

## Kapitel 7

# Das Rucksackproblem

Effiziente Algorithmen, SS 2018

Professor Dr. Petra Mutzel

VO 14/15 am 5./7. Juni 2018

# Bisher in der VO

## I. Effiziente Graphalgorithmen

- ② Starke Zusammenhangskomponenten
- ③ Matching Probleme
- ④ Maximale Flussprobleme
- ⑤ Amortisierte Analyse
- ⑥ Minimale Schnitte

## Im folgenden:

## II. Approximationsalgorithmen

- ⑦ Rucksackproblem, Bin Packing Problem
- ⑧ Traveling Salesman Problem
- ⑨ Erfüllbarkeitsprobleme

# Approximationsalgorithmen

zunächst Begriffe (Wdhlg. aus GTI bzw. TfAI)

## Optimierungsproblem

- Menge von Instanzen  $I$
- Funktion  $S$ , die für alle  $w \in I$  Menge zulässiger Lösungen  $S(w)$  angibt
- Bewertungsfunktion  $v$ , die für alle  $w \in I$  und alle  $s \in S(w)$  Wert  $v(s)$  angibt
- Optimierungsziel: Maximierung oder Minimierung

zugehöriges

## Entscheidungsproblem

Eingabe  $w \in I$ ,  $k \in \mathbb{N}$

entscheide, ob  $\text{OPT}(w) \leq k$  (bei Minimierung)  
bzw.  $\text{OPT}(w) \geq k$  (bei Maximierung)

# Kombinatorische Optimierungsprobleme

## Definition

**Kombinatorische Optimierungsprobleme** sind Optimierungsprobleme, die eine

- ① **endliche Grundmenge** besitzen
- ② aus deren Elementen sich die zulässigen Lösungen durch **Vereinigung** zusammensetzen lassen, und
- ③ eine Bewertungsfunktion für eine zulässige Lösung, die sich als **gewichtete Summe** der Bewertungen der Grundelemente schreiben lässt.

Man kann kombinatorische Optimierungsprobleme also **in endlicher Zeit** durch Enumeration aller zulässigen Lösungen optimal lösen.

# Beispiele für Kombinatorische Optimierungsprobleme

## Minimum Spanning Tree

Grundmenge: Kantenmenge

zulässige Lösungen: Teilmenge  $T$  der Kanten, die spannenden Bäumen entsprechen

Bewertungsfunktion: Summe der Kantengewichte der Kanten in  $T$

## Rucksackproblem

Grundmenge: Gegenstände

zulässige Lösungen: Teilmenge der Gegenstände, die in den Rucksack passen

Bewertung: Summe der Werte der eingepackten Gegenstände

# Klassifikation von Optimierungsproblemen

## Definition

Ein Optimierungsproblem gehört zu  $\mathcal{NPO}$ , falls

- Test, ob Eingabe  $w$  gültig ist, d.h.  $w \in I$ , in poly. Zeit geht,
- Test, ob Lösung  $s$  zulässig, d.h.  $s \in S(w)$ , in poly. Zeit geht,
- die Bewertung  $v$  in poly. Zeit berechenbar ist, und
- das zugehörige Entscheidungsproblem  $\in \text{NP}$  (konstruktive Lösungen können in Polynomialzeit verifiziert werden)

hier Wir betrachten nur Probleme aus  $\mathcal{NPO}$

## Definition (Güte)

Sei  $s \in S(w)$  (zul. Lösung) zu  $w \in I$ . Dann heißt

$r := \max \left\{ \frac{v(s)}{\text{OPT}(w)}, \frac{\text{OPT}(w)}{v(s)} \right\}$  die Güte der Lösung  $s$ .

