Teil 1

Approximationsalgorithmen

1 Einleitung

In diesem Kapitel wollen wir $\mathcal{NP}$-schwere Optimierungsprobleme untersuchen. Da es unwahrscheinlich ist, dass es für solche Probleme polynomiale Algorithmen gibt, die eine optimale Lösung berechnen, wollen wir polynomiale Approximationsalgorithmen entwerfen.

Bezeichne Π ein Optimierungsproblem und I eine Instanz zu Π. $\mathcal{A}$ sei ein Approximationsalgorithmus für Π, d.h. ein Algorithmus, der zu jeder Instanz I von Π eine zulässige, aber nicht notwendigerweise optimale Lösung berechnet.

- $\mathcal{A}(I)$ bezeichne den Wert der Lösung, die $\mathcal{A}$ für I berechnet.
- OPT(I) bezeichne den Wert einer optimalen Lösung für I.

$\mathcal{A}$(I) heisst *absoluter Approximationsalgorithmus*, falls es ein $K \in \mathbb{N}_0$ gibt, so dass gilt:

$$|\mathcal{A}(I) - \mathrm{OPT}(I)| \leq K \qquad \text{für alle Instanzen } I \text{ von } \Pi.$$

Es gibt nur wenige $\mathcal{NP}$-schwere Optimierungsprobleme, für die es polynomiale, absolute Approximierungsalgorithmen gibt. Häufiger bekannt sind Negativ-Aussagen folgender Form: Falls $\mathcal{P} \neq \mathcal{NP}$, so kann es keinen polynomialen, absoluten Approximationsalgorithmus zu Π geben.

Knapsack

gegeben:

Sei M eine Menge von Objekten. Seien

$$\omega : M \longrightarrow \mathbb{N}_0 \qquad \text{und} \qquad c : M \longrightarrow \mathbb{N}_0$$

Funktionen und W $\in \mathbb{N}_0$.

Problem:

Finde M' $\subseteq$ M mit

$$\sum_{a \in M} c(a) \text{ maximal} \qquad \text{und} \qquad \sum_{a \in M'} \omega(a) \leq \mathrm{W}.$$

Das Knapsack-Problem ist $\mathcal{NP}$-schwer.

Satz 1.1
Falls $\mathcal{P} \neq \mathcal{NP}$, so gibt es keinen (polynomialen) absoluten Approximationsalgorithmus $\mathcal{A}$ für KNAPSACK *mit*

$$|\mathcal{A}(I) - OPT(I)| \leq K \qquad \text{für alle } I \text{ von KNAPSACK und festes } K \in \mathbb{N}_0.$$

Beweis: Angenommen es gäbe einen solchen Algorithmus $\mathcal{A}$. Betrachte eine beliebige Instanz I mit M, $w\colon M \to \mathbb{N}_0$, $c\colon M \to N_0$ und $W \in \mathbb{N}_0$. Betrachte dazu Instanz I' mit M, w, W und $c'\colon M \to \mathbb{N}_0$ definiert durch

$$c'(x) := (K+1) \cdot c(x).$$

Für jedes I' liefert der $\mathcal{A}$ den Wert $\mathcal{A}(I')$ mit

$$|\mathcal{A}(I') - \mathrm{OPT}(I')| \leq K.$$

$\mathcal{A}(I')$ induziert einen Lösung M^* für I mit Wert $c(M^*)$, für die gilt:

$$|(K+1) \cdot c(M^*) - (K+1) \cdot \mathrm{OPT}(I)| \leq K.$$

Also ist

$$|c(M^*) - \mathrm{OPT}(I)| \leq \frac{K}{K+1} < 1.$$

Dies ist ein Widerspruch zu $\mathcal{P} \neq \mathcal{NP}$. □

2 Approximationsalgorithmen mit relativer Gütegarantie

Ein polynomialer Algorihmus $\mathcal{A}$, der für ein vorgegebenes Optimierungsproblem Π für jedes $\mathcal{A}(I)$ von Π einen Wert $\mathcal{A}(I)$ liefert mit

$$\mathcal{R}_{\mathcal{A}}(I) \leq K, \text{wobei}$$

$$\mathcal{R}_{\mathcal{A}}(I) \quad := \quad \begin{cases} \frac{\mathcal{A}(I)}{\mathrm{OPT}(I)} & \text{falls } \Pi \text{ Minimierungsproblem} \\ \\ \frac{\mathrm{OPT}(I)}{\mathcal{A}(I)} & \text{falls } \Pi \text{ Maximierungsproblem} \end{cases}$$

und K Konstante ($K \geq 1$), heisst *absoluter Approximationsalgorithmus* mit *relativer Gütegarantie* oder auch *relativer Approximationsalgorithmus*.

Zu einem relativen Approximationsalgorithmus zu Π definiere

$$\mathcal{R}_{\mathcal{A}} \quad := \quad \inf\{r \geq 1 : \ \mathcal{R}_{\mathcal{A}}(I) \leq r \quad \text{für alle } I \text{ von } \Pi\}.$$

$\mathcal{A}$ heisst *ε-approximierender Algorithmus* falls $\mathcal{R}_{\mathcal{A}} \leq 1 + \varepsilon$.

Allgemeines KNAPSACK-Problem

gegeben:

Seien $\omega_1, \ldots, \omega_n \in \mathbb{N}$, $c_1, \ldots, c_n \in \mathbb{N}_0$ und $W, C \in \mathbb{N}_0$.

Problem:

Existieren $x_1, \ldots, x_n \in \mathbb{N}_0$ mit folgenden Eigenschaften

$$\sum_{i=1}^{n} x_i w_i \leq W \qquad \text{und} \qquad \sum_{i=1}^{n} x_i c_i \leq C \ ?$$

Das bedeutet, dass von jedem Exemplar mehrere „eingepackt“ werden können. Dieses Problem ist natürlich auch $\mathcal{NP}$-vollständig.

Beispiel: Greedy-KNAPSACK-Algorithmus für $\max \sum_{i=1}^{n} x_i c_i$

Algorithmus:

1. Schritt: Zunächst berechne die „Gewichtsdichten“ $p_i := \frac{c_i}{w_i}$ und indiziere so, dass
$$p_1 \geq p_2 \geq \ \ldots \ \geq p_n \text{ für} i = 1, \ \ldots \ , n.$$

2. Schritt: Für $i = 1$ bis n führe aus

3. Schritt: Setze $x_i := \left\lfloor \frac{W}{w_i} \right\rfloor$ und $W := W - \left\lfloor \frac{W}{w_i} \right\rfloor \cdot w_i$.

Im 2. Schritt wird nach den Gewichtsdichten sortiert. An der Beschreibung des Algorithmus sieht man, dass soviele Exemplare der aktuellen Gewichtsklasse wie möglich eingepackt werden. Die Laufzeit des Greedy-KNAPSACK-Algorithmus wird offensichtlich durch das Sortieren der p_i in Schritt 1 dominiert. Da Sortieren in $\mathcal{O}(n \log n)$ möglich ist, ergibt sich für diesen Algorithmus eine Laufzeit von $\mathcal{O}(n \log n)$. ■

Satz 2.1
*Der Greedy-*KNAPSACK*-Algorithmus $\mathcal{A}$ erfüllt $\mathcal{R}_{\mathcal{A}} = 2$.*

Beweis: Œ sei $w_1 \leq W$. Es gilt offensichtlich:

$$\mathcal{A}(I) \geq c_1 \cdot X_1 = c_1 \cdot \left\lfloor \frac{W}{w_1} \right\rfloor \qquad \text{für alle } I, \text{ und}$$

$$\begin{aligned} \mathrm{OPT}(I) &\leq c_1 \cdot \left\lfloor \frac{W}{w_1} \right\rfloor \leq c_1 \cdot \left(\left\lfloor \frac{W}{w_1} \right\rfloor + 1 \right) \\ &\leq 2 \cdot c_1 \cdot \left\lfloor \frac{W}{w_1} \right\rfloor \\ &\leq 2 \cdot \mathcal{A}(I), \end{aligned}$$

also $\mathcal{R}_{\mathcal{A}} \leq 2$.

