JOHANNES KEPLER
UNIVERSITÄT LINZ

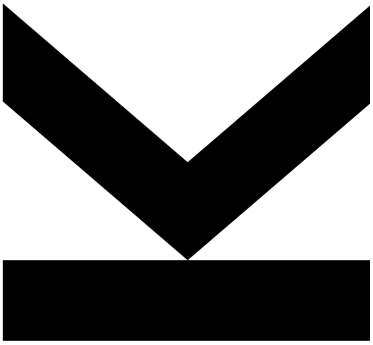
Eingereicht von
Ingolf Neumüller

Angefertigt am
Research Institute for
Symbolic Computation

Beurteiler
Assoc. Univ.-Prof. DI Dr.
Wolfgang Windsteiger

Februar 2020

Eine industrielle Anwendung von heuristischen Optimierungsverfahren für das Bin-Packing Problem



Masterarbeit

zur Erlangung des akademischen Grades

Diplom - Ingenieur

im Masterstudium

Industriemathematik

JOHANNES KEPLER
UNIVERSITÄT LINZ
Altenberger Straße 69
4040 Linz, Österreich
www.jku.at
DVR 0093696

Abstract

In this master thesis we deal with a concrete loading problem. We will formulate this problem with mathematical methods and then break it up into smaller sub-problems. It will arise a core problem that we model as a bin-packing problem. To this problem, we will make general theoretical reflections and, for example, take a closer look at the complexity. Afterwards, this core problem is solved under different aspects, different algorithms were implemented and compared.

Kurzfassung

In dieser Masterarbeit befassen wir uns mit einem konkreten Verladeproblem. Dieses Problem werden wir mit mathematischen Methoden formulieren und anschließend in kleinere Teilprobleme zerlegen. Es wird ein Kernproblem entstehen, dass wir als ein Bin-Packing Problem modellieren. Zu diesem Problem werden wir allgemeine theoretische Überlegungen anstellen und dabei beispielsweise die Komplexität dazu genauer betrachten. Anschließend wird dieses Kernproblem unter verschiedenen Gesichtspunkten gelöst, dabei wurden verschiedene Algorithmen implementiert und verglichen.

Inhaltsangabe

Abbildungsverzeichnis

Tabellenverzeichnis

Algorithmenverzeichnis

1.Einleitung

Immer mehr Menschen bestellen mittlerweile Kleidung, Elektrogeräte und vieles mehr aus dem Internet, sogar Urlaubsreisen werden inzwischen online gebucht, und das klassische „Brick and Mortar“ Business nimmt stetig ab. Österreich hinkt diesem Trend in Sachen Online-Lebensmittelhandel den anderen Ländern noch nach. (Siehe Abbildung1, die Prozentzahl gibt an, wie viele Menschen im Land online ihre Lebensmittel kaufen) Jedoch muss erwähnt werden, dass sich nun auch in Österreich beständig mehr Menschen dazu entschließen, ihre Nahrungsmittel nach Hause liefern zu lassen. Schließlich ist das Bestellen im Onlinestore ohne Stress und mit sehr geringem Aufwand schnell erledigt und spart zudem auch noch immens viel Zeit. Des Weiteren ist auch dem Schleppen der Lebensmittel ein Ende gesetzt. Es ist also nicht verwunderlich, dass der Trend auch hier ganz klar in diese Richtung geht.

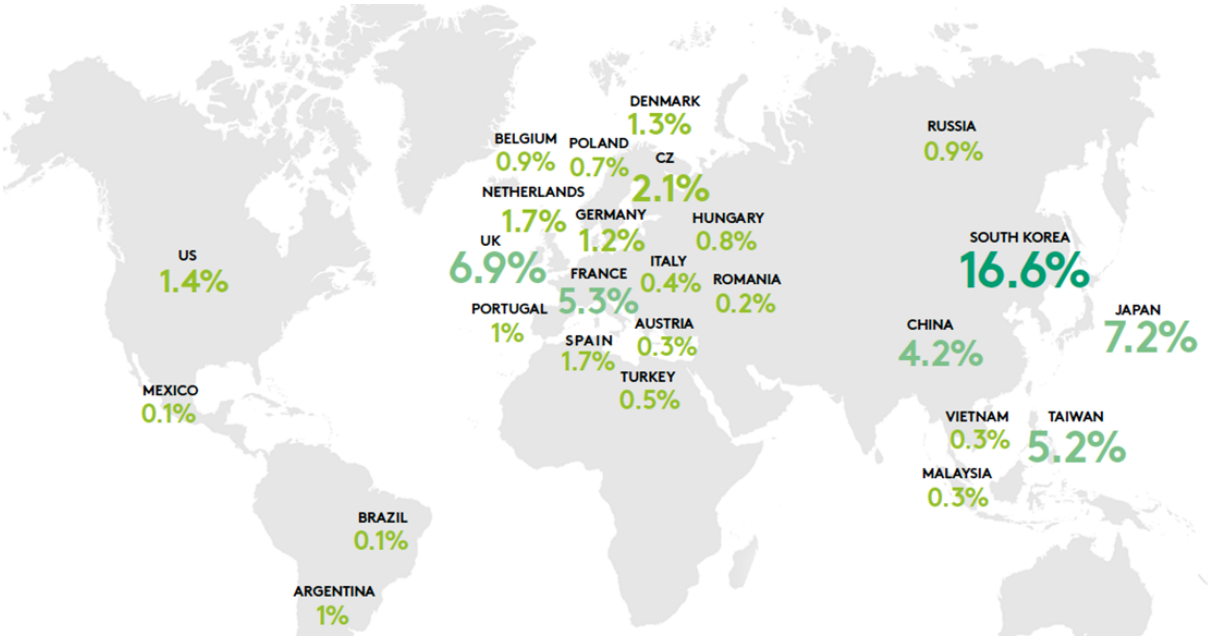


Abbildung 1 : Weltkarte Derzeit noch wenig Anteil im E – commerce aber mit starkem Wachstum (2016) [1]

In Abbildung 2 wird noch die kürzere und effizientere SupplyChain verdeutlicht, da hier die Ware nicht extra in einem Geschäft zwischengelagert werden muss, sondern direkt zum Kunden transportiert wird. Dadurch ist auch eine bessere Kühlkette gewährleistet!

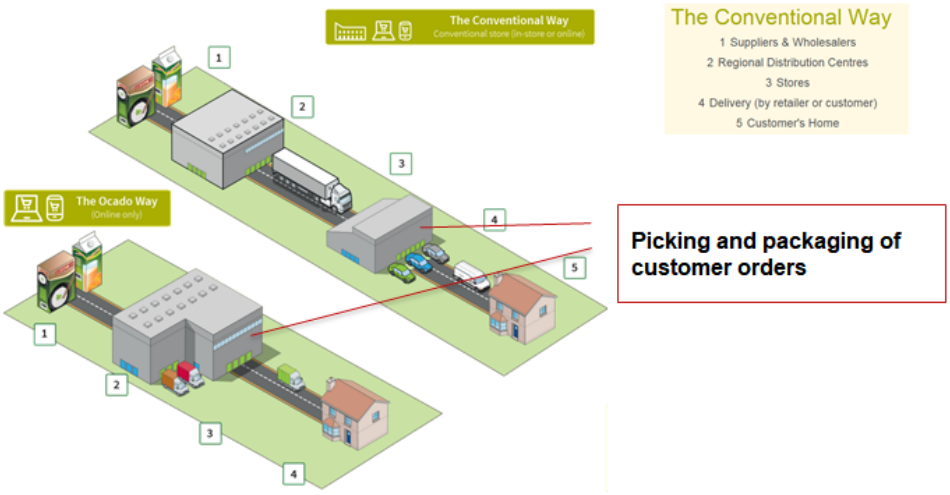


Abbildung 2 : SupplyChain [2]

Die oben diskutierten Aspekte stellen die Händler nun vor das Problem, dass sie nun selbst logistischen Mehraufwand betreiben müssen. Immerhin müssen die Artikel (im Englischen auch Items genannt) der einzelnen Aufträge zum einen intern kommissioniert und in geeignete Behälter gepackt werden. Zum anderen sollen die fertiggestellten Container (engl. Bins) auch noch in einem dafür vorgesehenen Transporter verladen und schnellstmöglich unbeschädigt zum Kunden transportiert werden. Um im Vorhinein Kosten zu sparen, wird zuallererst versucht, die Behältnisse so dicht wie möglich zu packen. An dieser Stelle ist dies am besten zu realisieren, wenn das Unterbringen mehrerer Aufträge in einem Container erlaubt wird. (Verdichtung von Aufträgen) Ein besonderes Augenmerk gilt ebenso dem Verladen der Bins in den LKW, da auch hier unentwegt Optimierungspotential vorhanden ist. Schlussendlich ist es nicht nur ärgerlich Luft zu transportieren, sondern auch Budget belastend, weshalb auch so viel Platz wie möglich im LKW sinnvoll genutzt werden soll, damit möglichst viele Kunden übergangslos beliefert werden können.
Da in der Industrie ohne Ausnahme der Kostendruck präsent ist, ist das Ziel nach Optimierung und Effizienz ein sehr wichtiger Punkt. Deshalb ist auch das Ersetzten von Menschen durch Maschinen in den letzten Jahren deutlich gestiegen und auch hier wird diese Arbeit als Grundstein für ein solches Vorgehen dienen. Denn an letzter Stelle steht das Ziel, das Befüllen der Behälter vollautomatisch abwickeln zu lassen.

Nachstehend ein Bild eines hoch automatischen Befüllungszentrums mit Transporten, die 3 Temperaturzonen besitzen.

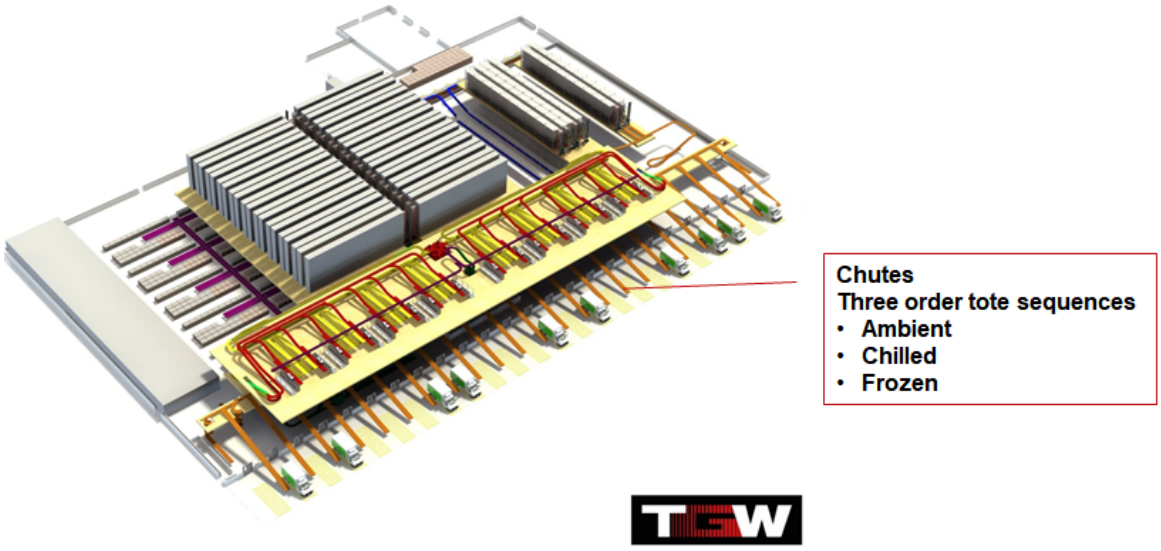


Abbildung 3 : ~ 5000 Aufträge werden hier in die Transporter gleichzeitig verladen [3]

Das Innenleben einer solchen Anlage sieht in etwa folgendermaßen aus:

Abbildung 4 : Automatisierte Anlage (von Ocado). Quellbehälter in grün, Zielbehälter (rot) mit drei Tüten.[2]

Die Behälter werden an einer “Picking Worksation“ befüllt. Zur Zeit muss hier noch ein Mensch arbeiten. In näherer Zukunft werden unsere Algorithmen und Erkenntnisse dazu verwendet, um diese Stelle einem Roboter zu überlassen. In Abbildung 5 ist beispielhaft eine solche Arbeitsstätte zu sehen.

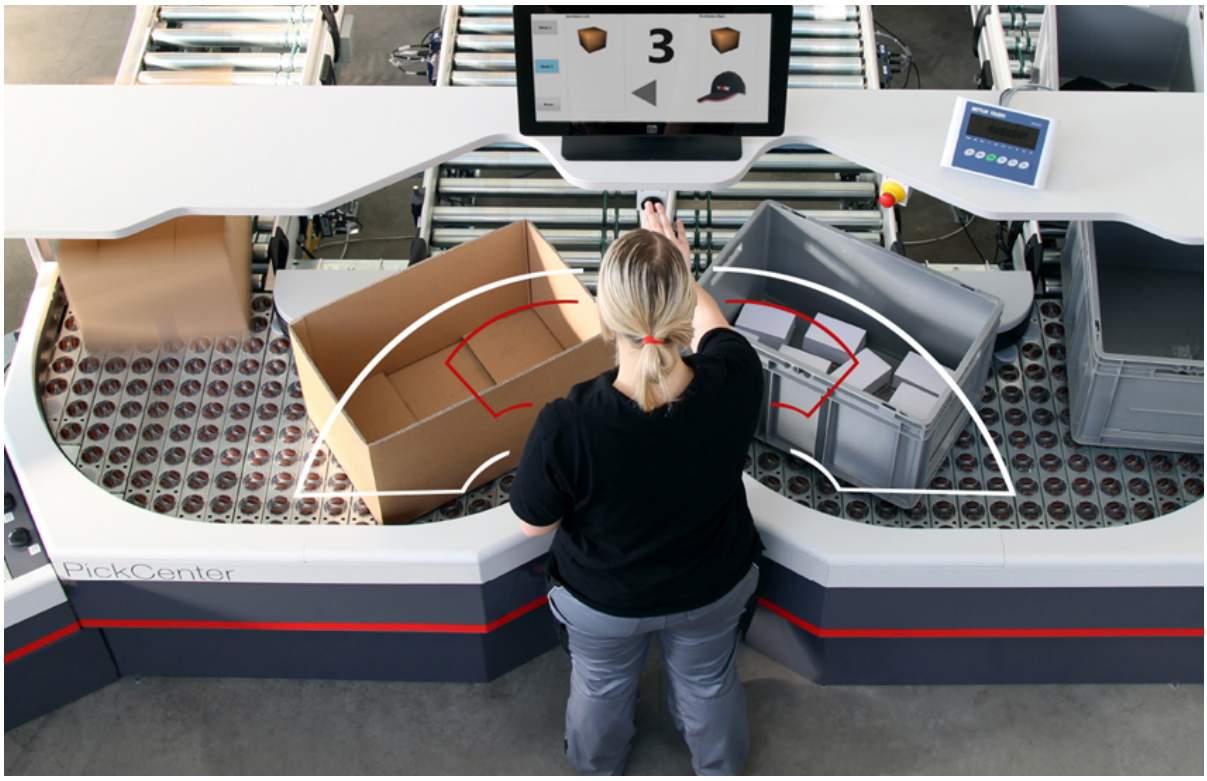


Abbildung 5 : Picking Workstation [3]

In dieser Arbeit werden wir nicht alle in der Einleitung angesprochenen Probleme lösen, wir werden uns nur auf das mathematische Teilproblem der Artikelzuordnung konzentrieren. Wir werden also eine Funktion suchen, die jedem Artikel eines Auftrags einen Behälter zuweist.

1.1 Zielsetzung

Die primäre Aufgabe dieser Arbeit ist es, ein Programmpaket zu entwickeln, das Algorithmen enthält, die das Zuordnungsproblem lösen und auch praktisch eingesetzt werden können. Wir werden dieses Paket im Rahmen dieser Arbeit als tgw.opt bezeichnen. Es werden mathematische Modelle entwickelt, die die Basis für unsere verschiedenen Ansätze bilden werden. Bei diesen Ansätzen soll insbesondere die Praxistauglichkeit untersucht werden, weswegen wir hierfür das Indiz der Laufzeit heranziehen.
Da das Auffinden einer optimalen Lösung unter Umständen sehr aufwendig sein kann, werden auch heuristische Ansätze evaluiert werden. Schließlich haben diese den Vorteil einer geringeren Laufzeit. Dem steht allerdings der Nachteil gegenüber, dass die Lösung von der optimalen abweichen kann. Aus diesem Grund wird hier die Differenz zwischen der Optimallösung und der Näherungslösung betrachtet, um die Qualität der Lösung beurteilen zu können.
Die Optimallösung zu finden, geht natürlich nur bis zu einer gewissen Inputgröße. Deshalb werden wir viele kleine Aufträge, die noch optimal lösbar sind, berechnen und versuchen, mit den Heuristiken nahe an diese Lösungen heranzukommen. Dies gibt uns die Hoffnung, dass diese auch für größere Aufträge gute Resultate erzielen.

1.2 Aufbau der Arbeit

Diese Masterarbeit wird in 7 Hauptkapitel unterteilt. Einleitend wird dem Leser die Problemstellung sowie die Zielsetzung dieser Arbeit näher gebracht (Kapitel 1). Kapitel 2 befasst sich mit der Problembeschreibung und mathematischen Modellierung unseres Problems. Dabei werden wir das Problem vereinfachen und auf ein bekanntes Problem zurückführen. In Kapitel 3 werden wir uns mit der Komplexitätstheorie und insbesondere mit den Komplexitätsklassen befassen und unser Problem dann geeignet klassifizieren. Weiters werden wir uns mit der linearen Optimierung beschäftigen und unser Problem dann alternativ als ein lineares Optimierungsproblem modellieren. Dabei werden wir auch geeignete Lösungsmethoden erörtern und das Problem exakt sowie näherungsweise Lösen. In Kapitel 4 finden sich kurz einige Erweiterungen, wie wir die Praxistauglichkeit unserer Software, die wir aus dem Wissen der vorherigen Kapiteln entwickelt haben, noch etwas erweitern. In Kapitel 5 finden wir dazu den Programmaufbau der Software. Die numerischen Resultate unserer Ansätze, die wir auf verschiedene Aufträge angewendet haben, zeigen wir dann in Kapitel 6. Abschließend wird eine Handlungsempfehlung für den Auftraggeber abgegeben. Weiters geben wir einen Ausblick auf zukünftige Verbesserungen der Methodiken (Kapitel 7).