# Klassifikation von Approximationsalgorithmen

**Definition** Polynomialzeitalgorithmus  $A$ ,  
der für alle Instanzen Lösung mit Güte  $\leq r$  liefert,  
heißt  **$r$ -Approximation**

**Definition** Polynomialzeitalgorithmus  $A$ ,  
der für jede Güte  $r = 1 + \varepsilon$  mit  $\varepsilon > 0$   
und alle Instanzen Lösung mit Güte  $\leq r$  liefert,  
heißt **polynomielles Approximationsschema**  
Laufzeit beispielsweise  $\mathcal{O}(n^{\frac{1}{\varepsilon}})$

**Definition** polynomielles Approximationsschema  $A$ ,  
dessen Laufzeit polynomiell ist  
in Eingabelänge **und**  $\varepsilon^{-1}$ ,  
heißt **echt polynomielles Approximationsschema**  
Laufzeit beispielsweise  $\mathcal{O}(n \cdot \frac{1}{\varepsilon})$

# Klassifikation von Optimierungsproblemen

- Definition** Optimierungsproblem  
mit  $r$ -Approximation für konstantes  $r \geq 1$   
heißt (konstant) approximierbar und  
gehört zur Klasse  $\mathcal{APX}$
- Definition** Optimierungsproblem  
mit polynomielltem Approximationsschema  
gehört zur Klasse  $\mathcal{PTAS}$
- Definition** Optimierungsproblem  
mit echt polynomielltem Approximationsschema  
gehört zur Klasse  $\mathcal{FPTAS}$
- Definition** Optimierungsproblem  
mit zugehörigem Entscheidungsproblem in  $P$   
gehört zur Klasse  $\mathcal{PO}$



# Klassifikation von Approximationsalgorithmen

Beobachtung  $\mathcal{PO} \subseteq FPTAS \subseteq PTAS \subseteq APX \subseteq NPO$

Man kann zeigen

$$\mathcal{PO} = FPTAS = PTAS = APX = NPO \Leftrightarrow P = NP$$

$$\mathcal{PO} \subsetneq FPTAS \subsetneq PTAS \subsetneq APX \subsetneq NPO \Leftrightarrow P \neq NP$$

Wunsch    Design-Techniken und interessante Beispiele  
für verschiedene Approximationsalgorithmen

# Design-Techniken für Approximationsalgorithmen

- 1 Die Greedy-Methode
- 2 Dynamische Programmierung
- 3 Inkrementelle Algorithmen für Partitionsprobleme
- 4 Spezielle, problemabhängige Verfahren
- 5 LP-basierte Verfahren
- 6 Lokale Suchverfahren

Wir werden Beispiele dazu in diesem und den folgenden Kapiteln kennenlernen.

# Design-Techniken im Verlauf der Vorlesung

- ① Die Greedy-Methode: Rucksack (2-Approximation) (Kap. 7)
- ② Dynamische Programmierung: Rucksack (echt poly. Approximationsschema) (Kap. 7)
- ③ Inkrementelle Algorithmen für Partitionsprobleme: Bin Packing (Kap. 7)
- ④ Spezielle, problemabhängige Verfahren: TSP (Kap. 8)
- ⑤ LP-basierte Verfahren: Max $k$ SAT (Kap. 9)
- ⑥ Lokale Suchverfahren: MaxCut (Kap. 10)

# Die Greedy-Methode

## Grundidee:

Baue iterativ eine Lösung auf und mache dabei jeweils den **lokal besten Schritt**

## Ausführung

- 1 Starte mit  $L = \emptyset$  und der Grundmenge  $S$  aus deren Elementen sich zulässige Lösungen zusammensetzen:  $S = \mathcal{S}$ .
- 2 Sei  $s^* \in S$  ein Element das zur besten lokalen Verbesserung führt unter allen Elementen  $s \in S$  für die  $L \cup \{s\}$  zu einer zulässigen Lösung erweitert werden kann; setze  $S \leftarrow S \setminus \{s^*\}$  und  $L \cup \{s^*\}$ ;
- 3 Falls kein solches  $s^*$  mehr existiert, gehe zu 4, sonst gehe zu 2.
- 4 Ausgabe der zulässigen Lösung  $L$ .