Behandle nun folgendes spezielle KNAPSACK-Beispiel: Sei $n = 2$, $c_1 = 2 \cdot w_1$, $c_2 = 2 \cdot w_2 - 1$, $w_2 = w_1 - 1$ und $W = 2 \cdot w_2$. Offensichtlich ist nun:

$$\frac{c_1}{w_1} = \frac{2 \cdot w_1}{w_1} = 2 > \frac{c_2}{w_2} = \frac{2 \cdot w_2 - 1}{w_2}.$$

$\mathcal{A}(I) = 2 \cdot w_1$ und $\mathrm{OPT}(I) = 4 \cdot w_2 - 2$, d.h.

$$\frac{\mathrm{OPT}(I)}{\mathcal{A}(I)} = \frac{2 \cdot w_1 - 3}{w_1} \xrightarrow{w_1 \to \infty} 2$$

d.h. ein besseres Greedy-Verfahren kann man nicht finden. Damit ist also $R_{\mathcal{A}} = 2$. □

3 Bin Packing (Optimierungsversion)

gegeben:

Sei M = $\{a_1, \ldots, a_n\}$ eine endliche Menge mit der Gewichtsfunktion

$$s\colon M \longrightarrow (0, 1].$$

gesucht:

Man möchte nun eine Zerlegung von M in eine minimale Anzahl von Teilmengen $B_1, \ldots, B_m$ finden, so dass

$$\sum_{a_i \in B_j} s(a_i) \leq 1 \qquad \text{für } 1 \leq j \leq m.$$

Das Entscheidungsproblem Bin Packing ist äquivalent zu dem Problem MULTIPROCESSOR SCHEDULING und daher $\mathcal{NP}$-vollständig.

Approximationsalgorithmen für Bin Packing

Next Fit (NF)

Algorithmus:

1. Schritt: Durchlaufe M beginnend mit a_1 und füge a_1 in B_1 ein.

2. Schritt: Wenn a_ℓ eingefügt werden muss, betrachte die letzte, nichtleere Menge B_j:

3. Schritt: Falls
$$s(a_\ell) \leq 1 - \sum_{a_i \in B_j} s(a_i),$$
dann füge a_ℓ in B_j ein; ansonsten füge a_ℓ in B_{j+1} ein.

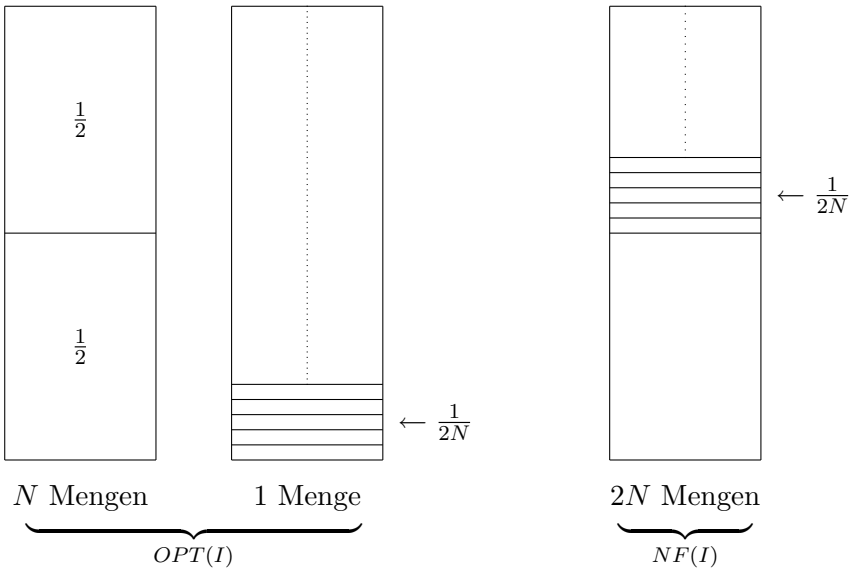
Die Laufzeit ist in $\mathcal{O}$(n).

Beispiel: Sei $n = 4 \cdot N$. Sei

$$s(a_i) = \begin{cases} \frac{1}{2} & i \text{ ungerade} \\ \frac{1}{2 \cdot N} & \text{sonst} \end{cases}$$

Next Fit benötigt $2 \cdot N$ Mengen B_j, während eine Optimallösung mit $N+1$ Mengen auskommt, das bedeutet

$$NF(I) = 2 \cdot \mathrm{OPT}(I) - 2.$$



■

Satz 3.1
NEXT FIT *erfüllt* $\mathcal{R}_{NF} = 2$.

Beweis: Sei I ein Beispiel für BIN PACKING mit NF(I) = k und $B_1, \ldots, B_k$ seien die benutzten Mengen. Sei für $1 \leq j \leq k$

$$s(B_j) = \sum_{a_i \in B_j} s(a_i)$$

Dann gilt folgende Ungleichung für $j = 1, \ldots, k-1$:

$$s(B_j) + s(B_{j+1}) > 1,$$

da ansonsten die Elemente aus B_{j+1} von NF in B_j eingefügt worden wären. Daraus folgt

$$\sum_{j=1}^{k} s(B_j) \quad > \quad \frac{k}{2} \qquad \text{falls } k \text{ gerade, beziehungsweise}$$

$$\sum_{j=1}^{k-1} s(B_j) \quad > \quad \frac{k-1}{2} \qquad \text{falls } k \text{ ungerade.}$$

Andererseits ist $\mathrm{OPT}(I) > \frac{k-1}{2}$, also $k = \mathrm{NF}(I) < 2 \cdot \mathrm{OPT}(I) + 1$. □

Bemerkung: 1
Für Beispiele mit $s(a_i) \leq s \leq \frac{1}{2}$ für $1 \leq i \leq n$ kann dieses Resultat noch verschärft werden. Für jede Menge B_j muss dann $s(B_j) > 1 - s$ sein, ausser

möglicherweise für B_k, $k = NF(I)$. Dann folgt:

$$OPT(I) \geq \sum_{j=1}^{k-1} s(B_j) > (k-1) \cdot (1-s),$$

$$\textit{also} \qquad k = NF(I) < \frac{1}{1-s} \cdot OPT(I) + 1.$$

Für $s \longrightarrow 0$ geht also $NF(I) \longrightarrow OPT(I) + 1$.

First Fit (FF)

Algorithmus:

1. Schritt: Durchlaufe M beginnend mit a_1 und füge a_1 in B_1 ein.

2. Schritt: Wenn a_ℓ eingefügt werden muss, durchlaufe alle nicht-leeren Mengen $B_1, \ldots, B_j$ und füge a_ℓ in die Menge B_r mit

$$r := \min \left\{ t \leq j+1 : \; s(a_l) \leq 1 - \sum_{a_i \in B_t} s(a_i) \right\}.$$

