Die dynamische Programmierung

Dynamische Programmierung

- Das Ausgangsproblem P_0 wird in Teilprobleme $P_1, \ldots, P_t$ aufgebrochen.
- Die Teilprobleme werden dann, einer Schwierigkeitshierarchie entsprechend, gelöst:
  - Die Schwierigkeitshierarchie ist ein gerichteter azyklischer Graph mit Knotenmenge $P_0, P_1, \ldots, P_t$.
  - Eine Kante (P_i, P_j) wird eingesetzt, wenn die Lösung zu Teilproblem P_i in der Berechnung der Lösung von P_j verwendet wird.

Divide & Conquer ist ein Spezialfall der dynamischen Programmierung:

- Die Schwierigkeitshierarchie entspricht einem Baum.
- Die Lösung eines Teilproblems im Divide & Conquer Verfahren wird nur von dem einen Eltern-Problem benötigt.
- In der dynamischen Programmierung werden Lösungen von vielen Elternknoten benötigt und sind deshalb abzuspeichern.

Das Rucksack-Problem

Zur Erinnerung: RUCKSACK

- Eine Instanz besteht aus n Objekten, wobei Objekt i das Gewicht $g_i \geqslant 0$ und den Wert $w_i \in \mathbb{N}$ besitzt. Ein Rucksack trägt Objekte mit Gesamtgewicht höchstens G.
- Bepacke den Rucksack, so dass das Gesamtgewicht nicht überschritten, aber der Wert eingepackter Objekte maximal ist.

Also: Maximiere $\sum_{i=1}^{n} w_i \cdot x_i$, so dass

$$\sum_{j=1}^{n} g_j \cdot x_j \leqslant G \text{ und } x_1, \ldots, x_n \in \{0, 1\}.$$

- RUCKSACK ist ein Problem der 0-1 Programmierung mit nur einer einzigen Ungleichung.
- RUCKSACK führt auf ein NP-vollständiges Sprachenproblem, kann aber sehr scharf approximiert werden.

Eine exakte Lösung

RUCKSACK kann für n Objekte und Wertesumme $W = \sum_{i=1}^{n} w_i$ in Zeit $O(n \cdot W)$ gelöst werden.

Wir arbeiten mit dynamischer Programmierung:

(1) ***Teilprobleme***: Bestimme für jedes $W^* \leqslant W$ und jedes $i = 1, \ldots, n$

$\text{Gewicht}_i(W^*) =$ das minimale Gesamtgewicht einer Auswahl aus den ersten i Objekten mit Wertesumme genau W^*.

(2) ***Lösung der Teilprobleme***:

$$\text{Gewicht}_i(W^*) = \min\{\text{Gewicht}_{i-1}(W^* - w_i) + g_i, \text{Gewicht}_{i-1}(W^*)\}$$

Die Laufzeit:

Höchstens $nW = n \cdot \sum_{i=1}^{n} w_i$ Teilprobleme, die in Zeit $O(1)$ gelöst werden.

Ein FPTAS für das Rucksack-Problem

Ein FPTAS für RUCKSACK

Der Algorithmus verbraucht bei großer Wertesumme W zuviel Zeit.

(1) Sei $(w_1, \ldots, w_n, g_1, \ldots, g_n, G)$ eine Instanz von RUCKSACK.
- Der Approximationsfaktor $1 + \varepsilon$ sei vorgegeben.
- Entferne alle Objekte, deren Gewicht die Gewichtsschranke G übersteigt.

(2) Packe nur das Objekt mit größtem Wert in den Rucksack. Der erhaltene Wert sei W_1.

(3) Skaliere Werte nach unten

$$s = \frac{\varepsilon \cdot \max_j w_j}{n} \quad \text{und} \quad w_i^* = \lfloor \frac{w_i}{s} \rfloor.$$

(4) Berechne eine exakte Lösung x für die neuen Gewichte. Der erhaltene Wert sei W_2.

(5) Gib die beste der beiden Bepackungen aus.

Analyse (1/2)

- Die neuen Werte sind durch $\frac{n}{\varepsilon}$ beschränkte natürliche Zahlen.
- Eine exakte Lösung gelingt in Zeit $O(n \cdot (n \cdot \frac{n}{\varepsilon})) = O(\frac{1}{\varepsilon} \cdot n^3)$.

- Sei B die gefundene Bepackung und B_{opt} die optimale Bepackung.
- Der Skalierungsfaktor ist $s = \frac{\varepsilon \cdot \max_j w_j}{n}$.

$$\begin{aligned}\sum_{i \in B_{\text{opt}}} w_i &\leqslant \sum_{i \in B_{\text{opt}}} s \cdot (\lfloor \frac{w_i}{s} \rfloor + 1) \leqslant \sum_{i \in B_{\text{opt}}} s \cdot \lfloor \frac{w_i}{s} \rfloor + sn \\ &\leqslant \sum_{i \in B} s \cdot \lfloor \frac{w_i}{s} \rfloor + sn \leqslant \sum_{i \in B} w_i + sn.\end{aligned}$$

Wir wissen:

Es ist $s = \frac{\varepsilon \cdot \max_j w_j}{n}$ und $\sum_{i \in B_{\text{opt}}} w_i \leqslant \sum_{i \in B} w_i + sn$.