2. Problembeschreibung und mathematische Modellierung

2.1 Verbale Beschreibung

Ein Lebensmittelhändler bekommt von mehreren Kunden zu verschiedenen Zeiten Online Bestellungen. Diese Aufträge sollen nun optimal in einzelne Behältnisse verpackt werden. Optimal bedeutet hier, dass man so wenig Bins wie möglich verwenden soll und in den einzelnen Containern einen hohen Füllgrad anstrebt. Für einen Auftrag dürfen mehrere Behältnisse verwendet werden. Diese Behältnisse haben nur innerhalb eines Lebensmittelhändlers einheitliche Größen.
Damit ein Artikel in einen Behälter verpackt werden darf, muss er die folgenden Restriktionen erfüllen:

Artikel:

Die Händler gliedern ihr Sortiment in die folgenden Teilbereiche:

DryGoods, ChilledGoods, FrozenGoods, Chemicals.

Inmitten einer Warengruppen darf sich auf Grund der Temperaturzonen und aus gesundheitsgefährdenden Gründen keine andere in demselben Container befinden. Des Weiteren ist nicht ausgeschlossen, dass sich auch fragile Artikel in einem Auftrag befinden können. Deshalb sollen zuerst die stabilen Artikel verpackt werden und anschließend die fragilen. Sollte kein Stabilitätsparameter vorhanden sein, so soll schwer vor leicht gelten. Ferner sind die Artikel noch stark heterogen.
Ein Artikel hat die Eigenschaften: Name, Artikelnummer, Gewicht, Volumen, Stabilität und Teilsortiment.

Aufträge:

Die Aufträge kommen als ein oder in mehreren File(s) in das System. Ein Auftrag besteht aus verschiedenen Artikeln. Jeder Artikel in dem Auftrag entspricht einer Zeile in dem File. Kommt ein Artikel in einem Auftrag beispielsweise x – mal vor, so sind in diesem File auch x Zeilen für diesen Artikel enthalten.

Behälter:

Die Behälter haben ein maximal zulässiges Gesamtgewicht, welches in unserem Fall auf 20kg pro Behälter vorgegeben ist und nicht überschritten werden darf. Die Abmessung eines jeden Behälters ist 600x400x400 (in mm). Außerdem ist es unzulässig, das maximale Volumen zu überschreiten. Weiters dürfen nur Artikel mit der gleichen Warengruppe in einen Behälter gepackt werden.

Transporter:

Die Transporter haben unterschiedliche Größen und Strukturen manche davon können bis zu 3 Temperaturzonen für die einzelnen Warengruppen auf der Ladefläche enthalten. Wir haben pro LKW nur eine Temperaturzone betrachtet.

2.2 Präzisierte Beschreibung

Sortiment:

Gegeben ist das Sortiment $S = \{a_1, ..., a_n\}$ bestehend aus $n \in \mathbb{N}$ unterschiedlichen Artikeln a_i. S unterteilt sich in die Teilsortimente $S_1, ... S_4$ so, dass

$$S = \bigcup_{i=1}^{4} S_i$$

$$S_k \bigcap S_j = \emptyset, \text{ für alle } k \neq j \text{ und } k, j \in \{1, ..., 4\}.$$

Damit ist $\{S_1, ..., S_4\}$ eine Partition von S. In dieser Arbeit assoziieren wir die Teilsortimente auf folgende Weise: $S_1 \hat{=}$ DryGoods, $S_2 \hat{=}$ ChilledGoods, $S_3 \hat{=}$ FrozenGoods, $S_4 \hat{=}$ Chemicals.

Artikel:

Ein Artikel $a_i \in S$ kann als das Tupel

$$(Name, Artikelnummer, Teilsortiment, Gewicht, Volumen, Stabilität)$$

aufgefasst werden. Dabei gilt

$Teilsortiment \in \{1, ..., 4\}$, $Gewicht \in \mathbb{R}^+$, $Volumen \in \mathbb{R}^+$, $Stabilität \in \{0, 1, 2, 3\}$,

Bemerkung:

- Für die Optimierung sind ausschließlich die letzten 4 Komponenten des Tupels relevant.
- Die Stabilität eines Artikels beeinflusst die Beladereihenfolge. Artikel mit höherer Stabilität sollen zuerst verpackt werden. Ist die $Stabilität = 0$ für einen Artikel, so hat dieser keinen solchen Parameter.

Aufträge:

Gegeben sind die Aufträge $I_1, ..., I_l$ mit $I_j \in S^{m_j}$, wobei m_j die Anzahl der Artikel im Auftrag j bezeichnet. Ein Auftrag kann also als Tupel von Artikeln aufgefasst werden. Da ein Artikel in einem Auftrag mehrfach auftreten kann, modellieren wir einen Auftrag als Tupel anstelle von Mengen. Deshalb muss auch nicht zwingend $m_j \leq n$ gelten.
Weiters sei die Gesamtzahl der Artikel in allen Aufträgen definiert als:

$$m := \sum_{j=1}^{l} m_j$$

Im Weiteren betrachten wir die Aufträge $I_1, ..., I_l$ als einen Gesamtauftrag $J \in S^m$ und mit der Schreibweise

$$t_1 \shortparallel t_2$$

bezeichnen wir das Zusammenhängen (Konkatenieren) der Tupel t_1 und t_2. Damit ergibt sich für den Gesamtauftrag

$$J = I_1 \shortparallel I_2 \shortparallel, ..., \shortparallel I_l$$

Behälter:

Weiters seien k Behälter $B = \{y_1, \ldots, y_k\}$ mit $y_i \in \mathbb{R}^+ \times \mathbb{R}^+$ gegeben, wobei $k \leq m$ ist. Jeder Behälter y_i ist ein Tupel mit den folgenden Komponenten:

$$(Volumen,\ zulässiges\ Beladegewicht) \in \mathbb{R}^+ \times \mathbb{R}^+$$

Schreibweisen:

Im Folgenden führen wir nun die folgenden kürzeren Schreibweisen ein:
Beispielsweise greifen wir auf das Teilsortiment a_3 des Artikel $a \in S$ mit $sort(a)$ zu. Nachstehend die komplette Liste der Schreibweisen für einen Artikel:

$$sort(a) := a_3$$
$$g(a) := a_4$$
$$v(a) := a_5$$
$$stab(a) := a_6$$

Analog dazu greifen wir für das Volumen und das zulässige Beladegewicht eines Behälter y $\in B$ wie folgt zu :

$$\overline{v}(y) := y_1$$
$$\overline{g}(y) := y_2$$

Problem Artikelzuordnung $A(m, \mathcal{J}, k, \mathcal{B})$

Gegeben: $m \in \mathbb{N}$, $\mathcal{J} \in S^m$, $k \leq m$, $\mathcal{B} = \{y_1, \ldots, y_m\}$

Gesucht:

1. eine Zahl $x \in \mathbb{N}$ mit $x \leq k$,

2. eine Funktion $f : \{1, \ldots, m\} \to \{1, \ldots, x\}$,

3. eine Permutation $\Pi : \{1, \ldots, m\} \to \{1, \ldots, m\}$ (bijektiv)

so, dass die folgenden Restriktionen erfüllt sind.

1) Jeder Behälter enthält maximal einen Auftrag:

$$\forall\, i = 1, \ldots, m\, \forall\, j = 1, \ldots, m : f(i) = f(j) \Rightarrow \exists\, t \in \mathcal{A}\, \exists\, \alpha, \beta : \mathcal{J}_i = I_{t,\alpha} \wedge \mathcal{J}_j = I_{t,\beta}$$

wobei $\mathcal{A} = \{\sigma, \ldots, \tau\}$ falls $\mathcal{J} = I_\sigma \shortparallel \ldots \shortparallel I_\tau$

2) Jeder Behälter enthält höchstens ein Sortiment:

$$\forall\, i = 1, \ldots, m\, \forall\, j = 1, \ldots, m : f(i) = f(j) \Rightarrow sort(\mathcal{J}_i) = sort(\mathcal{J}_j)$$

3) Stabilitätsrestriktion:

i) $stab(\mathcal{J}_i) = 0$: (Schwere vor leichte Artikel)

$$\forall\, i = 1,\ ..., m\, \forall\, j = 1,\ ..., m:$$
$$(f(i) = f(j) \wedge \Pi(i) < \Pi(j) \wedge stab(\mathcal{J}_i) = stab(\mathcal{J}_j)) \Rightarrow g(\mathcal{J}_i) \geq g(\mathcal{J}_j)$$

ii) $stab(\mathcal{J}_i) \neq 0$: (Stabile Artikel zuerst)

$$\forall\, i = 1,\ ..., m\, \forall\, j = 1,\ ... m:$$
$$(f(i) = f(j) \wedge \Pi(i) < \Pi(j) \wedge stab(\mathcal{J}_i) \neq 0 \wedge stab(\mathcal{J}_j) \neq 0) \Rightarrow stab(a_i) \geq stab(a_j)$$

4) Überfüllung der Behälter ist nicht gestattet:

$$Packing(f, x, \mathcal{J}, y)$$

wobei $Packing(f, x, \mathcal{J}, y) : \Leftrightarrow \forall\, s \in \{1,\ ..., x\} : \sum_{f(i) = s} g(\mathcal{J}_i) \leq \overline{g}(y_s) \ \wedge \ \sum_{f(i) = s} v(\mathcal{J}_i) \leq \overline{v}(y_s)$.

Zu minimieren gilt es nun die Anzahl $x \leq k$ der Behälter, für die eine Funktion f und eine Permutation Π wie oben beschrieben existiert.

Schreibweise: (x, f, Π) ist Lösung von $A(m, J, k, B)$ falls x, f und Π die obigen Eigenschaften haben.

Bemerkung:

α) Die beiden Fälle 3.i und 3.ii behandeln nicht alle möglichen Kombinationen der Stabilitätsrestriktion. In der Praxis sind dies aber die einzig möglichen Fälle, da in einem Auftrag entweder alle Artikel einen Stabilitätsparameter ungleich null haben oder keiner.

β) Beispielsweise könnte eine Zuordnung der Artikel auf die Behälter folgendermaßen aussehen:

f:	i	f(i)
	1	2
	2	2
	3	3
	4	1
	5	2
	6	3
	7	1

Abbildung 6 : Zuordnungsfunktion

Wichtig hierbei ist, dass die Funktion f nur auf den Indizes arbeitet. Die Funktion kennt nicht den konkreten Artikel, den sie verpackt, sondern lediglich den Index, der dem Artikel zugeteilt ist.

2.3 Vereinfachung des Problems

Hier werden wir versuchen, das Problem in leichter zu lösende Teilprobleme zu zerlegen. Im Folgenden werden wir das Hintereinandersetzten von Funktionen des Öfteren benötigen, weshalb wir dafür nun eine Schreibweise einführen.

Im Folgenden haben die Funktionen folgende Gestalt:

$$\begin{aligned} f_1 &: \{1, \ ..., k_1\} \to \{1, \ ..., l_1\} \\ f_2 &: \{1, \ ..., k_2\} \to \{1, \ ..., l_2\} \\ &\vdots \\ f_n &: \{1, \ ..., k_n\} \to \{1, \ ..., l_n\} \end{aligned} \tag{1.0}$$

Definition 1:

Für das *Hintereinandersetzten* von n Funktionen obiger Gestalt definieren wir:

$$f_1 \sqcup ... \sqcup f_n : \left\{1, \ ..., \sum_{i=1}^{n} k_i\right\} \to \left\{1, \ ..., \sum_{i=1}^{n} l_i\right\}$$

$$x \mapsto \begin{cases} f_1(x) & x \le k_1 \\ \vdots & \vdots \\ f_s\left(x - \sum_{i=1}^{s-1} k_i\right) + \sum_{i=1}^{s-1} l_i & x > \sum_{i=1}^{s-1} k_i \ \wedge \ x \le \sum_{i=1}^{s} k_i \\ \vdots & \vdots \\ f_n\left(x - \sum_{i=1}^{n} k_i\right) + \sum_{i=1}^{n} l_i & x > \sum_{i=1}^{n-1} k_i \end{cases}$$

Auf den ersten Blick sieht diese Definition kompliziert aus. Sie hängt aber lediglich die einzelnen Funktionen nahtlos nacheinander an. Folgendes Bild veranschaulicht dies. (Abbildung 7)

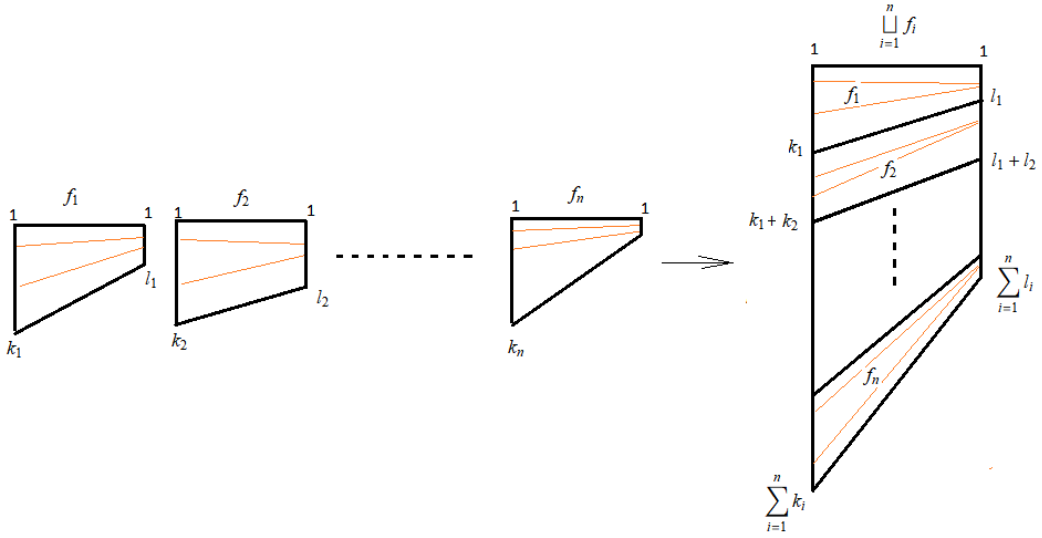


Abbildung 7 : Veranschaulichung des Hintereinandersetzen

Die wesentliche Eigenschaft dieses “Hintereinandersetzen” ist, dass die Definitionsbereiche sowie die Wertebereiche disjunkt aneinandergehängt werden, sodass alle einzelnen Funktionen nur in den eigenen Wertebereich abbilden. Mathematisch ist dies im folgenden Satz ausgedrückt :

Satz 1:

Für Funktionen der Form (1.0) gilt:

$$\forall\, x, y : (f_1 \bigsqcup \ldots \bigsqcup f_n)(x) = (f_1 \bigsqcup \ldots \bigsqcup f_n)(y) \Rightarrow \exists\, s \in \{1, \ldots n\}\, \exists\, \alpha, \beta :$$

$$\left((f_1 \bigsqcup \ldots \bigsqcup f_n)(x) = f_s(\alpha) + \sum_{i=1}^{s-1} l_i \;\wedge\; (f_1 \bigsqcup \ldots \bigsqcup f_n)(y) = f_s(\beta) + \sum_{i=1}^{s-1} l_i\right) \;\wedge$$

$$x = \sum_{i=1}^{s-1} k_i + \alpha\,,\; y = \sum_{i=1}^{s-1} k_i + \beta.$$

Der exakte Beweis dieses Satzes ist sehr lang und aufwendig. Außerdem müssten wir dafür die Definition 1 rekursiv formulieren, und könnten dann den Satz 1 per Induktion bewiesen. An dieser Stelle ersparen wir uns diesen Beweis für den allgemeinen Fall. Wir betrachten im Detail nur den Fall $n = 2$, in dem die wesentliche Beweisidee steckt, d.h.

$$f_1 \bigsqcup f_2 : \{1, \ldots, k_1 + k_2\} \to \{1, \ldots, l_1 + l_2\}$$

$$x \mapsto \begin{cases} f_1(x) & x \le k_1 \\ f_2(x - k_1) + l_1 & x > k_1 \end{cases}$$

Hilfssatz :

Sei $n = 2$. Für Funktionen der Form (1.0) gilt:

$$\forall\, x, y : (f_1 \bigsqcup f_2)(x) = (f_1 \bigsqcup f_2)(y) \Rightarrow \exists\, \alpha, \beta :$$
$$(f_1 \bigsqcup f_2)(x) = f_1(\alpha) \;\wedge\; (f_1 \bigsqcup f_2)(y) = f_1(\beta) \;\wedge\; x = \alpha\,, y = \beta \;\vee$$
$$(f_1 \bigsqcup f_2)(x) = f_2(\alpha) + l_1 \;\wedge\; (f_1 \bigsqcup f_2)(y) = f_2(\beta) + l_1 \;\wedge\; x = k_1 + \alpha\,,\; y = k_1 + \beta.$$

Beweis:

Wir nehmen an, dass gilt

$$(f_1 \bigsqcup f_2)(x) = (f_1 \bigsqcup f_2)(y) \qquad (1)$$

Es sind nun α und β so zu finden, dass gilt:

$$(f_1 \bigsqcup f_2)(x) = f_1(\alpha) \;\wedge\; (f_1 \bigsqcup f_2)(y) = f_1(\beta) \;\wedge\; x = \alpha\,, y = \beta \qquad (2)$$

oder

$$(f_1 \bigsqcup f_2)(x) = f_2(\alpha) + l_1 \;\wedge\; (f_1 \bigsqcup f_2)(y) = f_2(\beta) + l_1 \;\wedge\; x = k_1 + \alpha\,,\; y = k_1 + \beta \qquad (3)$$

Wie wir oben sehen können, ist die Funktion über eine Fallunterscheidung definiert. Damit ergeben sich nun 4 Fälle:

Fall 1: $x \le k_1, y \le k_1$

Damit ergibt sich für :

$$(f_1 \sqcup f_2)(x) = f_1(x)$$
$$(f_1 \sqcup f_2)(y) = f_1(y)$$

Eingesetzt in (2) ergibt sich nun:

$$f_1(x) = f_1(\alpha)$$
$$f_1(y) = f_1(\beta)$$

Wenn wir nun $\alpha = x$, $\beta = y$ wählen, ist die Behauptung gezeigt. Da wir nur (2) oder (3) zeigen müssen, brauchen wir nun den Fall (3) nicht mehr betrachten.

