Verfahren zur Lösung von 0-1-Rucksack Problemen
- Theorie und Implementierung -

Vorlesung im Wintersemester 2010/11

- Entwurf -

Ernst-Peter Beisel
Bergische Universität Wuppertal
Fachbereich C
Abteilung Mathematik

Oktober 2010

(**Achtung**: alle Skizzen fehlen in diesem Skript)

Inhaltsverzeichnis

Vorwort

Diese vierstündige Vorlesung ist der zentrale Inhalt des Moduls S.Kap.WM des Wintersemesters 2010/11 in seiner Form mit 6 LP ebenso wie mit 9 LP.

Studierende, die 6 LP erwerben wollen, unterziehen sich am Ende des Semesters einer mdl. Modulprüfung von 30 min, diejenigen, die 9 LP erwerben wollen, haben zusätzlich semesterbegleitend erfolgreich Programmieraufgaben in Matlab zu den Inhalten der Vorlesung zu erledigen.

Das Modul kann auf Antrag auch als Modul "Algorithmen und Datenstrukturen II" anerkannt werden (9LP).

Diskrete Optimierungsaufgaben sind bis auf wenige Ausnahmen NP-hard. Deswegen gilt die Bearbeitung diskreter Optimierungsaufgaben mit großer Dimension als besonders schwer. In dieser Vorlesung sollen theoretisch besonders einfach zu formulierende Aufgaben großer Dimension gelöst werden, sogenannte 0-1 Rucksack Probleme.

Wir wollen das Standard-Handwerkzeug der diskreten Optimierung anwenden und dies an die gegebene Struktur der Aufgabenstellung anpassen, so dass auch große Aufgaben erfolgreich gelöst werden können. Dabei sollen die entstehenden Verfahren programmiernah formuliert und überwiegend auch implementiert werden, um den Erfolg oder Misserfolg der Verfahren einschätzen und die einzelnen Ansätze vergleichen zu können. Es ist durchaus geplant, in Diskussionen der Vorlesung Verbesserungsideen zu entwickeln und direkt umzusetzen.

Wesentliche Grundlage für die Vorlesung und als Begleitliteratur empfohlen sind die beiden folgenden Quellen sowie die dort genannten Literaturbezüge:

- [*MT*] S. Martello / P. Toth: *Knapsack Problems - Algorithms and Computer Implementations*, John Wiley &Sons 1990
- [*P*] D. Pisinger: *Algorithms for Knapsack Problems*, PH.D. thesis 1995, University of Copenhagen

Vorlesungzeiten: Di 12 - 14 und Fr 10 - 12 (sowie ein noch zu bestimmender Ausweichtermin)

Beginn: 12.10.

Peter Beisel

Kapitel 1

Einführung

1.1 Rucksack-Probleme

Wir formulieren zunächst das zu untersuchende Problem und bringen es in ein Standardformat, um es einfacher lösen zu können. Anschließend zeigen wir Varianten der Problemstellung und einige theoretische Anwendungen auf.

1.1.1 Problemstellung

Rucksackprobleme (**Knapsack Problems**) sind ganzzahlige lineare Optimierungsaufgaben mit *nur einer* Restriktion

$$(KP) \quad \left\{ \begin{array}{ll} \text{maximiere} & c^T x \\ \text{s.d.} & a^T x \leq b \\ & 0 \leq x \leq u, \quad x \in \mathbf{Z}^n \end{array} \right. ,$$

bei vorgegebenen Daten $a = (a_1, ..., a_n)^T$, $c = (c_1, ..., c_n)^T \in \mathbf{R}^n$, $u = (u_1, ..., u_n)^T \in \mathbf{R}^n_+$, $b \in \mathbf{R}$.

Sie stehen als Modell für das Problem eines Wanderers, dessen Rucksack b Gewichteinheiten fasst und der das $j-$ten Gut jeweils bis zu einer **Kapazität** u_j auf die Reise mitnehmen möchte, wobei die Zahlen $a_j \in \mathbf{R}_+$ das **Gewicht** einer Einheit des Gutes angeben und die Zahlen $c_j \in \mathbf{R}_+$ dessen **Nutzen** bewerten. Er will eine nutzenoptimale Bepackung seines Rucksacks zusammenstellen. Natürlich kann man sich den Rucksack auch als Container oder anderen Behälter vorstellen oder den Nutzen speziell als Geldwert oder Gewinn interpretieren.

Es gibt mannigfaltige praktische Anwendungen des Rucksack Problems, die sich nicht nur auf das Packen von Gütern oder Gegenständen in vorgegebene Behälter beschränken. Z. B. können verschiedene Investments ausgewählt und zu einem Portfolio zusammengefasst werden, es können verschieden Speisen zu einem Menue zusammengestellt werden müssen. Generell beschreiben Rucksack Probleme *Beladungsprobleme, Verschnittprobleme, Probleme der Budget Kontrolle, des Finanzmanagements etc.*

Man spricht von einem **0-1-Knapsack Problem**, wenn alle $u_j = 1$ sind $(j = 1, ..., n)$. In diesem Falle geben die x_j die *Entscheidung* an, den Gegenstand j überhaupt mitzunehmen. Man kann unschwer einsehen, dass jedes Problem (KP) gleichwertig in ein 0-1 Knapsack Problem umgewandelt werden kann, indem die Kapazitätsgrenzen u_j alle eindeutig als Summe von Zweierpotenzen dargestellt und die Variablen x_j entsprechend in binäre Variablen zerlegt werden. Wir werden später darauf zurückkommen.

Trotz der Einfachheit in seiner Struktur ist das Problem (KP) theoretisch schwer zu lösen, es ist $\mathcal{NP}-$hard. In der Praxis jedoch ist es solide lösbar, seine Lösungsverfahren gelten bei komplexeren Problemen als Handwerkszeug.

Ein Beispiel

Verpackt werden sollen 6 Gegenstände j mit dem jeweiligen Nutzen c_j und dem Gewicht a_j $(j = 1, ..., 6)$. Der Rucksack fasse das Gesamtgewicht $b = 12$. Die Daten seien in Tabellenform gegeben

Gegenstand j	1	2	3	4	5	6
Nutzen c_j	8	8	6	10	12	12
Gewicht a_j	1	2	2	4	6	10

Als Optimierungsproblem der Form

$$\begin{array}{lll} \text{maximiere} & c^T x & \\ \text{s.d.} & \sum_{j=1}^{n} a_j x_j & \leq b \\ & 0 \leq x \leq u & x \in \mathbf{Z}_+^n \end{array}$$

lautet die Aufgabe

$$(P) \qquad \left\{ \begin{array}{ll} \text{maximiere} & 8x_1 + 8x_2 + 6x_3 + 10x_4 + 12x_5 + 12x_6 \\ \text{s.d.} & x_1 + 2x_2 + 2x_3 + 4x_4 + 6x_5 + 10x_6 \leq 12 \\ & x \in \{0,1\}^6 \end{array} \right.$$

Historie

In den 1950er Jahren wurde das Problem erstmalig exakt mittels Dynamischer Optimierung gelöst, seitdem gibt es viele differenzierte Verfahren, die (KP) exakt oder näherungsweise lösen. Heute gelten Branch and Bound Verfahren als Standard-Verfahren für (KP).

Wichtige Zeitdaten sind:

- 1957: **Näherungslösung** mit DANTZIGs Simplexverfahren
- 1967: KOLESAR wendet erstmals ein **B&B Verfahren** auf (KP) an.
- 1973 präsentieren INGARGIOLA AND KORSH die Idee der **Reduzierung** der Aufgabendimension
- 1974 gibt JOHNSON ein erstes **pseudo-polynomiales** Näherungsverfahren für (KP) an.
- 1975: erstes **voll-polynomiales** Näherungsverfahren für (KP) durch IBARRA AND KIM.
- 1980 reduzieren BALAS AND ZEMEL das Lösen von (KP) im Wesentlichen auf ein **Kern-Problem**
- 1980er und 1990er Jahre: MARTELLO UND TOTH sowie PISINGER u.a. **verfeinern die Methoden**.

1.1.2 Standardisierung des Problems

Zunächst kann in der Praxis davon ausgegangen werden, dass alle Daten des Problems *rational* sind. Dann können die Zielfunktion sowie die einzige Restriktion mit dem Hauptnenner der Daten multipliziert werden und es kann der Vektor u der Kapazitätsobergrenzen ganzzahlig abgerundet werden, so dass wir oBdA davon ausgehen können, dass *alle Daten ganzzahlig* sind.

Wir wollen nun einsehen, dass alle Daten sogar als natürliche Zahlen angenommen werden dürfen. Betrachten wir zunächst die Mengen

$$N^0 := \{j \in N : c_j \leq 0, a_j \geq 0\} \quad \text{und} \quad N^1 := \{j \in N : c_j \geq 0, a_j \leq 0\}$$

wobei wir zur Abkürzung $N := \{1, ..., n\}$ gesetzt haben.

Dann liegen die Werte für die $x_j,\ j \in N^0 \cup N^1$, in einer Optimallösung von (KP) von vorne herein fest: Gegenstände mit $j \in N^0$ verursachen durch ihr Gewicht eine

potentielle Reduzierung der Kapazität des Rucksacks, tragen durch ihren nichtpositiven Nutzen aber nicht positiv zur Maximierung der Zielfunktion bei. Sie haben also in der Optimallösung den Wert $x_j = 0$.

Umgekehrt vermindern die Gegenstände mit $j \in N^1$ nicht die Kapazität des Rucksacks, sie tragen aber potentiell positiv zum Nutzen bei, deshalb gilt in der Optimallösung hier $x_j = 1$. Insofern können wir oBdA davon aus gehen, dass im gestellten Problem $N^0 \cup N^1$ leer ist.

Damit dürfen wir insbesondere davon ausgehen, dass alle Daten c_j und a_j, $j \in N$, nichttrivial sind. Betrachten wir nun die Indexmengen

$$N^- := \{j \in N : c_j < 0, a_j < 0\} \quad \text{und} \quad N^+ := \{j \in N : c_j > 0, a_j > 0\}$$

Nach unseren Vorüberlegungen muss $N^- \cup N^+ = N$ gelten. Ist $N^- \neq \varnothing$, so kann für $j \in N^-$ zu $\tilde{x}_j := u_j - x_j$ übergegangen werden bei gleichzeitiger Abänderung $\tilde{c}_j := -c_j$, $\tilde{a}_j := -a_j$ und $b := b - u_j a_j$. Offensichtlich beschreibt diese Veränderung eine zur gegebenen Aufgabe äquivalente Aufgabe, bei der nun N^- einen Index weniger und N^+ einen Index mehr enthält. Auf diese Weise kann N^- nach und nach geleert werden. Wir können also oBdA davon ausgehen, dass von vorne herein $N^- = \varnothing$ gilt. In diesem Fall muss bei lösbarer Aufgabe natürlich auch $b \geq 0$ gelten.

Wir können also unterstellen, dass

$$\text{alle Daten der Aufgabe } (KP) \text{ natürliche Zahlen} \tag{1.1}$$

sind. Gleichzeitig können wir bei lösbarer Aufgabe annehmen, dass

$$\sum_{j=1}^{n} a_j u_j > b \tag{1.2}$$

gilt, da sonst "alle Gegenstände in voller Anzahl in den Rucksack passen" und es nichts zu optimieren gibt. Außerdem muss

$$a_j \leq b \text{ gelten für alle } j \in N, \tag{1.3}$$

da sonst der "Gegenstand j gar nicht in den Rücksack passt".

Als **Standardformat** für das Rucksackproblem legen wir also fest ein Problem (KP), das die Eigenschaften (1.1), (1.2) und (1.3) hat.

1.1.3 Varianten des klassischen Rucksackproblems

Zunächst betrachten wir Problemstellungen, die einfach durch *Spezialisierung des Klassischen Knapsack Problems* entstehen. Als solches kennen wir schon das 0-1-**Knapsack Problem**, bei dem in den Rucksack jeweil nur ein Exemplar eines Gegenstandes gepackt werden darf.

Entfallen die Kapazitätsgrenzen beim Problem (KP), d.h. gilt $u_j = \infty$ für alle j, so sprechen wir vom **unbeschränkten Knapsack Problem** (UKP)**.**

Gilt im Knapsack Problem für alle j die Gleichheit $c_j = a_j$, gibt also das Gewicht des Gegenstands j gleichzeitig seinen Nutzen an, so sprechen wir vom **Subset-Sum Problem** (SSP). In diesem Fall geht es also darum, das Gewicht des im Rucksack mitgenommenen Gegenständen möglichst groß zu machen, d.h. die Gewichtskapazität des Rucksacks optimal auszunutzen.

Eine weitere Spezialisierung des klassischen Rucksackproblems ist das **Change-Making Problem**. Bei diesem sind alle Nutzenkoeffizienten $c_j = 1$ und die Gewichtsrestriktion muss mit Gleichheit erfüllt werden

$$(CMP) \quad \begin{cases} \text{maximiere} & e^T x \\ \text{s.d.} & a^T x = b \\ & 0 \leq x \leq u, \quad x \in \mathbf{Z}^n \end{cases},$$

$e^T = (1, 1, ..., 1) \in \mathbf{R}^n$. Anwendung findet das Problem, wenn ein bestimmter Geldbetrag b in eine maximale Anzahl von Münztypen gewechselt werden soll, die natürlich nur in bestimmter Stückelung vorliegen; Einer, Zweier, Fünfer, Zehner, ...

Abarten des Klassischen Rucksackproblems, die die Form (KP) verlassen, aber noch Verwandtschaft in Form oder Modellbildung zu (KP) besitzen, sind die folgenden:

- das **Multiple-Choice Knapsack Problem** $(MCKP)$. Bei diesem muss die Auswahl der Gegenstände so erfolgen, dass aus jeder Teilmenge einer vorgegebenen *disjunkten Zerlegung* von N (*Partition*), $N =: N_1 \cup N_2 \cup ... \cup N_s$ jeweils genau ein Gegenstand in den Rucksack kommt.

$$(MCKP) \quad \begin{cases} \text{maximiere} & c^T x & \\ \text{s.d.} & a^T x \leq b & \\ & \sum_{j \in N_i} x_j = 1 & i = 1, .., s \\ & 0 \leq x \leq e, & x \in \mathbf{Z}^n \end{cases}$$

- das 0-1**-Multiple Knapsack Problem** (MKP). Dieses Rucksackproblem unterstellt das Vorhandensein von m Rucksäcken mit jeweiliger Kapazität b_i $(i = 1, ..., m)$, auf die die Gegenstände verteilt werden dürfen. Zur Formulierung geben wir den Variablen Doppelindizes

$$(MKP) \quad \begin{cases} \text{maximiere} & \sum_{i=1}^{m} \sum_{j=1}^{n} c_j x_{ij} & \\ \text{s.d.} & \sum_{j=1}^{n} a_j x_{ij} \leq b_i & i = 1, ..., m \\ & \sum_{i=1}^{m} x_{ij} \leq 1 & j = 1, ..., n \\ & 0 \leq x \leq e, & x \in \mathbf{Z}^{m \times n} \end{cases}$$

- eine Variante dieses Multiple Knapsack Problems ist das **Generalized Assignment Problem.** Bei diesem sind die Nutzenkoeffizienten und die Gewichtskoeffizienten auch vom Rucksack abhängig. Interpretiert werden kann das Problem so, dass n Jobs auf m Maschinen verteilt werden soll, wobei natürlich jeder Job höchstens einmal verteilt werden kann:

$$(GAP) \quad \begin{cases} \text{maximiere} & \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} & \\ \text{s.d.} & \sum_{j=1}^{n} a_{ij}x_{ij} \leq b_i & i = 1, ..., m \\ & \sum_{i=1}^{m} x_{ij} \leq 1 & j = 1, ..., n \\ & 0 \leq x \leq e, & x \in \mathbf{Z}^{m\times n} \end{cases}$$

- schließlich seien **Quadratische Knapsack Probleme** genannt, bei denen die Zielfunktion nicht linear, sondern quadratisch ist.

1.1.4 Theoretische Anwendungen

Neben den praktischen Anwendungen haben Rucksack Probleme auch theoretische Anwendungen. So treten sie bei komplexeren Problemen als Unterprobleme auf oder helfen bei der Lösung schwierigerer Probleme. Im Folgenden sollen zwei theoretische Anwendungen des Rucksack Problems benannt werden.

Reduzierung auf ein Knapsack Problem

Rucksack Probleme in Standardform sind **rein ganzzahlige (lineare) Optimierungsprobleme**, d.h. Optimierungsprobleme mit ganzzahligen Daten und ganzzahlig geforderten Variablen. Interessanterweise kann man zeigen, dass rein ganzzahlige lineare Optimierungsprobleme in gleichwertige Rucksack Probleme umgeformt, d.h. in Rucksackprobleme mit denselben Variablen und denselben (zulässigen) Lösungsmengen. Dies kann wie folgt eingesehen werden.

Vorgegeben sei das rein ganzzahlige lineare Optimierungsproblem mit zwei Restriktionen

$$(P) \quad \begin{cases} \text{Maximiere} & \sum_{j=1}^{n} p_j x_j & \\ \text{s.d.} & \sum_{j=1}^{n} w_{1j}x_j = c_1 & \\ & \sum_{j=1}^{n} w_{2j}x_j = c_2 & \\ & x_j \in [0, u_j] \cap \mathbf{Z}, & j = 1, ..., n \end{cases}$$

mit *ganzzahligen Daten*. Setzt man nun ein beliebiges $x \in ([0, u_j] \cap \mathbf{Z})^n$ in die linke Seite der ersten Restriktion ein, so hat man die Abschätzung

$$c_1 - \sum_{j=1}^{n} w_{1j}^{+}u_j \leq c_1 - \sum_{j=1}^{n} w_{1j}x_j \leq c_1 - \sum_{j=1}^{n} w_{1j}^{-}u_j$$

wobei $w_{1j}^{+} = \max\{w_{1j}, 0\}$ und $w_{1j}^{-} = \min\{w_{1j}, 0\}$ gesetzt wurden. Wir wählen nun eine natürliche Zahl

$$M > \max\left\{c_1 - \sum_{j=1}^{n} w_{1j}^{-} u_j, -c_1 + \sum_{j=1}^{n} w_{1j}^{+} u_j\right\}$$

Dann gilt also

$$\left|c_1 - \sum_{j=1}^{n} w_{1j} x_j\right| < M \tag{1.4}$$

Nun multiplizieren wir die zweite Restiktion im gegebenen Problem mit M und addieren diese Gleichung zur ersten Gleichung hinzu. So entsteht das folgende Problem

$$\left(\bar{P}\right) \quad \begin{cases} \text{Maximiere} & \sum_{j=1}^{n} p_j x_j \\ \text{s.d.} & \sum_{j=1}^{n} \left(w_{1j} + M w_{2j}\right) x_j = c_1 + M c_2 \\ & x_j \in [0, u_j] \cap \mathbf{Z}, \qquad\qquad j = 1, ..., n \end{cases}$$

Es gilt dann das folgende

Lemma: Die Aufgaben (P) und $\left(\bar{P}\right)$ haben identische Mengen zulässiger Lösungen.

Beweis: Natürlich ist die Lösungsmenge von (P) in der von $\left(\bar{P}\right)$ enthalten. Zu zeigen ist, dass auch das Umgekehrte gilt.

Sei also x eine zulässige Lösung von $\left(\bar{P}\right)$. Wir setzen

$$c_2 - \sum_{j=1}^{n} w_{2j} x_j := K \tag{1.5}$$

wobei $K \in \mathbf{Z}$ gelten muss nach Voraussetzung. Nun kann die Restriktion von $\left(\bar{P}\right)$ auch wie folgt geschrieben werden:

$$\left(c_1 - \sum_{j=1}^{n} w_{1j} x_j\right) + M\left(c_2 - \sum_{j=1}^{n} w_{2j} x_j\right) = 0 \tag{1.6}$$

d.h.

$$\left(c_1 - \sum_{j=1}^{n} w_{1j} x_j\right) + MK = 0$$

Daraus folgt

$$K = -\frac{1}{M}\left(c_1 - \sum_{j=1}^{n} w_{1j} x_j\right)$$

Wegen (1.4) folgt daraus

$$|K| < 1$$

Weil aber K ganzzahlig war, mus $K = 0$ gelten. Mit (1.5) erfüllt daher x die zweite Restriktion von (P) und wegen (1.6) auch die erste. □

Hat man eine rein ganzzahlige lineare Optimierungsaufgabe mit mehreren Restriktionen, so können diese unter Anwendung der soeben beschriebenen Konstruktion nacheinander in eine einzige rein ganzzahlige Restriktion umgewandelt werden, ohne dass sich dabei die Lösungsmenge ändert. Mit der weiter oben beschriebenen Technik kann diese Aufgabe dann in ein 0-1 Knapsack Problem im Standdardformat umgewandelt werden und als solches gelöst werden.

Zu beachten ist, dass die Koeffizienten der so entstehenden 0-1 Rucksack Probleme schnell sehr groß werden, was die praktische Lösbarkeit behindert.

Verschärfung von Restriktionen

Eine der Standardtechniken der diskreten Optimierung im Rahmen des Preprocessing ist die *Verschärfung vorliegender Restriktionen* mit dem Ziel, dass dadurch anzuwendende Branch and Bound Verfahren schneller ablaufen. Wie man dabei hilfreich Rucksack Probleme einsetzen kann, zeigt exemplarisch die folgende Überlegung.

Gegeben sei ein rein ganzzahliges lineares Optimierungsproblem

$$(P) \quad \begin{cases} \text{Maximiere} & \sum_{j=1}^{n} p_j x_j \\ \text{s.d.} & \sum_{j=1}^{n} w_{ij} x_j \leq c_i \quad i = 1, ..., m \\ & x_j \in [0, u_j] \cap \mathbf{Z}, \quad j = 1, ..., n \end{cases}$$

bei dem also alle Daten ganzzahlig sind. Wir wollen die Restriktionen der Aufgabe verschärfen, indem wir die rechten Seiten c_i erniedrigen, ohne den zulässigen Bereich zu verletzen.

Um eine einzelne Restriktion, z. B. die $i-$te, zu verschärfen, betrachtet man das zugeordnete *Subset Sum Problem*

$$(KP_i) \quad \begin{cases} \text{Maximiere} & z_i := \sum_{j=1}^{n} w_{ij} x_j \\ \text{s.d.} & \sum_{j=1}^{n} w_{ij} x_j \leq c_i \\ & x_j \in [0, u_j] \cap \mathbf{Z}, \quad j = 1, ..., n \end{cases}$$

Löst man dieses und findet man einen optimalen Zielfunktionswert $z_i < c_i$, so kann die $i-$te Restriktion der Aufgabe (P) offensichtlich zu $\sum_{j=1}^{n} w_{ij} x_j \leq z_i$ verschärft werden.

Entsprechend kann mit jeder Restriktion von (P) verfahren werden.

1.2 Grundlegende Techniken

Grundsätzliche Methode der Bearbeitung von gemischt ganzzahligen linearen Optimierungsproblemen, insbesondere also des Knapsackproblems ist ein Enumerieren aller potentiellen Lösungen, wobei durch Ausschluss unattraktiver Lösungen versucht wird, nur eine "kleine" Anzahl von Lösungen wirklich explizit aufzählen zu müssen. Erreicht werden kann dies durch geschicktes Abschätzen, wenn festgestellt wird, dass eine (größere) Anzahl betrachteter potentieller Lösungen keine Optimallösungen sein können, da ihr Zielfunktionswert "zu schlecht" ist.

Wir unterscheiden dabei zwischen

- der Branch and Bound Methode und
- der Methode des Dynamischen Programmierens

Im Folgenden werden diese bekannten Vorgehensweisen zunächst komprimiert zusammengefasst und dann am 0-1 Knapsack Problem erläutert.

1.2.1 Preprocessing beim 0-1-Knapsackproblem

Wir streben an, zunächst die B&B Methode und die Methode des dynamischen Programmierens anwenden auf das allgemeine 0-1-Knapsack Problem

$$(KP_{01}) \quad \begin{cases} \text{maximiere} & c^T x \\ \text{s.d.} & a^T x \leq b \\ & x \in \{0,1\}^n \end{cases}$$

wobei wir unterstellen, dass alle Daten der Aufgabe ganzzahlig und positiv seien. Auf die Forderung, dass $a_j \leq b$ gilt für alle $j = 1, ..., n$ und dass $\sum_{j=1}^{n} a_j > b$ gilt, wie im Standardformat gefordert, verzichten wir.

Beginnen wir mit der Untersuchung eines Zahlenbeispiels. Wir betrachten dazu das bereits oben genannte konkrete 0-1-Rucksackproblem

$$(P) \quad \begin{cases} \text{maximiere} & 8x_1 + 8x_2 + 6x_3 + 10x_4 + 12x_5 + 12x_6 \\ \text{s.d.} & x_1 + 2x_2 + 2x_3 + 4x_4 + 6x_5 + 10x_6 \leq 12 \\ & x \in \{0,1\}^6 \end{cases}$$

Vorab wollen wir die gegebene Aufgabenstellung für die Bearbeitung durch die B&B Methode oder durch Dynamische Programmierung vorbereiten. Dabei kann in der Regel die Dimension des Problems herabgesenkt werden, so dass weniger Daten zu verarbeiten sind bzw eine schnellere Bearbeitung erwartet werden kann.

Zu beachten ist, dass diese Techniken auch *während* der eigentlichen Bearbeitung der Aufgabe nutzbringend verwendet werden können.

Es erweist sich beim 0-1-Knapsackproblem (KP_{01}) i.a. als günstig, die Reihenfolge der betrachteten Variablen so festzulegen, dass die sogenannten **Nutzen-Gewichts-Quotienten**, **Effizienzen** oder **Nutzendichten** $\zeta_j := c_j/a_j$ bei steigendem j abfallend sind. Die ζ_j geben an, welchen Nutzen es bringt, eine Gewichtseinheit des $j-$ten Gegenstands in den Rucksack zu packen. Wegen der Sortierung der Nutzen-Gewichts-Quotienten ist es also sinnvoller, Gewichtseinheiten mit niedrigerem Index in den Rucksack zu packen also solche mit höherem Index.

In unserer konkreten Aufgabenstellung ist die gewünschte Sortierung bereits gegeben:

$$\text{Nutzen-Gewichts-Quotienten } \zeta_j\text{: } 8, 4, 3, \frac{5}{2}, 2, \frac{6}{5},$$

so dass keine Umsortierung erfolgen muss.

Liegt die gewünschte Sortierung nicht vor, so kann umsortiert werden. Dazu können klassische Sortierungsalgorithmen (z.B. *Quicksort*) verwendet werden, z. B. mit einer Laufzeit von $\mathcal{O}(n \log n)$.

Bis auf Widerruf setzen wir aber voraus, dass bei der vorgegebenen Aufgabe die Variablen bereits nach abfallenden Nutzen-Gewichts-Quotienten sortiert sind.

Upper and Lower Bounds

Wir wollen eine *obere und eine untere Schranke* für den optimalen Zielfunktionswert von (P) berechnen. Zu beachten ist dabei, dass (P) eine *Maximierungsaufgabe* darstellt, die Bedeutungen von oberer und unterer Schranke im Vergleich zur allgemeinen Darstellung also ausgetauscht sind.

Zur Ermittlung einer unteren und einer oberen Schranke für den optimalen Zielfunktionswert der gegebenen Aufgabe betrachten wir die leicht abgeschwächte Aufgabenstellung

$$(RP) \qquad \begin{cases} \text{maximiere} & z = 8x_1 + 8x_2 + 6x_3 + 10x_4 + 12x_5 + 12x_6 \\ \text{s.d.} & x_1 + 2x_2 + 2x_3 + 4x_4 + 6x_5 + 10x_6 \leq 12 \\ & x \in [0,1]^6 \end{cases}$$

bei der *keine der Variablen mehr ganzzahlig gefordert wird.*

> **Wir nennen allgemein** diese Abschwächung einer gemischt ganzzahligen linearen Aufgabenstellung, bei der lediglich auf die Forderung der Ganzzahligkeit verzichtet wird, **LP-Relaxation**.

Die so entstandene lineare Optimierungsaufgabe (RP) ist mit dem Simplexverfahren leicht zu lösen. Die folgenden Überlegungen vollziehen nach, wie das Simplexverfahren dabei günstig vorgehen kann.

> Im Zusammenhang mit den Nutzen-Gewichts-Quotienten (abfallend sortiert) **nennen wir allgemein** folgenden Index $k \in \{1, ..., n\}$ der Aufgabe (KP_{01}) **kritisch (break item)**, wenn für diesen gilt:
>
> $$\sum_{j=1}^{k-1} a_j \leq b < \sum_{j=1}^{k} a_j$$
>
> Der kritische Index gibt also den Index des Gegenstands an, der *gerade nicht mehr* in den Rucksack passt, wenn die Gegenstände in der Reihenfolge abfallender Nutzen-Gewichts-Quotienten in den Rucksack gepackt werden.

Man beachte, dass $k = 1$ in diesem Zusammenhang nur heißt, dass bereits der erste Gegenstand nicht in den Rucksack passt, dass also $a_1 > b$ gilt. In diesem Fall kann die Variable x_1 auf $x_1 = 0$ fixiert werden und die Dimension des Problems damit um eins abgesenkt werden.

In unserem Beispiel ist der Index $k = 5$ kritisch. Führt man den (ganzzahligen) *Schlupf* y in der gegebenen Restriktion ein, hier

$$y := 12 - (x_1 + 2x_2 + 2x_3 + 4x_4 + 6x_5 + 10x_6)\,,$$

löst diese Gleichung nach der *kritischen* Variablen x_k auf und ersetzt über diese Gleichung x_k in der Zielfunktion, so entsteht eine äquivalente Aufgabenformulierung, in unserem Beispiel

$$\begin{array}{lll} \text{maximiere } z \text{ mit} & z - 6x_1 - 4x_2 - 2x_3 - 2x_4 + 2y + 8x_6 = 24 \\ & x_5 + \frac{1}{6}x_1 + \frac{2}{6}x_2 + \frac{2}{6}x_3 + \frac{4}{6}x_4 + \frac{1}{6}y + \frac{10}{6}x_6 = 2 \\ & x \in [0,1]^6\,,\ y \geq 0 \end{array}$$

Diese Umrechnung der Aufgabe entspricht offenbar der Anwendung eines Simplex-Austauschschritts, der die kritische Variable von der Nichtbasis in die Basis bringt. Das Pivotelement für diesen ATS ist damit eindeutig festgelegt und positiv, da die Aufgabe im Standardformat vorliegt.

Spiegeln wir nun noch die ersten (vier) Variablen an der Obergrenze:

$\bar{x}_j := 1 - x_j, \quad j = 1, ..., 4$, so entsteht

$$\left(R\bar{P}\right) \qquad \left\{ \begin{array}{lll} \text{maximiere } z \text{ mit} & z + 6\bar{x}_1 + 4\bar{x}_2 + 2\bar{x}_3 + 2\bar{x}_4 + 2y + 8\bar{x}_6 = 38 \\ & \bar{x}_5 - \frac{1}{6}\bar{x}_1 - \frac{2}{6}\bar{x}_2 - \frac{2}{6}\bar{x}_3 - \frac{4}{6}\bar{x}_4 + \frac{1}{6}y + \frac{10}{6}\bar{x}_6 = \frac{3}{6} \\ & \bar{x} \in [0,1]^6\,,\ y \geq 0 \end{array} \right.$$

wobei wir der Einfachheit halber noch $\bar{x}_5 := x_5$ und $\bar{x}_6 := x_6$ gesetzt haben. Offensichtlich ist damit $\bar{x}_1 = ... = \bar{x}_4 = 0,\ \bar{x}_6 = 0,\ y = 0,\ \bar{x}_5 = \frac{1}{2}$ eine Optimallösung der letzten Aufgabenformulierung, d.h. $x_1 = x_2 = x_3 = x_4 = 1, \quad x_5 = \frac{1}{2}$ und $x_6 = 0$ mit Zielfunktionswert $s = 38$ ist eine Optimallösung der linearen Optimierungsaufgabe (RP)!

Aus den bisherigen Ergebnissen können wir für unser Beispiel zwei Konsequenzen ziehen:

1. Jede zulässige Lösung von (P) muss einen Zielfunktionswert ≤ 38 haben.
2. Durch Abrundung von x_5 auf 0 erhalten wir eine erste zulässige Lösung von $(P) : x' = (1, 1, 1, 1, 0, 0)^T$ mit Zielfunktionswert $z = 32$.

Bemerkung: Es lässt sich leicht nachprüfen, dass die Lösung x' nicht zu einer besseren zulässigen Lösung "erweitert" werden kann: diese Lösung verbraucht bereits 9 der 12 Gewichtseinheiten des Rucksacks, die restlichen Gegenstände sind zu schwer, um noch in den Rucksack zu passen.

Damit haben wir als *obere Schranke* für den optimalen Zielfunktionswert den Wert $U = 38$ gefunden und als *untere Schranke* den Wert $L = 32$. Der optimale Zielfunktionswert der Aufgabe (P) muss also zwischen 32 und 38 liegen.

Dantzigschranke Die oben durchgeführten äquivalenten Umformungen (ein Simplex-Austauschschritt und die Transformationen der ersten Variablen bis vor den kritischen Index) der LP-Relaxation (RKP_{01}) der gegebenen Aufgabe (KP_{01}) können auch leicht in völliger Allgemeinheit durchgeführt werden. Wir formulieren daraus die folgenden Erkenntnisse, die auf Dantzig (1957) zurückgehen.

Am Beispiel zu beobachten war, dass die Zielfunktionskoeffizienten in der äquivalenten Formulierung der Aufgabe nach dem Simplex-Schritt für die Indizes vor dem kritischen Index k nichtpositiv und ab k nichtnegativ sind. Das liegt an der speziellen Sortierung der Variablen! Dies lässt sich wie folgt einsehen:

Die Koeffizienten aller Variablen x_j $(j \neq k)$ in der neuen Zielfunktion ergeben sich wegen der Simplex Austauschregeln einheitlich zu

$$-c_j + \zeta_k a_j, \tag{1.7}$$

der Koeffizient von y zu

$$\zeta_k$$

und die neue rechte Seite in der Zielfunktion zu

$$\zeta_k b.$$

Die neuen Koeffizienten der Restriktion lauten

$$\frac{a_j}{a_k} \quad \text{für } x_j, \quad j = 1, ..., n, \quad j \neq k \quad \text{und} \quad \frac{1}{a_k} \quad \text{für } y,$$

die neue rechte Seite ist

$$\frac{b}{a_k}$$

Wegen der vorausgesetzten Anordnung der Nutzen-Gewichts-Quotienten ζ_j und der Definition des break items ergibt sich damit klar die beobachtete Verteilung der Vorzeichen in der neuen Zielfunktion, auch im allgemeinen Fall.

Also bietet sich auch im allgemeinen Fall die Transformation $\bar{x} := 1 - x_j$ für alle $j < k$ an, um das zugehörige Simplextableau zu einem Optimaltableau zu machen. Nach dieser Transformation zu $\left(RK\bar{P}_{01}\right)$ ergibt sich allgemein die neue rechte Seite der Restriktion mit der Abkürzung $\bar{b} := \left(b - \sum_{j=1}^{k-1} a_j\right)$ zu

$$\frac{b}{a_k} - \sum_{j=1}^{k-1} \frac{a_j}{a_k} = \frac{1}{a_k}\left(b - \sum_{j=1}^{k-1} a_j\right) = \frac{\bar{b}}{a_k} \quad \text{mit } 0 \leq \frac{\bar{b}}{a_k} < 1$$

während die neue rechte Seite der Zielfunktion den folgenden Wert hat

$$\zeta_k b - \sum_{j=1}^{k-1} (-c_j + \zeta_k a_j) = \sum_{j=1}^{k-1} c_j + \zeta_k \left(b - \sum_{j=1}^{k-1} a_j\right) = \sum_{j=1}^{k-1} c_j + \zeta_k \bar{b} \geq 0 \qquad (1.8)$$

Das zugehörige verkürzte Tableau ist also ein Optimaltableau zu $\left(RK\bar{P}_{01}\right)$!

Aus diesen Betrachtungen ergibt sich der folgende Satz, der bei der Lösung der Aufgabe (RKP_{01}) auf eine explizite Anwendung des Simplexverfahrens mit anschließender Variablentransformation verzichten kann.

Satz 1.1 *Vorgegeben sei ein 0-1-Knapsackproblem vom Format* (KP_{01})*. Es seien die Variablen nach abfallenden Nutzen-Gewichts-Quotienten geordnet und* $k \in \{1, ..., n\}$ *der kritische Index. Dann gibt* $\bar{x}$ *mit*

$$\bar{x}_j = \begin{cases} 1 & \text{für } j = 1, ..., k-1 \\ \frac{\bar{b}}{a_j} & \text{für } j = k \\ 0 & \text{für } j = k+1, ..., n \end{cases}$$

wobei

$$\bar{b} = b - \sum_{j=1}^{k-1} a_j$$

gesetzt wurde, die Optimallösung der LP-Relaxation an.
Der Zielfunktionswert dieser Lösung ist mit $\bar{c} := \sum_{j=1}^{k-1} c_j$ *und* $\bar{a} := \sum_{j=1}^{k-1} a_j$

$$\bar{z} = \sum_{j=1}^{k-1} c_j + c_k \frac{\bar{b}}{a_k} = \bar{c} + \zeta_k \bar{b} = \bar{c} + \zeta_k (b - \bar{a})$$

Daher ist die ***Dantzigschranke***

$$U_D := \sum_{j=1}^{k-1} c_j + \left\lfloor c_k \frac{\bar{b}}{a_k} \right\rfloor = \bar{c} + \left\lfloor \zeta_k \bar{b} \right\rfloor = \bar{c} + \lfloor \zeta_k (b - \bar{a}) \rfloor \tag{1.9}$$

eine obere Schranke für den optimalen Zielfunktionswert von (KP_{01}).

Beweis: Die Behauptungen des Satzes liest man wie gesehen unmittelbar an den obigen allgemeinen Formeln für den Simplexschritt ab, der die kritische Variable in die Basis bringt. □

Man überprüft leicht, dass sich bei unserer konkreten Ausgabenstellung aus der Formel (4.1) $U_D = 38$ ergibt: Es ist $\bar{b} = 12 - 1 - 2 - 2 - 4 = 3$ und $\bar{b}/a_5 = 1/6$. Daraus folgt $\bar{z} = 8 + 8 + 6 + 10 + \frac{3}{6}12 = 38 = U$.

Interpretation: Die LP-Lösung gibt die Verteilung der Gewichtsanteile auf die Gegenstände frei, d.h. Anteile der a_i Gewichtseinheiten des $i-$ten Gegenstandes können denen des $j-$ten Gegenstandes hinzugefügt werden, wenn dies einen höheren Nutzen bringt und die Obergrenze 1 nicht sprengt. Insofern erhöht es den Nutzen einer zulässigen Lösung der LP-Relaxation, wenn Gewichtsanteile von Gegenständen mit höherem Index zu einem Gegenstand mit niederem Index umgeschaufelt werden, sofern dessen zulässiges Gewicht bisher nur (echt) bruchteilig ausgenutzt ist. Eine Optimallösung entsteht also, wenn alle Gewichtsanteile soweit wie möglich "nach links" verschoben werden. Man beachte:

$$\zeta_k (b - \bar{a}) = \frac{c_k}{a_k} (b - \bar{a}) = \frac{(b - \bar{a})}{a_k} c_k \quad \text{mit } 0 \leq \frac{(b - \bar{a})}{a_k} < 1$$

wegen $\bar{a} + a_k > b$.

Dantziglösung Eine erste *zulässige Lösung* x'des Problems (KP_{01}) erhält man offensichtlich durch Abrunden der Lösung $\bar{x}$.

$$x'_j = \begin{cases} 1 & \text{für } j = 1, ..., k-1 \\ 0 & \text{für } j = k, ..., n \end{cases}, \; j = 1, ..., n$$

Wir nennen sie **Dantziglösung.** Ihr Zielfunktionswert gibt eine untere Schranke für den Zielfunktionswert der optimalen Lösung der gegebenen Aufgabe an.

Es zeigt sich, dass eine untere und eine obere Schranke leicht berechnet werden können, man benötigt dazu nur den kritischen Index k. Daher stellt sich daher die Frage, wie man diesen einfach ermitteln kann.

Die folgende, auf Binärsuche basierende Prozedur ermittelt den kritischen Index k schnell und elegant:

Prozedur LUDD (Lower Upper Bound Dantzigschranke Dantziglösung)

Input: (KP_{01})

Output: kritischer Index k, Dantzigschranke U, Dantziglösung $\bar{x}$ mit Zielwert Z

begin

setze $n1 = 1$ und $n2 = n$

bestimme für $j = 1, ..., n$ die Summen

$$\bar{a}_j = \sum_{i=1}^{j} a_i \quad \text{und} \quad \bar{c}_j = \sum_{i=1}^{j} c_i \tag{1.10}$$

setze $kritisch = false$

solange $kritisch = false$ führe aus:

begin

setze $k = \left\lfloor \frac{n1+n2}{2} \right\rfloor$

ist $\bar{a}_{k-1} \leq b < \bar{a}_k$ so setze $kritisch = true$

sonst

ist $\bar{a}_{k-1} > b$ (% k zu groß!)

setze $n2 = k - 1$

sonst (% k zu klein!)

setze $n1 = k + 1$

end

setze $Z = \bar{c}_{k-1}$, $U = Z + \left\lfloor \frac{c_k}{a_k} (b - \bar{a}_{k-1}) \right\rfloor$ und $\bar{x}_j = 1$, falls $j < k$, $\bar{x}_j = 0$, sonst.

end

Die Berechnung der Summen $\bar{a}_j$ und $\bar{c}_j$ benötigt einen Aufwand vom $\mathcal{O}(n)$, wenn sie rekursiv durchgeführt wird. Sie solange-Schleife benötigt wegen der Binärsuche

eine Zeit von $\mathcal{O}(\log n)$, der Rest ist $\mathcal{O}(1)$. Insgesamt hat die Prozedur also eine Laufzeit von

$$\mathcal{O}(\log n) + \mathcal{O}(n) + \mathcal{O}(1) = \mathcal{O}(n)$$

Man überprüft leicht, dass die Anwendung der Prozedur LUDD auf die obige Aufgabenstellung die bereits "zu Fuß" berechneten Ergebnisse liefert:

Beispiel Wenden wir also die Prozedur LUDD auf unser Beispiel an.

Gegenstand j	1	2	3	4	5	6
Nutzen c_j	8	8	6	10	12	12
Gewicht a_j	1	2	2	4	6	10

mit $b = 12$

Wir starten mit der Bildung von $k = \left\lfloor \frac{n+1}{2} \right\rfloor = 3$ und setzen $kritisch = false$. Es ist $\bar{a}_2 = 3$ und $\bar{a}_3 = 5$ und daher $\bar{a}_3 < b$. Also ist k zu klein. Wir setzen daher $k = \left\lfloor \frac{k+1+n}{2} \right\rfloor = 5$. Nun gilt $\bar{a}_4 = 9$ und $\bar{a}_5 = 15$. Wegen $9 \leq 12 < 15$ ist damit der kritische Index gefunden, wir setzen $kritisch = true$ und verlassen die solange-Schleife. Wir finden wie gehabt $Z = 32$ und $U = 38$.

Beachte: Alternativ zu LUDD können natürlich auch die Indizes von links nach rechts durchgegangen werden und jeweils der zugehörige Gegenstand in den Rucksack gepackt werden. Wenn erstmalig der Gegenstand nicht mehr in den Rucksack passt, ist der kritische Index gefunden. Auch diese Vorgehensweise hat die Komplexität $\mathcal{O}(n)$. (Übungsaufgabe)

Reduzierung der Aufgabenstellung

Aus den bisherigen Beobachtungen können wir eine weitere *Konsequenz* für die Bearbeitung unseres Beispiels ziehen. Wir suchen eine Optimallösung von (P), wobei wir bereits eine zulässige Lösung von (P) mit Zielwert z' kennen. Daher können wir uns darauf beschränken, bzgl. der Aufgabe $\left(R\bar{P}\right)$ zulässige Lösungen zu suchen, die die Ganzzahligkeitsbedingung für $\bar{x}_1, ..., \bar{x}_6$ erfüllen und deren Zielfunktionswert *echt besser (größer)* ist als z'!

Man sieht dem Optimaltableau der Aufgabe $\left(R\bar{P}\right)$ unmittelbar an, dass bei $z' = 32$ die Variablen $\bar{x}_1$ und $\bar{x}_6$ in echt besseren zulässigen Lösungen (im zulässigen Bereich von $\left(R\bar{P}\right)$) nicht den Wert 1 annehmen können und daher wegen der Ganzzahligkeitsforderung den Wert 0 haben müssen. Dies reduziert die mit der B&B Methode zu bearbeitenden Aufgabe auf

$$\left(\tilde{P}\right) \qquad \begin{cases} \text{maximiere} & z + 4\bar{x}_2 + 2\bar{x}_3 + 2\bar{x}_4 + 2y = 38 \\ \text{s.d.} & \bar{x}_5 - \frac{2}{6}\bar{x}_2 - \frac{2}{6}\bar{x}_3 - \frac{4}{6}\bar{x}_4 + \frac{1}{6}y = \frac{3}{6} \\ & \bar{x} \in \{0,1\}^6,\ y \geq 0 \end{cases}$$

wobei wir $\bar{x}$ weiterhin als Vektor mit 6 Komponenten führen, aber $\bar{x}_1 = 0$ und $\bar{x}_6 = 0$ gesetzt haben.

Damit haben wir im Rahmen eines **Preprocessing** die zu lösende Aufgabenstellung deutlich verkleinern können. Überprüfen wir im Folgenden, ob dies so auch im Allgemeinfall der Aufgabe (KP_{01}) gelingen kann.

Im Allgemeinfall ergibt sich aus der transformierten LP-Relaxation $\left(RK\bar{P}_{01}\right)$ der Aufgabe (KP_{01}), dass eine Variable $\bar{x}_i = 0$ (d.h. $x_i = 1$) gesetzt werden darf, $i \in \{1, ..., k-1\}$, wenn das Setzen von $\bar{x}_i = 1$ zu einer oberen Schranke der Restaufgabe führt, die kleiner gleich z' ist, also wenn z.B.

$$\left\lfloor \sum_{j=1}^{k-1} c_j + \zeta_k \bar{b} - (c_i - \zeta_k a_i) \right\rfloor \leq z',$$

(vgl. (1.8), (1.7)), d.h. wenn mit $\bar{c}_{i-1} := \sum_{j=1}^{i-1} c_j$ und $\bar{a}_{i-1} := \sum_{j=1}^{i-1} a_j$, $i = 1, ..., n$,

$$\bar{c}_{k-1} - c_i + \lfloor \zeta_k \left(b - \bar{a}_{k-1} + a_i\right) \rfloor \leq z' \tag{1.11}$$

gilt. Dies ist insofern nachvollziehbar, als ein Optimaltableau der Aufgabe $\left(RK\bar{P}_{01}\right)$ vorliegt und eine Erhöhung einer beliebigen Nichtbasisvariablen zu einer Absenkung des Zielfunktionswertes führt, während das Tableau dual zulässig bleibt.

Genauso kann eine Variable $\bar{x}_i = 0$ gesetzt werden für $i \in \{k+1, ..., n\}$, wenn gilt

$$\bar{c}_{k-1} + c_i + \lfloor \zeta_k \left(b - \bar{a}_{k-1} - a_i\right) \rfloor \leq z' \tag{1.12}$$

Diese Vorgehensweise, durchgeführt über alle $i = 1, .., n$, hat eine Komplexität vom $\mathcal{O}(n)$, wenn die Größen $\bar{c}$ und $\bar{a}$ bei Kenntnis des break item vorab bekannt sind. Davon kann nach Anwendung der Prozedur LUDD ausgegangen werden.

Wir wenden die Formeln an auf unser Beispiel: Mit $b = 12$ lauten die Daten der Aufgabe

Gegenstand j	1	2	3	4	5	6
Nutzen c_j	8	8	6	10	12	12
Gewicht a_j	1	2	2	4	6	10

Für das break item $k = 5$ gilt insbesondere $\zeta_5 = 2$. Es ergibt sich dann wegen $\bar{c} = 32$, $\bar{a} = 9$ und $z' = 32$ nach den Formeln (1.11) und (1.12)

für $i = 1 : \bar{c} - c_1 + \lfloor \zeta_5 \left(b - \bar{a} + a_1\right) \rfloor = 32 - 8 + \lfloor 2\left(12 - 9 + 1\right) \rfloor = 32 \leq 32$

für $i = 2 : \bar{c} - c_2 + \lfloor \zeta_5 \left(b - \bar{a} + a_2\right) \rfloor = 32 - 8 + \lfloor 2\left(12 - 9 + 2\right) \rfloor = 34 > 32$

für $i = 3 : \bar{c} - c_3 + \lfloor \zeta_5 \left(b - \bar{a} + a_3\right) \rfloor = 32 - 6 + \lfloor 2\left(12 - 9 + 2\right) \rfloor = 36 > 32$

für $i = 4 : \bar{c} - c_4 + \lfloor \zeta_5 \left(b - \bar{a} + a_4\right) \rfloor = 32 - 10 + \lfloor 2\left(12 - 9 + 4\right) \rfloor = 36 > 32$

für $i = 6 : \bar{c} + c_6 + \lfloor \zeta_5 (b - \bar{a} - a_6) \rfloor = 32 + 12 + \lfloor 2 (12 - 9 - 10) \rfloor = 30 \leq 32$

Damit können $x_1 = 1$ und $x_6 = 0$ fixiert werden.

Es ist klar, dass diese Vorgehensweise mit den Formeln (1.11) und (1.12) auf jedes 0-1 Rucksackproblem angewendet werden kann und dass sie einen Aufwand von $\mathcal{O}(n)$ hat. Wendet man also zunächst die Prozedur LUDD an und anschließend die beschriebene Reduzierung, so hat man auch insgesamt einen Aufwand von $\mathcal{O}(n)$.

Verschärfte Reduzierung Generell kann diese Art der Reduzierung einer 0-1 Knapsackaufgabe (KP_{01}) mit Hilfe der Dantzigschranke noch verschärft werden, wenn eine Verschlechterung der Laufzeit dabei in Kauf genommen wird. Die Idee dazu geht auf INGARGIOLA-KORSH (1973) zurück und wurde von MARTELLO UND TOTH (1988) verbessert.

Ausgangspunkt sind die Formeln (1.11), (1.12). Sie arbeiten mit dem kritischen Index der Aufgabe (KP_{01}). Wird $x_i = 0$ gesetzt, $i \in \{1, ..., k-1\}$, so entsteht eine neue Rucksack-Aufgabe (KP_i) mit den gleichen Daten, nur dass die Variable x_i verschwunden ist. Der kritische Index k' dieser neuen Aufgabe kann neu berechnet werden und liegt *potentiell rechts* von k. Berechnet man die Dantzigschranke der neuen Aufgabe, so ergibt sich eine sehr ähnliche Formel zu (1.11) (Gleichheit für $k = k'$)

$$\bar{c}_{k'-1} - c_i + \lfloor \zeta_{k'} (b - \bar{a}_{k'-1} + a_i) \rfloor \leq z', \tag{1.13}$$

die in der Regel schärfer ist.

Genau so kann man nach Festlegung von $x_i = 1$ für ein $i \in \{k+1, ..., n\}$ die neu entstandene Knapsack Aufgabe (KP_i) betrachten, die einen eigenen kritischen Index k'' besitzt, der *potentiell links* von k liegt und im Vergleich zur Formel (1.12) zu einer potentiell schärferen Schranke führt (Gleichheit für $k = k''$)

$$\bar{c}_{k''-1} + c_i + \lfloor \zeta_{k''} (b - \bar{a}_{k''-1} - a_i) \rfloor \leq z'. \tag{1.14}$$

Diese Formeln bilden die Grundlage für die folgende **Reduzierungs-Prozedur** für das 0-1-Knapsackproblem (KP_{01}). Sie bestimmt eine Menge JF_1 von Variablen, die von vorne herein auf den Wert 1 und eine Menge JF_0 von Variablen, die auf den Wert 0 gesetzt werden dürfen und verwendet dazu z.B. die Prozedur LUDD. Zusätzlich wird bei der Prüfung, ob $x_j = 0$ oder $x_j = 1$ gesetzt werden kann, auch $j = k$ in beiden Fällen zugelassen, dass sich die Mengen JF_0 und JF_1 überschneiden können. In diesem Fall entsteht ein Widerspruch und es ist die beste bislang bekannte zulässige Lösung optimal.

Ein Unterschied dieser Prozedur zu der zuerst verwendeten Vorgehensweise sind nicht nur die schärferen Ausschlussformel, sondern auch die permanente *potentielle Anhebung der unteren Schranke* L und eine *potentielle Verbesserung der oberen*

Schranke U. Erst zum Schluss wird mit der besten aktuellen unteren Schranke geprüft, welche Variablen fixiert werden können, sodass die beste bis dahin bekannte untere Schranke zum Einsatz kommt

Prozedur DDR (Dantzig Dantzig Reduzierung)

Input: (KP_{01}), kritischer Index k, obere Schranke U, zulässige Lösung x mit Zielwert L, Zielwert $\bar{c}$ der Dantziglösung $\bar{x}$.

Output: Mengen JF_1 und JF_0 von Indizes, die bei der weiteren Suche auf 1 bzw 0 fixiert werden können, eine potentiell verbesserte untere Schranke L und eine potentiell verbesserte obere Schranke U.

1. **Start:** Setze $JF_0, JF_1 := \emptyset$. Setze $\bar{U} = \bar{c}$.

2. **Schritt 1:** Für $i = 1, ..., k$ führe aus:

   Berechne die Dantziglösung x^i mit Zielwert z^i und die Dantzigschranke U_0^i der Aufgabe (KP_i), die entsteht, wenn $x_i = 0$ gesetzt wird.

   Gilt $U_0^i > \bar{U}$ und $i \neq k$, setze $\bar{U} = U_0^i$.

   Gilt $z^i > L$, setze $x := x^i$ und $L := z^i$. Gilt dann $U = L$, **stopp**, x ist optimal.

3. **Schritt 2:** Für $i = k, ..., n$ führe aus:

   Berechne die Dantziglösung x^i mit Zielwert z^i und die Dantzigschranke U_1^i der Aufgabe (KP_i), die entsteht, wenn $x_i = 1$ gesetzt wird.

   Gilt $U_1^i > \bar{U}$ und $i \neq k$, setze $\bar{U} = U_1^i$.

   Gilt $z^i > L$, setze $x := x^i$ und $L := z^i$. Gilt dann $U = L$, **stopp**, x ist optimal.

4. **Schritt 3:** Setze $U = \min\left\{U, \bar{U}\right\}$. Gilt $U = Z$, **stopp**, x ist optimal.

   Für $i = 1, ...k$ führe aus: Gilt $U_0^i \leq L$, setze $JF_1 = JF_1 \cup \{i\}$

   Für $i = k, ...n$ führe aus: Gilt $U_1^i \leq L$, setze $JF_0 = JF_0 \cup \{i\}$

   Gilt danach $JF_0 \cap JF_1 \neq \emptyset$, so ist x Optimallösung.

Satz 1.2 *Die Prozedur DDR arbeitet korrekt.*

Beweis: Die Festsetzung von JF_0 und JF_1 erfolgt per Auslotung: weil die berechneten oberen Schranken U^i zu schlecht sind im Vergleich mit dem Zielwert der best-bekannten zulässigen Lösung x, kann die probeweise vorgenommene Fixierung der Variablen in der Aufgabe (KP_i) nicht zu einer besseren Lösung als x passen, daher kann die Fixierung genau andersherum vorgenommen werden, wenn eine bessere Lösung als x gesucht wird.

Zeigt sich zum Schluss, dass k sowohl in JF_0 als auch in JF_1 aufgenommen wurde, so gilt $x_k = 0 = 1$, was nicht geht, also gibt es keine bessere Lösung als x.

Bleibt nur noch zu klären, warum U in der beschriebenen Weise aktualisiert werden kann. Dies ist nicht schwer einzusehen: denn entweder ist eine vorgegebene zulässige Lösung x' die Dantziglösung $\bar{x}$ von (KP_{01}) oder sie weicht von dieser *in mindestens einer Komponente* i ab. Dabei kann sie nicht *nur* in der Komponente k abweichen, da diese Lösung nicht einmal zu (RKP_{01}) zulässig wäre. Wenn x' also nicht mit der Dantziglösung $\bar{x}$ übereinstimmt, dann weicht sie in mindestens einer Komponente $i \neq k$ von dieser ab. Mithin ist der zu x' gehörige Zielwert z' kleiner oder gleich einer der berechneten oberen Schranken U^i, $i \neq k$ (oder gleich $\bar{c}_{k-1}$). □

Die Prozedur hat eine Laufzeit von $\mathcal{O}(n \log n)$, wenn in den Schritten 1 und 2 $(n+1)$-fach die Prozedur LUDD verwendet wird und zuvor die Summen (1.10) anfangs einmalig berechnet wurden.

Bemerkung: In der Praxis können diese Anwendungen der Prozedur LUDD noch verkürzt ablaufen. In Schritt 1 z. B. wird eine Variable links von k auf null gesetzt. Dies bedeutet, dass der kritische Index gegenüber der Originalaufgabe potentiell nach rechts rutscht. Man braucht daher auch nur rechts von k zu suchen. Ggf ist es hier sogar praktisch günstiger, direkt von k fortlaufend in den Indizes nach rechts zu suchen (anstelle der Binärsuche), da man annehmen darf, dass der neue kritische Index nicht allzuweit von k entfernt liegt. Entsprechendes gilt für Schritt 2 der Prozedur DDR.

Ferner ist zu bemerken, dass die Restaufgabe nach der Reduzierung, für die man als untere Schranke $(L+1)$ ansetzen darf, nicht unbedingt lösbar sein muss: in diesem Falle ist die zuletzt bekannte beste zulässige Lösung Optimallösung. Auch muss diese Restaufgabe nicht unbedingt vom Standardformat sein: Man kann alle Variablen zu null fixieren, deren Gewicht die Restkapazität echt übersteigt. Ferner können vorab alle zulässigen Lösungen getestet werden, bei denen die Restkapazität mit einem der noch freien Gewichte übereinstimmt.

Beispiel Zur Erläuterung wenden wir die Prozedur DDR auf unsere Aufgabe (P) an, führen dabei aber abweichend die Maximierung der U^i erst im Nachhinein durch.

Mit $b = 12$ lauten die Daten wieder

Gegenstand j	1	2	3	4	5	6
Nutzen c_j	8	8	6	10	12	12
Gewicht a_j	1	2	2	4	6	10

(1) Es ergibt sich wie bekannt $k = 5$, $x^T = (1, 1, 1, 1, 0, 0)$ mit $L = 32$, $U = 38$ und $\bar{U} = 32$

(2) für $x_1 = 0$ ergibt sich als kritischer Index $k' = 5$
mit $x^{1T} = (0, 1, 1, 1, 0, 0)$ und $z^1 = 24$ und $U_0^1 = 24 + \left\lfloor \frac{12}{6} 4 \right\rfloor = 32$
für $x_2 = 0$ ergibt sich als kritischer Index $k' = 5$
mit $x^{2T} = (1, 0, 1, 1, 0, 0)$ und $z^2 = 24$ und $U_0^2 = 24 + \left\lfloor \frac{12}{6} 5 \right\rfloor = 34$. Setze $\bar{U} = 34$
für $x_3 = 0$ ergibt sich als kritischer Index $k' = 5$
mit $x^{3T} = (1, 1, 0, 1, 0, 0)$ und $z^3 = 26$ und $U_0^3 = 26 + \left\lfloor \frac{12}{6} 5 \right\rfloor = 36$. Setze $\bar{U} = 36$
für $x_4 = 0$ ergibt sich als kritischer Index $k' = 6$
mit $x^{4T} = (1, 1, 1, 0, 1, 0)$ und $z^4 = 34$ und $U_0^4 = 34 + \left\lfloor \frac{12}{10} 1 \right\rfloor = 35$
setze also neu $x^T = (1, 1, 1, 0, 1, 0)$, $L = 34$
für $x_5 = 0$ ergibt sich als kritischer Index $k' = 6$
mit $x^{5T} = (1, 1, 1, 1, 0, 0)$ und $z^5 = 32$ und $U_0^5 = 32 + \left\lfloor \frac{12}{10} 3 \right\rfloor = 35$

(3) für $x_5 = 1$ ergibt sich als kritischer Index $k'' = 4$
mit $x^{5T} = (1, 1, 1, 0, 1, 0)$ und $z^5 = 34$ und $U_1^5 = 34 + \left\lfloor \frac{10}{4} 1 \right\rfloor = 36$
für $x_6 = 1$ ergibt sich als kritischer Index $k'' = 2$
mit $x^{6T} = (1, 0, 0, 0, 0, 1)$ und $z^6 = 20$ und $U_1^6 = 20 + \left\lfloor \frac{8}{2} 1 \right\rfloor = 24$

(4) wegen $L = 34$ folgt $JF_1 = \{1, 2\}$ und $JF_0 = \{6\}$. Ferner gilt $U = \min\left\{U, \bar{U}\right\} = 36$

Die Prozedur DDR liefert also ein verbessertes Ergebnis gegenüber unserer "Reduzierung zu Fuß". Dies liegt insbesondere daran, dass im Laufe der Anwendung der Prozedur eine neue bessere zulässige Lösung entdeckt wurde. Es zeigt sich aber auch, dass die einzelnen oberen Schranken schärfer sind.

1.2.2 Die Branch and Bound Methode

Die Branch and Bound Methode arbeitet nach dem Prinzip "Teile und Herrsche". Eine vorliegende, schwer zu lösende, gemischt ganzzahlige Optimierungsaufgabe wird immer wieder in mehrere Teilaufgaben "aufgeteilt", bis die einzelnen Teilaufgaben "beherrschbar" geworden sind.

Beschreibung der Methode

Gegeben sei also eine gemischt ganzzahlige Optimierungsaufgabe

$$(P) \quad \begin{cases} \text{minimiere} & f(x) \\ \text{s.d.} & x \in X \end{cases}$$

wobei f eine vorgegebene Zielfunktion, X eine vorgegebene Teilmenge des $\mathbf{R}^n$ sind und der Vektor x der Unbekannten in einigen Komponenten ganzzahlig gefordert ist.

X besteht also aus mehreren Zusammenhangskomponenten. (Wir unterstellen, dass die Aufgabe "leicht lösbar" ist, wenn X nur aus einer Zusammenhangskomponente besteht.) Wir wollen eine aus (P) abgeleitete Aufgabe (P') eine **Testaufgabe** nennen, wenn (P') unter Beibehaltung oder Abschwächung der Zielfunktion durch Einschränkung des zulässigen Bereichs B von (P) auf eine Vereinigung B' von Zusammenhangskomponenten von B entsteht. Die Testaufgabe (P') heißt dabei **echt**, wenn ihr zulässiger Bereich B' echte Teilmenge von B ist. Bildet man entsprechend Testaufgaben aus einer Testaufgabe (P'), so sprechen wir von aus (P') **abgeleiteten Testaufgaben**.

Die Branch and Bound Methode untersucht der Reihe nach Testaufgaben von (P) und führt dazu eine Liste $LIST$ "*noch zu verzweigender*" Testaufgaben. Sie lässt sich wie folgt formulieren:

Branch and Bound Methode

(0) **Start:** Anfänglich setze $LIST = \{(P)\}$. Sei z der Zielfunktionswert einer bekannten (zulässigen) Lösung x von (P) oder $z = \infty$, wenn keine zulässige Lösung bekannt ist.

(1) **Auswahl:** Ist $LIST$ leer, **stopp.**

Andernfalls wähle eine Testaufgabe (P') aus $LIST$ und entferne sie aus $LIST$.

(2) **Verzweigung:** Wähle eine endliche Menge von aus (P') abgeleiteten verschiedenen Testaufgaben $(P'_1), ..., (P'_s)$, deren zulässige Bereiche sich zum zulässigen Bereich von (P') vereinigen.

(3) **Untersuchung:** Für $i = 1, ..., s$ führe aus:

(a) Ist $s = 1$, so löse die Aufgabe (P'_1). Ansonsten untersuche die Testaufgabe (P'_1) mit geeigneter Methode.

(b) Ist aus der Untersuchung eine *optimale* Lösung x' der Aufgabe (P'_i) bekannt und ist der Zielfunktionswert z' dieser Lösung kleiner als z, so setze $z := z'$ und $x := x'$. Gehe zum nächsten i über.

(c) Ist aus der Untersuchung bekannt, dass die Zielfunktion auf (P'_i) unbeschränkt ist, **stopp** (die Aufgabe (P) ist nicht lösbar)

(d) Ist aus der Untersuchung bekannt, dass (P'_i) keine zulässige Lösung mit besserem Zielfunktionswert als z enthalten kann, so gehe zum nächsten i über.

(e) Andernfalls nehme die Testaufgabe (P'_i) in $LIST$ auf und gehe zum nächsten i über.

Gehe zu (1).

Endet die Anwendung der Methode auf eine Aufgabe (P) nach endlich vielen Schritten, so ist die zuletzt gespeicherte zulässige Lösung x Optimallösung von (P) mit Zielfunktionswert z oder es existiert keine zulässige Lösung, wenn keine solche gefunden wurde (es bleibt bei $z = \infty$).

Besteht der zulässige Bereich der gegebenen Aufgabe anfänglich aus endlich vielen Zusammenhangskomponenten und wird sichergestellt, dass oft genug in echte Testaufgaben verzweigt wird, so kann bewiesen werden, dass eine Anwendung der Methode immer nach endlich vielen Schritten abbricht.

Bezeichnung: Wir nennen eine untersuchte Testaufgabe **ausgelotet**, wenn für sie die Kriterien (b) oder (d) im Baustein (3) des Verfahrens erfüllt sind. Die Prüfung einer Testaufgabe, ob sie eine bessere zulässige Lösung als die aktuell bekannte enthalten kann, wollen wir **Ausloten** der Testaufgabe nennen.

Auslotung betreibt man im allgemeinen über das Ermitteln einer **unteren Schranke** s für den optimalen Zielfunktionswert der zu untersuchenden Testaufgabe. Ist z der beste Zielfunktionswert einer aktuell bekannten zulässigen Lösung von (P) und damit eine **obere Schranke** für den Zielfunktionswert einer Optimallösung von (P), so gilt:

Ist $s \geq z$, so kann die vorliegende Testaufgabe keine bessere zulässigen Lösung enthalten als die beste bereits bekannte. Die Testaufgabe ist damit ausgelotet.

Eine untere Schranke berechnet man in der Regel durch Abschätzen des Zielfunktionswertes einer geeignet gewählten **Relaxation** der Testaufgabe, d.h. durch Untersuchung der zulässigen Lösungen einer Abschwächung des zulässigen Bereichs der Testaufgabe bei beibehaltener oder abgeschwächter Zielfunktion. Ist die Relaxation mit einem bekannten Verfahren einfach lösbar, so ist natürlich der optimale Zielfunktionswert über dem ausgeweiteten zulässigen Bereich eine geeignete untere Schranke.

Anwendung der B&B Methode

Branch and Bound Ansätze zur Lösung von (KP_{01}) gibt es viele in der Literatur, fast alle beruhen auf dem Dakin-Ansatz. Zu nennen sind: Kolesar (1967), Greenberg und Hegerich (1970), Horowitz und Sahni (1974), Barr und Ross (1975), Laurière (1978), Lageweg und Lenstra (1972), Guinard und Spielberg (1972), Fayard und Plateau (1975), Veliev und Mamedov (1981), Nauss (1976), Martello und Todt (1977), Deo und Kowalik (1983), Suhl (1978) und Zoltners (1978).

Zur Einstimmung wenden wir die B&B Methode auf die bereits reduzierte Aufgabe

$$\left(\tilde{P}\right) \qquad \begin{cases} \text{maximiere} & z + 4\bar{x}_2 + 2\bar{x}_3 + 2\bar{x}_4 + 2y = 38 \\ \text{s.d.} & \bar{x}_5 - \frac{2}{6}\bar{x}_2 - \frac{2}{6}\bar{x}_3 - \frac{4}{6}\bar{x}_4 + \frac{1}{6}y = \frac{3}{6} \\ & \bar{x} \in \{0,1\}^6,\ y \geq 0 \end{cases}$$

an. Dabei wollen wir die Aufgabenstellung erneut reduzieren, wenn sich die Gelegenheit dazu ergibt. Außerdem sollen die Verzweigungsvariablen willkürlich gewählt werden und die Untersuchung der Testaufgaben, wenn möglich, "per Augenschein" vorgenommen werden.

(0) Zunächst setzen wir $\bar{x} = (0,0,0,0,0,0)^T$, $z = 32$ und $LIST = \left\{\left(\tilde{P}\right)\right\}$.