- **Fall 1:** $W_2 = \sum_{i \in B} w_i \geqslant \max_j w_j = W_1$. Dann

$$\frac{\sum_{i \in B_{\text{opt}}} w_i}{\sum_{i \in B} w_i} \leqslant \frac{\sum_{i \in B} w_i + sn}{\sum_{i \in B} w_i} \leqslant 1 + \frac{sn}{\sum_{i \in B} w_i} \leqslant 1 + \frac{sn}{\max_j w_j} \leqslant 1 + \varepsilon.$$

- **Fall 2:** $W_2 = \sum_{i \in B} w_i < \max_j w_j = W_1$.
  - Unsere Bepackung hatte den Wert $\max_j w_j$. Es ist

$$\frac{\sum_{i \in B_{\text{opt}}} w_i}{\max_j w_j} \leqslant \frac{\sum_{i \in B} w_i + sn}{\max_j w_j} \leqslant \frac{\max_j w_j + sn}{\max_j w_j} \leqslant 1 + \varepsilon,$$

  denn wir haben die Fallannahme benutzt.

Bin Packing

Bin Packing

- n Objekte mit Gewichten $0 \leqslant g_1, \ldots, g_n \leqslant 1$
- sind in möglichst wenige Behälter mit Kapazität 1 zu verteilen.

- Eine Anwendung: Die Verteilung von Werbespots auf möglichst wenige Programm-Unterbrechungen fester Länge.
- BIN PACKING ist schwieriger als PARTITION

  Für n natürliche Zahlen $x_1, \ldots, x_n$ entscheide, ob es eine Teilmenge $I \subseteq \{1, \ldots, n\}$ mit $\sum_{i \in I} x_i = \frac{1}{2} \cdot \sum_{i=1}^{n} x_i$ gibt.
- PARTITION ist ein BIN PACKING Problem mit Kapazität $\frac{1}{2} \cdot \sum_{i=1}^{n} a_i$ statt Kapazität 1.
  - Die Frage, ob zwei Bins reichen ist ein NP-vollständiges Problem, denn PARTITION ist NP-vollständig.
  - Es gibt keine $(\frac{3}{2} - \varepsilon)$-approximativen Algorithmen für BIN PACKING.

Next Fit,
First Fit und First Fit Decreasing

Next Fit

(1) n Objekte mit Gewichten $g_1, \ldots, g_n \in [0, 1]$ sind in Behälter zu verteilen.

(2) Bearbeite die Objekte der Reihe nach:

Füge das $i$te Objekt in den zuletzt geöffneten Behälter ein, wenn es „passt“. Ansonsten öffne einen neuen Behälter.

Kommentar:

- ★ Zwei aufeinanderfolgende Behälter tragen eine Gesamtbelastung größer als 1.
- ★ Mindestens $\lceil \sum_i g_i \rceil$ Behälter werden benötigt und höchstens $2 \cdot \lceil \sum_i g_i \rceil$ Behälter werden geöffnet.
- ★ Next-Fit ist 2-approximativ.

First Fit

(1) n Objekte mit Gewichten $g_1, \ldots, g_n$ sind in Behälter zu verteilen.

(2) Bearbeite die Objekte der Reihe nach:

- Füge das $i$te Objekt in den ersten Behälter ein, in den es passt.
- Passt das Objekt in keinen Behälter, dann öffne einen neuen Behälter.

Übungsaufgabe

Sei opt die minimale Behälterzahl. Dann öffnet First-Fit höchstens $1.7 \cdot \text{opt} + 2$ Behälter.

First Fit Decreasing

(1) n Objekte mit Gewichten $g_1, \ldots, g_n$ sind in Behälter zu verteilen.

(2) Sortiere die Objekte absteigend nach ihrem Gewicht.

(3) Bearbeite die Objekte der Reihe nach:

- Füge das $i$te Objekt in den ersten Behälter ein, in den es passt.
- Passt das Objekt in keinen Behälter, dann öffne einen neuen Behälter.

Kommentar: First-Fit-Decreasing verhält sich also wie First Fit **nachdem** die Ordnung der Objekte verändert wurde.

- Wir zeigen, dass First Fit Decreasing höchstens $3\mathrm{opt}/2 + 1$ Behälter öffnet.
- Tatsächlich kann sogar gezeigt werden, dass $11\mathrm{opt}/9 + 2$ Behälter genügen.

  Aber $(3/2 - \varepsilon)$-approximative Algorithmen waren doch ausgeschlossen?

