Technische Universität Braunschweig
Institut für Betriebssysteme und Rechnerverbund
Postfach 3329
D-38023 Braunschweig

SS 08

Notizen

Kompaktkurs Diskrete Optimierung

Henrik Peters

Bearbeitungsstand: 17. Juni 2008

Dozent: Prof. Dr. Sándor P. Fekete

Alle Angaben ohne Gewähr. Will sagen: Es können sich durchaus Unvollständigkeiten und/oder Fehler eingeschlichen haben!

1 Packungsprobleme allgemein

1.1 Vorspann

Gegeben:

- eine Menge von Objekten, evtl. mit Kosten und Volumen, Ausdehnung, etc.
- ein oder mehrere Container

Gesucht: eine «möglichst gute» Packung, d.h.

- eine Möglichkeit, alle Objekte im Container unterzubringen
  → *Zulässigkeit*
- eine Auswahl möglichst wertvoller Objekte, die zusammen in den Container passen
  → *Rucksack*
- eine Packung aller Objekte in möglichst wenige Container
  → *Umzugskartons*
- andere Varianten → *später*

Beispiele:

- Umzugsgüter in Kartons
- Koffer in Kofferraum
- Lieder auf CD
- Zuhörer im Hörsaal (während einer Klausur?)
- Veranstaltungen im Zeitplan
- Geschäftsstandorte in Innenstadt
- Chemiefabriken in Industriezonen

Unterscheidung:

- mehrdimensionales Packen (Koffer!)
- eindimensionales Packen (bspw. Gewicht, Zeit, Kosten, etc.)
- abstraktes Packen

Beispiel für abstraktere Situation

Gegeben:

- Menge von Personen P
- Konfliktbeziehungen: $\{v, w\}$ sind zerstritten und sollten nicht zur selben Party eingeladen werden
  → Menge von Personen $K = e_1, \ldots, e_m \rightarrow \{v_i, v_j\}$

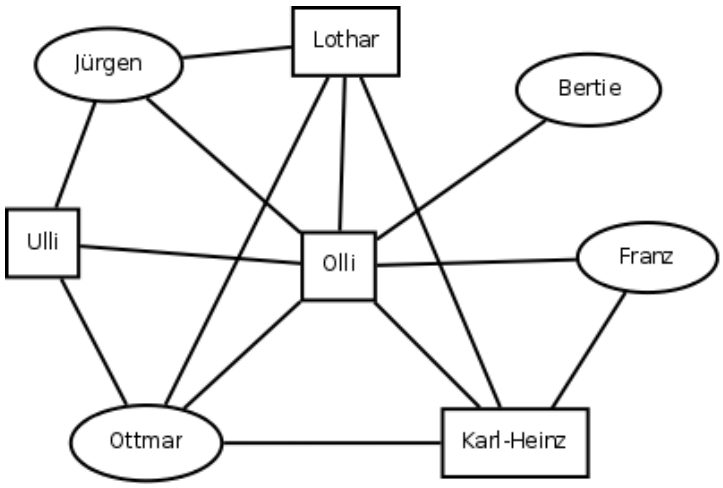

Abbildung 1: Konfliktbeziehungen

Gesucht: eine möglichst große Menge von Personen, die man stressfrei auf dieselbe Party einladen kann

Formaler:

Problem 1.1: (Unabhängige Menge)

Gegeben: ein Graph $G = (V, E)$

Gesucht: eine Menge $S \subset V$ von Knoten, so dass für alle $v, w \in S$ gilt $\{v, w\} \notin E$ und möglichst groß

1.2 «Partition» und Komplexität

Problem 1.2: (Entscheidungsproblem in 1D - Variante 1)

Gegeben: n Objekte $g_1, \ldots, g_n$, jeweils mit «Größe» $l_1, \ldots, l_n$, ein Container L

Frage: Passen die Objekte in den Container?

Beobachtung 1.3:

Problem 1.2 sehr einfach lösbar! Man prüft, ob

$$\sum_{i=1}^{n} l_i \leq L$$

Problem 1.4: (Entscheidungsproblem in 1D - Variante 2)

Gegeben: n Objekte $g_1, \ldots, g_n$, jeweils mit «Größe» $l_1, \ldots, l_n$, zwei Container, jeweils mit Größe L

Frage: Passen die Objekte in die Container?

Beispiel 1.5:

9 Objekte mit den Größen $\{29, 13, 17, 31, 35, 24, 31, 7, 53\}$ und zwei Container der Größe 120.

Problem 1.6: (PARTITION)

Gegeben: n Zahlen $l_1, \ldots, l_n \geq 0$ mit $\sum_{i=1}^{l} l_i = 2L$

Gesucht: $I \subset \{1, \ldots, n\}$ mit $\sum_{i \in I} l_i = L = \sum_{i \notin I} l_i$

Beobachtungen 1.7:

1. wenn es eine Lösung gibt, dann ist das leicht zu verifizieren
2. wenn es keine Lösung gibt, dann kann das u.U. schwer zu belegen sein ($\rightarrow 2^{n-1}$ mögliche Aufteilungen!)

Eigenschaft 1 charakterisiert die Klasse NP. Eigenschaft 2 deutet auf Schwierigkeiten bei der Konstruktion von Lösungen.

Glaubenssatz der Informatik: P $\neq$ NP.

Satz 1.8: (Karp 1972)

Das Problem PARTITION ist NP-vollständig.

Also: Wenn es einen polynomialen Algorithmus für PARTITION gäbe, wäre P=NP (und damit gäbe es sofort für alle anderen Probleme in NP einen polynomialen Algorithmus), bspw. auch für das «Party»-Problem 1.1 auf Seite 3.

Somit ist es unwahrscheinlich, dass es einen «idealen» Algorithmus gibt, des

- schnell ist,
- immer funktioniert und
- eine korrekte Lösung liefert.

1.3 Andere 1D-Probleme

Problem 1.9: (Bin Packing)

Gegeben: n Objekte mit Größe $l_1, \ldots, l_n \in [0, 1]$, Vorrat von Behältern der Größe 1.