Die Laufzeit ist in $\mathcal{O}(n^2)$. Offensichtlich ist $\mathrm{FF}(I) < 2 \cdot \mathrm{OPT}(I)$ für alle Instanzen I, denn es kann nach dem Anwenden von First Fithöchstens eine Menge B_j geben mit $s(B_j) \leq \frac{1}{2}$. Also gilt:

$$\mathrm{FF}(I) < 2 \cdot \sum_{a_i \in M} s(a_i) = 2 \cdot \sum_j s(B_j)$$

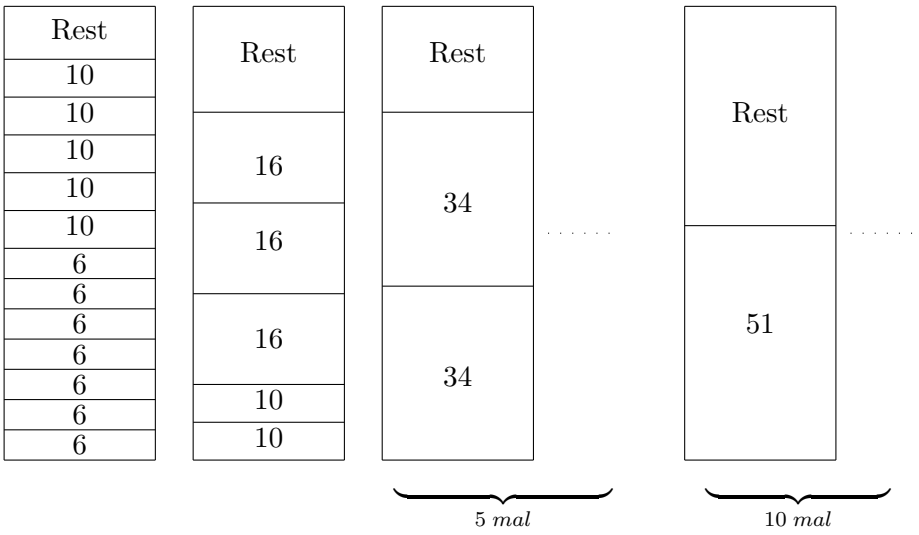
Mit $\mathrm{OPT}(I) \geq \sum_{a_i \in M} s(a_i)$ folgt die Behauptung.

Beispiel: Sei $B := 101$ „Grösse“ der Mengen (statt 1) und $n = 37$.

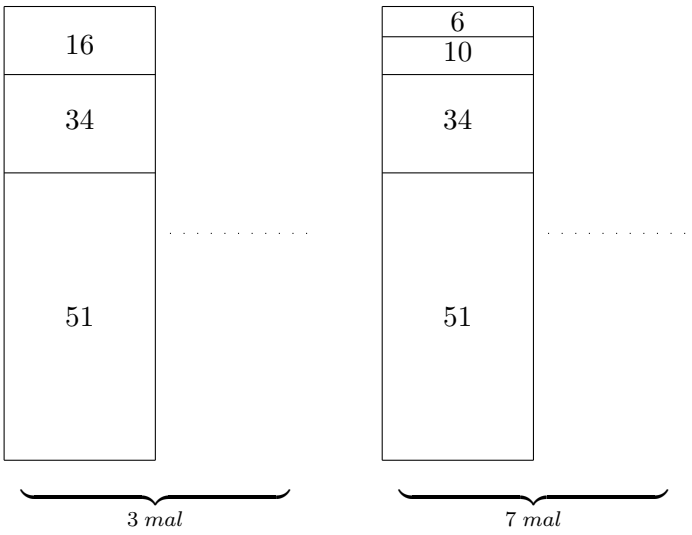
$$s(a_i) := \begin{cases} 6 & 1 \leq i \leq \;\; 7 \\ 10 & 8 \leq i \leq 14 \\ 16 & 15 \leq i \leq 17 \\ 34 & 18 \leq i \leq 27 \\ 51 & 28 \leq i \leq 37 \end{cases}$$

Dann ist FF(I) = 17 und OPT(I) = 10, d.h. $R_{FF}(I) = \frac{17}{10}$.

FF-Lösung:



Optimallösung:



■

Satz 3.2
Für jedes Beispiel I von Bin Packing *gilt:*

$$FF(I) < \frac{17}{10} \cdot OPT(I) + 1.$$

Beweisidee: Definiere eine Funktion

$$\omega : [0,1] \longrightarrow [0,1],$$

die man benutzt um das Verhältnis von FF(I) zu OPT(I) abzuschätzen, in dem man

$$\text{FF}(I) \quad \text{zu} \quad \omega(I) := \sum_{i=1}^{n} \omega(a_i)$$

beziehungsweise

$$\text{OPT}(I) \quad \text{zu} \quad \omega(I)$$

in Relation setzen. Sei $\omega(a)$ wie folgt definiert:

$$\omega(a) := \begin{cases} \frac{6}{5} \cdot a & \text{für } 0 \leq a < \frac{1}{6} \\ \frac{9}{5} \cdot a - \frac{1}{10} & \text{für } \frac{1}{6} \leq a < \frac{1}{3} \\ \frac{6}{5} \cdot a + \frac{1}{10} & \text{für } \frac{1}{3} \leq a \leq \frac{1}{2} \\ 1 & \text{für } \frac{1}{2} < a \leq 1 \end{cases}$$

Abkürzend benutzt man im folgenden $\omega(a_i)$ für $\omega(s(a_i))$.

Die Funktion ω erfüllt:

- Falls für $A \subseteq M$ gilt: $\sum_{a_i \in A} s(a_i) \leq 1$, dann gilt:

$$\omega(a_i) := \sum_{a_i \in A} \omega(a_i) \leq \frac{17}{10}$$

  Daraus lässt sich herleiten, dass

$$\sum_{a_i \in M} \omega(a_i) \leq \frac{17}{10} \cdot \mathrm{OPT}(I),$$

  d.h. $\mathrm{OPT}(I)$ kann man zu $\omega(I)$ in Relation setzen.

- Entsprechend verfährt man für $\mathrm{FF}(I)$ und erhält:

$$\sum_{a_i \in M} \omega(a_i) > FF(I) - 1.$$

□

Bemerkung: 2
Der Summand 1 ist bei dieser Abschätzung sicher vernachlässigbar. Definiere deshalb asymptotische Gütegarantie $\mathcal{R}_{\mathcal{A}}^{\infty}$*:*

$$\mathcal{R}_{\mathcal{A}}^{\infty} := \inf \left\{ r \geq 1 : \quad \begin{array}{l} \textit{Es gibt ein } N > 0, \textit{ so dass } R_{\mathcal{A}}(I) \leq r \\ \textit{für alle } I \textit{ mit } OPT(I) \geq N \end{array} \right\}$$

Dann ist $\mathcal{R}_{FF}^{\infty} = \frac{17}{10}$.
Zu weiteren Approximationsalgorithmen für Bin Packing *gibt es noch bessere asymptotische Gütegarantien als* $\frac{17}{10}$. *Als Beispiel sei hier* First Fit Decreasing *genannt mit einer asymptotischen Gütegarantie von* $\frac{11}{9}$.

4 Approximationsschemata

Kann es für $\mathcal{NP}$-schwere Optimierungsprobleme noch bessere Approximierbarkeitsresultate geben als Approximationsalgorithmen mit relativer Gütegarantie K, wobei K konstant ist ? Die Antwort auf diese Frage wird Gegenstand dieses Abschnitts sein.

Definition: 1
Ein (polynomiales) Approximationsschema (PAS) *für ein Optimierungsproblem* Π *ist eine Familie von Algorithmen* $\{\mathcal{A}_\varepsilon : \varepsilon > 0\}$, *so dass* $\mathcal{A}_\varepsilon$ *ein* ε*-approxmierender Algorithmus ist, d.h.* $R_{\mathcal{A}_\varepsilon} \leq 1 + \varepsilon$ *für alle* $\varepsilon > 0$. *Dabei bedeutet polynomial wie üblich polynomial in der Grösse des Inputs I. Ein Approximationsschema* $\{\mathcal{A}_\varepsilon : \varepsilon > 0\}$ *heisst* vollpolynomial (FPAS), *falls seine Laufzeit zu dem polynomial in* $\frac{1}{\varepsilon}$ *ist.*

Zunächst zeigen wir, dass für $\mathcal{NP}$-schwere Optimierungsprobleme ein FPAS in gewissem Sinne das beste ist, was wir erwarten können. Man kann beweisen, dass folgender Satz gilt:

Satz 4.1
Falls $\mathcal{P} \neq \mathcal{NP}$ ist und Π ein $\mathcal{NP}$-schweres Optimierungsproblem ist, so gibt es kein PAS $\{\mathcal{A}_\varepsilon : \varepsilon > 0\}$ für Π, bei dem die Laufzeit von $\mathcal{A}_\varepsilon$ zu dem polynomial in $\log \frac{1}{\varepsilon}$ (Kodierungslänge von $\frac{1}{\varepsilon}$) ist.

