein Behälter, in den einmal ein Gegenstand hineingetan wurde, wird nie wieder betrachtet. Er wird sozusagen „geschlossen“.

Wenn man Punkt 2 fallenlässt, braucht man evtl. mehr Speicher, um sich alle möglichen „geöffneten“ Behälter merken zu können.

Die Heuristik **FirstFit**

Der Algorithmus FirstFit öffnet erst dann einen neuen Behälter, wenn der betrachtete Gegenstand a_i nicht in einen der bereits geöffneten Behälter passt. Passt a_i in einen der geöffneten Behälter, so tut FirstFit ihn in den ersten passenden (daher der Name).

Mitteilung: Ist $I = \{a_1, \ldots, a_n\}$ eine Instanz von MBP, so findet FirstFit eine Lösung vom Wert $m_{FF}(I)$ mit

$$m_{FF}(I) \leq \lceil 1,7 \cdot m^*(I) \rceil$$ [siehe Kapitel 2 im Buch von Hochbaum]

Die Heuristik **FirstFitDecreasing**

Dieser Algorithmus lässt auch noch Punkt 1 fallen und sortiert zunächst die Eingabe, bevor der FirstFit-Algorithmus angewendet wird.

In seiner Doktorarbeit hat D. S. Johnson 1973 auf über 70 Seiten folgendes bewiesen:

Mitteilung: Ist $I = \{a_1, \ldots a_n\}$ eine Instanz von MBP, so findet FirstFitDecreasing eine Lösung vom Wert $m_{FFD}(I)$ mit

$$m_{FFD}(I) \leq \frac{11}{9} m^*(I) + 4$$

B. S. Baker hat 1985 im *Journal of Algorithms 6: 49–70* einen kürzeren Beweis für

$$m_{FFD} \leq \frac{11}{9} m^*(I) + 3$$

gegeben.
Wir beschränken uns im folgenden darauf, eine schlechtere Schranke zu beweisen.

Satz 4: Ist $I = \{a_1, \ldots, a_n\}$ eine Instanz von MBP, so findet FirstFitDecreasing eine Lösung vom Wert $m_{FFD}(I)$ mit

$$m_{FFD}(I) \leq 1{,}5 \cdot m^*(I) + 1.$$

Beweis: Betrachte $I = A \cup B \cup C \cup D$ mit

$$\begin{aligned} A &= \left\{ a_i \mid a_i > \frac{2}{3} \right\}, \\ B &= \left\{ a_i \mid \frac{2}{3} \geq a_i > \frac{1}{2} \right\}, \\ C &= \left\{ a_i \mid \frac{1}{2} \geq a_i > \frac{1}{3} \right\}, \\ D &= \left\{ a_i \mid \frac{1}{3} \geq a_i \right\}. \end{aligned}$$

Gibt es einen Behälter, der ausschließlich mit Gegenständen aus D gefüllt wurde, so sind alle Behälter —mit möglicher Ausnahme des zuletzt geöffneten— zu wenigstens $2/3$ gefüllt, woraus die Behauptung hier folgt, denn

$$m^*(I) \geq \lceil \sum a_i \rceil \geq \frac{2}{3} m_{FFD}(I).$$

Gibt es keinen Behälter, der nur Gegenstände aus D enthält, so gilt:

$$m_{FFD}(I) = m_{FFD}(I \setminus D),$$

da alle Behälter auch Gegenstände aus $I \setminus D$ enthalten und die Vorsortierung bewirkt, dass Gegenstände aus $I \setminus D$ bei der Instanz I nicht anders als bei der Instanz $I \setminus D$ behandelt werden.

Wir zeigen nun $m_{FFD}(I \setminus D) = m^*(I \setminus D)$, woraus $m_{FFD}(I) \leq m^*(I)$ folgt.

Jedenfalls ist $m(I \setminus (D \cup A)) + |A| = m(I \setminus D)$ für $m \in \{m^*, m_{FFD}\}$.

Gegenstände von B und C werden nun günstigenfalls „paarweise“ gegliedert, und zwar „kleine“ aus C und mit „großen“ aus B.

Das genau macht m_{FDD}, woraus $m^*(I \setminus (D \cup A)) = m_{FFD}(I \setminus (D \cup A))$ folgt.