Gesucht: Packung der Objekte in möglichst wenige Container, d.h. Partition von $\{1, \ldots, n\} = I_1 \cup I_2 \cup \ldots \cup I_k$ ($\cup$ steht hier für die disjunkte Vereinigung) mit $\sum_{i \in I_j} l_j \leq 1$ und k möglichst klein.

Korollar 1.10:

Bin Packing ist NP-vollständig, denn

1. Lösung leicht zu verifzieren!
2. wenn in P, dann P=NP

Beweis:

1. Bin Packing gehört zu NP, denn eine Lösung lässt sich schnell verifizieren!
2. Angenommen, wir hätten einen polynomialen Algorithmus für Bin Packing. Damit könnten wir für jede beliebige Instanz von PARTITION in polynomieller Zeit entscheiden, ob eine Lösung mit zwei Containern existiert. Damit hätten wir einen polynomialen Algorithmus für ein NP-vollständiges Problem - also wäre P=NP. □

Problem 1.11: (Knapsack)

Gegeben: n Objekte mit Größe $l_1, \ldots, l_n$, Wert $c_1, \ldots, c_n$, Container der Größe L.

Gesucht: Teilmenge $S \subset \{1, \ldots, n\}$ mit $\sum_{i \in S} l_i \leq L$ und $\sum_{i \in S} c_i$ möglichst groß.

Satz 1.12:

KNAPSACK ist NP-schwer (d.h. wenn es einen polynomiellen Algorithmus gibt, dann ist P=NP).

Beweis: Angenommen wir hätten einen polynomiellen Algorithmus für KNAPSACK, dann könnten wir auch PARTITION polynomiell lösen:

Für eine Eingabeinstanz $l_1, \ldots, l_n$ von Partitionen, setze:

$$c_i = l_i; \sum_{i=1}^{n} \frac{l_i}{2} = L$$

und wende den Algorithmus für KNAPSACK an.

Liefert der eine Lösung von Wert $\sum_{i \in S} c_i = L$, dann ist auch $\sum_{i \in S} l_i = L$, also gibt es eine Partition. Ansonsten gibt es keine!

Also wäre PARTITION polynomiell lösbar, also P=NP. □

2 Verfahren für eindimensionale Probleme

2.1 Dynamische Programmierung für SUBSET SUM

Zurück zu PARTITION:

Beispiel 2.1:

Gegeben: $4, 4, 8, 10, 14, 18 \rightarrow \sum_{i=1}^{6} l_i = 58 = 2L$

Gesucht: S mit $\sum_{i \in S} l_i = \sum_{i \notin S} l_i$

Also finde $S \subset \{1, \ldots, 6\}$ mit

$$\sum_{i \in S} l_i = \frac{\sum_{i=1}^{6} l_i}{2} = 29$$

Beobachtung: Nicht lösbar, da nur gerade Zahlen erzielbar sind!

Problem 2.2: (SUBSET SUM)

Gegeben: $l_1, \ldots, l_n$ und A

Gesucht: $S \subset \{1, \ldots, n\}$ mit $\sum_{i \in S} l_i = A$

Frage: Welche Zahlen A sind in Beispiel 2.1 erzielbar?

$\boxed{2}$	4	$\boxed{6}$	8	10	12	14	16	18	$\boxed{20}$	22	24	26	28
$\boxed{56}$	54	$\boxed{52}$	50	48	46	44	42	40	$\boxed{38}$	36	34	32	30

Wie zeigt man, dass die 20 nicht erzielbar ist?

Antwort:

- 18 kann nicht verwendet werden, sonst würde man eine 2 brauchen
- 14 kann nicht verwendet werden, sonst würde man eine 6 brauchen (und das geht mit dem Rest nicht)
- 10 kann nicht verwendet werden, sonst müsste man mit dem Rest $(4, 4, 8)$ eine 10 erzielen
- der Rest hat Summe 16, also ist 20 nicht erzielbar

Beobachtung: Man argumentiert «rückwärts», was nicht geht. Man kann auch vorwärts argumentieren, was geht.

Idee: Führe Buch über die Zahlen z, die mit den ersten i Zahlen $l_1, \ldots, l_i$ erzielbar sind.

$$E(i,z) = \begin{cases} 1, & \text{falls } z \text{ aus } l_1, \ldots, l_i \text{ erzielbar} \\ 0, & \text{falls } z \text{ aus } l_1, \ldots, l_i \text{ nicht erzielbar} \end{cases}$$

Dann gilt $E(0,z) = 0$ für alle $z \geq 1$ und $E(0,0) = 1$. Außerdem ist für $i \geq 1$: $E(i,z) = 1$ genau dann, wenn $E(i-1,z) = 1 \rightarrow$ es geht ohne l_i oder $E(i-1, z-l_i) = 1 \rightarrow$ es geht mit l_i.

i / z	0	2	4	6	8	10	12	14	16	18	20	22	24	26	28
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0
2	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0
3	1	0	1	0	1	0	1	0	1	0	0	0	0	0	0
4	1	0	1	0	1	1	1	1	1	1	0	1	0	1	0
5	1	0	1	0	1	1	1	1	1	1	0	1	1	1	1
6	1	0	1	0	1	1	1	1	1	1	0	1	1	1	1

Systematisch formuliert:

Algorithmus 2.3: (Dynamic Programming für SUBSET SUM)

Eingabe: ganze Zahlen $l_i, \ldots, l_n$, Intervallgrenze L (bspw. $\sum_{i=1}^{n} l_i$)

Ausgabe: Funktion $E : \{0, \ldots, n\} \times \{1, \ldots, L\} \to \{0, 1\}; (i, z) \mapsto E(i, z)$, die beschreibt, welche Zahlen z mit den ersten i Zahlen erzielbar sind.