# Bemerkungen zur Greedy-Methode

- **Kruskal** und **Prim** für das MST (Minimum Spanning Tree) Problem sind beide Greedy-Verfahren, die sogar zur Optimallösung führen.
- Dasselbe gilt für **Dijkstra** für kürzeste Wegeprobleme, wenn die Kantenkosten nicht-negativ sind.
- Im allgemeinen führen Greedy Verfahren nicht zur Optimallösung.
- Es gibt stets eine Reihenfolge, die zum Optimum führt, aber i.A. kann man diese nicht im vorhinein berechnen.

Im folgenden: Greedy für das Rucksackproblem

# Greedy für das Rucksackproblem

## Erinnerung

## Rucksackproblem (KP)

### Eingabe

$n$  Objekte mit

Gewichten  $g_1, g_2, \dots, g_n \in \mathbb{N}$

Nutzen  $v_1, v_2, \dots, v_n \in \mathbb{N}$

Gewichtsschranke  $G \in \mathbb{N}$

### Ausgabe

Bepackung  $B \subseteq \{1, 2, \dots, n\}$

mit  $\sum_{i \in B} g_i \leq G$

und  $\sum_{i \in B} v_i$  maximal

## Erinnerung

$KP \in \mathcal{NPO}$

zugehöriges Entscheidungsproblem NP-vollständig

## harmlos

$g_i \leq G$  für alle  $i$  voraussetzen ✓

# GREEDY-KNAPSACK

- 1 Sortiere die Elemente nicht-aufsteigend gemäß ihrem Nutzen-Gewichtverhältnis  $v_i/g_i$ . Sei  $(x_1, \dots, x_n)$  die sortierte Folge der Gegenstände.
- 2 Starte mit  $B = \emptyset$ .
- 3 Für  $j = 1, \dots, n$ :
- 4     Falls  $G \geq g_j$  dann packe Gegenstand  $x_j$  ein:  
 $B \leftarrow B \cup \{x_j\}$  und reduziere  $G \leftarrow G - g_j$ .
- 5 Ausgabe der zulässigen Bepackung  $B$ .

Laufzeit:  $\mathcal{O}(n \log n)$

# Analyse des Greedy

Seien  $m_G$  der von GREEDY-KNAPSACK erhaltene Zielfunktionswert und  $m^*$  der Optimalwert.

Wir betrachten die Instanz mit

$$\begin{aligned} v_i : & \quad 1 \quad 1 \quad \dots \quad 1 \quad (kn - 1) \\ g_i : & \quad \underbrace{1 \quad 1 \quad \dots \quad 1}_{n-1} \quad (kn) \end{aligned}$$

und  $G = kn$  mit beliebig großem  $k \in \mathbb{N}$ . Dann ist  
 $m^* = G - 1 = kn - 1$  und  $m_G = n - 1$  und deshalb

$$\frac{m^*}{m_G} = \frac{kn - 1}{n - 1} > k.$$

D.h. das Verhältnis kann beliebig schlecht werden. Wir können also  
 keine Güte (Approximationsfaktor) zeigen.



# Modifizierter Greedy

**Grund:** Der wertvollste Gegenstand wird nicht genommen.  
**Modifikation:** Wähle die Greedy Lösung **oder** einen Gegenstand mit größtem Wert  $v_{\max}$ .

## Theorem (Approximation Greedy Knapsack)

Für  $m_H := \max\{v_{\max}, m_G\}$  gilt

$$\frac{m^*}{m_H} < 2.$$

Wir haben nun also einen **2-approximativen Algorithmus** für das Rucksackproblem.

# Analyse des modifizierten Greedy

Sei  $j$  der Index des ersten nicht genommenen Gegenstands.

$$\text{Wert bis dahin: } \bar{v}_j = \sum_{i=1}^{j-1} v_i \leq m_G$$

$$\text{Gewicht bis dahin: } \bar{g}_j = \sum_{i=1}^{j-1} g_i \leq G$$

Vertauscht man ein gewähltes  $x_k \in \{x_1, x_2, \dots, x_{j-1}\}$  mit einem nicht gewähltem  $x_l \in \{x_j, x_{j+1}, \dots, x_n\}$  ohne Größenzuwachs ( $g_l \leq g_k$ ), so hat die neue partielle Lösung den Wert

$$\bar{v}' = \bar{v}_j - v_k + v_l.$$

Mit  $g_l \leq g_k$  und  $\frac{v_l}{g_l} \leq \frac{v_k}{g_k}$  erhalten wir  $v_l \leq v_k$  und deshalb  $\bar{v}' \leq \bar{v}_j$ .