First Fit Decreasing: Die Analyse (1/2)

Unterteile die Objekte in vier Gewichtsklassen:

$$A = \{i \mid \frac{2}{3} < g_i\}, \qquad B = \{i \mid \frac{1}{2} < g_i \leqslant \frac{2}{3}\},$$
$$C = \{i \mid \frac{1}{3} < g_i \leqslant \frac{1}{2}\}, \quad D = \{i \mid g_i \leqslant \frac{1}{3}\}.$$

- First-Fit-Decreasing verteilt zuerst die Objekte aus A:
  Jedes dieser schweren Objekte benötigt seinen eigenen Behälter.
- Danach werden die Objekte aus B verteilt.
  Auch jedes B-Objekt benötigt seinen eigenen Behälter.
- Schließlich wird jedes C-Objekt in den Behälter gepackt, dessen verbleibende Restkapazität minimal ist.
- Bisher ist die Verteilung optimal!
  - Ein B-Behälter erhält, wenn möglich, ein C-Objekt.
  - Sonst kann ein Behälters höchstens zwei C-Objekte aufnehmen.

First Fit Decreasing: Die Analyse (2/2)

- Wenn First-Fit-Decreasing keinen Behälter erzeugt, in dem nur Objekte aus D liegen, dann berechnet First-Fit-Decreasing eine optimale Lösung.
  Kommentar: Die Verteilung der Objekte in $A \cup B \cup C$ ist ja optimal.
- Ansonsten ist die Anzahl geöffneter Behälter durch $\frac{3}{2} \cdot \text{opt} + 1$ nach oben beschränkt.
  Kommentar: Werden neue Behälter geöffnet, dann besitzen alle (möglicherweise bis auf den letzten) Behälter ein Mindestgewicht von mehr als $\frac{2}{3}$.

Eine worst-case Eingabe für First Fit Decreasing

- Wir arbeiten mit $5n$ Objekten:
  - n Objekte haben das Gewicht $\frac{1}{2} + \varepsilon$,
  - n Objekte das Gewicht $\frac{1}{4} + 2\varepsilon$,
  - n Objekte das Gewicht $\frac{1}{4} + \varepsilon$
  - und $2n$ Objekte das Gewicht $\frac{1}{4} - 2\varepsilon$.
- Eine optimale Verteilung füllt genau $\frac{3}{2} \cdot n$ Behälter durch
  - n-maliges Zusammenstellen der Gewichte $\frac{1}{2} + \varepsilon$, $\frac{1}{4} + \varepsilon$ und $\frac{1}{4} - 2\varepsilon$,
  - $\frac{n}{2}$-maliges Zusammenstellen der Gewichte $\frac{1}{4} + 2\varepsilon$, $\frac{1}{4} + 2\varepsilon$, $\frac{1}{4} - 2\varepsilon$ und $\frac{1}{4} - 2\varepsilon$.
- Was macht First Fit Decreasing?
  - Zuerst werden n Behälter mit Gewicht $\frac{1}{2} + \varepsilon$ und danach zusätzlich mit Gewicht $\frac{1}{4} + 2\varepsilon$ bepackt.
  - Danach werden jeweils drei Objekte mit Gewicht $\frac{1}{4} + \varepsilon$, gefolgt von jeweils vier Objekten mit Gewicht $\frac{1}{4} - 2\varepsilon$ zusammengepackt.
  - Insgesamt werden $n + \frac{n}{3} + \frac{n}{2} = \frac{11 \cdot n}{6}$ Behälter benötigt.

Ein PTAS für Bin Packing

Aber das geht noch sehr viel besser

- **Die Idee**: Instanzen mit wenigen, nämlich G Gewichtsklassen lassen sich effizient exakt lösen.
- Warum? Verteile n Objekte mit Mindestgewicht ε und höchstens G verschiedenen Gewichten.

- Ein Behälter-Typ wird durch $(a_i : 1 \leqslant i \leqslant G)$ beschrieben.

  a_i ist die Anzahl der Objekte vom $i$ten Gewicht.
- Ein Behälter kann höchstens $\lfloor \frac{1}{\varepsilon} \rfloor$ Objekte aufnehmen.
  - Man kann zeigen, dass die Anzahl der Behälter-Typen durch $B = \binom{\lfloor \frac{1}{\varepsilon} \rfloor + G}{G}$ beschränkt ist: **Keine Abhängigkeit von n**.
- Wir stellen eine vollständige Bepackung durch den Vektor $(b_1, \ldots, b_B)$ dar, wobei b_i die Anzahl der Bins vom Typ i ist.

  Man kann zeigen, dass die Anzahl der verschiedenen Bepackungen durch $\binom{n+B}{B} = \mathrm{poly}(n)$ beschränkt ist.
- **Holzhammermethode**: Betrachte alle $\mathrm{poly}(n)$ Bepackungstypen.

Wir runden auf!

(1) Sortiere die n Objekte nach aufsteigendem Gewicht und zerlege die sortierte Reihenfolge in $G = \lceil \frac{1}{\varepsilon^2} \rceil$ Intervalle mit jeweils höchstens $E = \lfloor n \cdot \varepsilon^2 \rfloor$ Elementen.