Fall 2: $x > k_1, y > k_1$

Hier ergibt sich für:

$$(f_1 \sqcup f_2)(x) = f_2(x - k_1) + l_1$$
$$(f_1 \sqcup f_2)(y) = f_2(y - k_1) + l_1$$

Eingesetzt in (3) ergibt sich nun:

$$f_2(x - k_1) + l_1 = f_2(\alpha) + l_1$$
$$f_2(y - k_1) + l_1 = f_2(\beta) + l_1$$

Wenn wir nun $\alpha = x - k_1$, $\beta = y - k_1$ wählen, ist die Behauptung gezeigt. Da wir nur (2) oder (3) zeigen müssen, brauchen wir nun den Fall (2) nicht mehr betrachten.

Fall 3: $x \leq k_1, y > k_1$

Hier ergibt sich für:

$$(f_1 \sqcup f_2)(x) = f_1(x)$$
$$(f_1 \sqcup f_2)(y) = f_2(y - k_1) + l_1$$

Aus (1) folgt nun:

$$\underbrace{f_1(x)}_{\leq l_1} = \underbrace{f_2(y - k_1) + l_1}_{> l_1}$$ ↯

Fall 4: $x > k_1, y \leq k_1$

Analog zu Fall 3.

□

2.3.1 Behandlung der Restriktion 1)

Wir zerlegen $J = I_1 \amalg I_2 \amalg , \ldots, \amalg I_l$ in die einzelnen Aufträge I_i.
Wir betrachten nun einen einzelnen Auftrag I_i und für diesen Auftrag stehen k_i Behälter zur Verfügung, wobei $k_i \leq m_i$ ist.
Nun berechnen wir für $1 \leq i \leq l$ eine Lösung (x_i, f_i, Π_i) von $A(m_i, I_i, k_i, B_i)$, die die Restriktionen 2), 3) und 4) erfüllen und konstruieren daraus eine Lösung (x, f, Π) von $A(m, I, k, B)$, die zusätzlich Restriktion 1) erfüllt.

In jedem Teilproblem $A(m_i, I_i, k_i, B_i)$ besteht I_i aus nur einem Auftrag, sodass jede Lösung (x_i, f_i, Π_i) die Restriktion 1) automatisch erfüllt.

Eine Lösung (x, f, Π) von $A(m, J, k, B)$ ergibt sich dann wie folgt:

$$x = \sum_{i=1}^{l} x_i \ , \ f = f_1 \bigsqcup \ldots \bigsqcup f_l \ , \ \Pi = \Pi_1 \bigsqcup \ldots \bigsqcup \Pi_l$$

Wir zeigen, dass die Restriktion 1) für (x, f, Π) erfüllt ist, d.h. also:

$$\forall\, i = 1, \ldots, m \; \forall\, j = 1, \ldots, m : f(i) = f(j) \Rightarrow \exists\, t \in \mathcal{A} \; \exists\, \alpha, \beta : J_i = I_{t,\alpha} \wedge J_j = I_{t,\beta}$$

Seien i, j beliebig aber fix.Wir nehmen an, dass gilt:

$$f(i) = f(j) \qquad (*)$$

Wegen Satz 1 und $(*)$ muss gelten, dass

$$f(i) = f_s(\overline{\alpha}) + \sum_{\xi=1}^{s-1} l_\xi \quad \wedge \quad f(j) = f_s(\overline{\beta}) + \sum_{\xi=1}^{s-1} l_\xi \qquad \text{für ein } 1 \leq s \leq l \text{ und gewisse } \overline{\alpha}, \overline{\beta}$$

$$\text{mit } i = \sum_{\xi=1}^{s-1} k_\xi + \alpha \ , \ j = \sum_{\xi=1}^{s-1} k_\xi + \beta \ , \qquad \text{wobei } \alpha \leq k_s, \beta \leq k_s$$

Damit wissen wir:

$$1 \leq i - \sum_{\xi=1}^{s-1} k_\xi \leq k_s \Leftrightarrow 1 + \sum_{\xi=1}^{s-1} k_\xi \leq i \leq \sum_{\xi=1}^{s} k_\xi \qquad (\%)$$

$$1 \leq j - \sum_{\xi=1}^{s-1} k_\xi \leq k_s \Leftrightarrow 1 + \sum_{\xi=1}^{s-1} k_\xi \leq i \leq \sum_{\xi=1}^{s} k_\xi$$

Sei nun $t := s, \ \alpha := \overline{\alpha}$ und $\beta := \overline{\beta}$.
Zu zeigen ist:

$$J_i = I_{s,\overline{\alpha}} \wedge J_j = I_{s,\overline{\beta}}$$

Aufgrund der Gestalt von J wissen wir, dass $J_i = I_i$ gilt. Außerdem folgt aus $(\%)$ und mit der Wahl $t = s, \alpha = \overline{\alpha}, \beta = \overline{\beta}$, dass $I_i = I_{s,\overline{\alpha}} \wedge I_j = I_{s,\overline{\beta}}$.

■

Die Restriktionen 4) bleibt erhalten, da wir die Behälter nur auf die Sortimente aufteilen. Restriktion 3) bleibt erhalten, da wir stets eine Permutation wählen können, sodass die Restriktion 3) erfüllt bleibt. Restriktion 2) bleibt erhalten, da wir die einzelnen Sortimente in einem ganzen Auftrag als einen

separaten Auftrag betrachten. Im Nachhinein werden die Aufträge dann passend zusammengefügt.

Beispiel:
Es sei der Gesamtauftrag $J = I_1 \amalg I_2$ und die dazugehörigen Aufträge $I_1 = (a_1, \ldots, a_8)$ und $I_2 = (a_1, \ldots, a_6)$ gegeben. f_1 betrachtet die Indizes der Artikel aus I_1 und f_2 die Indizes der Artikel aus I_2.
Eine geeignete Lösung ist:

$f_1 \circ \Pi_1$:

i	$f_1(\Pi_1(i))$
1	1
2	2
3	1
4	1
5	2
6	3
7	3
8	4

$f_2 \circ \Pi_2$:

i	$f_2(\Pi_2(i))$
1	5
2	6
3	6
4	5
5	6
6	5

mit geeignetem Π_1 und Π_2.
Weiters haben wir für I_1: $x_1 = 4$ und für I_2 : $x_2 = 2$ Behälter.

Dann ist $f : f_1 \bigsqcup f_2$, $\Pi : \Pi_1 \bigsqcup \Pi_2$, $x = x_1 + x_2$ eine Lösung des Ausgangsproblems.

2.3.2 Behandlung der Restriktion 2)

In Abschnitt 2.3.1 haben wir das Originalproblem auf Teilprobleme $A(m_i, I_i, k_i, B_i)$ zurückgeführt. Wir behandeln nun die Lösung dieser Teilprobleme.
Zerlege den Auftrag $I_i \in S^{m_i}$ auf vier Teilaufträge $\mathcal{I}_1, \mathcal{I}_2, \mathcal{I}_3, \mathcal{I}_4$ und wähle eine Permutation $\sigma : \{1, .., m_i\} \to \{1, \ldots, m_i\}$, sodass gilt:

$$\mathcal{I}_1 \in S_1^{p_1}, \mathcal{I}_2 \in S_2^{p_2}, \mathcal{I}_3 \in S_3^{p_3}, \mathcal{I}_4 \in S_4^{p_4} \text{ mit } \sum_{j=1}^{4} p_j = m_i$$

$$\mathcal{I}_1 \amalg \mathcal{I}_2 \amalg \mathcal{I}_3 \amalg \mathcal{I}_4 = (I_{i,\sigma^{-1}(j)} \mid j = 1, \ldots, m_i) \qquad (\%)$$

Dies ist auf Grund der Sortimenteigenschaften möglich. Für jeden Teilauftrag $\mathcal{I}_j$ sind $k_{i,j}$ Behälter gegeben, wobei $k_{i,j} \leq p_j$ und $\sum_{j=1}^{4} k_{i,j} = k_i$ ist.

Für jeden dieser 4 Aufträge $\mathcal{I}_j$ berechnen wir nun eine Lösung $(x_{i,j}, f_{i,j}, \Pi_{i,j})$ des Problems $A(p_j, \mathcal{I}_j, k_{i,j}, B_{i,j})$, das die Restriktionen 3) und 4) erfüllt und konstruieren daraus eine Lösung (x_i, f_i, Π_i) von $A(m_i, I_i, k_i, B_i)$, die zusätzlich die Restriktion 2) erfüllt.

In jedem Teilproblem $A(p_j, \mathcal{I}_j, k_{i,j}, B_{i,j})$ besteht $\mathcal{I}_i$ aus nur einem Sortiment, sodass jede Lösung $(x_{i,j}, f_{i,j}, \Pi_{i,j})$ die Restriktion 2) automatisch erfüllt.

Eine Lösung (x_i, f_i, Π_i) von $A(m_i, I_i, k_i, B_i)$ ergibt sich dann wie folgt:

$$x_i = \sum_{j=1}^{4} x_{i,j} \quad , \quad f_i = (f_{i,1} \bigsqcup ... \bigsqcup f_{i,4}) \circ \sigma \quad , \quad \Pi_i = (\Pi_{i,1} \bigsqcup ... \bigsqcup \Pi_{i,4}) \circ \sigma$$

Wir zeigen nun, dass die Restriktion 2) für (x_i, f_i, Π_i) erfüllt ist, d.h. also :

$$\forall\, k = 1, \ ..., m_i \; \forall\, j = 1, \ ..., m_i : f_i(k) = f_i(j) \Rightarrow sort(I_{i,k}) = sort(I_{i,j})$$

Seien k, j beliebig aber fix. Nimm an :

$$f_i(k) = f_i(j) \Leftrightarrow (f_1 \bigsqcup ... \bigsqcup f_m)(\sigma(k)) = f_1 \bigsqcup ... \bigsqcup f_m)(\sigma(j)) \qquad (*)$$

Wegen Satz 1 und $(*)$ muss gelten, dass

$$f_i(\sigma(k)) = f_{i,s}(\overline{\alpha}) + \sum_{\xi=1}^{s-1} l_\xi \quad \wedge \quad f_i(\sigma(j)) = f_{i,s}(\overline{\beta}) + \sum_{\xi=1}^{s-1} l_\xi \qquad \text{für ein } 1 \le s \le 4 \text{ und gewisse } \overline{\alpha}, \overline{\beta}$$

$$\text{mit } \sigma(k) = \sum_{\xi=1}^{s-1} p_\xi + \overline{\alpha}, \ \sigma(j) = \sum_{\xi=1}^{s-1} p_\xi + \overline{\beta}, \qquad \text{wobei } \overline{\alpha} \le p_s, \overline{\beta} \le p_s$$

Damit wissen wir:

$$1 \le \sigma(k) - \sum_{\xi=1}^{s-1} p_\xi \le p_s \Leftrightarrow 1 + \sum_{\xi=1}^{s-1} p_\xi \le \sigma(k) \le \sum_{\xi=1}^{s} p_\xi \qquad (\%\%)$$

$$1 \le \sigma(j) - \sum_{\xi=1}^{s-1} p_\xi \le p_s \Leftrightarrow 1 + \sum_{\xi=1}^{s-1} p_\xi \le \sigma(j) \le \sum_{\xi=1}^{s} p_\xi$$

Nach einsetzten in $(*)$ erhalten wir somit :

$$f_{i,s}(\overline{\alpha}) = f_{j,s}(\overline{\beta}) \qquad (**)$$

Wir wissen, dass $(x_{i,s}, f_{i,s}, \Pi_{i,s})$ Restriktion 2) erfüllt, d.h. :

$$f_{i,s}(\overline{\alpha}) = f_{i,s}(\overline{\beta}) \Rightarrow sort(\mathcal{I}_{s,\overline{\alpha}}) = sort(\mathcal{I}_{s,\overline{\beta}})$$

Gemeinsam mit $(**)$ gilt daher :

$$sort(\mathcal{I}_{s,\overline{\alpha}}) = sort(\mathcal{I}_{s,\overline{\beta}})$$

Zu zeigen ist :

$$sort(I_{i,k}) = sort(I_{i,j})$$

Aus (%) und (%%) wissen wir :

$$I_{i,k} = \underbrace{(\mathcal{I}_1 \Vdash \mathcal{I}_2 \Vdash \mathcal{I}_3 \Vdash \mathcal{I}_4)_{\sigma(k)}}_{\mathcal{I}_{s,\overline{\alpha}}} \quad \wedge \quad I_{i,j} = \underbrace{(\mathcal{I}_1 \Vdash \mathcal{I}_2 \Vdash \mathcal{I}_3 \Vdash \mathcal{I}_4)_{\sigma(j)}}_{\mathcal{I}_{s,\overline{\beta}}}$$

■

Die Restriktionen 4) bleibt erhalten da wir die Behälter nur auf die Sortimente aufteilen. Restriktion 3) bleibt erhalten da wir stets eine Permutation wählen können, sodass die Restriktion 3) erfüllt bleibt.

Beispiel: (Fortsetzung)
Wir behandeln nun die Lösung der Teilprobleme. Wir betrachten hier nur den Auftrag $I_1 = (a_1, \ ..., a_8)$ für den Auftrag I_2 ist die Vorgangsweise analog.

Sei $\sigma^{-1} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 1 & 2 & 3 & 4 & 5 & 7 & 8 & 6 \end{pmatrix}$, dann gilt (%) und die Sortimente des Auftrages I_1 teilen sich wie folgt auf:

$$\mathcal{I}_1 = (a_1, \ ..., a_5), \ \mathcal{I}_2 = (a_7, a_8), \ \mathcal{I}_3 = (a_6), \ \mathcal{I}_4 = \emptyset$$

Analog zum vorherigen Beispiel arbeitet $f_{1,1}$ auf den Indizes des Auftrags $\mathcal{I}_1$ usw.
Geeignete Lösungsfunktionen seien

$f_{1,1} \circ \Pi_{1,1}$:

i	$f_{1,1}(\Pi_{1,1}(i))$
1	1
2	2
3	1
4	1
5	2

,

$f_{1,2} \circ \Pi_{1,2}$:

i	$f_{1,2}(\Pi_{1,2}(i))$
1	3
2	3

$f_{1,3} \circ \Pi_{1,3}$:

i	$f_{1,3}(\Pi_{1,3}(i))$
1	4

mit geeignetem $\Pi_{1,1}$, $\Pi_{1,2}$ und $\Pi_{1,3}$.
Dann haben wir für $\mathcal{I}_1$: $x_{1,1} = 2$ für $\mathcal{I}_2$: $x_{1,2} = 1$ und für $\mathcal{I}_3$: $x_{1,3} = 1$ Behälter.

Weiters ist $f_1 : (f_{1,1} \sqcup f_{1,2} \sqcup f_{1,3}) \circ \sigma$, $\Pi_1 : (\Pi_{1,1} \sqcup \Pi_{1,2} \sqcup \Pi_{1,3}) \circ \sigma$, $x_1 = x_{1,1} + x_{1,2} + x_{1,3}$ eine Lösung des Ausgangsproblems.

2.3.3 Behandlung der Restriktion 3)

In Abschnitt 2.3.2 haben wir das Teilproblem $A(m_i, I_i, k_i, B_i)$ auf weitere Teilprobleme $A(p_j, \mathcal{I}_j, k_{i,j}, B_{i,j})$ zurückgeführt. Wir behandeln nun die Lösung dieser Teilprobleme.

Wir betrachten nun einen einzelnen Auftrag $\mathcal{I}_j$ und wir berechnen für diesen Auftrag eine Lösung $(x_{i,j}, f_{i,j})$ von $A(p_j, \mathcal{I}_j, k_{i,j}, B_{i,j})$, die die Restriktion 4) erfüllt. Wir berechnen daraus eine Lösung, die die Restriktion 3 erfüllt. Dann gilt für jede Permutation Π_i die Restriktion 4), da eine Sortierung nichts am Gewicht oder Volumen ändert. Dann kann eine Permutation $\Pi_{i,j}$ gewählt werden, die 3) erfüllt. Dann ist $(x_{i,j}, f_{i,j}, \Pi_{i,j})$ eine Lösung von $A(p_j, \mathcal{I}_j, k_{i,j}, B_{i,j})$, die die Restriktionen 3) und 4) erfüllt.

Beispiel: (Fortsetzung)
Wir behandeln nun die Lösung der Teilprobleme. Wir betrachten hier nur das Teilsortiment $\mathcal{I}_1 = (a_1, \ldots, a_5)$ des Auftrages I_1. Für die anderen Teilsortimente ist die Vorgangsweise analog.
Eine mögliche Lösungsfunktion sei

$f_{1,1}$:

i	$f_{1,1}(i)$
1	1
2	2
3	1
4	1
5	2

mit $x_{1,1} = 2$ Behälter.