Listing 1: Algorithmus 2.3

```
E(0,0)=1;

// 1
FOR(z=1) TO L {
  E(0,z)=0;
}

// 2
FOR(i=1) TO n {
  FOR(z=0) TO L {
    IF((E(i-1,z)=1) OR (E(i-1,z-li)=1)) {
      E(i,z)=1;
    }
    ELSE {
      E(i,z)=0;
    }
  }
}
```

Satz 2.4:

Algorithmus 2.3 löst SUBSET SUM in $O(n \cdot L)$.

Beweis: Laufzeit ist klar, Korrektheit folgt aus Vordiskussion!

Frage 2.5:

Wir lösen ein NP-schweres Problem in $O(n \cdot L)$. Warum ist damit nicht P=NP? Wir haben doch ein NP-schweres Problem gelöst, oder nicht?

Antwort: $O(n \cdot L)$ ist nicht polynomiell in der Eingabegröße! L ist eine Zahl, für die wir nur $\log L$ als Inputlänge brauchen - und L ist <u>nicht</u> polynomiell beschränkt durch $\log L$!

Trotzdem ist das besser als nichts...

Definition 2.6:

Ein Algorithmus heißt pseudopolynomiell, wenn seine Laufzeit durch ein Polynom der Inputlänge und der größten vorkommenden Zahl beschränkt ist.

2.2 Dynamic Programming für KNAPSACK

Hier funktioniert eine ähnliche Idee wie für SUBSET SUM! Wir berechnen die erzielbaren Wertzahlen, wobei wir die Kapazitätsgrenze berücksichtigen. Mit $C = \sum_{i=1}^{n} c_i$ erhalten wir folgenden Algorithmus:

Algorithmus 2.7: (Dynamic Programming für KNAPSACK)

Eingabe: Positive, ganze Zahlen $l_1, \ldots, l_n, c_1, \ldots, c_n, L$

Ausgabe: Eine Teilmenge $S \subset \{1, \ldots, n\}$ mit $\sum_{i \in S} l_i \leq L$ und $\sum_{i \in S} c_i$ möglichst groß.

Listing 2: Algorithmus 2.7 Teil 1

```
1  // 1
2  x(0,0)=0;
3  FOR(k=1) TO C {
4    x(0,k)='unendlich';
5  }
6
7  // 2
8  FOR(i=1) TO n {
9    FOR(k=0) TO C {
10     S(i,k)=0;
11     x(i,k)=x(i-1,k);
12   }
13   FOR(k=ci) TO C {
14     IF(x(i-1,k-ci)+ci<=min(L,x(i,k))) {
15       x(i,k)=x(i-1,k-ci);
16       s(i,k)=1;
17     }
18   }
19 }
```

Sei $k = \max\{i \in \{0, \ldots, c\} : x(n, i) < \infty\}$, setze $s = \emptyset$

Listing 3: Algorithmus 2.7 Teil 2

```
1 // 3
2 FOR(i=n) TO 1 {
3   IF(s(i,k)=1) {
4     S.add(i);
5     k=k-ci;
6   }
7 }
```

Satz 2.8:

Algorithmus 2.7 findet eine Optimallösung in $O(n \cdot C)$.

Beweis: Die Laufzeit ist klar. Die Variable $x(i, k)$ beschreibt die kleinstmögliche Gesamtgröße einer Teilmenge $S \subset \{1, \ldots, i\}$ mit $\sum_{i \in S} c_i = k$. Der Algorithmus berechnet diese Funktion mit der Rekursionsformel

$$x(i,k) = \begin{cases} x(i-1, k-c_i) + l_i & \text{wenn } c_i \leq k \text{ und} \\ & x(i-1, k-c_i) + l_i \leq \min\{L, x(i-1,k)\} \\ x(i-1,k) & \text{sonst} \end{cases}$$

Die Variable $s(i, k)$ zeigt an, welcher der beiden Fälle gilt. Der Algorithmus enumeriert also alle Teilmengen $S \subset \{1, \ldots, n\}$ - bis auf die unzulässigen oder diejenigen, für die es eine bessere Lösung s' gibt, d.h.

$$\sum_{i \in S} c_i = \sum_{i \in S'} c_i \text{ und } \sum_{i \in S} l_i > \sum_{i \in S'} l_i$$

In $\boxed{3}$ wird die beste Teilmenge ausgewählt. □

2.3 Heuristiken für BIN PACKING

Umzugsproblem:

Gegeben: Objekte $l_1, \ldots, l_n \in [0, 1]$, Umzugskartons der Größe 1

Gesucht: Aufteilung in möglichst wenige Kartons

Idee: Packe, wie es passt!

Algorithmus 2.9:

Eingabe: Folge von Objekten $l_1, \ldots, l_n \in [0, 1]$

Ausgabe: Aufteilung in Gruppen, die jeweils in einen Karton passen

Listing 4: Algorithmus 2.9

```
1 j=1;
2 FOR(i=1) TO n {
3   IF(li 'passt nicht mehr in Karton' kj) {
4     'schliesse' kj;
5     j=j+1;
6   }
7   'packe' li 'in Karton' kj;
8 }
```

Beispiel 2.10:

Eingabe: 8 Teile der Größe $\frac{1}{2}$, 8 Teile der Größe $\frac{1}{8}$

Ausgabe: 5 Kartons (4 Kartons mit jeweils $2 \cdot \frac{1}{2}$ Teilen und 1 Karton mit $8 \cdot \frac{1}{8}$ Teilen)

Sind die Teile der Eingabe jedoch nicht so «optimal» vorsortiert, ist das Ergebnis ein anderes:

Eingabe: 16 Teile, immer abwechselnd ein $\frac{1}{2}$- und ein $\frac{1}{8}$-Teil

Ausgabe: 8 Kartons (jeweils mit einem $\frac{1}{2}$- und einem $\frac{1}{8}$-Teil)

NEXT FIT benötigt also in diesem Fall fast doppelt soviele Kartons wie im Optimalfall (das ist auch wirklich der schlechteste Fall).

Satz 2.11:

Algorithmus 2.9 liefert in $O(n)$ eine Lösung mit Wert $\mathrm{NF}(I)$, der

$$\mathrm{NF}(I) \leq 2 \cdot \mathrm{OPT}(I) - 1$$

erfüllt, d.h. weniger als der doppelte Wert des Optimums $\mathrm{OPT}(I)$.