(2) Runde die Gewichte in jedem Intervalls auf das größte Gewicht ihres Intervalls.

- Wir bezeichnen die neue Instanz mit „`auf`“
- und nennen die durch Abrundung auf das kleinste Gewicht eines jeden Intervalls erhaltene Instanz „`ab`“.

(3) Wende die obige **Holzhammermethode** auf die Instanz `auf` an; verteile aber nur Objekte mit Mindestgewicht ε.

- Wir verteilen Objekte mit $G = \lceil \frac{1}{\varepsilon^2} \rceil$ Gewichtsklassen und Mindestgewicht ε.
- Die Laufzeit bisher ist durch $\binom{n+B}{B}$ mit $B = \binom{\lfloor \frac{1}{\varepsilon} \rfloor + G}{G}$ beschränkt.

(4) Verteile die leichten Objekte (Gewicht $< \varepsilon$) mit First Fit.

Analyse (1/2)

Jede Gewichtsklasse besteht aus höchstens $E = \lfloor n \cdot \varepsilon^2 \rfloor$ Elementen.

- $\mathrm{bp}(x)$ sei die minimale Anzahl benötigter Behälter für Instanz x.
- Wenn y die vorliegende Instanz ist, dann ist $\mathrm{bp}(\texttt{ab}) \leqslant \mathrm{bp}(y) \leqslant \mathrm{bp}(\texttt{auf})$.
- Instanz `auf` besitzt höchstens E Behälter mehr als Instanz `ab`:

  Bis auf die E Gewichte des letzten Intervalls werden die Gewichte des $i$ten Intervalls von `auf` durch die Gewichte des $(i+1)$sten Intervalls von `ab` dominiert.
- Also ist $\mathrm{bp}(\texttt{auf}) \leqslant \mathrm{bp}(\texttt{ab}) + E \leqslant \mathrm{bp}(y) + E$.
- Jedes Objekt hat Mindestgewicht ε und deshalb $\mathrm{bp}(y) \geqslant n \cdot \varepsilon$.
  - Insbesondere $E = \lfloor n \cdot \varepsilon^2 \rfloor \leqslant \varepsilon \cdot \mathrm{bp}(y)$ und
  - $\mathrm{bp}(\texttt{auf}) \leqslant \mathrm{bp}(\texttt{ab}) + E \leqslant (1+\varepsilon) \cdot \mathrm{bp}(y)$.

Wir wissen

- $\mathrm{bp}(\texttt{auf}) \leqslant (1+\varepsilon)\cdot \mathrm{bp}(y)$.
- Wie schneiden wir bei der Verteilung der leichten Objekte ab?

- Wenn First-Fit keine neuen Behälter öffnet, dann haben wir die Approximationskonstante $1+\varepsilon$ erreicht.
- Wenn First-Fit insgesamt B Behälter öffnet, dann besitzen alle Behälter bis auf den letzten das Mindestgewicht $1-\varepsilon$.
- Das Gesamtgewicht aller Objekte beträgt mindestens $(B-1)\cdot(1-\varepsilon)$.
  - Also ist $\mathrm{bp}(y) \geqslant (B-1)\cdot(1-\varepsilon)$,
  - bzw. für $\varepsilon \leqslant \frac{1}{2}$,

$$B \leqslant \frac{\mathrm{bp}(y)}{1-\varepsilon} + 1 \leqslant (1+2\varepsilon)\cdot \mathrm{bp}(y) + 1.$$

Fazit

Wir haben einen $(1 + 2\varepsilon)$-approximativen Algorithmus erhalten.

- Die Laufzeit ist polynomiell in n, wenn ε fixiert ist.
- Aber das Polynom hat einen utopisch hohen Grad.

- Was haben wir dann gelernt?

  Die Methode der Gewichtsaufrundung –um wenige Gewichtsklassen zu erhalten– sowie die Sonderbehandlung leichter Objekte ist vielversprechend.
- Es gibt effiziente Algorithmen, die höchstens $\texttt{opt} + O(\log_2^2(N))$ Behälter öffnen: N ist die Summe aller Gewichte.
- Wir haben kein volles Approximationsschema erhalten.

  Und das ist auch unmöglich, weil ...

Minimum Makespan Scheduling

MINIMUM MAKESPAN SCHEDULING

- **n** Aufgaben $\mathbf{A_1}, \ldots, \mathbf{A_n}$ mit Laufzeiten $\mathbf{t_1}, \ldots, \mathbf{t_n}$ sind gegeben.
- Die Aufgaben sind so auf **m** Maschinen auszuführen, dass der „**Makespan**“,

  also die für die Abarbeitung aller Aufgaben anfallende maximale Last einer Maschine

  kleinstmöglich ist.