4.1 Ein PAS für Multiprocessor Scheduling

gegeben:

Seien n Jobs $J_1, \ldots, J_n$ mit Bearbeitungsdauer $p_1, \ldots, p_n$ und m identische Maschinen gegeben. Œ sei $m < n$.

gesucht:

Man möchte nun einen Schedule mit minimalem Makespan finden, d.h. eine Zuordnung der n Jobs auf die m Maschinen, bei der zu keinem Zeitpunkt zwei Jobs auf einer Maschine liegen und der Zeitpunkt, zu dem alle Jobs abgearbeitet sind, minimal ist.

$$Makespan := \max_{1 \leq j \leq m} \left(\sum_{J_i} p_i \right) \qquad \text{wobei } J_i \text{ auf Maschine j ist.}$$

Multiprocessor Scheduling ist $\mathcal{NP}$-vollständig (vgl. hierzu Übung 3).

Zunächst wollen wir aber das einfachere Verfahren List Scheduling betrachten, das die Gütegarantie $2 - \frac{1}{m}$ besitzt.

List Scheduling

Betrachte n Jobs $J_1, \ldots, J_n$ in beliebiger Reihenfolge als Liste angeordnet.

Algorithmus:

1. Schritt: Lege die ersten m Jobs auf die m Maschinen.

2. Schritt: Sobald eine Maschine frei ist, ordne ihr den nächsten der restlichen $n - m$ Jobs zu.

Dieses Verfahren hat eine Laufzeit von $\mathcal{O}$(n). Es kann auch angewendet werden, ohne dass alle Jobs zu Beginn bekannt sind. Ein Beispiel hierfür sind Online-Szenarien.

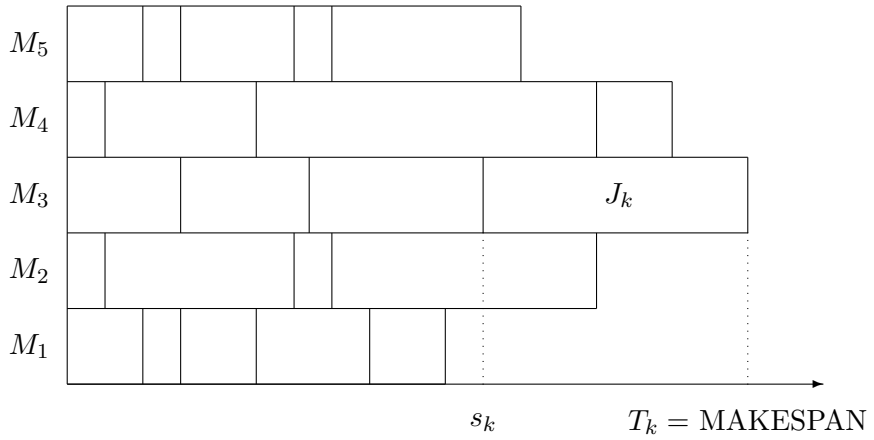
Satz 4.2
Für List Scheduling $\mathcal{A}$ gilt:

$$\mathcal{R}_\mathcal{A} = 2 - \frac{1}{m}$$

Beweis:

Bezeichne S_i die Startzeit von Job J_i und T_i die Abschlusszeit von Job J_i im durch $\mathcal{A}$ konstruierten Schedule. Falls J_k der zuletzt beendete Job ist, so ist $T_k = Makespan_{\mathcal{A}}$. Dann kann zu keinem Zeitpunkt vor S_k irgendeine Maschine untätig (idle) sein, d.h.

$$s_k \leq \frac{1}{m} \cdot \sum_{j \neq k} p_j$$



Sei T_{OPT} der optimale MAKESPAN. Für T_{OPT} gilt:

$$\begin{aligned} T_{\text{OPT}} &\geq p_k \quad \text{da } J_k \text{ ausgeführt werden muss, und ausserdem} \\ T_{\text{OPT}} &\geq \frac{1}{m} \cdot \sum_{j=1}^{m} p_j \end{aligned}$$

da diese untere Schranke die bestmögliche Auslastung der Maschinen repräsentiert. Andererseits gilt:

$$\begin{aligned} T_k &= S_k + p_k \\ &\leq \frac{1}{m} \cdot \sum_{j \neq k} p_j + p_k \\ &= \frac{1}{m} \cdot \sum_{j=1}^{m} p_j + \left(1 - \frac{1}{m}\right) \cdot p_k \\ &\leq T_{\text{OPT}} + \left(1 - \frac{1}{m}\right) \cdot T_{\text{OPT}} \\ &= \left(2 - \frac{1}{m}\right) \cdot T_{\text{OPT}} \end{aligned}$$

□

Basierend auf LIST SCHEDULING kann man nun PAS für MULTIPROCESSOR SCHEDULING angeben.

Algorithmus $\mathcal{A}_\ell$ für MULTIPROCESSOR SCHEDULING

mit n Jobs und ℓ konstant mit $1 \leq \ell \leq n$.

1. Schritt: Bestimme zunächst einen optimalen Schedule für $J_1, \ldots, J_\ell$. Dabei sollen $J_1, \ldots, J_\ell, J_{\ell+1}$ die $\ell + 1$ längsten Jobs sein mit den Bearbeitungszeiten $p_1 \geq p_2 \geq \cdots \geq p_\ell \geq p_{\ell+1}$.
2. Schritt: Ausgehend von diesem partiellen Schedule ordne den restlichen Jobs $J_{\ell+1}, \ldots, J_n$ Maschinen entsprechend LIST SCHEDULING zu.

$\mathcal{A}_\ell$ kann in polynomialer Laufzeit realisiert werden, da ℓ konstant angenommen wird in $\mathcal{O}(m^\ell + n)$.

Satz 4.3
Für $\mathcal{A}_\ell$ und $1 \leq \ell \leq n$ konstant, gilt:

$$R_{\mathcal{A}_\ell} \leq 1 + \frac{1 - \frac{1}{m}}{1 + \left\lfloor \frac{\ell}{m} \right\rfloor}$$

Beweis: Bezeichne T die Abschlusszeit des Schedules für $J_1, \ldots, J_\ell$ nach Schritt 1. Falls die Abschlusszeit des Schedules für $J_1, \ldots, J_n$ auch T ist, so ist $T = T_{\text{OPT}}$, d.h. $\mathcal{A}_\ell$ ist optimal. Sei also MAKESPAN$> T$ und J_i mit $i > \ell$ der zuletzt beendete Job, d.h. $Makespan_{\mathcal{A}_\ell} = T_i$, wobei T_i die Abschlusszeit von J_i ist. Im Zeitintervall $[0, T_i = p_i]$ muss dann jede Maschine belegt sein, da sonst J_i schon früher begonnen worden wäre. Es folgt:

$$\sum_{j=1}^{m} p_j \geq m \cdot (T_i - p_i) + p_i = m \cdot T_i - (m-1) \cdot p_i$$

Da $J_1, \ldots, J_\ell, J_{\ell+1}$ die $\ell + 1$ längsten Jobs sind, ist $p_i \leq p_{\ell+1}$, d.h. es gilt:

$$\begin{aligned} \sum_{j=1}^{m} p_j &\geq m \cdot (T_i - p_i) + p_i \\ &= m \cdot T_i - (m-1) \cdot p_i \\ &\geq m \cdot T_i - (m-1) \cdot p_{\ell+1} \end{aligned}$$