Beweis: Die Laufzeit ist klar. Außerdem gilt $l_{1,2j} > r_{2j-1}$ (mit l als erstem Gegenstand im nächsten Karton), d.h.

$$\sum_{l_i \in \{K_{2j-1}, K_{2j}\}} l_i > 1.$$

Also gilt

$$\left\lfloor \frac{\mathrm{NF}(I)}{2} \right\rfloor < \sum_{i=1}^{n} l_i.$$

Da $\left\lfloor \frac{\mathrm{NF}(I)}{2} \right\rfloor$ eine ganze Zahl ist, gilt

$$\frac{\mathrm{NF}(I) - 1}{2} \leq \left\lfloor \frac{\mathrm{NF}(I)}{2} \right\rfloor \leq \left\lceil \sum_{i=1}^{n} l_i \right\rceil - 1,$$

also gilt

$$\mathrm{NF}(I) \leq 2 \left\lceil \sum_{i=1}^{n} l_i \right\rceil - 1.$$

Da offensichtlich

$$\left\lceil \sum_{i=1}^{n} l_i \right\rceil \leq \mathrm{OPT}(I),$$

gilt die Behauptung. □

Der zweite Teil von Beispiel 2.10 zeigt, dass NEXT FIT kurzsichtig ist. Es wäre ja besser, die Kartons später noch für kleinere Gegenstände geöffnet zu lassen. Das liefert:

Beispiel 2.12:

Eingabe: wiederum 16 Teile, immer abwechselnd ein $\frac{1}{2}$- und ein $\frac{1}{8}$-Teil

Ausgabe: 5 Kartons (also eine optimale Lösung):

- Karton 1: $1 \cdot \frac{1}{2} + 4 \cdot \frac{1}{8}$ Teile
- Karton 2, 3 und 5: $2 \cdot \frac{1}{2}$ Teile
- Karton 4: $1 \cdot \frac{1}{8} + 1 \cdot \frac{1}{2} + 3 \cdot \frac{1}{8}$ Teile

Algorithmus 2.13: (FIRST FIT)

Eingabe: Folge von Objekten $l_1, \cdots, l_n \in [0, 1]$

Ausgabe: Aufteilung in Gruppen, die jeweils in einen Karton passen

Listing 5: Algorithmus 2.13

```
1 FOR(i=1) TO n {
2   'packe' i 'in den ersten Karton, in dem noch Platz ist';
3 }
```

Natürlich liefert FIRST FIT nicht immer eine Optimallösung! Ein Beispiel mit 6 Teilen der Größe 0.15, 6 Teilen der Größe 0.34 und 6 Teilen der Größe 0.51 liefert mit Hilfe von FIRST FIT das Ergebnis 10, wobei die Optimallösung 6 wäre.

Satz 2.14: (Johnson 1974, Garey 1976)

Es gilt

$$\mathrm{FF}(I) \leq \left\lceil \frac{17}{10} \cdot \mathrm{OPT}(I) \right\rceil$$

und es gibt Beispiele mit $\mathrm{OPT}(I)$ beliebig groß und

$$\mathrm{FF}(I) \geq \frac{17}{10}\Big(\mathrm{OPT}(I) - 1\Big).$$

Beweis: Kompliziert! Nicht hier ...

Noch etwas cleverer:

Algorithmus 2.15: (FIRST FIT DECREASING (FFD))

Eingabe & Ausgabe: wie FIRST FIT

Listing 6: Algorithmus 2.15

```
1 'sortiere die Objekte in absteigender Größe';
2 'wende FIRST FIT an';
```

Satz 2.16: (Garey & Johnson 1979)

FFD kann einen Faktor $\frac{11}{9}$ vom Optimum entfernt sein.

2.4 Untere Schranken für BIN PACKING

Reicht der Platz für eine Packung? Ideal wäre, wenn der Bereich der Enumeration minimiert würde, schwere Probleme sind jedoch nicht so.

NEIN	**? ? ?**	**JA**
Schnelle Antwort durch untere Schranke!	Enumeration	Schnelle Lösung durch Heuristik, d.h. obere Schranke!

Beste bekannte schnelle Schranke für BIN PACKING (2001)

Berechnung in $O(n)$ nach Sortieren der Objektgrößen (Fekete + Schepers) Einfache untere Schranke: $L_1(I) = \left\lceil \sum_{i=1}^{n} x_i \right\rceil \rightarrow$ so genannte «Volumenschranke»

Beobachtung 2.17: (Johnson 1973)

$L_1(I) \geq \frac{1}{2}\Big(\mathrm{OPT}(I) - 1\Big)$

Beispiel 2.18:

$2m \cdot 0,51$: Schranke liefert etwa m, optimal ist aber $\mathrm{OPT}(I) = 2m$.

Beobachtung 2.19: (Martello + Toth 1990)

1. L_1 ist relativ gut, wenn es relativ viele kleine Objekte gibt
2. durch Konzentration auf «große» Objekte kann man L_1 verbessern und erhält L_2

Satz 2.20: (Martello + Toth 1990)