Nun wählen wir ein $\Pi_{1,1}$ so, dass Restriktion 3 erfüllt ist. Wir nehmen an, dass schwer vor leicht gilt. Dazu sei nun:

$$g(a_1) = 3\,\text{kg},\ g(a_2) = 5\,\text{kg},\ g(a_3) = 10\,\text{kg},\ g(a_4) = 7\,\text{kg},\ g(a_5) = 4\,\text{kg}$$

Wähle nun $\Pi_{1,1} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 2 & 4 & 1 & 5 \end{pmatrix}$

Dann ist die Lösung:

$f_{1,1} \circ \Pi_{1,1}$:

$\Pi_{1,1}(i)$	$f_{1,1}(\Pi_{1,1}(i))$
3	1
2	2
4	1
1	1
5	2

mit $x_{1,1} = 2$ Behälter

2.3.4 Behandlung der Restriktion 4)

Diese Restriktion lässt sich nicht vereinfachen und muss modelliert werden.

Damit bleibt ein Kernproblem bestehen, welches in der Literatur als Bin-Packing Problem bekannt ist, und folgendermaßen formuliert werden kann:

Problem Bin–Packing

Eingabe:

$k \in \mathbb{N}$ Behälter $y \in B^k$ und ein Auftrag $J \in S^m$

Zulässige Lösung:

$x \in \mathbb{N}$ mit $x \leq k$ und $f : \{1, \ldots, m\} \to \{1, \ldots, x\}$ mit $Packing(f, x, J, y)$

Minimiere die Anzahl der Behälter:

Minimiere x, d.h.

$$\forall\, z < x \neg\, \exists\, f : \{1, \ldots, m\} \to \{1, \ldots, x\} : \text{Packing}(f, z, J, y)$$

Bemerkung :

Bei den Bin - Packing Problemen unterscheiden wir noch zusätzlich in Online - und Offline Verfahren :

Online Verfahren :

Die Objekte kommen nacheinander und müssen sofort gepackt werden, ehe das nächste betrachtet wird. Darüber hinaus ist nichts über die Sequenz, mit der die Artikel der Reihe nach ankommen bekannt, und nachträgliches Umsortieren ist nicht gestattet.

Offline Verfahren:

Hier ist im Vorhinein alles über die Objekte und deren Anzahl bekannt. Dies bedeutet, dass hier die Sequenz geeignet vorsortiert werden kann. Das ermöglicht im Allgemeinen bessere Ergebnisse als bei der Online Variante. (In unserem Fall liegt diese Variante vor, da wir alle Artikel zu Beginn bereits kennen.)

Im Folgenden werden wir zwei Beispiele zeigen, eine optimale Packung (wie wir sie erreichen wollen) und eine nicht optimale Packung. Der Übersicht halber zeigen wir nur die wichtigen Komponenten. Weiters sei bemerkt, dass die Werte ohne Rücksicht auf die Realität getroffen wurden.

Beispiel :

Wir betrachten den Auftrag $I_1 = (a_1, \ldots, a_5)$ für den gilt:

$$g(a_1) = 3\,\text{kg},\ g(a_2) = 5\,\text{kg},\ g(a_3) = 10\,\text{kg},\ g(a_4) = 7\,\text{kg},\ g(a_5) = 4\,\text{kg}$$

$$v(a_1) = 1000\,\text{cm}^3,\ v(a_2) = 200\,\text{cm}^3,\ v(a_3) = 100\,\text{cm}^3,\ v(a_4) = 700\,\text{cm}^3,\ v(a_5) = 200\,\text{cm}^3$$

Es seien genügend Behälter y_k mit jeweils $\overline{v}(y_k) = 2000\,\text{cm}^3$ und $\overline{g}(y_k) = 20\,\text{kg}$ gegeben.
Eine optimale Packung würde nun wie folgt aussehen:

f_1 :

i	$f_1(i)$
1	1
2	2
3	1
4	1
5	2

Somit sind im Behälter y_1 die Artikel a_1, a_3, a_4 enthalten und das Gesamtgewicht der Artikel in diesem Behälter wäre 20 kg und das Gesamtvolumen $1800\,\text{cm}^3$. Damit ist der Behälter optimal gefüllt.
Im Behälter y_2 sind die restlichen Artikel a_2 und a_5 enthalten, mit Gesamtgewicht 9 kg und Volumen $400\,\text{cm}^3$.

Eine nicht optimale Packung wäre die folgende:

f_2 :

i	$f_2(i)$
1	1
2	1
3	1
4	2
5	3

Wie wir sehen können, benötigt f_1 nur 2 Behälter während f_2 insgesamt 3 benötigt.

2.4 Literaturüberblick

Das Bin-Packing Problem ist unabhängig von der Dimension ein $\mathcal{NP}$-vollständiges Problem und das dazugehörige Optimierungsproblem ist $\mathcal{NP}$-schwer (Details dazu in Kapitel 3). Aus diesem Grund wird dieses Problem in der Literatur sehr häufig mit heuristischen Verfahren gelöst. Ein beliebter Ansatz sind sogenannte Greedy(=Gierige) Heuristiken (GH), die eine Lösung sukzessiv aufbauen, indem sie den Zielfunktionswert in jedem Schritt lokal optimieren. Immer häufiger werden metaheuristische Verfahren, wie zum Beispiel Simulated Annealing (SA) oder Tabu Search (TS) eingesetzt aber auch genetische Algorithmen sind ein oft gewählter Ansatz. Exakte Strategien wie Branch-and-Bound (B&B) und weitere Ansätze der Baumsuche (Branch-and-Cut(B&C) werden ebenfalls angewendet, jedoch nur bei sehr kleiner Anzahl der Artikel.

An dieser Stelle sei erwähnt, dass die Dimension des Bin-Packing Problems auf der Anzahl der betrachteten Eigenschaften der Artikel beruht. Wird nur eine Eigenschaft (beispielsweise Länge, Volumen, Gewicht) eines Artikels betrachtet, spricht man von einem eindimensionalen Bin-Packing Problem, wird zusätzlich noch eine zweite betrachtet (beispielsweise Länge und Breite oder Volumen und Gewicht) von einem zweidimensionalen Bin-Packing-Problem. Werden mehr als drei Eigenschaften betrachtet spricht man häufig auch von einem multi- oder mehrdimensionalen Bin-Packing Problem. Unser Problem ist ein zweidimensionales Problem, da wir das Volumen und das Gewicht betrachten.

Nachstehend in Tabelle 1 einige wichtige Literatur zu Bin-Packing Problemen.
Zuerst wird der Autor und dann das Publikationsdatum angegeben, daneben stehen die in der Literatur behandelte Dimension des Packing-Problems und das verwendete Verfahren. In der letzten Spalte ist die maximale Anzahl der Artikel, die getestet wurden, zu sehen.

Autor, Publikationsdatum	Dimension	Verfahren	Anzahl der Artikel
Bischoff und Ratcliff, 1995	3	GH	180
Martello und Vigo, 1995	2	B&B	100
Lodi et al. ,1998	2	TS	164
Martello et al. ,2000	3	B&B	90
Wang und Valenzuela, 2001	2	GA,GH	5000
Pisinger und Sigurd, 2007	2	B&C	100
Kratika Chandra et al. , 2014	2	GH	500

Tabelle 1 : Wichtige Literatur zu Bin-Packing Probleme

Es gibt noch zahlreiche Literatur für dieses Problem, die aber zumeist nur das eindimensionale Bin-Packing Problem (1D-BBP) behandeln oder spezielle Annahmen treffen. Meistens werden auch eher wenige Artikel verpackt, was der Realität nicht entspricht. Eine der wenigen Ausnahmen bildet die Publikation von Wang und Valenzuela, 2001, die bis zu 5000 Artikel getestet haben.

Eine offene Frage für die Heuristik First-Fit Decreasing (FFD) hat [Dósa, 2013] beantwortet.
Er hat die optimale Schranke für die maximale Anzahl an benötigten Behälter für das Bin-Packing Problem bzgl. der GH FFD gezeigt.

[Kratika Chandra, et al., 2014] hat statistisch gezeigt, dass der Best-Fit Algorithmus eine sehr gute Strategie für das Bin-Packing Problem ist.
Auf Grund der letzten beiden genannten Publikationen haben wir die beiden Heuristiken FFD und BFD näher betrachtet.

3. Theoretische Grundlagen

In diesem Kapitel werden die theoretischen Grundlagen für unser Problem gelegt. Zuerst wird aufgezeigt, dass es sich bei dem Bin-Packing Problem (BBP) um ein $\mathcal{NP}$-vollständiges Problem handelt. Danach wird die lineare Optimierung vorgestellt und dabei das BBP als ein lineares Optimierungsproblem modelliert. Dann werden wir Lösungsmethoden für lineare und ganzzahlige Optimierungsprobleme vorstellen. Dabei werden wir auf das B&B Verfahren auf SA und auf GH wie zum Beispiel FFD und BFD näher eingehen.

3.1 Komplexitätstheorie

Zuerst führen wir anschaulich einige Begriffe der Komplexitätstheorie ein, um dann im Anschluss zeigen zu können, dass unser Bin-Packing Problem ein so genanntes $\mathcal{NP}$-vollständiges Problem ist.

Wir machen dies nur informell, da dieses Thema zu umfangreich ist, um es im Rahmen dieser Masterarbeit bearbeiten zu können. Die Basis zu diesem Kapitel bilden die 4 Bücher [Korte&Vygen, 2008], [Gritzmann, 2013], [Bockenhauer et al., 2003] und [Joswig&Theobald, 2008].

3.1.1 Komplexität von Algorithmen

Algorithmen beurteilt man im Allgemeinen nach Rechenzeit oder Speicherplatzbedarf. Dieser Ressourcenbedarf wird in Abhängigkeit von den Inputdaten gemessen.
Die Zeitkomplexität $t_A(n)$ eines Algorithmus A wird bei uns als die maximale Anzahl von elementaren Schritten $steps(A_i)$ definiert, die A zur Lösung einer Instanz i eines Problems der Größe n benötigt. Dabei ist die Instanz eines der vielen verschiedenen Probleme, auf die der Algorithmus angewendet werden kann. Sei $\mathcal{I}_n$ die Klasse der Inputs der Größe n, dann können wir $t_A(n)$ auf folgende Weise definieren:

$$t_A(n) := \max_{i \in \mathcal{I}_n} steps(A_i)$$

Hier zählen wir also die maximale Anzahl an Schritten, die der Algorithmus im schlimmsten Fall für das Lösen der Instanz i benötigt. Aus diesem Grund wird diese Art der Laufzeitberechnung auch worst-case Laufzeit genannt. Es sei bemerkt, dass es auch noch weitere Arten von Zeitkomplexitäten gibt, beispielsweise die durchschnittliche Anzahl an Rechenschritten, diese würde dann wie folgt aussehen:

$$t_A(n) := \frac{1}{|\mathcal{I}_n|} \sum_{i \in \mathcal{I}_n} steps(A_i)$$

Falls nicht explizit erwähnt, sprechen wir stets von der ersten eingeführten Zeitkomplexität. Weiters wird angemerkt, dass elementare Schritte beispielsweise Variablenzuweisungen einfache arithmetische Operationen wie Addition, Subtraktion, Multiplikation, Division und auch der Zahlenvergleich sind. Ebenso zählen bedingte Sprünge wie (if-then, go to) zu solchen elementaren Schritten.

Da eben eine genaue Angabe der Komplexität von A meist nicht möglich ist, können wir zumindest das Wachstum der Funktion $t_A(n)$ in Abhängigkeit der Inputgröße n der Instanz i beschränken. Dies

geschieht, indem wir uns nur auf die dominanten Terme konzentrieren und konstante Faktoren außer Acht lassen. Dies ist natürlich ein grobes Maß, aber dadurch wird die Analyse von Algorithmen stark vereinfacht. Dies wird auch als *asymptotische Analyse* bezeichnet. vgl. [Joswig&Theobald, 2008, S.245]

Definition 2

$$O(g) := \{ f \mid f : \mathbb{N} \to \mathbb{R}^+, \exists\, c > 0\, \exists\, n_0 \in \mathbb{N}\, \forall\, n \geq n_0 : f(n) \leq c\, g(n) \}$$

$f \in O(g)$ (manchmal auch $f = O(g)$ geschrieben) bedeutet dann, dass f asymptotisch nicht schneller wächst als $g(n)$, wobei ebenfalls $g : \mathbb{N} \to \mathbb{R}^+$.

Mit anderen Worten: $f \in O(g)$ falls für ein genügend großes n eine Konstante $c > 0$ existiert, so dass $f(n)$ nicht größer als $c * g(n)$ ist.

Beispiel: $f(n) = 3n + 5 \leq 4n$ für alle $n \geq 5$ und somit gilt $f(n) \in O(n)$

An dieser Stelle geben wir noch ein kurzes Beispiel zur Laufzeit, um zu verdeutlichen, wie stark sich diese unterscheiden.

Beispiel: (Laufzeiten)
Bei 10^9 ausführbaren Rechenoperationen pro Sekunde benötigt man für:

* $1000^3 \sim 1$ Sekunde zur Berechnung.
* $20! \sim 80$ Jahre zur Berechnung.
* $2^{100} \sim 3000$ mal dem Alter des Universums zur Berechnung

Für Polynome der Form $P(n) = a_0 + a_1 n + \ldots + a_m n^m$ gilt :

$$P(n) \in O(n^m)$$

Beweis der letzten Aussage:

Nach Definition 2 ist zu zeigen:

$$P(n) \leq c\, g(n) \Leftrightarrow a_0 + a_1 n + \ldots + a_m n^m \leq c\, n^m \quad \text{für gewisse } c > 0 \text{ und } n \geq n_0$$

Wir beginnen mit dem Polynom:

$$P(n) = a_0 + a_1 n + \ldots + a_m n^m = n^m \left(\frac{a_0}{n^m} + \frac{a_1}{n^{m-1}} + \ldots + a_m \right) \leq (a_o + \ldots + a_m)\, n^m$$

Die letzte Abschätzung gilt nur unter der Voraussetztung $n \geq 1$. Dies gilt da: $n \geq n_0$ und $n_0 \in \mathbb{N}$.
Wenn wir nun wählen:

$$c = \sum_{i=0}^{m} a_i, \qquad (*)$$

dann können wir immer eine Konstante c finden, mit der $n_0 = 1$ ist. Denn:

$$P(n) \leq (a_o + \ldots + a_m)\, n^m \overset{(*)}{=} c\, n^m \leq c\, n^m$$

■

Dies bedeutet nun also, dass bei einem Polynom nur der Grad des höchsten Exponenten eine Rolle spielt.

Beispiel Sortierung:
Betrachten wir SelectionSort (SS). Der Input sei eine Liste von n Zahlen.
Prinzip:
Sei L die zu sortierende Liste S der sortierte Teil der Liste (vorne in L) und U der unsortierte Teil (dahinter). Zu Beginn gilt : $S = \emptyset,\ U = L$.

Der Algorithmus wählt in U das Minimum und fügt es S hinzu (indem er es mit dem ersten Element in U vertauscht und U danach um eine Position später beginnt). Nun ist L bis zu dieser Position sortiert. Anschließend wird diese Methodik solange wiederholt, bis L vollständig abgearbeitet worden ist. Am Ende gilt: $U = \emptyset$, $S = L$ aufsteigend sortiert.

Komplexität:
Im Schritt 1 sind $n-1$ Vergleiche nötig. Im Schritt 2 nur mehr $(n-1)-1 = n-2$ Vergleiche. Im Schritt $n-1$ ist nur mehr 1 Vergleich nötig.
In Summe sind dies also :$(n-1) + (n-2) + \ldots + 2 + 1 = \frac{n(n-1)}{2} = \frac{n^2}{2} - \frac{n}{2}$ Vergleiche plus $n-1$ Vertauschungen.
Demnach ist die Anzahl der Rechenschritte durch ein quadratisches Polynom begrenzt.
Somit gilt: $t_{SS}(n) \in O(n^2)$

3.1.2 Komplexitätsklassen $\mathcal{P}$ und $\mathcal{NP}$

In diesem Abschnitt widmen wir uns kurz den beiden wichtigsten Komplexitätsklassen $\mathcal{P}$ und $\mathcal{NP}$ und werden diese anhand von Beispielen näher erläutern, danach werden wir noch die schwersten Probleme der Komplexitätsklasse $\mathcal{NP}$ einführen, die sogenannten $\mathcal{NP}$-vollständigen Probleme. Diese sind so schwer, dass es vermutlich keinen Algorithmus gibt, der diese Probleme effizient löst. Effizient bedeutet in polynomialer Zeit.

Definition 3
Ein Algorithmus A hat eine **polynomielle Laufzeit** oder ist **polynomiell**, falls es ein $k \in \mathbb{Z}$ gibt, sodass gilt:

$$t_A(n) \in O(n^k)$$

In der nächsten Definition wird der Begriff des Entscheidungsproblems benötigt. Für eine genaue Definition eines Entscheidungsproblems sei an dieser Stelle auf [Korte&Vygen, 2008, S. 396] verwiesen. Wir werden diesen Begriff hier nur informell betrachten.
Bei einem Entscheidungsproblem wird der Input auf eine Bedingung getestet. Ist diese Bedingung erfüllt, so wird “ja” / 1 ausgegeben, anderenfalls “nein” / 0. Um ein Entscheidungsproblem eindeutig zu beschreiben, reicht es aus, die Menge der zulässigen Eingaben und die zu testende Bedingung anzugeben.

Definition 4
Die Klasse aller Entscheidungsprobleme, für die es einen polynomiellen Algorithmus gibt, wird mit $\mathcal{P}$ bezeichnet. [Korte&Vygen, 2008, S.396]

Ein Beispiel für ein Problem in $\mathcal{P}$ sind lineare Ungleichungen. Einen Beweis dafür gibt der Satz von Khachiyan, 1979 und ist in [Korte&Vygen, 2008, S.97] zu finden.
Ein weiteres Beispiel für ein Problem in $\mathcal{P}$ ist der Primzahltest: Für ein $n \in \mathbb{N}$ entscheide, ob n eine Primzahl ist oder nicht.