- Warum ist MINIMUM MAKESPAN SCHEDULING schwierig?
- Weil wir damit PARTITION

  Für Zahlen $x_1, \ldots, x_n \in \mathbb{N}$ entscheide, ob es eine Teilmenge $I \subseteq \{1, \ldots, n\}$ mit $\sum_{i \in I} w_i = (\sum_{i=1}^{n} w_i)/2)$ gibt.

  lösen können.
- Wie?
  - Interpretiere die Zahlen x_i als Laufzeiten.
  - Die Eingabe gehört genau dann zu PARTITION, wenn Makespan $(\sum_{i=1}^{n} w_i)/2)$ für zwei Maschinen erreichbar ist.

List Scheduling

List Scheduling

- Die Aufgaben sind gemäß einer „Liste“ abzuarbeiten.
- Weise eine Aufgabe der Maschine zu, die die gegenwärtig niedrigste Last besitzt.

- Wir zeigen, dass List Scheduling 2-approximativ ist.
- Sollte man nicht besser erst die langen, und erst danach die kurzen Aufgaben zuweisen?
  - Natürlich, wenn man den Luxus hat, alle Aufgaben sammeln zu können.
  - Wir erhalten dann einen $4/3$-approximativen Algorithmus.
- Geht es noch besser?
  - Aber ja, mit dynamischer Programmierung, wenn wir Laufzeiten runden.

List Scheduling ist 2-approximativ

- Angenommen, Prozessor i trägt die größte Last.
  Der Makespan T ist dann die Gesamtlaufzeit von Prozessor i.
- Prozessor i möge die Aufgabe j als letzte Aufgabe ausführen.
  - Zum Zeitpunkt $T - t_j$ sind alle Prozessoren „busy“: Ansonsten hätte eine freie Maschine die Aufgabe j früher übernommen. Also ist

$$\sum_{k=1}^{n} t_k \geqslant (m-1) \cdot (T - t_j) + T = mT - (m-1) \cdot t_j \geqslant mT - m \cdot t_j.$$

  - Dividiere durch m und $T - t_j \leqslant \frac{1}{m} \cdot \sum_{k=1}^{n} t_k$ folgt.
  - Sei opt der optimale Makespan. Dann ist
    $$T - t_j \leqslant \frac{1}{m} \cdot \sum_{k=1}^{n} t_k \leqslant \text{opt} \text{ und } t_j \leqslant \text{opt}.$$
  - Also ist $T = (T - t_j) + t_j \leqslant 2 \cdot \text{opt}$.

Lange Aufgaben zuerst

Der Approximationsfaktor sinkt auf höchstens $\frac{4}{3}$, wenn Aufgaben gemäß fallender Bearbeitungszeit präsentiert werden.

- Die Aufgaben sind nach fallenden Bearbeitungszeiten angeordnet: Es ist $t_1 \geqslant t_2 \geqslant \cdots t_m \geqslant t_{m+1} \geqslant \cdots \geqslant t_n$.
- Eine Maschine führt zwei der ersten $m+1$ Aufgaben aus.

  Es gilt $\texttt{opt} \geqslant 2t_{m+1}$ für den optimalen Makespan $\texttt{opt}$.
- Betrachte wieder die Maschine i mit größter Last, die zuletzt die Aufgabe j ausführen möge. Wenn unser Makespan T ist:

$$T - t_j \leqslant \frac{1}{m} \cdot \sum_{k=1}^{n} t_k \leqslant \texttt{opt} \text{ und } t_j \leqslant t_{m+1} \leqslant \frac{\texttt{opt}}{2}.$$

- Folglich ist $T = T - t_j + t_j \leqslant \texttt{opt} + \frac{\texttt{opt}}{2} = \frac{3\texttt{opt}}{2}$.
  - Der Approximationsfaktor höchstens $4/3$ kann mit komplizierterer Analyse gezeigt werden.

Ein PTAS für Minimum Makespan Scheduling

BIN PACKING und MINIMUM MAKESPAN SCHEDULING

- BIN PACKING: Minimiere die Anzahl der Behälter, wobei jeder Behälter die Kapazität 1 besitzt.
- MINIMUM MAKESPAN SCHEDULING: Minimiere die Kapazität für m Behälter.

- BIN PACKING hat ein polynomielles Approximationsschema.
- MINIMUM MAKESPAN SCHEDULING „sollte“ ebenfalls ein polynomielles Approximationsschema besitzen.

Wenn wir nur r Laufzeiten haben:

- n Objekte mit Gewichten aus der Menge $\{t_1, \ldots, t_r\}$ sind auf möglichst wenige Bins der Kapazität höchstens K zu verteilen.
- n_i Objekte haben Gewicht t_i.
- Berechne $\mathrm{Bins}_K(n_1, \ldots, n_r)$, die minimale Anzahl von Bins mit Kapazität K.

Wir berechnen $\mathrm{Bins}_K(n_1, \ldots, n_r)$ mit dynamischer Programmierung.