# Analyse des Greedy-Verfahrens ff

Das Wertgrößenverhältnis für die Gegenstände in

$\{x_j, x_{j+1}, \dots, x_n\}$  ist höchstens  $\frac{v_j}{g_j}$ .

Mit  $\bar{g}_j + g_j > G$  ( $x_j$  wurde nicht genommen) erhalten wir:

$$\begin{aligned} m^* &\leq \bar{v}_j + \overbrace{(G - \bar{g}_j)}^{< g_j} \frac{v_j}{g_j} \\ &< \bar{v}_j + v_j \end{aligned}$$

Fall 1:  $v_j \leq \bar{v}_j$ . Dann gilt:  $m^* < 2\bar{v}_j \leq 2m_G \leq 2m_H$

Fall 2:  $v_j > \bar{v}_j$ . Dann ist  $v_{\max} > \bar{v}_j$  und es gilt:

$$\begin{aligned} m^* &< \bar{v}_j + v_j \\ &\leq \bar{v}_j + v_{\max} \\ &< 2v_{\max} \\ &\leq 2m_H \end{aligned}$$

In beiden Fällen erhalten wir also  $m^* < 2m_H$ .



# Ein echt poly. Approx.-Schema für das Rucksackproblem

Grundlage: Pseudopolynomieller Algorithmus für KP basierend auf Dynamischer Programmierung

**Definiere**  $M_{i,V} := \min \left\{ \sum_{j \in B} g_j \mid B \subseteq \{1, 2, \dots, i\}, \sum_{j \in B} v_j = V \right\}$   
minimales Gewicht, um Nutzen genau  $V$  zu erzielen,  
durch Auswahl einiger der ersten  $i$  Gegenständen

„Randfälle“  $M_{0,0} = 0$   
 $M_{0,V} = \infty$  für  $V > 0$   
 $M_{i,V} = \infty$  für  $V < 0$

**Rek.-gleichung**  $M_{i,V} = \min\{M_{i-1,V}, M_{i-1,V-v_i} + g_i\}$

**Beobachtung** dynamische Programmierung  
 $\rightsquigarrow$  Laufzeit  $O(n \cdot V_{\max})$

# KP mit kleinen Nutzenwerten schnell lösen

**Laufzeit**  $O(n \cdot V_{\max})$   
mit dynamischer Programmierung

**klar**  $V_{\max} = \sum_{i=1}^n v_i \leq n \cdot \max\{v_1, \dots, v_n\}$  **geeignet**

**klar** Bepackung selbst mit gleichem Aufwand bestimmbar

## Theorem (DP für Rucksackproblem)

Es gibt einen Algorithmus für KP, der in Zeit  $O(n^2 \cdot \max\{v_1, \dots, v_n\})$  eine optimale Lösung berechnet.



# Ein winziges Beispiel

Eingabe  $G = 21, \begin{array}{c|c|c|c|c|c} v_i & 13 & 9 & 6 & 1 & 2 \\ \hline g_i & 17 & 13 & 11 & 4 & 3 \end{array}$

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	∞	∞	∞	∞	4	4
2	∞	∞	∞	∞	∞	3
3	∞	∞	∞	∞	∞	7
4	∞	∞	∞	∞	∞	∞
5	∞	∞	∞	∞	∞	∞
6	∞	∞	∞	11	11	11
7	∞	∞	∞	∞	15	15
8	∞	∞	∞	∞	∞	14
9	∞	∞	13	13	13	13
10	∞	∞	∞	∞	17	17
11	∞	∞	∞	∞	∞	14
12	∞	∞	∞	∞	∞	∞
13	∞	17	17	17	17	17
14	∞	∞	∞	∞	21	21
15	∞	∞	∞	24	24	20