(1) Wir wählen $\left(\tilde{P}\right)$ aus $LIST$, dann ist $LIST = \emptyset$.

(2) Wir verzweigen die Aufgabe $\left(\tilde{P}\right)$ in

(P_1) : $\left(\tilde{P}\right)$ mit Zusatzrestriktion $\bar{x}_4 = 1$

(P_2) : $\left(\tilde{P}\right)$ mit Zusatzrestriktion $\bar{x}_4 = 0$

(3) Zu untersuchen ist zunächst die Aufgabe

$$(P_1) \qquad \begin{cases} \text{maximiere} & z + 4\bar{x}_2 + 2\bar{x}_3 + 2y = 36 \\ \text{s.d.} & \bar{x}_5 - \frac{2}{6}\bar{x}_2 - \frac{2}{6}\bar{x}_3 + \frac{1}{6}y = \frac{7}{6} \\ & \bar{x} \in \{0,1\}^6,\ y \geq 0 \end{cases}$$

Wir erkennen, dass sich für $y = 1$ die zulässige Lösung $\bar{x}_2 = 0,\ \bar{x}_3 = 0,\ \bar{x}_5 = 1$ mit Zielfunktionswert $s = 34$ ergibt und speichern diese zulässige Lösung unter $\bar{x}' = (0,0,0,1,1,0)^T$ mit $s' = 34$. Setze also $\bar{x} = \bar{x}'$ und $z = z'$ als neue beste zulässige Lösung fest.

Diese bessere zulässige Lösung gestattet uns erneut, die zu bearbeitende Aufgabe $\left(\tilde{P}\right)$zu reduzieren: die Variable $\bar{x}_2$ kann auf den Wert $\bar{x}_2 = 0$ gesetzt werden, da mit $\bar{x}_2 = 1$ der bisher beste Zielfunktionswert einer zulässigen Lösung von $\left(\tilde{P}\right)$ nicht überschritten werden kann.

Also reduziert sich auch die aktuell zu untersuchende Aufgabe auf

$$(P_{1red}) \qquad \begin{cases} \text{maximiere} & z + 2\bar{x}_3 + 2y = 36 \\ \text{s.d.} & \bar{x}_5 - \frac{2}{6}\bar{x}_3 + \frac{1}{6}y = \frac{7}{6} \\ & \bar{x} \in \{0,1\}^6,\ y \geq 0 \end{cases}$$

Diese ist leicht untersucht: Für $\bar{x}_3 = 0$ ergibt sich die bereits bekannte zulässige Lösung, für $\bar{x}_3 = 1$ kann sich schon vom Zielfunktionswert her keine bessere Lösung ergeben. Daher ist (P_1) ausgelotet.

Zu untersuchen ist nun die Aufgabe

$$(P_{2red}) \qquad \begin{cases} \text{maximiere} & z + 2\bar{x}_3 + 2y = 38 \\ \text{s.d.} & \bar{x}_5 - \frac{2}{6}\bar{x}_3 + \frac{1}{6}y = \frac{3}{6} \\ & \bar{x} \in \{0,1\}^6,\ y \geq 0 \end{cases}$$

Diese Aufgabe hat für $\bar{x}_3 = 0$ die optimale Lösung $y = 3$ mit Zielfunktionswert $z = 32$, welche schlechter ist als der beste bereits bekannte und für $\bar{x}_3 = 1$ die optimale Lösung $y = 5$ mit Zielfunktionswert $s = 26$, welcher auch zu schlecht ist. Also ist auch (P_2) ausgelotet.

(1) Da nun $LIST = \emptyset$ gilt, ist die Anwendung der Methode bereits beendet.

Ergebnis ist: $x = (1,1,1,0,1,0)^T$ mit $z = 34$. □

Verfahren von Horowitz und Sahni Bei der Untersuchung der Testaufgaben in unserem Beispiel haben wir Informationen verwendet, die man durch bloßes Hinsehen erkannte. Das kann ein Rechner in der Regel nicht. Im folgenden formulieren

wir deshalb eine B&B Methode für das 0-1-Knapsackproblem (KP_{01}) in implementierbarem Pseudocode. Diese kommt ohne Variablentransformation aus und im Unterschied zum Beispiel besteht die Untersuchung der Testaufgabe dabei nur aus der Berechnung einer oberen Schranke nach der Dantzigformel, zulässige Lösungen werden nur entdeckt, wenn sämtliche Variablen einen festen Wert bekommen haben.

Wir wollen die B&B-Methode als Prozedur notieren. Diese geht in der Grundversion auf HOROWITZ UND SAHNI (1974) zurück. Sie setzt nach abfallenden Nutzen-Gewichts-Quotienten geordnete Daten voraus und verwendet intern, aber nicht explizit *LIFO* als Auswahlprozedur aus $LIST$. Als Verzweigungsvariable wird immer die noch nicht fixierte Variable mit dem kleinsten Index verwendet. Sie geht im Prinzip genau so vor, wie wir das bei dem bisherigen Beispiel gemacht haben, verwendet allerdings keine zwischenzeitliche Reduzierung. Der Vorteil dieses Verfahrens ist die elegante Algorithmisierung.

Folgende Notationen werden verwendet:

$\hat{x}$ = laufende Lösung

$\hat{z}$ = Zielwert der laufenden Lösung $\left(= \sum_{j=1}^{n} c_j \hat{x}_j\right)$

$\hat{b}$ = laufende Restkapazität $\left(= b - \sum_{j=1}^{n} a_j \hat{x}_j\right)$

x = beste bisher bekannte Lösung

z = Zielwert der besten bislang bekannten Lösung $\left(= \sum_{j=1}^{n} c_j x_j\right)$

r = kritischer Index

u = Dantzigschranke

f = Flagge aus $\{0, 1\}$ (wird verwendet, um die erste lokale obere Schranke global zu machen)

Prozedur HSV (Horowitz Sahni Verfahren)

Input: (KP_{01}), zulässige Lösung $\bar{x}$ mit Zielwert $\bar{z}$, obere Schranke U.

Output: Optimallösung x mit Zielfunktionswert z.

1. **Start** setze $z = \bar{z},\ \hat{z} = 0,\ \hat{b} = b,\ c_{n+1} := 0,\ a_{n+1} := \infty,\ j := 1,\ f = 0$
2. **lokale obere Schranke**
   bestimme $r = \min\left\{i : \sum_{k=j}^{i} a_k > \hat{b}\right\}$ `% Hier ist` $a_{n+1} = \infty$ `wichtig!`
   $u := \sum_{k=j}^{r-1} c_k + \left\lfloor \left(\hat{b} - \sum_{k=j}^{r-1} a_k\right) \frac{c_r}{a_r} \right\rfloor$ `% Hier ist` $c_{n+1} = 0$ `wichtig!`

ist $z \geq \hat{z} + u$, dann gehe zu 5.

ist $f = 0$ und $\hat{z} + u < U$, setze $U = \hat{z} + u$ und $f = 1$

3. **Vorwärtsschritt**

   solange $a_j \leq \hat{b}$ gilt, führe aus:

   $\hat{b} := \hat{b} - a_j, \quad \hat{z} := \hat{z} + c_j, \quad \hat{x}_j := 1, \quad j := j + 1$ `% Letztes` j `kritisch!`

   ist $j \leq n$, so setze $\hat{x}_j := 0, \quad j := j + 1$

   ist $j < n$, gehe zu 2.

   ist $j = n$, gehe zu 3.

4. **Aktualisierung**

   ist $\hat{z} > z$, so führe aus

   $z := \hat{z}$

   für $k = 1$ bis n setze $x_k := \hat{x}_k$

   Ist $z = U$, **stopp**

   setze $j := n$

   ist $\hat{x}_n = 1$ setze $\hat{b} := \hat{b} + a_n, \quad \hat{z} := \hat{z} - c_n, \quad \hat{x}_n := 0$ `% Seitwärtsschritt`

5. **Rückwärtsschritt**

   bestimme sofern möglich $i = \max\{k < j : \hat{x}_k = 1\}$, sonst **stopp**

   setze $\hat{b} := \hat{b} + a_i, \quad \hat{z} := \hat{z} - c_i, \quad \hat{x}_i := 0, \quad j := i + 1$

   gehe zu 2.

Zum besseren Verständnis wollen wir die *Prozedur HSV* auf unser Beispiel anwenden. Nach *erfolgter Reduzierung* sind bereits $x_1 = 1$ und $x_6 = 0$ fixiert. Wenden wir die Prozedur auf diese zu $\left(\tilde{P}\right)$ äquivalente Aufgabe an, so entsteht eine unmittelbare Vergleichsmöglichkeit zur bisherigen Rechnung.

Gegeben ist also das 0-1−Knapsack Problem mit den Daten

Gegenstand j	2	3	4	5
Nutzen c_j	8	6	10	12
Gewicht a_j	2	2	4	6

mit $b = 11$ und "Vornutzen" $\hat{z} = 8$

1. Zunächst setzen wir wie vorgeschrieben $z = 0,\ U = \infty,\ \hat{z} = 8,\ \hat{b} = b = 11,\ n = 5, c_6 := 0,\ a_6 := \infty,\ j := 2.$

2. Wir bestimmen den kritischen Index $r = 5$ und die Dantzigschranke $u = 30$. Es ist $z = 0 < 8 + 30 = \hat{z} + u$. Setze $U = \hat{z} + u = 38$.

3. Für $j = 2$ setze $\hat{b} = \hat{b} - a_2 = 11 - 2 = 9,\ \hat{z} = \hat{z} + c_2 = 8 + 8 = 16,\ \hat{x}_2 = 1,\ j = 3$

   Für $j = 3$ setze $\hat{b} = \hat{b} - a_3 = 9 - 2 = 7,\ \hat{z} = \hat{z} + c_3 = 16 + 6 = 22,\ \hat{x}_3 = 1,\ j = 4$

   Für $j = 4$ setze $\hat{b} = \hat{b} - a_4 = 7 - 4 = 3,\ \hat{z} = \hat{z} + c_4 = 22 + 10 = 32,\ \hat{x}_4 = 1,\ j = 5$

   Wegen $a_5 = 6 > 3 = \hat{b}$, prüfe nun:

   Es ist $j = 5 \leq 5 = n$, daher setze $\hat{x}_5 = 0$ und $j = 6$.

   Es ist $j > n$.

4. Es ist $\hat{z} = 32 > 0 = z$, also setze $z = \hat{z} = 32$, und $x^T = (1, 1, 1, 1, 0, 0)$.

   Setze $j = 5$. Es ist $\hat{x}_n = \hat{x}_5 = 0$.

5. Es ist $i = \max\{k < 5 : \hat{x}_k = 1\} = 4$.

   Setze $\hat{b} = \hat{b} + a_4 = 3 + 4 = 7,\ \hat{z} = \hat{z} - c_4 = 32 - 10 = 22,\ \hat{x}_4 = 0,\ j = 5$.

2. Es ist $r = \min\left\{i : \sum_{k=5}^{i} a_k > 7\right\} = 6,\ u = \sum_{k=j}^{r-1} c_k + \left\lfloor \left(\hat{b} - \sum_{k=j}^{r-1} a_k\right) \frac{c_r}{a_r} \right\rfloor = 12 + \left\lfloor 1 \cdot \frac{0}{\infty} \right\rfloor = 12$

   Es ist $z = 32 < 22 + 12 = 34$

3. Es ist $a_5 = 6 \leq 7 = \hat{b}$, also setze $\hat{b} = 7 - 6 = 1,\ \hat{z} = 22 + 12 = 34,\ \hat{x}_5 = 1,\ j = 6$.

   Es ist $j = 6 > 5 = n$

4. Es ist $\hat{z} = 34 > 32 = z$, also setze $z = 34,\ x^T = (1, 1, 1, 0, 1, 0)$ und $j = 5$.

   Es ist $\hat{x}_5 = 1$, daher setze $\hat{b} = \hat{b} + a_5 = 1 + 6 = 7,\ \hat{z} = \hat{z} - c_n = 34 - 12 = 22,\ \hat{x}_5 = 0$

5. Es ist $i = \max\{k < 5 : \hat{x}_k = 1\} = 3$.

   Setze $\hat{b} = \hat{b} + a_3 = 7 + 2 = 9,\ \hat{z} = \hat{z} - c_3 = 22 - 6 = 16,\ \hat{x}_3 = 0,\ j = 4$.

2. Es ist $r = \min\left\{i : \sum_{k=4}^{i} a_k > 9\right\} = 5,\ u = \sum_{k=j}^{r-1} c_k + \left\lfloor \left(\hat{b} - \sum_{k=j}^{r-1} a_k\right) \frac{c_r}{a_r} \right\rfloor = 10 + \left\lfloor 5 \cdot \frac{12}{6} \right\rfloor = 20$

   Es ist $z = 34 < 16 + 20 = 36$

3. Es ist $a_4 = 4 \leq 9 = \hat{b}$, also setze $\hat{b} = 9 - 4 = 5,\ \hat{z} = 16 + 10 = 26,\ \hat{x}_4 = 1,\ j = 5$.

   Es ist $j = 5 \leq 5$, also setze $\hat{x}_5 = 0,\ j = 6$.

3. Es ist $j > n$.

4. Es ist $\hat{z} = 26 \leq z = 34$, also setze $j = 5$. Es ist nicht $\hat{x}_5 = 1$.

5. Es ist $i = \max\{k < 5 : \hat{x}_k = 1\} = 4$. Setze $\hat{b} = 5 + 4 = 9,\ \hat{z} = 26 - 10 = 16,$ $\hat{x}_4 = 0,\ j = 5$.

2. Es ist $r = \min\left\{i : \sum_{k=5}^{i} a_k > 9\right\} = 6,\ u = \sum_{k=j}^{r-1} c_k + \left\lfloor \left(\hat{b} - \sum_{k=j}^{r-1} a_k\right) \frac{c_r}{a_r} \right\rfloor =$ $12 + \left\lfloor 3 \cdot \frac{0}{\infty} \right\rfloor = 12$
   Es ist $z = 34 > 16 + 12 = 28$

5. Es ist $i = \max\{k < 5 : \hat{x}_k = 1\} = 2$. Setze $\hat{b} = 9 + 2 = 11,\ \hat{z} = 16 - 8 = 8,$ $\hat{x}_2 = 0,\ j = 3$.

2. Es ist $r = \min\left\{i : \sum_{k=3}^{i} a_k > 11\right\} = 5,\ u = \sum_{k=j}^{r-1} c_k + \left\lfloor \left(\hat{b} - \sum_{k=j}^{r-1} a_k\right) \frac{c_r}{a_r} \right\rfloor =$ $6 + 10 + \left\lfloor 5\frac{12}{6} \right\rfloor = 26$
   Es ist $z = 34 \geq 8 + 26 = 34$.

5. Es ist kein $i = \max\{1 < k < 3 : \hat{x}_k = 1\}$ mehr bestimmbar. **Stopp**

Es wurde also die uns bereits bekannte Optimallösung $z = 34,\ x^T = (1, 1, 1, 0, 1, 0)$ bestimmt. Die folgende **Skizze** zeigt den Ablauf des Verfahrens:

Ein Verfahren mit Breitensuche

Horowitz und Sahni verwenden Tiefensuche (LIFO) und verzweigen bei gewählter Verzweigungsvariablen j immer zuerst zu $x_j = 1$ und dann zu $x_j = 0$. Sie beginnen dabei mit $x_1 = 1$.

Alternativ dazu kann die Tiefensuche mit einer anderen Auswahl der Verzweigungsvariablen verbunden werden: Als Verzweigungsvariable wird abgegriffen eine Variable, die in der Optimallösung der LP-Relaxation einen *nichtganzzahligen* Wert zugewiesen bekommt. Wie wir wissen, gibt es davon genau eine, die kritische Variable. Verzweigungsvariable wird also *immer die kritische Variable*. Diese Strategie wurde von Greenberg und Hegerich vorgeschlagen (1970). Diese verzweigen die Verzweigungsvariable x_k immer zuerst zu $x_k = 0$ und dann zu $x_k = 1$. Auch sie verwenden LIFO zum Durchwndern des Entscheidungsbaums und haben so wenig Verwaltungs- und Speicheraufwand.

Eine weitere Alternative besteht darin, *Breitensuche* zu verwenden, d.h. eine Liste $LIST$ noch zu verzweigender Testaufgaben zu führen und auf diese gemäß *höchster oberer Schranke* oder nach *FIFO* abzugreifen. Wir wollen eine solche Variante näher betrachten.

Zur Ausgestaltung des so beschriebenen Verfahrens wollen wir eine spezielle Notation für die in $LIST$ zu speichernden Testaufgaben (TP) einführen:

$$(TP): \quad \left(U \,\middle|\, D, \hat{b}, r \parallel j_1, j_2, ...\right)$$

Diese Notation beschreibt die folgenden Testaufgabe: Betrachtet wird (KP_{01}) mit Zusatzrestriktionen $x_{|j_i|} = 1$, wenn $j_i > 0$ und $x_{|j_i|} = 0$, wenn $j_i < 0$. Zugehörig zu dieser Aufgabe ist der kritische Index r, die Restkapazität $\hat{b}$, der Nutzen der Dantziglösung, D, und eine obere Schranke U, in der Regel die Dantzigschranke. Die Liste der Indizes kann leer sein oder bis zu n Indizes umfassen.

Wird die Aufgabe (TP) über die Verzweigungsvariable r verzweigt, so werden die Söhne

$$\begin{aligned} (TP') &: \quad (TP) \quad \text{mit Zusatzrestrikion } x_r = 1 \\ (TP'') &: \quad (TP) \quad \text{mit Zusatzrestrikion } x_r = 0 \end{aligned}$$

betrachtet. In unserer Notation schreibt sich dies zu

$$\begin{aligned} (TP') &: \quad \left(U' \,\middle|\, D', \hat{b}', r' \parallel j_1, j_2, ..., r\right) \\ (TP'') &: \quad \left(U'' \,\middle|\, D'', \hat{b}'', r'' \parallel j_1, j_2, ..., -r\right) \end{aligned}$$

wobei die vorderen Daten jeweils neu bestimmt werden müssen. Dabei geht man geschickterweise wie folgt vor:

- Im Falle (TP') geht man von r aus und sucht (mit Binärsuche) r' links von r, indem man zunächst $\hat{b}$ um a_r reduziert und D um c_r erhöht. Danach erhöht

man $\hat{b}$ systematisch um a_i und erniedrigt D um c_i, für absteigendes noch freies i, beginnend bei $i = r - 1$ und stoppt dies, wenn letztmalig $\hat{b} < 0$ gilt. Findet man so keinen kritischen Index, d.h. ist die Aufgabe unlösbar, so setzen wir $r' = 0$.

- Im Falle (TP'') geht man von r aus und sucht (mit Binärsuche) r' rechts von r. Ausgehend von $\hat{b}$ und D erniedrigt man $\hat{b}$ systematisch im a_i und erhöht D aum c_i, für ansteigendes noch freies i, beginnend bei $i = r + 1$ und stoppt dies, wenn erstmalig $\hat{b} < 0$ gilt. Gibt es keinen kritischen Index, d.h. ist die Aufgabe dadurch lösbar, dass alle freien Variablen den Wert 1 bekommen, so setzen wir $r' = \infty$.

- Die obere Schranke berechnet man als Dantzigschranke (beachte: $U'_D = D' + \hat{b}' \frac{c_{r'}}{a_{r'}}$). Zu beachten ist dabei, dass bei der Verzweigung nach Konstruktion die obere Schranke höchstens absinken kann. Ist also die Dantzigschranke U_D schlechter als die obere Schranke U der Vateraufgabe, so gilt diese weiterhin. Setze also $U' = \min\{U'_D, U\}$.

Beispiel Wir führen diese Methode an unserem Beispiel durch, wobei wir die Knotenauswahl nach der besten oberen Schranke treffen. Zu beachten ist, dass Testaufgaben mit $\hat{b} = 0$ oder $r = 0$ oder $r = \infty$ oder $U = D$ ausgelotet sind, also nicht weiter verzweigt werden müssen. Außerdem ist eine Aufgabe natürlich ausgelotet, wenn $U \leq \bar{L}$ gilt, $\bar{L}$ die laufende untere Schranke, die im Verfahren durch die Zielwerte der Dantziglösungen bestimmt bzw aktualisiert wird. Neben der globalen untere Schranke $\bar{L}$ führen wir auch die *globale obere Schranke* $\bar{U}$, die wegen der Knotenauswahl laufend aktualisiert werden kann. Eine Vorabuntersuchung der gegebenen Aufgabe oder eine Reduzierung ist nicht vorgesehen, könnte aber das Verfahren beschleunigen.

Vorgegeben sei also unsere Aufgabe

Gegenstand j	1	2	3	4	5	6
Nutzen c_j	8	8	6	10	12	12
Gewicht a_j	1	2	2	4	6	10

mit $b = 12$. Wir starten unsere Bearbeitung also bei der Testaufgabe

$$\left(TP^0\right): \quad (38\,|32, 3, 5\,\|\,)$$

wobei die Daten mit der Prozedur LUDD ermittelt werden. Die Aufgabe ist nicht ausgelotet, untere Schranke ist $\bar{L} = 32$, obere Schranke ist $\bar{U} = 38$.

Wir streichen (TP^0) aus der LIST und nehmen stattdessen die Aufgaben (TP^1) : $(36\,|34, 1, 4\,\|5)$, deren Daten sich wie oben beschrieben berechnen und (TP^2) :

$(35\,|32,3,6\,\|{-5})$ auf. Wir merken uns, dass wir nun eine zulässige Lösung mit Zielwert $z = 34$ kennen. Es ist also $\bar{L} = 34$ und $\bar{U} = 36$

$LIST := \{(TP^1): \quad (36\,|34,1,4\,\|5)\,, (TP^2): \quad (35\,|32,3,6\,\|{-5})\}$

Als nächstes entfernen wir (TP^1) aus LIST und verweigen diese Aufgabe in (TP^3) : $(34\,|30,1,2\,\|5,4)$ und $(TP^4): \quad (35\,|34,1,6\,\|5,-4)$. Die erste Aufgabe ist ausgelotet wegen $U \leq \bar{L}$. Es ergibt sich also neben $\bar{U} = 35$ nun

$LIST := \{(TP^2): \quad (35\,|32,3,6\,\|{-5})\,, (TP^4): \quad (35\,|34,1,6\,\|5,-4)\}$

Nun verzweigen wir $(TP^2): \quad (35\,|32,3,6\,\|{-5})$ in $(TP^5): \quad (24\,|20,1,2\,\|{-5},6)$ und $(TP^6): \quad (32\,|32,3,\infty\,\|{-5},-6)$. Beide Aufgaben sind dabei ausgelotet wegen zu schlechten Zielwertes. Es gilt also

$LIST := \{(TP^4): \quad (35\,|34,1,6\,\|5,-4)\}$

Die Aufgabe $(TP^4): \quad (35\,|34,1,6\,\|5,-4)$ verzweigt sich in $(TP^7): \quad (-\infty\,|,,0\,\|5,-4,6)$ und $(TP^8): \quad (34\,|34,1,\infty\,\|5,-4,-6)$. Beide Aufgaben sind ausgelotet, die Liste nun leer und das Verfahren beendet. Optimallösung ist die Dantziglösung von $(TP^4): x^T = (1,1,1,0,1,0)$ mit $z = 34$.

Der **Verzweigungsbaum** stellt sich wie folgt dar:

Zu beachten ist, dass die gebotene Sortierung von $LIST$ natürlich aufwändig ist. Man könnte stattdessen $FIFO$ benutzen, wobei die Beobachtung ist, dass eine so geführte Liste sich nicht stark von einer gemäß der oberen Schranke sortierten Liste unterscheidet. Alternativ kann man die Testaufgaben auch, ähnlich wie beim Kürzeste-Wege-Verfahren von Dial, in Körbe einbringen. (Übungsaufgabe)

1.2.3 Die Methode des Dynamischen Programmierens

Obwohl mit der Methode Dynamisches Programmieren allgemeinere Probleme gelöst werden können, beschreiben wir diese Vorgehensweise direkt für das 0-1-Knapsack

Problem im Standardformat

$$(KP_{01}) \quad \begin{cases} \text{maximiere} & c^T x \\ \text{s.d.} & a^T x \leq b \\ & x \in \{0,1\}^n \end{cases}$$

mit vorgegebenen Vektoren $a = (a_1, ..., a_n)^T,\ c = (c_1, ..., c_n)^T \in \mathbf{N}^n,\ b \in \mathbf{N}$.

Die Variablen der nun vorgegebenen Aufgabe (KP_{01}) sind also $x_1, ..., x_n \in \{0,1\}$. Aufgezählt werden die potentiellen Lösungen von (KP_{01}) in n Stufen, wobei auf der Stufe m die Varianten $\left(KP_m\left[\hat{b}\right]\right)$ von (KP_{01}) betrachtet werden, für die $x_{m+1} = ... = x_n = 0$ gesetzt und b durch $\hat{b}$ ersetzt sind, $0 \leq \hat{b} \leq b$ beliebig vorgegeben.

Den maximalen Zielfunktionswert einer zulässigen Lösung von $\left(KP_m\left[\hat{b}\right]\right)$ erhält man sicher als Maximum des optimalen Wertes der aus $\left(KP_m\left[\hat{b}\right]\right)$ abgeleiteten Aufgabe $\left(KP_m^0\left[\hat{b}\right]\right)$, bei der zusätzlich $x_m = 0$ und der aus $\left(KP_m\left[\hat{b}\right]\right)$ abgeleiteten Aufgabe $\left(KP_m^1\left[\hat{b}\right]\right)$, bei der zusätzlich $x_m = 1$ gefordert ist. Bezeichnen wir den optimalen Zielfunktionswert von $\left(KP_m\left[\hat{b}\right]\right)$ mit $f_m\left[\hat{b}\right]$. Dann gibt $f_{m-1}\left[\hat{b}\right]$ den optimalen Zielfunktionswert von $\left(KP_m^0\left[\hat{b}\right]\right)$ an, während der optimale Zielfunktionswert von $\left(KP_m^1\left[\hat{b}\right]\right)$ durch $f_{m-1}\left[\hat{b} - a_m\right] + c_m$ gegeben ist, falls $a_m \leq \hat{b}$ gilt.

Insgesamt ergibt sich die folgende Formel, die auch BELLMAN**sche Rekursionsformel** heißt:

$$f_m\left[\hat{b}\right] = \max\left\{f_{m-1}\left[\hat{b}\right], f_{m-1}\left[\hat{b} - a_m\right] + c_m\right\}$$

Die Methode läuft nun so ab, dass unter Anwendung der Bellmanschen Rekursionsformel von Stufe 1 bis zu Stufe n jeweils $f_m\left[\hat{b}\right]$ berechnet wird und zwar jedesmal *für alle* $0 \leq \hat{b} \leq b$. Nach dieser **Vorwärtsrechnung** kennt man für jedes $\hat{b}$, insbesondere also für $\hat{b} = b$ den optimalen Funktionswert $f_n\left[\hat{b}\right]$.

Leider kennt man dann noch nicht die optimalen Werte für $x_1..., x_n$! Diese müssen in einer anschließenden **Rückwärtsrechnung** ermittelt werden. Dazu speichert man bereits in der Vorwärtsrechnung sinnvollerweise für jeden Wert $f_m\left[\hat{b}\right]$ die Information, ob dieser Wert bei $x_m = 0$ oder bei $x_m = 1$ angenommen wurde. Dies kann in einer gesonderten Tabelle geschehen oder durch geeignete Markierung von $f_m\left[\hat{b}\right]$. Es bietet sich an, bei diesem an sich nicht negativen Wert ein *negatives Vorzeichen* zu setzen. Wir kennzeichnen damit den *ersten Fall* ($x_m = 0$) in der BELLMANschen

Gleichung. Dabei ist zu berücksichtigen, dass wegen des vorliegenden Standardformats $f_m\left[\hat{b}\right] = 0$ nur auftreten kann, wenn $x_m = 0$ gilt.

Anwendung der Dynamischen Programmierung

Natürlich ist ein Preprocessing wie oben beschrieben, insbesondere eine Reduzierung, auch vor der Bearbeitung durch ein Verfahren der Dynamischen Programmierung sinnvoll, weil dadurch der Datenumfang verringert wird. Versuchen wir daher, die Aufgabe (P) nach Reduzierung wie beschrieben mit dynamischer Programmierung zu lösen. Vorgegeben sei also das 0-1-Knapsack Problem mit den Daten

Gegenstand j	2	3	4	5
Nutzen c_j	8	6	10	12
Gewicht a_j	2	2	4	6

mit $b = 11$ und Vornutzen $\hat{z} = 8$

Wir führen die Vorwärtsrechnung tabellarisch durch.

$\hat{b}\backslash m$	1	2	3	4
0	0	0	0	0
1	0	0	0	0
2	8	-8	-8	-8
3	(8)	-8	-8	-8
4	8	14	-14	-14
5	8	(14)	(-14)	-14
6	8	14	18	-18
7	8	14	18	-18
8	8	14	24	-24
9	8	14	24	-24
10	8	14	24	26
11	8	14	24	(26)

Die eingeklammerten Werte geben dabei die optimalen Werte aus der Rückwärtsrechnung an. Als Optimallösung ergibt sich also $(x_2.x_3, x_4, x_5) = (1, 1, 0, 1)$ mit Zielwert $8 + 26 = 34$.

Zu beachten ist, dass man natürlich die Notierung von Null-Zeilen in der Tabelle unterlassen kann: ein $\hat{b} < \min\{a_j : \text{alle } j\}$ ist immer mit einer Null-Zeile verbunden. Außerdem ist zu beobachten, dass die Spaltenwerte betragsmäßig nicht abfallen können und deswegen die Einträge ab $\hat{b} = \sum_{i=1}^{m} a_i$ konstant gleich $f_m\left[\hat{b}\right] = \sum_{i=1}^{m} c_i$ und damit maximal bleiben. Genau so ist klar, dass in jeder Spalte m die Einträge in den Zeilen $i < a_m$ nichtpositiv sind und betragsmäßig mit denen der Spalte $m-1$ übereinstimmen.

Man kann also die Einträge in die Tabelle "sparsamer" gestalten. Hierin und in weiteren Beobachtungen liegt ein *Einsparungspotential*, mit dem wir uns später noch genauer beschäftigen (*dominierte Zustände*).

Eine Version nach Toth

Die folgende Version der Lösung des 0-1-Knapsack Problems mittels dynamischer Programmierung geht auf TOTH (1980) zurück. Motivation dieser Variante ist eine geeignete Algorithmisierung der Vorgehensweise zusammen mit dem Wunsch, den Speicherplatzbedarf einzuschränken. Letzteres kann erreicht werden, wenn die Spalten der obigen Tabelle immer wieder überschrieben werden. Dies geht allerdings nur, wenn die Rückwärtsrechnung im Prinzip bereits während der Vorwärtsrechnung erledigt. wird. Es muss dann für jeden Eintrag der $m-$ten Spalte der Tabelle bereits festliegen, welcher Vektor $X_{\hat{b}} := (x_1, ..., x_m)$ den dort notierten optimalen Zielfunktionswert realisiert. Dies hebt natürlich den Speicherplatzvorteil des Überschreibens der Spalten wieder auf, es sei denn, man verwendet den folgenden Trick von TOTH: Es gilt ja $x_i \in \{0, 1\}$ für alle $i = 1, ..., m$. Daher bietet es sich an, die Vektoren $X_{\hat{b}}$ *binär* zu notieren.

Zusammen mit der bereits oben festgehaltenen Beobachtung, dass sich in der $m-$ten Spalte gegenüber der $(m-1)-$ten Spalte höchstens die Einträge in den Zeilen a_m bis $\sum_{i=1}^{m} a_i$ bzw. b ändern, führen diese Feststellungen zu folgendem Verfahren, b ei dem wir bei *vorgegebenem* m folgende Notationen verwenden:

$$\begin{aligned} \nu &: = \min\left(\sum_{j=1}^{m-1} a_j, b\right) \\ d &: = 2^{m-1} \\ X_{\hat{b}} &: = (x_1, ..., x_m) \text{ in kodierter Form} \\ P_{\hat{b}} &: = f_m\left[\hat{b}\right] \end{aligned}$$

Dann beschreibt sich die Prozedur wie folgt:

Prozedur DP1 (Dynamische Programmierung 1)

Input: (KP_{01})

Output: Optimallösung x_{opt} mit Zielfunktionswert z_{opt}.

1. **Start:** Gegeben sei ein 0-1-Knapsack Problem in der obigen Notierung.

2. **Initialisierung:** Setze $\nu = a_1,\ d = 2,\ P_\nu = c_1,\ X_\nu = 1$ und
   für $\hat{b} = 0, ..., a_1 - 1$ setze $P_{\hat{b}} = 0,\ X_{\hat{b}} = 0$.

3. **Iteration: Für** $m = 2, ..., n$ führe aus:

   begin

   Ist $\nu < b$ % `hier werden noch fehlende Einträge in Spalte` $m-1$ `ergänzt. (*)`

   begin

   $u := \nu$

   $\nu := \min(\nu + a_m, b)$

   for $\hat{b} := u + 1$ to ν do

   begin

   $P_{\hat{b}} := P_u$

   $X_{\hat{b}} := X_u$

   end

   end

   for $\hat{b} := \nu$ to a_m step -1 do % `Einträge in Spalte` m

   if $P_{\hat{b}} < P_{\hat{b}-a_m} + c_m$ then

   begin

   $P_{\hat{b}} := P_{\hat{b}-a_m} + c_m$

   $X_{\hat{b}} := X_{\hat{b}-a_m} + d$

   end

   $d := 2d$

   end

4. **Ergebnis:** $z_{opt} = P_b$,dekodiere X_b und bestimme so x_{opt}

In der praktischen Durchführung führt man beide Werte, $P_{\hat{b}}$ und $X_{\hat{b}}$ gleichzeitig als $P_{\hat{b}}, X_{\hat{b}}$. Angewendet auf unsere Aufgabenstellung

Gegenstand j	2	3	4	5
Nutzen c_j	8	6	10	12
Gewicht a_j	2	2	4	6

mit $b = 11$ und Vorkosten $\hat{z} = 8$

ergibt sich folgender Ablauf des Verfahrens (die an sich überschriebenen Spalten werden hier nebeneinander geschrieben, es werden nur die neuen Daten notiert. Einträge vor der Überschreibung stehen in der Spalte links daneben)

b̂\m	1	2	3	4
0	0,0			
1	0,0			
2	8,1			
3	8,1			
4	8,1	14,3		
5		14,3		
6		14,3	18,5	
7		14,3	18,5	
8		14,3	24,7	
9			24,7	
10			24,7	26,11
11			24,7	26,11

Zur Dekodierung von $X_{11:}$ Es gilt $11 = 1 \cdot 2^0 + 1 \cdot 2^1 + 0 \cdot 2^2 + 1 \cdot 2^3$, also ist $x^T = (1, 1, 0, 1)$

Damit ergibt sich also das bekannte Ergebnis.

Zu beachten ist, dass der (leicht einzusehende) Aufwand des Verfahrens, $O(nb)$, praktisch sehr groß werden kann. Z. B. kann selbst bei wenigen Variablen ein sehr großer Aufwand getrieben werden müssen, wenn die Kapazität b des Rucksacks entsprechend groß ist.

Zu beachten ist desweiteren eine Problematik der kodierten Darstellung der optimalen Lösung in binärer Form: Programmiersprachen stellen normalerweise Integerdarstellungen nur bis zu einer beschränkten Zahlengröße zur Verfügung, z.B. bis 2^{32}. Auch reelle Zahlen sind in ihrer Darstellung begrenzt, etwa bis 10^{15}, wodurch noch Integer bis 2^{49} sicher dargestellt werden können. Dies reicht für die Bearbeitung des niederdimensionalen Bereichs (nach erfolgter Reduzierung!) von Knapsack-Aufgaben. Will man in einen mittleren Dimensionsbereich vordringen, etwa $n = 1000$, so müssen mehrere kodierte Speichervektoren verwendet werden. Benutzt man eine Integerdarstellung mit maximaler Größe von 2^{32}, so kann man durch Verwendung von 32 solcher Speichervektoren eine Integerzahl bis $(2^{32})^{32} = 2^{1024}$ darstellen.

1.3 Erfahrungen

Die in diesem Kapitel erarbeiteten Verfahren wurden in VBA Basic unter Excel programmiert und einem Test an zufällig erzeugten Rucksackproblemen verschiedener Struktur unterzogen. In diesem Abschnitt soll über die Erfahrungen mit den programmierten Verfahren berichtet werden.

1.3.1 Tests zu den Verfahren

Vor dem Austesten der Verfahren stand die Notwendigkeit, genügend viele allgemeine, aber unterschiedlich strukturierte Rucksack Probleme zu erzeugen.

Beispielerzeugung

Mit einem ebenfalls in VBA programmierten Beispielerzeugungsprogramm können Beispiele der folgenden Struktur erzeugt werden: Die Gewichte werden in vorgewählter Anzahl n in einem Bereich $[1, R]$ der natürlichen Zahlen zufällig gleichverteilt erzeugt. Die Kapazität b wird einheitlich auf die Hälfte der Summe aller Gewichte festgelegt.Zu den Gewichten a_i wird der Nutzen c_i zufällig gleichverteilt im Bereich $[a_i + \rho - r, a_i + \rho + r]$ mit vorgewählten Parametern $r, \rho \in \mathbf{N}_0$ erzeugt, wobei sichergestellt wurde, dass auch der Nutzen c_i im Bereich $[1, R]$ liegt. Ist dabei r sehr groß, etwa $r = R$, so wird der Nutzen c_i quasi unabhängig von a_i im Bereich $[1, R]$ ermittelt. Ist dagegen r eher klein, etwa $r = 10$ oder $r = 1$, so ist der Nutzen relativ fest an das Gewicht gebunden, der Nutzen schwankt in einem relativ schmalen Band um die Gewichte. Bei $\rho = 0$ und $r = 0$ ergeben sich *Subset Sum Probleme.*

Skizze:

Erfahrungsgemäß sind Rucksack Probleme mit kleinerem r viel schwieriger zu lösen

als mit großem r.

Angaben zu den Aufgaben und Tests

Aus Aufwandgründen wurde zu jedem der folgenden Strukturtypen jeweils nur eine Aufgabe erzeugt und getestet. Generell wurde zu jeder der Dimensionen $n = 50,\ n = 100,\ n = 500,\ n = 1000$ und $n = 5000$ je eine Aufgabe zu $r = 100,\ r = 10$, und $r = 1$ erzeugt, wobei in allen Fällen willkürlich $R = 100$ und $\rho = 10$ gewählt wurde. Die Aufgaben wurden generell als *nicht* nach den Nutzendichten sortiert erzeugt, mussten also in der Bearbeitung zunächst entsprechend sortiert werden. Es entspricht der realen Situation, dass Aufgaben unsortiert sind, so dass die Sortierung zur Bearbeitung dazugehört, wenn das anzuwendende Verfahren eine Sortierung voraussetzt.

In den Tests wurden also die Aufgaben zunächst sortiert mit dem ganz normalen Verfahren der sukzessiven Vertauschung. Die so sortierten Aufgaben wurden dann zuerst ohne Preprocessing den Verfahren HSV, BSF und DP1 unterworfen, wobei BSF die besprochene Breitensuche mit einer *FIFO* Verwaltung der Liste LIST bezeichnet.

Zusätzlich wurde das Verfahren DP1 als uDP1 ohne vorherige Sortierung getestet, da die Dynamische Programmierung die Sortierung nicht notwendig voraussetzt.

Anschließend wurden die gleichen Aufgaben noch einmal diesen Verfahren unterworfen, diesmal mit einem Preprocessing nach dem Sortieren: zunächst wurde die Prozedur LUDD angewandt, anschließend die Prozedur DDR.

Testergebnisse

Die folgenden Tabellen geben die Testergebnisse wider. Angegeben werden in jedem Fall die Anzahl der Iterationen it des Hauptprogamms und die insgesamt benötigte Zeit sec in Sekunden. Dabei war für die Hauptprogramme je eine Iterationen-Obergenze angegeben, bei der das Programm ergebnislos abgebrochen wurde

25 000 000 beim Verfahren HSV

3200 beim Verfahren BSF

Diese beiden Begrenzungen entstanden durch den Wunsch, die Laufzeit des Hauptverfahrens auf etwa eine Minute zu begrenzen. Wurde das Hauptprogramm ergebnislos abgebrochen, wird das bei der Iterationenanzahl mit "-" gekennzeichnet. In den Fällen, wo ein Preprocessing dem eigentlichen Verfahren vorangestellt wurde, ist dies mit einem vorangestellten "p" vor dem Verfahrensnamen gekennzeichnet. Zu beachten ist, dass das Preprocessing die Sortierung der Aufgabe voraussetzt.

Zunächst wurde mit $r = 100$ gerechnet.

n	50	50	100	100	500	500	1000	1000	5000	5000
	it	sec	it	sec	it	sec	it	sec	it	sec
HSV	128	0,00	384	0,00	1336	0,05	1116	0,16	1884	3,06
BSF	105	0,02	96	0,02	281	0,08	117	0,12	9	2,98
DP1	50	0,00	100	0,01	500	0,48	1000	1,92	5000	48,89
uDP1	50	0,00	100	0,05	500	1,12	1000	4,45	5000	111,31
pHSV	37	0,00	40	0,00	124	0,00	47	0,11	0	2,97
pBSF	40	0,00	54	0,02	230	0,03	31	0,11	0	2,98
pDP1	12	0,00	20	0,00	20	0,00	12	0,11	0	2,97

Die Iterationsanzahl 0 besagt, dass das Optimalergebnis bereits nach der Anwendung von DDR vorlag und daher kein Hauptverfahren folgen musste.

Danach wurde mit $r = 10$ gerechnet.

n	50	50	100	100	500	500	1000	1000	5000	5000
	it	sec	it	sec	it	sec	it	sec	it	sec
HSV	96	0,00	162	0,00	519	0,03	345	0,12	1737	3,14
BSF	46	0,00	75	0,00	143	0,05	67	0,14	-	87,36
DP1	50	0,00	100	0,01	500	0,48	1000	2,01	5000	48,36
uDP1	50	0,02	100	0,05	500	1,05	1000	4,36	5000	109,66
pHSV	50	0,05	10	0,00	0	0,03	0	0,11	0	3,02
pBSF	21	0,00	8	0,01	0	0,02	0	0,11	0	3,05
pDP1	11	0,00	5	0,00	0	0,03	0	0,11	0	3,02

Schließlich wurde mit $r = 1$ gerechnet

n	50	50	100	100	500	500	1000	1000	5000	5000
	it	sec	it	sec	it	sec	it	sec	it	sec
HSV	12T	0,03	28T	0,06	5T	0,05	23T	0,17	-	64,43
BSF	-	12,29	-	23,06	-	32,87	1168	1,61	-	86,16
DP1	50	0,00	100	0,01	500	0,47	1000	1,66	5000	44,12
uDP1	50	0,02	100	0,05	500	1,22	100	4,47	5000	116,28
pHSV	9466	0,03	24T	0,08	556	0,003	577	0,12	-	58,80
pBSF	-	12,44	-	23,44	-	23,50	1168	0,09	-	47,97
pDP1	35	0,00	57	0,02	68	0,05	33	0,11	1772	11,80

Zusätzlich wollen wir notieren, dass alleine das Sortieren die folgenden Zeiten in Anspruch nahm

n	50	100	500	1000	5000	10000
sec	0,0	0,0	0,0	0,1	3,0	12,1

während die Prozeduren LUDD und DDR in allen Dimensionen nur 0,0 sec benötigte. Dabei wurde zusätzlich eine Aufgabe mit $n = 10\,000$ generiert, um den Trend besser darzustellen.

Auswertung

Im Folgenden wollen wir einige Thesen aufstellen, die sich aus den Zahlen anbieten. Diese sind natürlich mit dem Makel behaftet, dass jeweils nur eine zufällige Aufgabe getestet wurde.

1. In der Tat scheinen Aufgaben mit kleinem r schwieriger zu lösen zu sein als solche mit großem r, wobei sich bei $r = 100$ und $r = 10$ kaum ein Unterschied ergab.
2. Die Anwendung des Preprocessing scheint sich zu lohnen bei DP1.
3. Das Verfahren HSV scheint generell eine große Anzahl von (billigen) Iterationen zu benötigen, die durch Preprocessing (erheblich) abgesenkt werden kann. Es erscheint weniger geeignet bei sehr kleinem r. Das Verfahren BSF scheint ungeeignet bei sehr kleinem r, wohingegen es sich bei größerem r sehr achtbar schlägt und mit HSV vergleichbar ist. Das Verfahren DP1, insbesondere mit Preprocessing, arbeitet bei großem r gut, bei kleinem r sehr gut. Es scheint relativ unabhängig von der Wahl von r zu sein.
4. Das Verfahren DP1 scheint, obwohl es keine Sortierung voraussetzt, sehr positiv auf die Sortierung zu reagieren. Es benötigt ohne Sortierung mehr als doppelt so viel Zeit wie mit Sortierung.
5. Die Sortierung benötigt mit steigendem n überproportional an Zeit.

1.3.2 Übungsaufgaben

Aufgabe 1

Lösen Sie die folgenden 0-1-Knapsack Probleme analog der oben beschriebenen Vorgehensweise, d.h.

1. Wenden Sie zur Verfahrensvorbereitung die Prozedur Preprocessing an.
2. Wenden Sie dann die B&B Methode an, explizit durch Führen von *LIST* mit zwischenzeitlicher Reduktion wie im obigen Beispiel, in der Version von Horowitz und Sahni oder mit dem Verfahren mit Breitensuche.

3. Lösen Sie dann die Aufgaben auch mit Dynamischer Programmierung, mit oder ohne Preprocessing.

a)

$$\begin{array}{ll} \text{maximiere} & 70x_1 + 20x_2 + 39x_3 + 37x_4 + 7x_5 + 5x_6 + 10x_7 \\ \text{s.d.} & 31x_1 + 10x_2 + 20x_3 + 19x_4 + 4x_5 + 3x_6 + 6x_7 \leq 50 \\ & x_1, ..., x_7 \in \{0, 1\} \end{array}$$

b)

$$\begin{array}{ll} \text{maximiere} & 5x_1 + 9x_2 + 12x_3 + 15x_4 + 18x_5 + 19x_6 \\ \text{s.d.} & 5x_1 + 11x_2 + 8x_3 + 8x_4 + 11x_5 + 14x_6 \leq 26 \\ & x_1, ..., x_6 \in \{0, 1\} \end{array}$$

c)

$$\begin{array}{ll} \max & z = 12x_1 + 5x_2 + 8x_3 + 7x_4 + 5x_5 + 5x_6 + 4x_7 + 3x_8 + 2x_9 + x_{10} \\ \text{s.d.} & 35x_1 + 27x_2 + 23x_3 + 19x_4 + 15x_5 + 15x_6 + 12x_7 + x_8 + 6x_9 + 3x_{10} \leq 39 \\ & x_1, ..., x_{10} \in \{0, 1\} \end{array}$$

Aufgabe 2

Man formuliere eine Prozedur der Komplexität $O(n)$, die den kritischen Index, eine obere und eine untere Schranke berechnet, indem die Indizes von links nach rechts einmal durchgegangen werden.

Aufgabe 3

Programmieren Sie unterschiedliche Versionen der Branch and Bound Methode mit Breitensuche (z.B. mit Körben) und vergleichen Sie sie numerisch an verschiedenen zufällig erstellten Problemen unterschiedlicher Größe und Struktur.

Aufgabe 4

Basteln Sie aus dem B&B Verfahren mit Breitensuche durch vorzeitigen Abbruch eine LU Prozedur. Testen Sie diese gegen LUDD.

Kapitel 2

Verbesserte Techniken

Die reale Laufzeit eines B&B Verfahrens zur Lösung von (KP_{01}) hängt wesentlich von der gewählten Verzweigungsstrategie und der zum Ausloten gewählten lokalen oberen Schranke, andererseits auch vom gutem Preprocessing ab: von scharfen globalen oberen und unteren Schranken und von einer effektiven Vorab-Reduzierung des Umfangs der Aufgabenstellung.

Wir wollen uns daher Gedanken machen, wie die bisher erarbeiteten Preprocessing-Prozeduren verbessert werden können, wobei wir eine Verbesserung sowohl in der Schärfe (bessere Schranken) als auch in der Geschwindigkeit (kürzere Laufzeit) im Sinn haben. Ferner wollen wir effektiv zu implementierende konkrete B&B Verfahren und Verfahren der Dynamischen Optimierung erarbeiten.

2.1 Verbesserte Schrankenberechnung

Unsere bisherigen Überlegungen laufen auf eine **Vierteilung des Lösungsprozesses** der gegebenen Aufgabe (KP_{01}) hinaus, bei dem das *Preprocessing* die Schritte 1 bis 3 umfasst:

1. **Schritt 1: sortieren.** Es wird eine beliebige Sortierungsprozedur gewählt, um die Variablen von (KP_{01}) im Sinne absteigender Nutzendichten zu sortieren. Eine schnelle Sortierungsprozedur ist *Quicksort*. In diesem Kapitel gehen wir *immer noch* davon aus, dass (KP_{01}) *bereits sortiert* ist.

2. **Schritt 2: abschätzen.** Es werden Prozeduren angewendet, die je eine möglichste scharfe untere Schranke L und obere Schranke U für den optimalen Zielwert von (KP_{01}) berechnen. Dabei wird gleichzeitig der kritische Index k ermittelt. Eine Prozedur, die dies leistet, ist beispielsweise *LUDD*.

3. **Schritt 3: reduzieren.** Es wird eine Prozedur angewendet, die vorab bereits möglichst viele der Variablen auf den Wert 1 oder 0 fixiert. Gleichzeitig sollen hier die bestehenden Schranken noch einmal verbessert werden. Eine Prozedur, die dies leistet, ist die Prozedur DDR.

4. **Schritt 4: lösen.** Schlussendlich wird eine Lösungsmethode auf die reduzierte Aufgabe angewendet, z.B. die *Prozedur HSV* oder die *Prozedur DP1* oder eine differenziertere Methode. Damit ist dann auch (KP_{01}) gelöst.

Zu beachten ist, dass jederzeit nach Verschärfung der oberen oder der unteren Schranke überprüft werden kann, ob $U = L$ gilt. In diesem Falle kann der gesamt Vorgang abgebrochen werden, weil die der unteren Schranke zugrundeliegende zulässige Lösung optimal ist.

Gleichzeitig stellt die Vorgehensweise bei Abbruch nach Schritt 2 oder Schritt 3 eine Heuristik zur Lösung von (KP_{01}) zur Verfügung, die eine gute zulässige Lösung einschließlich einer Abschätzung für den optimalen Zielwert liefert.

Wir wollen uns im Folgenden mit der Verbesserung der Prozeduren aller vier Schritte beschäftigen. Gehen wir also zunächst erneut auf die Berechnung von oberen und unteren Schranken für das 0-1-Knapsack Problem ein. Danach wollen wir die Reduzierung der gegebenen Aufgabe verbessern. Später soll die eigentliche Lösungsprozedur verbessert werden. Im nächsten Kapitel gehen wir dann auch auf Verbesserungsmöglichkeiten beim Sortieren ein.

Weiterhin setzen wir voraus, dass die Indizes nach abfallenden Nutzendichten sortiert sind, die Aufgaben aber sonst nicht notwendig im Standardformat.

2.1.1 Hilfreiche Abschätzungen

Zunächst einmal wollen wir Abschätzungen herleiten, die uns bei der Verbesserung der Schranken hilfreich sein werden.

Satz 2.1 *Gegeben sei ein 0-1Rucksackproblem* (KP_{01}) *mit kritischem Index* $k \in \{1, ..., n\}$ *und ein Index* $i \in \{1, ..., n\}$.

1. *Gilt für den kritischen Index* k *der Aufgabe* $k > i$, *so ist*
   $\sum_{j=1}^{i} c_j + \frac{c_i}{a_i}\left(b - \sum_{j=1}^{i} a_j\right) \geq \sum_{j=1}^{i+1} c_j + \frac{c_{i+1}}{a_{i+1}}\left(b - \sum_{j=1}^{i+1} a_j\right)$
2. *Gilt für den kritischen Index* k *der Aufgabe* $k < i$, *so ist*
   $\sum_{j=1}^{i} c_j + \frac{c_i}{a_i}\left(b - \sum_{j=1}^{i} a_j\right) \geq \sum_{j=1}^{i-1} c_j + \frac{c_{i-1}}{a_{i-1}}\left(b - \sum_{j=1}^{i-1} a_j\right)$

3. *Insbesondere gibt für alle* $i \in \{1, ..., n\}$ *der Wert* $U = \left\lfloor \sum_{j=1}^{i} c_j + \frac{c_i}{a_i} \left(b - \sum_{j=1}^{i} a_j \right) \right\rfloor$ *eine obere Schranke für* (KP_{01}) *an, die potentiell schwächer ist als die Dantzigschranke.*

Beweis: Es gelten folgende Abschätzungen:

1. Wegen $k > i$ gilt $b - \sum_{j=1}^{i} a_j \geq 0$ und wegen $\frac{c_i}{a_i} \geq \frac{c_{i+1}}{a_{i+1}}$ hat man

$$\begin{aligned}
& \sum_{j=1}^{i} c_j + \frac{c_i}{a_i} \left(b - \sum_{j=1}^{i} a_j \right) - \sum_{j=1}^{i+1} c_j - \frac{c_{i+1}}{a_{i+1}} \left(b - \sum_{j=1}^{i+1} a_j \right) \\
= & \sum_{j=1}^{i-1} c_j + \frac{c_i}{a_i} \left(b - \sum_{j=1}^{i-1} a_j \right) - \sum_{j=1}^{i} c_j - \frac{c_{i+1}}{a_{i+1}} \left(b - \sum_{j=1}^{i} a_j \right) \\
\geq & -c_i + \frac{c_i}{a_i} \left(b - \sum_{j=1}^{i-1} a_j \right) - \frac{c_i}{a_i} \left(b - \sum_{j=1}^{i} a_j \right) \\
= & -c_i + \frac{c_i}{a_i} a_i = 0
\end{aligned}$$

2. Wegen $k < i$ gilt $b - \sum_{j=1}^{i-1} a_j < 0$ und wegen $\frac{c_{i-1}}{a_{i-1}} \geq \frac{c_i}{a_i}$ hat man

$$\begin{aligned}
& \sum_{j=1}^{i} c_j + \frac{c_i}{a_i} \left(b - \sum_{j=1}^{i} a_j \right) - \sum_{j=1}^{i-1} c_j - \frac{c_{i-1}}{a_{i-1}} \left(b - \sum_{j=1}^{i-1} a_j \right) \\
\geq & c_i + \frac{c_i}{a_i} \left(b - \sum_{j=1}^{i} a_j \right) - \frac{c_i}{a_i} \left(b - \sum_{j=1}^{i-1} a_j \right) \\
= & c_i + \frac{c_i}{a_i} (-a_i) = 0
\end{aligned}$$

3. Es ist leicht zu sehen, dass für $i = k$ der Wert U die Dantzigsschranke angibt. Wendet man für $i \neq k$ die Abschätzung aus 1. oder 2. iteriert an, so ergibt sich die Behauptung (Start bei $i = k - 1$ bzw. $i = k + 1$)

□

Bemerkung: Die Behauptung kann auch im Sinne der oben angeführten Interpretation der freien Verschiebbarkeit von Gewichtsanteilen in einer LP-Relaxation interpretiert und begründet werden.

Aus dem Satz ziehen wir für die Aufgaben (KP_k^0) und (KP_k^1), bei denen in (KP_{01}) die Variable k auf 0 oder 1 fixiert ist, das

Korollar 2.2 *Gegeben sei ein 0-1Rucksackproblem* (KP_{01}) *mit kritischem Index* $k \in \{1, ..., n\}$. *Dann ergibt sich*

1. *für* (KP_k^0) *und* $k < n$, *wenn für dessen kritischen Index* $k_0 \leq n$ *gilt, eine obere Schranke* $U^0 = \left\lfloor \sum_{j=1}^{k-1} c_j + \frac{c_{k+1}}{a_{k+1}} \left(b - \sum_{j=1}^{k-1} a_j \right) \right\rfloor$, *die potentiell schwächer ist als die Dantzigschranke* U_D^0 *dieser Aufgabe.*

2. *für* (KP_k^1) *und* $k > 1$, *wenn für dessen kritischen Index* $k_1 \geq 1$ *gilt, eine obere Schranke* $U^1 = \left\lfloor \sum_{j=1}^{k} c_j + \frac{c_{k-1}}{a_{k-1}} \left(b - \sum_{j=1}^{k} a_j \right) \right\rfloor$, *die potentiell schwächer ist als die Dantzigschranke* U_D^1 *dieser Aufgabe.*

Beweis: Wir betrachten nacheinander die beiden Aufgaben

1. Wendet man den Satz auf die lösbare Aufgabe (KP_k^0) an mit $i = k + 1$, so ergibt sich wegen $n \geq k_0 > k$

$$\begin{aligned} U_D^0 &\leq \left\lfloor \sum_{j=1, j\neq k}^{k+1} c_j + \frac{c_{k+1}}{a_{k+1}} \left(b - \sum_{j=1, j\neq k}^{k+1} a_j \right) \right\rfloor \\ &= \left\lfloor \sum_{j=1}^{k-1} c_j + \frac{c_{k+1}}{a_{k+1}} \left(b - \sum_{j=1}^{k-1} a_j \right) \right\rfloor = U^0 \end{aligned}$$

2. Wendet man den Satz auf lösbares (KP_k^1) an mit $i = k - 1$, so ergibt sich wegen $1 \leq k_1 < k$

$$\begin{aligned} U_D^1 &\leq \left\lfloor c_k + \sum_{j=1}^{k-1} c_j + \frac{c_{k-1}}{a_{k-1}} \left([b - a_k] - \sum_{j=1}^{k-1} a_j \right) \right\rfloor \\ &= \left\lfloor \sum_{j=1}^{k} c_j + \frac{c_{k-1}}{a_{k-1}} \left(b - \sum_{j=1}^{k} a_j \right) \right\rfloor = U^1 \end{aligned}$$

□

Zu bemerken ist, dass die Formel für U^0 auch ohne die einschränkenden Bedingungen an k, k_0 gilt, wenn hilfsweise $c_{n+1} = 1$ und $a_{n+1} >> 1$ angenommen wird, so dass $\left\lfloor \frac{c_{k+1}}{a_{k+1}} \left(b - \sum_{j=1}^{k-1} a_j \right) \right\rfloor = 0$ gilt. Man kann auch vereinbaren, dass formal $\frac{c_{k+1}}{a_{k+1}} = 0$ eingesetzt wird. Ist $k = n$, so ist (KP_k^0) trivial lösbar und $k_0 = n + 1$

Genauso gilt die Formel für U^1 auch ohne Einschränkungen, wenn hier $c_0 >> 1$ und $a_0 = 1$ oder formal $\frac{c_0}{a_0} = \infty$ eingesetzt wird. Im Falle $k = 1$ ist nämlich $b < a_1$, also $b - a_1 < 0$. Dies liefert $U^1 = -\infty$, was genau der Tatsache entspricht, dass nach der Fixierung $x_1 = 1$ die gestellte Aufgabe nicht lösbar ist. Wir bezeichnen die so **erweitere Aufgabenstellung** mit $\left(KP_{01}^{+}\right)$, wobei *im Bedarfsfall auch noch rechts von* $n + 1$ entsprechend erweitert werden kann.

2.1.2 Erweiterte Dantziglösungen

Eine sich anbietende Möglichkeit, eine bessere zulässige Lösung als die Dantziglösung zu erhalten, ist, letztere zu erweitern, indem man der Reihe nach weitere Gegenstände in den Rucksack hineinpackt, sofern welche existieren, die noch hineinpassen. Wir nennen solche Lösungen **erweiterte Dantziglösungen.**

Gierige Lösung

Versucht man, die Dantziglösung noch *systematisch* zu erweitern, entsteht eine zulässige Lösung, die unter Durchgehen der noch nicht im Rucksack untergebrachten Gegenstände die jeweilige Restkapazität noch möglichst gut auszunutzt.

Prozedur LUDG (Lower Upper Dantzig Gierig)

Input: $\left(KP_{01}^{+}\right)$

Output: Kritischer Index k, obere Schranke U sowie eine zulässige Lösung x von (KP_{01}) mit Zielfunktionswert Z.

1. **Start:** Setze $Z = 0$ und $k = n + 1$.
2. **Kritischer Index**
   Für $j = 1, ..., n$ führe aus:
   Falls $a_j > b$, setze $x_j = 0$, $k = j$ und gehe zu 3.
   andernfalls setze $x_j = 1$, $Z = Z + c_j$ und $b = b - a_j$.
3. **Schranken**
   Setze $U := Z + \lfloor \zeta_k b \rfloor$
   Für $j = k + 1, ..., n$ führe aus:
   Falls $a_j > b$, setze $x_j = 0$,
   andernfalls setze $x_j = 1$, $Z = Z + c_j$ und $b = b - a_j$.

Die von der Prozedur bestimmte zulässige Lösung x' wollen wir **gierig erweiterte Dantziglösung** oder einfach **Gierige Lösung** von (KP_{01}) nennen. Sie zeichnet sich dadurch aus, dass $x_j = 1$ gilt für $j = 1, ..., k-1$ und $x_k = 0$, während die übrigen Komponenten 0 oder 1 sind. Die Prozedur LUDG konkurriert mit der Prozedur LUDD. Sie hat ebenfalls Aufwand $\mathcal{O}(n)$.

Wir bemerken, dass in unserem *Standardbeispiel* die Gierige Lösung mit der Dantziglösung übereinstimmt. Es ergibt sich also $Z = 32$ und $U = 38$.

Die Schwerpunkterweiterung

Der Nachteil der gierigen Lösung ist, dass sie eine Erweiterung strikt nach der Ordnung der Indizes sucht, also nur die Nutzendichte als Kriterium verwendet, *nicht aber das absolute Gewicht*. So kann es vorkommen, dass ein von der Nutzendichte *und dem* Gewicht interessanter Gegenstand nicht aufgenommen werden kann, weil unglücklicherweise "gerade zuvor" die Restkapazität durch Aufnahme eines Gegenstandes zu klein geworden ist.

Die folgende Alternative zur Prozedur LUDG ermittelt eine erweiterte Dantziglösung, die die Restkapazität unter Verwendung nur eines weiteren Gegenstandes mit maximalem Nutzen ausnutzt.

Prozedur LUDS (Lower Upper Dantzig Schwerpunkt)

Input: $\left(KP_{01}^{+}\right)$

Output: Kritischer Index k, obere Schranke U sowie eine zulässige Lösung x' von (KP_{01}) mit Zielfunktionswert Z.

1. **Start:** Setze $Z = 0$ und $k = n + 1$.
2. **Kritischer Index**

   Für $j = 1, ..., n$ führe aus:

   Falls $a_j > b$, setze $x'_j = 0$, $k = j$ und gehe zu Schritt 2,

   andernfalls setze $x'_j = 1$, $Z = Z + c_j$ und $b = b - a_j$.
3. **Schranken**

   Setze $U := Z + \lfloor \zeta_k b \rfloor$ und $\bar{c}_{k-1} = Z$

   setze $\hat{c}^0 = 0$ und $j_0 = k$.

   Für $j = k + 1, ..., n$ führe aus:

   Ist $a_j \leq b$ und $c_j > \hat{c}^0$, so setze $Z = \bar{c}_{k-1} + c_j$, $\hat{c}^0 = c_j$, $j_0 = j, x'_j = 0$,

Ist $j_0 \neq k$, setze $x'_{j_0} = 1$.

Die Prozedur ermittelt also neben der Dantzigschranke die erweiterte Dantziglösung mit dem Zielwert

$$Z := \bar{c}_{k-1} + \max\left\{c_j : \sum_{i=1}^{k-1} a_i + a_j \leq b,\ j > k\right\}$$

Sie hat ebenfalls die Komplexität $\mathcal{O}(n)$.

Eine Verschärfung der Dantzig-Schranke

Die folgenden Überlegungen zeigen, dass man bei Aufgaben, bei denen *keine echte erweiterte Dantziglösung* existiert, die obere Schranke im Nachhinein potentiell verschärfen kann.

Satz 2.3 *Gegeben sei ein 0-1Rucksackproblem* (KP_{01}) *mit kritischem Index* $1 < k \leq n$. *Gibt es keine echte erweitere Dantziglösung, sind also alle Gewichte* a_j *mit* $j \geq k$ *zu groß für die Restkapazität der Dantziglösung, so gibt*

$$U = \max\left\{\bar{c}, \bar{c} + \left\lfloor a_m \frac{c_k}{a_k} + \left(\bar{b} - a_m\right) \frac{c_{k-1}}{a_{k-1}} \right\rfloor\right\}$$

eine obere Schranke für (KP_{01}) *an, wobei wir wieder* $\bar{c} = \sum_{j=1}^{k-1} c_j$ *und* $\bar{b} = b - \sum_{j=1}^{k-1} a_j$ *sowie* $m = \arg\min\{a_j : j \geq k\} \in \{k, ..., n\}$ *gesetzt haben.*

Beweis: Wir argumentieren diesmal der Einfachheit halber mit dem Verschieben von Gewichtsanteilen.

Es sei x eine optimale zulässige Lösung von (KP_{01}). Liegen keine Gewichtsanteile rechts vom Index $(k-1)$, so muss x die Dantziglösung sein.

Gehen wir also davon aus, dass auch Gewichtsanteile rechts vom Index $(k-1)$ liegen. Gemäß der Definition von a_m müssen dann mindestens a_m Gewichtsanteile rechts von $(k-1)$ liegen. Nach Annahme muss

$$a_m > \bar{b} = b - \sum_{j=1}^{k-1} a_j \Longrightarrow b - a_m < \sum_{j=1}^{k-1} a_j$$

gelten. Da wir den Nutzen der Lösung x nach oben abschätzen wollen, können wir, in der LP-Relaxation argumentierend und wegen $k \leq n$, das Gesamtgewicht der Lösung auf b anfüllen, wobei der Nutzen der Lösung höchsten ansteigt. Es sei $w \geq a_m$ der

Gewichtsanteil der Lösung x, der rechts neben $(k-1)$ liegt. Es liegen also links genau $(b-w)$ Gewichtsanteile. Wäre $(w-a_m) > 0$, so könnte man diese Anteile so weit wie möglich nach links verschieben. Links von $(k-1)$ lägen danach höchstens

$$(b-w)+(w-a_m) = b-a_m < \sum_{j=1}^{k-1} a_j$$

Anteile. Da auf der rechten Seite der Ungleichung das Gewicht der Dantziglösung steht, passen also die $(b-w)+(w-a_m)$ Gewichtsanteile in den Indexbereich links von k hinein. Wegen des "Echt-kleiner-Zeichens" in der Ungleichung liegen die Gewichtsanteile nun so, dass die Gewichte der ersten $k'-1 < k-1$ Indizes voll ausgeschöpft sind und das des Gewichts $k' \leq k-1$ bruchteilig.

Der Anteil des Gewichts, der zum Index k' gehört, ist also $\left(b - a_m - \sum_{j=1}^{k'-1} a_j\right)$.

Nun verschieben wir auch noch die rechten a_m Gewichtsanteile so weit wie möglich im Bereich rechts von $(k-1)$ nach links. Sie liegen dann beim Index k.

Damit ergibt sich für den Nutzen dieser Lösung

$$N := \sum_{j=1}^{k'-1} c_j + \left(b - a_m - \sum_{j=1}^{k'-1} a_j\right) \frac{c_{k'}}{a_{k'}} + a_m \frac{c_k}{a_k}$$

Die Behauptung ist also richtig, wenn wir

$$\sum_{j=1}^{k'-1} c_j + \left(b - a_m - \sum_{j=1}^{k'-1} a_j\right) \frac{c_{k'}}{a_{k'}} + a_m \frac{c_k}{a_k} \leq \sum_{j=1}^{k-1} c_j + a_m \frac{c_k}{a_k} + \left(b - a_m - \sum_{j=1}^{k-1} a_j\right) \frac{c_{k-1}}{a_{k-1}}$$

zeigen können. Dies aber ergibt sich durch iterierte Anwendung des obigen Satzes 2.1 bei Anwendung auf die Aufgabe (KP_{01}) mit auf $(b-a_m)$ reduzierter Kapazität, für die der kritische Index $\leq k'$ ist. □

Für unser *Beispiel* bedeutet die Überlegung des letzten Satzes: eine potentiell bessere obere Schranke als die Dantzigschranke ergibt sich wegen $a_m = 6$ auch zu

$$U = \max\left\{32, 32 + \left\lfloor 6\frac{12}{6} + (3-6)\frac{10}{4} \right\rfloor\right\} = \max\{32, 36\} = 36.$$

Die Minimalerweiterung

Ebenfalls mit dem Aufwand $\mathcal{O}(n)$ kann man eine gewichtsmäßig minimale Erweiterung anstreben, die bei Nichtexistenz die oben genannte Verschärfung der oberen Schranke zulässt. Man berechnet in der Prozedur den eben verwendeten Index

$m = \arg\min\{a_j : j \geq k\}$ und nimmt, falls a_m noch zur Restkapazität der Dantziglösung, $b - \bar{a}_{k-1}$, passt, den Gegenstand m noch zur Lösung dazu. Geht dies nicht, so verwendet man die soeben hergeleitete Schrankenverschärfung. Dies führt zur

Prozedur LUDM (Lower Upper Dantzig Minimal)

Input: (KP_{01}^{+})

Output: Kritischer Index k, obere Schranke U sowie eine zulässige Lösung x' von (KP_{01}) mit Zielfunktionswert Z.