Berechnung von $\mathrm{Bins}_K(n_1, \ldots, n_r)$

(1) Die $n = \sum_{i=1}^{r} n_i$ Objekte sind in möglichst wenige Bins zu verteilen, wobei jeweils n_i Objekte das Gewicht t_i besitzen.

(2) Bestimme die Menge

$$\begin{aligned} \text{passt} \quad = \quad & \{(m_1, \ldots, m_r) \mid 0 \leqslant m_i \leqslant n_i \text{ für alle } i \text{ und} \\ & \mathrm{Bins}_K(m_1, \ldots, m_r) = 1\}. \end{aligned}$$

Kommentar: Wir prüfen, wann ein einziges Bin ausreicht. Laufzeit proportional zu $n_1 \cdots n_r \leqslant n^r$ genügt.

(3) Wir berechnen $\mathrm{Bins}_K(m_1, \ldots, m_r)$ für alle Kombinationen $(m_1, \ldots, m_r)$ mit $0 \leqslant m_i \leqslant n_i$ für $i = 1, \ldots, r$. Nutze aus

$$\mathrm{Bins}_K(m_1, \ldots, m_r) = 1 + \min_{i \in \text{passt}} \mathrm{Bins}_K(m_1 - i_1, \ldots, m_r - i_r).$$

Kommentar: Laufzeit $O(n^r)$ pro Teilproblem genügt. Insgesamt, bei n^r Teilproblemen, fällt Laufzeit $O(n^{2 \cdot r})$ an.

Die Grundidee

$\text{Bins}_K(n_1, \ldots, n_r)$ kann in Zeit $O(n^{2r})$ bestimmt werden.

- Überprüfe, wenn r nicht zu groß ist, ob Makespan K für m Maschinen erreichbar ist.
- Deshalb runde lange Laufzeiten in wenige verschiedene Laufzeiten.
  - Bestimme einen minimalen, mit m Maschinen erreichbaren Makespan K mit Binärsuche.
  - „Kurze“ Aufgaben fügen wir ganz zuletzt mit List Scheduling ein.

Ein polynomielles Approximationsschema (1/4)

(1) Gegeben sind n Aufgaben $A_1, \ldots, A_n$ mit den Laufzeiten $t_1, \ldots, t_n$. Der Approximationsparameter ε sei ebenfalls gegeben.

(2) Bestimme

$$M = \max\left\{ \frac{1}{m} \cdot \sum_{k=1}^{n} t_k,\ \max_k\{t_k\} \right\}.$$

Kommentar: Die optimale Bearbeitungszeit liegt im Intervall $[M, 2 \cdot M]$.

(3) Führe eine binäre Suche im Interval $[M, 2 \cdot M]$ mit den anfänglichen Intervallgrenzen $L = M$ und $R = 3 \cdot M/2$ durch. Das „Zwillingsintervall“ hat die Grenzen $3 \cdot M/2$ und $2 \cdot M$.

Ein polynomielles Approximationsschema (2/4)

(3a) Runde die Laufzeit t_j einer *langen* Aufgabe ($t_j > R \cdot \varepsilon$) ab:

$$t_j^* = R \cdot \varepsilon \cdot (1+\varepsilon)^k,$$

falls $t_j \in [R \cdot \varepsilon \cdot (1+\varepsilon)^k, R \cdot \varepsilon \cdot (1+\varepsilon)^{k+1}[$.

(3b) Bestimme eine **optimale** Ausführung für die abgerundeten Laufzeiten, wobei wir die Bin-Kapazität R annehmen.

Kommentar: Es gibt $r = \log_{1+\varepsilon} \frac{1}{\varepsilon}$ verschiedene lange Laufzeiten, denn $R \cdot \varepsilon \cdot (1+\varepsilon)^r = R$. Laufzeit $O(n^{2 \cdot r})$ reicht also.

Wenn die optimale Ausführung der langen Aufgaben $b \leqslant m$ Bins der Kapazität R erfordert, dann genügt Kapazität $R \cdot (1+\varepsilon)$ für die ursprünglichen Laufzeiten auf m Maschinen.

Ein polynomielles Approximationsschema (3/4)

(3c) Erhöhe die Kapazität von R auf $R \cdot (1 + \varepsilon)$.

- Nimm alle Rundungen zurück.
- Arbeite die kurzen Aufgaben A_i ab: Füge A_i in ein beliebiges Bin mit Restkapazität mindestens t_i ein oder mache ein neues Bin auf.

Schaffen wir nach Aufnahme der kurzen Aufgaben keine Bepackung mit $\leqslant m$ Bins der Kapazität $R \cdot (1 + \varepsilon)$, dann sind m Bins mit Last größer als R gefüllt.

Wir haben im falschen Intervall gesucht und sollten stattdessen das „Zwillingsintervall“ versuchen.

Ein polynomielles Approximationsschema (4/4)

(3d) Wo wird weiter gesucht?

- Wenn die Anzahl b der benötigten Bins höchstens m ist, dann setze die Binärsuche mit der linken Intervallhälfte von $[L, R]$ fort.