$L_2(I) \geq \frac{2}{3}\Big(\mathrm{OPT}(I) - 1\Big)$

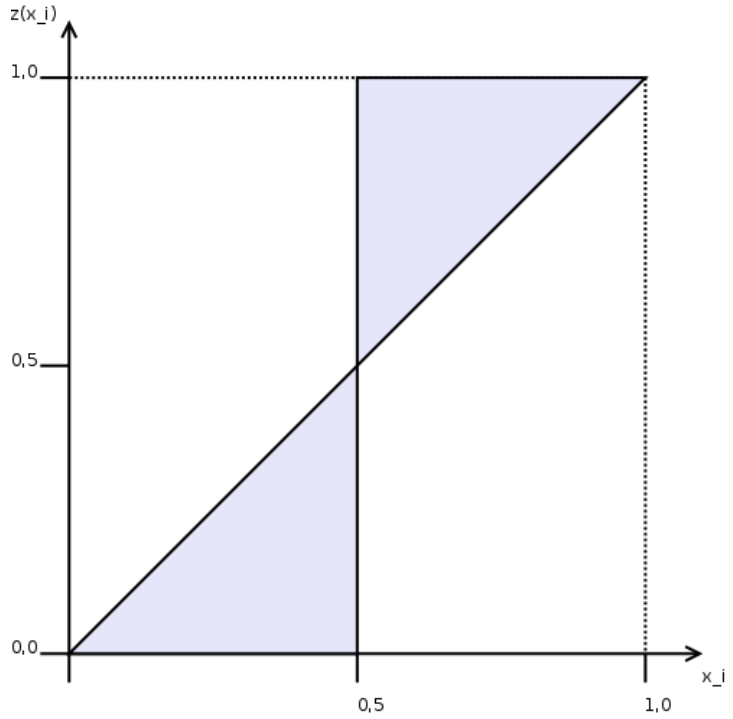

Abbildung 2: Was heißt «groß»?

Alle Objekte mit $x_i \geq \frac{1}{2}$ sind «groß». Das ist aber noch nicht immer gut: Bei $m \cdot 0,53$ und $m \cdot 0,49$ liefert L_1 uns $m+1$ als untere Schranke, die gerade gesehene Definition von «groß» liefert uns m. Die richtige Lösung hingegen wäre $m + \frac{m}{2} = \frac{3}{2}$. Somit muss «groß» geändert werden:

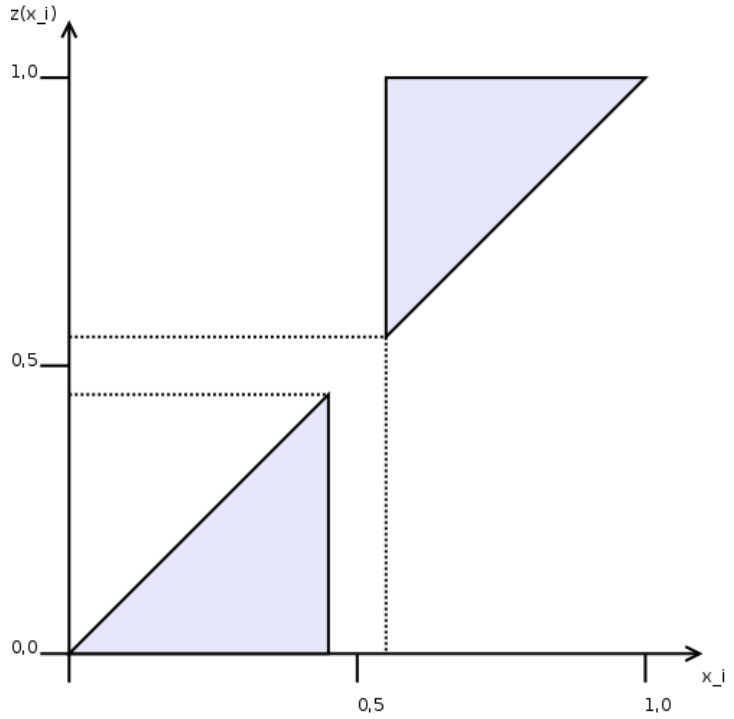

Abbildung 3: ε frei wählbar, Schranke bei $\frac{1}{2} + \varepsilon$

Schranke L_2: Sei $\varepsilon \in \left[0, \frac{1}{2}\right[$, dann teile auf:

- $G_\varepsilon(I) := \left\{ i \in \{1, \ldots, n\} \mid x_i > \frac{1}{2} + \varepsilon \right\}$
- $M_\varepsilon(I) := \left\{ i \in \{1, \ldots, n\} \mid \frac{1}{2} - \varepsilon \leq x_i \leq \frac{1}{2} + \varepsilon \right\}$
- $K_\varepsilon(I) := \left\{ i \in \{1, \ldots, n\} \mid x_i < \frac{1}{2} - \varepsilon \right\}$

Dann ist $\mid G_\varepsilon(I) \mid + \left\lceil \sum_{i \in M_\varepsilon(I)} x_i \right\rceil =: L_2^{(\varepsilon)}$ eine untere Schranke für die Zahl der Bins.

Für $\varepsilon = 0,02$ erhält man im Beispiel ca. $\frac{3}{2} \cdot m$, also die richtige Antwort. Das motiviert

$$L_2(I) := \max_{\varepsilon \in [0, \frac{1}{2}[} L_2^{(\varepsilon)}(I)$$

Wichtig dabei ist die **max**-Auswahl.

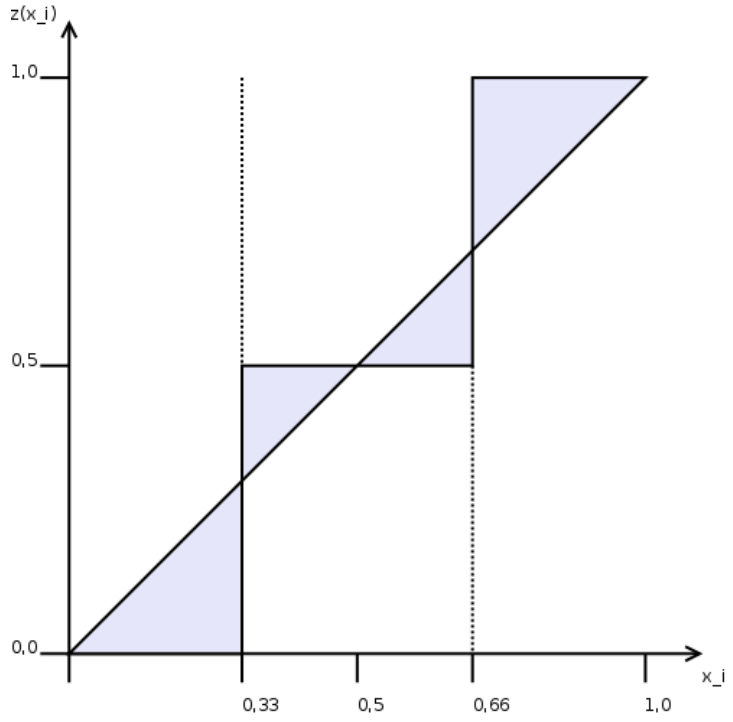

Abbildung 4: $\frac{1}{3} + \varepsilon$

Im Beispiel liefert diese Methode $\frac{3}{2}m \rightarrow$ die korrekte Antwort!