1. **Start:** Setze $Z = 0$ und $k = n + 1$.
2. **Kritischer Index**

   Setze $Z = 0$. Für $j = 1, ..., n$ führe aus:

   Falls $a_j > b$, setze $x'_j = 0$, $k = j$ und gehe zu Schritt 2,

   andernfalls setze $x'_j = 1$, $Z = Z + c_j$ und $b = b - a_j$.
3. **Schranken**

   Setze $U := Z + \lfloor \zeta_k b \rfloor$

   setze $am = a_k$ und $m = k$.

   Für $j = k + 1, ..., n$ führe aus:

   setze $x'_j = 0$

   Ist $a_j < am$, so setze $m = j$

   Ist $a_m > b$, so setze $U = \max\left\{Z, Z + \left\lfloor a_m \frac{c_k}{a_k} + (b - a_m)\frac{c_{k-1}}{a_{k-1}} \right\rfloor\right\}$,

   sonst setze $x'_m = 1$ und $Z = Z + c_m$.

In unserem *Beispiel* ergibt sich wegen der Nichtexistenz einer echten Dantzig-Erweiterung wieder $Z = 32$ und $U = 36$.

2.1.3 Obere Schranken durch Fallunterscheidung

In der Prozedur DDR haben wir die obere Schranke U durch eine Fallunterscheidung und Modellierung an der Dantziglösung potentiell verbessern können. Wir wollen diese Idee erneut aufgreifen und verfeinern.

Verallgemeinerte Duzinski/Walukiewicz Schranke

Wir modellieren nun die zu berechnende obere Schranke an einer beliebigen *Erweiterung* der Dantziglösung x'' mit Zielfunktionswert z'', z.B. an der Dantziglösung selber oder an der *Gierigen* Lösung. Dazu betrachten wir die folgende Fallunterscheidung: Betrachtet werde eine *optimale* zulässige Lösung x zu (KP_{01}).

- Entweder es ist $x = x''$
- Oder es ist $x \neq x''$ und $x_k = 1$. Dann muss eine der Komponenten x_j, $j = 1, ..., k-1$, den Wert null annehmen.
- Oder es ist $x \neq x''$ und $x_k = 0$. Dann weicht x an einer Stelle von x'' ab, bei der diese null ist. Würde sie nur an Stellen von x'' abweichen, an der diese eins ist, so wäre ihr Zielwert kleiner als der von x'' und sie wäre damit nicht optimal.

Dies führt uns zu folgender Schrankenkonstruktion

$$U_{VD} = \max \left\{ \begin{array}{c} z'' \\ \min\left\{U_k^1, \max\left\{U_j^0 : j < k\right\}\right\}, \\ \min\left\{U_k^0, \max\left\{U_j^1 : x_j'' = 0,\ j > k\right\}\right\} \end{array} \right\}$$

wobei

U_j^1 eine beliebige obere Schranke für das Rucksackproblem

$\left(KP_j^1\right) : (KP_{01})$ mit Zusatzrestriktion $x_j = 1$ und

U_j^0 eine solche für das Rucksackproblem

$\left(KP_j^0\right) : (KP_{01})$ mit Zusatzrestriktion $x_j = 0$

sei.

Wir nennen die oben genannte obere Schranke U_{VD} für (KP_{01}) **Verallgemeinerte Duzinski/Walukiewicz Schranke.** Sie hat je nach Aufwand für die Berechnung der einzelnen lokalen oberen Schranken in der Regel einen Aufwand von $\mathcal{O}(n)$ bis $\mathcal{O}(n^2)$.

Angewendet auf unser *Beispiel* ergibt sich jetzt wegen Nichterweiterbarkeit der Dantziglösung $x'' = (1, 1, 1, 1, 0, 0)$ mit $z'' = 32$ folgende Rechnung, wenn wir für U_j^1 bzw U_j^0 jeweils die Dantzigschranke für die beschriebenen Rucksackprobleme $\left(KP_j^1\right)$ bzw $\left(KP_j^0\right)$ wählen. In diesem Fall können wir die konkreten Schrankenwerte unmittelbar am Beispiel zur Prozedur DDR ablesen. Es ist

$$U_1^0 = 32,\ U_2^0 = 34,\ U_3^0 = 36,\ U_4^0 = 35,\ U_5^0 = 35, U_5^1 = 36,\ U_6^1 = 24$$

Von daher ergibt sich für unsere Schranke

$$\begin{aligned} U_{VD} &= \max\{32, \min\{36, \max\{32, 34, 36, 35\}\}, \min\{35, \max\{24\}\}\} \\ &= \max\{32, 36, 24\} = 36 \end{aligned}$$

Betrachten wir wichtige Spezialfälle der Schranke U_{VD}.

Verallgemeinerte Müller-Merbach Schranke Betrachten wir zunächst die Situation, bei der x'' die *normale Dantziglösung* mit Zielwert $\bar{c}_{k-1}$ darstellt und die Schranken U_k^1 und U_k^0 trivial, also unendlich sind. Dann vereinfacht sich die Formal zu

$$\begin{aligned} U &= \max \left\{ \begin{array}{c} \bar{c}_{k-1} \\ \min\left\{U_k^1, \max\left\{U_j^0 : j < k\right\}\right\}, \\ \min\left\{U_k^0, \max\left\{U_j^1 : \ j > k\right\}\right\} \end{array} \right\} \\ &= \max\left\{\bar{c}_{k-1}, U_j^0\,(j < k), U_j^1\,(j > k)\right\} \end{aligned}$$

bzw

$$U_{VM} = \max\{\bar{c}_{k-1}\} \cup \left\{U^j : j \neq k\right\}$$

wobei wir $U^j = U_j^0$ gesetzt haben für $j < k$ und $U^j = U_j^1$ für $j > k$. Wir nennen diese obere Schranke **Verallgemeinerte Müller-Merbach Schranke.**

Dies ist übrigens genau die Schranken-Konstruktion, die wir bereits in der Prozedur DDR verwendet haben. Das damals gerechnete Beispiel gilt auch hier.

Martello/Toth Schranke U_2 In der Systematik von Martello und Toth erhält die Dantzigschranke die Bezeichung U_1.

$$U_1 = U_D := \left\lfloor \sum_{j=1}^{k-1} c_j + \frac{c_k}{a_k}\left(b - \sum_{j=1}^{k-1} a_j\right)\right\rfloor =: \bar{c}_{k-1} + \zeta_k\,(b - \bar{a}_{k-1})$$

Die in dieser Systematik U_2 genannte obere Schranke erhält man also Spezialfall der Schranke U_{VD}.

Unterstellen wir dazu, dass die eben festgelegten Schranken U^j alle trivial, also unendlich sind. Dann nimmt die Formel für U_{VD} die folgende Form an:

$$U = \max\left\{z'', U_k^1, U_k^0\right\}$$

Weil aber z'' der Zielwert einer zulässigen Lösung zu (KP_k^0) ist, also $x_k = 0$ gilt, folgt $z'' \leq U_k^0$ und daher vereinfacht sich die Formel noch zu

$$U = \max\left\{U_k^1, U_k^0\right\}$$

Nun haben wir im zuletzt genannten Korollar gerade zwei solche Schranken U_k^1, U_k^0 hergeleitet. Diese können wir verwenden und erhalten als neue Schranke

$$\begin{aligned} U_2 &= \max\left\{\left\lfloor \sum_{j=1}^{k} c_j + \frac{c_{k-1}}{a_{k-1}}\left(b - \sum_{j=1}^{k} a_j\right)\right\rfloor, \left\lfloor \sum_{j=1}^{k-1} c_j + \frac{c_{k+1}}{a_{k+1}}\left(b - \sum_{j=1}^{k-1} a_j\right)\right\rfloor\right\} \\ &= \max\left\{\bar{c}_k + \zeta_{k-1}\left(b - \bar{a}_k\right), \bar{c}_{k-1} + \zeta_{k+1}\left(b - \bar{a}_{k-1}\right)\right\} \end{aligned}$$

Diese Schranke ist potentiell besser als die Dantzigschranke. Denn derzweite Teil der Formel ist offensichtlich kleiner gleich U_D, für den ersten Teil zeigt dies die folgende Rechnung

$$\begin{aligned} & \sum_{j=1}^{k-1} c_j + \frac{c_k}{a_k}\left(b - \sum_{j=1}^{k-1} a_j\right) - \left(\sum_{j=1}^{k} c_j + \frac{c_{k-1}}{a_{k-1}}\left(b - \sum_{j=1}^{k} a_j\right)\right) \\ = \; & -c_k + \left(\frac{c_k}{a_k} - \frac{c_{k-1}}{a_{k-1}}\right)\left(b - \sum_{j=1}^{k-1} a_j\right) + \frac{c_{k-1}}{a_{k-1}} a_k \\ = \; & a_k\left[-\frac{c_k}{a_k} + \frac{1}{a_k}\left(\frac{c_k}{a_k} - \frac{c_{k-1}}{a_{k-1}}\right)\left(b - \sum_{j=1}^{k-1} a_j\right) + \frac{c_{k-1}}{a_{k-1}}\right] \\ = \; & a_k\left[\left(\frac{c_{k-1}}{a_{k-1}} - \frac{c_k}{a_k}\right)\left(1 - \frac{1}{a_k}\left(b - \sum_{j=1}^{k-1} a_j\right)\right)\right] \\ = \; & \left(\frac{c_{k-1}}{a_{k-1}} - \frac{c_k}{a_k}\right)\left(\sum_{j=1}^{k} a_j - b\right) \geq 0 \end{aligned}$$

Die Schranke U_2 berechnet sich mit einem Aufwand von $\mathcal{O}(1)$, wenn die Zahlen $\bar{c}_k,\ \zeta_{k-1},\ \bar{a}_k,\ \bar{c}_{k-1},\ \zeta_{k+1},\ \bar{a}_{k-1}$ bereits bekannt sind bzw. von $\mathcal{O}(n)$, wenn diese zuvor berechnet werden müssen.

Im Beispiel: $U_2 = \max\left\{44 + \lfloor 2.5 \cdot (-3)\rfloor, 32 + \lfloor 1.2 \cdot 3\rfloor\right\} = 36$

Martello/Toth Schranke U_3 Jedes der beiden Probleme (KP_k^1) und (KP_k^0) mit fixiertem x_k bildet für sich wieder ein 0-1 Knapsackproblem. Diese Rucksack Probleme erben die Ordnung und die Form von (KP_{01}).

Für beide Probleme kann also, unter Berücksichtigung dieser Tatsache, ein eigenständiger kritischer Index ermittelt und daraus je eine Dantzigschranke berechnet werden. Auf diese Weise erhält man obere Schranken U^1 für (KP_k^1) und U^0 für (KP_k^0), die gemäß der Formel $U = \max\left\{U_k^1, U_k^0\right\}$ zur Berechnung einer oberen Schranke für

(KP_{01}) herangezogen werden können. Diesen Spezialfall der Schranke U_{VD} nennen wir U_3

$$\begin{aligned} U_3 &= \max\left\{\left\lfloor \sum_{j=1}^{k'-1} c_j + \frac{c_{k'}}{a_{k'}}\left(b - \sum_{j=1}^{k'} a_j\right)\right\rfloor, \left\lfloor \sum_{j=1}^{k''-1} c_j + \frac{c_{k''}}{a_{k''}}\left(b - \sum_{j=1}^{k''-1} a_j\right)\right\rfloor\right\} \\ &= \max\left\{\bar{c}_{k'-1} + \zeta_{k'}\left(b - \bar{a}_{k'-1}\right), \bar{c}_{k''-1} + \zeta_{k''}\left(b - \bar{a}_{k''-1}\right)\right\} \end{aligned}$$

wobei wir mit k' den kritischen Index der Aufgabe (KP_k^1) und mit k'' den kritischen Index der Aufgabe (KP_k^0) bezeichnet haben. **Beachte:** Es gilt immer $U_3 \leq U_2 \leq U_1$.

Im Beispiel gilt: $k' = 4,\ k'' = 6,\ U_3 = \max\left\{34 + \lfloor 2.5 \cdot 1 \rfloor, 32 + \lfloor 1.2 \cdot 3 \rfloor\right\} = 36$

2.1.4 Schranken unter Benutzung dualer Variablen

Wir haben gesehen, dass wir die Berechnung einer oberen Schranke dadurch verbessern können, dass wir Fallunterscheidungen einführen. Wir wollen nun alternative Schranken $U_j^0,\ U_j^1$ einbringen in die Formel für die Schranke U_{VD}, die auf der Dualitätstheorie beruhen.

Betrachten wir also erneut die Aufgabe (KP_{01}) mit geordneten Nutzen-Gewichts-Koeffizienten ζ_j, $j = 1, .., n$, und bekanntem kritischem Index k. Gemäß Formel (1.7) lauten die Zielfunktionskoeffizienten im optimalen Simplextableau

$$c_j^* := \left\{ \begin{array}{l} c_j - \zeta_k a_j, \text{ falls } j = 1, ..., k-1 \\ -c_j + \zeta_k a_j, \text{ falls } j = k+1, ..., n \end{array} \right. \geq 0 \qquad (j \neq k)$$

und der optimale Zielfunktionswert

$$z^* := \bar{c} + \zeta_k \bar{b}$$

wobei wir zur Abkürzung wieder $\bar{c} := \sum_{j=1}^{k-1} c_j$, $\bar{a} := \sum_{j=1}^{k-1} a_j$ und $\bar{b} := b - \bar{a}$ gesetzt haben. Die Koeffizienten c_j^* heißen bekanntlich **optimale Lagrange-Multiplikatoren.** Wir setzen noch zur Abkürzung

$$c_k^* = c_k - \zeta_k a_k = -c_k + \zeta_k a_k = 0$$

was in diesem Fall auch den optimalen Zielfunktionskoeffizienten zum Index k widergibt.

Wir halten fest: Die zulässige Dantziglösung x' hat den Zielfunktionswert $z' = \bar{c}$.

Müller-Merbach Schranke Wir haben anlässlich der Formeln (1.11) und (1.12) festgestellt, dass jede zu (KP_{01}) zulässige Lösung x, für die x_j von der Dantziglösung x' abweicht, $j \neq k$, einen Zielfunktionswert von höchstens

$$\left\lfloor \sum_{j=1}^{k-1} c_j + \zeta_k \bar{b} - (c_i - \zeta_k a_i) \right\rfloor$$

besitzen kann, wenn $j < k$ bzw von

$$\left\lfloor \sum_{j=1}^{k-1} c_j + \zeta_k \bar{b} - (-c_i + \zeta_k a_i) \right\rfloor$$

wenn $j > k$ gilt. Damit ergibt sich also für die Probleme $\left(KP_j^0\right)$ und $\left(KP_j^1\right)$, $j = 1, ..., n$, jeweils als sehr einfach zu berechnende obere Schranke

$$u_j := \left\lfloor z^* - c_j^* \right\rfloor$$

Setzen wir diese in die verallgemeinerte Müller-Merbach Formel $U_{VM} = \max\{\bar{c}\} \cup \{U^j : j \neq k\}$ ein, so erhalten wir die eigentliche **Müller-Merbach Schranke**

$$U_{MM} := \max\left\{\bar{c}, \left\lfloor z^* - \min\left\{c_j^* : j \neq k\right\}\right\rfloor\right\}$$

Die Müller-Merbach Schranke ist mit einem Aufwand von $\mathcal{O}(n)$ zu berechnen. Sie ist von der Konstruktion her potentiell besser als die Dantzigschranke: $U_{MM} \leq U_D$, da $u_j \leq U_D$ für alle $j \neq k$ sowie $\bar{c} \leq U_D$.

Wir sprechen sie analog zur Einteilung von Martello und Toth auch als **Schranke** U_4 an.

Wir wenden die Formeln an auf unser Beispiel: Mit $b = 12$ lauten die Daten der Aufgabe

Gegenstand j	1	2	3	4	5	6
Nutzen c_j	8	8	6	10	12	12
Gewicht a_j	1	2	2	4	6	10

Für den kritischen Index $k = 5$ gilt insbesondere $\zeta_5 = 2$. Wir kennen bereits die Dantzigschranke und den Zielwert der Dantziglösung: $z^* = 38$, $\bar{c} = 32$. Es ergibt sich dann wegen $c^* = (6, 4, 2, 2, 0, 8)$

$$U_{MM} = \max\{32, 38 - 2\} = 36 < 38 = U_D$$

Duzinski/Walukiewicz Schranke Analog können wir nun mit der allgemeineren Formel

$$U_{VD} = \max \left\{ \begin{array}{c} z'' \\ \min\left\{U_k^1, \max\left\{U_j^0 : j < k\right\}\right\}, \\ \min\left\{U_k^0, \max\left\{U_j^1 : x_j'' = 0,\ j > k\right\}\right\} \end{array} \right\}$$

verfahren und erhalten so die eigentliche **Duzinski/Walukiewicz Schranke**

$$U_{DW} = \max \left\{ \begin{array}{c} z'' \\ \min\left\{U_k^1, \left\lfloor z^* - \min\left\{c_j^* : j < k\right\}\right\rfloor\right\}, \\ \min\left\{U_k^0, \left\{\left\lfloor z^* - \min\left\{c_j^* : x_j'' = 0,\ j > k\right\}\right\rfloor\right\}\right\} \end{array} \right\}$$

Es ist klar, dass die Duzinski/Walukiewicz Schranke, bei gleicher Wahl der Schranken U_k^1 und U_k^0, potentiell besser ist als die Müller-Merbach Schranke (denn z'' ist Zielwert einer zulässigen Lösung, also gilt $z'' \leq U_{MM}$)! Im Original sind die Schranken U_k^1 bzw U_k^0 die Dantzigschranken für die Rucksackprobleme (KP_k^1) bzw (KP_k^0), wir wollen hier aber beliebige Schranken dieser Aufgaben zulassen.

Wir sprechen U_{DW} mit den Dantzigschranken analog zur Systematik von Martello und Toth auch als **Schranke** U_5 an. Es gilt $U_5 \leq U_4 \leq U_1$.

Angewendet auf unser Beispiel ergibt sich jetzt wegen $x'' = (1, 1, 1, 1, 0, 0)$ mit $z'' = 32$ folgendes, wenn wir für U_k^1 bzw U_k^0 jeweils die Dantzigschranke für die beschriebenen Rucksackprobleme (KP_k^1) bzw (KP_k^0)wählen:

Zunächst findet sich wie bereits bei der Anwendung der Prozedur DDR berechnet: $U^1 = 36$ und $U^0 = 35$. Damit erhält man aus $c^* = (6, 4, 2, 2, 0, 8)$

$$U_{DW} = \max\left\{\min\left\{36, (38-2)\right\}, \min\left\{35, \max\left\{(38-8)\right\}\right\}, 32\right\} = 36 \leq U_{MM}$$

2.2 Verbessertes Preprocessing

Im Überblick haben wir bislang die folgenden Formeln für spezielle obere Schranken erarbeitet:

$$
\begin{aligned}
U_1 &= U_D := \left\lfloor \sum_{j=1}^{k-1} c_j + \frac{c_k}{a_k}\left(b - \sum_{j=1}^{k-1} a_j\right)\right\rfloor \\
U_2 &= \max\left\{\left\lfloor \sum_{j=1}^{k} c_j + \frac{c_{k-1}}{a_{k-1}}\left(b - \sum_{j=1}^{k} a_j\right)\right\rfloor, \left\lfloor \sum_{j=1}^{k-1} c_j + \frac{c_{k+1}}{a_{k+1}}\left(b - \sum_{j=1}^{k-1} a_j\right)\right\rfloor\right\} \\
U_3 &= \max\left\{\left\lfloor \sum_{j=1}^{k'-1} c_j + \frac{c_{k'}}{a_{k'}}\left(b - \sum_{j=1}^{k'} a_j\right)\right\rfloor, \left\lfloor \sum_{j=1}^{k''-1} c_j + \frac{c_{k''}}{a_{k''}}\left(b - \sum_{j=1}^{k''-1} a_j\right)\right\rfloor\right\} \\
U_4 &= U_{MM} = \max\left\{\bar{c}, \left\lfloor z^* - \min\left\{c_j^* : j \neq k\right\}\right\rfloor\right\} \\
U_5 &= U_{DW} = \max\left\{\begin{array}{c} z'' \\ \min\left\{U_k^1, \left\lfloor z^* - \min\left\{c_j^* : j < k\right\}\right\rfloor\right\}, \\ \min\left\{U_k^0, \left\{\left\lfloor z^* - \min\left\{c_j^* : x_j'' = 0,\ j > k\right\}\right\rfloor\right\}\right\} \end{array}\right\}
\end{aligned}
$$

wobei $1 < k < n$ den kritischen Index von (KP_{01}) bezeichne, k' den von (KP_k^1) und k'' den von (KP_k^0), sofern diese existieren und U_k^1 und U_k^0 Dantzigschranken für (KP_k^1) und (KP_k^0) darstellen. Ferner kennen wir die Schrankenberechnung gemäß Satz 2.3.

Zur Berechnung der Schranken $U_1,\ U_2,\ U_4,\ U_5$ genügt also die Kenntnis des kritischen Index von (KP_{01}), bei der Schranke U_3 müssen zwei kritische Indizes neu ermittelt werden.

Die mit diesen neuen Techniken hergeleiteten oberen Schranken können mit den erweiterten Dantziglösungen kombiniert werden zu neuen LU-Prozeduren, sie können helfen, die Reduzierung der gegebenen Aufgabe zu verbessern. Wir wollen uns im Detail damit beschäftigen.

2.2.1 Verbesserte LU Prozeduren

Wir kennen bislang als Standard die Prozedur LUDD, die die Dantzigschranke und die Dantziglösung berechnet. Ferner kennen wir die Prozeduren LUDG, LUDS und LUDM, die die Berechnung der Dantzigschranke mit schärferen unteren Schranken verbinden. Wir wollen diese Prozeduren weiter verbessern, indem wir die genannten schärferen oberen Schranken einbauen, auch wenn wir dabei ggf. eine Verschlechterung der Laufzeit in Kauf nehmen müssen.

Ausgangspunkt unserer Überlegungen sind natürlich die Dantzigschranke als obere Schranke und der Zielfunktionswert der Dantziglösung als untere Schranke

$$U_1 = \bar{c}_{k-1} + \zeta_k \left(b - \bar{a}_{k-1}\right), \quad Z_1 = \bar{c}_{k-1}$$

mit $\bar{c}_{k-1} = \sum_{j=1}^{k-1} c_j$ und $\bar{a}_{k-1} = \sum_{j=1}^{k-1} a_j$.
Wir erarbeiten nun beispielhaft neue LU Prozeduren.

Schwerpunktlösung und Müller-Merbach Schranke U_4

Man kann die Müller-Merbach Schranke und U_2 gut mit der Prozedur LUDS kombinieren, um diese zu verbessern, ohne den Aufwand von $\mathcal{O}(n)$ zu verschlechtern.

Prozedur LUMS (Lower Upper Müller-Merbach Schwerpunkt)
Input: $\left(KP_{01}^{+}\right)$
Output: Kritischer Index k, obere Schranke U sowie eine zulässige Lösung x' von (KP_{01}) mit Zielfunktionswert Z.

1. **Start:** Setze $c^* = \infty$ und $a_m = \infty$ und $Z = 0$, $k = n + 1$.
2. **Kritischer Index**
   Für $j = 1, ..., n$ führe aus:
   Falls $a_j > b$, setze $x'_j = 0$, $k = j$, und gehe zu Schritt 2.
   sonst setze $x'_j = 1,\ Z = Z + c_j$ und $b = b - a_j$.
   Ist $k = n + 1$, setze $U = Z$, **stopp**, die Aufgabe ist trivial lösbar.
3. **Schranken**
   Für $j = 1, ..., k-1$ bilde $c_j^* = c_j - \zeta_k a_j$ und $c^* = \min\left\{c^*, c_j^*\right\}$.
   setze $\bar{c}_{k-1} = Z$ und $U = \max\left\{\bar{c}_{k-1} + c_k + \lfloor \zeta_{k-1}\left(b - a_k\right)\rfloor, \bar{c}_{k-1} + \lfloor \zeta_{k+1} b\rfloor\right\}$
   Setze $\hat{c}^0 = 0,\ x'_k = 0$ und $j_0 = k$.
   Für $j = k + 1, ..., n$ führe aus:
   Ist $a_j < a_m$ setze $a_m = a_j$
   bilde $c_j^* = -c_j + \zeta_k a_j$ und setze $c^* = \min\left\{c^*, c_j^*\right\},\ x'_j = 0$.
   Ist $a_j \leq b$ und $c_j > \hat{c}^0$, so setze $Z = \bar{c}_{k-1} + c_j,\ \hat{c}^0 = c_j,\ j_0 = j$
   setze $U = \min\left\{U, \max\left\{\lfloor \bar{c}_{k-1} + \zeta_k b - c^*\rfloor, \bar{c}_{k-1}\right\}\right\}$
   Ist $j_0 = k$, setze $U = \min\left\{U, \max\left\{\bar{c}_{k-1}, \bar{c}_{k-1} + \lfloor \zeta_k a_m + \zeta_{k-1}\left(b - a_m\right)\rfloor\right\}\right\}$
   sonst setze $x'_{j_0} = 1$

In unserem Beispiel erhält man, wie bereits oben berechnet, $Z = 32$ und $U = 36$.

Obere Schranke U_5 und Gierige Lösung

Betrachten wir nun Verbesserungsmöglichkeiten für die Prozedur LUDG. Man kann z.B. die Dantzigschranke U_1 durch die schärfere Schranke U_2 ersetzen. Dies ist allerdings noch steigerungsfähig, wenn man gleich die Schranke U_{DW} verwendet, bei der die Schranken U_k^1 bzw U_k^0 diejenigen aus der U_2 Schranke sind. Nähme man alternativ die Schranke U_3 als Ersatz für U_1, wäre dies ebenfalls eine Steigerung gegenüber dem Ersatz durch U_2.

Wiederum potentiell besser ist die Verwendung der Schranke U_{DW}, bei der die Schranken U_k^1 bzw U_k^0 diejenigen aus der U_3 Schranke sind, also die Dantzigschranken für die Aufgaben (KP_k^1) und (KP_k^0). Auch das ist wieder steigerungsfähig, wenn man diese Dantzig-Schranken U_k^1 und U_k^0 jeweils durch die entsprechenden U_2 Schranken ersetzt.

Man sieht, dass man die LU Prozedur durch Kombination der verschiedenen Techniken fast beliebig verschärfen (und verkomplizieren) kann. Wir wollen es beispielhaft bei der zuletzt beschriebenen Situation belassen.

In der folgenden Prozedur beschreibe z den Zielfunktionswert der Gierigen Lösung x von (KP_{01}). Gleichzeitig wollen wir gierige Lösungen x' mit Zielwert z' zu der Aufgabe (KP_k^0) und x'' mit Zielwert z'' zur Aufgabe (KP_k^1)berechnen, da wir ja nach Konstruktion über deren kritische Indizes k' und k'' verfügen.

Dann ist offensichtlich

$$Z := \max\{z, z', z''\}$$

eine untere Schranke für (KP_{01}), die potentiell besser ist als die untere Dantzigschranke $\bar{c}_{k-1}$. Man beachte aber, dass immer $z = z''$ gilt.

Die folgende LU-Prozedur zeigt, dass all dies mit einem Aufwand von $\mathcal{O}(n)$ zu berechnen ist.

Prozedur LU5G (Lower Upper 5 Gierig)

Input: $\left(KP_{01}^{+}\right)$

Output: Kritischer Index k, obere Schranke U sowie eine zulässige Lösung x von (KP_{01}) mit Zielfunktionswert Z.

1. **Start:** Setze $c' = c'' = \infty$ und $z = 0$ sowie $k = n + 1$

2. **Kritischer Index** k

   Für $j = 1, ..., n$ führe aus:

   Falls $a_j > b$, setze $x_j = 0$, $k = j$ und gehe zu 3.

   Andernfalls $x_j = 1$, $z = z + c_j$, $b = b - a_j$.

   Ist $k = n + 1$, setze $U = z$, **stopp**

3. **Gierige Lösung** x

   Für $j = 1, ..., k-1$ setze $c_j^* = c_j - \zeta_k a_j$ und $c' = \min\left\{c', c_j^*\right\}$,

   setze $\bar{c}_{k-1} = z$, $\bar{b}_{k-1} = b$, $x_k = 0$, $z^* = z + \zeta_k b$

   Für $j = k + 1, ..., n$ führe aus:

   Falls $a_j > b$, setze $x_j = 0$, bilde $c_j^* = -c_j + \zeta_k a_j$ und setze $c'' = \min\left\{c'', c_j^*\right\}$

   andernfalls setze $x_j = 1$, $z = z + c_j$ und $b = b - a_j$.

4. **Kritischer Index** k'

   Setze $k' = n + 1$, $b' = \bar{b}_{k-1}$.

   Für $j = k + 1, ..., n$, führe aus:

   Falls $a_j > b'$, setze $k' = j$ und gehe zu 5.

   Andernfalls setze $b' = b' - a_j$.

   Ist $k' = n + 1$, setze $U_k^0 = z$ und gehe zu 6.

5. **Schranke** U_k^0

   Setze $U_k^0 := z + \max\left\{\left\lfloor \frac{c_{k'+1}}{a_{k'+1}} b' \right\rfloor, c_{k'} + \left\lfloor \frac{c_{k'-1}}{c_{k'-1}} (b' - a_{k'}) \right\rfloor\right\}$, falls $k' - 1 \neq k$,

   andernfalls setze $U_k^0 := z + \max\left\{\left\lfloor \frac{c_{k'+1}}{a_{k'+1}} b' \right\rfloor, c_{k'} + \left\lfloor \frac{c_{k'-2}}{c_{k'-2}} (b' - a_{k'}) \right\rfloor\right\}$

6. **Kritischer Index** k''

   Setze $x_k'' = 1$, $k'' = 0$, $z'' = \bar{c}_{k-1} + c_k$, $b'' = \bar{b}_{k-1} - a_k \, (< 0)$.

   Für $j = k - 1, ..., 1$ (step -1) führe aus:

   Falls $b'' + a_j \geq 0$, setze $z'' = z'' - c_j$ und $b'' = b'' + a_j$, $k'' = j$ und gehe zu 7.

   andernfalls setze $x_j'' = 0$, $z'' = z'' - c_j$ und $b'' = b'' + a_j$.

   Ist $k'' = 0$, setze $U_k^1 = -\infty$ und gehe zu 8.

7. **Gierige Lösung** x'', **Schranke** U_k^1

   Setze $x_{k''}'' = 0$ und $U_k^1 := z'' + \max\left\{\left\lfloor \frac{c_{k''+1}}{a_{k''+1}} b'' \right\rfloor, c_{k''} + \left\lfloor \frac{c_{k''-1}}{c_{k''-1}} (b'' - a_{k''}) \right\rfloor\right\}$, falls

   $k''+1 \neq k$, andernfalls setze $U_k^1 := z''+\max\left\{\left\lfloor \frac{c_{k''+2}}{a_{k''+2}} b'' \right\rfloor, c_{k''} + \left\lfloor \frac{c_{k''-1}}{c_{k''-1}} (b'' - a_{k''}) \right\rfloor\right\}$

Für $j = k'' + 1, ..., n,\ j \neq k$ führe aus:

Falls $a_j > b'$, setze $x''_j = 0$.

Andernfalls setze $x''_j = 1,\ z'' = z'' + c_j$ und $b'' = b'' - a_j$.

8. **Globale Schranken**

Bilde $Z = \max\{z, z''\}$

Setze $U' = \min\{U^1_k, \lfloor z^* - c' \rfloor\}$ und $U'' = \min\{U^0_k, \lfloor z^* - c'' \rfloor\}$ sowie $U = \max\{U', U'', z\}$

Ist $Z = z''$, setze $x_j = x''_j$ für $j = k'', ..., n$

Beispiel Wir wenden die Prozedur an auf unser Beispiel: Mit $b = 12$ lauten die Daten der Aufgabe

Gegenstand j	1	2	3	4	5	6
Nutzen c_j	8	8	6	10	12	12
Gewicht a_j	1	2	2	4	6	10

Wir unterstellen ferner $c_0 = \infty,\ a_0 = 0,\ c_7 = 0,\ a_7 = \infty$.

2. Es ist $k = 5,\ x_1 = ... = x_4 = 1,\ z = 32$ und $b = 3$. Ferner gilt $c' = 2$.

3. Es gibt keine echte Erweiterung zur Dantziglösung. Es gilt $x_5 = x_6 = 0,\ c'' = 8,\ z^* = 38$.

4. Wir setzen $k' = 7,\ b' = 3$. Wir finden für $j = 6 : a_6 = 10 > 3,\ k' = 6,$

5. Setze

$$\begin{aligned} U^0_5 \quad &: \quad = z + \max\left\{ \left\lfloor \frac{c_{k'+1}}{a_{k'+1}} b' \right\rfloor, c_{k'} + \left\lfloor \frac{c_{k'-1}}{c_{k'-1}} (b' - a_{k'}) \right\rfloor \right\} \\ &= \quad 32 + \max\left\{ \left\lfloor \frac{0}{\infty} \cdot 3 \right\rfloor, 12 + \left\lfloor \frac{10}{4} (3 - 10) \right\rfloor \right\} \\ &= \quad 32 + \max\{0, 12 - 18\} = 32 \end{aligned}$$

6. Setze $x''_5 = 1,\ k'' = 0,\ z'' = 32 + 12 = 44,\ b'' = 3 - 6 = -3$. Wir betrachten $j = 4 :$ Es ist $b'' + a_4 = -3 + 4 > 0$, daher setze $z'' = 44 - 10 = 34,\ b'' = -3 + 4 = 1,\ k'' = 4$.

7. setze $x''_4 = 0$ und

$$\begin{aligned} U^1 \quad &: \quad = z'' + \max\left\{ \left\lfloor \frac{c_{k''+1}}{a_{k''+1}} b'' \right\rfloor, c_{k''} + \left\lfloor \frac{c_{k''-1}}{c_{k''-1}} (b'' - a_{k''}) \right\rfloor \right\} \\ &= \quad 34 + \max\left\{ \left\lfloor \frac{12}{10} \cdot 1 \right\rfloor, 10 + \left\lfloor \frac{6}{2} (1 - 4) \right\rfloor \right\} \\ &= \quad 34 + \max\{1, 1\} = 35 \end{aligned}$$

Wir betrachten $j = 6$. Weil $a_6 > b''$ gilt, setzen wir $x_6'' = 0$.

8. Es ist $Z = \max\{z, z''\} = \max\{32, 34\} = 34$. Wir bilden $U' = \min\{U_5^1, \lfloor z^* - c' \rfloor\} = \min\{35, \lfloor 38 - 2 \rfloor\} = 35$ und $U'' = \min\{U_5^0, \lfloor z^* - c'' \rfloor\} = \min\{32, \lfloor 38 - 8 \rfloor\} = 30$. Damit gilt $U = \max\{U', U'', z\} = \max\{35, 30, 32\} = 35$. Wegen $Z = z''$ setzen wir $x^T = (1, 1, 1, 0, 1, 0)$.

Martello/Toth Schranke U_6

Wir haben gesehen, dass wir über binäre Fallunterscheidung die Dantzigschranke durch das Maximum zweier (Dantzig-) Schranken ersetzen können. Jede dieser Schranken kann ggf wieder durch (binäre) Fallunterscheidung verbessert werden. Wir haben dieses "Prinzip der fortgesetzten binären Fallunterscheidung" ansatzweise bereits bei der Konstruktion unserer letzten LU Prozedur angewendet. Zu beobachten ist, dass dabei natürlich immer mehr (benachbarte) Indizes in die Betrachtung einbezogen werden. Im folgenden wollen wir untersuchen, wie man diese Idee systematisch anwenden kann. Dazu betrachten wir zunächst noch einmal das Beispiel der Schranke U_2.

U_2 unterstellt die Fallunterscheidung, dass die kritische Variable k einer alternativen Fixierung unterworfen wird: Man betrachtet (KP_{01}) zuerst mit der Zusatzrestriktion $x_k = 1$ und dann mit der Zusatzrestriktion $x_k = 0$ und berechnet für beide Fälle die obere Schranke nach Korollar 2.2

$$U_2 = \max\left\{ \left\lfloor \sum_{j=1}^{k} c_j + \frac{c_{k-1}}{a_{k-1}} \left(b - \sum_{j=1}^{k} a_j \right) \right\rfloor , \left\lfloor \sum_{j=1}^{k-1} c_j + \frac{c_{k+1}}{a_{k+1}} \left(b - \sum_{j=1}^{k-1} a_j \right) \right\rfloor \right\}$$

Im ersten Fall spielt also der Index $k-1$ eine hervorgehobene Rolle, im zweiten der Index $k+1$. Insgesamt spielt sich die Schrankeberechnung damit im Index-Intervall $[k-1, k+1]$ ab.

Martello und Toth ziehen aus der genannten Beobachtung die Idee, nicht nur eine Variable, die kritische, einer Fallunterscheidung in der Wertzuweisung zu unterwerfen, sondern systematisch alle Variablen in einem (kleinen) Bereich um die kritische Variable herum. Es werde ein Bereich $[r, t]$ ausgewählt mit $r, t \in N$, $r \leq k \leq t$ und mit Wertzuweisungen für $x_j \in \{0, 1\}$ enumeriert. Es sei $s := t - r + 1 \geq 1$, sodass für z.B. $s = 1$ sich $r = t = k$ ergibt.

Führt man *alle denkbaren* binären Wertzuweisungen durch, so ergeben sich insgesamt 2^s Wertzuweisungen $(\bar{x}_r, ..., \bar{x}_t)$ für $(x_r, ..., x_t)$. Für jede dieser Vorabzuweisungen kann das Rucksackproblem $\left(KP_{(\bar{x}_r, ..., \bar{x}_t)}\right)$ betrachtet werden, das aus der ursprünglichen Aufgabe zusammen mit den Fixierungen $x_r = \bar{x}_r, ..., x_t = \bar{x}_t$ entsteht. Insbesondere ergibt das Maximum der berechneten oberen Schranken der 2^s

eingeschränkten Probleme eine obere Schranke für (KP_{01}). Die Güte des Ergebnisse hängt natürlich von der Wahl von $[r,t]$ und von der Wahl der Formel für die einzelnen oberen Schranken ab.

Zu beachten ist, dass die Aufgabe $\left(KP_{(\bar{x}_r,...,\bar{x}_t)}\right)$ i.a. nicht im Standardformat vorliegen muss. Es können einzelne Gewichte zu groß sein, um in den Rest-Rucksack zu passen, es kann durch die Festlegungen $(x_r,...,x_t) = (\bar{x}_r,...,\bar{x}_t)$ ein unlösbares oder ein lösbares Problem entstehen! Deshalb muss bei der Berechnung der oberen Schranke von $\left(KP_{(\bar{x}_r,...,\bar{x}_t)}\right)$ besonders sorgsam vorgegangen werden. Sinnvollerweise setzt man die obere Schranke für nichtlösbare Aufgaben auf $(-\infty)$ fest. Dann kann abschließend einfach das Maximum aller ermittelten oberen Schranken gebildet werden.

Die kritischen Indizes der Aufgaben $\left(KP_{(\bar{x}_r,...,\bar{x}_t)}\right)$ sind nicht bekannt. Deshalb müssen sie berechnet werden, wenn jeweils die Dantzigschranke Verwendung finden soll.

Beispiele Wir überprüfen zwei verschiedene Wahlen von r und t.

Beispiel 1: Zur Berechnung der Schranken kann die LU-Prozedur LUDD herangezogen werden. Wir überprüfen die Vorgehensweise an unserem Beispiel, zunächst mit $r = t = k = 5$

Es sind zwei Vorabzuweisungen zu überprüfen, deren Ergebnisse sich bereits aus der obigen Anwendung der Prozedur DDR im Beispiel ablesen lassen:

1. Für $x_5 = 1$ ergibt sich als kritischer Index $k' = 4$. Als Dantziglösung erhält man $x'^T = (1,1,1,0,1,0)$ mit $z' = 34$ und $u = 34 + \left\lfloor \frac{1}{4}10 \right\rfloor = 36$.
2. Für $x_5 = 0$ ergibt sich als kritischer Index $k' = 6$. Als Dantziglösung ergibt sich $x'^T = (1,1,1,1,0,0)$ und $z' = 32$ und $u = 32 + \left\lfloor \frac{3}{10}12 \right\rfloor = 35$.

Insgesamt erhält man also $L = 34$ als untere Schranke und $U = 36$ als obere Schranke für (KP_{01}).

Beispiel 2: Nun untersuchen wir noch das Beispiel mit $r = k = 5$ und $t = 6$. Es sind $2^2 = 4$ Fälle zu unterscheiden (lexikographische Reihenfolge)

1. $x_5 = 1, x_6 = 1$: Diese Aufgabe ist nicht lösbar, $u = -\infty$
2. $x_5 = 1, x_6 = 0$: Als kritischer Index ergibt sich $k' = 4$. Die Dantziglösung ist $x^T = (1,1,1,0,1,0)$ mit $z = 34$ und $u = 34 + \left\lfloor \frac{10}{4}1 \right\rfloor = 36$.
3. $x_5 = 0, x_6 = 1$: Als kritischer Index ergibt sich $k' = 2$. Die Dantziglösung ist $x^T = (1,0,0,0,0,1)$ mit $z = 20$ und $u = 20 + \left\lfloor \frac{8}{2}1 \right\rfloor = 24$.

4. $x_5 = 0, x_6 = 0$: Die Aufgabe ist trivial lösbar. $u = 32 + \lfloor 0 \rfloor = 32$. Die Dantziglösung ist $x^T = (1, 1, 1, 1, 0, 0)$ mit $z = 32$.

Insgesamt erhält man also wieder $L = 34$ und $U = 36$.

Vermeidung der Neuberechnung kritischer Indizes Martello und Toth schlagen vor, zur Ermittlung der Schranken zu den Aufgaben $\left(KP_{(\bar{x}_r,...,\bar{x}_t)}\right)$ die wenig aufwändige Schrankenberechnung nach Satz 2.1 zu verwenden, weil dann kein lokaler kritischer Index berechnet werden muss. Zur Herleitung von entsprechenden Formeln wenden wir den Satz auf die jetzige Situation an. Wir setzen hier zur Abkürzung $\bar{x}_j = 1$ für $j = 1, ..., r-1$.

Korollar 2.4 *Betrachtet werde die Aufgabe* $\left(KP_{(\bar{x}_r,...,\bar{x}_t)}\right)$ *mit* $1 < r \leq t < n$. *Setze* $\hat{b} := b - \sum_{j=1}^{t} a_j \bar{x}_j$ *und* $\hat{c} = \sum_{j=1}^{t} c_j \bar{x}_j$.

1. *Ist* $\hat{b} < 0$, *so ist* $U = \left\lfloor \hat{c} + \frac{c_{r-1}}{a_{r-1}} \hat{b} \right\rfloor$ *eine sinnvolle obere Schranke.*

2. *Ist* $\hat{b} \geq 0$, *so ist* $U = \left\lfloor \hat{c} + \frac{c_{t+1}}{a_{t+1}} \hat{b} \right\rfloor$ *eine sinnvolle obere Schranke.*

Beweis: Für jedes $i \in \{1, ..., r-1\}$ bzw $i \in \{t+1, ..., n\}$ liefert die Formel aus dem dritten Teils von Satz 2.1 eine obere Schranke für $\left(KP_{(\bar{x}_r,...,\bar{x}_t)}\right)$. Im ersten Fall liegt dabei der kritische Index $\leq r-1$, im zweiten Fall $\geq t+1$. Daher hat man nur die Formel $U = \left\lfloor \sum_{j=1}^{i} c_j + \frac{c_i}{a_i} \left(b - \sum_{j=1}^{i} a_j \right) \right\rfloor$ für $i = r-1$ bzw. $i = t+1$ auf die Aufgabe $\left(KP_{(\bar{x}_r,...,\bar{x}_t)}\right)$ bei $\hat{b} = b - \sum_{j=1}^{t} a_j \bar{x}_j$ anzuwenden, um die Behauptung zu verifizieren. □

Bemerkung: Ist $r = 1$ oder $t = n$, so kann man wieder mit $\left(KP_{01}^{+}\right)$ arbeiten. Ist die Aufgabe $\left(KP_{(\bar{x}_r,...,\bar{x}_t)}\right)$ nicht lösbar, so geben die endlichen Werte U dennoch obere Schranken an, da ja eigentlich $U = -\infty$ gilt. In diesem Sinne gilt das obige Korollar auch für $1 \leq r \leq t \leq n$. Außerdem sei vermerkt, dass man die Schranken des Korollars dahingehend verschärfen kann, dass man in einem keleinen Bereich links und rechts neben $[r, t]$, etwa $[r - \alpha, t + \alpha]$ prüft, ob sich der kritische Index s dort befindet oder nicht. Je nach Ergebnis erhält man eine schärfere obere Schranke.

Beispiel Wir wenden dies in unserem Beispiel an bei $r = 3$, $k = t = 5$. Es sind dann 8 Wertzuweisungen für (x_3, x_4, x_5) zu diskutieren. Die zu betrachtende Aufgabe ist lösbar, wenn die Restkapazität nichtnegativ ist. Wir gehen aus von $L = 32$ und $U = 38$.

1. $(1,1,1)$: kritischer Index links, $u = \lfloor 16 + 28 + \frac{8}{2}(-3) \rfloor = 32$
2. $(1,1,0)$: kritischer Index rechts, $u = \lfloor 16 + 16 + \frac{12}{10}3 \rfloor = 35$
3. $(1,0,1)$: kritischer Index rechts, $u = \lfloor 16 + 18 + \frac{12}{10}1 \rfloor = 35$. Hier ergibt sich gleichzeitig als neue zulässige Lösung die Lösung $x^T = (1,1,1,1,0,1,0)$ mit Zielwert $z = 34$.
4. $(1,0,0)$: kritischer Index rechts, $u = \lfloor 16 + 6 + \frac{12}{10}7 \rfloor = 30$
5. $(0,1,1)$: kritischer Index links, $u = \lfloor 16 + 22 + \frac{8}{2}(-1) \rfloor = 34$
6. $(0,1,0)$: kritischer Index rechts, $u = \lfloor 16 + 10 + \frac{12}{10}5 \rfloor = 32$
7. $(0,0,1)$: kritischer Index rechts, $u = \lfloor 16 + 12 + \frac{12}{10}3 \rfloor = 31$
8. $(0,0,0)$: kritischer Index rechts, $u = \lfloor 16 + 0 + \frac{12}{10}9 \rfloor = 26$

Damit ergibt sich insgesamt $L = 34$ und $U = 35$.

Bei der obigen Rechnung wurde nicht geprüft, ob die in die Rechnung eingehenden Gegenstände überhaupt noch in die Restkapazität nach Fixierung, $\bar{b} = b - \sum_{j=r}^{t} a_j \bar{x}_j$, passen. Man kann dies ansatzweise nachträglich reparieren, wenn man die Fälle, die das Maximum der berechneten oberen Schranken darstellen, nachträglich bzgl dieser Frage checkt. Die globale obere Schranke kann dann ggf noch verbessert werden, wenn man herausfindet, dass die Berechnung der Schranke auf einem Gewicht aufbaut, das nach der Fixierung überhaupt nicht mehr in den Rucksack passt.

In unserem Beispiel ergeben sich folgende nachträgliche Berichtigungen, wenn man in $\left(KP_{01}^{+}\right)$ arbeitet:

1. $(1,1,0)$: kritischer Index rechts, das Gewicht $a_6 = 10$ passt wegen der Restkapazität $\bar{b} = 6$ nach Fixierung nicht in den Rest-Rucksack. Daher berechnet sich die Schranke nun zu $u = \lfloor 32 + \frac{0}{\infty}3 \rfloor = 32$
2. $(1,0,1)$: kritischer Index rechts, das Gewicht $a_6 = 10$ passt wegen $\bar{b} = 4$ nach Fixierung nicht in den Rest-Rucksack. Daher berechnet sich die Schranke nun zu $u = \lfloor 34 + \frac{0}{\infty}1 \rfloor = 34$. Hier ergibt sich gleichzeitig als neue zulässige Lösung die Lösung $x^T = (1,1,1,1,0,1,0)$ mit Zielwert $z = 34$. Wegen der Gleichheit der oberen und der unteren Schranke ist eine Optimallösung gefunden.

Es ergibt sich also in der Nachbereitung $L = U = 34$.

Eine LU Prozedur mit U_6 Die Berechnung der Schranke U_6 liefert, wie im obigen Beispiel zu erkennen, gleichzeitig eine mögliche Verbesserung der unteren Schranke. Man kann daher deren Berechnung zu einer LU Prozedur ausbauen. Dies soll im Folgenden geschehen.

Es möge das Intervall $[r,t]$ mit $1 < r \leq t < n$ vorgegeben sein und es sollen alle Elemente aus $(\bar{x}_r, ..., \bar{x}_t) \in [0,1]^s$ mit $s = t - r + 1$ enumeriert und dazu wie im Beispiel je eine zugehörige lokale obere Schranke für (KP_{01}) mit den Fixierungen $x_r = \bar{x}_r, ..., x_t = \bar{x}_t$ ausgerechnet werden. In der Prozedur wollen wir, wie im Beispiel, die aufzuzählenden Elemente aus $[0,1]^s$, beginnend mit $(1, ..., 1)$ *lexikographisch abfallend* enumerieren. Die zugehörigen Schranken berechnen wir mit Korollar 2.4. Dabei ist lediglich zu beachten, ob der Wert $\hat{b} := b - \sum_{j=1}^{t} \bar{x}_j a_j$ negativ ist oder nicht. Im ersten Fall liegt der kritische Index der Aufgabe (KP_{01}) mit den Fixierungen links von r, im zweiten Fall rechts von t und es kann genau die entsprechende Formel für die obere Schranke angewendet werden. Das Maximum der so erhaltenen lokalen oberen Schranken kann simultan berechnet werden. Gleichzeitig kann im Falle von $\hat{b} \geq 0$ beobachtet werden , ob der Wert $\hat{c} := \sum_{j=1}^{t} c_j \bar{x}_j$ eine neue bessere zulässige Lösung anzeigt.

Bleibt zu klären, wie das Durchgehen der Vektoren $(\bar{x}_r, ..., \bar{x}_t) \in [0,1]^s$ in lexikographisch abfallender Ordnung am besten organisiert wird. Eine Möglichkeit dazu ist, sich an der Prozedur HSV zu orientieren, bei der der Verzweigungsbaum mit LIFO links vor rechts durchgegangen wird. Gäbe es dort keinerlei Auslotung, so würden alle $(\bar{x}_r, ..., \bar{x}_t) \in [0,1]^s$ als Blätter des (vollständigen) Verzweigungsbaums genau einmal angelaufen. Die folgende Prozedur orientiert sich also an der Prozedur HSV. Sie ergänzt die gefundene beste zulässige Lösung zum Schluss noch um eine gierige Erweiterung.

Wir formulieren die Prozedur allerdings so, dass Knotenaufgaben, die garantiert unlösbar sind, zwar theoretisch emuneriert, aber praktisch von vorne herein weggelassen werden, da sie keine Beitrag zur Schranke liefern. Das sind die Knoten, für die $\bar{b} := b - \sum_{j=r}^{t} \bar{x}_j a_j < 0$ gilt.

Folgende Notationen werden verwendet:

$\hat{x}$ = laufende Lösung

$\hat{z}$ = Zielwert der laufenden Lösung

$\hat{b}$ = laufende Restkapazität

x = beste bisher bekannte Lösung

z = Zielwert der besten bislang bekannten Lösung $\left(= \sum_{j=1}^{n} c_j x_j\right)$

$\bar{u}$ = lokale obere Schranke

Prozedur LUU6 (LU mit Schranke U_6)
Input: $\left(KP_{01}^{+}\right)$, ferner Zahlen $r, t \in \mathbf{N}$ mit $1 \leq r \leq t \leq n$.
Output: zulässige Lösung x mit Zielwert z, obere Schranke U.

1. **Start:** Setze $z = 0,\ \hat{z} = \sum_{k=1}^{r-1} c_k,\ \hat{a} = \sum_{k=1}^{r-1} a_k,\ \hat{b} = b - \hat{a},\ j = r,\ U = 0$
   setze $x_i = 1$ für $i = 1, ..., r-1,\ x_i = 0$ für $i = t+1, ..., n$
2. **Vorwärtsschritt**
   für $i = j$ bis t führe aus
   ist $\bar{b} = \hat{b} - a_i + \hat{a} \geq 0$, setze $\hat{x}_i = 1$, $\hat{b} = \hat{b} - a_i,\ \hat{z} = \hat{z} + c_i$, sonst $\hat{x}_i = 0$.
   ist $\hat{b} < 0$ so setze $\bar{u} = \left\lfloor \hat{z} + \hat{b}\frac{c_{r-1}}{a_{r-1}} \right\rfloor$, andernfalls $\bar{u} = \left\lfloor \hat{z} + \hat{b}\frac{c_{t+1}}{a_{t+1}} \right\rfloor$
   ist $\bar{u} > U$, setze $U = \bar{u}$
   ist $\hat{z} > z$ und $\hat{b} \geq 0$, so führe aus
   $z = \hat{z}$
   für $k = r$ bis t setze $x_k := \hat{x}_k$
3. **Seitwärtsschritt**
   setze $\hat{x}_t = 0$, $\hat{b} = \hat{b} + a_t,\ \hat{z} = \hat{z} - c_t$
   ist $\hat{b} < 0$ so setze $\bar{u} = \left\lfloor \hat{z} + \hat{b}\frac{c_{r-1}}{a_{r-1}} \right\rfloor$, andernfalls $\bar{u} = \left\lfloor \hat{z} + \hat{b}\frac{c_{t+1}}{a_{t+1}} \right\rfloor$
   ist $\bar{u} > U$, setze $U = \bar{u}$
   ist $\hat{z} > z$ und $\hat{b} \geq 0$, so führe aus
   $z = \hat{z}$
   für $k = r$ bis t setze $x_k = \hat{x}_k$
4. **Rückwärtsschritt**
   bestimme sofern möglich $i = \max\{k < j : \hat{x}_k = 1\}$, sonst gehe zu 5.
   setze $\hat{b} = \hat{b} + a_i,\ \hat{z} := \hat{z} - c_i, \quad \hat{x}_i := 0, \quad j := i + 1$
   gehe zu 2.
5. **Gierige Erweiterung**
   setze $\hat{b} = b - \sum_{i=1}^{t} x_i a_i$
   Für $i = t+1$ bis n führe aus
   ist $a_i \leq \hat{b}$, so setze $x_i = 1$, $\hat{b} = \hat{b} - a_i$, $z = z + c_i$

Man überprüft leicht, dass die obigen Beispiele die Prozedur verifizieren!

2.2.2 Einsatz von Iterativen Verfahren

Zu beobachten ist, dass die Berechnung von U_6 wegen der mit s exponentiell wachsenden Größe von 2^s nur für kleine Zahlen $s = t-r+1$ praktisch erfolgreich angewendet werden kann, z.B. mit $r = k-5$ und $t = k+5$. Im folgenden versuchen wir nun, die Berechnung unterer und oberer Schranken im Sinne der bei der Berechnung von U_6 angewandten Methode auf eine breitere, allgemeinere Basis zu stellen, um auch einen größeren Bereich $[r, t]$ absuchen zu können.

Binärstrategie

Wir hatten die Berechnung der Schranke U_6 über eine immer wieder angewandte binäre Fallunterscheidung motiviert. In der Tat sind, wie gesehen und verwendet, die zu betrachteten Fixierungen der Variabeln im Bereich $[r, t]$ genau die Blätter eines Binärbaums, der durch fortgesetzte binäre Verzweigung entsteht.

Skizze:

Erfreulicherweise kann diese Art der Schrankenberechnung dahingehend verallgemeinert werden, dass von *irgendeinem Binärbaum* ausgegangen werden kann, der dadurch entsteht, dass *ausgewählte noch freie Variablen* an *ausgewählten Stellen des Verzweigungsbaums* durch Fixierung auf 0 oder 1 binär verzweigt werden oder nicht.

Betrachten wir dazu noch einmal U_2. Verzweigt man nach der Fixierung $x_k = 1$ und $x_k = 0$ die dann einbezogenen Variablen x_{k-1} und x_{k+1} noch einmal binär, so entsteht z.B. der Verzweigungsbaum

Erfolgt nun die Schrankenberechnung über Satz 2.1, so entsteht eine Formel, die das Maximum von vier lokalen Schranken bildet, entsprechend den vier Blättern des

Entscheidungsbaums. Wir benennen die so entstehende obere Schranke für (KP_{01}) mit U_{22}

$$U_{22} = \max\{u_1, u_2, u_3, u_4\}$$

wobei sich $u_1, ..., u_4$ wie folgt berechnen:

Bei den beiden linken Knoten des Entscheidungsbaums gelten die Fixierungen $(x_{k-1}, x_k) = (0,1)$ und $(x_{k-1}, x_k) = (1,1)$. Daher berechnen sich zugehörige obere Schranken für (KP_{01}) gemäß Satz 2.1 in den üblichen Bezeichnungen zu

$u_1 = \lfloor \bar{c}_{k-2} + c_k + \zeta_{k-2}\,(b - \bar{a}_{k-2} - a_k) \rfloor$,

falls die Restkapazität $(b - \bar{a}_{k-2} - a_k)$ negativ oder

$u_1 = \lfloor \bar{c}_{k-2} + c_k + \zeta_{k+1}\,(b - \bar{a}_{k-2} - a_k) \rfloor$,

falls die Restkapazität nichtnegativ ist sowie

$u_2 = \lfloor \bar{c}_{k-2} + c_{k-1} + c_k + \zeta_{k-2}\,(b - \bar{a}_{k-2} - a_{k-1} - a_k) \rfloor$.

Bei den beiden rechten Knoten des Entscheidungsbaums gelten die Fixierungen $(x_k, x_{k+1}) = (0,0)$ und $(x_k, x_{k+1}) = (0,1)$. Daher berechnen sich zugehörige obere Schranken für (KP_{01}) gemäß Satz 2.1 in den üblichen Bezeichnungen zu

$u_3 = \lfloor \bar{c}_{k-1} + \zeta_{k+2}\,(b - \bar{a}_{k-1}) \rfloor$ sowie

$u_4 = \lfloor \bar{c}_{k-1} + c_{k+1} + \zeta_{k-1}\,(b - \bar{a}_{k-1} - a_{k+1}) \rfloor$,

falls die Restkapazität $(b - \bar{a}_{k-2} - a_k)$ negativ oder

$u_4 = \lfloor \bar{c}_{k-1} + c_{k+1} + \zeta_{k+2}\,(b - \bar{a}_{k-2} - a_{k+1}) \rfloor$,

falls die Restkapazität nichtnegativ ist.

Für unser Beispiel bedeutet dies: $U_{22} = \max\{35, 35, 32, 36\} = 36$

Der in U_6 angesprochene Binärbaum dagegen verzweigt nacheinander und vollständig alle Variablen x_r bis x_t in der gleichen Reihenfolge. Zugelassen seien nun also ausdrücklich auch *Bäume*, die z. B. durch *Weglassen von Verzweigungen* im diesem Verzweigungsbaum entstehen. Durch Weglassen von eigentlich vorgesehenen Verzweigungen wird der Baum weniger umfangreich und erhält weniger Blätter. Die Rechnung wird kürzer, die zu berechnende obere Schranke wird allerdings auch potentiell weniger scharf.

Bei den nun betrachteten Binärbäumen müssen aber *immer beide* Fixierungen einer betrachteten Variablen auftreten, die auf 0 und die auf 1. Weil jede einzelne Binär-Verzweigung jeweils eine vollständige Fallunterscheidung darstellt und somit jede binäre Wertzuweisung zu $x_r, ..., x_t$ in einem der Blätter des Binärbaums zulässig ist, ergibt sich eine obere Schranke für (KP_{01}) als *Maximum* irgendwie berechneter lokaler oberer Schranken für die zu den *Blättern* gehörigen Restaufgaben $\left(KP_{01}^T\right)$, bei denen (KP_{01}) mit den Zusatzforderungen versehen ist, die die einzelnen Variablen

genau so fixieren, wie sich dies auf dem Rückwärtsweg vom Blatt aus im Binärbaum ergibt und alle anderen Variablen frei lassen. Es sind in einem Blatt also nicht immer alle $x_r, ..., x_t$ fixiert. Günstig ist es, wenn immer ein (zusammenhängender) Bereich $x_r, ..., x_m$ für ein $r \leq m \leq t$ vorliegt, sodass Korollar 2.4 zur Berechnung der lokalen oberen Schranken angewendet werden kann.

Die Restkapazitäten für die Aufgaben $\left(KP_{01}^{T}\right)$ müssen dabei gemäß den auf den Wert 1 vorgenommenen Variablen-Fixierungen festgelegt werden. Unlösbare Restaufgaben erhalten immer die lokale obere Schranke $u = -\infty$.

> Die hier beschriebene Verallgemeinerung der Schrankenberechnung durch Fallunterscheidung lässt also gegenüber der bisherigen Version eine veränderte Reihenfolge der Variablen bei der Fixierung zu sowie eine ungleichmäßig tiefe Verweigung einzelner Stränge des Binärbaumes.

Die Methode LUBB von Martello und Toth Wir können daher die oben dargestellte Berechnungsmöglichkeit der oberen Schranke mit Hilfe von Korollar 2.4 mit der Anwendung des B&B Verfahrens HSV (oder sogar mit anderen B&B Methoden zur Lösung eines 0-1 Rucksack Problems) kombinieren!

Bei der obigen Bearbeitung der Beispielaufgabe fällt auf, dass bei der Berechnung der oberen Schranke immer als erstes der Nutzen der Variablen $x_1, ..., x_{r-1}$ aufaddiert wird, also diese Variablen implizit auf 1 gesetzt werden. Man kann daher neben der Aufgabe $\left(KP_{(\bar{x}_r,...,\bar{x}_t)}\right)$ auch die Aufgabe $(KP[r,t])$ betrachten, bei der zu (KP_{01}) die zusätzlichen Restriktionen $x_1 = x_2 = ... = x_{r-1} = 1$ und $x_{t+1} = ... = x_n = 0$ hinzukommen und deren Kapazität $\bar{b}_r = \left(b - \sum_{j=1}^{r-1} a_j\right)$ lautet. Beide Aufgaben sind in gewisser Weise komplementär: bei $\left(KP_{(\bar{x}_r,...,\bar{x}_t)}\right)$ sind genau die Variablen $x_r, ..., x_t$ fixiert, alle anderen Variablen sind frei, bei $(KP[r,t])$ ist es umgekehrt.

Zulässige Lösungen von $(KP[r,t])$ ergeben durch Ergänzung ausserhalb von $[r,t]$ automatisch auch zulässige Lösungen von (KP_{01}). Wir verfolgen wie bei LUU6 die Zielsetzung, bei der Bearbeitung von $(KP[r,t])$ mit einem gewählten B&B Verfahren *gleichzeitig* eine scharfe obere Schranke für (KP_{01}) zu berechnen. Dies erfolgt nach der Idee von Martello/Toth: innerhalb des Verfahrens erfolgt jeweils eine *vollständige* Verzweigung der Testaufgaben, indem einzelne Variablen fixiert werden. Liegt keine vollständige Verzweigung vor, muss die Verzweigung entsprechend ergänzt werden.

Dies hat zur Folge, dass jede beliebige binäre Wertzuweisung an $x_r, ..., x_t$ zulässig ist bzgl. der Testaufgabe zu einem der *Blätter* des im Verfahren entstandenen Verzweigungsbaums, insbesondere also die Wertzuweisung aus einer Optimallösung von (KP_{01}). Deren Zielfunktionswert wird nach oben abgeschätzt, wenn eine obere

Schranke der entsprechenden Aufgabe $\left(KP_{(\bar{x}_r,...,\bar{x}_t)}\right)$ des zugehörigen Blatts berechnet wird, wobei $(\bar{x}_r, ..., \bar{x}_t)$ die zugehörigen Fixierungen wiederspiegele und die übrigen Variablen aus $x_r, ..., x_t$ frei gelassen werden. Zu beachten ist, dass die Aufgabe $(KP[r,t])$ grundsätzlich lösbar ist, da nach Konstruktion für den kritischen Index k von (KP_{01}) ja $r \leq k \leq t$ gilt.

Es entsteht also durch Anwendung des B&B Verfahrens auf $(KP[r,t])$ ein Entscheidungsbaum, dessen Blätter den dort *ausgeloteten* Testaufgaben entsprechen. Auslotung besteht, wenn die Testaufgabe unlösbar wird oder eine eigens berechnete lokale obere Schranke für $(KP[r,t])$ kleiner gleich der aktuellen unteren Schranke ist.

Damit ist klar, dass eine obere Schranke für (KP) erhalten wird, wenn man das Maximum U aller lokalen oberen Schranken u_T der Testaufgaben T bildet wird, die gemäß den obigen Formeln zu den Blättern des Verzweigungsbaums gehören. Die entsprechende Maximumbildung kann *simultan* zum Durchlauf des B&B Verfahrens erfolgen. Wir nennen diese Vorgehensweise **Methode LUBB** (LU Branch and Bound).

Als konkretes B&B Verfahren zur Ausgestaltung dieser Methode wählen wir im Folgenden die Prozedur HSV in einer verschärften Form, denn wir wollen darin zur Schrankenberechnung grundsätzlich die gegenüber der Dantzigschranke potentiell schärfere Schranke U_2 anwenden.

Die folgende *Prozedur LUHS* setzt dies um. Sie löst $(KP[r,t])$ für $1 \leq r \leq t \leq n$ und berechnet zu jedem Blatt des Entscheidungsbaums eine obere Schranke für $\left(KP_{(\bar{x}_r,...,\bar{x}_{t'})}\right)$, wenn $(\bar{x}_r, ..., \bar{x}_{t'})$ die Fixierungen des Blattes angibt. Zur Berechnung der zuletzt genannten Schranke wird Korollar 2.4 verwendet. Dabei ist zu beachten, dass ein Blatt des Entscheidungsbaum genau dann erreicht ist, wenn das Verfahren einen Rückwärtsschritt einleitet.

Diese lokalen oberen Schranken für $\left(KP_{(\bar{x}_r,...,\bar{x}_t)}\right)$ bezeichnen wir mit $\bar{u}$, die Maximumbildung wird dabei unmittelbar und iterativ vorgenommen. Zu beachten ist, dass man wegen U_2 zur Vermeidung von Fallunterscheidungen die Prozedur HSV auf $\left(KP[r,t]^+\right)$ anwenden muss. Dies kann dadurch erreicht werden, dass man die Daten a_j und c_j für $(KP[r,t])$ in aa_j und cc_j umtauft und dann $(KP[r,t])$ trivial erweitert, ohne die Originaldaten zu überschreiben. Zum Schluss erweitert die Prozedur die bis dahin beste zulässige Lösung in gieriger Weise.

Zu beachten ist, dass anstelle von U_2 natürlich auch eine schärfere Schranke, z. B. U_3 oder eine Mischform aus U_2 und U_3 verwendet werden kann. (Übungsaufgabe)

Prozedur LUHS (LU Horowitz Sahni)

Input: $\left(KP_{01}^{+}\right)$, zulässige Lösung $\bar{x}$ mit Zielwert $\bar{z}$, obere Schranke U. Ferner Zahlen $r, t \in \mathbf{N}$ mit $1 \leq r \leq t \leq n$.

Output: Zulässige Lösung x mit Zielfunktionswert z und obere Schranke U für (KP_{01})

1. **Start:** setze $z = \bar{z},\ \hat{z} = \sum_{k=1}^{r-1} c_k,\ \hat{b} = b - \sum_{k=1}^{r-1} a_k,\ j = r,\ \bar{u} = 0,\ f = 0$

   setze $x_k = 1$ für $k = 1, ..., r-1$ und $x_k = 0$ für $k = t+1, ..., n$.

   für $i = r$ bis t setze $aa_j = a_j$, $cc_j = c_j$. Ferner setze $a_0 = 0$, $c_0 = \infty, a_{n+1} = \infty,\ c_{n+1} = 0$ sowie $aa_{r-1} = 0,\ cc_{r-1} = \infty,\ aa_{t+1} = aa_{t+1} = \infty,\ cc_{t+1} = cc_{t+2} = 0$.