  Wir versuchen, den Makespan zu verkleinern.

- Sonst mache mit der linken Intervallhälfte des Zwillingsintervalls weiter.

  Wir erlauben einen größeren Makespan, da ein optimaler Schedule mit m Maschinen einen größeren Makespan als R besitzt.

- Halte, wenn die Länge des neuen Intervalls höchstens $M \cdot \varepsilon$ ist und gehe ansonsten zu Schritt (3a).

Kommentar: Die Binärsuche wird höchstens $\lceil \log_2 \frac{1}{\varepsilon} \rceil$-mal iteriert. Das Stoppen der Binärsuche verursacht einen weiteren additiven Anstieg des Makespans um $M \cdot \varepsilon \leqslant \mathtt{opt} \cdot \varepsilon$.

(4) Bestimme die Aufgabenverteilung der gefundenen Lösung.

Fazit

- Das Minimum Makespan Problem für n Aufgaben kann in Zeit $O(n^{2r} \cdot \lceil \log_2 \frac{1}{\varepsilon} \rceil)$ gelöst werden, wobei $r = \log_{1+\varepsilon} \frac{1}{\varepsilon}$.

  Wir haben höchstens $\lceil \log_2 \frac{1}{\varepsilon} \rceil$ rekursive Aufrufe mit Laufzeit $O(n^{2r})$ pro Aufruf.

- Die gefundene Lösung besitzt einen Makespan von höchstens $(1 + 2\varepsilon) \cdot \texttt{opt}$.

  Wenn wir keine Lösung mit höchstens m Bins der Kapazität $R \cdot (1 + \varepsilon)$ finden, dann gibt es keine Lösung mit m Maschinen der Kapazität R.

Minimum Makespan mit Prioritäten

MINIMUM MAKESPAN mit Prioritäten

- Für eine Menge P: Führe A_i vor A_j aus, wenn $(A_i, A_j) \in P$.
- Verteile die Aufgaben so auf die m Prozessoren, dass alle Aufgaben zum frühest möglichen Zeitpunkt ausgeführt werden.

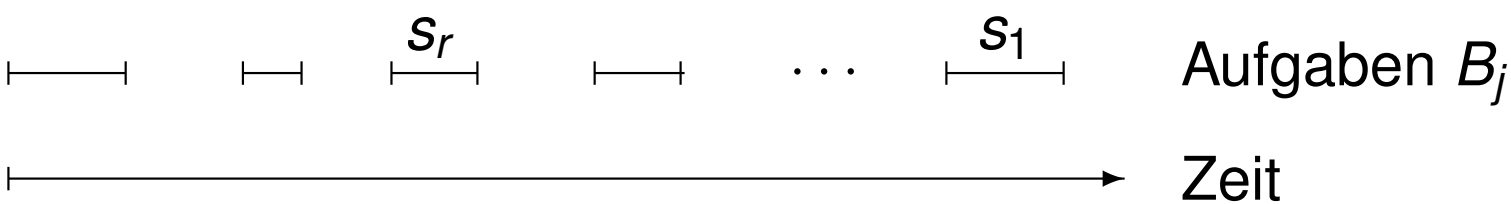
Der Algorithmus: Führe eine beliebige ausführbare Aufgabe auf einer frei gewordenen Maschine aus.

- Welche Aufgaben sind für den Makespan T verantwortlich?
  - Die Aufgabe B_1 terminiere als letzte.
  - Die Aufgabe B_{i+1} terminiere als letzte der Vorgänger von B_i.
- Wenn s_i die Laufzeit von B_i bezeichnet, dann ist $\sum_i s_i \leqslant \mathtt{opt}$.

MINIMUM MAKESPAN mit Prioritäten: Die Analyse

Es ist $\sum_i s_i \leqslant \texttt{opt}$.

- Die Abarbeitung der Aufgaben B_i:



- Alle Maschinen sind in den Lücken beschäftigt:
  Summe aller Lücken $\leqslant \texttt{opt}$.
- Also ist unser Makespan durch $2 \cdot \texttt{opt}$ beschränkt.

Unser Algorithmus ist 2-approximativ.

Das allgemeine Traveling Salesman Problem

Komplexität des Traveling Salesman Problems

- Ohne Einschränkung der Abstandsfunktion gibt es keine 2^n-approximativen Algorithmen.
- Wenn die Abstandsfunktion eine **Metrik** ist, dann ist der Algorithmus von Christofides $3/2$-approximativ. Ein polynomielles Approximationsschema existiert aber nicht, wenn P $\neq$ NP.
- Welche Approximationsfaktoren sind möglich, wenn eine kürzeste Rundreise in $\mathbb{R}^2$ mit der **Euklidschen Distanz** gesucht ist?

- Diese Frage wurde erst in 1996 nach über 20 Jahren von Sanjeev Arora beantwortet.