Als Nächstes benötigen wir den Begriff des Nichtdeterminismus. Auch hier wird dieser wieder informell eingeführt. Für eine Einführung in die Theorie des Nichtdeterminismus wird an dieser Stelle auf [Gritzmann, 2013, S. 224.ff] verwiesen.
Ein deterministischer Algorithmus hat für einen Input immer denselben Output. Das bedeutet, dass in jedem Zeitschritt der nachfolgende Arbeitsschritt des Algorithmus eindeutig bestimmt ist. Bei nichtdeterministischen Algorithmen ist dem nicht so, diese haben eine Freiheit bei der Wahl der folgenden Arbeitsschritte. Ihre einzelnen Arbeitsschritte sind daher nicht nachvollziehbar, bzw. man kann nicht nachvollziehen, warum gerade ein spezifischer Schritt ausgeführt wird. Aus diesem Grund ist bei gleichem Input stets ein anderer Output zu erwarten.

Definition 5
Ein Entscheidungsproblem ist genau dann in $\mathcal{NP}$, wenn es einen polynomiellen nichtdeterministischen Algorithmus hat. [Korte&Vygen, 2008, S.398]

Eine äquivalente Charakterisierung wäre, dass ein Entscheidungsproblem genau dann in $\mathcal{NP}$ liegt, wenn es einen deterministischen Algorithmus gibt, der eine Lösung für das korrespondierende Ausgangsproblem in polynomialer Zeit verifizieren kann. Wir haben die erstere gewählt, da diese in der Literatur zumeist verwendet wir.

Bemerkung:

Für ein Problem in $\mathcal{P}$ kann man eine Lösung in polynomieller Zeit finden. Anders ausgedrückt: Man kann ein Problem in polynomieller Zeit entscheiden.
Ein einem Ausgangsproblem zugeordnetes Entscheidungsproblem liegt genau dann in $\mathcal{NP}$, wenn es einen deterministischen Algorithmus gibt, der eine Lösung für das Ausgangsproblem in polynomieller Zeit verifizieren kann.

Für unser BBP sieht das folgendermaßen aus:
Das Ausgangsproblem:
Finde eine Zuordnungsfunktion f, die m Artikel auf k Behälter aufteilt, wobei k minimal sein soll.

Das dazugehörige Entscheidungsproblem:
Gibt es eine Zuordnungsfunktion f, die m Artikel auf k Behälter aufteilt, wobei k gegeben ist.

Redewendungen:
Wenn wir sagen, dass ein Problem in $\mathcal{NP}$ liegt, meinen wir stets das zu dem Ausgangsproblem korrespondierende Entscheidungsproblem. Weiters sprechen wir des Öfteren von polynomialer oder exponentieller Zeit. Diese Zeiten beziehen sich stets auf die Größe des Inputs. Wenn wir also sagen: Wir können eine Lösung in exponentieller Zeit finden, dann meinen wir immer in Abhängigkeit von der Größe des Inputs. Meistens ist damit die Anzahl der Elemente einer Liste gemeint.

Beispiel: ($\mathcal{NP}$ – Problem)**:**
Wir betrachten hier das Partitionsproblem (Partition).

Eingabe: $a_1, \ ..., \ a_N \in \mathbb{N}$ paarweise verschieden
Frage: Gibt es ein $K \subseteq \{1, \ ..., N\}$ mit $\sum_{i \in K} a_i = \sum_{i \in \{1,...,N\} \setminus K} a_i$

Z.B.:
Gegeben sei eine Liste mit den folgenden Einträgen $L := (3, 1, 4, 5, 7, 10)$.

Frage:
Gibt es nun eine Aufteilung auf 2 Listen L_1, L_2, sodass sich die Elemente in den beiden Listen zu 15 summieren?

Z.B. $L_1 = (1, \ 4, \ 10)$ und $L_2 = (3, 5, \ 7)$ wäre eine Lösung.

Eine Lösung zu überprüfen, lässt sich in polynomieller Zeit erledigen, eine zu finden, jedoch nur in exponentieller Zeit $O(2^N)$! Im schlimmsten Fall muss man alle Möglichkeiten ausprobieren und in unserem Fall wären dies $\frac{(2^N-2)}{2} = 2^{N-1} - 1$.

Bemerkung:

Es gibt verschiedene Versionen von Partition. In der Literatur ist der Input von Partition auch häufig eine Menge.

Bemerkung:

Eine der größten Fragen in der Informatik ist es nun, ob $\mathcal{NP} = \mathcal{P}$ gilt. Diese Frage ist von so großer Bedeutung, dass dies auch eines der Millennium-Probleme ist. ($\mathcal{P}$-$\mathcal{NP}$-Problem) Falls denn $\mathcal{P} = \mathcal{NP}$ gilt, bedeutet dies nun, dass alle Probleme in $\mathcal{NP}$ effizient gelöst werden können, was für viele praktische Aufgaben von immenser Bedeutung wäre, allerdings ist dies nicht in jedem Bereich erwünscht! Beispielsweise in der Kryptologie, denn auch das Knacken gängiger Verschlüsselungscodes ist ein $\mathcal{NP}$-Problem. Würde $\mathcal{P} = \mathcal{NP}$ gelten, bedeute dies, dass man die Verschlüsselungen effizient aushebeln könnte.

Folgende Abbildung zeigt den möglichen Zusammenhang zwischen den Komplexitätsklassen.

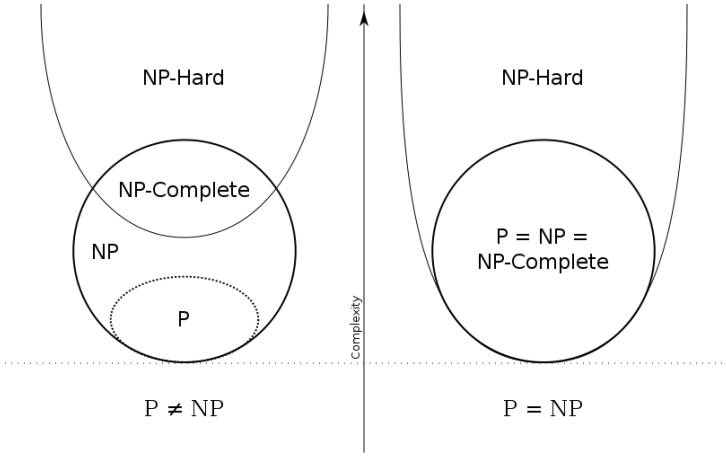


Abbildung 8. Möglicher Zusammenhang der Komplexitätsklassen [4]

Viele kombinatorische Optimierungsprobleme sind in $\mathcal{NP}$, und da wir nicht wissen, ob $\mathcal{P} = \mathcal{NP}$ gilt, können wir auch nicht sagen, ob es einen polynomielle Algorithmus gibt. Es ist jedoch möglich zu zeigen, dass ein Problem nicht leichter als ein anderes ist. Damit wir diese Idee nun verdeutlichen, führen wir nachfolgend den sehr wichtigen Begriff der polynomiellen Reduktion ein.

Definition 6

Seien P_1, P_2 zwei Probleme. Eine Funktion ω, die jedem Input I für P_2 eine Lösung zuordnet, heißt **Orakel** für P_2, wenn es ein Polynom $\pi : \mathbb{N}_0 \to \mathbb{N}_0$ gibt, so dass für alle $I : size(\omega(I)) \leq \pi(size(I))$. Wir sagen, dass ein Algorithmus A für P_1 ein Orakel ω für P_2 aufruft, wenn A einen Input I von P_2 berechnet und ihm (also A) dann das Ergebnis $\omega(I)$ zur Verfügung steht.

P_1 heißt polynomiell reduzierbar auf P_2, wenn es einen Algorithmus A gibt, der P_1 mittels polynomiell vieler elementarer Operationen auf Zahlen polynomieller Größe und polynomiell vieler Aufrufe eines Orakels ω für P_2 löst. Ist A ein solcher Algorithmus, so heißt A **polynomielle Reduktion** von P_1 auf P_2. Wir verwenden dafür die Schreibweise: $P_1 \leq_p P_2$.

Vgl.[Griezmann, 2013, S.140] oder [Korte, 2008, S.399]

Der nächste Satz liefert den Grund für die eben eingeführte Definition.

Satz 2

Gilt $E_1 \leq_p E_2$ mit $E_2 \in \mathcal{P}$, dann gilt auch $E_1 \in \mathcal{P}$

Beweis: siehe [Korte&Vygen, 2008, S.399]

■

Definition 7

Ein Optimierungs- oder Entscheidungsproblem E heißt $\mathcal{NP}$-schwer, falls sich jedes Problem in $\mathcal{NP}$ polynomiell auf E reduziert. [Korte&Vygen, 2008 , S.414]

Nun definieren wir die schwersten Probleme aus der Klasse $\mathcal{NP}$, die $\mathcal{NP}$-vollständigen Probleme.

Definition 8
Ein Entscheidungsproblem E heißt $\mathcal{NP}$-vollständig genau dann, wenn:

1) $E \in \mathcal{NP}$
2) E ist $\mathcal{NP}$-schwer

Auf einer besondere Variante von Satz 3 basiert die Theorie der $\mathcal{NP}$-Vollständigkeit:

Satz 3
Ist $E_1 \in \mathcal{NP}$ und gilt $E_2 \leq_p E_1$ mit E_2 ist $\mathcal{NP}$-vollständig, so ist auch E_1 $\mathcal{NP}$-vollständig.

Beweis: siehe [Blömer, 2010, S.30]

■

Beispiel: ($\mathcal{NP}$ – vollständiges Problem)
Wir betrachten hier das Teilsummenproblem (Subset – Sum).

Eingabe : Eine Menge M und $b \in \mathbb{N}$
Frage : Gibt es ein $K \subseteq M$, sodass $\sum_{i \in K} i = b$?

Z.B. ist es möglich, aus dieser Menge $M = \{24, 3, 8, 7, 16, 9, 10\}$ eine Teilmenge zu finden, deren Summe 30 ergibt. Hier würde die Antwort "Ja" lauten denn : $\{3, 8, 9, 10\}$.

Auch hier ist es wieder möglich, eine Lösung in polynomieller Zeit zu verifizieren, eine zu finden, würde jedoch $O(2^N)$ dauern, da man im schlimmsetn Fall alle möglichen Teilmengen (Potenzmenge) bilden muss .

Ein Beweis ist bspw. in [*M*.Sipser, 1997, *S* .291.*f*] gegeben.

Folgender Satz erleichtert uns, die $\mathcal{NP}$-Schwere zu zeigen, indem man sich die Transitivität der polynomiellen Reduktion zu nutze macht:

Satz 4
Ist $E_1 \leq_p E_2$ und $E_2 \leq_p E_3$, so gilt $E_1 \leq_p E_3$.

Beweis: siehe [Blömer, 2010, S.26]

■

Nun zeigen wir, dass das zuvor gezeigt Problem Partition $\mathcal{NP}$-Vollständig ist:

Problem Partition:

Eingabe: $a_1, \ldots, a_N \in \mathbb{N}$ paarweise verschieden
Frage: Gibt es ein $K \subseteq \{1, \ldots, N\}$ mit $\sum_{i \in K} a_i = \sum_{i \in \{1,\ldots,N\} \setminus K} a_i$

Partition ist ein Spezialfall von Subset-Sum, indem man die Frage wie folgt formuliert:

Gibt es eine Teilmenge K mit Summenwert $\frac{1}{2} \sum_{i=1}^{N} a_i$.

Damit wäre Partition $\leq_p$Subset-Sum gezeigt. Leider müssen wir im nächsten Satz aber die umgekehrte Richtung zeigen!

Satz 5
Partition ist $\mathcal{NP}$-vollständig.

Beweis:
Um zu zeigen, dass Partition $\mathcal{NP}$-vollständig ist, müssen wir nachweisen

1. Partition $\in \mathcal{NP}$
2. Partition ist $\mathcal{NP}$-schwer

Partition $\in \mathcal{NP}$ ist trivial, da es ein Spezialfall von Subset-Sum (:= SS) ist.

Nun zum interessanten Teil:
Für die $\mathcal{NP}$-Schwere müssen wir ein bekanntes Problem darauf reduzieren. (Satz 4)
Beispielsweise könnten wir zeigen:

Subset-Sum $\leq_p$Partition

Hierzu müssen wir es irgendwie schaffen, dass sich die zu bildende Teilsumme genau auf die Hälfte der Gesamtsumme reduzieren lässt.

Die Eingabe von SS sei $I = \{a_1, \ldots, a_n, b\}$ mit $a_i \in \mathbb{N}$ und $b \in \mathbb{N}$.

Es sei $A = \sum_{i=1}^{n} a_i$.

Wir bilden diese Eingabe für SS auf eine Eingabe für Partition ab, die aus $n+2$ Zahlen $a_1, \ldots, a_{n+2}$ bestehe. Die Eingabe für Partition sei $I' = (a_1, \ldots, a_{n+2})$ wobei $a_{n+1} = b + 1$ und $a_{n+2} = A - b + 1$ Ergänzungsvariablen sind.

Es sei $A' = \sum_{i \in I'} a_i = A + (b+1) + (A - b + 1) = 2A + 2$ die Gesamtsumme aller Zahlen in I'.

Die Reduktion $I \mapsto I'$ ist offensichtlich polynomiell zeit-beschränkt. Nun muss noch die Korrektheit

der Reduzierung gezeigt werden:

Also : Es gibt eine Auswahl $\mathcal{I} \subseteq \{1, \ldots, n\}$ mit $\sum_{i \in \mathcal{I}} a_i = b$,

wenn sich die Zahlen aus I' in zwei Mengen mit der gleichen Teilsumme $\frac{A'}{2} = A + 1$ zerlegen lassen.

Seien zwei Mengen M_1, M_2 mit gleicher Teilsumme $A + 1$ gegeben ist.

Die Addition der Hilfsvariablen zeigt uns :

$$a_{n+1} + a_{n+2} = A + 2 > A + 1$$

daher liegen a_{n+1} und a_{n+2} nicht in der gleichen Teilmenge

Also gilt o.B.d.A :

$$a_{n+1} \in M_1 \ \wedge \ a_{n+2} \in M_2.$$

Da sich die Mengen M_1 (sowie M_2) zu derselben Teilsumme $A + 1$ aufaddieren, müssen sich folglich die Elemente aus $\underbrace{M_2 \cap \{a_1, \ldots, a_n\}}_{A+1-(A-b+1)=A-b}$ zu b addieren, somit ist $\mathcal{I} = \{ i \mid a_i \in M_2\}$ eine Lösung für SS.

■

Nun haben wir alles zusammen, um uns unserem BBP zu widmen, siehe Seite 27.

Satz 6
Bin-Packing ist $\mathcal{NP}$-vollständig.

Beweis:
Dies folgt aus einer trivialen Reduktion von Partition auf Bin Packing, denn Bin Packing ist ein Spezialfall von Partition.

Also zeige: Partition $\leq_p$ Bin Packing

Aus Lösung von BPP muss Lösung für Partition konstruiert werden. Der Einfachheit halber konzentrieren wir uns nur auf die Gewichtsrestriktion selbiges gilt natürlich auch für die Volumenrestriktion.

Gegeben : Sei f eine Lösung von BPP $J = (a_1, \ldots, a_n)$, $y \in B^k$
Gesucht : $K = \{i \mid f(i) = 1\}$

Dazu wähle : $k = 2$ und $\overline{g}(y_j) = \frac{1}{2} \sum_{i=1}^{n} g(a_i)$ für $j = 1, 2$

Die Restriktionen $\sum_{f(i)=1}^{n} g(J_i) \leq \overline{g}(y_1)$ und $\sum_{f(i)=2}^{n} g(J_i) \leq \overline{g}(y_2)$ können nur mit Gleichheit erfüllt sein,
was in einer Lösung für Partition resultiert.

■

3.2 Lineare Optimierung

Wie wir bald sehen werden, kann das Bin-Packing Problem als lineares Optimierungsproblem aufgefasst werden. Lösungen für lineare Optimierungsprobleme werden schon seit jeher gesucht, und die dabei verwendeten Methoden wurden anfänglich während des zweiten Weltkrieges für militärische Zwecke entwickelt. Später erkannte man aber auch ihren wirtschaftlichen Nutzen und so assoziiert man heutzutage den Begriff des Operation Research (OR) mit einer Vielzahl an mathematischen Methoden zur Behandlung von wirtschaftlichen Fragestellungen.

Unter anderem zählen dazu :
*Lineare Optimierung,
*Nichtlineare Optimierung,
*Graphentheorie,
*Monte-Carlo-Simulation
*Spieltheorie
vgl. [Koop&Mook, 2018, S.1.f]

Der erste Punkt, die lineare Optimierung, was eines der Hauptverfahren des OR ist, dient für uns als Grundlage für die algorithmischen Ansätze des Bin-Packing Problems und wird nun im Folgenden näher behandelt. Später werden wir auch noch verschiedene Varianten der linearen Optimierung betrachten.

3.2.1 Lineares Programm

Ein lineares Optimierungsproblem (auch lineares Programm oder linear program (LP) genannt) besteht im Grunde aus einer Zielfunktion und aus den dazugehörigen Restriktionen (Nebenbedingungen). Der wesentliche Punkt der linearen Optimierung ist es, dass die Zielfunktion sowie die Restriktionen linear sind. Wir werden als Nächstes die Standardform eines linearen Optimierungsproblems angeben. Doch zuvor wird noch die Bedeutung der Notation in der folgenden Definition erklärt:
Wir haben n Variablen und m Restriktionen der Form $a_i^T x \leq \beta_i$ mit $1 \leq i \leq m$, wobei das β_i auch oft einfach “rechte Seite” genannt wird. Das ϕ nennt man die Zielfunktion und die dazugehörigen γ_i nennt man die Zielfunktionskoeffizienten. Ist bei der Standardform zusätzlich $\beta_i \geq 0$, so sagt man auch das LP liegt in kanonischer Form vor.