2. **lokale obere Schranke**

   bestimme $s = \min\left\{i : \sum_{k=j}^{i} aa_k > \hat{b}\right\}$. Existiert kein solches s, setze $u = \sum_{k=j}^{t} cc_k$,

   andernfalls bilde

   $$u = \sum_{k=j}^{s-1} cc_k + \max\left\{cc_s + \left\lfloor \frac{cc_{s-1}}{aa_{s-1}} \left(\hat{b} - \sum_{k=j}^{s} aa_k\right)\right\rfloor, \left\lfloor \frac{cc_{s+1}}{aa_{s+1}} \left(\hat{b} - \sum_{k=j}^{s-1} aa_j\right)\right\rfloor\right\}$$

   ist $z \geq \hat{z} + u$, dann berechne $\bar{u} = \max\left\{\bar{u}, \left\lfloor \hat{z} + \hat{b}\frac{c_j}{a_j}\right\rfloor\right\}$ und gehe zu 5.

   ist $f = 0$ und $\hat{z} + u < U$, setze $U = \hat{z} + u$ und $f = 1$

3. **Vorwärtsschritt**

   solange $aa_j \leq \hat{b}$ gilt, führe aus:

   $\hat{b} = \hat{b} - aa_j,\ \hat{z} = \hat{z} + cc_j, \quad \hat{x}_j = 1, \quad j = j + 1$

   ist $j \leq t$, so setze $\hat{x}_j = 0$ und $\bar{u} = \max\left\{\bar{u}, \left\lfloor \hat{z} + c_j + \left(\hat{b} - a_j\right)\frac{c_{r-1}}{a_{r-1}}\right\rfloor\right\}$ sowie

   $j = j + 1$

   ist $j < t$, gehe zu 2.

   ist $j = t$, gehe zu 3.

4. **Aktualisierung**

   ist $\hat{z} > z$, so führe aus

   $z = \hat{z}$

   für $k = r$ bis t setze $x_k = \hat{x}_k$

   Ist $z = U$, **stopp**

   setze $j = t$

setze $\bar{u} = \max\left\{\bar{u}, \left\lfloor \hat{z} + \hat{b}\frac{c_{j+1}}{a_{j+1}} \right\rfloor\right\}$

ist $\hat{x}_t = 1$, setze $\hat{b} = \hat{b} + aa_t,\ \hat{z} = \hat{z} - cc_t,\ \hat{x}_t := 0$ und $\bar{u} = \max\left\{\bar{u}, \left\lfloor \hat{z} + \hat{b}\frac{c_{j+1}}{a_{j+1}} \right\rfloor\right\}$

5. **Rückwärtsschritt**

   bestimme sofern möglich $i = \max\{k < j : \hat{x}_k = 1\}$, sonst gehe zu 6.

   setze $\hat{b} = \hat{b} + aa_i,\ \hat{z} = \hat{z} - cc_i, \quad \hat{x}_i = 0, \quad j = i + 1$

   gehe zu 2.

6. **Gierige Erweiterung**

   ist $\bar{u} < U$, setze $U = \bar{u}$.

   setze $\hat{b} = b - \sum_{i=1}^{t} x_i a_i$

   Für $i = t + 1$ bis n führe aus

   ist $a_i \leq \hat{b}$, so setze $x_i = 1$, $\hat{b} = \hat{b} - a_i$, $z = z + c_i$, sonst $x_i = 0$.

Beispiel Die Daten unseres Beispiels lauten bei $b = 12$ wie bekannt:

Gegenstand j	1	2	3	4	5	6
Nutzen c_j	8	8	6	10	12	12
Gewicht a_j	1	2	2	4	6	10

Wir setzen als bekannte zulässige Lösung nur die Nulllösung $\bar{x}$ mit $\bar{z} = 0$ voraus, als obere Schranke $U = \infty$. Wir wählen $r = 3$ und $t = 5$. Wir wenden die Prozedur an, geben aber gleichzeitig auch verschärfte Schranken an, die aus zusätzlicher Fixierung von Variablen außerhalb von $[r, t]$ und Anwendung von Satz 2.3 entstehen.

1. Zunächst setzen wir $z = 0,\ U = \infty,\ \hat{z} = 16,\ \hat{b} = 9,\ c = cc,\ a = aa,\ c_0 = \infty,\ a_0 = 0,\ c_6 = 0,\ a_6 = \infty,\ cc_0 = \infty,\ aa_0 = 0,\ cc_6 = 0,\ aa_6 = \infty,\ j = 3,\ f = 0,\ \bar{u} = 0$. Ferner setze $x_1 = x_2 = 1$ und $x_6 = 0$.

2. Wir bestimmen den kritischen Index $s = 5$. Wegen $s = t = 5$ setzen wir $u := \sum_{k=j}^{s} cc_k + \left\lfloor \frac{cc_{s-1}}{aa_{s-1}} \left(\hat{b} - \sum_{k=j}^{s} aa_k\right) \right\rfloor = 28 + \left\lfloor \frac{10}{4}(9 - 12) \right\rfloor = 20$. Wegen $f = 0$ setzen wir $U = \hat{z} + u = 16 + 20 = 36$. Setze $f = 1$.

3. Für $j = 3$ setze $\hat{b} = \hat{b} - aa_3 = 9 - 2 = 7,\ \hat{z} = \hat{z} + cc_3 = 16 + 6 = 22,\ \hat{x}_3 = 1,\ j = 4$

   Für $j = 4$ setze $\hat{b} = \hat{b} - aa_4 = 7 - 4 = 3,\ \hat{z} = \hat{z} + cc_4 = 22 + 10 = 32,\ \hat{x}_4 = 1,\ j = 5$

Wegen $aa_5 = 6 > 3 = \hat{b}$, prüfe nun:

Es ist $j = 5 \leq 5 = t$, daher setze $\hat{x}_5 = 0$ und

$\bar{u} = \max\left\{\bar{u}, \left\lfloor \hat{z} + c_j + \left(\hat{b} - a_j\right) \frac{c_{r-1}}{a_{r-1}} \right\rfloor\right\}$

$= \max\left\{0, \left\lfloor 32 + 12 + (3-6)\,\frac{8}{2} \right\rfloor\right\} = 32$. Setze $j = 6$. Es ist $j > t$.

Bemerkung: Man beachte, dass wegen $\bar{b} = 0$ und dadurch zusätzlich fixierter Variablen ein Wert von $\bar{u} = 28$ ermittelt werden kann.

4. Es ist $\hat{z} = 32 > 0 = z$, also setze $z = \hat{z} = 32$, und $x^T = (1, 1, 1, 1, 0, 0)$.

   Setze $j = 5$. Es ist $\hat{x}_t = \hat{x}_5 = 0$. Daher setze

   $\bar{u} = \max\left\{\bar{u}, \left\lfloor \hat{z} + \hat{b}\frac{c_{t+1}}{a_{t+1}} \right\rfloor\right\} = \max\left\{32, \left\lfloor 32 + 3 \cdot \frac{12}{10} \right\rfloor\right\} = 35$

   Bemerkung: Man beachte, dass man wegen $\bar{b} = 6$ einen Wert von $\bar{u} = 32$ finden kann.

5. Es ist $i = \max\{k < 5 : \hat{x}_k = 1\} = 4$.

   Setze $\hat{b} = \hat{b} + aa_4 = 3 + 4 = 7,\ \hat{z} = \hat{z} - cc_4 = 32 - 10 = 22,\ \hat{x}_4 = 0,\ j = 5.$

2. Es existiert kein $s = \min\left\{i : \sum_{k=j}^{i} aa_k > \hat{b}\right\}$, setze also

   $u = \sum_{k=j}^{t} cc_k = c_5 = 12$. Es ist $z = 32 < 22 + 12 = 34$

3. Es ist $aa_5 = 6 \leq 7 = \hat{b}$, also setze $\hat{b} = 7 - 6 = 1,\ \hat{z} = 22 + 12 = 34,\ \hat{x}_5 = 1,$ $j = 6$. Es ist $j = 6 > 5 = t$

4. Es ist $\hat{z} = 34 > 32 = z$, also setze $z = 34,\ x^T = (1, 1, 1, 0, 1, 0)$ und $j = 5$. Es ist $\hat{x}_5 = 1$, daher setze zunächst $\bar{u} = \max\left\{\bar{u}, \left\lfloor \hat{z} + \hat{b}\frac{c_{t+1}}{a_{t+1}} \right\rfloor\right\} =$ $\max\left\{35, \left\lfloor 34 + 1 \cdot \frac{12}{10} \right\rfloor\right\} = 35$, danach $\hat{b} = \hat{b} + aa_5 = 1 + 6 = 7,\ \hat{z} = \hat{z} - cc_5 =$ $34 - 12 = 22,\ \hat{x}_5 = 0$ und $\bar{u} = \max\left\{\bar{u}, \left\lfloor \hat{z} + \hat{b}\frac{c_{t+1}}{a_{t+1}} \right\rfloor\right\} = \max\left\{35, \left\lfloor 22 + 7 \cdot \frac{12}{10} \right\rfloor\right\} =$ 35

   Bemerkung: Die Alternativrechnung ergibt für den ersten Fall wegen $\bar{b} = 4$ den Wert $\bar{u} = 34$ und für den zweiten Fall wegen $\bar{b} = 10$ den Wert $\bar{u} = 34$.

5. Es ist $i = \max\{k < 5 : \hat{x}_k = 1\} = 3$.

   Setze $\hat{b} = \hat{b} + aa_3 = 7 + 2 = 9,\ \hat{z} = \hat{z} - cc_3 = 22 - 6 = 16,\ \hat{x}_3 = 0,\ j = 4.$

2. Es ist $s = \min\left\{i : \sum_{k=j}^{i} a_k > \hat{b}\right\} = 5$, $u := \sum_{k=j}^{s} cc_k + \left\lfloor \frac{cc_{s-1}}{aa_{s-1}} \left(\hat{b} - \sum_{k=j}^{s} aa_k\right) \right\rfloor =$ $22 + \left\lfloor (-1) \cdot \frac{10}{4} \right\rfloor = 19$. Es ist $z = 34 < 16 + 20 = 35$

3. Es ist $aa_4 = 4 \leq 9 = \hat{b}$, also setze $\hat{b} = 9 - 4 = 5,\ \hat{z} = 16 + 10 = 26,\ \hat{x}_4 = 1,$ $j = 5$.

   Es ist $j = 5 \leq 5$, also setze $\hat{x}_5 = 0,\ \bar{u} = \max\left\{\bar{u}, \left\lfloor \hat{z} + c_j + \left(\hat{b} - a_j\right) \frac{c_{r-1}}{a_{r-1}} \right\rfloor\right\} =$ $\max\left\{35, \left\lfloor 26 + 12 + (5-6)\,\frac{8}{2} \right\rfloor\right\} = 35$ und dann $j = 6$. Es ist $j > t$.

   Bemerkung: Wegen $\bar{b} = 8$ ergibt sich verschärft $\bar{u} = 34$.

4. Es ist $\hat{z} = 26 \leq z = 34$, also setze $j = 5$. Es ist $\hat{x}_5 = 0$. Setze $\bar{u} = \max\left\{\bar{u}, \left\lfloor \hat{z} + \hat{b}\frac{c_{j+1}}{a_{j+1}} \right\rfloor\right\} = \max\left\{35, \left\lfloor 26 + 5 \cdot \frac{12}{10} \right\rfloor\right\} = 35$

   Bemerkung: Wegen $\bar{b} = 8$ ergibt sich verschärft $\bar{u} = 34$

5. Es ist $i = \max\{k < 5 : \hat{x}_k = 1\} = 4$. Setze $\hat{b} = 5 + 4 = 9,\ \hat{z} = 26 - 10 = 16,$ $\hat{x}_4 = 0,\ j = 5$.

2. Es existiert kein $s = \min\left\{i : \sum_{k=j}^{i} a_k > \hat{b}\right\}$, setze also $u = \sum_{k=j}^{t} cc_k = 12$

   Es ist $z = 34 > 16 + 12 = 28$. Berechne $\bar{u} = \max\left\{35, \left\lfloor 16 + 9 \cdot \frac{12}{6} \right\rfloor\right\} = 35$

   Bemerkung: wegen $\bar{b} = 12$ ergibt sich $\bar{u} = \max\left\{34, \left\lfloor 16 + 9 \cdot \frac{12}{6} \right\rfloor\right\} = 34$

5. Es ist kein $i = \max\{k < 5 : \hat{x}_k = 1\}$ mehr bestimmbar.

6. Es ist $\bar{u} = 35 < 36 = U$, also setze $U = 35$ (verschärft $U = 34$)

   Es exitiert keine Gierige Erweiterung.

Man erhält also $z = 34$ und $U = 35$ bzw. im verschärften Fall $z = U = 34$.

Den Ablauf der Prozedur beschreibt die folgende **Skizze:**

Folgende *Empfehlungen* können ausgesprochen werden:

Eine gute Wahl ist $r = k - \alpha$, $t = k + \alpha$, wobei k den kritischen Index darstellt und $\alpha \in \mathbf{N}_0$ ein gewählter Parameter ist. Dabei kann α viel größer als bei LUU6 gewählt werden, ohne dass die Prozedur zeitmäßig aus dem Ruder läuft, z. B. $\alpha = \sqrt{n}$.

Aber auch $r = k$ und $t = n$ ist eine gute Wahl. Diese liefert als zulässige Lösung immer eine erweiterte Dantziglösung.

LU Prozeduren mit Breitensuche

LU-Verfahren entstehen auch, wenn exakte Verfahren, die in ihrem Verlauf die obere Grenze immer wieder reduzieren und immer wieder neue bessere zulässige Lösungen ermitteln, nach willkürlicher Zeit abgebrochen werden. In diesem Sinne entsteht ein LU Verfahren, wenn die im letzten Kapitel besprochene B&B Methode mit Breitensuche zugrunde gelegt wird. Dieses bietet sich insofern an, als bei dieser Methode untere und obere Schranke häufig geändert werden.

Ein kritischer Punkt dieses Verfahrens ist, dass eine Liste noch zu verzweigender Testaufgaben vorgehalten werden muss, deren Länge man schlecht vorhersagen kann und die im Laufe des Verfahrens sehr anwachsen kann. Eine sinnvolle Möglichkeit des vorzeitigen Abbruch besteht darin, die Größe dieses Speicherplatzes, d.h. die Länge der entsprechenden Liste, von vorneherein auf einen Vektor der Länge q zu begrenzen. Ist die Liste dann voll und müsste eine weitere Testaufgabe aufgenommen werden, so wird die Aufnahme verweigert. Stattdessen muss die lokale obere Schranke τ dieser Aufgabe gespeichert werden. τ stellt eine Grenze für die zu ermittelnde obere Schranke dar, die von der Heuristik nicht unterschritten werden kann. Wird später eine Testaufgabe angelaufen, deren obere Schranke kleiner gleich τ, so braucht diese nicht weiter verzweigt zu werden. Nach Ablauf des Verfahrens wird das Maximum von τ und der aktuell besten oberen Schranke als Ergebnis ausgegeben. Die Ermittlung der unteren Schranken (zulässigen Lösungen) ist davon nicht berührt. Ggf. kann man auch die Anzahl der bearbeiteten Testaufgaben willkürlich begrenzen.

Bei kleinem τ kann mit einer Sortierung der Liste gearbeitet werden. Bei größerem τ ist dies wohl zu aufwändig und es bietet sich die Verwendung von FIFO an. Arbeitet man mit FIFO, so ist eine Verwendung von Körben sinnvoll. Dies kann z.B. so erfolgen, dass die anfängliche Spanne zwischen unterer und ober Grenze, sprich die Differenz der Dantzigschranke und des Zielwertes der Dantziglösung in p (nicht notwendig gleich große) Blöcke aufgeteilt wird, in die die untersuchten Testaufgaben entlang ihrer oberen Schranken einsortiert werden. Ist dann ein Block abgearbeitet, so können die Körbe erneut in der Spanne zwischen oberer und unterer Grenze gebildet werden und neu einsortiert werden. Die Größe der Körbe kann

begrenzt werden und entsprechend dem ersten Vorschlag damit umgegangen werden. Wiederum kann auch die Gesamtzahl der zu bearbeitenden Testaufgaben begrenzt werden.

Die praktische Erfahrung zeigt, dass bei gewissen Aufgaben bereits die Absicht, einen einzigen Korb, der nur die Testaufgaben mit der Dantzigschranke enthält, zu leeren, zu langen Laufzeiten führt. Eine Minimalversion dieser LU Prozedur würde also mit nur zwei Körben arbeiten, einem, der die Testaufgaben mit der besten oberen Schranke enthält und einen, der die anderen Testaufgaben aufnimmt. Wir sprechen eine solche LU Prozedur mit dem Namen **Prozedur LUBS** (LU Breitensuche) an. (Übungsaufgabe)

LU Prozeduren mit Dynamischer Programmierung

Nach den Verfahren HSV und BSF können wir auch versuchen, das Verfahren DP1 als Hilfsverfahren bei der Konstruktion einer LU Prozedur einzusetzen. Dies bietet sich auch insofern an, als die dynamische Programmierung sich gerade bei "störrischen" Aufgaben als sehr wirkungsvoll erwiesen hat.

Ähnlich wie bei der Verwendung der Prozedur HSV zur Lösung der Aufgabe $(KP[r,t])$ und im Unterschied zur Prozedur BSF berechnet die Prozedur DP1 nicht automatisch lokale obere Schranken für die Aufgaben (KP_{01}) bei vorgenommenen Fixierungen von Variablen. Gleichzeitig ist DP1 kein Branch and Bound Verfahren und erzeugt keinen binär erzeugten Verzweigungsbaum. Es ist also eine neue Idee einzusetzen, wenn DP1 in eine LU Prozedur eingebaut werden soll.

Wird nach der zu erzeugenden LU Prozedur eine Reduzierung eingesetzt, so besteht die Hauptaufgabe der LU Prozedur darin, für eine gute untere Schranke zu sorgen, damit die Auslotung während der Reduzierung möglichst effektiv ist. Dies kann DP1 leisten: wenn es die Optimallösung von $(KP[r,t])$ berechnet, so stellt diese automatisch eine zulässige Lösung for (KP_{01}) dar. Eine neue LU Prozedur entsteht also dadurch, dass $(KP[r,t])$ durch DP1 gelöst wird und so für eine untere Schranke für (KP_{01}) sorgt. Begleitet werden kann diese untere Schranke von einer unabhängig davon berechneten oberen Schranke, etwa aus einer billigen LU-Prozedur.

Eine besonders einfache Version dieser Vorgehensweise ergibt sich, wenn man bei der Anwendung von DP1 die nicht ganz unproblematische Berechnung einer zur unteren Schranke gehörigen zulässigen Lösung unterlässt. Dann muss lediglich sichergestellt werden, dass in den folgenden Verfahrensteilen, etwa der Abschlussoptimierung eine zulässigen Lösung berechnet wird. Dies stellt man z. B. sicher, wenn man die berechnete untere Schranke z für die weitere Rechnung um 1 verringert. Dann gibt $z-1$ garantiert nicht den Zielwert einer Optimallösung wider und es muss mindestens eine zulässige Lösung noch berechnet werden.

Eine Prozedur, die diese Idee umsetzt und mit einer (scharfen) oberen Schranke kombiniert, sprechen wir als **Prozedur LUDP** an (Übungsaufgabe).

2.2.3 Verbesserte Reduzierung

Die Reduzierung der gegebenen Aufgabenstellung nach der Grundidee der im letzten Kapitel besprochenen Prozedur DDR ist umso wirkungsvoller, je schärfer die untere Schranke ist, die zur Verfügung steht. Kennt man bereits die Optimallösung, so wirkt diese natürlich am schärfsten auslotend. Wir formulieren nun eine Methode, die auf eine beliebige LU Prozedur als Unterprozedur zurückgreift.

Eine allgemeine Reduzierungsmethode

Wir orientieren uns an der Prozedur DDR, ersetzen aber die dort verarbeitete verallgemeinerte Müller Merbach Schranke durch die schärfere verallgemeinerte Duzinski/Walukiewicz Schranke und lassen beliebige LU Prozeduren als Unterprozeduren zu. Im Folgenden bezeichnen wir wieder mit (KP_i^p) die Aufgabe, die aus (KP_{01}) entsteht, wenn $x_i = p$ fixiert wird.

Methode ARM (allgemeine Reduzierungsmethode)

Input: (KP_{01}), kritischer Index k, obere Schranke U, zulässige Lösung x mit Zielwert Z, Zielwert $\bar{z}$ einer erweiterten Dantziglösung $\bar{x}$. Ferner sei $z^* = \bar{c}_k + \zeta_k (b - \bar{a}_k)$

Output: Mengen JF_1 und JF_0 von Indizes, die bei der weiteren Suche auf 1 bzw 0 fixiert werden können, eine potentiell verbesserte untere Schranke Z und eine potentiell verbesserte obere Schranke U.

1. **Start:** Setze $JF_0, JF_1 := \emptyset$. Setze $\bar{U}' = \bar{U}'' = 0$.

2. **Lokale Schranken für** (KP_i^0), $i < k$

   Für $i = 1, ..., k-1$ führe aus:

   Verwende eine LU Prozedur zur Berechnung einer zulässigen Lösung x^i mit Zielwert z^i und einer oberen Schranke U_i^0 der Aufgabe (KP_i^0).

   Berechne optional den optimalen Lagrange-Multiplikator c_i^* und setze $U_i^0 = \min\{U_i^0, z^* - c_i^*\}$.

   Gilt $U_i^0 > \bar{U}'$, so setze $\bar{U}' = U_i^0$.

   Gilt $z^i > Z$, setze $x := x^i$ und $Z := z^i$. Gilt dann $U = Z$, **stopp**, x ist optimal.

3. **Lokale Schranken für** (KP_i^1), $i > k$

   Für $i = k+1, ..., n$ führe aus:

Verwende eine LU Prozedur zur Berechnung einer zulässigen Lösung x^i mit Zielwert z^i und einer oberen Schranke U_i^1 der Aufgabe (KP_i^1).

Berechne optional den optimalen Lagrange-Multiplikator c_i^* und setze $U_i^1 = \min\{U_i^1, z^* - c_i^*\}$.

Gilt $\bar{x}_i = 0$ und $U_i^1 > \bar{U}''$, so setze $\bar{U}'' = U_i^1$.

Gilt $z^i > Z$, setze $x := x^i$ und $Z := z^i$. Gilt dann $U = Z$, **stopp**, x ist optimal.

4. **Lokale Schranken für** (KP_k^0) und (KP_k^1)

   Verwende eine LU Prozedur zur Berechnung je einer zulässigen Lösung x^k mit Zielwert z^k und einer oberen Schranke U_k^0 der Aufgabe (KP_k^0). Gilt $z^k > Z$, setze $x := x^k$ und $Z := z^k$. Gilt dann $U = Z$, **stopp**, x ist optimal.

   Verwende eine LU Prozedur zur Berechnung je einer zulässigen Lösung x^k mit Zielwert z^k und einer oberen Schranke U_k^1 der Aufgabe (KP_k^1). Gilt $z^k > Z$, setze $x := x^k$ und $Z := z^k$. Gilt dann $U = Z$, **stopp**, x ist optimal.

   Wähle je eine obere Schranke $\hat{U}_k^p$ für die Aufgaben (KP_k^p), $p = 0$ und $p = 1$.

5. **Globale Schranken**

   Setze $\bar{U}' = \min\left\{\hat{U}_k^1, \bar{U}'\right\}$ und $\bar{U}'' = \min\left\{\hat{U}_k^0, \bar{U}''\right\}$.

   Setze $U = \min\left\{U, \max\left\{\bar{z}, \bar{U}', \bar{U}''\right\}\right\}$.

   Wenn möglich, berechne zu x noch eine gierige ERweiterung

   Gilt $U = Z$, **stopp**, x ist optimal.

6. **Reduzierung**

   Für $i = 1, ...k$ führe aus: Gilt $U_i^0 \leq Z$, setze $JF_1 = JF_1 \cup \{i\}$

   Für $i = k, ...n$ führe aus: Gilt $U_i^1 \leq Z$, setze $JF_0 = JF_0 \cup \{i\}$

   Gilt danach $JF_0 \cap JF_1 \neq \emptyset$, **stopp**, x ist Optimallösung.

7. **Nachjustierung**

   Berechne die Restkapazität $b = b - \sum_{j \in JF_1} a_j$.

   Ist $b \leq 0$, **stopp**, x ist Optimallösung.

   Für alle $j \in \{1, ..., n\} \setminus (JF_1 \cup JF_0)$ führe aus:

   ist $a_j > b$, so setze $JF_0 = JF_0 \cup \{j\}$

   ist $a_j = b$, so aktualisiere ggf. die untere Schranke durch Festsetzung von $x_i = 1$ für $i \in JF_1 \cup \{j\}$, $x_i = 0$ sonst.

Die Laufzeit der Methode richtet sich natürlich insbesondere nach der Laufzeit der verwendeten LU Prozedur. Sie liegt aber bei mindestens $\mathcal{O}(n)$.

Bemerkung: Es kann durchaus sinnvoll sein, die Prozedur ARM mehrfach hintereinander anzuwenden (auf die Aufgabe mit den jeweils nicht fixierten Variablen), da sich für die U_j^p neue bessere Werte ergeben können. Dies kann wiederholt werden, bis sich keine Variablen mehr fixieren lassen. Allerdings steigt die Laufzeit dabei um den Faktor n, es sei denn, man begrenzt die Anzahl der Wiederholungen.

Gleichzeitig muss betont werden, dass die unveränderte Einbringung einer LU Prozedur als Unterprozedur nicht sinnvoll ist, man sollte die Prozeduren anpassen und überflüssige Operationen entfernen. Die folgenden Beispielprozeduren zeigen, wie.

Martello Toth Reduzierung

Die folgende erste *Beispielprozedur* für die Methode ARM wird im Wesentlichen von Martello und Toth so empfohlen (1988). Sie verwendet die Schranke U_2 zur Berechnung der lokalen oberen Schranken und die jeweilige Dantziglösung als zulässige Lösung, also eine LU Prozedur **LU2D** (LU U_2 Dantziglösung), die wir nicht explizit besprochen haben.

Wir geben hier eine gegenüber dem Original nochmals verbesserte Version im Sinne der allgemeinen Prozedur ARM an, die zusätzliche zulässige Lösungen berechnet und bei der anfänglich bereits eine obere und eine untere Schranke bekannt sind (z.B. aus der vorherigen Anwendung einer LU Prozedur). Als Version der Schranke U_{VD} verwenden wir wie die verallgemeinerte Duzinski/Walukiewicz Schranke mit $\hat{U}_k^1 = U_k^1$ und $\hat{U}_k^0 = U_k^0$, modelliert an der gierigen Lösung, die eigens berechnet wird.

Prozedur MTR (Martello Toth Reduzierung)

Input: $\left(KP_{01}^{+}\right)$, kritischer Index k, obere Schranke U, zulässige Lösung x mit Zielwert Z

Output: Mengen JF_1 und JF_0 von Indizes, die bei der weiteren Suche auf 1 bzw 0 fixiert werden können, eine potentiell verbesserte untere Schranke Z und eine potentiell verbesserte obere Schranke U.

1. **Start:** Setze $JF_0, JF_1 := \emptyset$ und $\bar{U}' = \bar{U}'' = 0$. Berechne für $i = 1, ...n$ die kumulierten Größen $\bar{c}_i = \sum_{j=1}^{i} c_j$ und $\bar{a}_i = \sum_{j=1}^{i} a_j$. Setze $i_0 = 0$.

2. **Gierige Lösung**

   Setze $\bar{z} = \bar{c}_{k-1}, \bar{b} = b - \bar{a}_{k-1}$ und $\bar{x}_i = 1$ für $i = 1, ..., k-1$ sowie $\bar{x}_k = 0$.

Für $i = k+1, ..., n$ führe aus:

ist $a_i \leq \bar{b}$, setze $\bar{x}_i = 1$ und $\bar{b} = \bar{b} - a_i,\ \bar{z} = \bar{z} + c_i$, sonst setze $\bar{x}_i = 0$.

Ist $\bar{z} > Z$, setze $x = \bar{x}$ und $Z = \bar{z}$

Gilt $U = Z$, **stopp**, (die Z zugrunde liegende Lösung x ist optimal).

3. **Lokale Schranken für** (KP_i^0), $i \leq k$

Für $i = 1, ..., k$ führe aus:

Finde einen Index $\bar{k}$ mit $\bar{a}_{\bar{k}-1} \leq b + a_i < \bar{a}_{\bar{k}}$

Setze $\bar{b}_i = b + a_i - \bar{a}_{\bar{k}-1}$, $z^i = \bar{c}_{\bar{k}-1} - c_i$ und $U_i^0 = z^i + \max\left\{\left\lfloor \frac{c_{\bar{k}+1}}{a_{\bar{k}+1}} \bar{b}_i \right\rfloor, \left\lfloor c_{\bar{k}} + \left(\bar{b}_i - a_{\bar{k}}\right) \frac{c_{\bar{k}-1}}{a_{\bar{k}-1}} \right\rfloor\right\}$

Ist $z^i > Z$, setze $Z = z^i$ und $i_0 = i,\ k_0 = \bar{k},\ \hat{b} = \bar{b}_i$.

Gilt $U = Z$, **stopp**, (die Z zugrunde liegende Lösung x ist optimal).

Gilt $U_i^0 > \bar{U}'$ und $i \neq k$, so setze $\bar{U}' = U_i^0$.

4. **Lokale Schranken für** (KP_i^1), $i \geq k$

Für $i = k, ..., n$ führe aus:

Finde den Index $\bar{k}$ mit $\bar{a}_{\bar{k}-1} \leq b - a_i < \bar{a}_{\bar{k}}$

Setze $\bar{b}_i = b - a_i - \bar{a}_{\bar{k}-1}$, $z^i = \bar{c}_{\bar{k}-1} + c_i$ und $U_i^1 = z^i + \max\left\{\left\lfloor \frac{c_{\bar{k}+1}}{a_{\bar{k}+1}} \bar{b}_i \right\rfloor, \left\lfloor c_{\bar{k}} + \left(\bar{b}_i - a_{\bar{k}}\right) \frac{c_{\bar{k}-1}}{a_{\bar{k}-1}} \right\rfloor\right\}$

Ist $z^i > Z$, setze $Z = z^i$ und $i_0 = i,\ k_0 = \bar{k},\ \hat{b} = \bar{b}_i$.

Gilt $U = Z$, **stopp**, (die Z zugrunde liegende Lösung x ist optimal).

Gilt $\bar{x}_i = 0,\ i \neq k$ und $U_i^1 > \bar{U}''$, so setze $\bar{U}'' = U_i^1$

5. **Globale Schranken**

Setze $\bar{U}' = \min\left\{U_i^1, \bar{U}'\right\}$ und $\bar{U}'' = \min\left\{U_i^0, \bar{U}''\right\}$.

Setze $U = \min\left\{U, \max\left\{\bar{z}, \bar{U}', \bar{U}''\right\}\right\}$.

Ist $i_0 > 0$, so führe aus

Ist $i_0 < k$, so setze $x_j = 1$ für $j = 1, ..., \bar{k} - 1$ und $x_j = 0$ für $j = k, ..., n$ sowie $x_{i_0} = 0$.

Ist $i_0 > k$, so setze $x_j = 1$ für $j = 1, ..., \bar{k} - 1$ und $x_j = 0$ für $j = k, ..., n$ sowie $x_{i_0} = 1$.

Ist $i_0 = k$ und $\bar{k} > k$, so setze $x_j = 1$ für $j = 1, ..., \bar{k} - 1$ und $x_j = 0$ für $j = k, ..., n$ sowie $x_{i_0} = 0$.

Ist $i_0 = k$ und $\bar{k} < k$, so setze $x_j = 1$ für $j = 1, ..., \bar{k} - 1$ und $x_j = 0$ für $j = k, ..., n$ sowie $x_{i_0} = 1$.

Für $j = k_0, ..., n$, $j \neq i_0$ führe aus

ist $a_j \leq \hat{b}$, so setze $x_j = 1$ und $\hat{b} = \hat{b} - a_j$, $Z = Z + c_j$.

Gilt $U = Z$, **stopp**, (x ist optimal).

6. **Reduzierung**

   Für $i = 1, ...k$ führe aus: Gilt $U_i^0 \leq Z$, setze $JF_1 = JF_1 \cup \{i\}$

   Für $i = k, ...n$ führe aus: Gilt $U_i^1 \leq Z$, setze $JF_0 = JF_0 \cup \{i\}$

   Gilt danach $k \in JF_0 \cap JF_1$, **stopp**, (die Z zugrunde liegende Lösung x Optimallösung.)

7. **Nachjustierung**

   Berechne die Restkapazität $b = b - \sum_{j \in JF_1} a_j$.

   Ist $b \leq 0$, **stopp**, die Z zugrunde liegende Lösung x Optimallösung.

   Für alle $j \in \{1, ..., n\} \setminus (JF_1 \cup JF_0)$ führe aus:

   ist $a_j > b$, so setze $JF_0 = JF_0 \cup \{j\}$

   ist $a_j = b$, so aktualisiere ggf. die untere Schranke durch Festsetzung von $x_i = 1$ für $i \in JF_1 \cup \{j\}$, $x_i = 0$ sonst.

Die Laufzeit der Prozedur ist $\mathcal{O}(n^2)$. Zu beachten ist allerdings, dass diese Laufzeit nur durch die jeweilige Suche nach den neuen kritischen Indizes zustande kommt. Da diese in der Regel unweit des globalen kritischen Index liegen, ist der Aufwand in der Praxis nicht so groß.

Betrachten wir erneut unser Beispiel. Hier brauchen wir bei Anwendung der Prozedur MTR nur die Verschärfungen gegenüber der Anwendung der Prozedur DDR zu betrachten. Wir halten uns dabei aber nicht sklawisch an die oben formulierte Prozedur. Insbesondere geben wir die jeweils beste zulässige Lösung unmittelbar an und berechnen sien nicht erst im Nachhinein.

Mit $b = 12$ lauten die Daten wieder

Gegenstand j	1	2	3	4	5	6
Nutzen c_j	8	8	6	10	12	12
Gewicht a_j	1	2	2	4	6	10

Wie wir wissen, gilt für den kritischen Index $k = 5$ und die Dantziglösung hat den Zielwert $Z = 32$, wohingegen die Dantzigschranke $U = 38$ lautet. Die gierige Lösung ist gleich der Dantziglösung.

(1) Wir berechnen $\bar{c} = (8, 16, 22, 32, 44, 56)$ und $\bar{a} = (1, 3, 5, 9, 15, 25)$

(2) für $x_1 = 0$ ergibt sich als kritischer Index $\bar{k} = 5$ mit $x^{1T} = (0,1,1,1,0,0)$ und $z^1 = 24$ sowie $\bar{b}_1 = 4$,

$U_1^0 = z^1 + \max\left\{\left\lfloor \frac{c_6}{a_6}\bar{b}_1 \right\rfloor, \left\lfloor c_5 + \left(\bar{b}_1 - a_5\right)\frac{c_4}{a_4}\right\rfloor\right\}$

$= 24 + \max\left\{\left\lfloor \frac{12}{10}\cdot 4\right\rfloor, \left\lfloor 12 + (4-6)\frac{10}{4}\right\rfloor\right\} = 31$

für $x_2 = 0$ ergibt sich als kritischer Index $\bar{k} = 5$ mit $x^{2T} = (1,0,1,1,0,0)$ und $z^2 = 24$ sowie $\bar{b}_2 = 5$, $U_2^0 = z^2 + \max\left\{\left\lfloor \frac{c_6}{a_6}\cdot\bar{b}_2\right\rfloor, \left\lfloor c_5 + \left(\bar{b}_2 - a_5\right)\frac{c_4}{a_4}\right\rfloor\right\} =$ $24 + \max\left\{\left\lfloor \frac{12}{10}\cdot 5\right\rfloor, \left\lfloor 12 + (5-6)\frac{10}{4}\right\rfloor\right\} = 33$

für $x_3 = 0$ ergibt sich als kritischer Index $\bar{k} = 5$ mit $x^{3T} = (1,1,0,1,0,0)$ und $z^3 = 26$ sowie $\bar{b}_3 = 5$, $U_3^0 = z^3 + \max\left\{\left\lfloor \frac{c_6}{a_6}\cdot\bar{b}_3\right\rfloor, \left\lfloor c_5 + \left(\bar{b}_3 - a_5\right)\frac{c_4}{a_4}\right\rfloor\right\} =$ $26 + \max\left\{\left\lfloor \frac{12}{10}\cdot 5\right\rfloor, \left\lfloor 12 + (5-6)\frac{10}{4}\right\rfloor\right\} = 35$

für $x_4 = 0$ ergibt sich als kritischer Index $\bar{k} = 6$ mit $x^{4T} = (1,1,1,0,1,0)$ und $z^4 = 34$ sowie $\bar{b}_4 = 1$, $U_4^0 = z^4 + \max\left\{\left\lfloor \frac{c_7}{a_7}\cdot\bar{b}_4\right\rfloor, \left\lfloor c_6 + \left(\bar{b}_4 - a_6\right)\frac{c_5}{a_5}\right\rfloor\right\} =$ $34 + \max\left\{\left\lfloor \frac{0}{1}\cdot 1\right\rfloor, \left\lfloor 12 + (1-10)\cdot\frac{12}{6}\right\rfloor\right\} = 34$

setze also neu $x^T = (1,1,1,0,1,0)$, $Z = 34$

für $x_5 = 0$ ergibt sich als kritischer Index $\bar{k} = 6$ mit $x^{5T} = (1,1,1,1,0,0)$ und $z^5 = 32$ sowie $\bar{b}_5 = 3$, $U_5^0 = z^5 + \max\left\{\left\lfloor \frac{c_7}{a_7}\cdot\bar{b}_5\right\rfloor, \left\lfloor c_6 + \left(\bar{b}_5 - a_6\right)\frac{c_5}{a_5}\right\rfloor\right\} =$ $32 + \max\left\{\left\lfloor \frac{0}{1}\cdot 3\right\rfloor, \left\lfloor 12 + (3-10)\frac{12}{6}\right\rfloor\right\} = 32$

(3) für $x_5 = 1$ ergibt sich als kritischer Index $\bar{k} = 4$ mit $x^{5T} = (1,1,1,0,1,0)$ und $z^5 = 34$ sowie $\bar{b}_5 = 1$, $U_5^1 = z^5 + \max\left\{\left\lfloor \frac{c_5}{a_5}\cdot\bar{b}_5\right\rfloor, \left\lfloor c_4 + \left(\bar{b}_5 - a_4\right)\frac{c_3}{a_3}\right\rfloor\right\} =$ $34 + \max\left\{\left\lfloor \frac{12}{6}\cdot 1\right\rfloor, \left\lfloor 10 + (1-4)\frac{6}{2}\right\rfloor\right\} = 36$

für $x_6 = 1$ ergibt sich als kritischer Index $\bar{k} = 2$ mit $x^{6T} = (1,0,0,0,0,1)$ und $z^6 = 20$ sowie $\bar{b}_6 = 1$, $U_6^1 = z^6 + \max\left\{\left\lfloor \frac{c_3}{a_3}\cdot\bar{b}_6\right\rfloor, \left\lfloor c_2 + \left(\bar{b}_6 - a_2\right)\frac{c_1}{a_1}\right\rfloor\right\} =$ $20 + \max\left\{\left\lfloor \frac{6}{2}\cdot 1\right\rfloor, \left\lfloor 8 + (1-2)\frac{8}{1}\right\rfloor\right\} = 23$

(4) $U = \min\left\{U, \max\left\{\bar{z}, \bar{U}', \bar{U}''\right\}\right\}$

$= \min\left\{38, \max\left\{32, \min\left\{35,36\right\}, \min\left\{23,32\right\}\right\}\right\} = 35.$

Eine gierige Erweiterung der bislang besten Lösung existiert nicht.

wegen $Z = 34 < 35 = U$ folgt $JF_1 = \{1,2,4,5\}$ und $JF_0 = \{6\}$.

(5) Es ergibt sich $b - a_1 - a_2 - a_4 - a_5 = 12 - 13 = -1$, also ist $Z = 34$ optimaler Zielwert und $x^T = (1,1,1,0,1,0)$ optimale Lösung.

Schnelles Reduzieren

Die Laufzeit der Prozedur MTR liegt mit $\mathcal{O}(n^2)$ ziemlich hoch. Man kann sie verbessern, wenn man den Aufwand für jedes i verringert. Dieser liegt zunächst bei n^2, weil ein neuer kritischer Index berechnet werden muss und ggf. eine neue zulässige Lösung. Der Aufwand für den kritischen Index kann auf $\mathcal{O}(\log n)$ reduziert werden, wenn die Suche in Binärbäumen stattfindet. Der Gesamtaufwand der Prozedur beträgt dann $\mathcal{O}(n \log n)$.

In Konkurrenz zur Prozedur MTR betrachten wir noch die folgende Prozedur, deren Zielsetzung die Reduzierung mit besonders geringer Laufzeit ist. Will man die Laufzeit $\mathcal{O}(n)$ erreichen, so dürfen für jedes i nur Operationen im Umfang von $\mathcal{O}(1)$ durchgeführt werden. Damit verbietet sich die generelle Neuberechnung von kritischen Indizes.

Eine Möglichkeit, die *oberen Schranken* mit einem Gesamtaufwand von $\mathcal{O}(n)$ zu berechnen, haben wir bereits im letzten Kapitel kennengelernt, wo wir mit dem Abschlusstableau der transformierten LP-Relaxation argumentiert haben. Wie wir inzwischen wissen, wurden dabei einfach die optimalen Lagrange Multiplikatoren berechnet und verwendet. Leider beinhaltete diese Vorgehensweise keine potentielle Verbesserung der unteren Schranke.

Auf der anderen Seite haben wir mit der Minimal-Erweiterung eine besonders billige Möglichkeit der Verbesserung der unteren Schranke kennengelernt. Die folgende Prozedur verknüpft beide Ideen zu einem Reduzierungsverfahren der Laufzeit $\mathcal{O}(n)$. Darin werden neue zulässige Lösungen x^i über ihre *Abweichungen zur Dantziglösung* festgehalten, notiert durch eine Menge E der abweichenden Indizes. Die beste zulässige Lösung kann nach Anwendung des Verfahrens aus den Informationen in E berechnet werden.

Prozedur LMR (Lagrange Minimal Reduzierung)

Input: $\left(KP_{01}^{+}\right)$, kritischer Index k, obere Schranke U, zulässige Lösung x mit Zielwert Z, Zielwert $\bar{z}$ einer erweiterten Dantziglösung $\bar{x}$. Ferner sei $z^* = \bar{c}_k + \zeta_k (b - \bar{a}_k)$

Output: Mengen JF_1 und JF_0 von Indizes, die bei der weiteren Suche auf 1 bzw 0 fixiert werden können, eine potentiell verbesserte untere Schranke Z und eine potentiell verbesserte obere Schranke U.

1. **Start:** Setze $JF_0, JF_1 := \emptyset$. Setze $\bar{U}' = \bar{U}'' = 0$. Ferner setze $\bar{c} = \sum_{j=1}^{k-1} c_j$ und $\bar{a} = \sum_{j=1}^{k-1} a_j$.

   Berechne die optimalen Lagrange Multiplikatoren c_j^* für $j = 1, ..., n$ und berechne $m_j := \arg\min \{a_i : i > j\}$ für $j = k-1, ..., n$.

2. **Lokale Schranken für** (KP_i^0), $i < k$

   Für $i = 1, ..., k-1$ führe aus:

   Berechne $\hat{b} = b - \bar{a} + a_i$. % `neuer kritischer Index` $\geq k$

   1. Fall: $\hat{b} - a_k < 0$. % k `ist kritischer Index`

      Ist $a_{m_{k-1}} \leq \hat{b}$, so setze $z^i = \bar{c} - c_i + c_{m_{k-1}}$, $E^i = \{i, m_{k-1}\}$ und

      $U_i^0 = \bar{c} - c_i + \left\lfloor \zeta_k \hat{b} \right\rfloor$

      sonst setze $z^i = \bar{c} - c_i$, $E^i = \{i\}$ und

      $U_i^0 = \bar{c} - c_i + \max\left\{ \left\lfloor a_{m_{k-1}} \frac{c_k}{a_k} + \left(\hat{b} - a_{m_{k-1}}\right) \frac{c_{k-1}}{a_{k-1}} \right\rfloor, 0 \right\}$

   2. Fall: $\hat{b} - a_k \geq 0$. Setze $\hat{b} = \hat{b} - a_k$. % `kritischer Index` $\geq k+1$

      Ist $a_{m_k} \leq \hat{b}$, so setze $z^i = \bar{c} - c_i + c_k + c_{m_k}$, $E^i = \{i, k, m_k\}$ und

      $U_i^0 = \bar{c} - c_i + c_k + \left\lfloor \zeta_{k+1} \hat{b} \right\rfloor$

      sonst setze $z^i = \bar{c} - c_i + c_k$, $E^i = \{i, k\}$ und % $k+1$ `kritisch`

      $U_i^0 = \bar{c} - c_i + c_k + \max\left\{ \left\lfloor a_{m_k} \frac{c_{k+1}}{a_{k+1}} + \left(\hat{b} - a_{m_k}\right) \frac{c_k}{a_k} \right\rfloor, 0 \right\}$

   Setze $U_i^0 = \min\left\{U_i^0, z^* - c_i^*\right\}$.

   Gilt $U_i^0 > \bar{U}'$, so setze $\bar{U}' = U_i^0$.

   Gilt $z^i > Z$, setze $E := E^i$ und $Z := z^i$. Gilt dann $U = Z$, **stopp**, das zu E gehörige x ist optimal.

3. **Lokale Schranken für** (KP_i^1), $i > k$

   Für $i = k+1, ..., n$ führe aus:

   Berechne $\hat{b} = b - \bar{a} - a_i$. % `neuer kritischer Index` $\leq k$

   1. Fall: $\hat{b} \geq 0$. % k `ist kritischer Index`

      a) $i \neq m_{k-1}$

      ist $a_{m_{k-1}} \leq \hat{b}$, so setze $z^i = \bar{c} + c_i + c_{m_{k-1}}$, $E^i = \{i, m_{k-1}\}$ und

      $U_i^1 = \bar{c} + c_i + \left\lfloor \zeta_k \hat{b} \right\rfloor$

      sonst setze $z^i = \bar{c} + c_i$, $E^i = \{i\}$ und

      $U_i^1 = \bar{c} + c_i + \max\left\{ \left\lfloor a_{m_{k-1}} \frac{c_k}{a_k} + \left(\hat{b} - a_{m_{k-1}}\right) \frac{c_{k-1}}{a_{k-1}} \right\rfloor, 0 \right\}$

      b) $i = m_{k-1}$. Dann setze $z^i = \bar{c} + c_i$, $E = \{i\}$ und $U_i^1 = \bar{c} + c_i + \left\lfloor \zeta_k \hat{b} \right\rfloor$

2. Fall: $\hat{b} < 0$. % `kritischer Index liegt` $\leq k-1$

setze $U_i^1 = \bar{c} + c_i + \left\lfloor \zeta_{k-1}\hat{b} \right\rfloor$

Setze $U_i^1 = \min\{U_i^1, z^* - c_i^*\}$.

Gilt $\bar{x}_i = 0$ und $U_i^1 > \bar{U}''$, so setze $\bar{U}'' = U_i^1$.

Gilt $z^i > Z$, setze $E = E^i$ und $Z := z^i$. Gilt dann $U = Z$, **stopp**,

das zu E gehörige x ist optimal.

4. **Lokale Schranke für** (KP_k^0)

Berechne $\hat{b} = b - \bar{a}$.

1. Fall: $\hat{b} - a_{k+1} < 0$. % $k+1$ `ist kritischer Index`

Ist $a_{m_k} \leq \hat{b}$, so setze $z^k = \bar{c} + c_{m_k}$, $E^i = \{m_k\}$ und $U_k^0 = \bar{c} + \left\lfloor \zeta_{k+1}\hat{b} \right\rfloor$

sonst setze $z^k = \bar{c}$, $E^k = \{\}$ und $U_k^0 = \bar{c} + \max\left\{ \left\lfloor a_{m_k}\frac{c_{k+1}}{a_{k+1}} + \left(\hat{b} - a_{m_k}\right)\frac{c_k}{a_k} \right\rfloor, 0 \right\}$

2. Fall: $\hat{b} - a_{k+1} \geq 0$. Setze $\hat{b} = \hat{b} - a_{k+1}$. % `kritischer Index` $\geq k+2$

Ist $a_{m_{k+1}} \leq \hat{b}$, so setze $z^k = \bar{c} + c_{k+1} + c_{m_{k+1}}$, $E^k = \{k+1, m_{k+1}\}$

und $U_k^0 = \bar{c} + c_{k+1} + \left\lfloor \zeta_{k+2}\hat{b} \right\rfloor$

sonst setze $z^k = \bar{c} + c_{k+1}$, $E^k = \{k+1\}$ und % $k+2$ `kritisch`

$U_k^0 = \bar{c} + c_{k+1} + \max\left\{ \left\lfloor a_{m_{k+1}}\frac{c_{k+2}}{a_{k+2}} + \left(\hat{b} - a_{m_{k+1}}\right)\frac{c_{k+1}}{a_{k+1}} \right\rfloor, 0 \right\}$

Gilt $z^k > Z$, setze $E = E^k$ und $Z := z^k$.

Gilt dann $U = Z$, **stopp**,

das zu E gehörige x ist optimal.

5. **Lokale Schranke für** (KP_k^1)

Berechne $\hat{b} = b - \bar{a} - a_k + a_{k-1}$.

1. Fall: $\hat{b} \geq 0$. % `kritischer Index ist` $k-1$

ist $a_{m_k} \leq \hat{b}$, so setze $z^i = \bar{c} + c_k - c_{k-1} + c_{m_k}$, $E^i = \{k-1, k, m_k\}$

und $U_k^1 = \bar{c} - c_{k-1} + c_k + \left\lfloor \zeta_{k-1}\hat{b} \right\rfloor$

sonst setze $z^k = \bar{c} + c_k - c_{k-1}$, $E^k = \{k-1, k\}$.

Berechne $\alpha = \min\{a_{k-1}, a_{m_k}\}$ und setze

$U_k^1 = \bar{c} + c_k - c_{k-1} + \max\left\{ \left\lfloor \alpha\frac{c_{k-1}}{a_{k-1}} + \left(\hat{b} - \alpha\right)\frac{c_{k-2}}{a_{k-2}} \right\rfloor, 0 \right\}$

2. Fall: $\hat{b} < 0$. % `kritischer Index liegt` $\leq k-2$

setze $U_k^1 = \bar{c} - c_{k-1} + c_k + \left\lfloor \zeta_{k-2} \hat{b} \right\rfloor$

Gilt $z^k > Z$, setze $E = E^k$ und $Z := z^k$.

Gilt dann $U = Z$, **stopp**,

das zu E gehörige x ist optimal.

6. **Reduzierung**

Setze $\bar{U}' = \min\left\{U_k^1, \bar{U}'\right\}$ und $\bar{U}'' = \min\left\{U_k^0, \bar{U}''\right\}$

Setze $U = \min\left\{U, \max\left\{\bar{z}, \bar{U}', \bar{U}''\right\}\right\}$. Gilt dann $U = Z$, **stopp**, x ist optimal.

Für $i = 1, ...k$ führe aus: Gilt $U_0^i \leq Z$, setze $JF_1 = JF_1 \cup \{i\}$

Für $i = k, ...n$ führe aus: Gilt $U_1^i \leq Z$, setze $JF_0 = JF_0 \cup \{i\}$

Gilt danach $JF_0 \cap JF_1 \neq \emptyset$, so ist das zu E gehörige x Optimallösung.

7. **Nachjustierung**

Berechne die Restkapazität $b = b - \sum_{j \in JF_1} a_j$.

Ist $b \leq 0$, **stopp**, die Z zugrunde liegende Lösung x Optimallösung.

Für alle $j \in \{1, ..., n\} \setminus (JF_1 \cup JF_0)$ führe aus:

ist $a_j > b$, so setze $JF_0 = JF_0 \cup \{j\}$

ist $a_j = b$, so aktualisiere ggf. die untere Schranke durch Festsetzung von $x_i = 1$ für $i \in JF_1 \cup \{j\}$, $x_i = 0$ sonst.

Bemerkung: Speichert man in E geringügig mehr Information, so kann, wie in MTR, vor der Reduzierung noch der Versuch gemacht werden, die bisher beste zulässige Lösung gierig zu erweitern.

Beispiel Überprüfen wir die Prozedur an unserem Beispiel. Mit $b = 12$ lauten die Daten wieder

Gegenstand j	1	2	3	4	5	6
Nutzen c_j	8	8	6	10	12	12
Gewicht a_j	1	2	2	4	6	10

Wie wir wissen, gilt für den kritischen Index $k = 5$, die Dantziglösung hat den Zielwert $z = 32$, wohingegen die Dantzigschranke $U = 38$ lautet. Wir setzen $\bar{x}$ gleich der Dantziglösung mit Zielwert $\bar{z}$.

Start: Setze $JF_0, JF_1 := \emptyset$. Setze $\bar{U}' = \bar{U}'' = 0$. Ferner setze $\bar{c} = \sum_{j=1}^{k-1} c_j = 32$ und $\bar{a} = \sum_{j=1}^{k-1} a_j = 9$

Berechne die optimalen Lagrange Multiplikatoren c_j^* für $j = 1, ..., n$ zu $c^* = (6, 4, 2, 2, 0, 8)$ und $(m_4, m_5, m_6) = (5, 6, 7)$

Lokale Schranken für (KP_i^0), $i < k$

$i = 1$: es ist $\hat{b} = b - \bar{a} + a_1 = 12 - 9 + 1 = 4$ und $\hat{b} - a_5 = 4 - 6 = -2 < 0$. Es ist $k = 5$ kritischer Index und $a_{m_4} = 6 > 4 = \hat{b}$. Setze also $z^1 = \bar{c} - c_1 = 32 - 8 = 24$ und $E^1 = \{1\}$ sowie $U_1^0 = \bar{c} - c_1 + \left\lfloor a_{m_{k-1}} \frac{c_k}{a_k} + \left(\hat{b} - a_{m_{k-1}}\right) \frac{c_{k-1}}{a_{k-1}} \right\rfloor = 24 + \left\lfloor 6 \cdot \frac{12}{6} + (4-6) \frac{10}{4} \right\rfloor = 31$. Es ist $31 = U_1^0 \leq z^* - c_1^* = 38 - 6 = 32$. Setze $\bar{U}' = 31$.

$i = 2$: es ist $\hat{b} = b - \bar{a} + a_2 = 12 - 9 + 2 = 5$ und $\hat{b} - a_5 = 5 - 6 = -1 < 0$. Es ist $k = 5$ kritischer Index und $a_{m_4} = 6 > 5 = \hat{b}$. Setze also $z^2 = \bar{c} - c_2 = 32 - 8 = 24$ und $E^2 = \{2\}$ sowie $U_2^0 = \bar{c} - c_2 + \left\lfloor a_m \frac{c_k}{a_k} + \left(\hat{b} - a_m\right) \frac{c_{k-1}}{a_{k-1}} \right\rfloor = 24 + \left\lfloor 6 \cdot \frac{12}{6} + (5-6) \frac{10}{4} \right\rfloor = 33$. Es ist $33 = U_2^0 \leq z^* - c_2^* = 38 - 4 = 34$. Setze $\bar{U}' = 33$.

$i = 3$: es ist $\hat{b} = b - \bar{a} + a_3 = 12 - 9 + 2 = 5$ und $\hat{b} - a_5 = 5 - 6 = -1 < 0$. Es ist $k = 5$ kritischer Index und $a_{m_4} = 6 > 5 = \hat{b}$. Setze also $z^3 = \bar{c} - c_3 = 32 - 6 = 26$ und $E^3 = \{3\}$ sowie $U_3^0 = \bar{c} - c_3 + \left\lfloor a_{m_{k-1}} \frac{c_k}{a_k} + \left(\hat{b} - a_{m_{k-1}}\right) \frac{c_{k-1}}{a_{k-1}} \right\rfloor = 26 + \left\lfloor 6 \cdot \frac{12}{6} + (5-6) \frac{10}{4} \right\rfloor = 35$. Es ist $35 = U_3^0 \leq z^* - c_3^* = 38 - 2 = 36$. Setze $\bar{U}' = 35$.

$i = 4$: es ist $\hat{b} = b - \bar{a} + a_4 = 12 - 9 + 4 = 7$ und $\hat{b} - a_5 = 7 - 6 = 1 > 0$. Setze $\hat{b} = 1$. Es ist $k + 1 = 6$ kritischer Index und $a_{m_k} = 10 > 1 = \hat{b}$. Setze also $z^4 = \bar{c} - c_4 + c_5 = 32 - 10 + 12 = 34$ und $E^4 = \{4, 5\}$ sowie $U_4^0 = \bar{c} - c_i + c_k + \left\lfloor a_{m_k} \frac{c_{k+1}}{a_{k+1}} + \left(\hat{b} - a_{m_k}\right) \frac{c_k}{a_k} \right\rfloor = 34 + \max\left\{ \left\lfloor 10 \cdot \frac{12}{10} + (1-10) \frac{12}{6} \right\rfloor, 0 \right\} = 34$. Es ist $34 = U_4^0 \leq z^* - c_3^* = 38 - 2 = 36$. Es bleibt bei $\bar{U}' = 35$. Setze $Z = 34$ und $E = \{4, 5\}$. Es ist nicht $U = Z$

Lokale Schranken für (KP_i^1), $i > k$

$i = 6$: es ist $\hat{b} = b - \bar{a} - a_6 = 12 - 9 - 10 = -7 < 0$. Der kritische Index liegt also ≤ 4 und es gilt $U_6^1 = \bar{c} - c_{k-1} + c_i + \left\lfloor \zeta_{k-1} \left(\hat{b} + a_{k-1}\right) \right\rfloor = 32 + 12 + \left\lfloor \frac{10}{4} (-7) \right\rfloor = 26$. Wegen $x_6'' = 0$ und $U_6^1 > 0 = \bar{U}''$ setze $\bar{U}'' = 26$

Lokale Schranke für (KP_k^0)

Es ist $\hat{b} = b - \bar{a} = 12 - 9 = 3$ und $\hat{b} - a_{k+1} = 3 - 10 = -7 < 0$. Der kritische Index ist also 6. Es ist $a_{m_k} = 10 > 3 = \hat{b}$. Setze $z^k = \bar{c} = 32$, $E^i = \{\}$ und $U_5^0 = \bar{c} + \max\left\{ \left\lfloor a_{m_k} \frac{c_{k+1}}{a_{k+1}} + \left(\hat{b} - a_{m_k}\right) \frac{c_k}{a_k} \right\rfloor, 0 \right\} = 32 + \max\left\{ \left\lfloor 10 \cdot \frac{12}{10} + (3-10) \frac{12}{6} \right\rfloor, 0 \right\} = 32$. Es ist nicht $z^k > Z$.

Lokale Schranke für (KP_k^1)

Es ist $\hat{b} = b - \bar{a} - a_k + a_{k-1} = 12 - 9 - 6 + 4 = 1 > 0$. Kritischer Index ist also 4. Es ist $a_{m_k} = 10 > 1 = \hat{b}$. Also setze $z^k = \bar{c} + c_k - c_{k-1} = 32 + 12 - 10 = 34$,

$E^k = \{4, 5\}$. Bilde $\alpha = \min\{a_{k-1}, a_{m_k}\} = \min\{4, 10\} = 4$ und setze $U_5^1 = \bar{c} + c_k - c_{k-1} + \max\left\{\left\lfloor \alpha \frac{c_{k-1}}{a_{k-1}} + \left(\hat{b} - \alpha\right) \frac{c_{k-2}}{a_{k-2}} \right\rfloor, 0\right\} = 34 + \max\left\{\left\lfloor 4 \cdot \frac{10}{4} + (1-4)\frac{6}{2} \right\rfloor, 0\right\} = 35$.

Reduzierung

Setze $\bar{U}' = \min\left\{U_k^1, \bar{U}'\right\} = \min\{35, 35\} = 35$ und $\bar{U}'' = \min\left\{U_k^0, \bar{U}''\right\} = \min\{32, 26\} = 26$

Setze $U = \min\left\{U, \max\left\{\bar{z}, \bar{U}', \bar{U}''\right\}\right\} = \min\{38, \max\{32, 35, 31\}\} = 35$. Wegen $Z = 34 < 35 = U$ folgt $JF_1 = \{1, 2, 4, 5\}$ und $JF_0 = \{6\}$.

Nachjustierung

Es ergibt sich $b - a_1 - a_2 - a_4 - a_5 = 12 - 13 = -1$, also ist $Z = 34$ optimaler Zielwert und $x^T = (1, 1, 1, 0, 1, 0)$ optimale Lösung. ■

Bemerkung: Die genannte Prozedur bezieht bei der Schrankenberechnung das Indexintervall $[k-1, k+1]$ mit ein. Dies kann wie folgt erweitert werden auf die Miteinbeziehung eines *beschränkten* Bereichs $[r, t] := [k - \alpha, k + \alpha]$ mit $\alpha \geq 1$, ohne dass die Laufzeit von $\mathcal{O}(n)$ erhöht wird: Suche immer zuerst das kritische Element innerhalb von $[r, t]$. Liegt es dort, so verwende eine Schrankenberechnung wie in MTR. Andernfalls verwende die Schrankenberechnung wie in LMR, mit Verschärfung wegen der Information, dass der kritische Index außerhalb von $[r, t]$ liegt.

Dies führt zu einer potentiell schärferen Reduzierung, obwohl die Laufzeit insgesamt immer noch bei $\mathcal{O}(n)$ bleibt, wenn α fest gewählt oder α von der Größenordnung $\mathcal{O}(\sqrt{n})$ ist. (Übungsaufgabe)

2.3 Verbesserung des Verfahrens HSV

Das Branch and Bound Verfahren von Horowitz und Sahni kann verbessert werden, wenn eine bessere Auslotung verwendet wird, die einerseits schärfere Schranken benutzt und andererseits mehr Auslotungsfälle betrachtet. Martello und Toth schlagen im Wesentlichen das folgende Verfahren vor, das diese Überlegungen berücksichtigt. Sie wenden die B&B Methode mit gleicher Strategie an wie Horowitz und Sahni (LIFO mit Verzweigung nach Dakin, die Verzweigungsvariable immer zuerst auf 1, dann auf 0 setzend), benutzen aber anstelle der Dantzigschranke die potentiell schärfere Schranke U_2. Gleichzeitig betrachten sie besondere Auslotungsfälle, die Informationen aus dem bisherigen Verlauf des Verfahrens ziehen und vermeiden unnötige Speicherungen, Suchläufe, Summierungen...

2.3.1 Formulierung des Verfahrens

Wir orientieren uns in der Darstellung an der der Prozedur HSV. Der Aufbau der folgenden Prozedur ist prinzipiell gleich, nur wird der Vorwärtsschritt im zwei Teile geteilt (Vorwärtssschritt1 und Sichern) und es wird gegenüber HSV ein Vorwärtsschritt2 hinzugefügt. Wir verwenden eine gegenüber dem Original vereinfachte und verbessserte Version, die die Verwandtschaft zur Prozedur HSV besser erkennen lässt, gleichzeitig verbessern wir die Möglichkeit, die (globale) obere Schranke zu verschärfen. Stoppt das Verfahren, so gibt die gespeicherte Lösung x mit Zielwert z eine Optimallösung für (KP_{01}) an.

Folgende Notationen werden verwendet:

$\hat{x}$ = laufende Lösung

$\hat{z}$ = Vornutzen der laufenden Lösung

$\hat{b}$ = laufende Restkapazität

x = beste bisher bekannte Lösung

z = Zielwert der besten bislang bekannten Lösung

r = laufender kritischer Index

r_k = Hilfsgröße bei der Berechnung des kritischen Index

j = erster Index der laufenden Aufgabe

c' = Nutzen der Dantziglösung der laufenden Aufgabe

a' = Gewicht der Dantziglösung der laufenden Aufgabe

Prozedur MT1 (Martello Toth 1)

Input: $\left(KP_{01}^{+}\right)$, eine zulässige Lösung x mit Zielfunktionswert z und eine obere Schranke U.

Output: Optimallösung x mit Zielfunktionswert z.

1. **Start** Setze $\hat{z} = 0,\ \hat{b} = b,\ j := 1,\ f = 0,\ l = 0$. Setze $r_k = k$ für $k = 1, ..., n$
   Setze $\hat{x}_k = 0$ und berechne $w_k := \min\{a_j : j > k\}$ für alle $k = 1, ..., n$.
   Berechne rekursiv die kumulierten Größen $\bar{a}_i = \sum_{j=1}^{i} a_j$ und $\bar{c}_i = \sum_{j=1}^{i} c_j$ für $i = 0, ..., n$

2. **Vorwärtsschritt 1**

   solange $a_j > \hat{b}$ gilt, führe aus:

   ist $z \geq \hat{z} + \left\lfloor \hat{b}\frac{c_{j+1}}{a_{j+1}} \right\rfloor$, so gehe zu 5., andernfalls setze $j = j + 1$

   suche $r = \min\left\{ i : \bar{a}_i > \hat{b} + \bar{a}_{j-1} \right\}$, startend bei $i = r_j$.

   setze $c' = \bar{c}_{r-1} - \bar{c}_{j-1}$ und $a' = \bar{a}_{r-1} - \bar{a}_{j-1}$

   ist $r \leq n$, so berechne $u := \max\left\{ \left\lfloor \left(\hat{b} - a'\right) \frac{c_{r+1}}{a_{r+1}} \right\rfloor, \left\lfloor c_r - \left(a_r - \left(\hat{b} - a'\right) \frac{c_{r-1}}{a_{r-1}}\right) \right\rfloor \right\}$

   andernfalls setze $u = 0$

   ist $z \geq \hat{z} + c' + u$, gehe zu 5.

   ist $f = 0$ und $U > \hat{z} + c' + u$, setze $U = \hat{z} + c' + u$ sowie $f = 1$ und $l = l + 1$

   ist $u = 0$, gehe zu 4.

3. **Sicherung**

   setze $\hat{b} = \hat{b} - a'$ und $\hat{z} = \hat{z} + c'$

   für $k = j$ bis $r - 1$ setze $\hat{x}_k = 1$ und $r_k = r$.

   setze $j = r$

   ist $\hat{b} \geq w_r$, so gehe zu 2.

   ist $z \geq \hat{z}$, so gehe zu 5.

   setze $c' = 0$

4. **Aktualisierung**

   setze $z = \hat{z} + c'$

   für $k = 1$ bis $j - 1$ setze $x_k = \hat{x}_k$

   für $k = j$ bis $r - 1$ setze $x_k = 1$

   für $k = r$ bis n setze $x_k = 0$

   ist $z = U$, **stopp**

5. **Rückwärtsschritt**

   finde $i = \max\{k < j : \hat{x}_k = 1\}$. Gibt es kein solches i, **stopp**

   setze $\hat{b} := \hat{b} + a_i, \quad \hat{z} := \hat{z} - c_i, \quad \hat{x}_i := 0, \quad j := i + 1$

   ist $i = l$, setze $f = 0$

   ist $\hat{b} - a_i \geq w_i$, gehe zu 2.

   setze $j = i$, $h = i$

6. **Vorwärtsschritt 2**

   setze $h = h + 1$

   ist $z \geq \hat{z} + \left\lfloor \hat{b}\frac{c_h}{a_h} \right\rfloor$, so gehe zu 5.

   ist $a_h = a_i$, so gehe zu 6.

   ist $a_h > a_i$, so führe aus:

      ist $a_h > \hat{b}$ oder $z \geq \hat{z} + c_h$, so gehe zu 6.

      setze $z = \hat{z} + c_h$

      für $k = 1$ bis n setze $x_k = \hat{x}_k$

      setze $x_h = 1$

      ist $z = U$, **stopp**

      setze $i = h$ und gehe zu 6.

   andernfalls führe aus:

      ist $\hat{b} - a_h < w_h$, so gehe zu 6.

      setze $\hat{b} = \hat{b} - a_h,\ \hat{z} = \hat{z} + c_h,\ \hat{x}_h = 1$ und $j = h + 1$

      gehe zu 2.

Zum besseren Verständnis wollen wir die Prozedur auf unser Beispiel anwenden, um damit auch eine Vergleichsmöglichkeit zur Anwendung der Prozedur HSV zu haben.