Da

$$\begin{aligned} T_{\text{OPT}} &\geq \frac{1}{m} \cdot \sum_{j=1}^{n} p_j \qquad \text{ist, gilt:} \\ T_i &\leq T_{\text{OPT}} + \frac{m-1}{m} \cdot p_{\ell+1} \ . \end{aligned}$$

Anderseits gilt:

$$T_{\text{OPT}} \quad \geq \quad p_{\ell+1} \cdot \left(1 + \frac{\ell}{\lfloor m \rfloor}\right),$$

denn in einem optimalen Schedule muss eine Maschine mindestens $1 + \frac{\ell}{\lfloor m \rfloor}$ der Jobs $J_1, \ldots, J_\ell, J_{\ell+1}$ bearbeiten und ausserdem ist

$$p_j \geq p_{\ell+1} \qquad \text{für } 1 \leq j \leq \ell.$$

Also ist

$$p_{\ell+1} \leq \frac{T_{\text{OPT}}}{1 + \frac{\ell}{\lfloor m \rfloor}} \quad \text{und damit folgt:}$$

$$\begin{aligned} R_{\mathcal{A}_\ell} = \frac{T_i}{T_{\text{OPT}}} \quad &\leq \quad 1 + \frac{1}{T_{\text{OPT}}} \cdot \left(1 - \frac{1}{m}\right) \cdot p_{l+1} \\ &\leq \quad 1 + \left(1 - \frac{1}{m}\right) \cdot \frac{1}{1 + \frac{\ell}{\lfloor m \rfloor}} \end{aligned}$$

□

Aus $\mathcal{A}_\ell$ kann nun ein PAS abgeleitet werden. Zu $\varepsilon > 0$ sei $\mathcal{A}_\varepsilon$ ein Algorithmus $\mathcal{A}_\ell$ mit ℓ so gewählt, dass $R_{\mathcal{A}_\ell} \leq 1 + \varepsilon$ ist, also

$$\frac{1 - \frac{1}{m}}{1 + \frac{\ell}{\lfloor m \rfloor}} \leq \varepsilon \quad \text{ist.}$$

Folgerung 4.1
Für Multiprocessor Scheduling *mit konstanter Maschinenanzahl existiert ein PAS.*

Bemerkung: 3
$\{\mathcal{A}_\ell : \ell > 0\}$ *ist* kein *FPAS, da die Laufzeit von* $\mathcal{A}_\ell$ *in* $\mathcal{O}(m^\ell + n)$ *nicht polynomial in* $\frac{1}{\varepsilon}$ *ist.*

Asymptotische PAS

Definition: 2
Ein asymptotisches PAS - *im folgenden mit* APAS *abgekürzt - ist eine Familie von Algorithmen* $\{\mathcal{A}_\varepsilon : \varepsilon > 0\}$, *so dass* $\mathcal{A}_\varepsilon$ *asymptotisch* ε*-approximativer Algorithmus für jedes* $\varepsilon > 0$ *ist, d.h.*

$$\mathcal{R}^\infty_{\mathcal{A}_\varepsilon} \leq 1 + \varepsilon.$$

Entsprechend nennt man $\{\mathcal{A}_\varepsilon : \varepsilon > 0\}$ *ein* asymptotisch vollpolynomiales PAS (AFPAS), *wenn* $\mathcal{A}_\varepsilon$ *zudem polynomial in* $\frac{1}{\varepsilon}$ *ist.*

Ein APAS für Bin Packing

Ziel dieses Abschnittes ist es ein APAS $\{\mathcal{A}_\varepsilon : \varepsilon > 0\}$ mit

$$\mathcal{A}_\varepsilon(I) \leq (1+\varepsilon) \cdot \mathrm{OPT}(I) + 1$$

für alle Instanzen I mit Laufzeit $\mathcal{O}(n)$ zu entwickeln. Dabei sei n die Anzahl der zu packenden Elemente, die exponentiell in $\frac{1}{\varepsilon}$ ist.

Eine Instanz I für Bin Packing bestehe also aus den Elementen $\{1, \ldots, n\}$ mit den Grössen $s_1, \ldots, s_n$, wobei $0 < s_i \leq n$. Um das Problem zu lösen,

benutzen wir folgende **Techniken**:

1. Restriktion von Bin Packing
2. Entfernen kleiner Elemente
3. lineares Gruppieren

Erinnerung 1
Es gilt offensichtlich für jede Instanz I von Bin Packing*:*

$$\begin{aligned} SIZE(I) := \sum_{i=1}^{n} s_i &\leq OPT(I) \leq n \quad \textit{und} \\ OPT(I) &\leq 2 \cdot SIZE(I) + 1 \end{aligned}$$

da FIRST FIT *immer eine solche Lösung findet. Œ sei $s_1 \geq \cdots \geq s_n \geq 0$.*

Restricted Bin Packing **RBP** $[\delta, m]$

Gegeben seien Grössen $V = \{v_1, \ldots, v_m\}$ mit $m < n$ derart, so dass für diese gilt: $1 \geq v_1 > v_2 > \cdots > v_m \geq \delta > 0$. Eine Instanz von RBP $[\delta, m]$ besteht aus den Elementen $\{1, \ldots, n\}$, deren Grössen s_i alle aus V sind und jeweils n_j Elemente der Grösse v_j vorkommen, d.h. $n = \sum_{j=1}^{m} n_j$.

Gesucht ist eine Partition von $\{1, \ldots, n\}$ mit minimaler Anzahl von Mengen B_ℓ, so dass für alle B_ℓ gilt:

$$\sum_{j \in B_\ell} s_j \leq 1$$

Betrachte nun eine Lösung für eine Instanz I von RBP $[\delta, m]$. Ein „BIN“ B der Lösung ist dann charakterisiert durch ein m - Tupel $(b_1, \ldots, b_m)$, wobei $0 \leq b_j \leq n_j$ mit der Bedeutung „in B sind b_j Elemente der Grösse v_j“ für $1 \leq j \leq m$. Ein m - Tupel $T = (T_1, \ldots, T_m)$ heisst *BIN-TYP*, falls $T_j \in \mathbb{N}_0$ und

$$\sum_{j=1}^{m} T_j \cdot v_j \leq 1.$$

Für eine feste Menge $V = \{v_1, \ldots, v_m\}$ und $0 < \delta < 1$ bezeichnet g die Anzahl möglicher, verschiedener *BIN-TYPEN*. Dann kann eine obere Schranke für g angegeben werden, die nur von g und m abhängt, bezeichnet mit $g(\delta,$ m).

Lemma 4.1
Sei $k = \frac{1}{\lfloor \delta \rfloor}$. Dann gilt:

$$g(\delta, m) \leq \binom{m+k}{k}$$

Beweis: Ein *BIN-TYP* $(T_1, \ldots, T_m)$ hat die Eigenschaft, dass

$$\sum_{j=1}^{m} T_j \cdot v_j \leq 1 \text{ und } T_j \geq 0 \text{ für } 1 \leq j \leq m.$$

Da $v_j \geq \delta$ für alle j, folgt:

$$\sum_{j=1}^{m} T_j \leq k$$

Jeder *BIN-TYP* entspricht einer Möglichkeit (geordnet) m, nicht-negative ganze Zahlen zu wählen, die sich zu höchstens k aufsummieren beziehungsweise $m + 1$ nicht-negative ganze Zahlen zu wählen, die sich genau zu k aufsummieren lassen. Die Anzahl dieser Möglichkeiten ist also die obere Schranke für $g(\delta, \mathrm{m})$. Dass diese Anzahl gerade

$$\binom{m+k}{k}$$

ist, lässt sich per Induktion mit einem einfachen Abzählargument beweisen. □

Eine Lösung einer Instanz I von RBP $[\delta, m]$ kann also allein dadurch charakterisiert werden, wieviele BINs eines jeden der $g(\delta, \mathrm{m}) = g$ *BIN-TYPEN* vorkommen, d.h. durch ein g - Tupel $\mathrm{X} = (x_1, \ldots, x_g)$, wobei x_t die Anzahl der BINs vom *BIN-TYP* T^t angibt für $1 \leq t \leq g$.