Definition 9 (Standardform eines linearen Optimierungsproblems)
Bei einem LP (über $\mathbb{R}$) sind folgende Daten gegeben:

$$m, n \in \mathbb{N} \quad \wedge \quad a_1, \ldots, a_m \in \mathbb{R}^n \quad \wedge \quad \beta_1, \ldots, \beta_m \in \mathbb{R} \quad \wedge \quad \gamma_1, \ldots, \gamma_n \in \mathbb{R}.$$

Ferner sei das lineare Funktional $\phi : \mathbb{R}^n \to \mathbb{R}$ durch

$$\phi(x) := \sum_{i=1}^{n} \gamma_i x_i$$

definiert, und es sei

$$F := \{x \in \mathbb{R}^n : a_1^T x \leq \beta_1 \wedge \ldots \wedge a_m^T x \leq \beta_m\}$$

die Menge der zulässigen Lösungen (oder auch zulässiger Bereich).
Gesucht ist:

$$\bar{x} \in F \text{ mit } \phi(\bar{x}) = \min_{x \in F} \phi(x)$$

Diese Standardform ist nicht, wie man vermuten könnte, allgemeine Konvention. In der Literatur findet man viele verschiedene Definitionen dafür, die aber alle durch die in Abschnitt 3.2.1.1 gezeigten Methoden ineinander übergeführt werden können.

Oft werden die Daten der LPs durch die Setzung

$$A := (a_1, \ldots, a_m)^T \in \mathbb{R}^{m \times n},\ b := (\beta_1, \ldots, \beta_m)^T \in \mathbb{R}^m,\ c := (\gamma_1, \ldots, \gamma_n)^T \in \mathbb{R}^n.$$

zusammengefasst, und die Aufgaben können dann kompakter geschrieben werden, wie zum Beispiel

$$\begin{aligned} &\min c^T x \\ &A x \leq b \end{aligned}$$

Die Ungleichung ist hier natürlich komponentenweise zu verstehen.
Wir sagen, ein Vektor $x \in \mathbb{R}^n$ ist eine *zulässige Lösung* für unser LP, wenn er $A x \leq b$ erfüllt. Der zulässige Bereich ist somit $F := \{x \in \mathbb{R}^n : A x \leq b\}$. Eine zulässige Lösung, für die das Minimum angenommen wird, heißt *optimale Lösung*.

Bemerkung:

Oft wird eine Nichtnegativitästbedingung $x_i \geq 0$ gefordert. Diese wird allerdings normalerweise nicht zu den m Restriktionen hinzugezählt, sondern gesondert betrachtet. Somit würde die Kurzschreibweise wie folgt aussehen:

$$\begin{aligned} &\min c^T x \\ &A x \leq b \\ &x \geq 0 \end{aligned}$$

3.2.1.1 Überführung in Standardform

Natürlich ist es möglich, dass wir zu Beginn ein Optimierungsproblem haben, welches nicht in Standardform vorliegt. Wir werden zeigen, wie wir solche Probleme auf unsere gewünschte Form zurückführen können.

Zum einen wäre denkbar, dass wir, anstatt zu minimieren, ϕ maximieren wollen. Hier können wir natürlich einfach $-\phi$ minimieren, was somit wieder einem Minimierungsproblem entspricht. Mit anderen Worten gilt:

$$\max_{x \in F} c^T x = -\min_{x \in F} (-c)^T x$$

Die $\geq$ und $\leq$ Restriktionen kann man durch Multiplikation mit -1 ineinander überführen. Eine Gleichungsbedingung $a^T x = \beta$ kann durch die beiden Ungleichungsbedingungen

$$a^T x \leq \beta \wedge a^T x \geq \beta$$

ersetzt werden. Haben wir nun Restriktionen, die strikt kleiner oder größer sind, so können wir dies mit Hilfe von so genannten Schlupfvariablen wieder auf Standardform bringen. Dieses Vorgehen funktioniert ebenfalls, wenn man Ungleichungen in Gleichungen überführen möchte. Wir zeigen dies am besten anhand eines kleinen Beispiels.

Beispiel:

Minimiere $x_1 + x_2$

unter den Nebenbedingungen:

$$x_1 + 3\,x_2 < 13$$
$$3\,x_1 + x_2 < 15$$
$$x_1 > 0,\ x_2 > 0$$

Wir führen die Schlupfvariablen x_3 und x_4 ein und ersetzten die kleiner Bedingung durch:

$$x_1 + 3\,x_2 + x_3 \leq 13$$
$$3\,x_1 + x_2 + x_4 \leq 15$$
$$x_i \geq 0 \text{ mit } i \in \{1,\ ..., 4\}$$

Bei der Überführung eines linearen Optimierungsproblems in eine mit Nichtnegativitätsbedingungen, die schließlich sehr oft gefordert wird, verwendet man folgenden Trick:

Aus einem gegebenen LP P erzeuge eine neues LP $\overline{P}$, indem jede freie Variable x_i durch die Differenz $\overline{e}_i - \underline{e}_i$ der beiden neuen Variablen ersetzt wird, und diese einer Nichtnegativitätsbedingung unterliegen, also $\overline{e}_i \geq 0$, $\underline{e}_i \geq 0$. Dadurch wird jeder zulässigen Lösung x des LPs P eine zulässige Lösung für das LP $\overline{P}$ zugeordnet. Setzt man umgekehrt

$$x_i := \overline{e}_i - \underline{e}_i$$

so führt jede Lösung x von $\overline{P}$ zu einer Lösung von P.

3.2.2 Varianten der linearen Optimierung

Bei der linearen Optimierung gilt für die Gesuchten: $x_i \in \mathbb{R}$. Hier wurde sogar gezeigt, dass man tatsächlich zum Lösen nur einen polynomialen Aufwand hat. Hier wird aber trotz des exponentiellen Aufwandes dennoch das Simplex-Verfahren verwendet, da dieses sehr schnell ist und der Nachteil des exponentiellen Aufwandes eher selten vorkommt. [Koop&Mook, 2018, S.99]

■ Ganzzahlige Optimierung

Hier gilt für die Gesuchten: $x_i \in \mathbb{Z}$. Diese Bedingung rührt meist aus der Praxis, da man hier meist nur ganze Einheiten produzieren will oder ganze Arbeiter zuteilen kann. Diese zusätzliche Einschränkung kann zu einem exponentiellen Aufwand zum Lösen eines solchen LPs führen. [Koop&Mook, 2018, S167] Geeignete Lösungsverfahren sind bspw. B&B, Schnittebenen-Verfahren sowie die B&C Methode.

■ Binäre Optimierung

In diesem Fall gilt für die Gesuchten: $x_i \in \{0, 1\}$. Dies ist ein Spezialfall der ganzzahligen Optimierung. Anwendungen sind hier bspw. die Modellierung von logischen Zusammenhängen. Auch die Modellierung des Bin-Packing Problems führt auf ein Problem dieser Art. Artikel i in Behälter j wird genommen $x_{ij} = 1$ andernfalls $x_{ij} = 0$.

■ 3.3 Alternative Modellierung des Bin-Packing Problems als ein lineares Optimierungsproblem

Um ein Problem zu erfassen, muss zuallererst ein mathematisches Modell aufgestellt werden, welches das Problem so gut wie möglich beschreibt. Dabei muss einerseits darauf geachtet werden, dass man --wenn möglich-- nicht die volle Komplexität des Problems in das Modell aufnimmt, da ansonsten das Problem schnell zu hohe Ausmaße annimmt und folglich eine Lösung in endlicher Zeit nicht mehr gefunden werden kann. Andererseits darf man das Problem auch nicht zu stark vereinfachen, da es in diesem Fall nicht die Realität widerspiegelt und somit unbrauchbar wird. Es muss somit ein Kompromiss bezüglich der Komplexität und der Vereinfachung in der Modellierung getroffen werden, um am Ende ein brauchbares Modell, welches nützliche Lösungen in annehmbarer Zeit findet, zu erhalten. Josef Kallrath hat dies gut zum Ausdruck gebracht: “die Modellentwicklung sollte dem Grundsatz ” So einfach wie möglich, so kompliziert wie nötig.” [...] folgen”. [Kallrath, 2013, S.8]

Weiters sei bemerkt, dass die Modellierung im OR eine zentrale Rolle spielt, da diese meist große Probleme verursacht, denn auf Grund des riesigen Anwendungsbereiches ist es kaum möglich, diese systematisch darzustellen.(vgl. [Koop&Mook, 2018, S.2])

Nachstehend sei der geschützte Planungsprozess des OR aufgezeigt. Wir haben diesen beispielhaft verwendet, um unsere Arbeit aufzubauen.

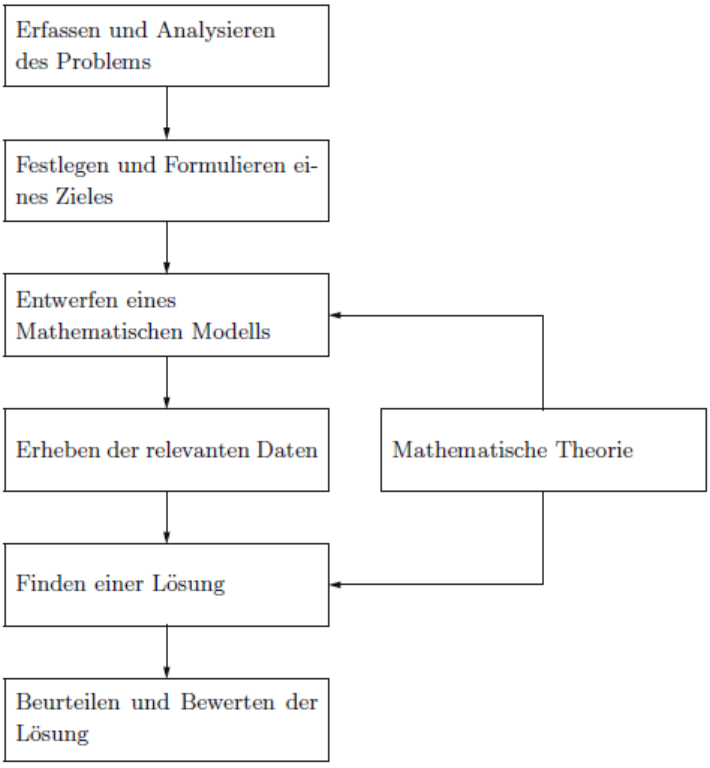


Abbildung 9. Geschützter Planungsprozess des OR [7, S.3]

Nun ist es an der Reihe, auf Basis der in den vorherigen Abschnitten erörterten Aspekte ein mathematisches Modell aufzustellen.

Problem: Bin-Packing als lineares Optimierungsproblem

Gegeben:

$k \in \mathbb{N}$ Behälter $y \in B^k$ und ein Auftrag $J \in S^m$

Gesucht:

Die minimale Anzahl der gefüllten Behälter (1.1) sowie die Zuordnung der Entscheidungsvariablen u_{ik}.

$$min \sum_{k=1}^{x} z_k \qquad (1.1)$$

unter den Nebenbedingungen:

$$(z_k - 1) \sum_{i=1}^{m} u_{ik} = 0 \qquad (k = 1, \ ..., x) \qquad (1.2)$$

$$\sum_{k=1}^{x} u_{ik} = 1 \qquad (i = 1, \ ..., m) \qquad (1.3)$$

$$\sum_{i=1}^{m} v(a_i)\, u_{ik} \;\leq\; \overline{v}(y_k) \qquad (k = 1,\ ...,\ x) \qquad (1.4)$$

$$\sum_{i=1}^{m} g(a_i)\, u_{ik} \;\leq\; \overline{g}(y_k) \qquad (k = 1,\ ...,\ x) \qquad (1.5)$$

$$u_{ik} \in \{0, 1\} \qquad (1.6)$$

$$z_k \in \{0, 1\} \qquad (1.7)$$

wobei:

$$u_{ik} = \begin{cases} 1 & \text{Artikel } i \text{ ist in Behälter } k \\ 0 & \text{sonst} \end{cases} \qquad (1.8)$$

$$z_k = \begin{cases} 1 & \text{Behälter } k \text{ wird mitgenommen} \\ 0 & \text{sonst} \end{cases} \qquad (1.9)$$

Erklärungen zum Modell:

(1.1) Ist die zu minimierende Zielfunktion. Hier versuchen wir, die minimale Anzahl an Bins zu finden.

(1.2) Ist eine Restriktion, die sicherstellt, dass z_k richtig gesetzt wird.

(1.3) Stellt sicher, dass alle Artikel verpackt wurden. (Auch Konvexitätsbedingung genannt)

(1.4) Ist eine klassische Restriktion gegen Überfüllung, sie sagt aus, dass ein Behälter nicht über seine Kapazität hinaus befüllt werden darf.

(1.5) Analog zu (1.4) nur für das Gewicht.

(1.6), (1.7) Sind sogenannte Entscheidungsvariablen, die entweder 1 oder 0 sind.

In (1.8) und (1.9) wurden die Entscheidungsvariablen definiert.

Für das Verständnis sei bemerkt, dass z_k nicht explizit im Modell gefordert werden muss, denn die Nebenbedingung (1.2) stellt diesen Zusammenhang sicher. Denn wäre einerseits der Ausdruck aus (1.2):

$$\sum_{i=1}^{m} u_{ik} = 0 \;\text{ mit } k \text{ fest}$$

würde dies bedeuten, dass keiner der m Artikel im Behälter k ist, womit dieser leer wäre. Damit ist für beliebiges z_k (1.2) sicherlich stets gleich null. Da die Zielfunktion aber minimiert werden soll, ist die einzige logische Wahl $z_k = 0$.

Wäre nun andererseits der Ausdruck:

$$\sum_{i=1}^{m} u_{ik} \neq 0 \;\text{ mit } k \text{ fest}$$

wäre ein Artikel im Behälter k und somit nicht leer. Die einzige Möglichkeit, damit (1.2) nun erfüllt wäre, ist $z_k - 1 = 0$. Damit erzwingen wir $z_k = 1$ für einen gefüllten Behälter und der obige Zusammenhang ergibt sich.

Es ist unschwer zu erkennen, dass die Nebenbedingung (1.2) nicht linear ist. Deshalb muss sie, damit sie im Rahmen eines linearen Programms verwendet werden kann, linearisiert werden. Dies geschieht mit Hilfe der "Big M-Methode": Man kann natürlich (1.2) auch schreiben als :

$$(1 - z_k) \sum_{i=1}^{m} u_{\mathrm{ik}} = 0 \qquad (k = 1, \ ..., x)$$

Eine Restriktion der Form:

$$f(x) \ = \ a(x) \ * \ P(x) = c \ \text{ mit } \ a(x) \in \{0, 1\}$$

kann man darstellen als:

1) $c \geq P(x) - (1 - a(x)) * M$

2) $c \leq P(x) + (1 - a(x)) * M$

3) $c \geq -a(x) * M$

4) $c \leq a(x) * M$

wobei $M \geq max\{P(x)\}$

Führt man dieses Vorgehen mit $a(x) = \ 1 - z_k$ und $P(x) = \ \sum_{i=1}^{m} u_{\mathrm{ik}}$ auf unsere Problemstellung aus erhält man folgende Resultate:

Aus 1) ergibt sich :

$$z_k \geq \frac{1}{M} \sum_{i=1}^{m} u_{\mathrm{ik}}$$

Aus 2) ergibt sich :

$$-z_k \leq \frac{1}{M} \sum_{i=1}^{m} u_{\mathrm{ik}}$$

kann man weglassen, da $z_k \in \ \{0, 1\}$ und der rechte Ausdruck ist stets positiv.

Aus 3) und 4) ergibt sich:

$$z_k \leq \ 1$$

kann man ebenfalls weglassen.

Da wir $M \geq max\{P(x)\}$ wählen müssen, bietet es sich in unserer Situation an $M = m$ zu wählen.
Damit ergibt sich die Nebenbedingung (1.2) zu :

$$\frac{1}{m} \sum_{\mathrm{i}=1}^{\mathrm{m}} u_{\mathrm{ik}} \leq z_k \qquad (k \ = \ 1, \ ..., \ x)$$

Das neue Modell (BPP-LP) ergibt sich also zu :

$$min \sum_{k=1}^{x} z_k$$

$$\frac{1}{m} \sum_{i=1}^{m} u_{ik} \leq z_k \qquad (k = 1, \ldots, x)$$

$$\sum_{k=1}^{x} u_{ik} = 1 \qquad (i = 1, \ldots, m)$$

$$\sum_{i=1}^{m} v(a_i)\, u_{ik} \leq \overline{v}(y_k) \qquad (k = 1, \ldots, x)$$

$$\sum_{i=1}^{m} g(a_i)\, u_{ik} \leq \overline{g}(y_k) \qquad (k = 1, \ldots, x)$$

$$u_{ik} \in \{0, 1\}$$
$$z_k \in \{0, 1\}$$

wobei

$$u_{ik} = \begin{cases} 1 & \text{Artikel } i \text{ ist in Behälter } k \\ 0 & \text{sonst} \end{cases}$$

$$z_k = \begin{cases} 1 & \text{Behälter } k \text{ wird mitgenommen} \\ 0 & \text{sonst} \end{cases}$$

Aus einer Lösung des BPP-LP lässt sich über den Entscheidungsvariablen u_{ik} eine Lösung für das BPP rekonstruieren.

3.4 Lösungsmethoden für lineare und ganzzahlige Optimierungsprobleme

In diesem Abschnitt gehen wir näher auf die Lösungsmethoden für lineare und besonders für ganzzahlige Optimierung ein, darunter konzentrieren wir uns hauptsächlich auf das exakte Branch-and-Bound Verfahren, die Metaheuristik Simulated Annealing und einer Greedy-Heuristik. Der Vollständigkeit halber erwähnen wir aber auch noch kurz die Idee des Simplex-Verfahrens, das die Standardmethode für das Lösen von LPs ist. Zuerst klären wir nun die Begriffe des exakten Verfahrens, der Metaheuristik sowie der Greedy Heuristik.