Gegeben ist also das 0-1−Knapsack Problem mit den Daten

Gegenstand j	1	2	3	4	5	6
Nutzen c_j	8	8	6	10	12	12
Gewicht a_j	1	2	2	4	6	10

mit $b = 12$

Wir lösen die Aufgabe ohne Ausnutzung einer zuvor ermittelten unteren oder oberen Schranke. Wir setzen also anfänglich $x = 0,\ z = 0$ und $U = \infty$.

1. Zunächst setzen wir wie vorgeschrieben $z = 0,\ \hat{z} = 0,\ \hat{b} = b = 12,\ c_7 := 0,\ a_7 := \infty,\ j := 1,\ U = \infty,\ f = 0,\ l = 0,\ \hat{x}_k = 0$ für $k = 1, ..., 6$, und $w^T = (2, 2, 4, 6, 10, \infty)$ sowie $\bar{a}^T = (1, 3, 5, 9, 15, 25)$, $\bar{c}^T = (8, 16, 22, 32, 44, 56)$, zusätzlich $\bar{a}_0 = 0$, $\bar{c}_0 = 0$.

2. Es ist $a_1 \leq \hat{b}$. Suche also den kritischen Index: $r = \min\left\{i : \bar{a}_i > \hat{b} + \bar{a}_{j-1}\right\} = 5$ sowie $c' = \bar{c}_{r-1} - \bar{c}_{j-1} = \bar{c}_4 = 32$ und $a' = \bar{a}_{r-1} - \bar{a}_{j-1} = \bar{a}_4 = 9$. Es $r \leq n$, also berechne

$$u = \max\left\{\left\lfloor (12 - 9)\,\frac{12}{10} \right\rfloor, \left\lfloor 12 - (6 - (12 - 9))\,\frac{10}{4} \right\rfloor\right\} = \max\{3, 4\} = 4$$

Es ist $z \leq 0 + 32 + 4$. Wegen $f = 0$ setze $U = \hat{z} + c' + u = 36$ und $l = 1$.

Es ist $u > 0$.

3. Setze $\hat{b} = 12 - 9 = 3$, $\hat{z} = 0 + 32 = 32$. Ferner setze $\hat{x}_1 = \hat{x}_2 = \hat{x}_3 = \hat{x}_4 = 1$, $j = 6$.

   Es ist $3 = \hat{b} < w_5 = 10$ und $z = 0 < 32 = \hat{z}$. Also setze $c' = 0$.

4. Setze $z = \hat{z} + c' = 32$ und $x_1 = x_2 = x_3 = x_4 = 1$, $x_5 = 0$ und $x_6 = 0$. Es ist $z < U$

5. Man findet $i = 4$. Setze $\hat{b} = 3 + 4 = 7$, $\hat{z} = 32 - 10 = 22$, $\hat{x}_4 = 0$, $j = 5$.

   Es ist $\hat{b} - a_4 = 7 - 4 < 6 = w_4$, also setze $j = 4$, $h = 4$.

6. Setze $h = 5$. Es ist $32 < 22 + \left\lfloor 7\frac{12}{6} \right\rfloor = 36$ und $a_5 \neq a_4$.

   Wegen $a_5 > a_4$, $a_5 < \hat{b}$ und $32 < 22 + 12$ setze $z = 22 + 12 = 34$, $x_1 = x_2 = x_3 = 1$, $x_4 = 0$, $x_5 = 1$, $x_6 = 0$.

   Es ist $z = 34 < 36 = U$, also setze $i = 5$

6. Setze $h = 6$. Es ist $34 \geq 22 + \left\lfloor 7\frac{12}{10} \right\rfloor = 30$

5. Es wird $i = \max\{k < 4 : \hat{x}_k = 1\} = 3$ und $\hat{b} = 7 + 2 = 9$, $\hat{z} = 22 - 6 = 16$, $\hat{x}_3 = 0$, $j = i + 1 = 4$. Es ist $\hat{b} - a_3 = 9 - 2 \geq w_3 = 4$.

2. Es ist $a_4 \leq \hat{b}$. Es ist $r = \min\{i : \bar{a}_i > 9 + 5\} = 5$, $c' = \bar{c}_4 - \bar{c}_3 = 10$, $a' = \bar{a}_4 - \bar{a}_3 = 4$ und wegen $r \leq n$

$$u = \max\left\{ \left\lfloor (9-4)\frac{12}{10} \right\rfloor, \left\lfloor 12 - (6 - (9-4))\frac{10}{4} \right\rfloor \right\} = \max\{6, 9\} = 9$$

   Es ist $34 < \hat{z} + c' + u = 16 + 10 + 9 = 35$ und $u > 0$

3. Setze $\hat{b} = 9 - 4 = 5$ und $\hat{z} = 16 + 10 = 26$ sowie $\hat{x}_4 = 1$, $j = r = 5$.

   Es ist $5 = \hat{b} \leq w_5 = 10$ und $z = 34 \geq \hat{z} = 26$

5. Es ist $i = \max\{k < 5 : \hat{x}_k = 1\} = 4$.

   Setze $\hat{b} = \hat{b} + a_4 = 5 + 4 = 9$, $\hat{z} = \hat{z} - c_4 = 26 - 10 = 16$, $\hat{x}_4 = 0$, $j = 5$. Es ist $i > l = 1$

   Es ist $\hat{b} - a_4 = 9 - 4 = 5 < 6 = w_4$. Setze $j = 4$, $h = 4$.

6. Setze $h = 5$. Es ist $34 \geq 16 + \left\lfloor 9\frac{12}{6} \right\rfloor = 34$

5. Es ist $i = \max\{k < 5 : \hat{x}_k = 1\} = 2$.

   Setze $\hat{b} = \hat{b} + a_2 = 9 + 2 = 11,\ \hat{z} = \hat{z} - c_2 = 16 - 8 = 8,\ \hat{x}_2 = 0,\ j = 3$.

   Es ist $\hat{b} - a_2 = 11 - 2 = 9 > 2 = w_2$.

2. Es ist $a_3 < \hat{b}$. Ferner ist $r = \min\{i : \bar{a}_i > 11 + 3\} = 5$, somit gilt $c' = \bar{c}_4 - \bar{c}_2 = 16$ und $a' = \bar{a}_4 - \bar{a}_2 = 6$. Es $r \leq n$, also berechne

$$u = \max\left\{\left\lfloor (11-6)\,\frac{12}{10}\right\rfloor, \left\lfloor 12 - (6 - (11-6))\,\frac{10}{4}\right\rfloor\right\} = \max\{6, 9\} = 9$$

   Es ist $z = 34 \geq 8 + 16 + 9 = 33 = \hat{z} + c' + u$.

5. Es ist $i = \max\{k < 3 : \hat{x}_k = 1\} = 1$. Setze $\hat{b} = 11 + 1 = 12,\ \hat{z} = 8 - 18 = 0,$ $\hat{x}_1 = 0,\ j = 2,\ f = 0$.

   Ferner ist $\hat{b} - a_1 = 12 - 1 = 11 > 2 = w_1$

2. Es ist $a_2 < \hat{b}$. Ferner ist $r = \min\{i : a_i > 12 + 1\} = 5$,dann gilt $c' = \bar{c}_4 - \bar{c}_1 = 24$ und $a' = \bar{a}_4 - \bar{a}_1 = 8$. Es $r \leq n$, also berechne

$$u = \max\left\{\left\lfloor (12-8)\,\frac{12}{10}\right\rfloor, \left\lfloor 12 - (6 - (12-8))\,\frac{10}{4}\right\rfloor\right\} = \max\{4, 7\} = 7$$

   Es ist $z = 34 > 0 + 24 + 7 = \hat{z} + c' + u$

5. Es existiert kein $i = \max\{k < 2 : \hat{x}_k = 1\}$. **Stopp**

Es wurde also die uns bereits bekannte Optimallösung $z = 34,\ x^T = (1, 1, 1, 0, 1, 0)$ bestimmt. Die folgende Skizze zeigt den Ablauf des Verfahrens. Sie notiert die Festsetzungen des Verfahrens zum Vektor $\hat{x}$ jeweils bis zu einem Rückwärtsschritt und die Veränderungen von z und U.

Skizze:

2.3.2 Erläuterung zum Verfahren

Im folgenden sollen einige der Vorschriften des Verfahrens näher erläutert werden. Zunächst sei betont, dass die für das Verfahren vorausgesetzte zulässige Lösung durchaus aus $x = 0$ mit $z = 0$ bestehen kann und die obere Schranke aus $U = \infty$. Auf der anderen Seite können x, z und U auch aus der Vorabanwendung einer LU-Prozedur entspringen.

Start Anfänglich setze die Werte der laufenden Lösung auf die Nulllösung. Ferner werden einige technische Größen initialisiert: z.B. die kumulierten Gewichte und Nutzenwerte. Die Flagge f wird benötigt, wenn eine lokale obere Schranke global gemacht werden soll, l ist ein Hilfsindex dazu. Zusätzlich werden vorab die Größen w_k berechnet, die jeweils rechts vom Index k das kleinste Gewicht festhalten. Es wird $\hat{x}$ auf den Nullvektor gestetzt.

Vorwärtsschritt1 Der Vorwärtsschritt1, startend beim Index j, setzt so viele Variablen wie möglich nacheinander auf den Wert $\hat{x}_k = 1$. Dabei werden erste Variablen k, deren Gewicht nicht in den Rucksack passt, auf $\hat{x}_k = 0$ gesetzt. Die Anzahl der 1en hintereinander ist durch den kritischen Index r bestimmt. Dieser wird zunächst berechnet und daraus auch die zugehörigen (lokale) obere Schranke $c' + u$. Dabei wird über r_j die Kenntnis aus dem bisherigen Verlauf des Verfahrens ausgenutzt, welche aufeinanderfolgenden Gewichte garantiert in die Restkapazität passen.

Ist die Flagge $f = 0$, so wird globale obere Schranke U ggf. verbessert. Dies ist wie folgt zu verstehen: ist $f = 0$, so befindet sich das Verfahren am obersten offenen Knoten und es gilt $x_k = 0$, für alle $k \leq l$, während alle Knoten mit $x_l = 1$ bereits

abgearbeitet und ausgelotet sind. Das laufende z gibt den Zielwert der besten zulässigen Lösung mit $x_l = 1$ an, $\hat{z} + c' + u$ eine obere Schranke für den Zielwert der zulässigen Lösungen mit $x_l = 0$.

$u = 0$ ergibt sich übrigens, wenn die lokale Dantziglösung lokal optimal ist. j beschreibt immer den ersten Gegenstand der lokalen Aufgabe mit Restkapazität $\hat{b}$, für den $\hat{x}_j$ neu festgelegt werden soll.

Sicherung Hier werden $\hat{x}_k,\ \hat{z},\ \hat{b}$ auf den Stand gebracht und es wird mit der Größe w_r geprüft, ob ein weiterer Vorwärtsschritt1 erfolgreich sein kann. Zudem wird die Hilfsgröße r_k aktualisiert.

Aktualisierung Hier wird die laufende Lösung gespeichert. Dies wird nur angelaufen, wenn eine bessere zulässige Lösung gefunden wurde.

Rückwärtsschritt Der Rückwärtsschritt vollführt in einem ausgeloteten Knoten ein Backtracking gemäß LIFO. Trifft man dabei auf den obersten offenen Knoten, so wird $f = 0$ gesetzt.

Ferner wird der Übergang zum Vorwärtsschritt2 eingeleitet.

Vorwärtsschritt2 In den Vorwärtsschritt2 springt das Verfahren in der folgenden Situation: es wurde eine Verzweigung durchgeführt, die Setzung $\hat{x}_i = 1$ wurde ausgelotet, es ist nun $\hat{x}_i = 0$ gesetzt. Dabei passt der $i-$te Gegenstand zur Zeit in den Rucksack. Diese Information wird ausgenutzt zu einem verkürzten Vorwärtsschritt und liefert die Motivation, das Gewicht a_i als Referenz heranzuziehen.

Es gilt also zur Zeit $\hat{b} - a_i > 0$. Der Vorwärtsschritt1 wird angelaufen, wenn sogar $\hat{b} - a_i \geq w_i$ gilt, wenn also ausser i noch ein weiterer Gegenstand in den Rucksack passt. Gilt dies nicht, wird der Vorwärtsschritt2 angesprungen. Es gilt dann: passt ein Gegenstand $h > i$ mit $a_h \geq a_i$ in den Rucksack, so passt mit Sicherheit *kein weiterer* hinein.

Man versucht, den Gegenstand $h := i + 1$ einzupacken. Ist dann die obere Schranke nicht zu schlecht, betrachtet man zwei Fälle: $a_h > a_i$ und $a_h < a_i$. Gilt $a_h = a_i$, so passt a_h zwar in den Rucksack, führt aber zu einer *potentiell schlechteren* Lösung als die bereits ausgelotete. Man erhöht h um eins und läuft erneut den Vorwärtsschritt2 an.

Fall "$a_h > a_i$" : passt Gegenstand h hinein, und ist die Lösung mit $x_i = 0, ..., x_{h-1} = 0,\ x_h = 1,\ x_{h+1} = 0, ..., x_n = 0$ besser als die bestbekannte, so wird sie festgehalten und danach ein Seitwärtsschritt gemacht ersetze $\hat{x}_h = 1$ durch $\hat{x}_h = 0$ und führe

erneut einen Vorwärtsschritt2 durch. Man beachte dazu: es gilt dabei $\hat{b} - a_h < \hat{b} - a_i < w_i \leq w_h$.

Fall "$a_h < a_i$" : es passt nach Konstruktion nun h in den Rucksack. Falls dann aber kein weiterer Gegenstand hineinpasst ($\hat{b} - a_h < w_h$), so ist die Lösung, die nur h einpackt schlechter als die, die stattdessen i einpackt und das ist bereits untersucht. Also geht man in diesem Fall erneut zum Vorwärtsschritt2. Andernfalls geht man zum Vorwärtsschritt1.

2.3.3 Weiterentwicklung von MT1

Der numerische Erfolg von HSV zeigt, wie vorteilhaft dieser B&B Ansatz ist. Die Steigerung der Effektivität zur die Verbesserungen von MT1 (siehe unten) zeigen, dass es sich lohnt, diesen Ansatz weiterzuentwickeln. Die folgenden Ideen könnten dabei Verwendung finden.

Weitere Verbesserungsmöglichkeiten

Eine der wesentlichen Veränderungen von MT1 gegenüber HSV ist das Verwenden von Informationen über den bisherigen Verlauf des Verfahrens im Vorwärtsschritt 2. Dort wird die Kenntnis ausgenutzt, dass in bestimmten Situationen aus dem bisherigen Verlauf des Verfahren bekannt ist, das die aktuelle Restkapazität eine bestimmte positive Größe hat.

Diese Idee kann dahingehend ausgeweitet werden, dass man im Verfahren weitere Informationen über den bisherigen Verlauf speichert, z.B. den jeweiligen Anlaufindex für einen potentiellen Rückwärtsschritt.

Auch kann eine Verschärfung der lokalen Schranken in Erwägung gezogen werden. Etwa kann die Anwendung von Satz 2.3 in Betracht gezogen oder anstelle der oberen Schranke U_2 die Schranke U_{23} verwendet werden. (Übungsaufgabe)

Die Prozedur LUMT

Da die Prozedur MT1 eine Verfeinerung der Prozedur HSV ist und über HSV eine scharfe und schnelle LU-Prozedur formuliert werden konnte, kann eine solche, potentiell noch schnellere LU-Prozedur auch auf der Grundlage von MT1 erarbeitet werden. Dazu muss lediglich zu jedem Blatt des entstehenden Verzweigungsbaums (wie skizziert) eine lokale obere Schranke berechnet werden. Das Maximum der lokalen oberen Schranken bildet dann wieder eine globale obere Schranke für (KP_{01}). Wir nennen sie **LUMT.**

Zu beachten ist wie bei LUHS, dass der Verzweigungsbaum vollständig sein muss, dass also jeder Knoten einmal in $x_i = 0$ und $x_i = 1$ verzweigt wird, wenn x_i die Verzweigungsvariable des Knotens ist. Wenn diese Vollständigkeit des Verzweigungsbaums, wie bei HSV, von vorne herein nicht gegeben ist, muss sie nachträglich hergestellt werden. Dies geschieht völlig analog zu LUHS. (siehe Teilabschnitt über das Verfahren MT2)

Ein intergiertes Verfahren

Natürlich kann MT1 wie HSV alleine oder nach Anwendung eines Preprocessing auf eine vorsortierte Aufgabe (KP_{01}) angewendet werden. Zwei Aspekte können bei einer Anwendung des Preprocessing unter Verwendung von LUMT als bedauerlich angesehen werden:

1. eine Anwendung von MT1 auf die reduzierte Aufgabe wiederholt viele Rechnungen, die bereits zuvor durch LUMT durchgeführt wurden
2. es wird die Reduzierung nur vorab durchgeführt und nicht bei Verschärfung der unteren Schranke wiederholt.

Diese Mängel können durch eine Anwendung des Verfahrens MT1 mit integriertem Preprocessing, MT1i, geheilt werden. Diese integrierte Vorgehensweise soll im Folgenden näher erläutert werden, die genauere Ausgestaltung aber dem Hörer überlassen werden.

Man verwendet zunächst billige, aber leistungsfähige Prozeduren zum Preprocessing, z.B. LU5G und LMR. Danach wendet man LUMT auf das Indizes-Intervall $[k, n]$ an, wobei k den kritischen Index beschreibt. Dadurch verschärfen sich die Schranken der Aufgabe potentiell erneut. Hat sich insbesondere die untere Schranke verbessert, kann mit den bereits vorhandenen oberen Schranken U_k^p, $p = 1$ oder $p = 0$, eine erneute Reduzierung der Aufgabenstellung versucht werden.

Wendet man nun erneut LUMT auf das Index-Intervall $[k-1, n']$, $n' \leq n$ die aktuelle Dimension der um die am weitesten rechts stehenden Fixierungen reduzierten Aufgabe, an, so kann der vorherige Verlauf der Prozedur LUMT als Teil der neuen Anwendung dieser Prozedur angesehen werden.

Nach erneuter Reduzierung kann dieser Prozess fortgesetzt werden, solange bis man mit $k - \alpha$, $\alpha \in \mathbf{N}$ die letzte noch freie Variable erreicht hat und die Aufgabe gelöst ist. (Übungsaufgabe)

2.4 Verbesserte Dynamische Optimierung

Eine Verbesserung der oben dargestellten Methode der dynamischen Optimierung, DP1, kann sich ergeben, wenn die Anzahl der gespeicherten und damit auch verwalteten Daten weiter reduziert wird.

2.4.1 Dominierte Zustände

Ein Ansatzpunkt für eine Verbesserung des Verfahrens DP1 erschließt sich aus der folgenden

Beobachtung: Betrachtet werde die Aufgabe (KP_{01}) und ein Eintrag $\left(\hat{b}, P_{\hat{b}}\right)$ in der Bearbeitungstabelle von (KP_{01}) durch DP1 auf der Stufe m, wobei wie vereinbart $\hat{b}$ die Kapazität der aus (KP_{01}) abgeleiteten Aufgabe $\left(KP_m\left[\hat{b}\right]\right)$ mit $x_{m+1} = ... = x_n = 0$ und $P_{\hat{b}} := f_m\left[\hat{b}\right]$ desssen Nutzen sind. Der Wert $X_{\hat{b}}$, der $(x_1, ..., x_m)$ in kodierter Form angibt wurde hier weggelassen. Wir wollen ein solches Zweitupel einen **Zustand** des Verfahrens nennen. Gleichzeitig betrachten wir noch einen weiteren Zustand auf der gleichen Stufe m, (y, P_y).

Gilt nun $y \leq \hat{b}$ und $P_y \geq P_{\hat{b}}$ und $\left(\hat{b}, P_{\hat{b}}\right) \neq (y, P_y)$, so kann in DP1 jeder optimale Zielfunktionswert, der im Rahmen der Rückwärtsrechnung auf der $m_$ten Stufe über $\left(\hat{b}, P_{\hat{b}}, X_{\hat{b}}\right)$ gewonnen wird, auch über (y, P_y, X_y) gewonnen werden! Denn der letztere Zustand verbraucht potentiell weniger Gewichtsresourcen und erzielt einen potentiell höheren Nutzen. Fügt man also die Werte von $x_{m+1}, ..., x_n$ aus der Optimallösung zu (y, P_y, X_y) hinzu, so erhält man eine zulässige Lösung mit potentiell höherem Zielfunktionswert, also ebenfalls eine Optimallösung.

Bezeichnung: Wir sagen, (y, P_y) **dominiert** $\left(\hat{b}, P_{\hat{b}}\right)$.

Zu beachten ist, dass diese Relation transitiv ist: dominiert (y, P_y) den Zustand $\left(\hat{b}, P_{\hat{b}}\right)$ und (w, P_w) den Zustand (y, P_y), so dominiert auch (w, P_w) den Zustand $\left(\hat{b}, P_{\hat{b}}\right)$.

Es ist zur Einsparung von Arbeitsaufwand und Speicherplatz daher anzustreben, die dominierten Zustände im Verfahren DP1 *möglichst wegzulassen*, denn eine Optimallösung kann ja wie beobachtet aus undominierten Zuständen gewonnen werden. Wir stellen fest, dass die Maximumbildung in der Bellman Rekursionsformel bereits als Aussondern von dominierten Zuständen interpretiert werden kann.

Geometrische Interpretation: Man kann sich den Begriff der dominierten Zustände leicht aus einer multikriteriellen Sichtweise geometrisch klarmachen: Man sucht ja eine Lösung des gegebenen Problems *mit möglichst hohem Nutzen bei möglichst geringem Gewicht*. Wenn man also die Einschränkung des Gewichts aufhebt und stattdessen die Reduzierung des Gewichts als zweite Zielsetzung ansieht, so entstehen Punktepaare (Gewicht,Nutzen), die eine Punktewolke ergeben, wenn man alle möglichen Kombinationen $x_1, ..., x_n \in \{0, 1\}$ betrachtet.

Skizze:

Die zu undominierten Zuständen gehörigen Punkte bilden genau die **Pareto-optimalen** oder **effizienten** Punkte. Dies sind definitionsgemäß die Punkte, zu denen sich oberhalb und links keine weiteren Punkte befinden.

Es ist dann auch geometrisch klar, dass es immer genügt, nur die Pareto-optimalen Punkte zu kennen, um die Optimallösung der gestellten Aufgabe zu erhalten: betrachten wir etwa eine Punktwolke aus $2^3 = 8$ Punkten mit vier effizienten Punkten, entstanden über DP1 aus den ersten drei Variablen.

Fügt man dann eine vierte Variable hinzu, so entsteht folgendes Bild mit 16 Punkten:

Wir sehen, dass die neuen effizienten Punkte aus den alten hervorgehen. Man kann sich also auf jeder Stufe auf die Pareto-optimalen Punkte beschränken!

Beispiel

Zur Erläuterung der Auswirkung des Weglassens von dominierten Zuständen betrachten wir noch einmal unser Standard-Beispiel und seine Bearbeitung mit DP1.

Gegenstand j	2	3	4	5
Nutzen c_j	8	6	10	12
Gewicht a_j	2	2	4	6

mit $b = 11$ und Vornutzen $\hat{z} = 8$

Dort ergab sich die folgende linke Tabelle. Entfernt man *im Nachhinein* die laut Definition dominierten Zustände, so ergibt sich daraus die rechte Tabelle.

$\hat{b}\backslash m$	1	2	3	4
0	0,0	0,0	0,0	0,0
1	0,0	0,0	0,0	0,0
2	8,1	8,1	8,1	8,1
3	8,1	8,1	8,1	8,1
4	8,1	14,3	14,3	14,3
5	8,1	14,3	14,3	14,3
6	8,1	14,3	18,5	18,5
7	8,1	14,3	18,5	18,5
8	8,1	14,3	24,7	24,7
9	8,1	14,3	24,7	24,7
10	8,1	14,3	24,7	26,11
11	8,1	14,3	24,7	26,11

$\rightsquigarrow$

$\hat{b}\backslash m$	1	2	3	4
0	0,0	0,0	0,0	0,0
1				
2	8,1	8,1	8,1	8,1
3				
4		14,3	14,3	14,3
5				
6			18,5	18,5
7				
8			24,7	24,7
9				
10				26,11
11				

Die Optimallösung der Aufgabe, zu ermitteln in der letzten Spalte bei $\hat{b} = 11$, wird bei $\hat{b} = 10$ abgelesen. Dies signalisiert, dass im Optimalfall nur 10 der 11 Einheiten benutzt werden.

Nun wäre es ja sinnvoll, wenn ein Verfahren der Dynamischen Optimierung gleich nur undominierte Zustände berechnet. Dazu liefert die obige geometrische Darstellun der Zustände die folgenden **Idee**: sind die undommierten Zustände auf Stufe $(m-1)$ nach aufsteigendem Gewicht sortiert vorgegeben, so kann man alle diese Zustände einmal durchgehen (abgezählt mit einem Laufindex h), als Zustände der Stufe m bei $x_m = 0$ und gleichzeitig noch einmal durchgehen (abgezählt mit einem Laufindex i) und zu diesen jeweils a_m im Gewicht und c_m beim Nutzen hinzuaddieren. Dies ergibt insgesamt doppelt so viele potentziell undominierte Zustände für

Stufe m wie zuvor in stufe $(m-1)$. Ordnet man den zweiten Typ der Zustände, sofern sie einen Dominanztest bestehen, bei aufsteigendem Gewicht entsprechend ein, so findet man alle undominierten Zustände auf Stufe m. Dabei liefert die Abzählung über den Laufindex h die übergeordnete Abzählung, die des Laufindex i wird jeweils nachgezogen und über Fallunterscheidungen eingeordnet. Die neu ermittelten Zustände auf Stufe m zählen wir mit dem Laufindex k ab.

2.4.2 Beschreibung eines verkürzten Verfahrens

Die folgende Weiterentwicklung des Verfahrens $DP1$ berechnet nun nur noch nicht dominierte Zustände, entfermt also dominierte Zustände sofort und nicht erst im Nachhinein. Dies führt oft zu einem kürzeren Ablauf.

Wir benutzen folgende Notationen, jeweils bezogen auf die Stufe m bei fest vorgegebem m :

$d = 2^{m-1}$

$i =$ Zähler der nichtdominierten Zustände, für die $x_m = 1$ gesetzt wird

$X1_i = (x_1, ..., x_{m-1})$ in kodierter Form

$W1_i =$ Gewicht $\hat{b}$ des $i-$ten Zustands $\left(\sum_{j=1}^{m-1} a_j x_j\right)$

$P1_i =$ Nutzen $f_m\left[W1_i\right]$ des $i-$ten Zustands $\left(\sum_{j=1}^{m-1} c_j x_j\right)$

$k =$ Zähler der neu zu speichernden Zustände für Stufe m

$h =$ Zähler der alten Zuständeaus Stufe $m-1$, die wegen $x_m = 0$ übernommen werden

y, p und x sind Zwischenspeicher, es gilt immer $y = W1_i + a_m$

Prozedur DP2 Input: (KP_{01})

Output: Optimallösung x_{opt} mit Zielwert z_{opt}

1. **Start:** Setze $W1_0 = 0,\ P1_0 = 0,\ X1_1 = 0,\ s = 1,\ d = 2,\ W1_1 = a_1,\ P1_1 = c_1,\ X1_1 = 1, W1_2 = \infty$.

2. **Iteration:**

   Für $m = 2$ bis n führe aus:

   setze $i = 0,\ k = 0,\ h = 1,\ y = W1_i + a_m = a_m,$

   $W2_0 = 0,\ P2_0 = 0,\ X2_0 = 0$

solange $\min(y, W1_h) \leq b$ führe aus:

Fall $W1_h < y$:

setze $p = P1_h$ und $x = X1_h$

ist $p > P2_k$, so setze $k = k+1,\ W2_k = W1_h,\ P2_k = p,\ X2_k = x$

setze $h = h+1$

Fall $W1_h = y$:

setze $p = P1_h$ und $x = X1_h$

ist $P1_i + c_m > p$, so setze $p = P1_i + c_m$ und $x = X1_i + d$

setze $i = i+1$ und $y = W1_i + a_m$

ist $p > P2_k$, so setze $k = k+1,\ W2_k = W1_h,\ P2_k = p,\ X2_k = x$

setze $h = h+1$

Fall $W1_h > y$:

ist $P1_i + c_m > P2_k$, so setze $k = k+1$,

$W2_k = y,\ P2_k = P1_i + c_m,\ X2_k = X1_i + d$

setze $i = i+1,\ y = W1_i + a_m$

setze $d = 2d$ und $W1_{k+1} = \infty$

benenne $W2,\ P2,\ X2$ in $W1,\ P1,\ X1$ um

3. **Ergebnis:**

dekodiere $X1_k$, um x_{opt} zu erhalten und setze $z_{opt} = P1_k$

Ohne Beweis sei benannt, dass die Prozedur die Komplexität $\mathcal{O}(\min(2^{n+1}, nb))$ hat. Die Prozedur $DP2$ hat also einen geringeren Aufwand als $DP1$, wenn $2^{n+1} < nb$ gilt. Das folgende Beispiel zeigt, dass zumindest bei Handrechnung das Verfahren in der Praxis recht aufwendig ist.

Beispiel

Bearbeiten wir zuerst unser (reduziertes) Beispiel formal mit dem Verfahren und versuchen wir dann, Details besser zu verstehen. Hier sind noch einmal die Daten

Gegenstand j	2	3	4	5
Nutzen c_j	8	6	10	12
Gewicht a_j	2	2	4	6

mit $b = 11$ und Vorkosten $\hat{z} = 8$

Start: zum Start setzen wir $W1_0 = 0,\ P1_0 = 0,\ X1_1 = 0,\ d = 2,\ W1_1 = 2,\ P1_1 = 8,\ X1_1 = 1$

Zu Stufe $m = 2$:

Zunächst setze $i = 0,\ k = 0,\ h = 1,\ y = 2$, und

l	W1	P1	X1	W2	P2	X2
0	0	0	0	0	0	0
1	2	8	1			
2	∞					

es ist $\min(y, W1_1) = \min(2, 2) = 2 \leq 11 = b$

Fall $W1_1 = y$:

setze $p = P1_h = 8$ und $x = X1_h = 1$. Ferner ist $P1_0 + c_2 = 0 + 6 < 8$. Setze $i = 1$ und $y = W1_1 + a_2 = 2 + 2 = 4$

nun gilt $p = 8 > 0 = P2_k$, also setze $k = 1,\ W2_1 = 2,\ P2_1 = 8,\ X2_1 = 1,\ h = 2$

Es gilt nun $i = 1,\ k = 1,\ h = 2,\ y = 4$ und

l	W1	P1	X1	W2	P2	X2
0	0	0	0	0	0	0
1	2	8	1	2	8	1
2	∞					

Prüfe erneut $\min(y, W1_h) \leq b$. Es ist $\min(4, \infty) = 4 \leq 11$

Fall $W1_2 > y$.

Daher prüfe $P1_i + c_m > P2_k$, d.h. $8 + 6 > 0$, dies ist der Fall, also setze $k = k + 1,\ W2_k = y,\ P2_k = P1_i + c_m,\ X2_k = X1_i + d$, d.h. $k = 2,\ W2_2 = 4,\ P2_2 = 8 + 6 = 14,\ X2_2 = 3$, und $i = i + 1,\ y = W1_i + a_m$, d.h. $i = 2,\ y = \infty + 2 = \infty$.

Es gilt nun $i = 2,\ k = 2,\ h = 2,\ y = \infty$ und

l	W1	P1	X1	W2	P2	X2
0	0	0	0	0	0	0
1	2	8	1	2	8	1
2	∞			4	14	3

Prüfe erneut $\min(y, W1_h) \leq b$. dies ist wegen $y = \infty$ nicht der Fall. Wir setzen $s = 2,\ d = 4$, benennen $W2, P2, X2$ um in $W1, P1, X1$ und erhöhen m um 1.

Zu Stufe $m = 3$:

Zunächst setze $i = 0,\ k = 0,\ h = 1,\ y = 4$, und

l	W1	P1	X1	W2	P2	X2
0	0	0	0	0	0	0
1	2	8	1			
2	4	14	3			
3	∞					

Nun prüfe $\min(y, W1_h) \leq b$. Es ist $\min(4, 2) \leq 11$.

Fall $W1_h < y$.

Also setze $p = P1_h = 8$ und $x = X1_h = 1$. Es ist $p > P2_k$, daher setze $k = k + 1,\ W2_k = W1_h,\ P2_k = p,\ X2_k = x$, d.h. $k = 1,\ W2_1 = 2,\ P2_1 = 8,\ X2_1 = 1$. Ferner setze $h = 2$.

Es gilt $s = 2,\ i = 0,\ k = 1,\ h = 2,\ y = 4$, und

l	W1	P1	X1	W2	P2	X2
0	0	0	0	0	0	0
1	2	8	1	2	8	1
2	4	14	3			
3	∞					

Nun prüfe $\min(y, W1_h) \leq b$. Es ist $\min(4, 4) \leq 11$.

Fall $W1_h = y$.

Setze $p = P1_h$ und $x = X1_h$, also $p = 14$ und $x = 3$. Prüfe $P1_i + c_m > p$, d.h. $0 + 10 > 14$. Dies ist nicht der Fall.

Ferner setze $i = i + 1 = 1$ und $y = W1_i + a_m = 2 + 4 = 6$

Prüfe nun $p > P2_k$, also $14 > 8$. Dies ist der Fall, also setze $k = k + 1,\ W2_k = W1_h,\ P2_k = p,\ X2_k = x$, d.h. $k = 2,\ W2_2 = 4,\ P2_2 = 14,\ X2_2 = 3$. Setze $h = h + 1 = 3$

Es gilt $i = 1,\ k = 2,\ h = 3,\ y = 6$, und

l	W1	P1	X1	W2	P2	X2
0	0	0	0	0	0	0
1	2	8	1	2	8	1
2	4	14	3	4	14	3
3	∞					

Nun prüfe $\min(y, W1_h) \leq b$. Es ist $\min(6, \infty) \leq 11$.

Fall $W1_h > y$.

Nun prüfe $P1_i + c_m > P2_k$, d.h. $8+10 > 10$. Also setze $k = k+1,\ W2_k = y,\ P2_k = P1_i + c_m,\ X2_k = X1_i + d$, d.h. $k = 3,\ W2_3 = 6,\ P2_3 = 8 + 10 = 18,\ X2_3 = X1_1 + d = 1 + 4 = 5$.

Setze $i = i + 1 = 2,\ y = W1_i + a_m = 4 + 4 = 8$

Es gilt $i = 2,\ k = 3,\ h = 3,\ y = 8$, und

l	W1	P1	X1	W2	P2	X2
0	0	0	0	0	0	0
1	2	8	1	2	8	1
2	4	14	3	4	14	3
3	∞			6	18	5

Nun prüfe $\min(y, W1_h) \leq b$. Es ist $\min(8, \infty) \leq 11$.

Fall $W1_h > y$

Nun prüfe $P1_i + c_m > P2_k$, d.h. $14+10 > 10$. Also setze $k = k+1,\ W2_k = y,\ P2_k = P1_i + c_m,\ X2_k = X1_i + d$, d.h. $k = 4,\ W2_4 = 8,\ P2_4 = 14 + 10 = 24,\ X2_4 = X1_2 + d = 3 + 4 = 7$.

Setze $i = i + 1 = 3,\ y = W1_i + a_m = \infty + 4 = \infty$

Es gilt $i = 3,\ k = 4,\ h = 3,\ y = \infty$, und

l	W1	P1	X1	W2	P2	X2
0	0	0	0	0	0	0
1	2	8	1	2	8	1
2	4	14	3	4	14	3
3	∞			6	18	5
4				8	24	7

Nun prüfe $\min(y, W1_h) \leq b$. Es ist $\min(\infty, \infty) > 11$.

Setze $d = 2d = 8$, Benenne um.

Zu Stufe $m = 4$:

Es gilt $i = 0,\ k = 0,\ h = 1,\ y = 6$, und

l	W1	P1	X1	W2	P2	X2
0	0	0	0	0	0	0
1	2	8	1			
2	4	14	3			
3	6	18	5			
4	8	24	7			
5	∞					

Nun prüfe $\min(y, W1_h) \leq b$. Es ist $\min(6, 2) \leq 11$.

Fall $W1_h < y$

Also setze $p = P1_h = 8$ und $x = X1_h = 1$. Es ist $p > P2_k$, daher setze $k = k + 1,\ W2_k = W1_h,\ P2_k = p,\ X2_k = x$, d.h. $k = 1,\ W2_1 = 2,\ P2_1 = 8,\ X2_1 = 1$. Ferner setze $h = 2$.

Es gilt $i = 0,\ k = 1,\ h = 2,\ y = 6$, und

l	W1	P1	X1	W2	P2	X2
0	0	0	0	0	0	0
1	2	8	1	2	8	1
2	4	14	3			
3	6	18	5			
4	8	24	7			
5	∞					

Nun prüfe $\min(y, W1_h) \leq b$. Es ist $\min(6, 4) \leq 11$.

Fall $W1_h < y$

Also setze $p = P1_h = 10$ und $x = X1_h = 4$. Es ist $p > P2_k$, daher setze $k = k + 1,\ W2_k = W1_h,\ P2_k = p,\ X2_k = x$, d.h. $k = 2,\ W2_2 = 4,\ P2_2 = 10,\ X2_2 = 4$. Ferner setze $h = 3$.

Es gilt $i = 0,\ k = 2,\ h = 3,\ y = 6$, und

l	W1	P1	X1	W2	P2	X2
0	0	0	0	0	0	0
1	2	8	1	2	8	1
2	4	14	3	4	14	3
3	6	18	5			
4	8	24	7			
5	∞					

Nun prüfe $\min(y, W1_h) \leq b$. Es ist $\min(6, 6) \leq 11$.

Fall $W1_h = y$.

Setze $p = P1_h$ und $x = X1_h$, also $p = 18$ und $x = 5$. Prüfe $P1_i + c_m > p$, d.h. $0+12 > 18$. Dies ist nicht der Fall. Setze $i = i+1 = 1$ und $y = W1_i + a_m = 2+6 = 8$

Prüfe nun $p > P2_k$, also $18 > 10$. Dies ist der Fall, also setze $k = k + 1,\ W2_k = W1_h,\ P2_k = p,\ X2_k = x$, d.h. $k = 3,\ W2_3 = 6,\ P2_3 = 18,\ X2_3 = 5$. Setze $h = h + 1 = 4$

Es gilt $i = 1,\ k = 3,\ h = 4,\ y = 8$, und

l	W1	P1	X1	W2	P2	X2
0	0	0	0	0	0	0
1	2	8	1	2	8	1
2	4	14	3	4	14	3
3	6	18	5	6	18	5
4	8	24	7			
5	∞					

Nun prüfe $\min(y, W1_h) \leq b$. Es ist $\min(8, 8) \leq 11$.

Fall $W1_h = y$.

Setze $p = P1_h$ und $x = X1_h$, also $p = 24$ und $x = 7$. Prüfe $P1_i + c_m > p$, d.h. $8+12 > 24$. Dies ist nicht der Fall. Setze $i = i+1 = 2$ und $y = W1_i + a_m = 4+6 = 10$

Prüfe nun $p > P2_k$, also $24 > 18$. Dies ist der Fall, also setze $k = k + 1,\ W2_k = W1_h,\ P2_k = p,\ X2_k = x$, d.h. $k = 4,\ W2_4 = 8,\ P2_4 = 24,\ X2_4 = 7$. Setze $h = h + 1 = 5$

Es gilt $i = 2,\ k = 4,\ h = 5,\ y = 10$, und

l	W1	P1	X1	W2	P2	X2
0	0	0	0	0	0	0
1	2	8	1	2	8	1
2	4	14	3	4	14	3
3	6	18	5	6	18	5
4	8	24	7	8	24	7
5	∞					

Nun prüfe $\min(y, W1_h) \leq b$. Es ist $\min(10, \infty) \leq 11$.

Fall $W1_h > y$.

Nun prüfe $P1_i + c_m > P2_k$, d.h. $14+12 > 18$. Also setze $k = k+1,\ W2_k = y,\ P2_k = P1_i + c_m,\ X2_k = X1_i + d$, d.h. $k = 5,\ W2_5 = 10,\ P2_5 = 14 + 12 = 26,\ X2_5 = 2 + 8 = 11$.

Setze $i = i + 1 = 3,\ y = W1_i + a_m = 6 + 6 = 12$

Es gilt $i = 3,\ k = 5,\ h = 5,\ y = 12$, und

l	W1	P1	X1	W2	P2	X2
0	0	0	0	0	0	0
1	2	8	1	2	8	1
2	4	14	3	4	14	3
3	6	18	5	6	18	5
4	8	24	7	8	24	7
5	∞			10	26	11

Nun prüfe $\min(y, W1_h) \leq b$. Es ist $\min(12, \infty) > 11$.

Setze also $d = 16$ und benenne um-

Ergebnis: Als Endtableau ergibt sich

l	W1	P1	X1
0	0	0	0
1	2	8	1
2	4	14	3
3	6	18	5
4	8	24	7
5	10	26	11

mit der Optimallösung $x^T = (1, 1, 0, 1)$ mit $z = 26 + 8 = 34$.

Das Verfahren liefert also genau das von den dominierten Zuständen befreite Tableau, wie wir es oben schon vorausgesagt hatten.

Beobachtung: Da das Vefahren rein additiv arbeitet, verläuft die Rechnung im Beispiel identisch, wenn man zuvor alle Gewichte und die Kapazität z.B. mit dem Faktor 10 multipliziert. Hier zeigt sich der Vorteil von DP2 gegenüber DP1. Während die Anwendung von DP2 unverändert durch die Multiplikation abläuft, wird die von DP1 ernorm aufgebläht, weil zu jeder ganzen Zahl zwischen 0 und b eine Zeile gehört.

Erläuterung des Verfahrens

Versuchen wir nun zu verstehen, wie das Verfahren im Detail funktioniert. Zunächst werde festgestellt, dass während eines Durchgangs (zur Berechnung der Zustände der Stufe m) immer $y = W1_i + a_m$ gilt. Dies gilt anfänglich durch Festsetzung, später wird immer dann $y = W1_i + a_m$ gesetzt, wenn i geändert wird, d.h. $i = i + 1$ gesetzt wird.

Schlüsselpunkt ist nun der folgende

Satz 2.5 *Das Verfahren produziert auf jeder Stufe eine Abfolge von Zuständen mit strikt ansteigendem $W1$ und $P1$ Werten.*

Beweis: Wir führen den Beweis durch Induktion über die Spalten.

$m = 1$: Der Anfang des Verfahrens wird zum Start gesetzt. Er erfüllt die Behauptung.

$m \Longrightarrow m + 1$: Gehen wir also davon aus, dass die Zustände in Stufe m berechnet und dabei die Werte $W1_j$ und $P1_j$ strikt ansteigend seien mit ansteigendem j. Betrachten wir eine Situation, in der eine Zeile $(W2_j, P2_j, X2_j)$ zur Stufe $(m + 1)$ neu geschrieben wird. Neu geschrieben wird eine solche Zeile in den drei Fällen im Baustein **Iteration.** Betrachten wir also nacheinander diese drei Fälle.

Es seien h, i und k die vom Verfahren gesetzten Indizes, k bezeichnet dabei die zuletzt geschriebene neue Zeile, h die laufende alte Zeile und i die letzte alte Zeile, zu der y festgelegt wurde. Alle $W1_j$ seien nach Induktionsannahme strikt ansteigend sortiert, ebenfalls alle neuen Zeilen bis einschließlich k.

Fall $W1_h \leq y$: Hier wird die alte Zeile h einfach in die neue Zeile $(k + 1)$ kopiert, wenn $P1_h > P2_k$ gilt. Für den Nutzen gilt also die Behauptung, für das Gewicht muss $W1_h > W2_k$ nachgewiesen werden. Überprüfen wir dazu, wie die neue Zeile k geschrieben wurde.

Geschah dies im ersten oder im zweiten Fall, so gilt $W2_k = W1_{h'}$ für ein $h' < h$ und die Behauptung gilt nach Induktionsannahme. Geschah dies im dritten Fall, so wurde $W2_k = y$ mit $W1_{h'} > y$ für ein $h' \leq h$. also gilt auch hier die Behauptung.

Fall $W1_h = y$: Hier ist die Argumentation die gleiche wie im vorherigen Fall, nur dass nicht notwendig die alte Zeile kopiert wird, sondern ggf. der TZustand $(W1_i, P1_i, X1_i)$ als neue Zeile geschrieben wird.

Fall $W1_h > y$: In diesem Fall wird die neue Zeile $k + 1$ nicht kopiert sondern neu berechnet. Wiederum wird über die Vorschriften dafür gesorgt, dass $P2_{k+1} > P2_k$ gilt. Es ist also erneut $W2_{k+1} > W2_k$ nachzuprüfen, d.h. es muss diskutiert werden, wie die neue Zeile k entstanden ist.

Geschah dies im Fall $W1_h < y$, so ist nichts zu zeigen, da sich y nicht verändert hat und daher nach Konstruktion $W2_{k+1} = y > W1_{h'} = W2_k$ für ein $h' \leq h$ gilt.

Geschah dies im Fall $W1_h = y$, so gilt nach dem Aufschreiben der neuen Zeile k, dass y echt angewachsen ist, also wieder $W2_{k+1} = y > W1_{h'} = W2_k$ für ein $h' \leq h$ gilt.

Geschah dies im Fall $W1_h > y$, so gilt $W2_{k+1} = y$ für das aktuelle y und $W2_k = y$ für das vorherige y, sagen wir $W2_k = y'$. Es gilt aber $y' < y$ wegen $W1_{i+1} > W1_i$

nach Konstruktion und Induktionsannahme. □

Nach Kenntnis des Satzes ist nun klar, dass die jeweilige Abfrage vor dem Schreiben einer neuen Zeile: $P2_{k+1} > P2_k$ sicherstellt, dass die zu schreibende neue Zeile nicht von *den* bisherigen neuen Zeilen dominiert wird. Im Nachhinein wird keine Zeile einer Spalte von anderen Zeilen der Spalte dominiert. Im Sinne der Transitivität des Dominierens ist damit jeder geschriebene Zustand nicht dominiert.

Gleichzeitig werden alle denkbaren neuen Zustände gemäß Bellmans Gleichung konstruiert und überprüft, ob sie geschrieben werden können. Dadurch ist geklärt, dass alle Zustände des Verfahrens DP1 durch einen Zustand des Verfahrens DP2 dominiert werden.

2.4.3 Das Verfahren DPT

Toth hat darauf hingewiesen, dass auch DP2 noch viele unnötige Zustände produziert, nämlich solche, die letztlich nicht zur Optimallösung führen. Er schlug folgende **zusätzliche Regeln** für die Anwendung von DP1 oder DP2 vor:

Fall DP1: Wenn ein Zustand in der $m-$ten Stufe ein Gewicht W besitzt, für das

$$(i) \quad W + \sum_{j=m+1}^{n} a_j < b$$

oder

$$(ii) \quad b - \min_{m<j\leq n} \{a_j\} < W < b$$

gilt, so kann dieser Zustand nicht den optimalen Zustand produzieren und kann ausgeschlossen werden. Beide Regeln sind leicht einzusehen.

Fall DP2: hier gelten die gleichen Regeln, bis auf die Ausnahme, dass in jeder Stufe das höchste Gewicht W, das die Regel (i) erfüllt, beibehalten wird ebenso wie der allerletzte Zustand.

Die Anwendung dieser Reglen führt bei unserem *Beispiel*

Gegenstand j	2	3	4	5
Nutzen c_j	8	6	10	12
Gewicht a_j	2	2	4	6

mit $b = 11$ und Vorkosten $\hat{z} = 8$

im Nachhinein zu folgenden Zuständen.

Fall DP1:

$\hat{b}\backslash m$	1	2	3	4
0	0,0	0,0	0,0	0,0
1	0,0	0,0	0,0	0,0
2	8,1	8,1	8,1	8,1
3	8,1	8,1	8,1	8,1
4	8,1	14,3	14,3	14,3
5	8,1	14,3	14,3	14,3
6	8,1	14,3	18,5	18,5
7	8,1	14,3	18,5	18,5
8	8,1	14,3	24,7	24,7
9	8,1	14,3	24,7	24,7
10	8,1	14,3	24,7	26,11
11	8,1	14,3	24,7	26,11

$\rightsquigarrow$

$\hat{b}\backslash m$	1	2	3	4
0	0,0			
1	0,0	0,0		
2	8,1	8,1		
3	8,1	8,1		
4	8,1	14,3		
5	8,1	14,3	14,3	
6	8,1	14,3		
7	8,1	14,3		
8	8,1			
9	8,1			
10				
11	8,1	14,3	24,7	26,11

Fall DP2:

$\hat{b}\backslash m$	1	2	3	4
0	0,0	0,0	0,0	0,0
1				
2	8,1	8,1	8,1	8,1
3				
4		14,3	14,3	14,3
5				
6			18,5	18,5
7				
8			24,7	24,7
9				
10				26,11
11				

$\rightsquigarrow$

$\hat{b}\backslash m$	1	2	3	4
0	0,0	0,0		
1				
2	8,1	8,1		
3				
4		14,3	14,3	
5				
6				
7				
8			24,7	
9				
10				26,11
11				

Auf eine genaue Algorithimisierung der Verfahrens (wie DP2) wird verzichtet.

Bemerkung: Die Prozedur DP2 erzeugt, wie gesehen, in der Regel sehr viel weniger Zustände als die Prozedur DP1. Allerdings ist die Berechnung der Zustände bei DP2 aufwändiger als bei DP1. Daher kann es sinnvoll sein, DP1 und DP2 zu kombinieren. Toth hat ebenfalls eine solche Kombinations-Prozedur vorgeschlagen, die auf heuristischen Umschaltregeln zwischen DP1 und DP2 beruht. Darauf soll hier nicht näher eingegangen werden.

2.5 Erfahrungen

2.5.1 Tests zu den Verfahren

Martello und Toth (1990) geben in ihrem Buch Ergebnisse von Computertests zu MT1 an, über die hier als erstes berichtet werden soll. Danach wird von eigenen Tests berichtet.

Beispielaufgaben und Verfahren

Zum Vergleich der Verfahren haben sie jeweils 20 Aufgaben gleicher Struktur unterschiedlicher Größe und Datenausprägung getestet und dann Durchschittswerte angegeben.

Die Aufgabenklassen von je 20 Aufgaben mit zufälligen Daten wurden wie folgt erzeugt:

- Klasse "unkorreliert": die Daten a_j und c_j sind gleichverteilt im Bereich $[1, 100]$
- Klasse "schwach korreliert": die a_j sind gleichverteilt im Bereich $[1, 100]$, die Daten c_j gleichverteilt im Bereich $[a_j - 10, a_j + 10]$.
- Klasse "stark korreliert": die a_j sind gleichverteilt im Bereich $[1, 100]$, die Nutzenkoeffizienten $c_j = a_j + 10$.

Zu beachten ist, dass bei stärker korrelierten Daten die Effizienzen nicht so stark schwanken, was zu erhöhten Schwierigkeiten beim Lösen der Aufgaben führt. Martello und Toth betonen, dass sie schwach korrelierten Aufgabenstellungen am ehesten wirklichkeitsgetreu sind.

Die Kapazität wurde durchgehende zu $b = 0.5 * \sum_{j=1}^{n} a_j$ festgesetzt. Die Dimensionen der Probleme wurden zu $n = 50,\ n = 100,\ n = 200$ festgelegt.

Untersucht wurden unter anderen die folgenden Verfahren

- Das Verfahren HSV von Horowitz und Sahni mit und ohne vorgeschaltete Reduzierung MTR.
- Das Verfahren MT1 von Martello und Toth mit und ohne vorgeschaltete Reduzierung MTR
- Das Verfahren DPT von Toth mit vorgeschalteter Reduzierung MTR

Martello und Toth haben die untersuchten Verfahren in Fortran IV programmiert und auf einer CDC-Cyber 730 implementiert und getestet.

Laufzeitvergleich

Die folgenden Tabellen geben die Laufzeiten der Verfahren zur Lösung der Aufgaben in Sekunden an, jeweils als Durchschnittswert von 20 zufälligen Aufgaben. alle Zeitangaben beinhalten die Zeit für die Vollsortierung. Deshalb sei zunächst die Sortier-Zeit (in Sekunden) für die Aufgaben angegeben:

n	50	100	200
Zeit	0,008	0,018	0,041

Es folgen die Zeitangaben für die einzelnen Klassen. Gesamt-Bearbeitungszeiten über 500 Sekunden für die Aufgabengruppen werden mit "-" gekennzeichnet.

Unkorrelierte Aufgaben

n	HSV	MTR+HSV	MT1	MTR+MT1	MTR+DPT
50	0,031	0,016	0,016	0,013	0,020
100	0,075	0,028	0,030	0,026	0,043
200	0,237	0,065	0,068	0,057	0,090

Als Ergebnis kann man sehen, dass sich der Einsatz der Reduzierungsroutine MTR auf jedem Fall lohnt. Ferner sieht man die deutliche Dominanz von MT1 über HSV und dass die Dynamische Optimierung DPT durchaus mithalten kann. Insgesamt empfiehlt sich für diese Aufgabenklasse, MT1 mit vorgeschaltetem MTR einzusetzen.

Schwach korrelierte Aufgaben

n	HSV	MTR+HSV	MT1	MTR+MT1	MTR+DPT
50	0,038	0,025	0,022	0,020	0,071
100	0,079	0,042	0,040	0,031	0,158
200	0,185	0,070	0,069	0,055	0,223

Bei den schwach korrelierten Aufgaben hat die Dynamische Optimierung DPT größere Schwierigkeiten. Auch fast alle anderen Laufzeiten sind schlechter, wobei der Eindruck entsteht, dass die prozentuale Erhöhung mit steigendem n abnimmt, sogar negativ werden kann.

Stark korrelierte Aufgaben

n	HSV	MTR+HSV	MT1	MTR+MT1	MTR+DPT
50	–	–	4,870	4,019	0,370
100	–	–	–	–	1,409
200	–	–	–	–	3,936

Hier zeigt sich, welche großen Schwierigkeiten aus der starken Korrelierung entstehen können. HSV mit und ohne MTR ist nicht in der Lage, die Aufgaben in der gegebenen Zeit zu lösen, MT1 nur bei Aufgaben mit $n = 50$. Dagegen schafft DPT alle Aufgaben innerhalb der Zeitgrenze und ist daher bei dieser Struktur den anderen Verfahren überlegen.

Weitere Tests

In eigenen Test wurden verschiedene LU Prozeduren zusammen mit den besprochenen Versionen von MTR und MT1 getestet, um die Wirkung der einzelnen LU Prozeduren und die Reduzierungsleistung der Prozedur MTR besser einschätzen zu können. Es wurden die gleichen Parameter wie im letzten Kapitel für die Aufgabenerzeugung verwendet: n gibt die Dimension des Beispiels an, $[0, R]$ die Schwankungsbreite der Daten a_j, Die Daten c_j sind dann von der Form $c_j = a_j + \rho + \tilde{c}_j$ mit $\tilde{c}_j \in [-r, r]$.

Bei den folgenden Tabellen gibt z immer den Zielwert der besten bislang bekannten zulässigen Lösung (Untere Schranke) und u immer die aktuelle obere Schranke an, in der Entwicklung von links nach rechts bei Ánwendung der LU Prozedur, von MTR und MT1. Z_{LU} gibt die Laufzeit der LU Prozedur in Sekunden, Z_{MTR} die der Prozedur MTR und schließlich GZeit die Gesamtbearbeitungszeit an. Alle diese Zeiten beinhalten zur besseren Vergleichbarkeit nicht die Zeit der Sortierung. Dim gibt die reduzierte Dimension nach Anwendung von MTR an.

LUHSf bezeichnet die Prozedur LUHS auf $[r, t] = [k - \alpha, k + \alpha]$ mit fixem $\alpha = 5$, LUHSv mit $\alpha = \sqrt{n}$, LUHSe bezeichnet LUHS auf dem Intervall $[r, t] = [k, n]$ und "ohne" kennzeichnet das Auslassen jeglicher LU Prozedur, ohne2 das Auslassen der LU Prozedur und der Reduzierung.

Beispiel : 100_1000_10_10 ($n = 100, R = 1000,\ \rho = 10,\ r = 10,\ z_o = 24773$)

LU	z	u	Z_{LU}	z	u	Dim	Z_{MTR}	GZeit
LUDD	24214	24777	0,000	24761	24776	97	0,000	0,000
LUMS	24759	24777	0,000	24761	24776	97	0,000	0,016
LU5G	24689	24776	0,000	24761	24776	97	0,000	0,000
LUU6	24773	24776	0,000	24773	24776	27	0,000	0,015
LUHSf	24773	24777	0,000	24773	24776	27	0,000	0,000
LUHSv	24773	24777	0,000	24773	24776	27	0,000	0,000
LUHSe	24771	24776	0,000	24771	24776	46	0,000	0,000
ohne	0	∞	0,000	24761	24776	97	0,000	0,000
ohne2	0	∞	0,000	0	∞	100	0,000	0,000

Beispiel : 500_1000_10_0 ($n = 500, R = 1000,\ \rho = 10,\ r = 0,\ z_o = 125994$)

LU	z	u	Z_{LU}	z	u	Dim	Z_{MTR}	GZeit
LUDD	125797	125996	0,000	125993	125996	193	0,000	1,109
LUMS	125797	125996	0,000	125993	125996	193	0,000	1,093
LU5G	125797	125996	0,000	125993	125996	193	0,000	1,109
LUU6	125895	125996	0,000	125993	125996	193	0,000	1,100
LUHSf	125895	125996	0,000	125993	125996	193	0,000	1,078
LUHSv	124994	125996	0,000	125994	125996	128	0,000	16,03
LUHSe	124797	125996	0,000	125993	125996	193	0,000	1,109
ohne	0	∞	0,000	125993	125996	193	0,000	1,093
ohne2	0	∞	0,000	0	∞	500	0,000	1,094

Beispiel : 1000_1000_0_1000 ($n = 1000, R = 1000,\ \rho = 0,\ r = 1000,\ z_o = 282901$)

LU	z	u	Z_{LU}	z	u	Dim	Z_{MTR}	GZeit
LUDD	282614	282902	0,000	282890	282902	106	0,000	0,000
LUMS	281887	282902	0,000	282890	282902	106	0,000	0,000
LU5G	281889	282902	0,000	282890	282902	106	0,000	0,000
LUU6	281896	282902	0,000	282896	282902	67	0,015	0,015
LUHSf	281896	282902	0,000	282896	282902	67	0,000	0,000
LUHSv	281901	282902	0,016	282901	282902	9	0,016	0,032
LUHSe	281889	282902	0,016	282890	282902	106	0,000	0,016
ohne	0	∞	0,000	282890	282902	106	0,000	0,000
ohne2	0	∞	0,000	0	∞	1000	0,000	0,015

Beispiel : 10000_1000_10_1000 ($n = 10000, R = 1000,\ \rho = 10,\ r = 1000,\ z_o = 4040518$)

LU	z	u	Z_{LU}	z	u	Dim	Z_{MTR}	GZeit
LUDD	4039944	4040519	0,000	4040515	4040519	62	0,016	0,016
LUMS	4040504	4040519	0,000	4040515	4040519	62	0,031	0,031
LU5G	4040515	4040519	0,000	4040515	4040519	62	0,031	0,031
LUU6	4040516	4040519	0,000	4040516	4040519	46	0,016	0,047
LUHSf	4040516	4040519	0,031	4040516	4040519	46	0,031	0,031
LUHSv	4040518	4040519	0,015	4040518	4040519	6	0,016	0,031
LUHSe	4040518	4040519	0,047	4040518	4040519	6	0,016	0,063
ohne	0	∞	0,000	4040515	4040519	62	0,031	0,031
ohne2	0	∞	0,000	0	∞	10000	0,000	2,03

Die (sicher nicht repräsentativen) numerischen Auswertungen legen folgende Vermutungen nahe:

1. die LU Prozedur LUHSv liefert in der Regel die beste untere Schranke, während die obere Schranke nahezu unabhängig von der LU Prozedur ist und damit vom Aufwand, der zu ihrer Berechnung betrieben wird.

2. Die Prozedur gleicht bei der unteren Schranke Schwächen der vorangehenden LU Prozedur aus, wenngleich sich hier die iterativen LU Prozeduren hervortun, unter ihnen wieder eindeutig LUHSv als beste. Die obere Schranke verbessert sich kaum, sondern bleibt zumeist gleich.

3. Die durch MTR erreibare Reduzierung der Dimension der Aufgabe ist beachtlich und hängt eindeutig von der Güte der erreichten unteren Schranke ab. Insofern sammeln hier die iterativen LU Prozeduren wieder Pluspunkte, insbesondere LUHSv.

4. Ernüchtend ist, dass bei den bearbeiteten Aufgaben der zeitliche Gesamtaufwand von den eingesetzten Methoden nahezu unabhängig ist, von wenigen Ausreißern einmal abgesehen. Erstanlich ist, dass MTR asls alleinige Preprocessing Methode hierbei so gut mithalten kann.

5. unkorellierte Aufgaben mit 10000 können durch die angewandten Methoden gelöst werden.

Zusätzlich wurde noch die praktische Laufzeit der Prozedur MT1 gegenüber der Prozedur HSV getestet, und zwar auf einer Teilmenge der Beispielaufgaben aus Kapitel 2. Zu betonen ist, dass diesmal die Zeit der Vorsortierung der Aufgaben mit einbezogen ist. Die Zeit zum sortieren einer Aufgabe mit 50 oder 500 Variablen benötigt keine massbare Zeit, die von 5000 Variablen etwa 3,0 Sekunden. Ferner sei darauf hingewiesen, dass in beiden Fällen die Iterationen über die Anzahl der Aufrufe des (Haupt-)Vorwärtsschritts gezählt werden.

$r = 100$

dim	50	50	500	500	5000	5000
	it	sec	it	sec	it	sec
HSV	128	$0,00$	1336	$0,05$	1884	$3,06$
MT1	54	$0,00$	376	$0,03$	3	$3,078$

$r = 10$

dim	50	50	500	500	5000	5000
	it	sec	it	sec	it	sec
HSV	96	$0,00$	519	$0,03$	1737	$3,03$
MT1	45	$0,00$	6	$0,03$	2	$3,109$

$r = 1$

dim	50	50	500	500	5000	5000
	it	sec	it	sec	it	sec
HSV	$12T$	$0,00$	$5T$	$0,05$	–	$64,43$
MT1	812	$0,00$	9	$0,02$	2910	$3,67$

Man kann vermuten, dass dass HSV immer deutlich mehr Iterationen benötigt als MT1, dass aber im unkorellierten und schwach korellierten Bereich die Leistungsfähigkeit beider Methoden vergleichbar ist. Im höherdimensionalen und stärker korellierten Bereich könnte MT1 Vorteile haben.

2.5.2 Übungsaufgaben

Aufgabe 1

Versuchen Sie, die Prozedur LUHS dadurch zu verbessern, dass die Berechnung der oberen Schranke verschärft wird. Dies könnte durch eine gemischte Verwendung der Schranken U_2 und U_3 geschehen. Man sucht im Bereich $[r,t]$ immer zuerst einen neuen (lokaler) kritischen Index. Wenn ein solcher gefunden wird, dann wird dieser verwendet wird. Wird keiner gefunden, so berechnet man die Schranke wie bisher.

Aufgabe 2

Man verwende die Eigenschaft den Breitensuchverfahrens, immer wieder die obere und untere Schranke zuu aktualisieren, um daraus eine LU Prozedur zu bauen, die iteriert vorgeht, aber nicht auf einem Intervall $[r,t]$ arbeitet.

Aufgabe 3

Man konstruiere eine neue LU Prozedur, indem man dynamische Optimeirung auf ein ein gewähltes Indizes-Teilintervall $[r,t]$ anwendet und somit sehr scharfe untere Schranken berechnet. Man baue dabei durch Vorabverwendung einer billigen LU Prozedur eine zusätzliche Möglichkeit ein, eine obere Schranke zu berechnen.

Aufgabe 4

Man verbessere die Prozedur LMR, indem man lokale kritische Indizes auf einem vorgegebenen Intervall $[r,t]$ sucht und erst im Fall, dass man nicht fündig wird, Satz 2.1 zur Berechnung der oberen Schrank heranzieht. Wird dabei $[r,t]$ beschränkt, also von n unabhängig gehalten, so ändert sich die Komplexität der Prozedur nicht.

Aufgabe 5

Man verbessere die Prozedur MT1 wie im obigen Abschnitt empfohlen, durch Einarbeitung von weiteren Techniken zur Berechnung der oberen Schranken, etwa Satz 2.3.

Aufgabe 6

Man bastle aus MT1 eine LU Prozedur LUMT, die ähnlich wie bei LUHS durch Berechnung lokaler oberer Schranken eine Möglichkeit schafft, gleichzeitig mit der Lösung der Aufgabe auf eine eingeschränkten Intervall $[r,t]$ eine scharfe obere Schranke für (KP_{01}) zu berechnen.

Aufgabe 7

Man verwende die in Aufgabe 6 konstruierte LU Prozedur LUMT wie oben beschrieben iteriert, um MT1 um eine immer wiederkehrende Reduzierung zu erweitern.

Kapitel 3

Große 0-1-Rucksack Probleme

3.1 Aufgaben großer Dimension

Es kann experimentell festgestellt werden, dass bei sehr großen $0-1$-Rucksack Problemen die Vorsortierung der Daten nach abfallendem Nutzen-Gewichts-Quotienten *mehr als drei Viertel* der gesamten Bearbeitungszeit einnimmt, wenn die bisher betrachteten Algorithmen angewendet werden. Hier besteht Verbesserungsbedarf. Es ist angebracht zu überlegen, ob auf die vollständige Sortierung verzichtet werden kann.

Wie wir wissen, wird die Sortierung der Daten zunächst mal ausgenutzt, um eine untere und eine obere Schranke zu berechnen. Im Folgenden wollen wir erkennen, dass solche Schranken auch ohne vollständige Sortierung berechnet werden können und damit auch ein Preprocessing durchgeführt werden kann. Bei großen 0-1-Rucksack Problemen darf man sich also Zeit-Vorteile versprechen, wenn man die vollständige Vorsortierung der Variablen vermeiden kann.

Vorgegeben sei weiterhin das 0-1-Rucksack Problem

$$(KP_{01}) \quad \begin{cases} \text{maximiere} & c^T x \\ \text{s.d.} & a^T x \leq b \\ & 0 \leq x \leq e, \quad x \in \mathbf{Z}^n \end{cases}$$

mit vorgegebenen Vektoren $a = (a_1, ..., a_n)^T$, $c = (c_1, ..., c_n)^T \in \mathbf{Z}^n$, $e = (1, ..., 1)^T \in \mathbf{Z}^n_+$, $b \in \mathbf{Z}$.

3.1.1 Preprocessing

Wie schon im Fall der vorsortierter Aufgaben benötigen wir zum Preprocessing scharfe, aber einfach zu berechnende untere und obere Schranken für (KP_{01}). Die

einfachste Möglichkeit, solche zu erlangen bietet wieder die LP Relaxation.

Dantzigschranken

Essentiell zur Berechnung der Schranken ist die Kenntnis des kritischen Index. Die folgende Prozedur zur Bestimmung des kritischen Index ohne vorherige Sortierung geht auf Balas und Zemel (1980) zurück.

Bevor eine solche Prozedur notiert werden kann, muss allerdings erst einmal festgelegt werden, ob bzw. wie der kritische Index in Abwesenheit einer Sortierung definiert werden kann.

Kritische Partition Wir definieren zunächst anstelle eines kritischen Index eine *kritische Partition:*

Definition: Eine Partition der Menge $N := \{1, ..., n\} = JH \cup JC \cup JL$ mit den Eigenschaften:

1. Es gibt ein $\zeta > 0$ mit

$$\begin{aligned} \zeta_j &> \zeta \quad \text{für alle } j \in JH \\ \zeta_j &= \zeta \quad \text{für alle } j \in JC \\ \zeta_j &< \zeta \quad \text{für alle } j \in JL \end{aligned}$$

2. und es gilt

$$\sum_{j \in JH} a_j \leq b < \sum_{j \in JH \cup JC} a_j$$

heißt **kritische Partition von** N.

Dass eine solche kritische Partition ohne vollständige Sortierung bestimmt werden kann, zeigt die Prozedur, die wir gleich erarbeiten werden.