Beobachtung 2.21:

$\frac{2}{3}$ ist der bestmögliche Faktor für L_2: $3m \cdot (\frac{1}{3} + \varepsilon) \rightarrow$ Optimum liefert $\frac{3}{2}m$. L_2 liefert (genau wie L_1) aber nur etwa m.

Noch eine Variante liefert die richtige Lösung bspw. für $4m \cdot (0,26) \rightarrow \frac{4}{3}m$:

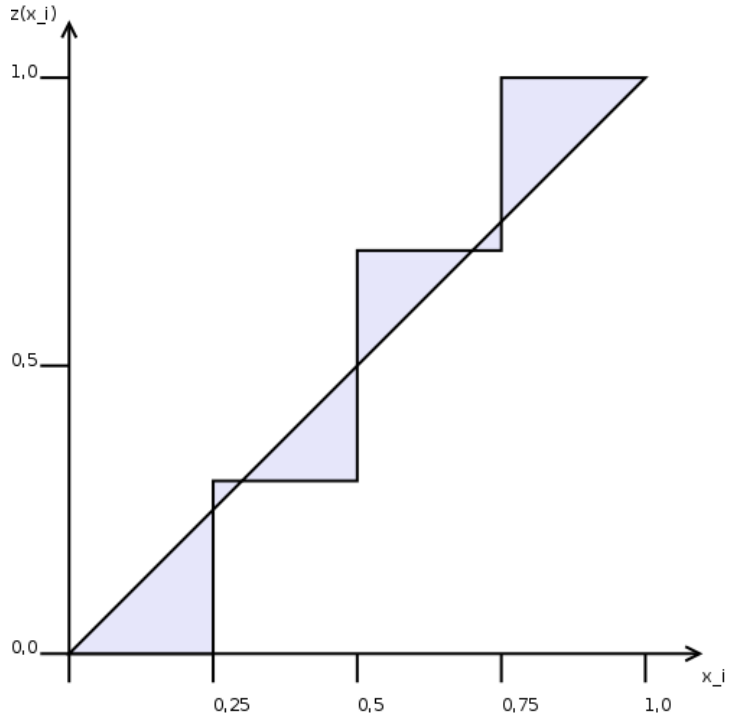

Abbildung 5: Ein weiteres Beispiel

Definition 2.22:

Sei $k \in \mathbb{N}$. Dann setzen wir $n^{(k)} : [0,1] \to [0,1]$.

$$x \mapsto \begin{cases} x & \text{für } x(k+1) \in \mathbb{Z} \\ \lfloor (k+1)x \rfloor \frac{1}{k} & \text{sonst} \end{cases}$$

Definition 2.23: (Dualzulässige Funktionen)

Sei $u : [0,1] \to [0,1]$. $x \mapsto u(x)$. u heißt dualzulässig, wenn gilt

$$\sum_{i \in S} x_i \leq 1 \Rightarrow \sum_{i \in S} u(x_i) \leq 1$$

(Vorstellung: u ist eine zulässige Bilanzierungsfunktion!)

Satz 2.24:

Sei $k \in \mathbb{N}$. Dann ist $n^{(k)} : [0,1] \to [0,1]$.

$$x \mapsto \begin{cases} x & \text{für } x(k+1) \in \mathbb{Z} \\ \lfloor (k+1)x \rfloor \frac{1}{k} & \text{sonst} \end{cases}$$

ist dualzulässig.

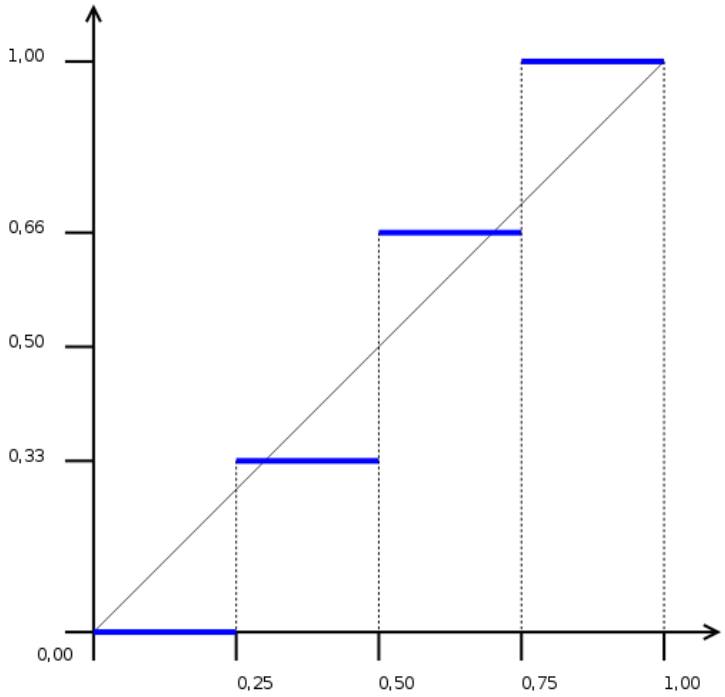

Abbildung 6: $k = 3$

Beweis: Sei S eine endliche Menge nichtnegativer Zahlen mit $\sum_{x \in S} x \leq 1$.

Wir zeigen: $\sum u^{(k)}(x) \leq 1$.

Sei $T := \{x \in S \mid x(k+1) \in \mathbb{Z}\}$.

Wenn $T = S$, dann ist $\sum_{x \in S} u^{(k)}(x) = \sum_{x \in S} x \leq 1$.