Beachte:

Nicht alle *BIN-TYPEN* gehören zu einer zulässigen Lösung, da die Zulässigkeit für alle $j \in \{1, \ldots, m\}$ erfordert:

$$\sum_{t=1}^{g} x_t \cdot T_j^t \leq n_j$$

Beispiel 4.1
Gegeben sei folgende Instanz I mit $n = 20$, $m = 5$ und $\delta = \frac{1}{8}$. Ferner sei

$v_1 = 1$	$n_1 = 3$
$v_2 = 3$	$n_2 = 3$
$v_3 = \frac{1}{2}$	$n_3 = 6$
$v_4 = \frac{1}{4}$	$n_4 = 3$
$v_5 = \frac{1}{8}$	$n_5 = 5$

Ausserdem hat man

$$\begin{array}{lllllll} T^1 & = & (1,0,0,0,0), & & T^2 & = & (0,1,0,0,0), \\ \ldots, & & (0,1,0,1,0), & \ldots, & T^t & = & (0,0,1,0,4), \\ \ldots, & & (0,0,0,2,4), & \ldots & & & \end{array}$$

als BIN-Typen. Es gibt maximal

$$\binom{5+8}{5} \geq g \text{ BIN-Typen.}$$

Eine Lösung ist $X = (x_1, \ldots, x_g)$*, beispielsweise* $(3, 3, \ldots, 0, \ldots, 1, \ldots,)$*. Ein g - Tupel X, bei dem zum Beispiel* T^1 *mindestens viermal oder* T^t *mindestens zweimal vorkommt, ist keine zulässige Lösung.*

Lineares Programm zu RBP $[\delta, m]$

Sei A eine $q \times m$ - Matrix, deren t-te Zeile dem m - Tupel T^t entspricht und sei N := $(n_1, \ldots, n_m)$. Dann ist

$$\sum_{t=1}^{q} x_t \cdot T_j^t = n_j \qquad \text{äquivalent zu} \qquad X \cdot A \leq N$$

für alle $j \in \{1, \ldots, \text{m}\}$. Die Anzahl der BINs in einer Lösung X ist einfach:

$$\sum_{t=1}^{q} x_t = (1, \ldots, 1) \cdot x^{transponiert}$$

Folgerung 4.2
Eine optimale Lösung von RBP $[\delta, m]$ *entspricht einer ganzzahligen Lösung des folgenden „Integer linear Program“ (ILP(I)):*

Minimiere

$$1^{transponiert} \cdot X^{transponiert}$$

unter der Bedingung $x_i \geq 0$ *für* $1 \leq i \leq q$ *und*

$$x \cdot A = N.$$

Die Anzahl der q an Zeilen von A ist exponentiell in δ und m. Wenn man jedoch voraussetzt, dass δ und m konstant sind, so hängt die Grösse des $ILP(I)$ nur von n ab und das $ILP(I)$ kann in einer Zeit, die linear in n ist, aufgestellt werden. Wir haben in Kapitel 2 gezeigt, dass „Integer Linear Programming“ $\mathcal{NP}$-vollständig ist. Ein ILP, bei dem die Anzahl der Variablen konstant ist, kann jedoch in einer Laufzeit, die linear in der Anzahl der Nebenbedingungen ist, gelöst werden. Dieses Ergebnis stammt von H. W. Lenstra und ist aus dem Jahr 1983. Die Anzahl der Variablen hier ist q und nur abhängig von δ und m.

Folgerung 4.3
Eine Instanz I von RBP $[\delta, m]$ *kann in* $\mathcal{O}(n + f(\delta, m))$ *gelöst werden, wobei* $f(\delta, m)$ *eine Konstante ist, die unabhängig von* δ *und* m *ist.*

Entfernen kleiner Elemente

Lemma 4.2
Sei I eine Instanz für Bin Packing. *Zu δ mit $0 < \delta \leq \frac{1}{2}$ sei eine Teillösung für I gegeben, bei der alle Elemente der Grösse $s_i > \delta$ in β BINs gepackt werden können. Dann kann diese Teillösung erweitert werden zu einer Lösung von I, bei der die Anzahl der BINs höchstens*

$$\max\{\beta, (1 + 2 \cdot \delta) \cdot OPT(I) + 1\} \text{ ist.}$$

Beweis: Um das Lemma zu beweisen benutzt man First Fit um die Teillösung zu einer Lösung zu erweitern, wobei zunächst immer wieder die β BINs der Teillösung behandelt werden. Wenn First Fit alle „kleineren" Elemente in die ersten β BINs packt, dann ist die Behauptung erfüllt. Bestehe also die Lösung aus $\beta' > \beta$ BINs. Wie bei der Abschätzung, dass allgemein

$$FF \leq 2 \cdot \mathrm{OPT} + 1$$

gilt, sind hier alle bis auf höchstens ein BIN mindestens zu $1 - \delta$ gefüllt. Also gilt nun:

$$SIZE(I) \quad \geq \quad (1 - \delta) \cdot (\beta' - 1)$$

Da

$$\begin{aligned} SIZE(I) &\leq \mathrm{OPT}(I) \qquad \text{folgt nun:} \\ \beta' &\leq \frac{1}{1-\delta} \cdot \mathrm{OPT}(I) + 1 \\ &\leq \frac{1 + \delta - 2 \cdot \delta^2}{1 - \delta} \cdot \mathrm{OPT}(I) + 1 \\ &= (1 + 2 \cdot \delta) \cdot \mathrm{OPT}(I) + 1. \end{aligned}$$

□

Lineares Gruppieren

Eine Instanz I von Bin Packing wird in eine Instanz von RBP $[\delta, m]$ überführt für ein geeignetes δ und m, ohne dass sich der Wert einer optimalen Lösung „zu sehr" verändert. Zu einer Instanz I von Bin Packing und zu k sei $m := \frac{n}{\lfloor k \rfloor}$. Definiere ferner Gruppen G_j, $j \in \{1, \ldots, m+1\}$, von Elementen der Instanz I, so dass G_1 die k grössten Elemente, G_2 die k nächstgrössten, usw. enthält, d.h.

$$\begin{aligned} G_j &:= \{(j-1) \cdot k, \ldots, j \cdot k\} \qquad \text{für } j = 1, \ldots, m \text{ und} \\ G_{m+1} &:= \{m \cdot k + 1, \ldots, n\} \end{aligned}$$

wobei die Grösse von Element i s_i sei mit $1 \geq s_1 \geq s_2 \geq \cdots \geq s_n \geq \delta > 0$.

Definition: 3
Man definiert zu zwei Instanzen I_1 und I_2 von Bin Packing *mit*

$$\begin{array}{ll} I_1 & \text{enthält Elemente der Grösse} \quad x_1 \geq x_2 \geq \cdots \geq x_n \\ I_2 & \text{enthält Elemente der Grösse} \quad y_1 \geq y_2 \geq \cdots \geq y_n \end{array}$$

die Relation „$\geq$“ durch

$$I_1 \geq I_2 \Longleftrightarrow x_i \geq y_i \qquad \text{für alle } i \in \{1, \ldots, n\}.$$

Man sagt dann „I_1 dominiert I_2“.