	0	1	2	3	4	5
16	∞	∞	∞	∞	28	24
17	∞	∞	∞	∞	∞	27
18	∞	∞	∞	∞	∞	31
19	∞	∞	∞	28	28	28
20	∞	∞	∞	∞	32	32
21	∞	∞	∞	∞	∞	31
22	∞	∞	30	30	30	30
23	∞	∞	∞	∞	34	34
24	∞	∞	∞	∞	∞	33
25	∞	∞	∞	∞	∞	37
26	∞	∞	∞	∞	∞	∞
27	∞	∞	∞	∞	∞	∞
28	∞	∞	∞	41	41	41
29	∞	∞	∞	∞	45	45
30	∞	∞	∞	∞	∞	44
31	∞	∞	∞	∞	∞	48

Ausgabe 15 (mit  $B = \{1, 5\}$ )

# Ein echt polynomielles Approximationsschema für KP

**Beobachtung** wenn alle  $v_i$  polynomiell klein  
Polynomialzeitalgorithmus verfügbar

**Idee** mache alle  $v_i$  polynomiell klein  

$$v_i \rightsquigarrow \tilde{v}_i := \left\lfloor \frac{v_i}{k} \right\rfloor$$
 $k$  gerade ausreichend groß

**zentrale Einsicht** Zulässigkeit davon unberührt

**Hoffnung** Rundungsfehler werden nicht zu groß

# Eine modifizierte, abgerundete Instanz

**ursprüngliche Instanz** Gewichte  $g_1, \dots, g_n \in \mathbb{N}$   
 Nutzen  $v_1, \dots, v_n \in \mathbb{N}$   
 Gewichtsschranke  $G \in \mathbb{N}$

**modifizierte Instanz** Gewichte  $g_1, \dots, g_n \in \mathbb{N}$   
 Nutzen  $\tilde{v}_1, \dots, \tilde{v}_n \in \mathbb{N}$   
 mit  $\tilde{v}_i := \lfloor \frac{v_i}{k} \rfloor$  mit  $k := \frac{\varepsilon \cdot \max\{v_1, \dots, v_n\}}{(1+\varepsilon) \cdot n}$   
 Gewichtsschranke  $G \in \mathbb{N}$

## Beobachtungen

- $k > 0$
- Bepackungen zulässig für beide Instanzen oder keine Instanz



# Laufzeit des pseudopolynomiellen Algorithmus

$$\begin{aligned}
 & O(n^2 \cdot \max\{\tilde{v}_1, \tilde{v}_2, \dots, \tilde{v}_n\}) \\
 = & O\left(n^2 \cdot \max\left\{\left\lfloor \frac{v_1}{k} \right\rfloor, \left\lfloor \frac{v_2}{k} \right\rfloor, \dots, \left\lfloor \frac{v_n}{k} \right\rfloor\right\}\right) \\
 = & O\left(n^2 \cdot \max_{1 \leq i \leq n} \left\{ \left\lfloor \frac{(1+\varepsilon) \cdot n \cdot v_i}{\varepsilon \cdot \max\{v_1, v_2, \dots, v_n\}} \right\rfloor \right\}\right) \\
 = & O\left(n^2 \cdot \left\lfloor \frac{(1+\varepsilon) \cdot n}{\varepsilon} \right\rfloor\right) = O\left(\frac{n^3}{\varepsilon} + n^3\right)
 \end{aligned}$$

also polynomiell in Eingabelänge und  $1/\varepsilon$

also  $k$  jedenfalls groß genug

# Abschätzung der Güte

**Betrachte** optimale Lösung  $B'$  für ursprüngliche Instanz  
 optimale Lösung  $B$  für modifizierte Instanz  
 berechnet durch pseudopolynomiellen Algorithmus

**zu zeigen** 
$$\frac{\sum_{i \in B'} v_i}{\sum_{i \in B} v_i} \leq 1 + \varepsilon$$

## Beobachtungen

- $\frac{v_i}{k} \geq \tilde{v}_i \geq \frac{v_i}{k} - 1$  (wegen  $\tilde{v}_i := \lfloor \frac{v_i}{k} \rfloor$ )
- $\sum_{i \in B} \tilde{v}_i \geq \sum_{i \in B'} \tilde{v}_i$   
 weil  $B$  optimal für modifizierte Instanz