Betrachte also $T \neq S$. Dann ist

$$\begin{aligned}
&(k+1)\sum_{x\in T} u^{(k)}(x) + k\sum_{x\in S\setminus T} u^{(k)}(x) \\
&= (k+1)\sum_{x\in T} x + \sum_{x\in S\setminus T} \lfloor (k+1)x \rfloor \\
&< (k+1)\sum_{x\in T} x + (k+1)\sum_{x\in S\setminus T} x = (k+1)\sum_{x\in S} x
\end{aligned}$$

Nach Definition sind $(k+1)\sum_{x\in T} u^{(k)}(x)$ und $k\sum_{x\in S\setminus T} u^{(k)}(x)$ ganzzahlig, also gilt wegen $\sum_{x\in S} x \leq 1$.

$$(k+1)\sum_{x\in T} u^{(k)}(x) + k\sum_{k\in S\setminus T} u^{(k)}(x) \leq k$$

Also gilt

$$\sum_{x\in S} u^{(k)}(x) \leq \frac{(k+1)}{k}\sum_{x\in T} u^{(k)}(x) + \sum_{x\in S\setminus T} u^{(k)}(x) \leq 1$$

Satz 2.25:

Sei $\varepsilon \in [0, \frac{1}{2}]$. Dann ist $U^{(\varepsilon)} : [0,1] \to [0,1]$ mit

$$x \mapsto \begin{cases} 1 & \text{für } x < 1-\varepsilon \\ x & \text{für } \varepsilon \leq x \leq 1-\varepsilon \\ 0 & \text{für } x < \varepsilon \end{cases}$$

dualzulässig.

Beweis: Sei S eine endliche Menge nichtnegativer Zahlen mit $\sum_{x\in S} x \leq 1$. Betrachte:

1. S enthält ein Element größer als $1-\varepsilon$, dann sind alle anderen Objekte kleiner als ε, also $\sum_{x\in S} u^{(\varepsilon)}(x) = 1$.
2. Wenn alle Objekte kleiner als $1-\varepsilon$ sind, dann gilt $\sum_{x\in S} u^{(\varepsilon)}(x) \leq \sum_{x\in S} x \leq 1$. □

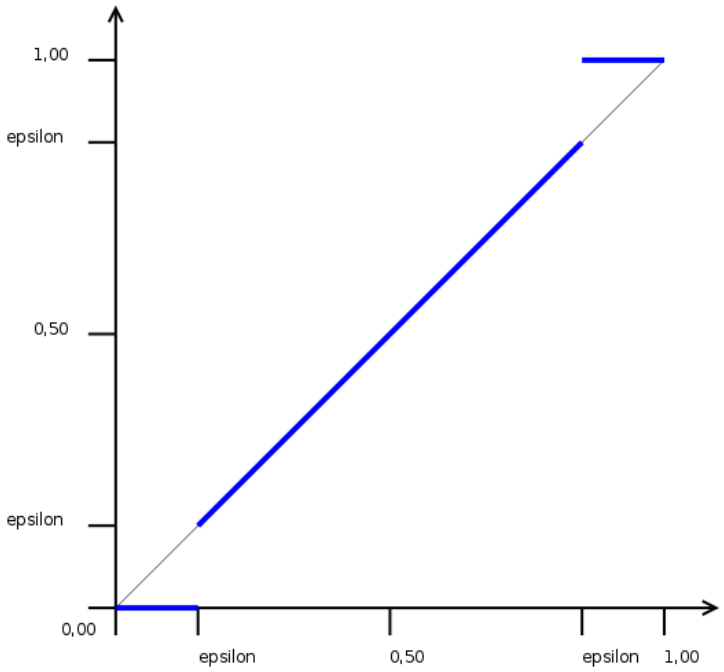

Abbildung 7: Grafische Darstellung zu Satz 2.25

Beobachtung: Hintereinanderausführung dualzulässiger Funktionen liefert dualzulässige Funktion!

Also: $f, g : [0,1] \to [0,1]$ dualzulässig, dann ist

$$\sum_{x \in S} x \leq 1 : \sum_{x \in S} (f \circ g)(x) = \sum_{x \in S} f(g(x)) \leq 1, \text{ denn } \sum_{x \in S} g(x) \leq 1$$

Betrachte folgende Schranken:

$$L_2(I) := \max_{\varepsilon \in [0, \frac{1}{2}]} L_1\Big(U^{(\varepsilon)}(I)\Big)$$

$$L_2^{(k)}(I) := \max_{\varepsilon \in [0, \frac{1}{2}]} L_1\Big(u^{(k)} \circ U^{(\varepsilon)}(I)\Big)$$

$$\text{Dann } L_*^{(p)}(I) := \max\Big\{L_2(I), \max_{k=2,\ldots,p} L_2^{(k)}(I)\Big\}$$

Satz 2.26:

Sei I eine BIN PACKING-Instanz mit Objekten $x_1, \ldots, x_n$, für die $x_i > \frac{1}{3}$ gilt. Dann liefert $L_*^{(2)}(I)$ den korrekten Optimalwert.

Beweis: O.B.d.A betrachte $x_1 \geq x_2 \geq \ldots \geq x_n$. Betrachte nun eine Optimallösung mit m Bins, jedes Bin enthält höchstens zwei Objekte.