Folgerung 4.4
Falls $I_1 \geq I_2$, so gilt

$$\begin{array}{rcll} SIZE(I_1) & \geq & SIZE(I_2) & \text{und} \\ OPT(I_1) & \geq & OPT(I_2). & \end{array}$$

Nun gilt offensichtlich:

$$G_1 \geq G_2 \geq G_3 \geq \cdots > G_m$$

Sei nun $v_j := s_{(j-1)\cdot k+1}$ Grösse des grössten Elementes in G_j. Definiere Gruppen H_j, $j \in \{1, \ldots, m\}$ bestehend aus jeweils k Elementen der Grösse v_j und H_{m+1} bestehend aus $|G_{m+1}|$ Elementen der Grösse V_{m+1}. Dann gilt:

$$H_1 \geq G_1 \geq H_2 \geq G_2 \geq \cdots \geq H_m \geq G_m$$

Definiere nun zu einer Instanz I von Bin Packing zwei Instanzen

$$\begin{array}{lll} I_{LO} \text{ für „I Low“} & \text{aus} & H_2 \cup H_3 \cup \cdots \cup H_{m+1} \quad \text{(m Elemente!)} \\ I_{HI} \text{ für „I High“} & \text{aus} & H_1 \cup H_2 \cup \cdots \cup H_{m+1} \quad \text{(m+1 Elemente!)} \end{array}$$

Dann ist I_{LO} eine Instanz von RPB $[\delta, m]$ und $I_{HI} \geq I$.

Lemma 4.3
Es gilt:

$$\begin{array}{rcccccl} OPT(I_{LO}) & \leq & OPT(I) & \leq & OPT(I_{HI}) & \leq & OPT(I_{LO}) + k \quad \text{und} \\ SIZE(I_{LO}) & \leq & SIZE(I) & \leq & SIZE(I_{HI}) & \leq & SIZE(I_{LO}) + k \end{array}$$

Beweis: Betrachte Instanz I_x bestehend aus $G_1 \cup G_2 \cup \cdots \cup G_{m-1} \cup X$, wobei $X \subseteq G_m$ mit $|X| = |G_{m+1}| = |H_{m+1}|$. Dann gilt: $I_{LO} \leq I_x$ und daraus folgt:

$$\mathrm{OPT}(I_{LO}) \leq \mathrm{OPT}(I) \qquad \text{und} \qquad SIZE(I_{LO}) \leq SIZE(I)$$

I_{HI} entsteht aus I_{LO} durch Hinzufügen von H_1. Aus einer Lösung von I_{LO} entsteht durch Hinzufügen von maximal k zusätzlichen BINs eine Lösung von I_{HI}, also gilt:

$$\begin{array}{rcll} \mathrm{OPT}(I_{HI}) & \leq & \mathrm{OPT}(I_{LO}) + k & \text{und trivialerweise gilt:} \\ SIZE(I_{HI}) & \leq & SIZE(I_{LO}) + k & \end{array}$$

Die nun noch fehlende Ungeleichung folgt aus $I \leq I_{HI}$. □

Mittels Sortieren kann aus einer beliebigen Instanz I für Bin Packing in $\mathcal{O}(n \log(n))$ I_{LO} und I_H konstruiert werden und aus einer optimalen Lösung für I_{LO} eine Lösung für I mit dem Wert $\mathrm{OPT}(I_{LO}) + k$.

APAS $\{\mathcal{A}_\varepsilon : \varepsilon > 0\}$ für Bin Packing

Nach der geleisteten Vorarbeit kann man nun das vollständige APAS für Bin Packing formulieren:

Algorithmus:

Input: Sei I eine Instanz mit n Elementen der Grössen $s_1 \geq \cdots \geq s_n$ und ε.

1. Setze $\delta := \frac{\varepsilon}{2}$.
2. Betrachte Instanz J der Elemente aus I mit Grösse mindestens δ. J ist dann eine Instanz von RBP $[\delta, m]$.
3. Setze $k := \left\lceil \frac{\varepsilon^2}{2} \cdot n' \right\rceil$.
4. Berechne zu J und k Instanz J_{LO} von RBP $[\delta, m]$ und J_{HI} bestehend aus J_{LO} mit H_1, wobei $|H_1| = k$ ist und für $m := \frac{n'}{\lfloor k \rfloor}$ benutzt wird.
5. Berechne jetzt eine optimale Lösung von J_{LO} durch optimales Lösen des $\mathrm{ILP}(J_{LO})$.
6. Füge die k Elemente aus H_1 in maximal k zusätzliche BINs.
7. Berechne aus dieser Lösung für J_{HI} Lösung für J mit derselben Anzahl BINs.
8. Erweitere diese Lösung von J mit First Fit zu einer Lösung von I.

Satz 4.4
Für obigen Algorithmus gilt:

$$\mathcal{A}_\varepsilon(I) \leq (1 + \varepsilon) \cdot OPT(I) + 1 \qquad \textit{und}$$

die Laufzeit von $\mathcal{A}_\varepsilon$ ist in $\mathcal{O}(n \cdot \log(n) + c_\varepsilon)$, wobei c_ε eine „Konstante“ ist, die von ε abhängt.

Beweis: Die Laufzeit hängt nur vom Sortieren in Schritt 4 ab und von dem Lösen des $\mathrm{ILP}(J_{LO})$ in Schritt 5. Die Laufzeit von $\mathrm{ILP}(J_{LO})$ ist linear in n und einer von ε abhängigen Konstanten. Die Lösung für J benötigt maximal $\mathrm{OPT}(J_{LO}) + k$ viele BINs. Da alle Elemente in J mindestens Grösse $\delta = \frac{\varepsilon}{2}$ haben, muss $\mathrm{SIZE}(I) \geq \varepsilon \cdot \frac{n'}{2}$ sein. Also gilt:

$$k \leq \frac{\varepsilon^2}{2} \cdot n' + 1 \leq \varepsilon \cdot SIZE(J) + 1 \leq \varepsilon \cdot \mathrm{OPT}(J) + 1$$

Mit Lemma ... folgt dann:

$$\mathrm{OPT}(J_{LO}) + k \leq \mathrm{OPT}(J) + \varepsilon \cdot \mathrm{OPT}(J) + 1 \leq (1+\varepsilon) \cdot \mathrm{OPT}(J) + 1$$

und mit Lemma ... folgt weiter:

$$\begin{array}{rcll} \mathcal{A}_\varepsilon(I) & \leq & \max\{(1+\varepsilon) \cdot \mathrm{OPT}(J) + 1, (1+\varepsilon) \cdot \mathrm{OPT}(I) + 1\} & \text{und da} \\ \mathrm{OPT}(J) & \leq & OPT(I) & \text{folgt weiter:} \\ \mathcal{A}_\varepsilon(I) & \leq & (1+\varepsilon) \cdot \mathrm{OPT}(I) + 1. & \end{array}$$

□

Bemerkung: 4
Für die Laufzeit von $\mathcal{A}_\varepsilon$ ist der Aufwand für das Lösen des ILP(J_{LO}) in Schritt 5 entscheidend. Die Laufzeit ist exponentiell in $\frac{1}{\varepsilon}$, da die Anzahl der Nebenbedingungen des ILP exponentiell in $\frac{1}{\varepsilon}$ ist.

AFPAS für Bin Packing

Aus dem konstruierten APAS kann ein AFPAS konstruiert werden, indem optimales Lösen des $ILP(J_{LO})$ ersetzt wird durch eine „nicht-ganzzahlige“ Lösung und darauf die Technik des „Rundens“ angewendet wird.

Runden nicht-ganzzahliger Lösungen

Man betrachtet wieder das Problem RBP $[\delta, m]$. Dieses Problem kann formuliert werden als:

$$\text{Minimiere} \quad 1 \cdot X^{transponiert}$$

unter den Bedingungen

- $X_i \geq 0 \qquad$ für alle $1 \leq i \leq q$ und
- $X \cdot A = N$,

wobei A eine $q \times m$ - Matrix ist und N ein m - Vektor. Man betrachtet nun die *Relaxierung* des $ILPs$, in dem man die Ganzzahligkeitsbedingung für die x_i weglässt. Man erhält dadurch ein *Lineares Programm* (LP) zu dem ILP. Bezeichne nun im folgenden $LIN(I)$ den Wert einer (nicht notwendig ganzzahligen) optimalen Lösung des $LP(I)$.