  Arora wurde dafür in 2010 mit dem Gödel Preis ausgezeichnet.
- Aber zuerst, wie berechnet man eine kürzeste Rundreise im allgemeinen Fall?

Bestimmung einer kürzesten Rundreise

(1) Für jede Teilmenge $S \subseteq \{v_2, \ldots, v_n\}$ und jeden Punkt $v_m \notin S$ bestimme $\text{Rundreise}(S, v_m)$,

die minimale Länge einer Reise von v_1 nach v_m, wobei alle Punkte in S zu besuchen sind.

(2) $\text{Rundreise}(\emptyset, v_i) = d(v_1, v_i)$ für $i = 2, \ldots, n$.

(3) Die „Rekursionsgleichung“

$$\text{Rundreise}(S, v_m) = \min_{w \in S} \{\text{Rundreise}\,(S \setminus \{w\}, w) + d(w, v_m)\}$$

(4) Die kürzeste Rundreise hat dann die Länge $\text{Rundreise}(\{v_2, \ldots, v_n\}, v_1)$.

Wir haben $n \cdot 2^n$ Teilprobleme. Da jedes Teilproblem in Zeit $O(n)$ gelöst werden kann, genügt Zeit $O(n^2 \cdot 2^n)$.

Das Euklid'sche Traveling Salesman Problem

Das Euklidsche TSP in $\mathbb{R}^2$

- n Punkte im $\mathbb{R}^2$ sind gegeben.
- Besuche alle Punkte mit einer möglichst kurzen Rundreise. Die Distanz zwischen zwei Punkten ist ihre euklidsche Distanz.

- O.B.d.A. ist $Q = [0, L_0] \times [0, L_0]$ das kleinste Quadrat, das alle Punkte einschließt.
- Q besitzt zwei Punkte auf gegenüberliegenden Seiten.
  - Eine kürzeste Rundreise wird also mindestens die Distanz vom ersten zum zweiten Punkt und zurück durchlaufen.
  - Wenn $\texttt{opt}$ die Länge einer kürzesten Rundreise ist, folgt also $\texttt{opt} \geqslant 2L_0$.

Wir arbeiten auf einem Gitter

1. Lege ein Gitter mit Seitenlänge $\frac{\varepsilon \cdot L_0}{8n}$ über Q und
2. verschiebe jeden der n Punkte zum nächstliegenden Gitterpunkt.

- Die Länge einer beliebigen Rundreise steigt an um höchstens

$$2n \cdot \frac{\varepsilon L_0}{8n} \leqslant \varepsilon \cdot \mathtt{opt}/8.$$

- Zuletzt teile jede Komponente durch $\varepsilon L_0/(64n)$.
  - Alle Punkte, die sich ja jetzt auf den Gitterpunkten befinden, besitzen ganzzahlige Koordinaten.
  - Alle auf verschiedenen Gitterpunkten sitzende Punkte haben den Mindestabstand 8.
  - Das alle Punkte einschließende Quadrat Q hat die Seitenlänge

$$L = L_0/(\varepsilon L_0/(64n)) = (64n)/\varepsilon = O(n/\varepsilon).$$

Eine allererste Idee

- Zerlege Q in seine vier Teilquadrate der jeweiligen Länge $L/2$.
- Bestimme kurze Rundreisen für jedes Teilquadrat rekursiv und setze sie zusammen.
- Betrachte eine optimale Rundreise R:
  - R zerfällt in Teilrundreisen $R_1, R_2, \ldots$, wobei R_{i+1} genau dann beginnt, wenn die Teilrundreise R_i ihr Teilquadrat verlässt.
  - Insbesondere stößt R möglicherweise mehrmals in dasselbe Teilquadrat und besucht natürlich jeweils verschiedene Punktmengen.

Wieso ist das eine Idee?

Legale Rundreisen

- Setze die Aufteilung des Quadrats Q in jeweils vier Teilquadrate rekursiv fort.
  - Modelliere durch einen Quadtree T_Q, also 4-ären Baum.
  - Markiere die Wurzel von T_Q mit Q,
    die vier Kinder der Wurzel mit jeweils ihrem Teilquadrate von Q.
  - Stoppe den Markierungsprozess an den Knoten, deren Teilquadrat nur einen Punkt enthält: Diese Knoten werden zu Blättern.
- Sei v ein Knoten der Tiefe i, der mit Teilquadrat Q' markiert ist.
  - In der Aufteilung von Q' in vier Teilquadrate werden zwei Trennlinien benutzt, die wir als Trennlinien der Schicht i bezeichnen.
  - Auf jeder Ecke setze eine Tür ein und verteile auf jeder Trennlinie m „**Türen**“ im gleichen Abstand.

    m ist eine Zweierpotenz mit $m \in [\frac{\log_2 n}{\varepsilon}, 2\frac{\log_2 n}{\varepsilon}]$.

Eine **Rundreise** ist genau dann **legal**, wenn sie ein Teilquadrat des Baums T_Q nur über eine Tür betritt und verläßt.