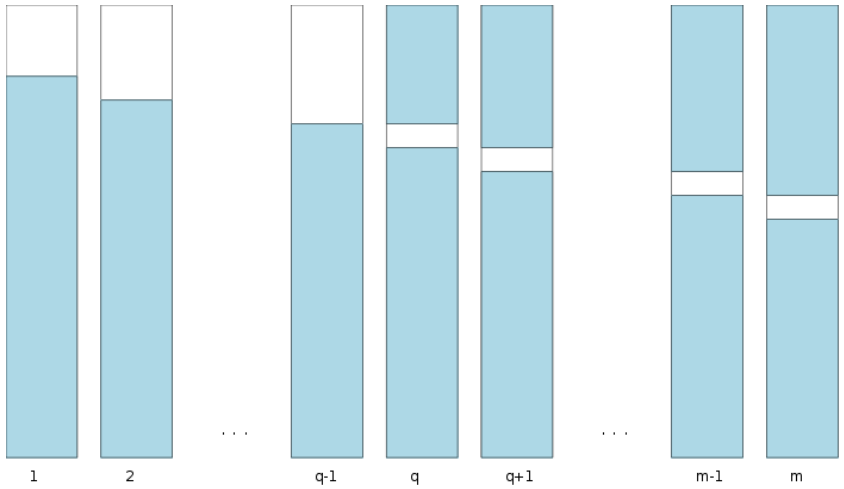

Abbildung 8: $1, \ldots, q-1$ je ein Objekt, $q, \ldots, m$ je zwei Objekte

Diese Lösungsstruktur kann ich folgendermaßen erreichen und daher auch annehmen:

1. sortiere Bins in absteigender Ordnung nach größerem Objekt
2. schiebe kleine Objekte soweit wie möglich nach rechts (kein Problem mit Kapazitäten wg. Größe!)
3. vertausche ggf. kleinere Objekte, um Sortierung der kleineren Objekte nach Größe herzustellen (wieder problemlos möglich!)

Wegen $x_i > \frac{1}{3}$ bekommen wir $u^{(2)}(x_i) = \frac{\lfloor 3x_i \rfloor}{2} \geq \frac{1}{2}$. Für $n \geq 2m-1$ erhalten wir

$$L_2^{(2)}(I) \geq L_1(u^{(2)}(I)) = \left\lceil \sum_{i=1}^{n} n^{(2)}(x_i) \right\rceil \geq \left\lceil \frac{2m-1}{2} \right\rceil = m$$

Betrachte also $n < 2m - 1$. Dann sind also mindestens Bins 1 und 2 nur mit einem Objekt bestückt. Falls nun $x_m > \frac{1}{2}$, dann ist $x_i > \frac{1}{2}$ für alle $i \geq m$, also ist $U^{(\frac{1}{2})}(x_i) = 1$, d.h.

$$L_2(I) \geq L_1\Big(U^{(\frac{1}{2})}(I)\Big) \geq \Big\lceil \sum_{i=1}^{m} U^{(\frac{1}{2})}(x_i) \Big\rceil = m$$

Betrachte also $x_m \leq \frac{1}{2}$. Falls $x_{m-1} + x_m > 1$, betrachte $\varepsilon := x_m \leq \frac{1}{2}$. Für $i \leq m-1$ haben wir $x_i \geq x_{m-1} > 1 - x_m = 1 - \varepsilon$, also $u^{(2)} \circ u^{(\varepsilon)}(x_i) = u^{(2)}(I) = 1$, wegen $\frac{1}{3} < x_m \leq \frac{1}{2}$ erhalten wir $u^{(2)} \circ U^{(\varepsilon)}(x_m) = u^{(2)}(x_m) = \frac{1}{2}$. Also

$$L_2^{(2)}(I) \geq \Big\lceil \sum_{i=1}^{m} u^{(2)} \circ U^{(\varepsilon)}(x_i) \Big\rceil = \Big\lceil (m-1) + \frac{1}{2} \Big\rceil = m$$

Betrachte also $x_{m-1} + x_m \leq 1$. Angenommen, für alle $i^* \in \{2m - n, \ldots, n\}$ gilt $x_i + x_{2m-i-1} \leq 1$, dann kann man alle kleineren Objekte zwei Bins nach links schieben, x_m und x_{m-1} zusammenpacken - und damit eine Lösung mit nur $m-1$ Bins konstruieren.

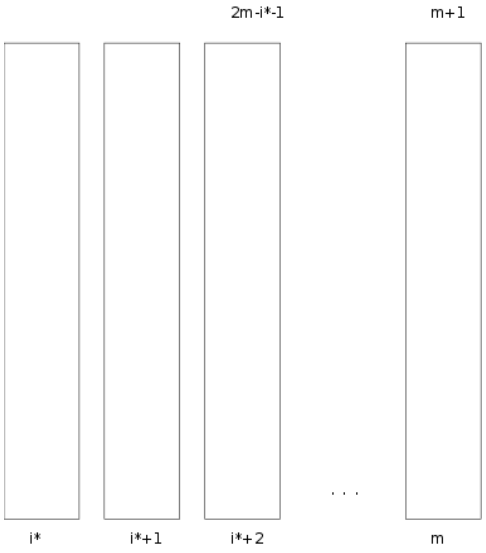

Abbildung 9: Veranschaulichung

Betrachte also ein i^* mit $x_{i^*} + x_{2m-i^*-1} > 1$. Betrachte $\varepsilon := x_{2m-i^*-1}$. Also gilt für $i \in \{1, \ldots, i^*\}$

$$x_i \geq x_{i^*} > 1 - \varepsilon, \text{ d.h. } u^{(2)} \circ u^{(\varepsilon)}(x_i) = u^{(2)}(1) = 1$$

Für $i \in \{i^* + 1, \ldots, 2m - i^* - 1\}$ gilt

$$x_i \geq x_{2m-i^*-1} = \varepsilon, \text{ d.h. } u^{(2)} \circ U^{(\varepsilon)}(x_i) > u^{(2)}(x_i) \geq \frac{1}{2}$$

Zusammen:

$$\begin{aligned} L_2^{(2)}(I) &\geq \Big\lceil \sum_{i=1}^{i^*} u^{(2)} \circ U^{(\varepsilon)}(x_i) + \sum_{i=i^*+1}^{2m-i^*-1} u^{(2)} \circ U^{(2)}(x_i) \Big\rceil \\ &\geq \Big\lceil \sum_{i=1}^{i^*} 1 + \sum_{i=i^*+1}^{2m-i^*-1} \frac{1}{2} \Big\rceil \\ &= \Big\lceil i^* + \frac{(2m - 2i^* - 1)}{2} \Big\rceil = \Big\lceil m - \frac{1}{2} \Big\rceil \\ &= m \ \square \end{aligned}$$