Interpretation:

Eine Lösung einer Bin Packing-Instanz I von RBP $[\delta, m]$, die zu einer Lösung des $LP(I)$ korrespondiert, wäre eine Lösung, in der *Bruchteile von*

Elementen in *Bruchteile von BINs* gepackt werden. Die Grösse einer solchen optimalen Lösung ist gerade $SIZE(I)$. Allerdings würden die Bedingungen des LP keine beliebigen Bruchteile zulassen, sondern in jedem Bruchteil eines BINs müssten die Elemente denselben Bruchteil haben, d.h. anstatt ganzzahliger Anzahlen von *BIN-TYPEN* sind „gebrochenzahlige“ Anzahlen erlaubt.

Relaxierung des ILP

Aus der Theorie der linearen Programmierung übernehmen wir hier den Zusammenhang zwischen LPs und $ILPs$. Œ sei dazu $rang(A) = m$ und die ersten m Zeilen von A bilden eine Basis. Dann heisse die Lösung X^* mit $x_i = 0$ für $i > m$ *Basislösung.*

Es gilt:

Für jedes LP gibt es eine optimale Lösung, die eine Basislösung ist (siehe hierzu die Theorie der Linearen Programmierung).

Im folgenden bezeichne *LIN(I)* den Wert einer nicht notwendig ganzzahligen, optimalen Lösung des LPs zu einer Instanz I.

Lemma 4.4
Für alle Instanzen I zu RBP $[\delta, m]$ gilt:

$$SIZE(I) \leq LIN(I) \leq OPT(I) \leq LIN(I) + \frac{m+1}{2}.$$

Beweis: Die ersten beiden Ungleichungen sind klar. Sei nun Y eine Basislösung des $LP(I)$. Dann benutzt Y höchstens m verschiedene *BIN-TYPEN*. Würde man jede Komponente von Y zum nächsten ganzzahligen Wert aufrunden, so würde der Wert der Lösung also um höchstens m erhöht. Wir wollen hier allerdings eine schärfere Schranke (Summand $\frac{m+1}{2}$) beweisen. Dazu definiert man

$$\begin{array}{lcll} q\text{-Vektor} & W & \text{durch} & w_i := \lfloor Y_i \rfloor \qquad \text{und} \\ & Z & \text{durch} & z_i := Y_i - w_i \end{array}$$

mit $1 \leq i \leq q$, d.h. W ist ganzzahliger und Z ist gebrochenzahliger Rest von Y. Sei J Instanz von RBP $[\delta, m]$, die aus den Elementen von I besteht, die nicht in der ganzzahligen Teillösung, die durch W gegeben ist, in BINs gepackt werden. Z induziert dann eine gebrochenzahlige Lösung für J, die höchstens m *BIN-TYPEN* benutzt, die jeweils zu einem Bruchteil, der kleiner als 1 ist, benutzt werden. Also gilt:

$$SIZE(J) \leq LIN(J) \leq \sum_{i=1}^{n} z_i \leq m$$

Es gilt wiederum

$$\begin{aligned} \mathrm{OPT}(J) &\leq 2 \cdot SIZE(J) + 1 \quad \text{und offensichtlich ist wieder} \\ \mathrm{OPT}(J) &\leq m, \end{aligned}$$

da Aufrunden jedes positiven Eintrags in Z auf ein 1 eine Lösung mit Wert m liefert. Damit ergibt sich

$$\begin{aligned} \mathrm{OPT}(J) &\leq \min\{m, 2 \cdot SIZE(J) + 1\} \\ &\leq SIZE(J) + \min\{m - SIZE(J), 2 \cdot SIZE(J) + 1\} \\ &\leq SIZE(J) + \frac{m+1}{2} \end{aligned}$$

Man muss allerdings $\mathrm{OPT}(I)$ durch $SIZE(I)$ abschätzen und m beschränken. Es gilt allerdings

$$\begin{aligned} \mathrm{OPT}(I) &\leq \mathrm{OPT}(I-J) + \mathrm{OPT}(J) \\ &\leq \textstyle\sum_{i=1}^{m} w_i + SIZE(J) + \frac{m+1}{2} \\ &\leq \textstyle\sum_{i=1}^{m} w_i + LIN(J) + \frac{m+1}{2} \\ &\leq \textstyle\sum_{i=1}^{m} w_i + \sum_{i=1}^{m} z_i + \frac{m+1}{2} \\ &= LIN(I) + \frac{m+1}{2} \end{aligned}$$

□

Bemerkung: 5
Der Beweis zu obigem Lemma ist konstruktiv. Aus einer Lösung des $LP(I)$ kann man eine Lösung zu I konstruieren mit obiger Schranke. Die Anzahl der Bedingungen in $LP(I)$ ist wieder exponentiell in $\frac{1}{\varepsilon}$. Man kann aber folgenden Satz beweisen:

Satz 4.5
Es gibt einen vollpolynomialen Algorithmus $\mathcal{A}$ zur Lösung einer Instanz von RBP $[\delta, m]$ mit

$$\mathcal{A}(I) \leq LIN(I) + \frac{m+1}{2} + 1.$$

Dieses Verfahren stammt von KARMAKAR *und* KARP *aus dem Jahre 1982.*

Lemma 4.5
Mit „linearem Gruppieren“ kann zu einer Instanz I von RBP $[\delta, m]$ eine Instanz I_{LO} von RBP $[\delta, m]$ und eine Gruppe H_1 konstruiert werden, so dass für $I_{HI} = H_1 \cup I_{LO}$ gilt:

$$\begin{aligned} LIN(I_{LO}) &\leq LIN(I) \\ &\leq LIN(I_{HI} \\ &\leq LIN(I_{LO}) + k \end{aligned}$$

Der Beweis zu diesem Lemma verläuft analog zum vorhergehenden Lemma.

Ergebnis:

Man hat so ein AFPAS $\{\mathcal{A}_\varepsilon : 1 \geq \varepsilon > 0\}$ für eine Eingabe mit n Elementen mit den Grössen $\{s_1, \ldots, s_n\}$ erhalten. Ersetze nun im APAS Schritt 5 durch Schritt 5'. Löse dann J_{LO} entsprechend dem letzten Satz.

Satz 4.6
Es gilt die folgende Abschätzung:

$$\mathcal{A}_\varepsilon \leq (1+\varepsilon) \cdot OPT(I) + \frac{1}{\varepsilon^2} + 3$$

Beweis: Es gilt:

$$LIN(J_{LO}) + 1 + \frac{m+1}{2} \leq \mathrm{OPT}(I) + \frac{1}{\varepsilon^2} + 2$$

nach der Wahl von m mit

$$m = \frac{n'}{\lfloor k \rfloor} = \frac{n'}{\frac{\varepsilon^2}{2} \cdot n'} = \frac{2}{\varepsilon^2}.$$

Andererseits ist

$$\begin{aligned} OPT(J) &\geq SIZE(J) \geq n' \cdot \frac{\varepsilon}{2} \quad \text{und damit ist} \\ k = \frac{\lceil n' \cdot \varepsilon^2 \rceil}{2} &\leq 2 \cdot \mathrm{OPT}(J) + 1 \leq \varepsilon \cdot \mathrm{OPT}(I) + 1. \end{aligned}$$

Nun ist aber

$$\begin{aligned} \mathcal{A}_\varepsilon &\leq LIN(J_0) + 1 + \frac{m+1}{2} + k \quad \text{und daraus folgt} \\ \mathcal{A}_\varepsilon &\leq (1+\varepsilon) \cdot \mathrm{OPT}(I) + \frac{1}{\varepsilon^2} + 3. \end{aligned}$$

□