Die asymptotische Schärfe der Mitteilung

Beispiel: Betrachte für $n > 0$ (n durch 6 teilbar), die Instanz I_{5n} mit:

n	Gegenstände der Größe $1/2 + \epsilon$	(Typ 1)
n	Gegenstände der Größe $1/4 + 2\epsilon$	(Typ 2)
n	Gegenstände der Größe $1/4 + \epsilon$	(Typ 3)
$2n$	Gegenstände der Größe $1/4 - 2\epsilon$	(Typ 4)

Bestmöglicherweise braucht man $\frac{3}{2}n$ Behälter:
n Behälter sind mit je einem Gegenstand vom Typ 1, 3 und 4 gefüllt,
$\frac{n}{2}$ Behälter sind mit je zwei Gegenständen vom Typ 2 und zwei vom Typ 4 gefüllt.

Dagegen würde FFD wie folgt füllen:
n Behälter mit je einem Gegenstand vom Typ 1 und 2,
$\frac{n}{3}$ Behälter mit je drei Gegenständen vom Typ 3,
$\frac{n}{2}$ Behälter mit je vier Gegenständen vom Typ 4.
$\rightsquigarrow \frac{m_{FFD}(I_{5n})}{m^*(I_{5n})} = \frac{\frac{11}{6}n}{\frac{3}{2}n} = \frac{11}{9}$.

Graphenfärben ist ein Problem, das nicht sofort als Partitionsproblem „vorliegt“:

I : Graph=(V, E)
S : $\{f : V \to \mathbb{N} \mid f$ „Färbung“, d.h. $\forall\{v_1, v_2\} \in E : f(v_1) \neq f(v_2)\}$
m : $|f(V)|$
opt : min.

Graphenfärben ist insofern ein Partitionierungsproblem, als dass V in „gleichfarbige Mengen“ unterteilt werden soll.

Die entsprechende *Partitionierungsheuristik* würde also einen neuen „Knoten“ v_i zu einer „alten“ Farbmenge V_c (Partitionsmenge) hinzufügen, sofern kein Nachbar von v_i bereits in V_c liegt. Widrigenfalls würde eine neue Farbe gewählt.

Satz 5: Betrachte die Knotenmenge des Eingabegraphen $G = (V, E)$ als geordnet: $(v_1, \ldots, v_n)$. (Reihenfolge, in der die Partitionsheuristik arbeitet)
Es sei $G_i = G(\{v_1, \ldots, v_i\})$; $d_i(v)$ bezeichne den Grad von v in G_i.
Die Partitionsheuristik benötigt dann höchstens

$$\max_{1 \leq i \leq n} \min(d_n(v_i), i - 1) + 1$$

viele Farben zum Färben von G.

Beweis: Es sei k_i die Farbanzahl, die der Algortihmus zum Färben von G_i benötigt. Wird keine neue Farbe eingeführt, so gilt in dieser Notation $k_i = k_{i-1}$.
Sonst gilt: $k_i = k_{i-1} + 1$ und $d_i(v_i) \geq k_{i-1}$ (sonst bräuchte man keine neue Farbe).
Wir wollen zeigen:

$$k_j \leq \max_{1 \leq i \leq j} (d_i(v_i)) + 1$$

$$\begin{aligned} j = 0, 1 \quad &: \quad \surd \\ j \to j+1 \quad &: \quad \text{Ist } k_{j+1} = k_j, \text{ so ist die Behauptung gemäß IV wahr.} \\ &\quad\ \text{Ist } k_{j+1} = k_j + 1, \text{ so gilt:} \\ &\quad\ k_{j+1} = k_j + 1 \leq d_{j+1}(v_{j+1}) + 1 \leq \max_{1 \leq i \leq j+1} d_i(v_i) + 1. \end{aligned}$$

Wegen $d_i(v_i) \leq d_n(v_i)$ und $d_i(v_i) \leq i - 1$ folgt die Behauptung. □

Folgerung: Ist Δ der Maximalgrad von G, so liefert die Partitionsheuristik eine Färbung von G mit höchstens $\Delta + 1$ vielen Farben bei beliebiger Anordnung der Knoten.

Um den Ausdruck

$$\max_{1 \le i \le n} \min(d_n(v_i), i - 1)$$

möglichst klein zu bekommen, erscheint das Anordnen der Knoten des Graphen nach absteigendem Grad sinnvoll.
Leider gestattet ein solcher Algorithmus im Allgemeinen keine vernünftigen Güteabschätzungen (und dies ist kein Zufall, wie wir noch später sehen werden).

Betrachte nämlich $G = (V, E)$ mit

$$V = \{x_1, \ldots, x_n, y_1, \ldots, y_n\}, \quad E = \{\{x_i, y_j\} \mid i \neq j\}$$

sowie die Knotenanordnung

$$(x_1, y_1, x_2, y_2, \ldots)$$

Obwohl der Graph zweifärbbar ist (in „x“- und „y“-Farben) benötigt (auch die vorsortierte) Partitionsheuristik hier n Farben (gemäß den Knotenindizes).