Ein exaktes Verfahren liefert für jeden Input eine beweisbare optimale Lösung eines Optimierungsproblems. Eine Heuristik ist für ein bestimmtes Problem zugeschnitten und begnügt sich lediglich mit einer guten zulässigen Lösung, die im Allgemeinen nicht optimal ist, dafür aber einen geringeren Rechenaufwand hat. Bei Greedy Heuristiken ist noch zu erwähnen, dass jedes Element des Inputs genau einmal betrachtet wird, und die Entscheidung, wie das Element verarbeitet wird, wird niemals rückgängig gemacht. Mit anderen Worten eine getroffene Entscheidung wird für den restlichen Ablauf des Programms als fix betrachtet. Eine Metaheursitk ist eine allgemeine Vorschrift, wie ein nahezu beliebiges Problem gelöst werden kann. Die einzelnen Schritte müssen jedoch für jedes Problem spezifisch angepasst werden. Weiters durchsuchen sie den Lösungsraum nach Nachbarlösungen und zielen darauf ab, sukzessive bessere Lösungen zu finden.
Alle Methoden beziehen sich im Folgenden auf ein Minimierungsproblem.

3.4.1 Standardmethode zum Lösen von linearen Optimierungsproblemen

Das Simplex-Verfahren (kurz Simplex) geht auf den amerikanischen Mathematiker George Bernard Dantzig zurück und löst ein LP exakt oder stellt dessen Unlösbarkeit oder Unbeschränktheit fest.

Betrachten wir zunächst den kontinuierlichen Fall.
Wie wir bereits in Abschnitt 3.2.1 gesehen haben, ist der zulässige Bereich $\{x \in \mathbb{R}^n : Ax \leq b\}$ dieser wird auch *Polyeder* genannt. Weiters benötigen wir im Folgenden kurz den Begriff der Ecke. Eine Menge $M \subset \mathbb{R}^n$ heißt *konvex*, wenn mit je zwei Punkten $x \in M$ und $y \in M$ für jede reelle Zahl λ mit $0 \leq \lambda \leq 1$ auch der Punkt $z = (1 - \lambda)\, x + \lambda y$ zu M gehört. Ist $M \subset \mathbb{R}^n$ konvex, so heißt der Punkt $z \in M$ extremer Punkt (oder auch *Eckpunkt*, *Ecke*) von M, wenn es keine zwei verschiedenen Punkte $x, y \in M$ gibt, so dass gilt: $z = (1 - \lambda)\, x + \lambda y$ für eine reelle Zahl λ mit $0 < \lambda < 1$. Vgl. [Koop&Mook, 2018, S48]

Der allgemeine Simplex setzt immer ein solches Polyeder voraus, von dem er in einer gültigen Ecke aus startet, und in jedem Schritt in eine neue Ecke übergeht, die einen besseren oder zumindest nicht schlechteren Zielfunktionswert aufweist. Beachte: Für den Simplex muss eine gültige Ecke als Startlösung vorhanden sein.

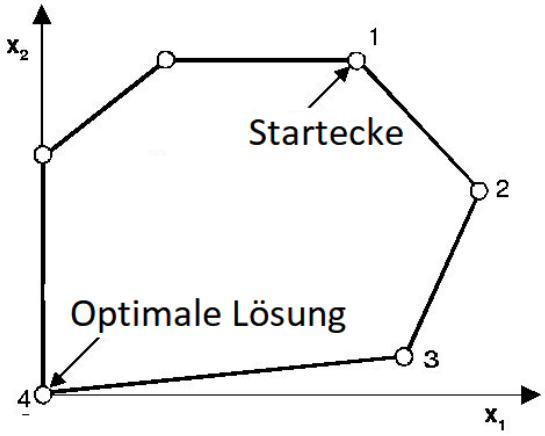


Abbildung 10. Illustration Simplex

Ein Simplex sucht also immer in einer Ecke nach der Lösung. Denn bei linearen Optimierungsproblemen wird die optimale Lösung auch stets am Rand des zulässigen Bereichs angenommen vgl. [Koop&Mook, 2018, S53].

Bei ganzzahligen Optimierungsproblemen, wie zum Beispiel bei unserem Bin-Packing Problem, muss die Lösung allerdings nicht am Rand des zulässigen Bereichs angenommen werden. Damit können wir hier nicht mit dem allgemeinen Simplex arbeiten und sehen uns weiter Lösungsmethoden an.

3.4.2 Branch-and-Bound

Wie wir gleich sehen werden, ist das B&B-Verfahren ein geeignetes Mittel, um ganzzahlige lineare Optimierungsprobleme zu lösen. Aus diesem Grund werden wir hier kurz und knapp das Wichtigste über dieses Verfahren aufzeigen. Schließlich werden wir dieses Verfahren auch verwenden, um unser Bin-Packing Problem zu lösen. Am Ende wird noch der Pseudocode angegeben, damit man die genaue Vorgehensweise dieses Verfahrens nachvollziehen kann.

Dieses Verfahren wird deshalb auch gerne genommen, weil es möglich ist, eine völlige Enumeration aller Lösungen zu vermeiden. Nach Korte et al. ist das B&B-Verfahren für viele kombinatorische Optimierungsprobleme sogar der beste Ansatz, um eine optimale Lösung zu finden! vgl. [Korte&Vygen, 2008, S.596]

Das Verfahren besteht im Wesentlichen aus zwei Schritten dem Verzweigen (Branch) und dem Beschränken (Bound).
Im Branch-Schritt versucht man, das Problem in zwei Teilprobleme zu zerteilen, die leichter sein sollen als das Originalproblem. Dabei lässt man meist die Ganzzahligkeitsbedingung weg und löst dann das Problem ohne diese. Dabei erhält man dann eine Lösung, die i.A. nicht ganzzahlig ist. Ist die Lösung ganzzahlig, so hat man eine Lösung gefunden und ist fertig. Im anderen Fall gibt es mindestens eine Komponente, die nicht ganzzahlig ist. Danach rundet man die Komponente, die nicht ganzzahlig ist einmal auf die nächst größere ganze Zahl auf und einmal auf die nächst kleinere ganze Zahl ab. Dadurch hat man dann zwei Teilprobleme. Durch rekursives Ausführen dieses Schrittes erhält man schließlich eine Baumstruktur und die Lösung des Problems entspricht der Suche im gesamten Baum. Wie der nächste zu bearbeitende Knoten im Baum ausgewählt wird, wird nach einer Regel

bestimmt. Wir verwenden die FIFO-Regel (First-in-First-Out). Diese besagt: Wähle jenes Teilproblem als Nächstes aus, welches als Erstes in den Baum eingefügt wurde. Mit anderen Worten könnte man die FIFO-Regel auch als Breitensuche beschreiben.
Eine weitere Regel wäre z.B. die LIFO-Regel (Last-In-First-Out). In diesem Fall sucht man den Baum zuerst in der Tiefe nach Lösungen ab. Dabei findet man schnell eine ganzzahlige Lösung, die möglicher weise aber einen schlechten Zielfunktionswert hat.

Im Bound-Schritt wird versucht, auf Basis von Schranken frühzeitig Teile des Baumes als "schlecht" oder "unbrauchbar" zu klassifizieren und dadurch unnötige Berechnungen einzusparen.

Eine obere Schranke lässt sich leicht finden, denn jede zulässige Lösung ist zu Beginn eine obere Schranke. Ist keine Lösung bekannt, so kann auch ∞ als obere Schranke dienen.

Die untere Schranke entspricht dem Weg von der Wurzel des Baumes bis zum aktuellen Teilproblem. Ist die untere Schranke größer als die obere, wird dieser Teil des Baumes nicht weiter betrachtet. Ist die Schranke hingegen kleiner als die obere, so wird weiter verzweigt. Ist die untere Schranke eine zulässige Lösung und besser als die aktuelle obere Schranke, so wird die untere die neue obere Schranke. Es sei an dieser Stelle bemerkt, dass für die untere Schranke fast immer eine Relaxation zu Grunde gelegt wird.

■ LP-Relaxation

Hier wird zur Abschwächung bei den ganzzahligen LPs meist die Ganzzahligkeitsbedingung $x_i \in \mathbb{Z}$ weggelassen. Bspw. wird $x_i \in \mathbb{N}_0$ durch $x_i > 0$ und $x_i \in \{0, 1\}$ durch $0 \leq x_i \leq 1$ ersetzt. Damit erhält man wieder ein normales LP, welches man kontinuierlich lösen kann. Diese erhaltene Lösung könnte man dann durch Runden auf die nächste ganze Zahl als eine Näherungslösung für das ganzzahlige LP ansehen. Jedoch ist hier Vorsicht geboten, denn die gerundete Lösung kann unter Umständen gar nicht in der zulässigen Lösungsmenge enthalten sein. Weiters hat Koop [Koop&Mook, 2018, S.168] ein Beispiel gezeigt, wo die gerundete Lösung weit von der ganzzahligen Lösung entfernt ist.

Nachstehend wird ein allgemeiner Pseudocode für das B&B-Verfahren angegeben.

Algorithm 1. General B&B method

Input: An instance of a minimization problem.
Output: An optimal solution *bestvalue*

1. *activeset* := {root};
2. *bestvalue* := ∞; (upperbound)
3. *currentbest* := NULL;
4. while *activeset* ≠ ∅ do
5. **choose a branching node, node $k \in$ *activeset*;**
6. ***activeset* = *activeset*\{*k*};**
7. **generate the children of node *k*, *child i*, *i* = 1 to 2,**
and corresponding lower bounds ob_i;
8. **for *i* = 1 to 2 do**
9. **if ob_i worse than *bestvalue* then kill *child i*;**
10. **else if child *i* is a complete solution then**
11. ***bestvalue* := value at child *i*, *currentbest* := *child i*;**
12. **else *activeset* = *activeset* $\cup$ {*i*}**
13. **end for**
14. end while

Algorithmus 1: Allgemeiner Pseudocode B&B

Bemerkung:

activeset ist die Menge, die noch genauer untersucht werden muss.
currentbest speichert die bisher beste Lösung.
bestvalue gibt an, ob es sich lohnt, weiter zu expandieren oder nicht. Also, ob man den "Ast" im Baum noch weiter verfolgt oder auf Grund einer Schranke es sich nicht mehr lohnt, weiter nach unten zu rechnen, da ohnehin nur schlechtere Lösungen auftauchen würden.

Das hier beschriebene B&B-Verfahren findet immer eine optimale Lösung. Weiters hängt die Laufzeit sehr stark von den Schranken ab. Im worst case wird das Verfahren zu einer vollständigen Enumeration und hat somit eine Laufzeit von $O(2^n)$ wobei n die Größe des Inputs wäre. (In unserem Fall wären das die Anzahl der Variablen u_{ik} zuzüglich den z_k) Somit gilt:

$$t_{BB}(n) \in O(2^n)$$

Für das bessere Verständnis wird das hier präsentierte Verfahren anhand eines einfachen Beispiels demonstriert.

Beispiel:

$$\text{Minimiere } \phi_{P_0}(x_1, x_2) = 2x_1 + x_2$$

unter den Nebenbedingungen

$$\begin{aligned} 4x_1 + 4x_2 &\geq 10 && (*) \\ 2x_1 + 11x_2 &\geq 11 && (**) \\ -4x_1 + 2x_2 &\leq 1 && (***) \\ x_1, x_2 &\geq 0, \text{ ganzzahlig} \end{aligned}$$

Da eine genaue obere Schranke nicht gegeben werden kann, wird diese in unserem Fall als $\overline{P_0} = \infty$ festgesetzt. Die obere Schranke ist der optimale Zielfunktionswert des angepassten Problems. Das heißt, dass wir das Ausgangsproblem zunächst ohne die Ganzzahligkeitsbedingung lösen werden. (Dies kann man einerseits grafisch oder andererseits mit dem Simplex lösen.)
Wie man nachrechnen kann, ist die optimale Lösung unseres Problems P_0 ohne die Ganzzahligkeitsbedingung $(x_1, x_2) = (0.67,\ 1.83) \notin \mathbb{Z}^2$.

$$\phi_{P_0}(0.67,\ 1.83) = 2 * 0.67 + 1.83 = 3.17.$$

Dies liefert uns eine obere Schranke $\overline{P_0} = 4$.
Nun subtrahieren wir die nächstgrößeren ganzen Zahlen unserer Lösung und, die mit der kleineren Differenz, ist unsere erste Verzweigung.

$$\begin{aligned} 1 - 0.67 &= 0.33 \\ 2 - 1.83 &= 0.17 \end{aligned}$$

Damit beginnen wir die Verzweigung mit der Variablen x_2. Nun nehmen wir die nächst größere und die nächst kleinere Zahl von x_2. Damit gibt es nun 2 Möglichkeiten für x_2 :

Fall 1: $x_2^r \geq 2$
Fall 2: $x_2^l \leq 1$

Der “rechte Ast” im Baum ist für den Fall 1 der “linke Ast” für den Fall 2 (Bounding). Nach der FIFO Regel fangen wir nun mit dem rechten Ast an.

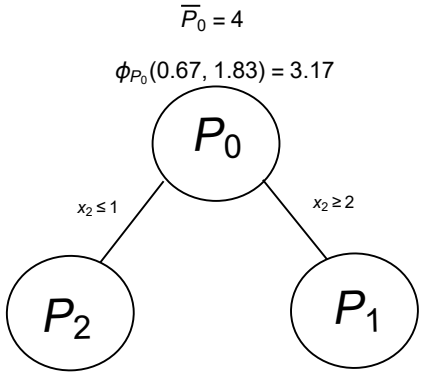


Abbildung 11. Starte Verzweigung

Es gilt nun die zusätzliche Restriktion $x_2 \geq 2$. Grafisch verschiebt sich nun der zulässige Bereich nach oben. Eine optimale Lösung ist gegeben durch $(x_1, x_2) = (0.75,\ 2)$ mit dem Zielfunktionswert $\phi_{P_1} = 3.5$.
Diese Lösung ist nicht ganzzahlig und deshalb auch nicht zulässig. Somit muss weiter verzweigt werden. Nun machen wir dasselbe für x_1 wie zuvor für x_2.

Fall 1 : $x_1^l \leq 0$
Fall 2 : $x_1^r \geq 1$

Es gilt nun die zusätzliche Restriktion $x_1 \geq 1$. Es gilt aber immer noch auch $x_2 \geq 2$.
Nun berechnen wir für $x_1 \geq 1$ eine Lösung.

Eine optimale Lösung ist gegeben durch: $(x_1, x_2) = (1,\ 2)$ mit dem Zielfunktionswert $\phi_{P_3} = 4$.
Da es sich hierbei um eine ganzzahlige Lösung handelt, ist diese zulässig für das Ausgangsproblem. Wie wir aus dem vorherigen Abschnitt wissen, muss unsere gerundete Schranke $\overline{P_0} = 4$, die von der optimalen Lösung aufgerundet wurde, nicht in der zulässigen Lösungsmenge enthalten sein. Da wir aber nun eine zulässige Lösung mit diesem Zielfunktionswert gefunden haben, ist $\overline{P_0} = \overline{P_3} = 4$ nun auch die optimale Lösung. Demnach muss hier nicht weiter verzweigt werden.

Der Vollständigkeithalber zeigen wir auch noch die Verzweigung für P_4.
Nun gilt die zusätzliche Restriktion $x_1 \leq 0$. Es gilt aber immer noch auch $x_2 \geq 2$.

Durch Bedingung $x_1 \leq 0$ müsste der optimale Punkt auf der $y-$Achse liegen. Dies ist aber auf Grund der Restriktion $(***)$ nicht möglich und demnach existiert keine zulässige Lösung. Dieser Zweig müsste nun auch nicht mehr weiter betrachtet werden. Nachstehend der Baum, der sich durch die Verzweigungen gebildet hat.

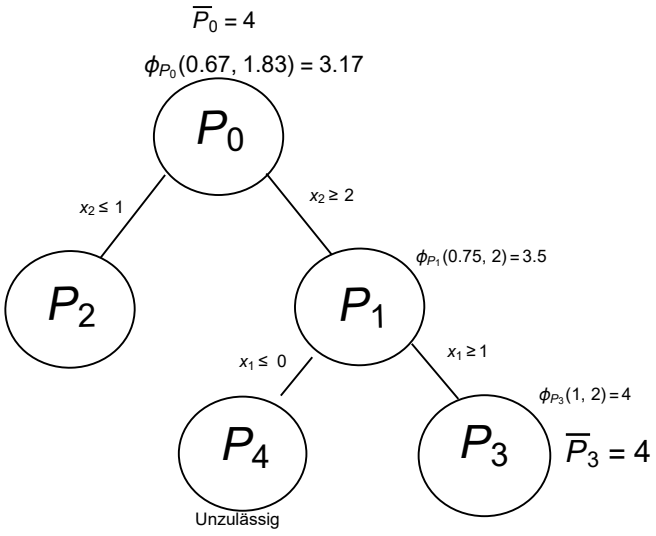


Abbildung 12. Weg durch den Baum zu $\overline{P}_3$

Implementierungsnotiz

Wir haben mit dem zuletzt gezeigten Verfahren das Bin-Packing Problem exakt gelöst. (Natürlich nur für kleine Instanzen.) Dazu haben wir die Optimierungsumgebung “Gurobi” verwendet. Diese wurde lizensiert und in C# eingebettet, danach haben wir auf Basis der Dokumentation von Gurobi das eigens für diese Problemstellung entwickelte mathematische Modell aus Kapitel 3.3 implementiert. Der Algorithmus bekommt also als Input die Liste der Artikel und die Parameter für die Behälter ($\overline{v}$, $\overline{g}$) und liefert als Output die gefüllten Behälter. Zur Verwendung von Gurobi stelle man sich eine Klasse, wie sie aus C++ oder C# bekannt ist, vor. Gurobi besteht im Wesentlichen aus vielen verschiedenen --zum Teil-- verschachtelten Klassen, die alle für die Behandlung von Optimierungsproblemen entwickelt worden sind. Die Verwendung von Gurobi unterscheidet sich nicht stark von der Verwendung anderer Klassen. Eine Dokumentation zur Verwendung findet sich hier: http://www.gurobi.com/documentation/

Bemerkung:

Es wurde C# gewählt, da dies dem Unternehmensstandard des Auftraggebers entspricht.

3.4.3 Simulated Annealing

Die Basis für dieses Kapitel bilden die beiden Bücher: [Aarts&Korst, 1989] und [Hromkovič, 1998]. In diesem Abschnitt wollen wir die wichtigsten Parameter und Eigenschaften des Simulated Annealing besprechen.