Ist eine kritische Partition bestimmt, so kann leicht ein kritischer Index im alten Sinne ermittelt werden. Man betrachtet die Mengen JH, JL als intern (virtuell) nach abfallenden Nutzen-Gewichts-Quotienten sortiert, belässt JC in der vorliegenden Reihenfolge und hat dann eine Gesamtsortierung von N nach abfallenden Nutzen-Gewichts-Quotienten. Der zu dieser Sortierung zu berechnende kritische Index befindet sich nach Konstruktion im Bereich JC und kann ohne wirkliche Sortierung der Mengen JH und JL leicht bestimmt werden:

Man setzt $JC = \{e_1, ..., e_s\}$ mit $s := |JC|$. Dann ermittelt man den eindeutig bestimmten Index $k \in JC$, für den

$$\sum_{j=1}^{k-1} a_{e_j} \leq b - \sum_{j \in JH} a_j < \sum_{j=1}^{k} a_{e_j}$$

Offensichtlich legt k den zuvor beschriebenen kritischen Index fest. Es gilt übrigens nach Konstruktion

$$\zeta_j = \zeta_k \quad \text{für alle } j \in JC$$

Ist dieser Index bestimmt, kann noch eine andere Partition von N festgeschrieben werden

$$N = J_1 \cup \{k\} \cup J_0$$

mit $J_1 := JH \cup \{e_1, ..., e_{k-1}\}$, $J_0 := JL \cup \{e_{k+1}, ..., e_s\}$. Hierbei gilt

$$\begin{aligned} \zeta_j &\geq \zeta_k \quad \text{für alle } j \in J_1 \\ \zeta_j &\leq \zeta_k \quad \text{für alle } j \in J_0 \end{aligned}$$

Sind der kritische Index k und J_1, J_0 einmal bestimmt, so kann die **Dantzig-Schranke** als obere Schanke berechnet werden: Setzt man zur Abkürzung $\bar{z} = \sum_{j \in J_1} c_j$ und $\bar{a} = \sum_{j \in J_1} a_j$, so gilt

$$U = \bar{z} + \lfloor \zeta_k (b - \bar{a}) \rfloor$$

Eine untere Schranke für (KP_{01}) wird natürlich angegeben durch den Zielfunktionswert $\bar{z}_k$ der **Dantziglösung** $\bar{x}^T = (\bar{x}_1, ..., \bar{x}_n)$ mit

$$\bar{x}_j = \begin{cases} 1, & \text{falls } j \in J_1 \\ 0, & \text{sonst} \end{cases}$$

Dies ist nach Konstruktion eine zulässige Lösung zu (KP_{01}).

Eine potentiell bessere untere Schranke bekommt man auf einfache Weise, indem man nach der Dantziglösung noch eine **Vorwärts-Schwerpunkt Schranke** z_f und eine **Rückwärts-Schwerpunkt Schranke** z_b bestimmt:

$$\begin{aligned} z_b &= \max_{j \in J_1} \{\bar{z} + c_k - c_j : \bar{a} + a_k - a_j \leq b\} \\ z_f &= \max_{j \in J_0} \{\bar{z} + c_j : \bar{a} + a_j \leq b\}, \end{aligned}$$

sofern existent. Als untere Schranke L nimmt man den besseren der beiden Werte

$$L = \max \{z_b, z_f\}$$

Berechnet man z_b und z_f in dieser Reihenfolge, so kann man gleichzeitig noch zwei andere Kandidaten für eine untere Schranke mit nahezu dem gleichen Aufwand berechnen:

1. Unterstellt man, dass die Indizes in J_0 zumindest *grob nach abfallenden Effizienzen* sortiert sind, so macht auch der ganz normale "Greedy-Prozess" Sinn: ausgehend von $\bar{z}$ kann man der Reihe nach die Indizes aus J_0 durchgehen und immer dann, wenn das angetroffenen a_j noch in die (laufende aktualisierte) Restkapazität passt, den Gegenstand in den Rucksack aufnehmen. Wir bezeichnen die so erhaltene untere Schranke mit z_g

2. Das Gleiche kann man machen ausgehend von z_b, denn dies ist der Nutzen einer zulässigen Lösung, die im Abänderung zur Dantziglösung $x_k = 1$ und genau ein $x_j = 0$, $j \in J_1$, gesetzt hat. Auch diese Lösung kann geeignet durch Gegenstände aus J_0 ergänzt werden. Wir bezeichnen die so erhaltene untere Schranke mit z_e.

Damit ergibt sich die untere Schranke nun zu

$$L = \max\{z_e, z_f, z_g\}$$

Ferner sei darauf hingewiesen, dass problemlos auch die Minimal-Erweiterung der Dantziglösung verwendet werden kann, da diese lediglich die Kenntnis der Partition $N = J_1 \cup \{k\} \cup J_0$ voraussetzt. Analog zum Bisherigen gilt: Sucht man zuvor für jeden Index j das maximale Gewicht links von j, so kann man auch $x_k = 1$ setzen und prüfen, ob die so überschrittenen Kapazität durch setzen von $x_j = 0$ für ein $j < k$ wieder ins Positive gebracht werden kann. Danach kann eine Minimalerweiterung nach rechts vorgenommen werden.

Berechnung der kritischen Partition Eine Idee zur Bestimmung einer kritischen Partition $N = JH \cup JC \cup JL$ ist die folgende: Ausgehend von einer bereits bestehenden (ggf. trivialen) Partition $N = JH \cup JC \cup JL$, die noch keinen kritischen Nutzen-Gewichts-Quotienten nach obiger Definition festlegt, wählt man versuchsweise einen der zu JC gehörigen Quotienten λ und sortiert die Daten relativ zu λ um.

Prozedur LUDSu (LU Dantzig Schwerpunkt unsortiert)

Input: (KP_{01})

Output: Untere Schranke L und Obere Schranke U, Bereiche $J1, J0$ und kritischer Index k.

1. **Start:** Setze $JH := \varnothing, JL := \varnothing$ und $JC := N$. Setze $\bar{b} := b$ und $fertig := false$

2. **Kritischer Index:**

   While not $fertig$ do

   begin

   wähle einen Wert λ aus $R := \{\zeta_j : j \in JC\}$ % (*)

   Setze $JB := \{j \in JC : \zeta_j > \lambda\}, JS := \{j \in JC : \zeta_j < \lambda\}, JE := \{j \in JC : \zeta_j = \lambda\}$

   Setze $b' := \sum_{j \in JB} a_j,\ b'' := \sum_{j \in JB \cup JE} a_j$

   if $b' \leq \bar{b} < b''$ then $fertig := true$

   else if $b'' \leq \bar{b}$ then % λ `zu groß`

   setze $JH := JH \cup JB \cup JE$ und $JC := JS$, $\bar{b} := \bar{b} - b''$

   else % λ `zu klein`

   Setze $JL := JL \cup JS \cup JE$, $JC := JB$

   end

   end

   end

   Setze $JH := JH \cup JB,\ JL := JL \cup JS,\ JC =: \{e_1, ..., e_s\},\ \bar{b} := \bar{b} - b'$

   Setze $k := \min\left\{j : \bar{b} < \sum_{j=1}^{j} a_{e_j}\right\}$ und

   $J1 := JH \cup \{e_1, ..., e_{k-1}\},\ J0 := JL \cup \{e_{k+1}, ..., e_s\},\ \bar{b} = \bar{b} - \sum_{j=1}^{k} a_{e_j}$

3. **Schranken**

   Setze $\bar{z} = \sum_{j \in J_1} c_j$ und $U = \bar{z} + \left\lfloor \zeta_k \bar{b} \right\rfloor$ und berechne eine untere Schranke L unter Verwendung von

$$\begin{aligned} z_f &= \max_{j \in J0} \left\{\bar{z} + c_j : a_j \leq \bar{b}\right\} \quad \text{und} \\ z_b &= \max_{j \in J1} \left\{\bar{z} + c_k - c_j : a_k - a_j \leq \bar{b}\right\} \end{aligned}$$

Zu beachten ist, dass die Prozedur LUDSu genau wie die Prozedur LUDS eine Komplexität von $\mathcal{O}(n)$ hat, wenn man in (*) als λ immer den *Median der Werte in* R auswählt. Dies ist wie folgt einzusehen: Offensichtlich hat der Baustein **Schranken** die Laufzeit $\mathcal{O}(n)$, wenn begrenzt oft die Indizes aus J_1 und J_0 durchgegangen werden. Bleibt also nur der Baustein **Kritischer Index** zu betrachten. Den Median einer Menge von m Elementen zu finden benötigt eine Zeit von $\mathcal{O}(m)$. Also benötigt

jede Iteration der *While* Schleife eine Zeit von $\mathcal{O}(|JC|)$. Da wegen der Medianbildung in jeder Iteration die Anzahl der Elemente von JC zumindest halbiert wird, gilt die Behauptung.

Man kann anstelle des Medians der Menge $R := \{\zeta_j : j \in JC\}$ auch einen anderen Wert λ aus R auswählen. In der Regel steigt dann aber die Komplexität der Prozedur auf bis zu $\mathcal{O}(n^2)$ an.

Bemerkungen: Die Teilbereiche JB, JH, JC, JL, JS sollten als geordnete Listen aufgefasst werden und ihre Vereinigungen sollten die darin enthaltenen Anordnungsstrukturen in der in der Prozedur genannten Reihenfolge aufrechterhalten. Dann entsteht eine *grobe Teilordnung* der Indexmenge N. Eine solche hatten wir bei der Ermittlung der Greedy Lösung angesprochen.

Reduzierung

Mit Hilfe der Prozedur LUDSu als Unterprozedur der Prozedur ARM kann nun genau so eine Reduzierung der Aufgabenstellung betrieben werden wie mit Prozedur LUDS oder einer anderen LU Prozedur. Man kann dazu die Prozedur ARM unverändert übernehmen. Auch die Bemerkung im Anschluss an die Prozedur ARM gilt unverändert. Die Komplexität der Prozedur ist $\mathcal{O}(n^2)$

Dennoch ist Verwendung der Prozedur LUDSu in der Praxis aufwändiger, weil hier wegen der immer wieder zu berechnenden kritischen Indizes k' der aus den Fixierungen entstehenden Teilaufgaben eine *permanente (Teil-)Sortierung* inbegriffen ist. Deshalb kann man alternativ obere Schranken für die Teilaufgaben mit Satz 2.1 berechnen, die keine neuen kritischen Indizes k' benötigen, sondern mit den kritischen Index k auskommen, der ja bereits vorliegt. In diesem Fall verzichtet man allerdings auch auf die Berechnung neuer unterer Schranken im Rahmen der Reduzierungsprozedur.

Ein Beispiel

Wir illustrieren die Prozedur ARM mit Vorabanwendung der Prozedur LUDSu und Berechnung der oberen Schranken der Teilaufgaben an einem Beispiel. Dazu betrachten wir ein neues unsortiertes Zahlenbeispiel

j	1	2	3	4	5	6	7
c_j	39	5	37	70	10	20	7
a_j	20	3	19	31	6	10	4
ζ_j	1,950	1,667	1,947	2,258	1,667	2,000	1,750

mit $n = 7$ und $b = 50$.

Wenden wir also die Prozedur ARM an und verwenden wir dabei "geordnete Mengen":

1. Zunächst setzen wir $JF_1 = JF_0 = \varnothing$
2. Nun wenden wir die Prozedur LUDSu zur Bestimmung des kritischen Index an. Dazu setzen wir
   (a) $JH = JL = \varnothing$, $JC = N$, $b = \bar{b} = 50$ und $fertig = false$.
   (b) der Median der Quotienten ist $\zeta_3 = 1,947$. Es gilt also $JB = \{1, 4, 6\}$, $JE = \{3\}$, $JS = \{2, 5, 7\}$.
   Ferner ist $b' = 61$, $b'' = 80$ und $b' > b$. Daher setzte $JH = \varnothing, JC = \{1, 4, 6\}$, $JL = \{3\} \cup \{2, 5, 7\}$.
   Erneuter Eintritt in die While-Schleife. Es ist nun $\lambda = 6$, $JB = \{4\}$, $JE = \{6\}$, $JS = \{1\}$ und $b' = 31$, $b'' = 41$ und damit $b'' < b$. Setze also $JH = \{4\} \cup \{6\}$, $JC = \{1\}$ und $JL = \{3\} \cup \{2, 5, 7\}$ und $\bar{b} = 50 - 41 = 9$. Weil nun $0 = b' \leq \bar{b} < b'' = 20$ gilt, wird $fertig$ auf $true$ gesetzt. Ergebnis ist dann $N = J1 \cup \{1\} \cup J0$ mit $J1 = \{4\} \cup \{6\}$ und $J0 = \{3\} \cup \{2, 5, 7\}$.
   (c) Es ist nun $U = 70 + 20 + \lfloor 1.95 * 9 \rfloor = 107$. Ferner berechnen wir mit $\bar{b} = 9$, $\bar{z} = 90$ und

   $$z_b = \bar{z} + c_k - \min\left\{c_j : a_k - a_j \leq \bar{b}, j \in J1\right\} = 90 + 39 - \min\{70\} = 59$$

   angenommen bei $x^T = (1, 0, 0, 0, 0, 1, 0)$ sowie

   $$z_f = \bar{z} + \max\left\{c_j : a_j \leq \bar{b}, j \in J_0\right\} = 90 + \max\{5, 10, 7\} = 100$$

   angenommen bei $x^T = (0, 0, 0, 1, 1, 1, 0)$.
   Ferner ergibt sich dann $z_g = 90 + 10 + 5 = 105$, angenommen bei $x^T = (0, 1, 0, 1, 1, 1, 0)$ und $z_e = 59 + 37 = 96$, angenommen bei $x^T = (1, 0, 1, 0, 0, 1, 0)$.
   Es ist also $L = 105$, angenommen bei $x^T = (0, 1, 0, 1, 1, 1, 0)$ als aktuell beste zulässige Lösung.
3. Nun betrachten wir die Unterprobleme, bei denen $x_j = 0$ gesetzt wird für $j \in J1$.
   Für $j = 4$ ergibt sich als $U^4 = 90 - 70 + \lfloor 1.95 \cdot (9 + 31) \rfloor = 98 \leq 105$.
   Für $j = 6$ ergibt sich als $U^4 = 90 - 20 + \lfloor 1.95 \cdot (9 + 10) \rfloor = 107 > 105$.

4. Als Nächstes betrachten wir die Unterprobleme, bei denen $x_j = 1$ gesetzt wird für $j \in J0$.

   Für $j = 2$ ergibt sich $U^2 = 90 + 5 + \lfloor 1.95 \cdot (9 - 3) \rfloor = 106 > 105$.

   Für $j = 3$ ergibt sich $U^3 = 90 + 37 + \lfloor 1.95 \cdot (9 - 19) \rfloor = 107 > 105$.

   Für $j = 5$ ergibt sich $U^5 = 90 + 10 + \lfloor 1.95 \cdot (9 - 6) \rfloor = 105 \leq 105$.

   Für $j = 7$ ergibt sich $U^7 = 90 + 7 + \lfloor 1.95 \cdot (9 - 4) \rfloor = 106 > 105$.

5. Die Reduzierung ergibt also , dass $x_4 = 1$ und $x_5 = 0$ fixiert werden können. Weil nun die Restkapazität bei $\hat{b} = 50 - 31 = 19$ liegt, ergibt die Nachjustierung als weitere Fixierung $x_1 = 0$. Wegen $a_3 = \hat{b} = 19$ ermittelt man, dass die zulässige Lösung $x^T = (0, 0, 1, 1, 0, 0, 0)$ den Zielwert $z = 107$ hat, was wegen der gleich großen oberen Schranke als Optimalwert erkannt wird.

Bemerkung: Man kann zusätzlich oder alternativ die über die optimalen Lagrange-Multiplikatoren berechneten oberen Schranken für die Aufgaben $\left(KP_j^0\right)$ mit $j \in J1$ und $\left(KP_j^1\right)$ mit $j \in J0$ ermitteln. In unserem Beispiel berechnet man nach der Formel $U^j = \left\lfloor z^* - \left| c_j - a_j \frac{c_k}{a_k} \right| \right\rfloor$ folgendes:

1. Betrachten wir die Unterprobleme, bei denen $x_j = 0$ gesetzt wird für $j \in J1$.

   Für $j = 4$ ergibt sich als $U^4 = \lfloor 107.55 - |70 - 31 * 1.95| \rfloor = 98 \leq 105$. Also setze $JF_1 = \{4\}$

   Für $j = 6$ ergibt sich als $U^4 = \lfloor 107.55 - |20 - 10 * 1.95| \rfloor = 107 > 105$.

2. Als Nächstes betrachten wir die Unterprobleme, bei denen $x_j = 1$ gesetzt wird für $j \in J0$.

   Für $j = 2$ ergibt sich $U^2 = \lfloor 107.55 - |5 - 3 * 1.95| \rfloor = 106 > 105$.

   Für $j = 3$ ergibt sich $U^3 = \lfloor 107.55 - |37 - 19 * 1.95| \rfloor = 107 > 105$.

   Für $j = 5$ ergibt sich $U^5 = \lfloor 107.55 - |10 - 6 * 1.95| \rfloor = 105 \leq 105$.

   Für $j = 7$ ergibt sich $U^7 = \lfloor 107.55 - |7 - 4 * 1.95| \rfloor = 106 > 105$.

Damit ergeben sich genau die gleichen Schranken wie zuvor.

3.1.2 Kernbereiche

Über die Prozeduren LUDS oder LUDSu erhält man immer mit der Dantziglösung eine erste zulässige Lösung zum vorgegebenen 0-1 Rucksack Problem. Man hat experimentell festgestellt, dass in den meisten Fällen die optimale Lösung des Problems *nur in wenigen Komponenten* von der Dantziglösung abweicht. Darüber hinaus gilt sogar, dass die abweichenden Komponenten dabei *mehr oder weniger in der Nähe des break item* liegen. Die folgende Graphik gibt für zufällig erzeugte 0-1 Rucksackprogbleme mit 1000 Variablen nach Vorsortierung die relative Häufigkeit der Abweichung der Optimallösung an einer Komponente j von der Dantziglösung an (wobei die Aufgaben durch geeignetes b auf $k = 500$ getrimmt wurden).

Skizze (PISINGER (1995))

Es zeigt sich also, dass im Durchschnitt die überwiegende Mehrheit der Abweichungen in einem engen Bereich im den kritischen Index anzutreffen sind. Je nach individueller Aufgabe können die Abweichungen allerdings sehr verschieden liegen. Wir legen fest:

> Der **Kernbereich** einer Aufgabe wird definiert als das Index-Intervall, das zwischen der am weitestens links stehenden und der am weitesten rechts stehenden Abweichung der Optimallösung von der Dantziglösung festgelegt ist.

Dieser Kernbereich ist in seiner Ausdehnung natürlich von vorne herein nicht bekannt. Man kann versuchen, ihn zu schätzen. Experimentellen Untersuchungen von Martello/Toth zufolge lässt sich bei Aufgaben mit $n \geq 200$ die Ausdehnung dieses Kernbereichs C mit hoher Wahrscheinlichkeit durch $[k - \sqrt{n}, k + \sqrt{n}]$ eingrenzen. Der Kernbereich hat also im Durchschnitt eine Breite von höchstens $\delta := 2\sqrt{n} + 1$, ist aber möglicherweise viel kleiner.

Strategie

Da die Sortierung aller Variablen der Aufgabe bei großer Dimension so aufwändig ist, versucht man, stattdessen *zwei kleinere Teilbereiche* zu sortieren. Der erste Teilbereich besteht aus einem vorgewählten Bereich von Variablen, die in einem sortierten Zustand in enger Nähe zum break item liegen. Hierfür wählt man eine **Schätzung des Kerns** der Form $[k-\alpha, k+\alpha]$ mit vorgewähltem α, etwa $\alpha = 25$ (Vorschlag vom Balas/Zemel) oder $\alpha = \sqrt{n}$ (Vorschlag vom Martello/Toth), oder aber ein unsymmetrisches Teilintervall $[r, t]$ ähnlicher Abmessung..

Der zweite Teilbereich besteht aus den Variablen, die nach einer erfolgten Reduzierung der Aufgabe übrigbleiben. Es wird angestrebt, dabei *beide* Bereiche möglichst klein zu halten. Der erste Teilbereich kann willkürlich gewählt werden, der zweite ergibt sich in Abhängigkeit von der ersten Wahl. Will man diesen klein halten, so muss man dafür sorgen, dass die Reduzierung möglichst erfolgreich ist. Dies gelingt, wenn die ermittelte untere Schranke möglichst gut ist, was bedeutet, dass man eine möglichst gute zulässige Lösung finden muss. Es bietet sich also an, sich bei der Wahl des ersten Teilbereichs am Kern, bzw. einer Kernschätzung zu orientieren.

Die folgende Vorgehensweise zur Lösung sehr großer 0-1 Rucksackprobleme adaptiert die bereits verwendete Vierteilung des Lösungsprozesses bei vorsortierten Aufgaben auf den unsortierten Fall. Sie wurde von Balas und Zemel vorgeschlagen (1980):

1. Man sucht mittels Teilsortierung einen um das break item annähernd symmetrische Kernschätzung C innerhalb einer Partition

   $$H \cup C \cup L = N,$$

   für die $\zeta_j \geq \zeta_l$ für alle $j \in H,\ l \in C$ und $\zeta_j \leq \zeta_l$ für alle $j \in L,\ l \in C$. Dabei muss C garantiert den kritischen Index enthalten. Danach sortiert man die Indizes in C nach abfallenden Nutzen-Gewichts-Koeffizienten.

2. Man verwendet eine LU Prozedur, die mit dieser Struktur kompatibel ist, die also nicht auf eine Sortierung von H und L angewiesen ist.

3. Man betreibt Reduzierung der gegebenen Aufgabe über eine Reduzierungsprozedur, die ebenfalls nicht auf die Sortierung der Bereiche H und L angewiesen ist.

4. Man sortiert die nach Fixierung einzelner Variablen übrig gebliebene Aufgabe zu Ende und löst sie exakt.

Annäherung des Kernbereichs

Versuchen wir nun abzuklären, wie man günstig eine Kernschätzung C ermittelt.

Kernschätzung BZC Eine Möglichkeit der Teilsortierung der Daten, die eine Kernschätzung liefert, ergibt sich aus der oben beschriebenen Prozedur LUDGu. Diese ermittelt, wenn Zwischenergebnisse zu den entstehenden "geordneten Mengen" notiert werden, eine Partition der folgenden Art von N :

$$N = H_1 \cup H_2 \cup ... \cup H_{s_1} \cup \{k\} \cup L_{s_2} \cup ... \cup L_2 \cup L_1 \tag{3.1}$$

wobei die Nutzen-Gewichts-Quotienten innerhalb der Teilbereiche ungeordnet sind, ansonsten aber von links nach rechts abfallen.

Andererseits kann der Teilsortierungsprozess unterbrochen werden, wenn grob

$$N = H_1 \cup H_2 \cup ... \cup H_{s_1'} \cup C \cup L_{s_2'} \cup ... \cup L_2 \cup L_1 \tag{3.2}$$

erreicht ist mit $|C| \approx \delta$ für ein vorgewähltes δ, wobei C den dann noch nicht bekannten kritischen Index garantiert enthält. Wir nennen eine solche Zerlegung **teilsortierte Zerlegung** oder einfach **Teilsortierung** von N.

Hat man diese, so verfährt man wie oben vorgesehen: man setzt

$$H = H_1 \cup H_2 \cup ... \cup H_{s_1'} \quad \text{und} \quad L = L_{s_2'} \cup ... \cup L_2 \cup L_1,$$

sortiert C und bestimmt darin den kritischen Index.

Bemerkung: Liegt der kritische Index danach am Rand von C, so kann dieser Mangel ggf. durch Aufnahme eines Nachbarbereichs von C in den Schätz-Kern behoben werden. Im Nachhinein kann C auch durch Verschieben von Elementen in äußere Nachbarbereiche annähernd symmetrisch gestaltet werden.

Diese Modifikation der Prozedur LUDGu geht, ebenso wie die Prozedur LUDGu selbst, auf Balas/Zemel (1980) zurück.

Prozedur BZC (Balas/Zemel Core)

input: (KP_{01}) , δ, η

output: H, C, L, $\bar{b}$

setze zunächst $JH = JL = \varnothing$, $JC = \{1, ..., n\}$, $\bar{b} = b$, $l = 0$

wähle ein λ aus JC

solange nicht $fertig$ und $|JC| > \delta$ und $l < \eta$ führe aus:

begin

wähle einen Wert λ aus $R := \{\zeta_j : j \in JC\}$ % (*)

Setze $JB := \{j \in JC : \zeta_j > \lambda\}, JS := \{j \in JC : \zeta_j < \lambda\}, JE := \{j \in JC : \zeta_j = \lambda\}$

Setze $b' := \sum_{j \in JB} a_j,\ b'' := \sum_{j \in JB \cup JE} a_j$

if $b' \leq \bar{b} < b''$ then $fertig := true$

else if $b'' \leq \bar{b}$ then % λ `zu groß`

setze $JH := JH \cup JB \cup JE$ und $JC := JS$, $\bar{b} := \bar{b} - b''$

else % λ `zu klein`

Setze $JL := JL \cup JS \cup JE$, $JC := JB$

end

end

$l = l + 1$

end

Setze $H := JH \cup JB,\ L := JL \cup JS,\ C := JC,\ \bar{b} = \bar{b} - b'$

Sortiere C und ermittle den kritischen Index k.

Ggf. korrigiere C, damit k möglichst zentral liegt.

Beispiel In unserem Beispiel bietet sich nach der obigen Rechnung z. B. für $\delta = 3$ bereits nach der ersten Iteration die folgende Zerlegung an: $H = \varnothing$, $C = \{1, 4, 6\}$ und $L = \{3\} \cup \{2, 5, 7\}$. Sortiert man C, so lautet dies $C = \{4, 6, 1\}$, wobei 1 den kritischen Index darstellt. Dieser liegt also am rechten Rand von C. Also nimmt man z.B. den Index 3 in C auf und verschiebt vielleicht aus Symmetriegründen den Index 4 aus C in den noch leeren Bereich H. Danach ergibt sich die Partition

$$N = H_1 \cup C \cup L_1 = \{4\} \cup \{6, 1, 3\} \cup \{2, 5, 7\}$$

wobei C sortiert ist und das mittlere Element das break item darstellt.

Man kann dann z.B. die Schranke U_2 berechnen und eine zulässige Lösung der Teilaufgabe, die nur die Variablen $x_6,\ x_1,\ x_3$ frei zulässt und über die Restkapazität $\bar{b} = b - a_4 = 50 - 31 = 19$ verfügt.

Es gilt

$$U_2 = \max\left\{90 + \left\lfloor 9 \cdot \frac{37}{19} \right\rfloor, 129 + \left\lfloor (-11)\,\frac{20}{10} \right\rfloor\right\} = 107$$

und die Optimallösung der angesprochenen Teilaufgabe ist offensichtlich $x_3 = 1,\ x_1 = x_6 = 0$ mit Zielwert $z = 37$. Daraus ergibt sich die zulässige Lösung $x^T = (0, 0, 1, 1, 0, 0, 0)$

mit Zielwert $z = 70 + 37 = 107$, sodass wiederum erkannt wird, dass der optimale Zielfunktionswert 107 lautet.

Also ergibt sich auch, dass $\{6, 1, 3\}$ den eigentlichen Kern umfasst und dass aus dieser Kenntnis eines guten Schätz-Kerns eine erhebliche Verkürzung der Gesamt-Bearbeitungszeit resultieren kann.

BALAS UND ZEMEL empfehlen $\delta = 50$ und haben experimentell herausgefunden, dass es algorithmisch sehr günstig ist, als λ immer den *Median der ersten drei Elemente* aus R zu wählen. Manche Autoren verwenden auch den Median dreier anders ausgesuchter Werte aus R.

Kernschätzung PPC Die folgende Prozedur **PPC** von PISINGER (1995) stellt eine Alternative zu BZC dar. Sie legt ihr Schwergewicht auf ein *verbessertes Vorsortieren der Daten.*

Vorgegeben sei eine Höchstbreite $\delta \geq 2$ des Schätzkerns. Ergebnis der Prozedur ist die gewünschte Zerlegung von N. Die Indizes f, g, i, j beschreiben *Stellen* zwischen 1 und n, die (mit Daten der) items der Aufgabe belegt sind. Mit $[f, g]$ beschreiben wir den Bereich der Stellen zwischen f und g im Bereich N.

Im Original eine rekursive Prozedur beschreiben wir zum besseren Vergleich zu der Prozedur BZC eine *iterative Version* von PPC.

Prozedur PPC (Pisinger Partsort Core)

input: (KP_{01}), δ, η

output: H, C, L, $\bar{b}$

setze $f = 1$ und $g = n$ sowie $l = 0$ und $\bar{a} = 0$.

solange $(g - f) \geq \delta$ und $l \leq \eta$ führe aus

setze $m := \lfloor (f + g)/2 \rfloor$ und tausche die Inhalte der Stellen f, m, g so, dass die Effizienzen zu diesen drei Stellen abfallend sind.

setze $i := f$; $j := g$; $\hat{a} := \bar{a}$; .

wiederhole

wiederhole $\hat{a} := \hat{a} + a_i$; $i := i + 1$; **bis** die Effizienz der Stelle i kleiner gleich der der Stelle m ist;

wiederhole $j := j - 1$; **bis** die Effizienz der Stelle j größer gleich der der Stelle m ist;

ist $(i < j)$ **dann** vertausche die Inhalte von i und j;

bis $i > j$

ist $(\hat{a} > b)$ **dann** setze $L := L \cup [i, g]\,;\ g := i - 1;$

sonst setze $H := H \cup [f, i-1]\,;\ f := i;\ \bar{a} := \hat{a};$

setze $l = l + 1$

setze $C = [f, g]$ und $\bar{b} = b - \bar{a}$. Sortiere C nach abfallenden Effizienzen.

Bemerkung: Es ist sinnvoll, sich für jedes H_i der Teilsortierung den zugehörigen Wert $\bar{a}$ als $\bar{a}_i$ zu merken, ggf. kann auch der entsprechende Nutzen als $\bar{c}_i$ notiert werden.

Beispiel Zum besseren Verständnis betrachten wir unser unsortiertes Beispiel:

j	1	2	3	4	5	6	7
c_j	39	5	37	70	10	20	7
a_j	20	3	19	31	6	10	4
e_j	1,950	1,667	1,947	2,258	1,667	2,000	1,750

mit $n = 7$ und $b = 50$. Wir setze $\delta = 3$ und $\eta = 5$ fest. Setze $f = 1,\ g = n,\ l = 0,\ \bar{a} = 0$.

Zunächst gilt $g - f = 7 - 1 = 6 > 3 = \delta$. Daher startet die Solange-Schleife mit $m := \lfloor (f+g)/2 \rfloor = 4$ und wir sortieren die Effizienzen e_1, e_4, e_7 um: $e_4 > e_1 > e_7$ mit $e_m = 1,95$. Setze $i = 1,\ j = 7,\ \hat{a} = 0$. Damit gilt die Reihenfolge

$$(4, 2, 3, 1, 5, 6, 7)$$

Nun gilt zunächst für $i = 1 : e_4 > 1,95$, daher setze $\hat{a} = 31$ und $\hat{c} = 70$, $i = 2$. Es gilt dann $e_2 = 1,667 < 1,95$, daher bricht die zweite Wiederhole-Schleife hier ab.

In der dritten Wiederhole-Schleife wird nur $j = 7$ durchgewinkt, die Schleife stoppt mit $j = 6$.

Wegen $i < j$ werden nun die Inhalte der Stellen 2 und 6 getauscht und es gilt die Reihenfolge

$$(4, 6, 3, 1, 5, 2, 7)$$

Nun starten die beiden Wiederhole-Schleifen wegen $i = 2 < 6 = j$ erneut:

Die 2. Schleife lässt 6 passieren und endet bei $i = 3$, es gilt dann $\hat{a} = 31 + 10 = 41$ und $\hat{c} = 70 + 20 = 90$.

Die 3. Schleife lässt 2 und 5 passieren und endet bei $j = 4$.

Wegen $i < j$ werden nun die Inhalte der Stellen 3 und 4 getauscht und es gilt die Reihenfolge

$$(4, 6, 1, 3, 5, 2, 7)$$

Nun starten die beiden Wiederhole-Schleifen wegen $i = 3 < 4 = j$ erneut:

Die 2. Schleife lässt 1 passieren und endet bei $i = 4$, es gilt dann $\hat{a} = 41 + 20 = 61$ und $\hat{c} = 90 + 39 = 129$.

Die 3. Schleife lässt 3 passieren und endet bei $j = 3$.

Wegen $j < i$ endet die erste Wiederhole-Schleife und es gilt weiterhin die Reihenfolge

$$(4, 6, 1, 3, 5, 2, 7)$$

Wegen $\hat{a} = 61 > 50 = b$ liegt der kritische Index im Bereich $[f, j] = [1, 3]$ und wir setzen $L = L \cup [i, g] = [4, 7] = (3, 5, 2, 7)$. Setze $g = i - 1 = 3$, $l = 0 + 1 = 1$ und starte erneut in der solange-Schleife. Da aber $g - f = 3 - 1 \leq 3$, wird kein Durchlauf mehr durchgeführt.

Der Schätz-Kern ist gefunden. Die Zerlegung von N lautet $N = C \cup L_1$ mit $C = (4, 6, 1)$ und $L_1 = (3, 5, 2, 7)$ sowie $\bar{b} = b - \bar{a} = b = 50$. Auch hier könnte der Schätzkern nachbearbeitet werden.

Im Nachgang wird C vollsortiert. C ist hier aber bereits sortiert und es kann aus C der kritische Index ausgehend von $\bar{b} = 50$ ermittelt werden. Die Dantziglösung lautet $x^T = (0, 0, 0, 1, 0, 1, 0)$ mit Gewicht 41 und Nutzen 90.

Berechnung der Schranken

Zunächst einmal steht uns die obige Prozedur LUDSu zur Verfügung. Man kann auch anstelle von LUDS auch LUMS auf den unsortierten Fall übertragen, da es bei der Müller-Merbachschranke nicht auf die Anordnung der Indizes ankommt, es muss nur der kritische Index identifiziert sein. Überhaupt kann man prüfen, welche der von uns explizit angesprochenen LU Prozeduren für vorsortierte Aufgaben problemlos auf den unsortierten Fall übertragen werden können, weil die interne Anordnung der Indizes außerhalb des sortierten Bereich $[r, t]$ überhaupt nicht verwendet wird. Neben LUMS trifft dies zu für die Prozeduren, die von vorne herein über einen Teilbereich $[r, t]$ definiert wurden wie LUU6, LUHS. Diese Prozduren können nahezu unverändert angewendet werden. Zu beachten ist dabei, dass bei Anwendung von Satz 2.1 oder Korollar 2.4 Effizienzen zur Schrankenberechnung nur aus dem geordneten Bereich C verwendet werden.

Reduzierung

Auch bei der Reduzierung kann auf die Prozeduren des sortierten Falls zurückgegriffen werden, wenn bei der Schrankenberechnung beachtet wird, dass nur der Bereich C wirklich sortiert ist. Neu berechnete kritische Indizes müssen also in C liegen, wenn sie zur Schrankenberechnung herangezogen werden sollen und Nutzendichten, die zur Schrankenberechnung nach Satz 2.1 verwendet werden, müssen ebenfalls zum Bereich C gehören.

Konkret kann man wie bereits oben besprochen vorgehen und jeweils die Nutzendichte des *kritischen Index* verwenden. Andererseits kann man für die Teilaufgaben mit einem fixierten Index je einen neuen kritischen Index innerhalb von $[r, t]$ suchen bzw, wenn dieser nicht existiert, jeweils die Nutzendichte von r oder t verwenden.

3.2 Das Verfahren MT2

Martello und Toth (1988) haben ein Verfahren gemäß der oben genannten Strategie vorgeschlagen, das sich als sehr erfolgreich erwiesen hat. Es soll im Folgenden vorgestellt werden. Wir besprechen nacheinander die Umsetzung der vier Teilabschnitte der Vorgehensweise. Das resultierende Verfahren nennen wir **Prozedur MT2 (Martello Toth 2).** Es besteht aus

1. der Teilsortierungs-Prozedur MTC, die eine Variation von BZC darstellt. Bei dieser soll der Schätzkern in einem vorgegebenen Korridor liegen, damit der eigentliche Kern eine möglichst große Chance hat, im Schätz-Kern enthalten zu sein. Ist der tatsächlich berechnete Schätzkern dann zu groß, soll die gesamte Aufgabe sortiert werden.

2. der LU Prozedur LUMT, die die auf den Schätzkern begrenzte Aufgabenstellung exakt löst und gleichzeitig eine möglichst scharfe obere Schranke für (KP_{01}) berechnet.

3. der Reduzierungsprozedur MTR+, die die uns bekannte Reduzierungs-Prozedur MTR auf den unsortierten Fall ausdehnt. Die geschieht dadurch, dass Schranken nach MTR berechnet werden, soweit die Sortierung dies zulässt und ansonsten Schranken nach Satz 2.1 berechnet werden.

4. die Prozedur MT1 nach Sortierung auf die Restaufgabe angewendet wird.

Wir besprechen diese vier Teile der Lösungsstrategie naacheinander im Detail.

3.2.1 Kernschätzung

Martello und Toth berechnen in einer Prozedur einen Schätz-Kern, dessen Größe in einem *bestimmten Korridor* liegt. Die Prozedur wird bei Erfolg oder nach einer vorgegebenen Schrittanzahl η abgebrochen. Zielsetzung ist es, einen Schätzkern C zu ermitteln, dessen Größe einer vorgewählten Richtgröße $\delta > 0$ ungefähr entspricht. Die Prozedur versucht dies zu realisieren, indem immer wieder der Bereich H der Variablen, die links neben dem Kern liegen und der Bereich L der Variablen, die rechts neben dem Kern liegen, verändert wird.

Wir verwenden die Abkürzung $M3\,(S)$ für den Median aus *dem ersten, dem mittleren und dem letzten Nutzen-Dichte- Quotient* der (angeordneten) Index-Menge S. Ist dabei $|S| = 2$, nehme man den kleineren der beiden Quotienten, bei $|S| = 1$, den einzigen vorhandenen Quotienten. $\beta \in [0,1]$ sind Pufferparameter, l der Laufindex der Iterationen.

Prozedur MTC

Input: (KP_{01}), $\alpha, \beta \in [0,1]$, $\eta, \delta \in \mathbf{N}$

Output: H, C, $L, \bar{b}$

setze zunächst $H = L = \varnothing$, $C = \{1, ..., n\}$, $\bar{b} = b$, $l = 0$, $\lambda = M3\,(C)$

solange $|C| > (1+\beta)\,\delta$ und $l < \eta$ führe aus:

setze $G = \left\{ j \in C : \frac{c_j}{a_j} > \lambda \right\}$, $S = \left\{ j \in C : \frac{c_j}{a_j} < \lambda \right\}$, $E = \left\{ j \in C : \frac{c_j}{a_j} = \lambda \right\}$

sowie $b' = \sum_{j \in G} a_j$ und $b'' = b' + \sum_{j \in E} a_j$

ist $b' \leq \bar{b} < b''$, so führe aus:

ist $|E| \geq (1-\alpha)\,\delta$, so führe aus: % `angestrebte Situation erreicht`

Es sei $E := \{e_1, ..., e_q\}$

setze $\sigma = \min \left\{ j : \sum_{i=1}^{j} a_{e_i} > \bar{b} - b' \right\}$, $s = e_\sigma$ und

$C = \{e_r, ..., e_t\}$ mit frei gewählten r, t so, dass $t - r + 1 \approx \delta$ und $\frac{t+r}{2} \approx s$

ferner setze $L = L \cup S \cup \{e_{t+1}, ..., e_q\}$ und $H = H \cup G \cup \{e_1, ..., e_{r-1}\}$

sonst setze $\lambda = M3\,(S)$, falls $|G \cup E| < \delta$ bzw $\lambda = M3\,(G)$ andernfalls.

sonst führe aus: % `es gilt` $b'' \leq \bar{b}$ `oder` $\bar{b} < b'$

ist $\bar{b} < b'$ so führe aus: % k `liegt in` G

ist $|G| < (1-\alpha)\,\delta$, so setze $\lambda = M3\,(S)$, sonst setze

$L = L \cup S \cup E$, $C = G$, $\lambda = M3\,(C)$ % `hier wird` L `erweitert`

sonst führe aus: % `es gilt` $b'' \leq \bar{b}$ `%` k `liegt in` S

ist $|S| < (1-\alpha)\,\delta$ so setze $\lambda = M3\,(G)$, sonst setze

$H = H \cup G \cup E,\ C = S,\ \bar{b} = \bar{b} - b'',\ \lambda = M3\,(C)$ % `hier wird` H `erweitert`

setze $l := l + 1$

Beispiel Wir verschaffen uns einen Eindruck von der Arbeitsweise von MTC durch Anwendung auf unser ungeordnetes Beispiel. Dieses lautete:

j	1	2	3	4	5	6	7
c_j	39	5	37	70	10	20	7
a_j	20	3	19	31	6	10	4
ζ_j	1,950	1,667	1,947	2,258	1,667	2,000	1,750

mit $n = 7$ und $b = 50$. Wir wählen zur Ausführung von MTC die Steuerungsparameter wie folgt: $\alpha = 0.4,\ \beta = 0.5$. Es soll höchstens $\eta = 5$ Iterationen geben und der Kern soll ungefähr $\delta = 3$ items umfassen. Wir werden wie schon bei der Anwendung von BZC einmal erreichte Sortierungsstrukturen mitschleppen, um eine möglichst differenzierte Partition von N zu erhalten. Dies geschieht durch explizite Angabe von Vereinigungen.

Zunächst setzen wir $H = L = \varnothing$, $C = \{1, ..., n\}$, $\bar{b} = b = 50$, $l = 0$ und wählen λ als mittleren Wert von $\zeta_1, \zeta_4, \zeta_7$, also $\lambda = \zeta_1 = 1,950$. Da $|C| = 7 > 1,5 * 3$, führen wir einmal die Solange-Schleife durch. Es ergibt sich:

$$G = \{4, 6\}\,,\ E = \{1\}\,,\ S = \{2, 3, 5, 7\}$$

sowie $b' = 41$ sowie $b'' = 61$. Damit gilt $b' \leq \bar{b} < b''$, aber nicht $|E| = 1 \geq 0.6 * 3 = (1-\alpha)\,\delta$. Also setzen wir $\lambda = M3\,(G) = \zeta_6 = 2,000$. setze $l = 1$.

Ein erneutes Springen in die Solange-Schleife liefert bei gleichem C :

$$G = \{4\}\,,\ E = \{6\}\,,\ S = \{1\} \cup \{2, 3, 5, 7\}$$

sowie $b' = 31$ sowie $b'' = 41$. Nun gilt nicht mehr $b' \leq \bar{b} < b''$, sondern $b'' = 41 \leq 50 = \bar{b}$. Da nicht $|S| < (1-\alpha)\,\delta$ gilt, setzen wir $H = H \cup G \cup E = \{4\} \cup \{6\}$, $C = S = \{1\} \cup \{2, 3, 5, 7\}$, $L = \varnothing,\ \bar{b} = \bar{b} - b'' = 50 - 41 = 9,\ \lambda = M3\,(C) = \zeta_3 = 1,947$ und damit wird die Solange-Schleife erneut aufgerufen, bei $l = 2$.

Wiederum lässt $|C| = 5 > 1,5 * 3 = 4,5$ die Durchführung der Solange-Schleife zu. Es gilt

$$G = \{1\}\,,\ E = \{3\}\,,\ S = \{2, 5, 7\}$$

sowie $b' = 20$ sowie $b'' = 39$. Nun gilt $\bar{b} = 9 < 20 = b'$ und wir setzen, weil $|G| = 1 < 0.5 * 3\,(1-\alpha)\,\delta$ erneut $\lambda = M3\,(S) = 1,667$. Setze $l = 3$. Dies liefert bei gleichem C :

$$G = \{1\} \cup \{3\} \cup \{7\}\,,\ E = \{2\} \cup \{5\}\,,\ S = \{\}$$

sowie $b' = 43$ sowie $b'' = 52$. Nun gilt $\bar{b} = 9 < b'$ und nicht $|G| = 3 < (1-\alpha)\,\delta$, also setze $L = L \cup S \cup E = \{2\} \cup \{5\}$, $C = G = \{1\} \cup \{3\} \cup \{7\}\,,\ \lambda = M3\,(C) = 1,947$, $l = 4$ und es wird erneut versucht, einen Solange-Schritt durchzuführen.

Allerdings ist nicht $|C| = 3 > 4.5 = (1+\beta)\,\delta$, sodass die Schleife nicht mehr durchlaufen wird.

Als Ergebnis ergibt sich die letzte Festsetzung von H, L, C

$$H = \{4\} \cup \{6\}\,,\ C = \{1\} \cup \{3\} \cup \{7\}\,,\ L = \{2\} \cup \{5\}$$

(was im Übrigen schon einer Vollsortierung entspricht.)

Das break item $k = 1$ liegt am linken Rand von C. Deshalb nehmen wir den linken Nachbarn vom C noch mit in den Kern auf, so dass sich nun

$$H = \{4\}\,,\ C = \{6\} \cup \{1\} \cup \{3\} \cup \{7\}\,,\ L = \{2\} \cup \{5\}$$

ergibt.

3.2.2 Berechnung der Schranken

Martello und Toth schlagen vor, auf eine Ausnutzung des Schätz-Kerns C zu verzichten, wenn dieser zu groß ausgefallen ist. Als empirisch ermittelte Ober-Grenze geben sie dabei den Wert $(1-\alpha) \cdot n$ an. Ist diese Grenze überschritten, so sollen die gesamte Aufgabe sortiert und danach die Prozeduren MTR und MT1 angewendet werden.

Ist der Schätz-Kern nicht zu groß, so wendet man nach der Kernschätzung und der Sortierung des Schätz-Kerns die Methode LUBB in Form der folgenden Prozedur LUMT (auf der Grundlage der Prozedur MT1) zur Berechnung einer zulässigen Lösung mit Zielwert z sowie einer oberen Schranke U an. Dabei wird das Intervall $[r,t]$, auf dem die Prozedur basiert, wegen der zu berechnenden Schranke U_2 auf beiden Seiten *echt innerhalb* des *sortierten Schätz-Kerns* gewählt. Es ist darauf zu achten, dass der kritische Index k in $[r,t]$ liegt, sonst muss C erweitert werden.

Die folgende Prozedur wandelt MT1, angewandt auf die auf $[r,t]$ eingeschränkte Aufgabe, in eine LU Prozedur, die die Optimallösung der auf $[r,t]$ eingeschränkten Aufgabe als untere Schranke (einschließlich des entsprechenden Vornutzens) berechnet und nach der Methode LUBB eine obere Schranke. Dabei wird völlig analog zu Umwandlung von HSV in LUHS vorgegangen.

Es werde der Einfacheit halber unterstellt, dass die Indizes nach der Teilsortierung *in natürlicher Reihenfolge* vorliegen, also die Sortierung physikalisch erfolgte. Es sei wieder darauf hingewiesen, dass ein Ersatz der verwendeten U_2 Schranke durch die U_{23} Schranke zu erheblicher Effizienzsteigerung führen kann. Details seien dem Leser überlassen.

Prozedur LUMT (LU Martello Toth)

Input: $\left(KP_{01}^{+}\right)$, zulässige Lösung $\bar{x}$ mit Zielwert $\bar{z}$, obere Schranke U. Ferner Zahlen $r, t \in \mathbf{N}$ mit $1 \leq r \leq t \leq n$.

Output: Zulässige Lösung x mit Zielfunktionswert z und obere Schranke U für (KP_{01})

1. **Start:** setze $z = \bar{z},\ \hat{z} = \sum_{k=1}^{r-1} c_k, \bar{a} = \sum_{k=1}^{r-1} a_k,\ \hat{b} = b - \bar{a},\ \bar{c} = \sum_{k=1}^{r-1} c_k\ j = r,$ $\bar{u} = 0,\ f = 0$. Setze $r_k = k$ für $k = 1, ..., n$. setze $x_k = 1$ für $k = 1, ..., r-1$ und $x_k = 0$ für $k = t+1, ..., n$. Setze $\hat{x}_k = 0$ für $k = r, ..., t+1$

   für $i = r$ bis t setze $aa_j = a_j$, $cc_j = c_j$. Ferner setze $a_0 = 0$, $c_0 = \infty, a_{n+1} = \infty,\ c_{n+1} = 0$ sowie $aa_{r-1} = 0,\ cc_{r-1} = \infty,\ aa_{t+1} = aa_{t+2} = \infty,\ cc_{t+1} = cc_{t+2} = 0$. Berechne für $i = r-1, ..., t+1$ die kumulierten Größen $\bar{a}_i = \bar{a} + \sum_{j=r}^{i} aa_j$ und $\bar{c}_i = \bar{c} + \sum_{j=r}^{i} cc_j$

2. **Vorwärtsschritt 1**

   solange $aa_j > \hat{b}$ gilt, führe aus:

   ist $z \geq \hat{z} + \left\lfloor \hat{b} \frac{cc_{j+1}}{aa_{j+1}} \right\rfloor$, so gehe zu 5.,

   andernfalls setze $\bar{u} = \max\left\{\bar{u}, \left\lfloor \hat{z} + c_j + \left(\hat{b} - a_j\right) \frac{c_{r-1}}{a_{r-1}} \right\rfloor\right\}$ und $j = j+1$

   suche $r = \min\left\{i : \bar{a}_i > \hat{b} + \bar{a}_{j-1}\right\}$, startend bei $i = r_j$.

   setze $c' = \bar{c}_{s-1} - \bar{c}_{j-1}$ und $a' = \bar{a}_{s-1} - \bar{a}_{j-1}$

   ist $r \leq n$, so berechne $u := \max\left\{\left\lfloor \left(\hat{b} - a'\right) \frac{cc_{s+1}}{aa_{s+1}} \right\rfloor, \left\lfloor cc_q - \left(aa_q - \left(\hat{b} - a'\right) \frac{cc_{s-1}}{aa_{s-1}}\right) \right\rfloor\right\}$

   andernfalls setze $u = 0$

   ist $z \geq \hat{z} + c' + u$, setze $\bar{u} = \max\left\{\bar{u}, \left\lfloor \hat{z} + \hat{b} \frac{c_j}{a_j} \right\rfloor\right\}$ und gehe zu 5.

   ist $f = 0$ und $U > \hat{z} + c' + u$, setze $U = \hat{z} + c' + u$ sowie $f = 1$ und $l = l + 1$

   ist $u = 0$, gehe zu 4.

3. **Sicherung**

   setze $\hat{b} = \hat{b} - a'$ und $\hat{z} = \hat{z} + c'$

   für $k = j$ bis $s - 1$ setze $\hat{x}_k = 1$ und $r_k = r$.

   setze $j = s$

   ist $\hat{b} \geq w_s$, so gehe zu 2.

   ist $z \geq \hat{z}$, so gehe zu 5.

   setze $c' = 0$

4. **Aktualisierung**

   setze $z = \hat{z} + c'$

   für $k = 1$ bis $j - 1$ setze $x_k = \hat{x}_k$

   für $k = j$ bis $s - 1$ setze $x_k = 1$

   für $k = s$ bis n setze $x_k = 0$

   ist $z = U$, **stopp**

5. **Rückwärtsschritt**

   setze $\bar{u} = \max\left\{\bar{u}, \left\lfloor \hat{z} + \hat{b}\frac{c_j}{a_j} \right\rfloor\right\}$

   finde $i = \max\{r \leq k < j : \hat{x}_k = 1\}$. Gibt es kein solches i, gehe zu 7.

   setze $\hat{b} := \hat{b} + aa_i, \quad \hat{z} := \hat{z} - cc_i, \quad \hat{x}_i := 0, \quad j := i + 1$

   ist $i = l$, setze $f = 0$

   ist $\hat{b} - aa_i \geq w_i$, gehe zu 2.

   setze $j = i$, $h = i$

6. **Vorwärtsschritt 2**

   ist $h > j$, so setze $\bar{u} = \max\left\{\bar{u}, \left\lfloor \hat{z} + c_h + \left(\hat{b} - a_h\right)\frac{c_{r-1}}{a_{r-1}} \right\rfloor\right\}$

   setze $h = h + 1$

   ist $z \geq \hat{z} + \left\lfloor \hat{b}\frac{cc_h}{aa_h} \right\rfloor$, so setze $j = h$ und gehe zu 5.

   ist $aa_h = aa_i$, so gehe zu 6.

   ist $aa_h > aa_i$, so führe aus:

   ist $aa_h > \hat{b}$ oder $z \geq \hat{z} + cc_h$, so gehe zu 6.

   setze $z = \hat{z} + cc_h$

   für $k = 1$ bis n setze $x_k = \hat{x}_k$

setze $x_h = 1$

ist $z = U$, **stopp**

setze $i = h$ und gehe zu 6.

andernfalls führe aus:

ist $\hat{b} - aa_h < w_h$, so gehe zu 6.

setze $\hat{b} = \hat{b} - aa_h,\ \hat{z} = \hat{z} + cc_h,\ \hat{x}_h = 1$ und $j = h + 1$

gehe zu 2.

7. **Gierige Erweiterung**

ist $\bar{u} < U$, setze $U = \bar{u}$.

setze $\hat{b} = b - \bar{a} - \sum_{j=r}^{t} x_i a a_i$

Für $i = t + 1$ bis n führe aus

ist $a_i \leq \hat{b}$, so setze $x_i = 1$, $\hat{b} = \hat{b} - a_i$, $z = z + c_i$, sonst $x_i = 0$.

Erläuterung zum Verfahren Im Prinzip besteht das Verfahren aus der Anwendung der Prozedur MT1 auf den Bereich $[r, t] \subset C$. Weil dabei der innere Bereich $[r, t]$ (genauso wie der gesamte Bereich der Variablen N) trivial erweitert wird, müssen die Daten im inneren Bereich umbenannt werden (hier durch Verdoppelung der Buchstaben), da sonst die Daten von $r - 1$ und $t + 1$ überschrieben würden, welche aber separat gebraucht werden. Dies alles geschieht im Baustein Start.

Ansonsten muss für jedes Blatt des entstehenden Entscheidungsbaums eine obere Schranke berechnet werden, deren Maximum letztlich die globale obere Schranke bildet. Ein Blatt des Entscheidungsbaums entsteht, wenn im Verfahren ein Rückwärtsschritt eingeleitet wird. Daher wird hier wir in LUHS die Schranke

$$\bar{u} = \max \left\{ \bar{u}, \left\lfloor \hat{z} + \hat{b} \frac{c_j}{a_j} \right\rfloor \right\}$$

eingefügt, wobei j den aktuellen Index bezeichnet.

Gleichzeitig muss wie bei LUHS darauf geachtet werden, dass der Entscheidungsbaum vollständig binär verzweigt. Ist dies nicht der Fall, muss eine entsprechende Verzweigung ergänzt werden. Dies geschieht im Verfahren in den beiden Vorwärtsschritten über die schon in LUHS verwendete SChrankenberechnung

$$\bar{u} = \max \left\{ \bar{u}, \left\lfloor \hat{z} + c_j + \left(\hat{b} - a_j \right) \frac{c_{r-1}}{a_{r-1}} \right\rfloor \right\}$$

wobei j wieder den laufenden Index angibt. Im Baustein 6. ist allerdings darauf zu achten, dass beim ersten Anlaufen dieses Bausteins diese Schranke nicht berechnet wird, das an dieser Stelle die binäre Verzweigung schon besteht. Wir schleißen den ersten Anlauf über die Besingung $h > j$ aus, wobei hier h den aktuellen Index bezeichnet und j ein Bezugsindex ist.

Schließlich wird zum Schluss im Baustein 7. versucht, die berechnete zulässige Lösung noch gierig zu erweitern.

Beispiel Dieses lautete:

j	1	2	3	4	5	6	7
c_j	39	5	37	70	10	20	7
a_j	20	3	19	31	6	10	4
ζ_j	1,950	1,667	1,947	2,258	1,667	2,000	1,750

mit $n = 7$ und $b = 50$.

Der ermittelte Schätz-Kern $\{6, 1, 3, 7\}$ ist bereits sortiert, die Grenze $(1 - \alpha) \cdot n = 0,6 \cdot 7 = 4,2$ nicht überschritten. Wir können daher direkt die Prozedur LUMT anwenden und lösen die Aufgabe $(KP[3, 4]) = (KP\{1, 3\})$, bei der nur die Variablen x_1 und x_3 variieren und eine Restkapazität von $\bar{b} = 9$ vorliegt. Die Menge $\{1, 3\}$ liegt auf beiden Seiten echt innerhalb von $C = \{6, 1, 3, 7\}$, der kritische Index liegt bei item 1.

Anwendung der Prozedur LUMT auf $(KP\{1, 3\})$ Die Restkapazität der Aufgabe beträgt $\hat{b} = 50 - 31 - 10 = 9$, der Vornutzen $\hat{z} = 70 + 20 = 90$. Es liegt somit eine zulässige Lösung $\bar{x}$ mit diesen Daten vor. Die zugehörige Dantzigschranke für (KP_{01}) lautet:

$U = c_4 + c_6 + \lfloor \zeta_1(b - a_4 - a_6) \rfloor = 70 + 20 + \left\lfloor \frac{39}{20} \cdot (50 - 31 - 10) \right\rfloor = 107$

Untersuchen wir zunächst, ob $aa_3 = a_1$ in den Restrucksack passt. Dies ist wegen $20 > 9$ nicht der Fall. Ferner gilt $z = \bar{z} = 90 < 90 + \left\lfloor 9 \cdot \frac{37}{19} \right\rfloor = \hat{z} + \left\lfloor \hat{b} \cdot \frac{cc_4}{aa_4} \right\rfloor$, daher berechne

$\bar{u} = \hat{z} + c_1 + \left\lfloor \zeta_6(\hat{b} - a_1) \right\rfloor = 90 + 39 + \left\lfloor \frac{20}{10} \cdot (9 - 20) \right\rfloor = 107$

Nun untersuchen wir, ab a_3 in den Restrucksack passt. Dies ist wegen $a_3 = 19$ ebenfalls nicht der Fall , daher berechne $\bar{u} = \hat{z} + c_3 + \left\lfloor \zeta_6(\hat{b} - a_3) \right\rfloor = 90 + 37 + \left\lfloor \frac{20}{10} \cdot (9 - 19) \right\rfloor = 107$. Ferner gilt $z = \bar{z} = 90 \geq 90 + \lfloor 9 \cdot 0 \rfloor = \hat{z} + \left\lfloor \hat{b} \cdot \frac{cc_5}{aa_5} \right\rfloor$, und $\bar{u} = \max \left\{ \bar{u},\ \hat{z} + \left\lfloor \zeta_3 \hat{b} \right\rfloor \right\} = \max \left\{ 107,\ 90 + \left\lfloor \frac{37}{19} \cdot 9 \right\rfloor \right\} = 107$

Nun finden wir im Baustein 5. kein i mehr, sodass wir zu Baustein 7. springen. Hier bleibt es bei $U = 107$. Die gierige Lösung ergibt sich in der vorliegenden Reihenfolge der Variablen zu $x^T = (0, 1, 0, 1, 0, 1, 1)$ mit $z = 90 + 12 = 102$.

3.2.3 Reduzierung

Die Reduzierung geschieht nun im Prinzip mit der Reduzierungsprozedur MTR, wir erweitern diese Prozedur auf den nun vorliegenden Fall. Es sei wieder k der bekannte kritische Index der Aufgabe.

Setzt man $x_j = 0$ für ein $j \leq k$, so liegt der neue kritische Index k' potentiell rechts von k. Liegt dieser in C, so berechnet man die Schranke U_2 für diese Aufgabe, sonst die Schranke nach Satz 2.1 mit Hilfe des am meisten rechts liegenden Nutzen-Gewichtsquotienten in C.

Setzt man $x_j = 1$ für ein $j \geq k$, so liegt der neue kritische Index k' potentiell links von k. Liegt dieser in C, so berechnet man die Schranke U_2 für diese Aufgabe, sonst die Schranke nach Satz 2.1 mit Hilfe des am meisten links liegenden Nutzen-Gewichtsquotienten in C.

Wir formulieren die oben beschriebene Prozedur so, dass sie auch außerhalb von MT2 angewendet werden kann. Dabei unterstellen wir der Einfachheit halber wieder, dass die Indizes in der natürlichen Reihenfolge bereits nach abfallenden Effizienzen teilsortiert sind. Es sei wieder darauf hingewiesen, dass ein Ersatz der verwendeten U_2 Schranke durch die U_{23} Schranke zu erheblicher Effizienzsteigerung führen kann. Die nähere Ausgestaltung sei dem Leser überlassen.

Prozedur MTR+

Input: (KP_{01}), kritischer Index k, eine zulässige Lösung mit Zielwert z, eine obere Schranke U, eine Teilsortierung von N in $N = J1 \cup JC \cup J0$, von denen JC das break item enthält. Es sei $JC = \{x_r, ..., x_t\}$ und nach abfallenden Nutzen-Gewichts-Quotienten sortiert, $J1 = \{x_1, ..., x_{r-1}\}$ und $J0 = \{x_{t+1}, ..., x_n\}$.

Output: Mengen JF_1 und JF_0 von Indizes, die bei der weiteren Suche auf 1 bzw 0 fixiert werden können, untere Schranke Z, obere Schranke U.

1. **Start:** Setze $JF_0, JF_1 := \emptyset$ und $\bar{U}' = \bar{U}'' = 0$. Berechne $\bar{c} = \sum_{j \in J1} c_j$, $\bar{a} = \sum_{j \in J1} a_j$ und $\bar{b} = b - \bar{a}$ sowie für $i = r, ...t$ die kumulierten Größen $\bar{c}_i = \sum_{j=r}^{i} c_j$ und $\bar{a}_i = \sum_{j=r}^{i} a_j$. Setze ferner $\bar{a}_{r-1} = \bar{c}_{r-1} = 0$. Setze $i_0 = 0$.

2. **Gierige Lösung**

   Setze $U := \min\left(U, \bar{c} + \bar{c}_{k-1} + \frac{c_k}{a_k}\left(\bar{b} - \bar{a}_{k-1}\right)\right)$ und berechne die gierige Lösung $\bar{x}$ mit Zielwert $\bar{z}$ zu (KP_{01}) gemäß der vorliegenden Reihenfolge der Indizes.

   Ist $\bar{z} > z$, so setze $x = \bar{x}$ und $Z = \bar{z}$

   Gilt $U = Z$, **stopp**, (x ist optimal).

3. **Lokale Schranken für** (KP_i^0), $i \leq k$

   Für $i \in J1$ oder $i = r, ..., k$ führe aus:

   ist $\bar{b} + a_i < \bar{a}_t$, so führe aus:

   Finde den Index $\bar{k}$ mit $\bar{a}_{\bar{k}-1} \leq \bar{b} + a_i < \bar{a}_{\bar{k}}$

   Setze $\bar{b}_i = b + a_i - \bar{a}_{\bar{k}-1}$, $z^i = \bar{c} + \bar{c}_{\bar{k}-1} - c_i$

   Ist $z^i > Z$, setze $Z = z^i$ und $i_0 = i,\ k_0 = \bar{k}$, $\hat{b} = \bar{b}_i$.

   Falls $r < \bar{k} < t$, setze $U_i^0 = \bar{c} + \bar{c}_{\bar{k}-1} - c_i + \max\left\{\left\lfloor \frac{c_{\bar{k}+1}}{a_{\bar{k}+1}}\hat{b}\right\rfloor, \left\lfloor c_{\bar{k}} + \left(\hat{b} - a_{\bar{k}}\right)\frac{c_{\bar{k}-1}}{a_{\bar{k}-1}}\right\rfloor\right\}$,

   andernfalls setze $U_i^0 = \bar{c} + \bar{c}_{\bar{k}-1} + c_i + \left\lfloor \hat{b}\frac{c_{\bar{k}}}{a_{\bar{k}}}\right\rfloor$.

   sonst setze $U_i^0 = \bar{c} + \bar{c}_t - c_i + \left\lfloor \left(\bar{b} + a_i - \bar{a}_t\right)\frac{c_t}{a_t}\right\rfloor$ und $z^i = \bar{c} + \bar{c}_t - c_i$

   Ist dabei $z^i > Z$, setze $Z = z^i$ und $i_0 = i,\ k_0 = t$, $\hat{b} = \bar{b}_i$.

   Gilt $U = Z$, **stopp**, (x ist optimal).

   Gilt $U_i^0 > \bar{U}'$ und $i \neq k$, so setze $\bar{U}' = U_i^0$.

4. **Lokale Schranken für** (KP_i^1), $i \geq k$

   Für $i = k, ..., t$ oder $i \in J0$ führe aus:

   ist $\bar{b} - a_i \geq 0$, so führe aus:

   Finde den Index $\bar{k}$ mit $\bar{a}_{\bar{k}-1} \leq \bar{b} - a_i < \bar{a}_{\bar{k}}$

   Setze $\bar{b}_i = b - a_i - \bar{a}_{\bar{k}-1}$, $z^i = \bar{c} + \bar{c}_{\bar{k}-1} + c_i$

   Ist $z^i > Z$, setze $Z = z^i$ und $i_0 = i,\ k_0 = \bar{k}$, $\hat{b} = \bar{b}_i$.

   Falls $r < \bar{k} < t$, setze $U_i^1 = \bar{c} + \bar{c}_{\bar{k}-1} + c_i + \max\left\{\left\lfloor \frac{c_{\bar{k}+1}}{a_{\bar{k}+1}}\hat{b}\right\rfloor, \left\lfloor c_{\bar{k}} + \left(\hat{b} - a_{\bar{k}}\right)\frac{c_{\bar{k}-1}}{a_{\bar{k}-1}}\right\rfloor\right\}$,

   andernfalls setze $U_i^1 = \bar{c} + \bar{c}_{\bar{k}-1} + c_i + \left\lfloor \hat{b}\frac{c_{\bar{k}}}{a_{\bar{k}}}\right\rfloor$.

   sonst setze $U_i^1 = \bar{c} + c_i + \left\lfloor \left(\bar{b} - a_i\right)\frac{c_r}{a_r}\right\rfloor$

   Gilt $U = Z$, **stopp**, (x ist optimal)

   Gilt $\bar{x}_i = 0,\ i \neq k$ und $U_i^1 > \bar{U}''$, so setze $\bar{U}'' = U_i^1$

5. **Globale Schranken**

   Setze $\bar{U}' = \min\left\{U_i^1, \bar{U}'\right\}$ und $\bar{U}'' = \min\left\{U_i^0, \bar{U}''\right\}$.

   Setze $U = \min\left\{U, \max\left\{\bar{z}, \bar{U}', \bar{U}''\right\}\right\}$.

   Ist $i_0 > 0$, so führe aus

   Ist $i_0 < k$, so setze $x_j = 1$ für $j = 1, ..., \bar{k} - 1$ und $x_j = 0$ für $j = k, ..., n$ sowie $x_{i_0} = 0$.

   Ist $i_0 > k$, so setze $x_j = 1$ für $j = 1, ..., \bar{k} - 1$ und $x_j = 0$ für $j = k, ..., n$ sowie $x_{i_0} = 1$.

   Ist $i_0 = k$ und $\bar{k} > k$, so setze $x_j = 1$ für $j = 1, ..., \bar{k} - 1$ und $x_j = 0$ für $j = k, ..., n$ sowie $x_{i_0} = 0$.

   Ist $i_0 = k$ und $\bar{k} < k$, so setze $x_j = 1$ für $j = 1, ..., \bar{k} - 1$ und $x_j = 0$ für $j = k, ..., n$ sowie $x_{i_0} = 1$.

   Für $i = k_0, ..., n,\ i \neq i_0$ führe aus

   ist $a_i \leq \hat{b}$, so setze $x_i = 1$, $\hat{b} = \hat{b} - a_i$, $Z = Z + c_i$, sonst $x_i = 0$.

   Gilt $U = Z$, **stopp**, (x ist optimal.)

6. **Reduzierung**

   Für $i = 1, ...k$ führe aus: Gilt $U_i^0 \leq Z$, setze $JF_1 = JF_1 \cup \{i\}$

   Für $i = k, ...n$ führe aus: Gilt $U_i^1 \leq Z$, setze $JF_0 = JF_0 \cup \{i\}$

   Gilt danach $k \in JF_0 \cap JF_1$, **stopp**, (x ist Optimallösung.)

7. **Nachjustierung**

   Berechne die Restkapazität $b = b - \sum_{j \in JF_1} a_j$.

   Ist $b \leq 0$, **stopp**, die Z zugrunde liegende Lösung x Optimallösung.

   Für alle $j \in \{1, ..., n\} \setminus (JF_1 \cup JF_0)$ führe aus:

   ist $a_j > b$, so setze $JF_0 = JF_0 \cup \{j\}$

   ist $a_j = b$, so aktualisiere ggf. die untere Schranke durch Festsetzung von $x_j = 1$.