# Abschätzung zur Güte

$B'$ : optimale Lösung für ursprüngliche Instanz

$B$ : berechnete optimale Lösung für modifizierte Instanz

$$\begin{aligned}
 \sum_{i \in B} v_i &\geq k \cdot \sum_{i \in B} \tilde{v}_i \geq k \cdot \sum_{i \in B'} \tilde{v}_i \geq k \cdot \sum_{i \in B'} \left( \frac{v_i}{k} - 1 \right) \\
 &= \left( \sum_{i \in B'} v_i \right) - k \cdot |B'| = \text{OPT} - k \cdot |B'| \\
 &\geq \text{OPT} - \frac{\varepsilon \cdot \max\{v_1, v_2, \dots, v_n\}}{(1 + \varepsilon) \cdot n} \cdot n \\
 &= \text{OPT} - \frac{\varepsilon}{1 + \varepsilon} \cdot \max\{v_1, v_2, \dots, v_n\} \\
 &\geq \text{OPT} - \frac{\varepsilon}{1 + \varepsilon} \cdot \text{OPT} \\
 &= \text{OPT} \cdot \left( 1 - \frac{\varepsilon}{1 + \varepsilon} \right) = \text{OPT} \cdot \frac{1}{1 + \varepsilon}
 \end{aligned}$$

# Zur Güte

Wir haben

$$\sum_{i \in B} v_i \geq \text{OPT} \cdot \frac{1}{1+\varepsilon}$$

$$= \left( \sum_{i \in B'} v_i \right) \cdot \frac{1}{1+\varepsilon}$$

also

$$\frac{\sum_{i \in B'} v_i}{\sum_{i \in B} v_i} \leq 1 + \varepsilon \checkmark$$

Theorem (FPTAS für Rucksackproblem)

$\text{KP} \in \mathcal{FPTAS}$



# Design-Techniken im Verlauf der Vorlesung

- ① Die Greedy-Methode: Rucksack (2-Approximation) (Kap. 7)
- ② Dynamische Programmierung: Rucksack (echt poly. Approximationsschema) (Kap. 7)
- ③ Inkrementelle Algorithmen für Partitionsprobleme: Bin Packing (Kap. 7)
- ④ Spezielle, problemabhängige Verfahren: TSP (Kap. 8)
- ⑤ LP-basierte Verfahren: Max $k$ SAT (Kap. 9)
- ⑥ Lokale Suchverfahren: MaxCut (Kap. 10)

# Ein Inkrementeller Algorithmus für Bin Packing

Allgemein: Inkrementelle Algorithmen für Partitionsprobleme

**Partitionsprobleme:** Optimierungsprobleme, deren zulässige Lösungen Partitionen der endlichen Grundmenge entsprechen

**Mögliche Ziele:** Minimale Anzahl an Partitionen, Minimiere maximale Partition, ...

**Inkrementeller Algorithmus:** Baut inkrementell eine Ausgabepartition auf

**Beispiele:** Machine Scheduling, Bin Packing, Graphfärbung

# Inkrementelle Algorithmen für Partitionsprobleme

## Grundidee

- ① **Sortiere** die Elemente der endlichen Grundmenge  $\mathcal{S}$ . Sei  $\langle x_1, x_2, \dots, x_n \rangle$  die sortierte Folge
- ② Setze  $P \leftarrow \{\{x_1\}\}$
- ③ Für  $i = 2, \dots, n$ : Falls  $x_i$  zu einer Partitionsmenge  $p \in P$  hinzugefügt werden kann, dann setze  $p \leftarrow p \cup \{x_i\}$ .  
Sonst füge eine neue Partitionsmenge  $\{x_i\}$  zu  $P$  hinzu
- ④ Ausgabe der Partition  $P$

# Das Bin Packing Problem

**Eingabe:** Endliche Menge  $I = \{a_1, a_2, \dots, a_n\}$

$$\forall i \in \{1, 2, \dots, n\} : a_i \in (0, 1], a_i \in \mathbb{Q}$$

**Ausgabe:** Partition  $\{B_1, B_2, \dots, B_k\}$  von  $I$  mit **minimalem  $k$** , so dass

$$\sum_{a_i \in B_j} a_i \leq 1 \text{ für alle } j \in \{1, 2, \dots, k\}.$$

- Suche die kleinste Anzahl an Kisten (**Bins**), in die  $n$  Gegenstände mit Volumina  $a_1, \dots, a_n$  gepackt (**Packing**) werden können (1-dimensional). Die Kisten sind identisch und haben Größe 1.
- Entscheidungsproblem (existiert Bepackung mit  $\leq K$  Kisten?) ist  $\mathcal{NP}$ -vollständig
- Beispiel: Für Tischlerei kleinere Stücke aus großen Holzblöcken schneiden, so dass Abfall minimiert wird
- Packungsprobleme in der Praxis: 2-D- bzw. 3-D-Bin Packing



# First Fit Decreasing für Bin Packing

## Algorithmus: First Fit Decreasing (FFD)

- ① Sortiere die Elemente nicht-aufsteigend nach ihren Volumina  $a_i$ . Sei  $x_1 \geq x_2 \geq \dots \geq x_n$  die sortierte Folge.
- ② Weise  $x_1$  die Kiste  $B_1$  zu.  $B_1$  ist nun die einzige offene Kiste.
- ③ Für  $i = 2, \dots, n$ : Weise  $x_i$  derjenigen offenen Kiste mit dem kleinsten Index zu, in die  $x_i$  passt. Falls  $x_i$  in keine bereits geöffnete Kiste mehr passt, dann öffne eine neue Kiste und weise  $x_i$  dieser zu.

## Beobachtungen:

- korrekt
- Laufzeit  $\mathcal{O}(n^2)$

# Analyse First Fit Decreasing

## Theorem

Sei  $I$  eine Instanz des Bin Packing Problems

- $m_{\text{FFD}}(I)$ : Lösungswert der FFD Heuristik
- $m^*(I)$ : optimaler Lösungswert

Es gilt:

$$m_{\text{FFD}}(I) \leq \frac{3}{2}m^*(I) + 1$$

# Beweis zur Analyse von FFD

Wir partitionieren die sortierte Folge  $x_1 \geq x_2 \geq \dots \geq x_n$  wie folgt:

$$A = \left\{ x_i \mid a_i > \frac{2}{3} \right\}$$

$$B = \left\{ x_i \mid \frac{2}{3} \geq a_i > \frac{1}{2} \right\}$$

$$C = \left\{ x_i \mid \frac{1}{2} \geq a_i > \frac{1}{3} \right\}$$

$$D = \left\{ x_i \mid \frac{1}{3} \geq a_i \right\}$$

Betrachte FFD-Lösung und unterscheide zwei Fälle:

**Fall 1:** Es gibt eine Kiste, die nur Gegenstände aus  $D$  enthält

**Fall 2:** Keine Kiste enthält nur Gegenstände aus  $D$

# Beweis zur Analyse von FFD – Fall 1

**Fall 1:** Es gibt eine Kiste, die nur Gegenstände aus  $D$  enthält

⇒ Es gibt höchstens eine Kiste mit Belegung  $< \frac{2}{3}$  (die letzte geöffnete)

**denn:** D-Gegenstände haben Größe  $a_i \leq \frac{1}{3}$ .

Wenn es also eine zweite Kiste geben würde,  
hätte man diese mit den D-Gegenständen weiter gefüllt

⇒ Bis auf eine Kiste alle  $\geq \frac{2}{3}$  belegt

⇒ Behauptung folgt

## Beweis zur Analyse von FFD – Fall 2

**Fall 2:** Keine Kiste enthält nur Gegenstände aus  $D$

Sei  $I'$  die Instanz  $I$  **ohne** die  $x_i \in D$

- Beobachtung:  $m_{\text{FFD}}(I') = m_{\text{FFD}}(I)$
- **Wir zeigen:**  $m_{\text{FFD}}(I') = m^*(I')$ , also FFD für  $I'$  optimal!  
 $\Rightarrow m_{\text{FFD}}(I) = m^*(I)$ , also FFD für  $I$  optimal!