Beispiel Wenden wir nun diese Prozedur auf unser Beispiel an:

Bekannt sind zur gegebenenAufgabe die untere Schranke $z = 102$ als Zielwert der zulässigen Lösung $x^T = (0, 1, 0, 1, 0, 1, 1)$ und die obere Schranke $U = 107$. Ferner setzen wir folgende Teilsortierung als bekannt voraus

$$J1 = \{4\},\ JC = \{6\} \cup \{1\} \cup \{3\} \cup \{7\},\ J0 = \{2\} \cup \{5\}$$

Dabei ist JC nach abfallenden Nutzen-Gewichts-Quotienten sortiert.

Zum **Start** setzen wir $JF_0, JF_1 := \emptyset$ und berechnen $\bar{c} = \sum_{j \in J1} c_j = 70$, $\bar{a} = \sum_{j \in J1} a_j = 31$ und $\bar{b} = b - \bar{a} = 19$. Ferner berechnen wir zu JC die kumulierten Werte

$$\bar{c}_6 = 20,\ \bar{c}_1 = 59,\ \bar{c}_3 = 96,\ \bar{c}_7 = 103$$

sowie

$$\bar{a}_6 = 10;\ \bar{a}_1 = 30,\ \bar{a}_3 = 49,\ \bar{a}_7 = 53.$$

Gierige Lösung: Der kritische Index ergibt sich zu $k = 1$, ferner ergibt sich die Gierige Lösung zu $\bar{x}^T = (0, 1, 0, 1, 0, 1, 1)$ mit Zielwert $\bar{z} = 102$ und damit $Z = \max\{z, \bar{z}\} = \max\{102, 102\} = 102$. Ferner ergibt sich

$U = \min\left(U, \bar{c} + \bar{c}_{k-1} + \frac{c_k}{a_k}\left(\bar{b} - \bar{a}_{k-1}\right)\right) = \min\left(107, 70 + 20 + \lfloor 1.95 * (19 - 10) \rfloor\right) = 107$. Es ist nicht $U = Z$.

Lokale Schranken für (KP_i^0), $i \leq k$**:** Wir betrachten nun nacheinander die Indizes $i = 4,\ 6,\ 1$.

zu $i = 4$: es ist $\bar{b} + a_i = 19 + 31 = 50 < 53 = \bar{a}_t$, daher bestimmen wir den lokalen kritischen Index $\bar{k}$ über $\bar{a}_{\bar{k}-1} \leq \bar{b} + a_i < \bar{a}_{\bar{k}}$ zu $\bar{k} = 7$. Wir ermitteln als lokale Restkapazität $\hat{b} = \bar{b} + a_i - \bar{a}_{\bar{k}-1} = 19 + 31 - 49 = 1$ und als Aktualisierung der unteren Schranke $Z = \max\{Z, \bar{c} + \bar{c}_{\bar{k}-1} - c_i\} = \max\{Z, 70 + 96 - 70\} = 102$. Da ferner $\bar{k} = t$ gilt, berechnen wir als zugehörige obere Schranke $U_4^0 = \bar{c} + \bar{c}_{\bar{k}-1} + c_i + \left\lfloor \hat{b} \frac{c_{\bar{k}}}{a_{\bar{k}}} \right\rfloor = 70 + 96 - 70 + \lfloor 1 \cdot 1.75 \rfloor = 97$. Es ist nicht $U = Z$. setze $\bar{U}' = 97$.

zu $i = 6$: es ist $\bar{b} + a_i = 19 + 10 = 29 < 53 = \bar{a}_t$, daher bestimmen wir den lokalen kritischen Index $\bar{k}$ über $\bar{a}_{\bar{k}-1} \leq \bar{b} + a_i < \bar{a}_{\bar{k}}$ zu $\bar{k} = 1$. Wir ermitteln als lokale Restkapazität $\hat{b} = \bar{b} + a_i - \bar{a}_{\bar{k}-1} = 19 + 10 - 10 = 19$ und als Aktualisierung der unteren Schranke $Z = \max\{Z, \bar{c} + \bar{c}_{\bar{k}-1} - c_i\} = \max\{102, 70 + 20 - 20\} = 102$. Da ferner $r < \bar{k} < t$ gilt, berechnen wir als zugehörige obere Schranke aus

$$\begin{aligned} U_6^0 &= \bar{c} + \bar{c}_{\bar{k}-1} - c_i + \max\left\{\left\lfloor \frac{c_{\bar{k}+1}}{a_{\bar{k}+1}} \hat{b} \right\rfloor, \left\lfloor c_{\bar{k}} + \left(\hat{b} - a_{\bar{k}}\right) \frac{c_{\bar{k}-1}}{a_{\bar{k}-1}} \right\rfloor\right\} \\ &= 70 + 20 - 20 + \max\left\{\lfloor 1.947 \cdot 19 \rfloor, \lfloor 39 + (19 - 20) \cdot 2 \rfloor\right\} = 107 \end{aligned}$$

Es ist nicht $U = Z$. Setze $\bar{U}' = 107$

zu $i = 1$: es ist $\bar{b} + a_i = 19 + 20 = 39 < 53 = \bar{a}_t$, daher bestimmen wir den lokalen kritischen Index $\bar{k}$ über $\bar{a}_{\bar{k}-1} \leq \bar{b} + a_i < \bar{a}_{\bar{k}}$ zu $\bar{k} = 3$. Wir ermitteln als lokale Restkapazität $\hat{b} = \bar{b} + a_i - \bar{a}_{\bar{k}-1} = 19 + 20 - 30 = 9$ und als Aktualisierung der

unteren Schranke $Z = \max\{Z, \bar{c} + \bar{c}_{\bar{k}-1} - c_i\} = \max\{102, 70 + 59 - 39\} = 102$. Da ferner $r < \bar{k} < t$ gilt, berechnen wir als zugehörige obere Schranke aus

$$\begin{aligned} U_1^0 &= \bar{c} + \bar{c}_{\bar{k}-1} - c_i + \max\left\{\left\lfloor \frac{c_{\bar{k}+1}}{a_{\bar{k}+1}}\hat{b}\right\rfloor, \left\lfloor c_{\bar{k}} + \left(\hat{b} - a_{\bar{k}}\right)\frac{c_{\bar{k}-1}}{a_{\bar{k}-1}}\right\rfloor\right\} \\ &= 70 + 59 - 39 + \max\{\lfloor 1.75 \cdot 9\rfloor, \lfloor 37 + (9 - 19) \cdot 1.95\rfloor\} = 107 \end{aligned}$$

Es ist nicht $U = Z$. Es bleibt bei $\bar{U}' = 107$ wegen $i = k$.

Lokale Schranken für (KP_i^1), $i \geq k$**:** Wir betrachten nun nacheinander die Indizes $i = 1,\ 3,\ 7,\ 2,\ 5$.

zu $i = 1$: Es ist $\bar{b} - a_1 = 19 - 20 = -1 < 0$. Daher berechnet sich die zugehörige obere Schranke zu $U_1^1 = \bar{c} + c_i + \left\lfloor \left(\bar{b} - a_i\right)\frac{c_r}{a_r}\right\rfloor = 70 + 39 + \lfloor(-1)\,1.95\rfloor = 107.0$.

zu $i = 3$: es ist $\bar{b} - a_3 = 19 - 19 = 0$ und damit gilt $\bar{k} = 6$. Es folgt $\hat{b} = \bar{b} - a_i - \bar{a}_{\bar{k}-1} = 19 - 19 - 0 = 0$ und $Z = \max\{Z, \bar{c} + \bar{c}_{\bar{k}-1} + c_i\} = \max\{102, 70 + 0 + 37\} = 107$. Damit gilt nun $Z = U$ und damit ist das Verfahren zuende, eine explizite Reduzierung der Aufgabe ist nicht nötig: die dem letzten Z zugrunde liegende zulässige Lösung , $x^T = (0, 0, 1, 1, 0, 0, 0)$, ist optimal.

Bemerkung: Man kann die vorliegende Prozedur MTR+ noch verfeinern, wenn man die Techniken der Reduzierungsprozedur LMR verwendet. (Übungsaufgabe)

3.2.4 Exakte Lösung

Hat die Reduzierung ergeben, dass noch eine Restaufgabe zu lösen ist, so erfolgt dies, indem zunächst die Restaufgabe *nach abfallenden Nutzen-Gewichts-Quotienten sortiert wird.* Schließlich wendet man noch MT1 an, um die Restaufgabe und damit die gegebene Aufgabe endgültig zu lösen.

In unserem *Beispiel* erübrigt sich der Schritt, da die Optimallösung bereits gefunden wurde.

Zu beachten ist allerdings *folgende Besonderheit bei der Anwendung vom MT2*: Durch Anwendung der Prozedur LUMT wurde bereits die *Optimallösung* der auf $[r, t]$ eingeschränkten Aufgabenstellung berechnet. Ergab nur die Reduzierung , dass die Menge H aus der Partition in JF_1 enthalten ist und genauso die Menge L in JF_0, so liegt die Optimallösung der gegebenen Aufgabe bereits vor und es kann auf die Lösung der Restaufgabe verzichtet werden.

3.3 Erfahrungsbericht

Im Folgenden berichten wir über Erfahrungen mit den dargestellten Prozeduren zur Lösung von (KP_{01}), wie sie in der Literatur notiert sind. Dabei entsprechen die getesteten Prozeduren nicht immer genau den von uns formulierten. Es ergibt sich dennoch eine ungefähre Einschätzung der Leistungsfähigkeit.

3.3.1 Vergleich der LU-Prozeduren

Bricht man das MT2 Verfahren nach der LU Prozedur ab, so entsteht eine Prozedur für den unsortierten Fall, die in Konkurrenz zu der oben dargestellten Prozedur LUDGu tritt. Wir wollen dieses Verfahren **Prozedur LUMTu** nennen.

Martello und Toth geben in ihrem Buch (1990) eine Vergleich zwischen LUMTu und einer LU-Prozedur, die die Dantzigschranke zur Bestimmung einer oberen Schranke und die Gierige Heuristik zur Bestimmung einer unteren Schranke verwendet. Diese Prozedur trägt den Namen **S(0)** und löst die gegebene Aufgabe näherungsweise in polynomialer Zeit. Leider ist sie im genannten Buch nur in einer Version für geordnete Probleme angegeben. Es ist daher nicht ganz klar, wie im unsortierten Fall verfahren wurde. Wir geben den Vergleich trotzdem an.

Die von Martello und Toth verwendete Version von MT2 benutzt die Parameter $\delta = 2\sqrt{n},\ \alpha = 0.2,\ \beta = 1.0$ und $\eta = 20$. Die approximative Version dagegen verwendet die Parameter $\delta = 5,\ \alpha = 0.0,\ \beta = 1.0,\ \eta = 200$. Programmiert wurde in FORTRAN und verwendet wurde eine HP 9000/840. Die gemessenen Zeiten, die jeweils den Durchschnitt von 20 gleichartigen Aufgaben darstellen, werden in Sekunden angegeben. Getestet wurden Aufgaben mit zufälligen Daten im Bereich $[1, 1000]$ und mit $b = 0.5 * \sum_{j=1}^{n} a_j$. In Klammern wird jeweils die prozentuale Abweichung der unteren Schranke von der oberen Schranke (oder der Optimallösung, sofern bekannt) angegeben.

Laufzeitvergleich

n	MT2	LUMTu	S(0)
500	0,067	0,029 (0,0076)	0,049 (0,4912)
1000	0,122	0,058 (0,0042)	0,105 (0,2121)
5000	0,515	0,296 (0,0018)	0,618 (0,0554)
10000	0,872	0,641 (0,0008)	1,320 (0,0204)
50000	3,562	3,399 (0,0001)	8,071 (0,0043)
100000	7,001	6,865 (0,0001)	16,35 (0,0023)
250000	17,135	16,69 (0,0001)	44,72 (0,0009)

Es lassen sich daraus *zwei Schlüsse* ziehen.

1. LUMTu distanziert deutlich eine bekannte LU-Prozedur, die mit einfachen Schranken arbeitet.

2. Es lohnt sich kaum, nur LUMTu anzuwenden, MT2 ist fast genau so schnell und arbeitet optimal.

3.3.2 Vergleich der Kernschätzungen

Im Folgenden vergleichen wir die drei uns bekannten Kernschätzungen theoretisch und experimentell.

Theoretischer Vergleich

Zur Ermittlung eines Schätz-Kerns haben wir bereits drei Verfahren kennengelernt:

- Die *Prozedur BZC* von Balas und Zemel, die auf der Prozedur LUDGu aufbaute. Letztere hatte als vorrangiges Ziel, den Kritischen Index bei einem unsortierten Knapsack Problem zu ermitteln. Durch leichte Abwandlung entstand BZC.
- Die *Prozedur MTC* operierte ähnlich wie die Prozedur BZC, ihr Hauptaugenmerk lag auf der Bestimmung eines möglichst symmetrischen Schätz-Kerns vorgegebenen Größe.
- Die *Prozedur PPC* ist ebenfalls eine Abwandlung von LUDGu. Sie bringt eine Vertauschungsoperation in die Teilsortierung ein. (Bei den beiden bisherigen Verfahren blieb die relative Reihenfolge der Indizes in H und L unverändert.)

Alle drei Verfahren übergeben nach Abschluss, sofern Zwischenergebnisse gespeichert werden, eine teilweise geordnete Menge der Form (3.2)

$$N = H_1 \cup H_2 \cup ... \cup H_h \cup C \cup L_l \cup ... \cup L_2 \cup L_1$$

wobei die Effizienzen innerhalb der Teilbereiche ungeordnet sind, ansonsten aber von links nach rechts abfallen. Der Schätzkern C enthält den kritischen Index.

Alle Verfahren basieren auf der Idee des Quicksort, bei der die Elemente einer zu sortierenden Menge zunächst bzgl eines willkürlich ausgesuchten Vergleichslelements λ getrennt werden. Dieses Vergleichselement ist im Falle von BZC der Median der ersten drei Elemente, im Falle der beiden anderen Verfahren der Median des ersten,

des mittleren und des letzten Elementes. Im Gegensatz zu Quicksort werden aber nicht alle so entstehenden Teilbereiche wieder nach dem gleichen Prinzip getrennt, sondern nur der Teilbereich, zu dem der kritische Index gehört.

Eine weitere Besonderheit ist, dass die zu λ gleichen Elemente bei den beiden ersten Verfahren in einer abgetrennten Teilmenge gesammelt werden, um erst später zu entscheiden, welchem der beiden Hauptteilbereiche sie zugeschlagen werden.

Eine Schwäche des klassischen Verfahrens BZC ist die Abhängigkeit von möglicherweise unglücklichen $\lambda-$Werten und eine daraus resultierende zu feine Partition und entsprechend längere Laufzeit. Dem versucht MTC durch eine Untergrenze an die Größe von C zu begegnen. Hilfsmittel dazu ist eine Nachiteration, wenn eine bestimmte Größe nicht erreicht wird, auch in dem Fall, wo die angestrebte Zerlegung eigentlich schon erreicht ist, aber C zu klein wäre. Dadurch kommen mehr $\lambda's$ ins Spiel und die Abhängigkeit von schlechten $\lambda-$Werten wird verringert.

Das Verfahren PPC dagegen bringt bessere $\lambda-$Werte ins Spiel, indem es die Reihenfolge der Items permanent verbessert und damit die Chance auf bessere $\lambda-$Werte erhöht.

Numerischer Vergleich

In der Bachelor-Arbeit von Varga (2008) wurden die drei Kernschätzungen experimentell verglichen. Dabei wurden wie schon zuvor Aufgaben unterschiedlicher Größe und unterschiedlichen Typs untersucht und das Verfahren MT2 darauf angewendet, wobei die erste Phase MTC auch alternativ durch BZC bzw PPC ersetzt wurde.

Betrachtet wurden erneut Aufgaben mit unkorrelierten, schwach korrelierten, stark korrelierten Daten sowie Subset Sum Probleme. Dabei haben die Aufgaben wieder jeweils n Variablen mit unterschiedlichen n und die Daten schwanken im Bereich $[1, R]$ für unterschiedliche $R \in \mathbf{N}$. Als Kapazität wurde immer die Hälfte des potentiellen Gesamtgewichts gewählt. Es wurde die von Martello und Toth vorgeschlagenen $\delta = 2\sqrt{n},\ \alpha = 0.2,\ \beta = 1.0$ verwendet und $\gamma = 20$ eingesetzt. Bei PPC wurde zur besseren Vergleichbarkeit mit MTC die Schätz-Kern-Obergrenze δ durch $2 * \delta$ ersetzt.

Vergleich der Kern-Algorithmen bei unkorrelierten Daten Die Verfahren wurden unter MS Excel in VBA programmiert und liefen auf einem normalen PC. Es ergaben sich die folgenden Ergebnisse als Mittelwerte von jeweils 10 zufälligen Aufgaben gleichen Typs.

Zunächst wurden die **absoluten Laufzeiten der Gesamtalgorithmen** gemessen.

Zeit in sec	n/R	500	1000	5000	10000
MT2-BZC	10	0,231	0,915	21,371	83,722
MT2-MTC	10	0,266	0,803	20,867	84,403
MT2-PPC	10	0,243	0,734	20,369	84,990

Zeit in sec	n/R	500	1000	5000	10000
MT2-BZC	100	0,259	0,872	18,794	69,005
MT2-MTC	100	0,266	0,848	17,974	69,305
MT2-PPC	100	0,251	0,772	17,589	67,234

Zeit in sec	n/R	500	1000	5000	10000
MT2-BZC	1000	0,283	0,805	18,849	76,664
MT2-MTC	1000	0,270	0,906	18,570	70,302
MT2-PPC	1000	0,258	0,817	17,520	68,841

Man sieht, dass die drei Varianten sehr ähnliche Gesamtlaufzeiten haben, mit leichten Vorteilen für MT2-PPC.

Danach wurden die **absoluten Laufzeiten der Kern-Schätzungen** verglichen.

Zeit in sec	n/R	500	1000	5000	10000
MT2-BZC	10	0,106	0,406	8,597	33,870
MT2-MTC	10	0,147	0,346	8,572	33,463
MT2-PPC	10	0,097	0,237	5,197	21,385

Zeit in sec	n/R	500	1000	5000	10000
MT2-BZC	100	0,125	0,406	9,187	30,964
MT2-MTC	100	0,141	0,384	8,040	30,959
MT2-PPC	100	0,087	0,228	5,247	21,177

Zeit in sec	n/R	500	1000	5000	10000
MT2-BZC	1000	0,156	0,363	9,078	33,974
MT2-MTC	1000	0,135	0,406	8,691	31,761
MT2-PPC	1000	0,093	0,222	5,168	21,109

Hier zeigt sich eine deutliche Laufzeit-Überlegenheit von PPC, während die anderen beiden in etwa gleich auf liegen. Auf die Gesamtzeit scheinen sich die großen Unterschiede aber kaum auszuwirken.

Die nächste Tabelle gibt den **prozentualen Zeit-Anteil** innerhalb von MT2 an, den die Kernberechnung in Anspruch nahm. Da die Gesamtlaufzeiten alle recht ähnlich waren, ergeben sich die kleinsten prozentualen Anteile natürlich für PPC. Es zeigt sich, dass der prozentuale Anteil wesentlich geringer ist als die zuvor genannten 75% bei Vollsortierung.

Zeit-Anteil %	n/R	500	1000	5000	10000
MT2-BZC	10	45,293	44,201	40,052	40,378
MT2-MTC	10	55,277	43,111	41,152	39,383
MT2-PPC	10	39,857	32,287	25,729	25,163

Zeit-Anteil %	n/R	500	1000	5000	10000
MT2-BZC	100	48,123	46,249	48,619	44,866
MT2-MTC	100	52,511	45,135	44,670	44,629
MT2-PPC	100	34,278	29,533	29,847	31,569

Zeit-Anteil %	n/R	500	1000	5000	10000
MT2-BZC	1000	54,850	44,889	47,864	44,586
MT2-MTC	1000	49,149	44,495	46,503	45,123
MT2-PPC	1000	36,101	27,327	29,530	30,692

Die einzelnen Kernalgorithmen hatten unterschiedliche Zielgrößen der Schätz-Kerne. Die gemessenen Schätz-Kerne spiegeln dies wider. Die folgende Tabelle gibt die **absolute Größe der Schätz-Kerne** an, wie sie sich nach Ablauf von Stategie-Teil 1 ergeben. Dabei wurden die von den Verfahren ermittelten Schätz-Kerns nicht nachträglich manipuliert. Man sieht, dass MTC die größten Schätz-Kerne und BZC die kleinsten Schätz-Kerne produziert.

Zu beachten seien vorab die aus den obigen Parameterfestlegungen resultierenden

Zielkorridore für die absolute Größe des Schätzkerns:

$\alpha * \delta \setminus n$	500	1000	5000	10000
$0.2 * 2\sqrt{n}$	8.944	12.649	28.284	40.000
$1 * 2\sqrt{n}$	44.721	63.246	141.420	200.000
$2 * 2\sqrt{n}$	89.443	126.490	282.840	400.000

Gemessen wurden dann die folgenden Größen:

Anzahl items	n/R	500	1000	5000	10000
MT2-BZC	10	28,200	33,600	80,200	139,667
MT2-MTC	10	54,600	92,600	203,400	335,000
MT2-PPC	10	29,400	55,800	191,800	157,333

Anzahl items	n/R	500	1000	5000	10000
MT2-BZC	100	24,400	31,200	85,600	121,333
MT2-MTC	100	62,600	94,400	229,200	269,333
MT2-PPC	100	56,600	84,000	164,600	211,333

Anzahl items	n/R	500	1000	5000	10000
MT2-BZC	1000	30,600	43,200	117,000	103,000
MT2-MTC	1000	65,000	100,200	171,200	281,333
MT2-PPC	1000	63,000	66,600	213,250	243,000

Die folgenden Tabellen geben die wichtigste Information: wie effektiv sind die ermittelten Schätz-Kerne? Wir wollen das daran bemessen, wie stark die Schätz-Kerne in Strategie-Teil 3 reduziert werden. Wir notieren die **Reduktion als prozentualen Anteil des Schätz-Kerns** nach Strategie-Teil 1, also die Größe der Restaufgabe nach Ablauf der Reduktion als Anteil des Schätz-Kerns nach Ablauf von Strategie Teil 3. (Prozentangaben größer als 100% besagen, dass der Schätz-Kern nach der Reduzierung anwächst.)

Reduzierung %	n/R	500	1000	5000	10000
MT2-BZC	10	33,002	74,838	195,622	175,237
MT2-MTC	10	18,449	22,324	57,122	68,844
MT2-PPC	10	35,140	40,722	62,181	253,854

Reduzierung %	n/R	500	1000	5000	10000
MT2-BZC	100	17,942	76,969	13,378	14,466
MT2-MTC	100	10,392	5,205	3,650	7,030
MT2-PPC	100	11,456	6,285	9,349	8,802

Reduzierung %	n/R	500	1000	5000	10000
MT2-BZC	1000	15,124	14,961	5,284	10,185
MT2-MTC	1000	7,048	7,385	4,119	3,271
MT2-PPC	1000	11,166	9,382	2,816	4,265

Vergleich der Kern-Algorithmen bei schwach korrelierten Daten Wir notieren hier nur die **absoluten Laufzeiten der Gesamtalgorithmen** , bei den Kernschätzungen ergibt sich ein ähnliches Bild wie bei den unkorrelierten Daten mit Ausnahme der Reduzierung bei $R = 10$. Hier tritt genau das Gegenteil ein, die Reduzierung ist sehr effektiv, bzw. das Gesamtverfahren ist nach Strategie Teil 2 bereits abgeschlossen.

Zeit in sec	n/R	500	1000	5000	10000
MT2-BZC	10	0,263	0,798	18,802	70,068
MT2-MTC	10	0,276	0,830	18,030	73,685
MT2-PPC	10	0,230	0,755	17,331	70,640

Zeit in sec	n/R	500	1000	5000	10000
MT2-BZC	100	0,270	0,788	18,564	69,237
MT2-MTC	100	0,275	0,870	17,662	67,846
MT2-PPC	100	0,244	0,772	16,939	66,745

Zeit in sec	n/R	500	1000	5000	10000
MT2-BZC	1000	0,272	0,886	17,833	80,377
MT2-MTC	1000	0,275	0,862	17,356	73,222
MT2-PPC	1000	0,242	0,777	17,114	70,386

Auch hier zeigen sich alle drei Varianten fast gleichauf, mit leichten Vorteilen für PPC.

Vergleich der Kern-Algorithmen bei stark korrelierten Daten Wir notieren hier nur die **absoluten Laufzeiten der Gesamtalgorithmen**, bei den Kernschätzungen ergibt sich ein ähnliches Bild wie bei den unkorrelierten Daten.

Zeit in sec	n/R	500	1000	5000	10000
MT2-BZC	10	0,273	0,917	21,937	79,138
MT2-MTC	10	0,263	0,872	20,717	79,880
MT2-PPC	10	0,231	0,853	21,100	85,823

Zeit in sec	n/R	500	1000	5000	10000
MT2-BZC	100	0,281	0,728	55,798	146,276
MT2-MTC	100	0,275	0,791	55,407	148,570
MT2-PPC	100	0,250	0,737	54,360	147,427

Zeit in sec	n/R	500	1000	5000	10000
MT2-BZC	1000	0,310	0,836	37,553	243,266
MT2-MTC	1000	0,280	0,858	26,861	190,455
MT2-PPC	1000	0,270	0,777	26,980	181,555

Auch hier zeigen sich alle drei Varianten fast gleichauf, es ist kein einheitlicher Trand auszumachen.

Vergleich der Kern-Algorithmen "subset sums" Bei diesem Aufgabentyp erweist sich MT2-BZC als überfordert (Gesamtlaufzeit > 1000 sec.), was durchweg an der Kern-Schätzung liegt.

Zunächst wurden die **absoluten Laufzeiten der Gesamtalgorithmen** gemessen.

Zeit in sec	n/R	500	1000	5000	10000
MT2-MTC	10	1,287	4,956	121,601	514,961
MT2-PPC	10	0,208	0,698	16,079	21,063

Zeit in sec	n/R	500	1000	5000	10000
MT2-MTC	100	1,297	4,971	121,724	483,640
MT2-PPC	100	0,208	0,789	16,102	69,773

Zeit in sec	n/R	500	1000	5000	10000
MT2-MTC	1000	1,263	4,956	122,554	482,242
MT2-PPC	1000	0,433	0,743	17,220	68,838

Man sieht hier einen deutlichen Laufzeitvorteil für MT2-PPC.

Vergleichen wir nun die **absoluten Laufzeiten der Kern-Schätzungen**.

Zeit in sec	n/R	500	1000	5000	10000
MT2-MTC	10	1,219	4,708	115,958	492,531
MT2-PPC	10	0,083	0,204	5,219	20,703

Zeit in sec	n/R	500	1000	5000	10000
MT2-MTC	100	1,235	4,719	116,094	461,390
MT2-PPC	100	0,067	0,188	5,250	20,672

Zeit in sec	n/R	500	1000	5000	10000
MT2-MTC	1000	1,239	4,708	115,390	460,125
MT2-PPC	1000	0,083	0,135	5,250	20,672

Auch hier eine deutliche Überlegenheit von PPC. Insbesondere zeigt sich, dass sie lange Gesamtlaufzeit von MT2-MTC in der Kernschätzung begründet liegt. Dies kann auch an der nächsten Tabelle abgelesen werden.

Diese Tabelle gibt den **prozentualen Zeit-Anteil** innerhalb von MT2 an, den die Kernberechnung in Anspruch nahm. Es zeigt sich, dass dieser Anteil bei MT2-MTC unabhängig von R und n bei ungefähr 95% liegt, während dieser bei MT2-PPC grob um die 30% schwankt.

Zeit-Anteil %	n/R	500	1000	5000	10000
MT2-MTC	10	94,725	95,002	95,359	95,644
MT2-PPC	10	38,815	29,153	32,458	98,291

Zeit-Anteil %	n/R	500	1000	5000	10000
MT2-MTC	100	95,301	94,917	95,375	95,399
MT2-PPC	100	31,998	23,828	32,606	29,628

Zeit-Anteil %	n/R	500	1000	5000	10000
MT2-MTC	1000	98,148	95,010	94,154	95,414
MT2-PPC	1000	24,887	17,854	30,489	30,205

Die folgende Tabelle gibt die **absolute Größe der Schätz-Kerne** an, wie sie sich nach Ablauf von Stategie-Teil 1 ergeben. Man sieht, dass MTC sehr kleine Schätz-Kerne und PPC Schätz-Kerne der üblichen Größe produziert.

Anzahl items	n/R	500	1000	5000	10000
MT2-MTC	10	1,667	1,000	1,667	5,000
MT2-PPC	10	63,000	125,000	157,000	313,000

Anzahl items	n/R	500	1000	5000	10000
MT2-MTC	100	1,000	1,667	1,667	1,000
MT2-PPC	100	63,000	125,000	157,000	313,000

Anzahl items	n/R	500	1000	5000	10000
MT2-MTC	1000	1,000	1,000	3,000	1,000
MT2-PPC	1000	63,000	125,000	157,000	313,000

Fazit: Man sollte durchgehend die Version MT2-PPC benutzen, sie ist in der Regel die schnellste und stabilste.

3.3.3 Übungsaufgaben

Aufgabe 1

Man formuliere eine Reduzierungsprozedur MTR++, die aus eine Synthese der Prozeduren MTR+ und LMR besteht, d.h. man bringe die in LMR verwendeten Techniken in MTR+ ein.

Kapitel 4

Expandierende Verfahren

Das Konzept, Knapsack Probleme nach der Strategie von Balas/Zemel zu lösen, hat einen entscheidenden Nachteil: Der Kern der gegebenen Aufgabe ist nicht von vorneherein bekannt und muss daher geschätzt werden. Schätzt man ihn groß genug, so liefert ggf. bereits das Preprocessing die Optimallösung des Problems, zu relativ geringen Kosten. Je größer der Schätz-Kern, desto mehr Aufwand muss aber dazu betrieben werden.

Schätzt man dagegen zu klein, so muss man zweimal ein unter Umständen sehr ähnliches Knapsackproblem lösen: die zum Schätz-Kern gehörige Aufgabe und die zum Schluss übrig bleibende. Beide Aufgaben können bis auf wenige Variablen übereinstimmen, verwenden kann man die Schritte aus der ersten Bearbeitung aber nicht.

4.1 Grundversion eines expandierenden B&B-Verfahrens

Pisinger (1995) machte daher den Vorschlag, die Anwendung des B&B Verfahrens in der Nähe des break item zu beginnen und von dort aus die Sortierung und die Fixierung der Variablen mehr oder weniger symmetrisch nach außen zu bewegen, so dass schnell der Kern der Aufgabe fixiert und so die Optimallösung gefunden ist.

Er setzt diese Idee um über ein B&B Verfahren, das in besonderer Art verzweigt. Wir beschreiben die Vorgehensweise zunächst in allgemeiner Form und nur *für voll vorsortierte Aufgabenstellungen.*

Zugrunde gelegt werde die Aufgabe (KP_{01}). Zu dieser betrachten wir Testaufgaben der Form $(KP[\bar{x}_r^t])$, für die ein Bereich $F = [r,t] \subset N$ vorgegeben ist, auf dem alle Variablen bereits auf die Werte $\bar{x}_r^t := (\bar{x}_r, ..., \bar{x}_t)^T$ fixiert sind. Wir wollen auch den leeren Bereich F zulassen. Diesen beschreiben wir mit $[r,t] := [s, s-1]$ für ein gewähltes $s \in N$.

Vorgegeben sei nun die Testaufgabe $(KP[\bar{x}_r^t])$, die wir verzweigen wollen. Zur Abkürzung setzen wir $\bar{c} = \sum_{j=1}^{r-1} c_j + \sum_{j=r}^{t} \bar{x}_j c_j$ und $\bar{b} = b - \bar{a} = b - \sum_{j=1}^{r-1} a_j - \sum_{j=r}^{t} \bar{x}_j a_j$ und unterscheiden zwei Fälle:

Fall 1: Ist $\bar{b} \geq 0$, so liegt der kritische Index $\bar{k}$ der Testaufgabe rechts von t, d.h. es gilt $\bar{k} > t$. In diesem Fall wird wie folgt verzweigt: die Festlegungen

$$\begin{aligned}
\left(K\bar{P}\right) &: \quad x_{t+1} = \bar{x}_{t+1} = 0,\ x_{t+2} = \bar{x}_{t+2} = 0, ..., x_n = \bar{x}_n = 0 \\
\left(KP\left[\bar{x}_r^{t+1}\right]\right) &: \quad x_{t+1} = \bar{x}_{t+1} = 1 \\
\left(KP\left[\bar{x}_r^{t+2}\right]\right) &: \quad x_{t+1} = \bar{x}_{t+1} = 0,\ x_{t+2} = \bar{x}_{t+2} = 1 \\
\left(KP\left[\bar{x}_r^{t+3}\right]\right) &: \quad x_{t+1} = \bar{x}_{t+1} = 0,\ x_{t+2} = \bar{x}_{t+2} = 0,\ x_{t+3} = \bar{x}_{t+3} = 1 \\
&\ldots \\
(KP[\bar{x}_r^n]) &: \quad x_{t+1} = \bar{x}_{t+1} = 0, ..., x_{n-1} = \bar{x}_{n-1} = 0,\ x_n = \bar{x}_n = 1
\end{aligned}$$

werden zu $(KP[\bar{x}_r^t])$ hinzugenommen und führen zu den Testaufgaben mit den angegebenen Namen. Es handelt sich dabei um eine vollständige Verzweigung. In der Tat: sei x eine zulässige Lösung von $(KP[\bar{x}_r^t])$. Dann sind entweder alle $x_{t+1}, ..., x_n$ gleich null oder mindestens eine dieser Variablen ist 1. Die Verzeigung führt über in neue Testaufgaben, in denen wiederum ein zusammenhängender Bereich $[r', t']$ fixiert ist.

Diese Verzweigung hat den Charme, dass sie in der Regel wegen Auslotung gar nicht vollständig durchgeführt werden muss. Dies ergibt sich aus folgenden Überlegungen:

- Die Testaufgabe $\left(K\bar{P}\right)$ ist sehr einfach zu lösen und damit ausgelotet: die Optimallösung ist offensichtlich gegeben durch

  $$x_1 = 1, ..., x_{r-1} = 1$$

  mit Zielfunktionswert $\bar{c}$. Diese erweitert sich unter Einbeziehung der Fixierungen unmittelbar zu einer zulässigen Lösung von (KP_{01}). Es wird natürlich immer zuerst nach $\left(K\bar{P}\right)$ verzweigt, da die entstehende Testaufgabe ausgelotet ist, erscheint diese Verzweigung aber nicht explizit.

- Es brauchen also nur die Aufgaben $(KP[\bar{x}_r^j])$ mit $j \geq t+1$ weiter untersucht und ggf. verzweigt zu werden. Zu den Aufgaben $\left(K\bar{P}[\bar{x}_r^j]\right) : (KP[\bar{x}_r^j])$ ohne die jeweils letzte Fixierung $x_j = 1$, $(j \geq t+1)$, kann leicht und in einheitlicher Weise eine obere Schranke u_j angegeben werden:

  $$u_j = \bar{c} + \left\lfloor \bar{b} \frac{c_j}{a_j} \right\rfloor$$

  Das ergibt sich aus Satz 2.1, weil ja für den kritischen Index $\bar{k}$ dieser Aufgaben gilt: $\bar{k} \geq j$. Diese Schranke gilt dann natürlich auch für $(KP[\bar{x}_r^j])$. Daraus

schließen wir: Gilt $u_j \leq L$ mit aktuell gültiger unterer Schranke L, so ist die Testaufgabe $(KP\,[\bar{x}_r^j])$ ausgelotet!

Es ist unmittelbar einzusehen, dass wegen der vorausgesetzten Sortierung der Effizienzen die betrachteten oberen Schranken u_j mit steigendem j potenziell abnehmen. Daher gilt: ist $(KP\,[\bar{x}_r^j])$ auf die beschriebene Weise als ausgelotet erkannt worden, so sind auch die Aufgaben $(KP\,[\bar{x}_r^i])$ mit $i \geq j$ alle ausgelotet!

Fall 2: Ist $\bar{b} < 0$, so liegt der kritische Index $\bar{k}$ der Testaufgabe links von r, d.h. es gilt $\bar{k} < r$. In diesem Fall verzweigen wir analog nach links:

$$\begin{array}{rcl}
\left(K\bar{P}\right) & : & x_{r-1} = \bar{x}_{r-1} = 1,\ x_{r-2} = \bar{x}_{r-2} = 1, ..., x_1 = \bar{x}_1 = 1 \\
\left(KP\left[\bar{x}_{r-1}^t\right]\right) & : & x_{r-1} = \bar{x}_{r-1} = 0 \\
\left(KP\left[\bar{x}_{r-2}^t\right]\right) & : & x_{r-1} = \bar{x}_{r-1} = 1,\ x_{r-2} = \bar{x}_{r-2} = 0 \\
\left(KP\left[\bar{x}_{r-3}^t\right]\right) & : & x_{r-1} = \bar{x}_{r-1} = 1,\ x_{r-2} = \bar{x}_{r-2} = 1,\ x_{r-3} = \bar{x}_{r-3} = 0 \\
& ... & \\
\left(KP\left[\bar{x}_1^t\right]\right) & : & x_{r-1} = \bar{x}_{r-1} = 1, ..., x_2 = \bar{x}_2 = 1,\ x_1 = \bar{x}_1 = 0
\end{array}$$

Es handelt sich auch hier offensichtlich um eine vollständige Verzweigung. Die Aufgabe $\left(K\bar{P}\right)$ ist wieder als erstes zu bearbeiten und erscheint nicht explizit. Wegen $\bar{b} < 0$ besitzt sie nämlich keine zulässige Lösung und ist dementsprechend sofort ausgelotet.

Auch zu den Aufgaben $\left(K\bar{P}\left[\bar{x}_j^t\right]\right)$ mit $1 \leq j \leq r-1$, bei denen die zuletzt genannte Fixierung $x_j = 0$ in $\left(KP\left[\bar{x}_j^t\right]\right)$ jeweils weggelassen wird, kann eine obere Schranke nach Satz 2.1 unmittelbar angegeben werden:

$$u_j = \bar{c} + \left\lfloor \bar{b}\frac{c_j}{a_j} \right\rfloor$$

wobei diesmal die Schranken wegen der vorausgesetzten Sortiertheit mit abfallendem j abfallen, da ja $\bar{b} < 0$ gilt.

Damit können die Testaufgaben $\left(KP\left[\bar{x}_j^t\right]\right)$ bei geeigneter unterer Schranke L wieder ausgelotet werden, sobald $u_j \leq L$ gilt und es ist klar, dass wenn $\left(KP\left[\bar{x}_j^t\right]\right)$ auf diese Weise ausgelotet ist, dass dann auch $(KP\,[\bar{x}_i^t])$ ausgelotet ist für $i \leq j$. Wiederum gilt diese Schranke auch für $\left(KP\left[\bar{x}_j^t\right]\right)$.

4.1.1 Das Verfahren

Im Folgenden beschreiben wir nun das aus diesen Vorüberlegungen resultierende B&B Verfahren, wenn als *Verwaltungsstrategie LIFO* angewendet wird. Wir tun

dies der Einfachheit halber in *rekursiver Form,* die LIFO besonders gut angepasst ist. Dazu beschreiben wir eine Prozedur **Auslote,** die die oben beschriebene Verzweigungsstruktur realisiert und aufgrund der rekursiven Struktur des Verfahrens den gesamten unterhalb gelegenen Ast auslotet. Diese Prozedur ruft sich also selbst auf. Die Anfangssituation wird durch das Hauptprogramm **Prozedur expcore1** festgelegt. Gestartet wird, wie von Pisinger vorgeschlagen, bei $[r,t] = [k, k-1]$.

Wir unterstellen, dass die Vektoren $x, \bar{x} \in \{0,1\}^n$ und der Wert z globale Variablen sind, die ständig mit aktuellen Werten belegt sind. x gibt die aktuell beste Lösung an, z deren Zielwert. $\bar{x}$ gibt den laufenden Lösungsvektor an.

Die Prozedur Auslote $[r,t]$ übergibt die Daten r,t, wobei $[r,t]$ das Intervall der aktuell fixierten Variablen bezeichne. Es gilt $\bar{x}_1 = ... = \bar{x}_{r-1} = 1$ und $\bar{x}_{t+1} = ... = \bar{x}_n = 0$.

Prozedur expcore1

Input: (KP_{01}), kritischer Index k, zulässige Lösung x mit Zielwert z, obere Schranke U.

Output: zulässige Lösung x mit Zielwert z.

setze $\bar{x}$ gleich der Dantziglösung und Auslote $[k, k-1]$.

Prozedur: Auslote $[r,t]$ berechne $\bar{c} := \sum_{j=1}^{n} \bar{x}_j c_j$ und $\bar{b} := b - \bar{a} := b - \sum_{j=1}^{n} \bar{x}_j a_j$.

ist $\bar{b} \geq 0$, so führe aus:

ist $\bar{c} > z$, so setze $z = \bar{c}$ und $x = \bar{x}$. `% Berücksichtigung vom` $\left(K\bar{P}\right)$

ist $U = z$, **stopp**

für $j = t+1$ bis n führe aus:

ist $u_j = \bar{c} + \left\lfloor \bar{b} \frac{c_j}{a_j} \right\rfloor \leq z$, **return**.

setze $\bar{x}_j = 1$ und Auslote $[r,j]$,

setze $\bar{x}_j = 0$.

sonst führe aus:

für $j = r-1$ bis 1 $(step: -1)$ führe aus:

ist $u_j = \bar{c} + \left\lfloor \bar{b} \frac{c_j}{a_j} \right\rfloor \leq z$, **return**

setze $\bar{x}_j = 0$ und Auslote $[j,t]$,

setze $\bar{x}_j = 1$

Bemerkung: Man kann sich klarmachen, dass die jetzt verwendete Verzweigung auch durch eine Abfolge von binären Verzweiungen beschrieben werden kann. Die Testaufgabaend er Verzweigung sind die bätter eines Binärbaumes. Z.B im Fall 1:

Skizze:

Man kann die Prozedur Auslote daher dahingehend abändern, dass die beiden "für-Schleifen" durch je einen weiteren Aufruf der Auslote-Prozedur ersetzt werden.

Beispiel

Überprüfen wir dieses Verfahren an unserem sortierten Beispiel: Mit $b = 12$ lauten die Daten der Aufgabe

Gegenstand j	1	2	3	4	5	6
Nutzen c_j	8	8	6	10	12	12
Gewicht a_j	1	2	2	4	6	10

Der kritische Index liegt hier bei $k = 5$.

Wir starten unser Verfahren mit der Festsetzung von $x^T = (0, 0, 0, 0, 0, 0)$ mit $z = 0$ und $\bar{x}^T = (1, 1, 1, 1, 0, 0)$ und $U = \infty$.

Auslote $[5, 4]$: Es ist $\bar{c} = 8+8+6+10 = 32 > 0 = z$, $\bar{b} = 10-1-2-2-4 = 3 \geq 0$. Daher setze $z = 32$ und $x^T = (1, 1, 1, 1, 0, 0)$.

Setze $j = 5$ und prüfe $u_5 = 32 + \left\lfloor 3 \cdot \frac{12}{6} \right\rfloor = 38 > 32 = z$.

Setze $\bar{x}_5 = 1$. Es gilt also $\bar{x}^T = (1, 1, 1, 1, 1, 0)$

Auslote $[5, 5]$: Es ist $\bar{c} = 8+8+6+10+12 = 44$, $\bar{b} = 10-1-2-2-4-6 = -3 < 0$.

Damit setze $j = 4$ und prüfe $u_4 = \left\lfloor 44 - 3 \cdot \frac{10}{4} \right\rfloor = 36 > 32 = z$.

Setze also $\bar{x}_4 = 0$. Es gilt also $\bar{x}^T = (1, 1, 1, 0, 1, 0)$

Auslote $[4, 5]$: Es ist $\bar{c} = 8 + 8 + 6 + 12 = 34$, $\bar{b} = 10 - 1 - 2 - 2 - 6 = 1 > 0$. Es ist $\bar{c} > z = 32$.

Daher ist eine neue zulässige Lösung gefunden: $x^T = (1,1,1,0,1,0)$ mit $z = 34$.

Setze nun $j = 6$ und prüfe $u_6 = \lfloor 34 + 1 \cdot \frac{12}{10} \rfloor = 35 > 34 = z$.

Also setze $\bar{x}_6 = 1$. Es gilt dann $\bar{x}^T = (1,1,1,0,1,1)$

Auslote $[4,6]$: Es ist $\bar{c} = 8+8+6+10+12 = 44$, $\bar{b} = 10-1-2-2-6-10 = -11 < 0$.

Setze $j = 3$ und prüfe $u_3 = 44 + \lfloor (-11) \cdot \frac{6}{2} \rfloor = 11 < 32 = z$.

setze $\bar{x}_6 = 0$, es gilt nun wieder $\bar{x}^T = (1,1,1,0,1,0)$. Return.

In $[4,5]$ kann keine weitere Verzweigung durchgeführt werden wegen $j = 6$.

Setze $\bar{x}_4 = 1$, es gilt dann $\bar{x}^T = (1,1,1,1,1,0)$. Return.

Zurück in $[5,5]$ setzen wir $j = 3$ und prüfen $u_3 = \lfloor 44 - 3 \cdot \frac{6}{2} \rfloor = 35 > 34 = z$.

Setze also $\bar{x}_3 = 0$, es gilt dann $\bar{x}^T = (1,1,0,1,1,0)$.

Auslote $[3,5]$: Es ist $\bar{c} = 8 + 8 + 10 + 12 = 38$, $\bar{b} = 10 - 1 - 2 - 4 - 6 = -1 < 0$.

Daher setze $j = 2$ und prüfe $u_2 = \lfloor 38 - 1 \cdot \frac{8}{2} \rfloor = 34 > 34 = z$. Dies ist nicht der Fall.

Setze wieder $\bar{x}_3 = 1$, es gilt also $\bar{x}^T = (1,1,1,1,1,0)$, return.

Zurück zu $[5,5]$. Setze hier $j = 2$ und prüfe $u_2 = \lfloor 44 - 3 \cdot \frac{8}{2} \rfloor = 32 > 34 = z$. Dies ist nicht der Fall,

Setze daher $\bar{x}_5 = 0$, so dass wieder $x^T = (1,1,1,1,0,0)$ gilt. Return.

Zurück zu $[5,4]$. Setze hier $j = 6$ und prüfe $u_5 = 32 + \lfloor 3 \cdot \frac{12}{10} \rfloor = 35 > 34 = z$.

Also setze $\bar{x}_6 = 1$, wodurch $x^T = (1,1,1,1,0,1)$ gilt.

Auslote $[5,6]$: Es ist $\bar{c} = 8+8+6+10+12 = 44$, $\bar{b} = 10-1-2-2-4-10 = -9 < 0$.
Setze $j = 4$ und prüfe $u_4 = \lfloor 44 - 9 \cdot \frac{10}{4} \rfloor = 21 > 34 = z$. Das ist nicht der Fall,

Setze $\bar{x}_6 = 0$, es gilt dann $x^T = (1,1,1,1,0,0)$, return.

Damit ist das Verfahren beendet. Ergebnis: $x^T = (1,1,1,0,1,0)$ mit $z = 34$. Den Verlauf des Verfahrens beschreibt der folgende Baum: (zu durchlaufen von links nach rechts)

4.1.2 Pisingers Version

Das Original-Verfahren von PISINGER, hier bezogen auf die *vorsortierte Aufgabenstellung*, ist noch besser an die rekursive Vorgehensweise angepasst. Wir beschreiben die *Auslote* entsprechende Prozedur als **Function expbranch** und das Hauptprogramm **ExpKnap**.

Bei diesem werden

$$\bar{c} = \sum_{j=1}^{r-1} c_j + \sum_{j=r}^{t} \bar{x}_j c_j \quad \text{und} \quad \bar{b} = b - \bar{a} = b - \sum_{j=1}^{r-1} a_j - \sum_{j=r}^{t} \bar{x}_j a_j$$

bereits übergeben. Eine Besonderheit gegenüber der bisher notierten Version ist, dass die gesuchte Lösung x in Form ihrer Abweichung von der Dantziglösung notiert wird. Die Menge E beschreibt die Komponenten, in der x von der Dantziglösung abweicht. Dadurch wird das Führen der Lösung $\bar{x}$ überflüssig.

Das Hauptprogramm bestimmt zunächst den kritischen Index. Man beachte die geänderte Anfangssituation bzgl r und t.

Function expknap(c, a, x, b, n)

$c_0 := 1;\ a_0 := 0;$

$c_{n+1} := 0;\ a_{n+1} := 1;$

$k := 0;\ \bar{b} := b;\ \bar{c} := 0;$

while $\big(a_k \le \bar{b}\big)$ **do** $\bar{c} := \bar{c} + c_k;\ \bar{b} := \bar{b} - a_k;\ k := k + 1;$ **endwhile**;

expbranch$\big(k-1, k, \bar{c}, \bar{b}\big)$;

bestimme die optimale Lösung aus E.

return z.

Function expbranch$\big(r, t, \bar{c}, \bar{b}\big)$: **boolean**

var improved: boolean

$improved := false$

if $\big(\bar{b} \ge 0\big)$, **then**

  if $(\bar{c} > z)$ **then** $improved := true;\ z := \bar{c},\ E := \varnothing$; **fi**

  repeat

    if $\bar{c} + \left\lfloor \bar{b}\frac{c_t}{a_t} \right\rfloor \le z$ **then return** $improved$; **fi**

if expbranch$\big(r, t+1, \bar{c}+c_t, \bar{b}+a_t\big)$ **then** $improved := true$; $E := E \cup t$; **fi**

$t = t + 1$;

forever

else

repeat

if $\bar{c} + \left\lfloor \bar{b}\frac{c_s}{a_s} \right\rfloor \leq z$ **then return** $improved$; **fi**

if expbranch$\big(\bar{c}-c_r, \bar{b}-a_r, r-1, t\big)$ **then** $improved := true$; $E := E \cup s$; **fi**

$s := s - 1$;

forever

Man überprüfe, dass die Anwendung von expknap auf unser Beispiel den gleichen Verlauf nimmt wie die der bisherigen Version des Verfahrens. (Übungsaufgabe)

4.2 Verbesserungen

Im Folgenden versuchen wir, das Verfahren expcore1 noch zu verbessern, indem wir es analog zu expbranch besser an die Rekursion anpassen, aber auch schärfere Schranken verwenden.

4.2.1 Lokale obere Schranken

In unserem Verfahren benutzen wir die folgende Formel zur Ermittlung einer oberen Schranke für $(KP\,[\bar{x}_r^j])\,,\ j \notin [r,t]$

$$u_j = \bar{c} + \left\lfloor \bar{b}\frac{c_j}{a_j} \right\rfloor$$

mit $\bar{c} = \sum_{j=1}^{r-1} c_j + \sum_{j=r}^{t} \bar{x}_j c_j$ und $\bar{b} = b - \bar{a} = b - \sum_{j=1}^{r-1} a_j - \sum_{j=r}^{t} \bar{x}_j a_j$. Wir haben gesehen: diese Formel sorgt für obere Schranken, die nach außen hin abfallen und somit die Verzweigung nach außen begrenzen. Man kann den Verzweigungsvorgang sofort abbrechen, wenn erstmals eine Auslotung stattfindet.

Diese Formel ist nicht die schärfste, denn sie basiert darauf, die äußerste Fixierung der Variablen einfach zu ignorieren. Tut man dies nicht, so entstehen potentiell schärfere Formeln:

Fall 1: Es gelte $\bar{b} \geq 0$. Berechnet werden soll eine Formel für eine obere Schranke für $(KP[\bar{x}_r^j])$ mit $j \geq t+1$ und den Fixierungen $\bar{x}_{t+1} = 0, ..., \bar{x}_{j-1} = 0, \bar{x}_j = 1$. Eine obere Schranke bekommt man dann offenbar nach der Formel

$$\bar{u}_j = \bar{c} + c_j + \left\lfloor \left(\bar{b} - a_j\right) \frac{c_{j+1}}{a_{j+1}} \right\rfloor, \quad \text{falls } \left(\bar{b} - a_j\right) \geq 0$$

bzw

$$\bar{u}_j = \bar{c} + c_j + \left\lfloor \left(\bar{b} - a_j\right) \frac{c_{r-1}}{a_{r-1}} \right\rfloor, \quad \text{falls } \left(\bar{b} - a_j\right) < 0$$

Fall 2: Es gelte $\bar{b} < 0$, aber $b - \sum_{i=r}^{t} \bar{x}_i a_i \geq 0$. Dann ist die Aufgabe $(KP[\bar{x}_r^j])$ auf jeden Fall lösbar, ihr kritischer Index liegt links von r. Berechnet werden soll eine Formel für $\left(KP\left[\bar{x}_j^t\right]\right)$ mit $j \leq r-1$ und den Fixierungen $\bar{x}_{r-1} = 1, ..., \bar{x}_{j+1} = 1, \bar{x}_j = 0$. Eine obere Schranke bekommt man dann offenbar nach der Formel

$$\bar{u}_j = \bar{c} - c_j + \left\lfloor \left(\bar{b} + a_j\right) \frac{c_{t+1}}{a_{t+1}} \right\rfloor, \quad \text{falls } \left(\bar{b} + a_j\right) \geq 0$$

bzw

$$\bar{u}_j = \bar{c} - c_j + \left\lfloor \left(\bar{b} + a_j\right) \frac{c_{j-1}}{a_{j-1}} \right\rfloor, \quad \text{falls } \left(\bar{b} - a_j\right) < 0$$

Fall 3: Es gelte $\bar{b} < 0$ und $b - \sum_{i=r}^{t} \bar{x}_i a_i < 0$. Dann ist $(KP[\bar{x}_r^j])$ nicht lösbar, wir setzen $\bar{u}_j = -\infty$.

4.2.2 Verbesserte Schranken für $(KP[\bar{x}_r^t])$

Ist im Rahmen des Verfahrens auf der Stufe, wo $(KP[\bar{x}_r^t])$ verarbeitet wird, ein Teil der Verzweigungen bereits durchgeführt, so sind diese Knoten bereits ausgelotet und es kann eine verschärfte Schranke für $(KP[\bar{x}_r^j])$ angegeben werden.

Fall 1: Es sei $\bar{b} \geq 0$. Dann ist zunächst $u_{t+1} = \bar{c} + \left\lfloor \bar{b} \frac{c_{t+1}}{a_{t+1}} \right\rfloor$ eine obere Schranke für für den Zielwert aller zulässigen Lösungen in $(KP[\bar{x}_r^t])$.

Wurden bereits die Verzweigungen zu $t+1, ..., j-1$ durchgeführt, wurden also $\bar{x}_{t+1} = ... = \bar{x}_{j-1} = 0$ fixiert, so gibt $u_j = \bar{c} + \left\lfloor \bar{b} \frac{c_j}{a_j} \right\rfloor$ eine obere Schranke für den Zielwert der *restlichen zulässigen* Lösungen in $(KP[\bar{x}_r^t])$ an. Dies kann analog zur U_2-Formel verbessert werden zu:

$$\tilde{u}_j = \max\left\{\hat{u}_j, u_{j+1}\right\}$$

wobei $\hat{u}_j$ eine beliebige obere Schranke für $(KP[\bar{x}_r^j])$ ist, z.B. $\hat{u}_j = \bar{u}_j$.

Dies ergibt anfänglich

$$\tilde{u}_{t+1} = \max\left\{\bar{u}_{t+1}, u_{t+2}\right\},$$

d.h $\tilde{u}_{t+1}$ ist eine obere Schranke für $(KP\left[\bar{x}_r^t\right])$.

Fall 2: Es gelte $\bar{b} < 0$, aber $b - \sum_{i=r}^{t} \bar{x}_i a_i \geq 0$. Dann ist zunächst $u_{r-1} = \bar{c} + \left\lfloor \bar{b} \frac{c_{r-1}}{a_{r-1}} \right\rfloor$ eine obere Schranke für für den Zielwert aller zulässigen Lösungen in $(KP\left[\bar{x}_r^t\right])$.

Wurden bereits die Verzweigungen zu $r-1, ..., j+1$ durchgeführt, wurden also $\bar{x}_{r-1} = ... = \bar{x}_{j+1} = 1$ fixiert, so gibt $u_j = \bar{c} + \left\lfloor \bar{b} \frac{c_j}{a_j} \right\rfloor$ eine obere Schranke für den Zielwert der *restlichen zulässigen* Lösungen in $(KP\left[\bar{x}_r^t\right])$ an. Dies kann analog zur U_2-Formel verbessert werden zu:

$$\tilde{u}_j = \max\left\{\hat{u}_j, u_{j-1}\right\}$$

wobei $\hat{u}_j$ eine beliebige obere Schranke für $\left(KP\left[\bar{x}_j^t\right]\right)$ ist, z.B. $\hat{u}_j = \bar{u}_j$.Dies ergibt anfänglich

$$\tilde{u}_{r-1} = \max\left\{\bar{u}_{r-1}, u_{r-2}\right\},$$

d.h $\tilde{u}_{r-1}$ ist eine obere Schranke für $(KP\left[\bar{x}_r^t\right])$.

Fall 3: Es gelte $\bar{b} < 0$ und $b - \sum_{i=r}^{t} \bar{x}_i a_i < 0$. In diesem Fall ist $(KP\left[\bar{x}_r^t\right])$ unlösbar.

4.2.3 Das verbesserte Verfahren

Formulieren wir nun das Verfahren neu mit den verbesserten Schranken. Dabei wollen wir zusätzlich zu r und t die folgenden Werte bereits übergeben: Grundnutzen $\hat{c}$ und Restkapazität $\hat{b}$ sowie die Restkapazität $\hat{b}_0$ nach Fixierung der Variablen in $[r, t]$. Es gilt also grundsätzlich für $(KP\left[\bar{x}_r^t\right])$

$$\hat{c} = \sum_{j=1}^{r-1} c_j + \sum_{j=r}^{t} \bar{x}_j c_j \quad \text{und} \quad \hat{b} = b - \bar{a} = b - \sum_{j=1}^{r-1} a_j - \sum_{j=r}^{t} \bar{x}_j a_j \quad \text{und} \quad \hat{b}_0 = b - \sum_{j=r}^{t} \bar{x}_j a_j$$

Dadurch gehen wir, wie in der Prozedur expbranch, auf die rekursive Form des Verfahren besser ein und stellen gleichzeitig sicher, dass nicht unnütz verzweigt wird. Allerdings bleiben wir der Übersichtlichkeit halber dabei, die Vektoren x und $\bar{x}$ als globale Variablen zu führen.

Das Hauptprogramm erweitert die Aufgabenstellung um eine geeignete Randsituation zu den Indizes $j = 0$ und $j = n + 1$.

Prozedur expcore2

Input: (KP_{01}), kritischer Index k, zulässige Lösung x mit Zielwert z, obere Schranke U.

Output: zulässige Lösung x mit Zielwert z.

setze $\bar{x}$ gleich der Dantziglösung.

setze $c_0 := 1;\ a_0 := 0;\ c_{n+1} := 0;\ a_{n+1} := 1;$

berechne $\hat{c} := \sum_{j=1}^{k-1} c_j \quad \hat{b} := b - \bar{a} := b - \sum_{j=1}^{k-1} a_j$ und setze $\hat{b}_0 = b$.

ist $\hat{c} > z$ und $\hat{b} \geq 0$, so setze $z = \hat{c}$ und $x = \bar{x}$. Gilt nun $z = U$, **stopp**

Auslote2 $\left(KP\left[k, k-1, \hat{c}, \hat{b}, \hat{b}_0\right]\right)$.

Prozedur: Auslote2 $\left(KP\left[r, t, \hat{c}, \hat{b}, \hat{b}_0\right]\right)$

Setze $\bar{c} = \hat{c}$ und $\bar{b} = \hat{b}$ und $\bar{b}_0 = \hat{b}_0$.

ist $\bar{b} \geq 0$, so führe aus: % Fall 1

für $j = t+1$ bis n führe aus:

bilde $\hat{c} = \bar{c} + c_j,\ \hat{b} = \bar{b} - a_j$ und $\hat{b}_0 = \bar{b}_0 - a_j$.

ist $\hat{b} \geq 0$ und $\hat{c} > z$, so setze $z = \hat{c}$ und $x = \bar{x}$ sowie $x_j = 1$. Gilt nun $z = U$, **stopp**

ist $\hat{b}_0 \geq 0$, so führe aus:

setze $\bar{u}_j = \hat{c} + \left\lfloor \hat{b} \frac{c_{j+1}}{a_{j+1}} \right\rfloor$, falls $\hat{b} \geq 0$, sonst $\bar{u}_j = \hat{c} + \left\lfloor \hat{b} \frac{c_{r-1}}{a_{r-1}} \right\rfloor$

ist $\bar{u}_j > z$, so setze $\bar{x}_j = 1$ und Auslote2 $\left(KP\left[r, j, \hat{c}, \hat{b}, \hat{b}_0\right]\right)$

ist $u_{j+1} = \bar{c} + \left\lfloor \bar{b} \frac{c_{j+1}}{a_{j+1}} \right\rfloor \leq z$, return

setze $\bar{x}_j = 0$

sonst führe aus: % Fall 2

für $j = r-1$ bis 1 $(step : -1)$ führe aus:

bilde $\hat{c} = \bar{c} - c_j,\ \hat{b} = \bar{b} + a_j$ und $\hat{b}_0 = \bar{b}_0$, falls $j = r-1$, sonst $\hat{b}_0 = \hat{b}_0 - a_{j+1}$

ist $\hat{b}_0 < 0$, return

ist $\hat{b} \geq 0$, $\hat{c} > z$, so setze $z = \hat{c}$ und $x = \bar{x}$ sowie $x_j = 0$. Gilt nun $z = U$, **stopp**

setze $\bar{u}_j = \hat{c} + \left\lfloor \hat{b} \frac{c_{t+1}}{a_{t+1}} \right\rfloor$, falls $\hat{b} \geq 0$, sonst $\bar{u}_j = \hat{c} + \left\lfloor \hat{b} \frac{c_{j-1}}{a_{j-1}} \right\rfloor$

ist $\bar{u}_j > z$, so setze $\bar{x}_j = 0$ und Auslote2 $\left(KP\left[r, t, \hat{c}, \hat{b}, \hat{b}_0\right]\right)$

ist $u_{j-1} = \bar{c} + \left\lfloor \bar{b}\frac{c_{j-1}}{a_{j-1}} \right\rfloor \leq z$, return.

setze $\bar{x}_j = 1$

sonst:

Beispiel

Überprüfen wir das verbesserte Verfahren erneut an unserem sortierten Beispiel: Mit $b = 12$ lauten die Daten der Aufgabe

Gegenstand j	1	2	3	4	5	6
Nutzen c_j	8	8	6	10	12	12
Gewicht a_j	1	2	2	4	6	10

Der kritische Index liegt hier bei $k = 5$.

Wir starten unser Verfahren mit der Festsetzung von $r = 5,\ t = 4$ sowie

$x^T = (0, 0, 0, 0, 0, 0)\,,\ z = 0$ und $\bar{x}^T = (1, 1, 1, 1, 0, 0)\,.$

Berechne $\hat{c} = 8 + 8 + 6 + 10 = 32 > 0 = z$, $\hat{b} = 10 - 1 - 2 - 2 - 4 = 3$, $\hat{b}_0 = 12$.

Daher setze als neue zulässige Lösung fest: $z = 32$ und $x^T = (1, 1, 1, 1, 0, 0)\,.$

Auslote2 $(KP\,[5, 4, 32, 3, 12])$: Es ist $\bar{c} = 32$, $\bar{b} = 3 \geq 0$, $\bar{b}_0 = 12$ % (Fall 1)

Setze $j = 5$. Es ist $\hat{c} = \bar{c} + c_5 = 32 + 12 = 44,\ \hat{b} = \bar{b} - a_5 = 3 - 6 = -3 < 0,\ \hat{b}_0 = 12 - 6 = 6 \geq 0$.

Daher berechne $\bar{u}_5 = \hat{c} + \left\lfloor \hat{b} \cdot \frac{c_4}{a_4} \right\rfloor = 44 + \left\lfloor (-3) \cdot \frac{10}{4} \right\rfloor = 36 > 32 = z$.

Also setze $\bar{x}_5 = 1$, so dass nun $\bar{x}^T = (1, 1, 1, 1, 1, 0)$ und

Auslote2 $(KP\,[5, 5, 44, -3, 6])$: Es ist $\bar{c} = 44$, $\bar{b} = -3 < 0,\ \bar{b}_0 = 6$ % (Fall 2)

Damit setze $j = 4$. Es ist $\hat{c} = \bar{c} - c_4 = 44 - 10 = 34 > 32 = z,\ \hat{b} = \bar{b} + a_4 = -3 + 4 = 1 \geq 0,\ \hat{b}_0 = \bar{b}_0 = 6$.

Daher ist eine neue zulässige Lösung gefunden: $x^T = (1, 1, 1, 0, 1, 0)$ mit $z = 34$.

Bilde dann $\bar{u}_4 = \hat{c} + \left\lfloor \hat{b}\frac{c_6}{a_6} \right\rfloor = 34 + \left\lfloor 1 \cdot \frac{12}{10} \right\rfloor = 35$.

Setze daher $\bar{x}_4 = 0$, womit $\bar{x}^T = (1, 1, 1, 0, 1, 0)$ gilt und

Auslote2 $(KP\,[4, 5, 34, 1, 6])$: Es ist $\bar{c} = 34$, $\bar{b} = 1 \geq 0,\ \bar{b}_0 = 6$ % (Fall 1)

Setze nun $j = 6$. Es ist $\hat{c} = \bar{c} + c_6 = 34 + 12 = 46,\ \hat{b} = \bar{b} - a_6 = 1 - 10 = -9 < 0,\ \hat{b}_0 = 6 - 10 = -4 < 0.$

Da $j = 6 = n$, gehe zurück zu % (Fall 2)

$(KP\,[5, 5, 44, -3, 6])$ und setze $\bar{x}_4 = 1$. Es gilt wieder $\bar{x}^T = (1, 1, 1, 1, 1, 0)$

Nun setzen wir $j = 3$ und bilden $\hat{c} = \bar{c} - c_3 = 44 - 6 = 38,\ \hat{b} = \bar{b} + a_3 = -3 + 2 = -1,\ \hat{b}_0 = \hat{b}_0 - a_j = 6 - 4 = 2.$

Also bilde $\bar{u}_3 = \hat{c} + \left\lfloor \hat{b}\frac{c_2}{a_2} \right\rfloor = 38 + \left\lfloor (-1) \cdot \frac{8}{2} \right\rfloor = 34 \leq 34 = z.$

Prüfe daher $u_2 = \bar{c} + \left\lfloor \bar{b}\frac{c_2}{a_2} \right\rfloor = 44 + \left\lfloor (-3) \cdot \frac{8}{2} \right\rfloor = 32 \leq 34 = z$. Zurück zu % (Fall 1)

$(KP\,[5, 4, 32, 3, 12])$. Hier setze $\bar{x}_5 = 0$, so dass $\bar{x}^T = (1, 1, 1, 1, 0, 0)$ gilt.

Setze nun $j = 6$ und bilde $\hat{c} = \bar{c} + c_6 = 32 + 12 = 44,\ \hat{b} = \bar{b} - a_6 = 3 + 10 = -7,\ \hat{b}_0 = \bar{b}_0 - a_j = 12 - 10 = 2.$

Also bilde $\bar{u}_6 = \hat{c} + \left\lfloor \hat{b}\frac{c_4}{a_4} \right\rfloor = 44 + \left\lfloor (-7) \cdot \frac{10}{4} \right\rfloor = 26 \leq 34 = z.$

Prüfe $u_7 = \bar{c} + \left\lfloor \bar{b}\frac{c_7}{a_7} \right\rfloor = 32 + \left\lfloor 3 \cdot \frac{0}{1} \right\rfloor = 32 \leq 34 = z$. Return und fertig.

Damit ist das Verfahren beendet. Ergebnis: $x^T = (1, 1, 1, 0, 1, 0)$ mit $z = 34$. Den Verlauf des Verfahrens beschreibt der folgende Baum:

Bemerkung: Man kann $\bar{x}$ bzw x auch als *Abweichung von der Dantziglösung* darstellen, wie in Pisingers Algorithmus. (Übungsaufgabe)

4.2.4 Diskussion

Die zuletzt besprochenen B&B Verfahren können auch mit anderer Strategie als LIFO und iterativ anstelle von rekursiv durchgeführt werden, z.B. mit einer Knotenauswahl nach der besten oberen Schranke. Wesentlich erscheint, dass die Testaufgaben *immer* von der Form $(KP\,[\bar{x}_r^t])$ sind, unwesentlich, welche Strategie beim Durchlauf des Entscheidungsbaums angewendet wird.

Bei der Betrachtung des obigen Beispiels fällt auf, dass im Grunde nur immer die die lokalen Schranken $\bar{u}_j$ und u_{j+1} bzw u_{j-1} berechnet und mit der unteren Schranke verglichen werden. Dies entspricht einer permantenten Anwendung der U_2-Formel, also der schon früher verwendeten binären Verzweigung! Wie bereits beobachtet, können die beiden Schranken daher zur Ermittlung einer oberen Schranke für die vorliegende Testaufgabe herangezogen werden. Dann stellt das Verfahren bei vorzeitigem Abbruch eine LU-Prozedur mit Breitensuche dar.

Es empfiehlt sich, die bisherige Verzweigung als fortgesetzte binäre Verzweigung zu betrachten. Wir formulieren dies nicht aus, sondern demonstrieren die Vorgehensweise nur am Beispiel. Zu beachten ist allerdings, dass nun nicht $\bar{x}$ als globale Variable behandelt werden kann, da dieser Vektor gleichzeitig mit mehreren Belegungen auftritt. Wir benutzen daher eine neue Notation für die Testaufgabe: $\left(KP\left[(\bar{x}_r,...,\bar{x}_t)_r^t,\hat{c},\hat{b},\hat{b}_0\right],u\right)$, wobei $\hat{b}_0$ die laufende Restkapazität allein aus der Fixierung und u eine obere Schranke für diese Aufgabe darstellt. (Ausformulierung: Übungsaufgabe).

Betrachten wir wieder die Aufgabe

Gegenstand j	1	2	3	4	5	6
Nutzen c_j	8	8	6	10	12	12
Gewicht a_j	1	2	2	4	6	10

mit $b = 12$ und kritischem Index $k = 5$. Wir wenden die Version expcore2 an, allerdings mit der Best-Strategie.

Wir starten unser Verfahren mit der Festsetzung von $r = 5,\ t = 4$ und $x^T = (0,0,0,0,0,0)$, $z = 0$. Berechne $\hat{c} = 8+8+6+10 = 32$, $\hat{b} = 10-1-2-2-4 = 3$, $\hat{b}_0 = 12$. Wir setzen $U = \infty$.

Auslote $(KP[5,4,32,3,10],\infty)$: Es ist $\bar{c} = 32 > 0 = z$, $\bar{b} = 3 \geq 0$. Daher setze als neue zulässige Lösung fest: $z = 32$ und $x^T = (1,1,1,1,0,0)$. Untere Schranke ist also $L = 32$

Setze $j = 5$. Es ist $\hat{c} = \bar{c} + c_5 = 32 + 12 = 44,\ \hat{b} = \bar{b} - a_5 = 3 - 6 = -3 < 0,\ \hat{b}_0 = 10 - 6 = 4 \geq 0$. Daher berechne $\bar{u}_5 = \hat{c} + \left\lfloor \hat{b}\frac{c_4}{a_4} \right\rfloor = 44 + \left\lfloor (-3)\,\frac{10}{4} \right\rfloor = 36$ und $u_6 = \bar{c} + \left\lfloor \bar{b}\frac{c_6}{a_6} \right\rfloor = 32 + \left\lfloor 3\frac{12}{10} \right\rfloor = 35$. Damit gilt sich als globale ober Schranke $U = \max\{35,36\} = 36$.

Unsere Liste mit den noch zu verzweigenden Testaufgaben besteht aus

$\left(KP\left[(1)_5^5,44,-3,4\right],36\right)$ und $\left(KP\left[(0)_5^5,32,3,10\right],35\right)$.

Wir wählen zur weiteren Verzweigung die erste Testaufgabe aus, verzweige

$\left(KP\left[(1)_5^5, 44, -3, 4\right], 36\right)$: Es ist $\bar{c} = 44$, $\bar{b} = -3 < 0$, $\bar{b}_0 = 4$. Damit setze $j = 4$. Es ist $\hat{c} = \bar{c} - c_4 = 44 - 10 = 34$, $\hat{b} = \bar{b} + a_4 = -3 + 4 = 1 \geq 0$, $\hat{b}_0 = \bar{b}_0 = 4$. Hier lesen wir bereits eine neue zulässige Lösung ab: $x^T = (1, 1, 1, 0, 1, 0)$ mit $z = 34$. Damit gilt $L = 34$.

Bilde $\bar{u}_4 = \hat{c} + \left\lfloor \hat{b}\frac{c_6}{a_6} \right\rfloor = 34 + \left\lfloor 1\frac{12}{10} \right\rfloor = 35$ und $u_3 = \bar{c} + \left\lfloor \bar{b}\frac{c_3}{a3} \right\rfloor = 44 + \left\lfloor (-3)\,\frac{6}{2} \right\rfloor = 35$. Damit ergibt sich $U = 35$.

Unsere Liste besteht nun aus: $\left(KP\left[(0)_5^5, 32, 3, 10\right], 35\right)$, $\left(KP\left[(0, 1)_4^5, 34, 1, 4\right], 35\right)$ und $\left(KP\left[(1, 1)_4^5, 44, -3, 0\right], 35\right)$ mit oberer Schranke $u = 35$. Die globale obere Schranke für die gegebene Aufgabe laute also $U = 35$.

Wir brechen ab. Ergebnis: $L = 34$, $U = 35$.

Zusammenfassung: Insgesamt lässt sich die angewendete Vorgehensweise wie folgt beschreiben:

Betrachtet wird ein B&B Verfahren, bei dem die Testaufgaben von der Form $\left(KP\left[r, t, \hat{c}, \hat{b}, \hat{b}_0\right], u\right)$ sind für geeignete $r, t \in \mathbf{N}$. Dabei sind die Variablen $x_r, ..., x_t$ auf $\bar{x}_r, ..., \bar{x}_t$ fixiert, es gelte

$$\hat{c} := \sum_{j=1}^{r-1} c_j + \sum_{j=r}^{t} \bar{x}_j c_j \quad \hat{b} := b - \bar{a} := b - \sum_{j=1}^{r-1} a_j - \sum_{j=r}^{t} \bar{x}_j a_j,\ \hat{b}_0 = b - \sum_{j=r}^{t} \bar{x}_j a_j$$

Anfänglich werde die Aufgabe $\left(KP\left[\bar{x}_k^{k-1}, \hat{c}, \hat{b}, b\right], u\right)$ betrachtet, k der kritische Index der vorgegebenen Aufgabe, $\hat{c}$ der Nutzen der Dantziglösung und $\hat{b}$ die Restkapazität der Dantziglösung.

Verzweigt wird eine gegebene Testaufgabe $\left(KP\left[\bar{x}_r^t, \hat{c}, \hat{b}, \hat{b}_0\right], u\right)$ wie folgt:

Ist $\hat{b} \geq 0$, so betrachte die beiden Söhne, die durch Hinzunahme der Fixierung (in dieser Reihenfolge) $x_{t+1} = 1$ und $x_{t+1} = 0$ entstehen. Zu beachten ist dabei, dass durch die Festlegung $x_{t+1} = 0$ die Werte $\hat{c}$, $\hat{b}$ und $\hat{b}_0$ nicht verändert werden. Bei der Fixierung von $x_{t+1} = 1$ verändern sich diese Werte. Gilt danach $\hat{b}_0 < 0$, so kann auf die Verzweigung verzichtet und gleich $x_{t+1} = 0$ fixiert werden. $\bar{u}_{t+1}$ gibt eine obere Schranke für den ersten und u_{t+2} eine obere Schranke für den zweiten Sohn an.

Ist $\hat{b} < 0$, so betrachte die beiden Söhne, die durch Hinzunahme der Fixierung (in dieser Reihenfolge) $x_{r-1} = 0$ und $x_{r-1} = 1$ entstehen. Zu beachten ist dabei, dass durch die Festlegung $x_{r-1} = 1$ die Werte $\hat{c}$ und $\hat{b}$ nicht verändert werden. $\bar{u}_{r-1}$ gibt eine obere Schranke für den ersten und u_{r-2} eine obere Schranke für den zweiten Sohn an.

Eine Testaufgabe ist ausgelotet, wenn die genannten oberen Schranke für die beiden Söhne schlechter als die untere Schranke ist. Eine neue untere Schranke ist gefunden, wenn $\hat{c} > z$ und $\hat{b} \geq 0$ gilt.

4.3 Ausweitung auf den nicht-sortierten Fall

Es ist eine weitere charmante Eigenschaft des bisher skizzierten Verfahrens, dass es eine elegante Ausweitung auf den nicht-sortierten Fall zulässt. Die Idee dabei ist, von einem sortierten Schätz-Kern auszugehen und die Sortierung *bei Bedarf* nach außen auszuweiten. Die folgende Prozeduren stellen eine Abwandlung des Core-Algorithmus PPC dar. Sie führt eine Teilsortierung des Teilbereichs $[f, g]$ von N durch, wenn dieser garantiert entweder links oder rechts vom bisher fixierten Bereich liegt.

Zunächst wird die Kernschätzung PPC durchgeführt, bei der die Zwischenergebnisse gespeichert werden. Das Ergebnis der Kernschätzung ist also eine Teilsortierung

$$N = H_1 \cup H_2 \cup ... \cup H_{s_1} \cup C \cup L_{s_2} \cup ... \cup L_2 \cup L_1$$

Dabei sei $C =: [p, q]$, $H = H_1 \cup H_2 \cup ... \cup H_{s_1}$ und $L = L_{s_2} \cup ... \cup L_2 \cup L_1$.

4.3.1 Teilsortierung links

Wir beschreiben zunächst die Teilsortierung links. Bei Bedarf soll also der fixierte Bereich nach links erweitert werden. Dazu wähle das Intervall $H_{s_1} =: [f, g]$ aus, um es weiter zu strukturieren.

Ergebnis der folgenden **Prozedur erweiterelinks** (f, g) ist die gewünschte teilsortierte Zerlegung

$$[f, g] = H'_1 \cup H'_2 \cup ... \cup H'_h \cup C',$$

wobei C' aus mindestens ein Element besteht und sortiert ist. Nach Anwendung der Prozedur wird C mit C' geordnet vereinigt, d.h. $p := f$ gesetzt, und die Liste $H'_1, H'_2, ..., H'_h$ der bisherigen Liste H hinten angefügt.

procedure erweiterelinks (f, g)

solange $(g - f) \geq 2$ führe aus

setze $m := \lfloor (f + g)/2 \rfloor$ und tausche die Inhalte der Stellen f, m, g so, dass die Effizienzen zu diesen drei Stellen abfallend sind.

setze $i := f;\ j := l;$

wiederhole

wiederhole $i := i + 1$; bis die Effizienz der Stelle i kleiner gleich der der Stelle m ist;

wiederhole $j := j - 1$; bis die Effizienz der Stelle j größer gleich der der Stelle m ist;

ist $(i < j)$ dann vertausche die Inhalte von i und j;

bis $i > j$

setze geordnet $H' := H' \cup \{[f, i-1]\}$; $f := i$;

ist $(g - f = 1)$, und tausche ggf. die Inhalte der Stellen f, g so, dass die Effizienzen zu diesen Stellen abfallend sind.

setze $p = f$ und vereinige geordnet H und H'.

4.3.2 Teilsortierung rechts

Analog ergibt sich eine Prozedur zur Erweiterung der Sortierung nach rechts. Mit dem Ergebnis dieser Prozedur wird analog zu erweiterelinks (f, g) verfahren.

procedure erweitererechts (f, g)

solange $(g - f) \geq 2$ führe aus

setze $m := \lfloor (f + g)/2 \rfloor$ und tausche die Inhalte der Stellen f, m, g so, dass die Effizienzen zu diesen drei Stellen abfallend sind.

setze $i := f$; $j := l$;

wiederhole

wiederhole $i := i + 1$; bis die Effizienz der Stelle i kleiner gleich der der Stelle m ist;

wiederhole $j := j - 1$; bis die Effizienz der Stelle j größer gleich der der Stelle m ist;

ist $(i < j)$ dann vertausche die Inhalte von i und j;

bis $i > j$

setze geordnet $L' := L' \cup \{[i, g]\}$; $g := i - 1$;

ist $(g - f = 1)$, so tausche ggf. die Inhalte der Stellen f, g so, dass die Effizienzen zu diesen Stellen abfallend sind.

setze $q = g$ und vereinige geordnet L und L'.

4.3.3 Einbau der Sortierung

Diese Prozeduren können nun in die Verfahren expcore1 und expcore2 eingebaut werden, um so auch unsortierte Probleme lösen zu können. Wir zeigen am Beispiel expcore1 wie.

Prozedur expcore1U

Input: (KP_{01}), kritischer Index k, zulässige Lösung x mit Zielwert z, obere Schranke U, teilsortierte Zerlegung $N = H_1 \cup ... \cup H_{s_1} \cup C = [p,q] \cup L_{s_2} \cup ... \cup L_1$

Output: zulässige Lösung x mit Zielwert z.

setze $\bar{x}$ gleich der Dantziglösung und AusloteU $(KP[k, k-1])$.

Prozedur: AusloteU $(KP[r,t])$

berechne $\bar{c} := \sum_{j=1}^{n} \bar{x}_j c_j$ und $\bar{b} := b - \bar{a} := b - \sum_{j=1}^{n} \bar{x}_j a_j$.

ist $\bar{b} \geq 0$, so führe aus:

ist $\bar{c} > z$, so setze $z = \bar{c}$ und $x = \bar{x}$; ist $z = U$, **stopp**

für $j = t+1$ bis n führe aus:

ist $j = q+1$, wähle den Teilbereich $L_i =: [f,g]$ mit höchstem Index und erweitererechts (f,g)

ist $u_j = \bar{c} + \left\lfloor \bar{b} \frac{c_j}{a_j} \right\rfloor \leq z$, return.

setze $\bar{x}_j = 1$ und AusloteU $(KP[r,j])$,

setze $\bar{x}_j = 0$.

sonst führe aus:

für $j = r-1$ bis 1 $(step: -1)$ führe aus:

ist $j = p-1$, wähle den Teilbereich $H_i =: [f,g]$ mit höchstem Index und erweiterelinks (f,g)

ist $u_j = \bar{c} + \left\lfloor \bar{b} \frac{c_j}{a_j} \right\rfloor \leq z$, return

setze $\bar{x}_j = 0$ und AusloteU $(KP[j,t])$,

setze $\bar{x}_j = 1$

Beispiel

Zum besseren Verständnis betrachten wir unser unsortiertes Beispiel:

j	1	2	3	4	5	6	7
c_j	39	5	37	70	10	20	7
a_j	20	3	19	31	6	10	4
e_j	1,950	1,667	1,947	2,258	1,667	2,000	1,750

mit $n = 7$ und $b = 50$. Wir kennen bereits eine geeignete Teilsortierung zu dieser Aufgabe

$$N = C \cup L_1 \quad \text{mit} \quad C = \{4, 6, 1\} \quad \text{und} \quad L_1 = \{3, 5, 2, 7\}$$

mit kritischem Index $k = 3$ (in der Umsortierung) und Dantziglösung $\bar{x}^T = (0, 0, 0, 1, 0, 1, 0)$ mit Gewicht 41 und Nutzen $z = 90$. Es ist $C = [p, q] = [1, 3]$. Gleichzeitig kennen wir aus dem früheren Preprocessing bereits eine gute untere Schranke, $z = 105$, angenommen bei $x^T = (0, 1, 0, 1, 1, 1, 0)$ und die obere Schranke $U = 107$.

Wenden wir auf diese Situation expcore1U an. Wir setzen $\bar{x}^T = (0, 0, 0, 1, 0, 1, 0)$ und rufen

AusloteU $(KP[3, 2])$ auf: Es ist $\bar{c} = 70 + 20 = 90 < 105 = z$ und $\bar{b} = 50 - 31 - 10 = 9 \geq 0$. Setze $j = 3 < 4 = q + 1$. Berechne also $u_3 = \bar{c} + \left\lfloor \bar{b} \frac{c_3}{a3} \right\rfloor = 90 + \lfloor 9 \cdot 1.95 \rfloor = 107 > z = 105$. Setze $\bar{x}_1 = 1$, also $\bar{x}^T = (1, 0, 0, 1, 0, 1, 0)$ und

AusloteU $(KP[3, 3])$: Es ist $\bar{c} = 90+39 = 129$ und $\bar{b} = 9-20 = -11 < 0$. Setze daher $j = 2 > 0 = p - 1$. Berechne $u_2 = \bar{c} + \left\lfloor \bar{b} \frac{c_2}{a_2} \right\rfloor = 129 + \lfloor (-11) \cdot 2.0 \rfloor = 107 > z = 105$. Setze $\bar{x}_6 = 0$, d.h. $\bar{x}^T = (1, 0, 0, 1, 0, 0, 0)$ und

AusloteU $(KP[2, 3])$: Es ist $\bar{c} = 129 - 20 = 109$ und $\bar{b} = -11 + 10 = -1 < 0$. Setze daher $j = 1 > 0 = p - 1$. Berechne $u_1 = \bar{c} + \left\lfloor \bar{b} \frac{c_1}{a_1} \right\rfloor = 109 + \lfloor (-1) \cdot 2.258 \rfloor = 106.0 > z = 105$. Setze $\bar{x}_4 = 0$, d.h. $\bar{x}^T = (1, 0, 0, 0, 0, 0, 0)$

AusloteU $(KP[1, 3])$: Es ist $\bar{c} = 109-70 = 39 < 105 = z$ und $\bar{b} = -1+31 = 30 > 0$. Setze $j = 4 = q + 1$

erweitererechts $(4, 7)$: es ist $7 - 4 \geq 2$. Daher bilde $m = \lfloor (7 + 4)/2 \rfloor = 5$ und lese ab: $e_3 = 1.947$, $e_5 = 1.667$ und $e_7 = 1.750$. Daher muss zunächst die Reihenfolge geändert werden: vertausche in L_1 die Einträge 5 und 7. Dann gilt $L_1 = \{3, 7, 2, 5\}$ (L_1 ist damit bereits richtig sortiert!) Wegen $(7 - 4) > 2$ setze $i = 4$ und $j = 7$. Wir erhöhen nun i systematisch, lassen 3 und 7 passieren und stoppen bei $i = 6$. Nun senken wir j systematisch und stoppen direkt wegen der Gleichheit der beiden letzten Effizienzen bei $j = 6$. Damit gilt $i = j = 6$. Wir müssen erneut in die solange-Schleife und erhalten nun $i = 7$ und $j = 5$. Wegen $i > j$ stoppt die Schleife

hier. Nun setze $L' := L' \cup \{[i,g]\} = \{[7,7]\}$; und $g := i-1 = 6$; Da $[4,6]$ nun bereits sortiert ist, lautet die neue Teilsortierung von N

$$N = C \cup L_1 \quad \text{mit} \quad C = \{4,6,1,3,7,2\} \quad \text{und} \quad L_1 = \{5\} \quad \text{sowie } p = 1,\ q = 6$$

Weiter in $(KP\,[1,3])$: Wir bilden $u_4 = \bar{c} + \left\lfloor \bar{b}\frac{c_4}{a_4} \right\rfloor = 39 + \lfloor 30 \cdot 1.947 \rfloor = 97 < 105 = z$. Daher gehe zurück zu $(KP\,[2,3])$. Setze wieder $\bar{x}_4 = 1$, also $\bar{x}^T = (1,0,0,1,0,0,0)$. Weil hier bereits $j = 1$ galt, gehen wir weiter zurück zu

$(KP\,[3,3])$. Hier setzen wir zunächst $\bar{x}_6 = 1$, d.h. $\bar{x}^T = (1,0,0,1,0,1,0)$ und dann $j = 1 > 0 = p-1$ und berechnen $u_1 = \bar{c} + \left\lfloor \bar{b}\frac{c_1}{a1} \right\rfloor = 129 + \lfloor (-11) \cdot 2.258 \rfloor = 104 < 105 = z$. Damit ist auch dieser Knoten ausgelotet und wir gehen zurück zu

$(KP\,[3,2])$. Hier war zuletzt $j = 3$ gesetzt worden. Also setze $\bar{x}_1 = 0$, d.h. $\bar{x}^T = (0,0,0,1,0,1,0)$ und erhöhe auf $j = 4 < 7 = q+1$. Berechne $u_4 = \bar{c} + \left\lfloor \bar{b}\frac{c_4}{a_4} \right\rfloor = 90 + \lfloor 9 \cdot 1.947 \rfloor = 107 > 105 = z$. Setze daher $\bar{x}_3 = 1$, also $\bar{x}^T = (0,0,1,1,0,1,0)$ und

AusloteU $(KP\,[3,4])$: Es ist $\bar{c} = 90 + 37 = 127$ und $\bar{b} = 9 - 19 = -10 < 0$. Setze daher $j = 2 > p-1 = 0$. Teste $u_2 = \bar{c} + \left\lfloor \bar{b}\frac{c_2}{a_2} \right\rfloor = 127 + \lfloor (-10) \cdot 2.0 \rfloor = 107 > 105 = z$, daher setze $\bar{x}_6 = 0$, also $\bar{x}^T = (0,0,1,1,0,0,0)$ und

AusloteU $(KP\,[2,4])$: Es ist $\bar{c} = 127 - 20 = 107 > 105$ und $\bar{b} = -10 + 10 = 0$, damit ist eine bessere zulässige Lösung gefunden. Setze $z = 107$ und $x^T = \bar{x}^T = (0,0,1,1,0,0,0)$. Da nun $z = U$ gilt, **stopp.**

4.3.4 Reduzierung

Das bisherige Konzept zur Lösung eines unsortierten 0-1 Knapsack Problems sieht also ähnlich zum Balas/Zemel Konzept zunächst eine Teilsortierung vor, dann die Anwendung einer LU Prozedur und schließlich die Lösung des Problems mit einem expandierenden Verfahren. Dies sind drei der vier Strategieteile aus dem Balas/Zemel Konzept. Aber auch der fehlende Teil des Konzepts, die Reduzierung, kann erfolgreich intergiert werden.

Im ersten Schätzkern C und im Laufe von dessen Erweiterung kann immer wieder geprüft werden, ob eine Fixierung der aktuell betrachteten Variablen auf eine von der Dantziglösung abweichenden Term sich lohnt, bzw. ob diese Fixierung näher untersucht werden muss oder nicht. Ein *globales* Kriterium für eine mögliche Vorab-Fixierung, das unserem Satz 2.1 entspringt und nur den kritischen Index k der gegebenen Aufgabe verwendet, wäre

$$\bar{c} + c_j + \left\lfloor \left(\bar{b} - a_j\right) \frac{c_k}{a_k} \right\rfloor \leq z$$

wenn sich die zu testende Variablen rechts vom kritischen Index befindet und

$$\bar{c} - c_j + \left\lfloor \left(\bar{b} + a_j\right) \frac{c_k}{a_k} \right\rfloor \leq z$$

sonst. Ist dieses Kriterium erfüllt, so kann diese Variable im Verfahren direkt fixiert werden. Zu beachten ist dabei, dass das Ergebnis der Abfrage sehr wohl vom Zeitpunkt der Abfrage abhängt. Mit fortschreitendem Verfahren wird die untere Schranke schärfer, eine Fixierung also wahrscheinlicher.

In den bisherigen Verfahren gibt es eine solche Abfrage nicht. Hier müsste man vor jeder Verzweigung testen, ob das Kriterium erfüllt ist. Allerdings würde wegen der rekursiven Struktur bzgl ein un derselben Variablen vielfach das Kriterium überprüft werden müssen.

Man hat zwei Möglichkeiten, mit dieser Mehrfachanwendung umzugehen:

1. Man führt die Abfrage ein erstes Mal aus und merkt sich das Ergebnis. Dann kann bei späteren eventuell notwendigen Abfragen nur dieses Ergebnis abgefragt werden. Dabei gehen einem aber diejenigen Fixierungen verloren, die sich aus einer schärferen unteren Schranke ergeben würden. Besser wäre:

2. Man überprüft das Kriterium zunächst für alle j aus C und dann bei jeder Erweiterung des Kerns. Stellt man fest, dass eine Variable mit dem Index j nach diesem Kriterium bereits fixiert werden kann, so schiebt man sie durch Vertauschen an den Rand von C bzw in einen benachbarten Teilbereich. Sind alle Variablen eines Teilbereichs untersucht und die betreffenden Variablen nach außen geschoben, so ordnet man sie dem nächsten äußeren Nachbarbereich zu. Dort werden sie bei der nächsten Erweiterung zwar *erneut getestet* und nach außen geschoben, aber wegen der vermutlich sehr eingeschränkten Anzahl der Erweiterungen ist dies in der Regel günstiger als die Bearbeitung mit der ersten Methode.

Zwei einfache Prozduren, die diese Verschiebung nach außen leisten, sind die folgenden.

Für die Verschiebung nach Rechts setzen wir voraus, dass der Teilbereich $L_s = [i, j]$ entweder den Bereich rechts von $i = k$ in C oder einen anderen Teilbereich L_s, $s \geq 1$ rechts von C beschreibt. In beiden Fällen sei L_{s-1} der unmittelbar rechts benachbarte Teilbereich von L_s. Gilt $s = 1$, setze $L_{s-1} = L_0 = \varnothing$

verschiebenachrechts (i,j) setze $m=i$ und $l=j$

solange $m \leq l$ führe aus:

ist $\bar{c}+c_m+\left\lfloor \left(\bar{b}-a_m\right)\frac{c_k}{a_k}\right\rfloor \leq z$, vertausche die Inhalte von m und l und setze $l=l-1$

sonst setze $m=m+1$

Vereinige $[l+1,j]$ geordnet mit L_{s-1}

ist $[i,l]$ nicht leer, setze $L_s=[i,l]$

Für die Verschiebung nach Links setzen wir voraus, dass der Teilbereich $H_s=[i,j]$ entweder den Bereich vom linken Rand von C bis $j=k-1$ ist oder einen anderen Teilbereich $H_s,\ s\geq 1$,links von C beschreibt. In beiden Fällen sei H_{s-1} der unmittelbar links benachbarte Teilbereich von H_s. Gilt $s=1$, setze $H_{s-1}=H_0=\varnothing$

verschiebenachlinks (i,j) setze $m=i$ und $l=j$

solange $m \leq l$ führe aus:

ist $\bar{c}-c_l+\left\lfloor \left(\bar{b}+a_l\right)\frac{c_k}{a_k}\right\rfloor \leq z$, swap (m,l) und setze $m=m+1$

sonst setze $l=l-1$

Vereinige $[i,m-1]$ geordnet mit H_{s-1},

ist $[m,j]$ leer, setze $H_s=[m,j]$

Beispiel In unserem Beispiel haben wir anfänglich den linken Teil von $C=[1,3]$ zu testen, da das break item am rechten Rand sitzt. Wir führen also verschiebenachlinks $(1,2)$ durch. Dabei gilt $C=\{4,6,1\}$, $\bar{c}=90,\ \bar{b}=9,\ e_k=1.95$, wobei k am rechten Rand von C liegt. Setze $H_0=\varnothing$.

setze $m=1$ und $l=2$. Wir beginnen die solange-Schleife. Für $m=1$ ergibt sich

$\bar{c}-c_l+\left\lfloor \left(\bar{b}+a_l\right)\frac{c_k}{a_k}\right\rfloor = 90-20+\lfloor(9+10)\cdot 1.95\rfloor = 107 > 105$, also setze $l=1$. Prüfe nun

$\bar{c}-c_l+\left\lfloor \left(\bar{b}+a_l\right)\frac{c_k}{a_k}\right\rfloor = 90-70+\lfloor(9+31)\cdot 1.95\rfloor = 98 < 105$, also tausche die Inhalte von 1 und 1. Dies verändert C nicht. Setze $m=2$.

Hier endet die Solange-Schleife. Es ist $[i,m-1]=[1,1]$. Daher setze $H_0=\{4\}$.

Es ist $[l+1,j]=[2,2]$, also ist nun $C=\{6,1\}$.

Im Rahmen der Anwendung von expcore1U hatten wir auch C über den Bereich $L_1=[4,7]=\{3,5,2,7\}$ der Prozedur erweitererechts $(4,7)$ zu unterwerfen. Bei

einbezogener Reduzierung ist zuvor L_1 der Prozedur verschiebenachrechts $(4, 7)$ zu unterwerfen. Dabei gilt wieder $\bar{c} = 90,\ \bar{b} = 9,\ e_k = 1.95$.

Setze $m = 4$ und $l = 7$. Wir steigen in die solange-Schleife ein.

$\bar{c} + c_m + \left\lfloor \left(\bar{b} - a_m\right) \frac{c_k}{a_k} \right\rfloor = 90 + 37 + \lfloor (9 - 19) \cdot 1.95 \rfloor = 107 > 105 = z$. Setze also $m = 5$.

$\bar{c} + c_m + \left\lfloor \left(\bar{b} - a_m\right) \frac{c_k}{a_k} \right\rfloor = 90 + 10 + \lfloor (9 - 6) \cdot 1.95 \rfloor = 105 \leq 105 = z$. Vertausche also die Plätze 5 und 7: es gilt nun $L_1 = \{3, 7, 2, 5\}$. Setze dann $l = 6$.

$\bar{c} + c_m + \left\lfloor \left(\bar{b} - a_m\right) \frac{c_k}{a_k} \right\rfloor = 90 + 7 + \lfloor (9 - 4) \cdot 1.95 \rfloor = 106 > 105 = z$. Setze also $m = 6$.

$\bar{c} + c_m + \left\lfloor \left(\bar{b} - a_m\right) \frac{c_k}{a_k} \right\rfloor = 90 + 5 + \lfloor (9 - 3) \cdot 1.95 \rfloor = 106 > 105 = z$. Setze also $m = 7$.

Damit endet die Solange-Schleife. Es ist $[l + 1, j] = [7, 7]$. Da L_1 das äußere L war, setze nun $L_0 = \{5\}$. Setze $L_1 = [i, m - 1] = [4, 6] = \{3, 7, 2\}$.

4.3.5 Das Verfahren PEV

Insgesamt hat sich also ein Vorgehensweise ergeben, die im weiteren Sinne ebenfalls vom Balas/Zemel Typ ist:

1. Zunächst wird PPC angewendet, um eine Teilsortierung zu erreichen.

$$N = H_1 \cup H_2 \cup ... \cup H_{s_1} \cup C \cup L_{s_2} \cup ... \cup L_2 \cup L_1$$

2. Danach wird C sortiert und eine LU-Prozedur angewendet, die eine gute zulässige Lösung x mit Zielwert z sowie eine obere Schranke U liefert.

3. Als Nächstes wird C reduziert und bereits fixierbare Items nach außen verschoben.

4. Nun wird ein expandierendes Verfahren angewendet, wobei C laufend erweitert und vor jeder Erweiterung zunächst reduziert wird.

Abschließend formulieren wir noch einmal das erarbeitete Verfahren im Überblick: Wir unterstellen, dass eine Teilsortierung vorgenommen und mit einer dazu verträglichen LU Prozedur eine obere und eine untere Schranke berechnet wurden.

Prozedur PEV (Pisingers Expandierendes Verfahren)

Input: (KP_{01}), kritischer Index k, zulässige Lösung x mit Zielwert z,teilsortierte Zerlegung $N = H_1 \cup ... \cup H_{s_1} \cup C = [p,q] \cup L_{s_2} \cup ... \cup L_1$

Output: zulässige Lösung x mit Zielwert z.

setze $\bar{x}$ gleich der Dantziglösung.

setze $c_0 := 1;\ a_0 := 0;\ c_{n+1} := 0;\ a_{n+1} := 1;$

berechne $\hat{c} := \sum_{j=1}^{k-1} c_j \quad \hat{b} := b - \bar{a} := b - \sum_{j=1}^{k-1} a_j$ und setze $\hat{b}_0 = b$.

ist $\hat{c} > z$ und $\hat{b} \geq 0$, so setze $z = \hat{c}$ und $x = \bar{x}$. Gilt nun $z = U$, **stopp**

Löse $\left(KP\left[k, k-1, \hat{c}, \hat{b}, \hat{b}_0\right]\right)$.

Prozedur: Löse $\left(KP\left[r, t, \hat{c}, \hat{b}, \hat{b}_0\right]\right)$

Setze $\bar{c} = \hat{c}$ und $\bar{b} = \hat{b}$ und $\bar{b}_0 = \hat{b}_0$.

ist $\bar{b} \geq 0$, so führe aus: % Fall 1

für $j = t+1$ bis n führe aus:

ist $j = q+1$, wähle den Teilbereich $L_i =: [f,g]$ mit höchstem Index und erweitererechts (f,g)

bilde $\hat{c} = \bar{c} + c_j,\ \hat{b} = \bar{b} - a_j$ und $\hat{b}_0 = \bar{b}_0 - a_j$.

ist $\hat{b} \geq 0$ und $\hat{c} > z$, so setze $z = \hat{c}$ und $x = \bar{x}$ sowie $x_j = 1$. Gilt nun $z = U$, **stopp**

ist $\hat{b}_0 \geq 0$, so führe aus:

setze $\bar{u}_j = \hat{c} + \left\lfloor \hat{b} \frac{c_{j+1}}{a_{j+1}} \right\rfloor$, falls $\hat{b} \geq 0$, sonst $\bar{u}_j = \hat{c} + \left\lfloor \hat{b} \frac{c_{r-1}}{a_{r-1}} \right\rfloor$

ist $\bar{u}_j > z$, so führe aus:

setze $\bar{x}_j = 1$ und Löse $\left(KP\left[r, j, \hat{c}, \hat{b}, \hat{b}_0\right]\right)$

setze $\bar{x}_j = 0$

sonst: ist $u_{j+1} = \bar{c} + \left\lfloor \bar{b} \frac{c_{j+1}}{a_{j+1}} \right\rfloor \leq z$, return

sonst führe aus: % Fall 2

für $j = r-1$ bis 1 $(step: -1)$ führe aus:

ist $j = p-1$, wähle den Teilbereich $H_i =: [f,g]$ mit höchstem Index und erweiterelinks (f,g)

bilde $\hat{c} = \bar{c} - c_j$, $\hat{b} = \bar{b} + a_j$ und $\hat{b}_0 = \bar{b}_0$, falls $j = r - 1$, sonst $\hat{b}_0 = \hat{b}_0 - a_{j+1}$

ist $\hat{b}_0 < 0$, return

ist $\hat{b} \geq 0$, $\hat{c} > z$, so setze $z = \hat{c}$ und $x = \bar{x}$ sowie $x_j = 0$. Gilt nun $z = U$, **stopp**

setze $\bar{u}_j = \hat{c} + \left\lfloor \hat{b} \frac{c_{t+1}}{a_{t+1}} \right\rfloor$, falls $\hat{b} \geq 0$, sonst $\bar{u}_j = \hat{c} + \left\lfloor \hat{b} \frac{c_{j-1}}{a_{j-1}} \right\rfloor$

ist $\bar{u}_j > z$, so führe aus:

setze $\bar{x}_j = 0$ und Löse $\left(KP \left[r, t, \hat{c}, \hat{b}, \hat{b}_0 \right] \right)$

setze $\bar{x}_j = 1$

sonst: ist $u_{j-1} = \bar{c} + \left\lfloor \bar{b} \frac{c_{j-1}}{a_{j-1}} \right\rfloor \leq z$, return.

procedure erweitererechts (f, g)

Verschiebenachrechts (f, g)

solange $(g - f) \geq 2$ führe aus

setze $m := \lfloor (f + g)/2 \rfloor$ und tausche die Inhalte der Stellen f, m, g so, dass die Effizienzen zu diesen drei Stellen abfallend sind.

setze $i := f$; $j := l$;

wiederhole

wiederhole $i := i + 1$; bis die Effizienz der Stelle i kleiner gleich der der Stelle m ist;

wiederhole $j := j - 1$; bis die Effizienz der Stelle j größer gleich der der Stelle m ist;

ist $(i < j)$ dann vertausche die Inhalte von i und j;

bis $i > j$

setze geordnet $L' := L' \cup \{[i, g]\}$; $g := i - 1$;

ist $(g - f = 1)$, so tausche ggf. die Inhalte der Stellen f, g so, dass die Effizienzen zu diesen Stellen abfallend sind.

setze $q = g$ und vereinige geordnet L und L'.

procedure erweiterelinks (f, g)

Verschiebenachlinks (f, g)

solange $(g - f) \geq 2$ führe aus

setze $m := \lfloor (f + g)/2 \rfloor$ und tausche die Inhalte der Stellen f, m, g so, dass die Effizienzen zu diesen drei Stellen abfallend sind.

setze $i := f;\ j := l;$

wiederhole

wiederhole $i := i + 1;$ bis die Effizienz der Stelle i kleiner gleich der der Stelle m ist;

wiederhole $j := j - 1;$ bis die Effizienz der Stelle j größer gleich der der Stelle m ist;

ist $(i < j)$ dann vertausche die Inhalte von i und j;

bis $i > j$

setze geordnet $H' := H' \cup \{[f, i - 1]\};\ f := i;$

ist $(g - f = 1)$, und tausche ggf. die Inhalte der Stellen f, g so, dass die Effizienzen zu diesen Stellen abfallend sind.

setze $p = f$ und vereinige geordnet H und H'.

verschiebenachrechts (i, j) setze $m = i$ und $l = j$

solange $m \leq l$ führe aus:

ist $\bar{c} + c_m + \left\lfloor \left(\bar{b} - a_m\right) \frac{c_k}{a_k} \right\rfloor \leq z$, vertausche die Inhalte von m und l und setze $l = l - 1$

sonst setze $m = m + 1$

Vereinige $[l + 1, j]$ geordnet mit L_{s-1}

ist $[i, l]$ nicht leer, setze $L_s = [i, l]$

verschiebenachlinks (i, j) setze $m = i$ und $l = j$

solange $m \leq l$ führe aus:

ist $\bar{c} - c_l + \left\lfloor \left(\bar{b} + a_l\right) \frac{c_k}{a_k} \right\rfloor \leq z$, swap (m, l) und setze $m = m + 1$

sonst setze $l = l - 1$

Vereinige $[i, m - 1]$ geordnet mit H_{s-1},

ist $[m, j]$ leer, setze $H_s = [m, j]$

Bemerkung: Fügt man in das Verfahren noch die bereits oben angedeutete Aktualisierung der globalen oberen Schranke ein, so wird es noch einmal beschleunigt (Übungsaufgabe)

4.4 Zentrale Dynamische Optimierung

Das Verfahren PEV hat in der von Pisinger getesteten Version, stärker noch als MT2, Schwierigkeiten beim Lösen von stark korrelierten Aufgabenstellungen. Auf der anderen Seite haben wir gesehen, dass Methoden der dynamischen Optimierung, zumindest bei niederdimensionalen Beispielen, sehr wohl in der Lage sind, stark korrelierte Aufgaben in akzeptabler Zeit zu lösen. Leider zeigen die numerischen Experimente, dass auch hier die Bearbeitungzeit mit steigender Dimension sehr stark ansteigt, so dass zu befürchten ist, dass bei mittleren und großen Dimensionen auch mit der Methode DPT kein Erfolg zu erzielen ist.

Eine weitere Idee von Pisinger kann hier Abhilfe schaffen. Dieser schlägt vor, das Vorgehen bei der dynamischen Optimierung dahingehend abzuwandeln, dass die Variablen nicht in der Reihenfolge $x_1, x_2, ...$ abgearbeitet werden, sondern dass bei der kritischen Variablen begonnen wird und dann abwechselnd der Bereich der schon bearbeiteten Variablen links und rechts erweitert wird. Wir wollen uns genauer anschauen, wie dies abläuft.

4.4.1 Start mit dem kritischen Index

Zunächst einmal stellt eine Veränderung der Reihenfolge der Variablen bei Einbeziehung in den Prozess der dynamischen Optimierung kein Problem da, im Gegenteil, wir hatte bereits weiter oben betont, dass eine Vorabsortierung der Variablen bei der dynamischen Optimierung nicht nötig ist und die Variablen beliebig (un-)sortiert sein dürfen.

Wir wollen aber wie folgt abändernd in die Anwendung der Bellmanschen Rekusion eingreifen. Bisher setzte man die Aufgabe (KP_{01}) mit den Zusatzrestriktionen $x_{m+1} = 0, ..., x_n = 0$ als bereits gelöst voraus und gab dann die Variable x_{m+1} frei, d.h. ließ sie auch den Wert 1 annehmen, dies alles bei beliebig vorgegebener Kapazität $\hat{b} \in \mathbf{N}$ des Rucksacks. Die optimalen Zielfunktionswerte $f_{m+1}\left[\hat{b}\right]$ der neuen Aufgaben ergaben sich dann nach der Bellman-Formel

$$f_{m+1}\left[\hat{b}\right] = \max\left\{f_m\left[\hat{b}\right],\ f_m\left[\hat{b} - a_{m+1}\right] + c_{m+1}\right\}, \quad \text{falls } \left(\hat{b} - a_{m+1}\right) \geq 0$$

für alle $\hat{b} \in \mathbf{N}$, wobei jeweils die Möglichkeit, $x_{m+1} = 1$ zu setzen mit der bereits bekannten Möglichkeit, $x_{m+1} = 0$ zu setzen, verglichen wird.

Nun betrachten wir Testaufgaben der Form $\left(KP\left[r,t\right],\hat{b}\right)$, die aus (KP_{01}) mit den Zusatzrestiktionen $x_1 = 1, ..., x_{r-1} = 1$ und $x_{t+1} = 0, ..., x_n = 0$ entstehen. Dabei ist $[r,t]$ ein " Intervall" in $\mathbf{N}$, das den kritischen Index k enthält. $\hat{b}$ gibt wieder die Kapazität des Rucksacks an. Wenn wir dann die Variable $(t+1)$ in den dynamischen Prozess neu aufnehmen wollen, so ergeben sich die Zielfunktionswerte $f_{r,t+1}\left[\hat{b}\right]$ der neuen Aufgaben wie im bisherigen Fall durch die Formel

$$f_{r,t+1}\left[\hat{b}\right] = \max\left\{f_{r,t}\left[\hat{b}\right],\ f_{r,t}\left[\hat{b}-a_{t+1}\right]+c_{t+1}\right\}, \quad \text{falls } \left(\hat{b}-a_{t+1}\right) \geq 0$$

Wenn wir aber die Variable $(r-1)$ in den dynamischen Prozess neu aufnehmen wollen, so ergeben sich die Zielfunktionswerte $f_{r-1,t}\left[\hat{b}\right]$ der neuen Aufgaben analog durch die Formel

$$f_{r-1,t}\left[\hat{b}\right] = \max\left\{f_{r,t}\left[\hat{b}\right],\ f_{r,t}\left[\hat{b}+a_{r-1}\right]-c_{r-1}\right\}$$

wobei zunächst die Möglichkeit, $x_{r-1} = 1$ zu setzen abgelesen und dann die Möglichkeit, $x_{r-1} = 0$ zu setzen, neu berechnet wird.

Ausgangspunkt der Berechnungen ist natürlich für die Aufgabe $\left(KP\left[k,k-1\right],\hat{b}\right)$

$$f_{k,k-1}\left[\hat{b}\right] = \begin{array}{ll} -\infty, & \text{falls } \hat{b} = 0, ..., \bar{a}_{k-1}-1 \\ \bar{c}_{k-1}, & \text{falls } \hat{b} \geq \bar{a}_{k-1} \end{array}$$

für alle $\hat{b} \in \mathbf{N}$, wobei wie zuvor für alle $i \in \mathbf{N}_0$

$$\bar{a}_i = \sum_{j=1}^{i} a_j \quad \text{und} \quad \bar{c}_i = \sum_{j=1}^{i} c_j$$

gesetzt wurde. Der Nutzen $-\infty$ drückt aus, dass dieser Zustand gar nicht möglich ist.

Es ist klar, dass mit dieser abgeänderten Bellmanschen Rekursion die gegebene Aufgabe (KP_{01}) bei beliebiger Reihenfolge der Neuaufnahme an $[r,t]$ angrenzender Variabler vollständig gelöst werden kann, wenn gleichzeitig die jeweils dem optimalen Zielwert zugrunde liegende zulässige Lösung mitgeführt wird.

4.4.2 Ausloten der Zustände

Zunächst mal halten wir fest, dass wir, wie gesehen, auf jeder Stufe immer nur die undominierten Zustände betrachten müssen, ohne dadurch einen optimalen Zustand der Gesamtaufgabe zu verlieren. Wir bezeichnen die Menge der noch zu betrachtenden Zustände der Stufe $[r,t]$ mit $S\left[r,t\right]$.

Auf der anderen Seite haben wir bei den zuletzt besprochenen B&B Verfahren gesehen, wie wirkungsvoll Auslotung durch Vergleich von oberen und unteren Schranke ist. Unsere oben beschriebene neue Konstrukion der Dynamischen Optimierung ist immer noch so geartet, dass wir auf jeder Stufe $[r,t]$ des Prozesses Nutzenwerte, die zu zulässigen Lösungen der Aufgabe (KP_{01}) gehören, ablesen können: dies sind die Zustände mit $\hat{a} \leq b$, denn die implizit gleich eins gesetzten Variablen tragen ja zum Gewicht des Zustands mit bei. Der beste dieser Werte bildet eine untere Schranke z.

Obere Schranken sind nicht so einfach zu haben. Unsere bisherigen Abschätzungen nach oben haben alle die Vollsortierung oder zumindest eine kontrollierte Teilsortierung vorausgesetzt. Wir wollen aber darauf zurückgreifen und **unterstellen bis auf weiteres, dass die Aufgabe vollsortiert sei.** Dabei hoffen wir gemäß unserer bisherigen Erfahrung, dass im gegenteiligen Falle bei Bedarf eine vom kritischen Index nach außen wachsende Sortierung zu ereichen sei.

Liegt also Vollsortierung vor, so haben die angesprochenen zulässigen Lösungen eine besonders gute Chance, gute untere Schranken zu sein. Gleichzeitig können wir wie schon im B&B Verfahren eine gute obere Schranke angeben für alle zulässigen Lösungen von (KP_{01}), die auf dem bereits untersuchten Teilbereich $[r,t]$ einen ausgesuchten Zustand $(\hat{a},\hat{c})$ darstellen:

$$\begin{aligned} u &= \hat{c} + (b-\hat{a})\,\frac{c_{t+1}}{a_{t+1}}, \quad \text{falls } (b-\hat{a}) \geq 0 \quad \text{bzw.} \\ u &= \hat{c} + (b-\hat{a})\,\frac{c_{r-1}}{a_{r-1}}, \quad \text{falls } (b-\hat{a}) < 0 \end{aligned} \tag{4.1}$$

Da wir bei der dynamischen Optimierung unterstellen können, dass wir auf der Stufe $[r,t]$ alle undominierten Zustande kennen, können wir für jeden der Zustände prüfen, ob $u > z$ gilt, also überhaupt eine Chance besteht, dass der betrachtete Zustand bei Fortführung der dynamischen Optimierung zu einer besseren Lösung führt als die, die aktuell als beste bekannt ist. Ist das nicht der Fall, kann der betrachtete Zustand vom weiteren Verfahren ausgeschlossen werden! Auf diese Weise kann die Menge der noch zu betrachtenden Zustände immer weiter ausgedünnt werden und es ist zu hoffen, dass die dynamische Optimierung gar nicht bis zu Ende durchgeführt werden muss.

4.4.3 Verfahren für vorsortierte Probleme

Bisher hat unser Verfahren die folgende Form: Wir betrachten immer nur die nichtdominierten Zustände und prüfen die Undominiertheit wie im Verfahren DP2, indem wir die möglichen Zustände bzgl. aufsteigender Gewichte anordnen. Ergibt sich dabei ein nicht-ansteigendes Gewicht, so ist der betreffende Zustand dominiert. Die zum Zustand zugehörige Lösung x notieren wir in der Form $\eta := \{j_1, j_2, ...\}$ von Indizes, deren Variablen von der Dantziglösung abweichen. Wir notieren die Zustände unter Einbeziehung der Information über die zugehörige Lösung in der Form (Gewicht, Nutzen, Lösung), allgemein geschrieben als $(g, h.\eta)$. Wir skizzieren die beschriebene Vorgehensweise als

Methode ZDO (Zentrale Dynamische Optimierung)

Input: (KP_{01}) , der kritische Index k, eine zulässige Lösung mit Zielwert z

Output: eine optimale Lösung x mit Zielwert z

1. **Start:** Setze $[r, t] = [k, k-1]$ und setze die geordnete Menge S der Zustände fest zu $S = \langle(\bar{a}, \bar{c}, \{\})\rangle$ mit $\bar{a} = \sum_{j=1}^{k-1} a_j$ und $\bar{c} = \sum_{j=1}^{k-1} c_j$. Wähle $s = k$ als rechts angrenzenden Index an $[r, t]$. Setze $w = a_s$ und $p = c_s$ und $S' = S$.

2. **Neue Zustände**: Berechne die Zustände des um s erweiterten Intervalls $[r, t]$, indem S'' gebildet wird aus den Zuständen $(g'', h'', \eta'') = (g + a, h + p, \eta \cup \{s\})$ für alle Zustände (g, h, η) aus S'. Bilde dann $S = S' \cup S''$ als nach Gewichten geordnete Vereinigung, bei der dominierte Zustände entfernt werden. Ergibt sich dabei ein besserer zulässiger Zustand als der beste bereits bekannte, aktualisiere z und speichere das zugehörige η.

3. **Auslotung:** Nenne das betrachtete Intervall einschließlich s wieder $[r, t]$ und überprüfe die Zustände aus S auf Auslotung mittels der Formeln (4.1). Entferne die ausgeloteten Zustände aus S.

4. **Abbruch:** Ist $S = \varnothing$, stopp, (z und η beschreiben die optimale Lösung)

   Andernfalls wähle eine an $[r, t]$ angrenzende Variable s. Geht dies nicht, stopp, (z und η beschreiben die optimale Lösung)

   Setze $S' = S$ und $w = a_s, p = c_s$, sofern $s = t + 1$, sonst setze $w = -a_s$ und $p = -c_s$ und gehe zu 2.

Beispiel

Benutzen wir wieder unser sortiertes Beispiel *nach Reduzierung* zur Demonstration von Prozedur ZDP.

Gegenstand j	2	3	4	5
Nutzen c_j	8	6	10	12
Gewicht a_j	2	2	4	6

mit $b = 11$

und Vorkosten $\hat{z} = 8$, Vorgewicht $\hat{a} = 1$ bei $x_1 = 1$. Die kritische Variable ist die mit dem Index 5.

Wir beginnen mit der Berechnung von $S\,[5, 4]$ und geben dort nur den möglichen Zustand an: Gewicht 8 und Nutzen 24. Hierdurch hat man einen ersten besten zulässigen Zustand, d.h. wir setzen $z = 24$. Die zugehörige zulässige Lösung lautet $\{\}$. Also ergibt sich

$$S = \langle (8, 24, \{\}) \rangle$$

Wir gehen über zu $[5, 5]$: hier erhalten wir einen neuen zusätzlichen undominierten Zustand, insgesamt

$$S = \langle (8, 24, \{\})\,;\ (14, 36, \{5\}) \rangle$$

Wir überprüfen Auslotung:

Nach der Formel $u = \hat{c} + (b - \hat{a})\,\frac{c_{t+1}}{a_{t+1}}$ beim ersten Zustand. Hier setzen wir $c_6 = 0$, $a_6 = 1$. Wegen $u = 24 \leq 24 = z$ ergibt sich Auslotung.

Nach der Formel $u = \hat{c} + (b - \hat{a})\,\frac{c_{r-1}}{a_{r-1}}$ für den zweiten Zustand. $u = \hat{c} + (b - \hat{a})\,\frac{c_4}{a_4} = 36 + \left\lfloor (11 - 14)\,\frac{10}{4} \right\rfloor = 28.0$

Dieser Zustand lebt also wegen $u = 28 > 24 = z$ weiter und wir gehen über zu $[4, 5]$. Es ergeben sich die undominierten Zustände

$$S = \langle (10, 26, \{5, 4\})\,;\ (14, 36, \{5\}) \rangle$$

Wir erhalten einen neuen besten zulässigen Zustand: $(10, 26, \{5, 4\})$, was z auf $z = 26$ heraufsetzt. Test auf Auslotung:

zu $(14, 36, \{5\})$: $u = \hat{c} + (b - \hat{a})\,\frac{c_{r-1}}{a_{r-1}} = 36 + \left\lfloor (11 - 14)\,\frac{6}{2} \right\rfloor = 27.0 > 26 = z$

zu $(10, 26, \{5, 4\})$: $u = \hat{c} + (b - \hat{a})\,\frac{c_{t+1}}{a_{t+1}} = 26 + \left\lfloor (11 - 10)\,\frac{0}{1} \right\rfloor = 26.0 \leq 26 = z$

Es bleibt der Zustand $(14, 36, \{5\})$ zu betrachten, wir gehen über zu $[3, 5]$. Es ergeben sich die undominierten Zustände

$$S = \langle (12, 30, \{5, 3\})\,;\ (14, 36, \{5\}) \rangle$$

Prüfen wir die Auslotung:

zu $(14, 36, 15) : u = \hat{c} + (b - \hat{a}) \frac{c_{r-1}}{a_{r-1}} = 36 + \lfloor (11 - 14) \frac{8}{2} \rfloor = 24.0 < 26 = z$

zu $(12, 30, 13) : u = \hat{c} + (b - \hat{a}) \frac{c_{r-1}}{a_{r-1}} = 30 + \lfloor (11 - 12) \frac{8}{2} \rfloor = 26.0 \leq 26 = z.$

Da alle Zustände ausgelotet sind, endet das Verfahren, noch bevor die Variable x_2 einbezogen wird.

Optimaler Zustand ist $(10, 26, \{5, 4\})$. Unter Berücksichtigung von Vorgewicht und Vornutzen ergibt sich für (KP_{01}) die Optimallösung $x^T = (1, 1, 1, 0, 1)$ mit Gewicht 11 und Nutzen 34.

4.4.4 Ausweitung auf den unsortierten Fall

Eine Erweiterung auf den unsortierten Fall ist nach den Ausführungen im letzten Abschnitt natürlich kein Problem, es wird die gleiche Strategie angewendet wie bei PEV:

1. Zunächst wird PPC angewendet, um eine Teilsortierung zu erreichen.

$$N = H_1 \cup H_2 \cup ... \cup H_{s_1} \cup C \cup L_{s_2} \cup ... \cup L_2 \cup L_1$$

2. Danach wird C sortiert und eine LU-Prozedur angewendet, die eine gute zulässige Lösung x mit Zielwert z sowie eine obere Schranke U liefert.

3. Als Nächstes wird C reduziert und bereits fixierbare Items nach außen verschoben.

4. Nun wird ein ZDP Verfahren angewendet, wobei C laufend erweitert werden muss, wobei vor jeder Erweiterung zunächst reduziert wird.

Wir formulieren das resulierende Verfahren ZDP' in einer etwas näher an der Implementierung liegen Form:

Methode ZDP

Input: (KP_{01}), kritischer Index k, zulässige Lösung x mit Zielwert z, obere Schranke U, teilsortierte Zerlegung $N = H_1 \cup ... \cup H_{s_1} \cup C = [p, q] \cup L_{s_2} \cup ... \cup L_1$

Output: eine optimale Lösung x mit Zielwert z

1. **Start:** Setze $[r,t] = [k, k-1]$ und setze die geordnete Menge S der Zustände fest zu $S = \langle(\bar{a}, \bar{c}, \{\})\rangle$ mit $\bar{a} = \sum_{j=1}^{k-1} a_j$ und $\bar{c} = \sum_{j=1}^{k-1} c_j$. Wähle $s = k$ als angrenzenden Index an $[r,t]$.

   Ist $k+1$ nicht in C enthalten, so führe aus:

   Bezeichne mit $[f,g]$ den Bereich L_{s_2}. verschiebenachrechts (f,g)

   erweitererechts $(f, m-1)$

   Ist $k-1$ nicht in C enthalten, so führe aus:

   Bezeichne mit $[f,g]$ den Bereich H_{s_1}. verschiebenachlinks (f,g)

   erweiterelinks $(m-1, g)$

   Setze $w = a_s$ und $p = c_s$ und $S' = S$.

2. **Neue Zustände**: Berechne die Zustände des um s erweiterten Intervalls $[r,t]$, indem S'' gebildet wird aus den Zuständen $(g'', h'', \eta'') = (g + a, h + p, \eta \cup \{s\})$ für alle Zustände (g, h, η) aus S'. Bilde dann $S = S' \cup S''$ als nach Gewichten geordnete Vereinigung, bei der dominierte Zustände entfernt werden. Ergibt sich dabei ein besserer zulässiger Zustand als der beste bereits bekannte, aktualisiere z und speichere das zugehörige η.

3. **Auslotung:** Wir nennen das betrachtete Inervall einschließlich s wieder $[r,t]$ und überprüfe die Zustände aus S auf Auslotung mittels der Formeln (4.1). Entferne die ausgeloteten Zustände aus S.

4. **Abbruch:** Ist $S = \varnothing$, stopp, (z und η beschreiben die optimale Lösung)

   Andernfalls wähle eine an $[r,t]$ angrenzende Variable s. Geht dies nicht, stopp, (z und η beschreiben die optimale Lösung)

   Setze $S' = S$ und $w = a_s, p = c_s$, sofern $s = t+1$, sonst setze $w = -a_s$ und $p = -c_s$ und gehe zu 2.

4.5 Erfahrungsbericht

Pisinger (1995) hat seine Verfahren expknap (einschließlich PPC, Reduzierung und Erweiterung) sowie minknap ausführlich getestet. Wir greifen diese Testergebnisse hier auf.

4.5.1 Ergebnisse zu expknap

Die Testaufgabe unterteilen sich genau wie letzen Kapitel in unterschiedlich strukturierte Aufgaben unterschiedlicher Dimension. Diesmal wurden pro Aufgabenstruktur 50 Aufgaben getestet und der jeweilige Durchschnitt angegeben. Die Verfahren wurden in ANSI-C programmiert und auf einer HP9000/730 getestet. Laufzeiten von über einer Stunde wurden mit "-" markiert. Die definitionen für unkorrelierte, schwach korrelierte, stark korreliert Aufgaben sind diegleichen wie zuvor. Die Kapazität ist auf die Hälfte des Gesamtgewichts festgelegt.

Sortierung

Zunächst gibt er eine Tabelle an, in der eine Vollsortierung (mit Quicksort) gegen die Teilsortierung PPC getestet wird. Dazu werden unkorrelierte 0-1 Knapsack Probleme erzeugt und die Durchschnittszeiten in Sekunden angegeben

n	100	500	1000	5000	10000	50000	100000
PPC	0,000	0,000	0,001	0,004	0,009	0,057	0,128
Quicksort	0,001	0,002	0,004	0,026	0,055	0,361	0,834

Man erkennt den linearen Anstieg der Laufzeit der Teilsortierung, während die Laufzeit von Quicksort stärker ansteigt.

Reduktion

In der nächsten Tabelle wird der Gebrauch der Reduzierungskriterien während des Verfahrens gezählt und als prozentualer Anteil von n angegeben. Zu beachten ist dabei, dass trotz Verschiebung nach außen die fixierbaren items mehrfach auf Reduzierung bzw. Verschiebung getestet werden, so das Prozentsätze über 100% auftreten können.

unkorreliert

r\n	100	500	1000	5000	10000	50000	100000
100	82	51	29	1	0	0	0
1000	87	84	88	91	74	13	4
10000	93	95	97	97	101	97	96

schwach korreliert

r\n	100	500	1000	5000	10000	50000	100000
100	83	19	20	1	0	0	0
1000	140	123	112	30	10	3	3
10000	128	121	120	110	110	92	12

Stark korreliert

r\n	100	500	1000	5000	10000	50000	100000
100	130	-	-	-	-	-	-
1000	198	-	-	-	-	-	-
10000	221	-	-	-	-	-	-

Subset sums

r\n	100	500	1000	5000	10000	50000	100000
100	64	1	0	0	0	0	0
1000	222	176	87	0	0	0	0
10000	255	338	375	195	120	0	0

Man erkennt ein Ansteigen mit wachsendem r und ein Abfallen mit wachsendem n. Ferner steigen die Zahlen an bei engerer Struktur der Aufgaben. Stark korrelierte Aufgaben sind nur sehr begrenzt lösbar.

Anzahl der Iterationen

Als nächstes zahlen wir die Anzahl der Aufrufe von "Verzweige" im Verfahren (in Hundert)

unkorreliert

r\n	100	500	1000	5000	10000	50000	100000
100	1	1	1	0	0	0	0
1000	1	6	21	110	107	15	12
10000	2	14	34	336	1048	4446	7234

schwach korreliert

r\n	100	500	1000	5000	10000	50000	100000
100	2	1	1	0	0	0	0
1000	20	102	136	56	99	71	3
10000	17	188	541	4031	7071	5853	3610

Stark korreliert

r\n	100	500	1000	5000	10000	50000	100000
100	17780	-	-	-	-	-	-
1000	476874	-	-	-	-	-	-
10000	373863	-	-	-	-	-	-

Subset sums

r\n	100	500	1000	5000	10000	50000	100000
100	0	0	0	0	0	0	0
1000	8	4	3	0	0	0	0
10000	115	87	99	49	23	0	0

Man erkennt, warum stark korrelierte Aufgaben eine so lange Laufzeit haben und subset sums nicht.

Größe des Kerns

Nun geben wir die Größe des Kerns als prozentualen Bruchteil von n an.

unkorreliert

r\n	100	500	1000	5000	10000	50000	100000
100	14,1	3,9	2,1	0,2	0,1	0,0	0,0
1000	14,8	5,5	4,1	2,0	1,2	0,3	0,1
10000	16,7	7,0	4,6	2,0	1,6	0,6	0,5

schwach korreliert

r\n	100	500	1000	5000	10000	50000	100000
100	21,5	3,9	2,7	0,2	0,0	0,0	0,0
1000	41,9	17,2	10,7	2,1	0,9	0,3	0,1
10000	37,6	16,1	12,3	5,4	3,8	1,8	0,6

Stark korreliert

r\n	100	500	1000	5000	10000	50000	100000
100	49,6	-	-	-	-	-	-
1000	83,5	-	-	-	-	-	-
10000	97,1	-	-	-	-	-	-

Subset sums

r\n	100	500	1000	5000	10000	50000	100000
100	25,5	0,5	0,2	0,1	0,0	0,0	0,0
1000	86,6	43,2	18,6	0,1	0,0	0,0	0,0
10000	100,0	85,4	83,9	34,1	18,8	0,0	0,0

Diese Information kann man heranziehen, um die Formel für den Schätz-Kern von Martello und Toth zu überprüfen (Übungsaufgabe).

Gesamtlaufzeit

Es wird die Gesamtlaufzeit des Verfahrens in der implementierten Version in Sekunden angegeben.

unkorreliert

r\n	100	500	1000	5000	10000	50000	100000
100	0,00	0,00	0,00	0,01	0,01	0,07	0,16
1000	0,00	0,00	0,00	0,02	0,03	0,09	0,19
10000	0,00	0,00	0,01	0,05	0,16	0,72	1,28

schwach korreliert

r\n	100	500	1000	5000	10000	50000	100000
100	0,00	0,00	0,00	0,01	0,01	0,06	0,16
1000	0,00	0,02	0,02	0,01	0,03	0,09	0,19
10000	0,00	0,03	0,08	0,57	0,99	0,93	0,71

Stark korreliert

r\n	100	500	1000	5000	10000	50000	100000
100	35,67	-	-	-	-	-	-
1000	66,17	-	-	-	-	-	-
10000	30,74	-	-	-	-	-	-

Subset sums

r\n	100	500	1000	5000	10000	50000	100000
100	0,00	0,00	0,00	0,01	0,01	0,05	0,11
1000	0,00	0,00	0,00	0,00	0,01	0,05	0,11
10000	0,02	0,02	0,02	0,03	0,03	0,05	0,11

Vergleich mit MT2

Es wurde MT2 analog implementiert und auf der gleichen Maschine getestet.

unkorreliert

r\n	100	500	1000	5000	10000	50000	100000
100	0,00	0,00	0,00	0,01	0,02	-	-
1000	0,00	0,00	0,00	0,02	0,03	0,14	0,28
10000	0,00	0,00	0,01	0,04	0,07	0,33	1,58

schwach korreliert

r\n	100	500	1000	5000	10000	50000	100000
100	0,00	0,00	0,00	0,01	0,01	0,11	0,25
1000	0,00	0,01	0,01	0,02	0,05	0,30	-
10000	0,00	0,01	0,02	0,09	0,16	0,31	1,33

Stark korreliert

r\n	100	500	1000	5000	10000	50000	100000
100	10,59	-	-	-	-	-	-
1000	5,21	-	-	-	-	-	-
10000	21,15	-	-	-	-	-	-

Subset sums

r\n	100	500	1000	5000	10000	50000	100000
100	0,00	0,00	0,00	0,00	0,01	0,06	0,12
1000	0,00	0,00	0,00	0,00	0,01	0,06	0,13
10000	0,01	0,02	0,02	0,03	0,02	0,08	0,15

Man sieht, dass beide Verfahren in etwa gleich schnell sind, dass aber expknap das stabilere Verhalten zeigt.

4.5.2 Ergebnisse zu minknap

Betrachten wir zuerst, wie erfolgreich Dominieren und Auslotung zur Reduktion der mitgeführten Zustände sind.

Maximale Anzahl der erhaltenen Zustände

Die folgenden Angaben sind Anzahlen in Tausend (gerundet)

unkorreliert

R\n	100	300	1000	3000	10000	30000	100000
100	0	0	0	0	0	2	4
1000	0	0	1	1	3	4	4
10000	0	0	1	1	6	9	19

schwach korreliert

R\n	100	300	1000	3000	10000	30000	100000
100	0	0	1	0	1	0	0
1000	0	1	2	4	5	11	27
10000	0	1	3	7	12	32	65

Stark korreliert

R\n	100	300	1000	3000	10000	30000	100000
100	1	1	3	4	8	15	37
1000	8	14	25	45	74	130	250
10000	78	148	287	425	764	1407	2547

Subset sums

R\n	100	300	1000	3000	10000	30000	100000
100	0	1	0	0	0	0	0
1000	6	5	5	5	6	6	5
10000	63	53	55	60	52	55	48

Laufzeitvergleich zwischen minknap und MT2

Die folgenden Tabellen geben die *Laufzeiten für minknap* in Sekunden an. (Laufzeiten unter 24 Stunden)

unkorreliert

R\n	100	300	1000	3000	10000	30000	100000
100	0,00	0,00	0,00	0,00	0,01	0,03	0,12
1000	0,00	0,00	0,00	0,01	0,01	0,03	0,10
10000	0,00	0,00	0,00	0,01	0,03	0,07	0,17

schwach korreliert

R\n	100	300	1000	3000	10000	30000	100000
100	0,00	0,00	0,00	0,00	0,01	0,03	0,10
1000	0,00	0,00	0,00	0,01	0,01	0,03	0,12
10000	0,00	0,00	0,01	0,03	0,05	0,08	0,16

Stark korreliert

R\n	100	300	1000	3000	10000	30000	100000
100	0,00	0,02	0,09	0,24	1,25	3,19	14,02
1000	0,03	0,18	0,63	2,30	10,41	43,28	178,45
10000	0,44	1,63	7,62	42,89	163,15	496,47	2208,10

Subset sums

R\n	100	300	1000	3000	10000	30000	100000
100	0,00	0,00	0,00	0,00	0,01	0,05	0,11
1000	0,00	0,01	0,01	0,01	0,02	0,04	0,12
10000	0,06	0,06	0,06	0,06	0,07	0,10	0,20

Die folgenden Tabellen geben die *Laufzeiten für MT2* in Sekunden auf der gleichen Maschine an (Laufzeiten unter 24 Stunden)

unkorreliert

R\n	100	300	1000	3000	10000	30000	100000
100	0,00	0,00	0,00	0,00	0,02	2,47	-
1000	0,00	0,00	0,00	0,01	0,03	0,07	0,28
10000	0,00	0,00	0,01	0,02	0,07	0,20	0,56

schwach korreliert

R\n	100	300	1000	3000	10000	30000	100000
100	0,00	0,00	0,00	0,00	0,02	0,05	0,25
1000	0,00	0,00	0,01	0,01	0,04	0,61	-
10000	0,01	0,01	0,02	0,07	0,16	0,26	0,56

Stark korreliert

R\n	100	300	1000	3000	10000	30000	100000
100	2,78	-	-	-	-	-	-
1000	2,68	-	-	-	-	-	-
10000	21,16	-	-	-	-	-	-

Subset sums

R\n	100	300	1000	3000	10000	30000	100000
100	0,00	0,00	0,00	0,00	0,01	0,03	0,12
1000	0,00	0,00	0,00	0,00	0,01	0,03	0,12
10000	0,01	0,02	0,02	0,02	0,03	0,05	0,15