*Begründung:* Für Elemente aus  $D$  keine neue Kiste notwendig

**Noch zu Zeigen:**  $m_{\text{FFD}}(I') = m^*(I')$

# FFD Analyse: $m_{\text{FFD}}(I') = m^*(I')$

Noch zu Zeigen:  $m_{\text{FFD}}(I') = m^*(I')$

Jede Bepackung hat folgende Eigenschaften:

- 1 Gegenstände aus  $A$  sind immer allein in einer Kiste ( $a_i > \frac{2}{3}$ )  
→ FFD macht nichts falsch
- 2 Höchstens ein  $B$ -Gegenstand pro Kiste ( $a_i > \frac{1}{2}$ )  
→ FFD macht nichts falsch
- 3 Eine Kiste kann höchstens zwei Gegenstände enthalten:  
“ $B + C$ ” oder “ $C + C$ ”  
→ Nur Verteilung der  $C$ -Gegenstände *kritisch*

Beobachtung (\*): Wenn Anzahl der “ $B + C$ ”-Kisten max.  $\Rightarrow$  opt.

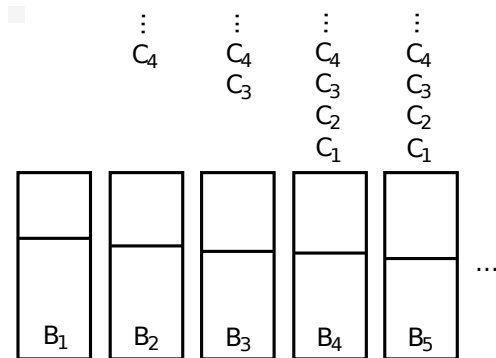
Denn: Zwei beliebige  $C$ -Gegenstände passen in eine “ $C + C$ ”-Kiste  
→ FFD macht nichts falsch

Argument für (\*): da FFD in nicht-aufsteigender Reihenfolge packt, kommt jeder  $C$ -Gegenstand zum größtmöglichen  $B$ -Gegenstand  $\Rightarrow$  optimal



# FFD Analyse: $m_{\text{FFD}}(I') = m^*(I') - \text{Anschauung}$

Legende:  $B_1, \dots, B_5, C_1, \dots, C_4$ : sortierte B- bzw. C-Gegenstände



# FFD Analyse: $m_{\text{FFD}}(I') = m^*(I') - \text{Anschauung 2}$

Legende:  $B_1, \dots, B_5, C_1, \dots, C_4$ : sortierte B- bzw. C-Gegenstände

					⋮
	X	X	X	X	$C_4$
		X	X	X	$C_3$
			X	X	$C_2$
			X	X	$C_1$
					...
$B_1$	$B_2$	$B_3$	$B_4$	$B_5$	



# Diskussion zum Bin Packing Problem

- Aufwändige Fallunterscheidung ermöglicht folgende Verschärfung:  $m_{\text{FFD}}(I) \leq \frac{11}{9}m^*(I) + 1$   
→ Diese Schranke ist scharf, d.h. es existieren Beispiele mit der Eigenschaft
- **Best Fit Decreasing Heuristik (BFD)**: wähle jeweils die am besten passende Kiste (also diejenige mit dem kleinsten Restplatz, in die der Gegenstand geht). Für BFD gilt gleiche Schranke wie für FFD, obwohl die Lösungswerte i.A. nicht identisch sind.
- Verzichtet man auf die Vorsortierung (z.B. Online Optimierung: Daten werden erst nach und nach bekannt), dann kann man zeigen, dass die **First Fit Heuristik** die folgende Schranke erfüllt:  $m_{\text{FF}}(I) \leq \frac{17}{10}m^*(I) + 2$ .