Simulated Annealing (SA) ist ein metaheuristisches Verfahren zum Auffinden von Näherungslösungen. Die Idee entstammt aus der Physik, etwa beim Glühen von Metallen. Nach dem Schmelzen des Metalls sorgt die langsame Abkühlung dafür, dass die Atome genügend Zeit haben, sich neu zu ordnen

und dabei stabile Kristalle bilden können. Dies hat einen energiearmen Zustand nahe dem Optimum zur Folge, was bedeutet, dass am Ende wieder ein stabiles Metall erreicht wird. Es bestehen folgende Analogien zu den folgenden Optimierungsproblemen.

- Zulässige Lösungen eines kombinatorischen Optimierungsproblems entsprechen den Zuständen im physikalischem System.
- Die Kosten einer Lösung entsprechen der Energie des Zustandes.

Eine große Besonderheit des SA ist es, dass dieses Verfahren in Abhängigkeit einer Akzeptanzwahrscheinlichkeit auch schlechtere Lösungen zulässt, damit es nicht in einem lokalen Optimum hängen bleibt. (Siehe Abbildung 13).

Definition 10

Seien x, y zwei zulässige Lösungen eines kombinatorischen Optimierungsproblems und $f(x)$, $f(y)$ die Kosten der jeweiligen Lösung. Das Akzeptanzkriterium berechnet nun, ob y gegenüber x akzeptiert wird durch die folgende *Akzeptanzwahrscheinlichkeit*:

$$P_{T,x,f}(y) = \begin{cases} 1 & f(y) \leq f(x) \\ e^{\frac{f(x)-f(y)}{T}} & f(y) > f(x) \end{cases}$$

wobei $T \in \mathbb{R}^+$ den Kontrollparameter (control parameter) bezeichnet. T wird auch noch Temperaturparameter genannt. Als *Akzeptanzkriterium* verwenden wir $z \leq P_{T,x,f}(y)$ mit einer auf [0, 1]-gleichverteilten Zufallszahl z .

Bemerkung:

Für eine gleichverteilte Zufallsvariable $X = G[0, 1]$ ist die Wahrscheinlichkeit, dass diese eine Zahl $x \in [0, p]$ annimmt genau p. Mit anderen Worten es gilt :

$$P(\{X \leq p\}) = p$$

Diese Eigenschaft werden wir später im Algorithmus ausnützen.

Es wird sich zeigen, dass dieser Kontrollparameter T eine monoton fallende Folge ist, die wir im Folgenden mit T_t bezeichnen. Dabei gilt es zu beachten, dass bei fallendem T_t die Wahrscheinlichkeit, eine schlechtere Lösung zu akzeptieren, sinkt. Weiters werden für $T_t \to 0$ gar keine Verschlechterungen mehr akzeptiert. Außerdem wird gegen Ende des SA x wahrscheinlich eine sehr gute Lösung sein, was die Differenz $f(x) - f(y)$ für eine schlechtere Lösung y wahrscheinlich groß macht. Weiters wird der Kontrollparameter T_t stets kleiner. Insgesamt wird also die Akzeptanzwahrscheinlichkeit stets kleiner, je länger die Prozedur läuft. Das bedeutet, dass die Wahrscheinlichkeit, eine schlechtere Lösung zu akzeptieren, gegen Ende der Prozedur extrem unwahrscheinlich wird und sich dieses Verfahren daher wie ein Gradientenverfahren verhält.

Ferner kann unter bestimmten Voraussetzungen sogar gezeigt werden, dass dieses Verfahren gegen die optimale Lösung konvergiert.[Aarts&Korst, 1989, S.18f] Leider gilt dieses Resultat erst im Grenzwert $n \to \infty$ wobei n die Anzahl der Iterationen ist. Außerdem erlaubt uns dieses Verfahren auch nicht festzustellen, ob das globale Optimum bereits erreicht wurde oder wie weit wir vom globalen Optimum entfernt sind. Deshalb muss ein Kompromiss bezüglich der Laufzeit und der Qualität der Lösung vereinbart werden.

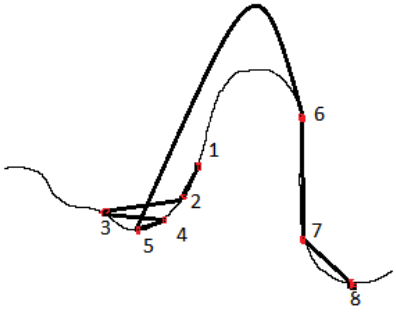


Abbildung 13. Diese Abbildung illustriert den Fall, dass der Algorithmus ein lokales Minimum gefunden hat aber dennoch über den "Berg" zu einem besseren Minimum findet, indem er den Bereich um den neuen Punkt untersucht.

Das Verhalten des SA wird maßgeblich durch die Anfangstemperatur T_0 durch die Dekrementierungsfunktion das ist jene Funktion, die die T_t im Laufe des Algorithmus verkleinert, und durch das Stopp-Kriterium (stop criterion) beeinflusst. Diese Parameter zusammen bezeichnet man in der Literatur oft als cooling schedule. Für all diese Parameter gibt es keine allgemein gültige Wahl, vielmehr muss für jede Problemstellung eine passende gefunden werden, was in den meisten Fällen empirisch erfolgt.

Der Startwert T_0

T_0 sollte so gewählt werden, dass zu Beginn beinahe alle Lösungen akzeptiert werden, um dadurch mögliche Muster in der Endlösung zu vermeiden. Grundsätzlich gibt es aber viele verschiedene Ansätze, wie T_0 gewählt werden kann. Ferner sei bemerkt, dass man grundsätzlich völlig frei in der Wahl von T_0 ist.

Dekrementieren des Kontrollparameters T_t

Auch hier gibt es eine Vielzahl an Möglichkeiten, dies zu erreichen. Eine, die am häufigsten genutzt wird, ist die geometrische:

$$T_{t+1} = \alpha\, T_t\,,\ t = 1, 2,\ ...,$$

wobei α eine Konstante kleiner (jedoch nahe) bei eins ist. Typischerweise ist $\alpha \in [0.8, 0.99]$.

Weitere mögliche Vorgehensweisen sind z.B.:

- **Nichtmonoton**

  Hier wird die Temperatur nicht immer abgesenkt, es sind Szenarien möglich, wo diese wieder erhöht wird. [Hajek et al., 1989]

- **Sehr geringe Abkühlung**

  Diese haben die Form $T_{t+1} = \frac{T_t}{1+T_t\, a}$, wobei a sehr klein ist. Dies ist ein Beispiel für eine Funktion, wo man nach jedem Durchgang die Temperatur verringert.

- **Adaptiv**

  Hier wird die Senkung der Temperatur vom Verhalten des Algorithmus abhängig gemacht.

Endwert des Kontrollparameters

Hier spielt das Stopp-Kriterium eine wichtige Rolle. Einerseits soll dieser Wert klein genug sein, um gute Lösungen zu erhalten, andererseits auch nicht zu klein, um lange Rechenzeiten zu vermeiden. Auch hier wird meist empirisch gehandelt. Deshalb wird das Stopp-Kriterium nicht nur vom Kontrollpa rameter abhängig gemacht, sondern auch noch von der Veränderung der Lösung. Das bedeutet, sollte nach einer bestimmten Anzahl an Durchläufen sich nichts an x geändert haben, brich ebenfalls ab.

Gleichgewichtsbedingung

Die Gleichgewichtsbedingung (Equilibrium condition) legt fest, wie lange der Algorithmus mit der derzeitigen Temperatur nach Nachbarlösungen sucht. Diese sollte so ausgelegt sein, dass sie zu Beginn weniger Durchläufe gestattet und gegen Ende länger erfüllt bleibt. Schließlich wird am Anfang lediglich zwischen zufälligen Lösungen hin und her gesprungen. Mit der Zeit sollte der Lösungsraum jedoch genauer durchforstet werden, um eine gute Lösung zu finden.

Nachstehend wird nun ein allgemeiner Pseudocode des SA gegeben. Im Grunde besteht das Verfahren nur aus zwei ineinander verschachtelten Schleifen.

Algorithm 2. Simulated Annealing

Input: Instance of a problem
Output: Final state *x*

```
1. Generate startsolution x of the input instance.
2. Initialize control parameter T
3. do
4.         do
5.                 Generate a neighbour solution y
6.                 if random(0,1) < P(T,x,y,f) then
7.                         x = y
8.                 else
9.                         reject
10.        while (Equilibrium)
11.        calculate new control parameter T
12. while (stop criterion)
```

Algorithmus 2. Simulated Annealing

Implementierungsnotiz

Beim Simulated Annealing gibt es sehr viele Parameter und viele verschiedene Möglichkeiten, bestimmte Schranken oder Kriterien festzulegen. Dabei gibt es auch keine allgemeine Vorgehensweise, wie die Parameter oder Kriterien zu wählen sind. Vielmehr muss dies für jedes Problem speziell festgelegt werden.

Bemerkung:

In der präzisierten Beschreibung haben wir eine Zuordnungsfunktion $f:\{1,\ ...,\ m\} \to \{1,\ ...,\ x\}$ gesucht. Diese wird in der Programmierung dargestellt als eine Liste $(i \mid f(i) = j)_{j \in \{1,...,x\}}$

Startlösung

Grundsätzlich kann als Startlösung x jede Lösung genommen werden. Man könnte auch direkt die unveränderte Inputliste als Startlösung wählen. Empirisch haben wir aber festgestellt, dass wir mit einer qualitativ guten Startlösung schneller zu einer qualitativ guten Endlösung gelangen. Deshalb haben wir die Startlösung x mit unserem Best-Fit erzeugt.

Kontrollparameter

Den Kontrollparameter T_t haben wir zu Anfang auf 1 gesetzt. Also $T_0 = 1$, das bedeutet volle "Aufheizung", damit werden in den ersten Schleifendurchgängen einige schlechtere Lösungen akzeptiert.

Generierung einer Nachbarlösung

$U(x)$ ist die Menge von Nachbarlösungen zu $x = (a_1,\ ...,\ a_n)\ n \in \mathbb{N}$, wobei wir x mit der Heuristik Best-Fit erzeugen.(Siehe Abschnitt 3.4.4.2) Eine Nachbarlösung $y \in U(x)$ wird ebenfalls durch die Heuristik Best-Fit erzeugt. Nun ist der Input für die Heuristik die vorherige Lösung x, wobei wir zuvor noch beliebig viele Komponenten aus x vertauscht haben. Beispielsweise wird die i-te Komponente des Lösungsvektors x mit der j-ten Komponente vertauscht. Das bedeutet, in der Lösungsliste steht an der Stelle a_i nun a_j. Welche Komponente ausgewählt wird, wird zufällig bestimmt. Zu Beginn ist die Anzahl der paarweisen Vertauschung von Komponenten aus x noch recht hoch. Mit fallendem T_t werden auch die Vertauschungen weniger. Diese berechnen sich wie folgt:

$$Swaps\ = \lfloor T_t * |S_i| * 4.5 \rfloor$$

$|S_i|$ ist die Anzahl der Elemente in der jeweiligen Warengruppe. Diese wird dadurch bestimmt, für welche Artikelgruppe der Algorithmus gerade ausgeführt wird und wird pro kompletten Durchlauf natürlich nicht verändert.
Die zusätzliche Konstante 4.5 dient dazu, dass am Ende noch einige wenige Vertauschungen möglich sind.

Kostenfunktion und Akzeptanzwahrscheinlichkeit

Die Kostenfunktion f misst einfach die Anzahl der benötigten Behälter. Je mehr Behälter, desto teurer die Lösung. Als Akzeptanzwahrscheinlichkeit haben wir jene aus Definition 10 verwendet. Diese ist eine in der Literatur häufig verwendete, da diese die gewünschte Eigenschaft hat, am Beginn viele Lösungen zuzulassen und sich am Ende in ein Gradientenverfahren verwandelt.

Gleichgewichtsbedingung K_t

Wie wir wissen, legt diese Bedingung fest, wie lange wir in der aktuellen "Temperaturzone" verharren. Wir haben eigentlich statt einer Bedingung eine Schranke gewählt, die die gewünschte Eigenschaft hat. Da aber in der Literatur ständig von einer Bedingung gesprochen wird, haben wir dies der Vollständigkeit halber übernommen. Die Schranke sieht wie folgt aus:

$$K_0 := 5$$

$$K_t = K_{t-1} + 3$$

Berechnung des Kontrollparameters

Wir haben die geometrische Folge gewählt, wie wir es unter "Dekrementieren des Kontrollparameters" gezeigt haben. Also:

$$T_0 := 1$$

$$T_{k+1} = \alpha\, T_k \text{ mit } \alpha = 0.83$$

Stopp Kriterium

Dieses Kriterium haben wir von zwei Werten abhängig gemacht:

- $T_k < 0.001$
- x wurde in 1000 Durchläufen nicht verändert

Damit vermeiden wir ebenfalls unnötige Rechenzeiten.

3.4.4 Greedy Heuristiken

In diesem Kapitel geben wir die wichtigsten Grundlagen zu Heuristiken für das Bin-Packing Problem und stellen die für die Anwendung relevante Best-Fit Decreasing Heuristik vor. Die Basis bildet die Literatur von: [Korte&Vygen, 2008], [Hüftle , 2006]

3.4.4.1 Theoretische Grundlagen zu Greedy Heuristiken für das Bin-Packing Problem

Die Grundidee ist es, durch das Wiederholen einer simplen Prozedur schrittweise Artikel für Artikel eine Lösung aufzubauen. Dabei wird in jedem einzelnen Schritt eine neue Lösungskomponente einer bisher gefundenen Teillösung hinzugefügt. Eine Teillösung entspricht beim Bin-Packing einer Abbildung:

$$f_t : \{1, \ldots, t\} \to \{1, \ldots, x\}$$

Lösungen werden iterativ beginnend mit $f_0 : \emptyset \to \{1, \ldots x\}$ derart erzeugt, dass für jedes $t \in \mathbb{N}$ gilt:

$$f_t(i) = f_{t-1}(i) \qquad \text{für alle } 1 \leq i < t.$$

Das Hinzufügen einer neuer Lösungskomponente in f_t ist dann $f_t(t)$.
Verschiedene Heuristiken unterscheiden sich darin, wie die Lösungskomponenten ausgewählt werden. Die Wahl der nächsten Lösungskomponente wird aber immer nur auf Basis des jeweiligen betrachteten Artikel ausgewählt, d.h. zukünftige Artikel beeinflussen die aktuelle Wahl nicht. Ferner werden getroffene Wahlen nicht wieder rückgängig gemacht. Weiters soll f_t möglichst wenige Behälter verwenden, was bedeutet, f_t soll den Artikel t, wenn möglich, in einen schon benutzten Behälter packen.

Für jede Instanz I: $A(I)$ ist diejenige zulässige Lösung von Problem "Bin-Packing" die Algorithmus A bei Input I berechnet. OPT(I) sei die Anzahl der Behälter, die ein optimaler Algorithmus für die Instanz I benötigt hätte.
Da wir hier keine exakte Lösung mehr suchen, wollen wir Lösungen qualitativ beurteilen. Der Idealfall für Algorithmus A wäre, dass für jede Instanz I das Resultat garantiert nur um einem konstanten Betrag vom Optimum abweicht. Also etwas der Form:

$$A(I) - \text{OPT}(I) \leq c$$

Einen solchen Algorithmus nennt man in der Literatur oft absoluter Approximationsalgorithmus. Dies ist in der Praxis meist zu restriktiv und wir müssen uns mit sogenannten relativen und asymptotischen Approximationsalgorithmen begnügen.

Definition 11
Sei P ein Optimierungsproblem und $k \geq 1$. Ein asymptotischer *k-Approximationsalgorithmus* für P ist ein polynomieller Algorithmus A für P, sodass für alle Instanzen I von P gilt:

$$A(I) \leq k\, OPT(I) + c \qquad \text{für ein fixes } c \in \mathbb{R}_0^+$$

Kann $c = 0$ gewählt werden, so nennen wir den Algorithmus relativer *k-Approximationsalgorithmus*.
Wir sagen auch: A hat die (relative/asymptotische) *Approximationsgüte k*.
[vgl. Korte&Vygen, 2008, S.445 / S.423]

Bemerkung:

Da wir uns stets mit einem Minimierungsproblem beschäftigen, wurde in der Definition 11 nur die relevante Ungleichung gezeigt. Der Vollständigkeit halber wird aber hier auch die Ungleichung für ein Maximierungsproblem gegeben:

$$\frac{1}{k} OPT(I) - c \leq A(I)$$

Satz 7
Für den Fall $\mathcal{P} \neq \mathcal{NP}$ gilt: Es gibt keinen relativen k-Approximationsalgorithmus für BIN-PACKING mit $k < \frac{3}{2}$.

Beweis:

Nehmen wir an, dass ein polynomialer Approximationsalgorithmus A für jede lösbare Instanz unser Bin-Packing Problem existiert mit $k < \frac{3}{2}$. Nun können wir für das $\mathcal{NP}$-vollständige Problem *Partition* einen polynomialen Algorithmus entwickeln. Dies können wir mit Hilfe unserer Reduktion bewerkstelligen. Also $Partition \leq_p BinPacking$. Nun wenden wir den Approximationsalgorithmus A auf die transformierte Eingabe an. Berechnet nun A eine Lösung mit 2 Behältern, so hat *Partition* eine Lösung mit $k = 1$ damit würde gelten $\mathcal{P} = \mathcal{NP}$. Wenn jedoch A eine Lösung mit mindestens 3 Behältern berechnet, benötigt eine optimale Lösung, da $k < \frac{3}{2}$ ist, mehr als 2 Behälter und folglich hat *Partition* gar keine Lösung im Widerspruch zur Lösbarkeit.

■

Bemerkung:

Der gerade gesehene Beweis nutzt aus, dass das Problem Partition zu unserem Bin-Packing Problem mit zwei oder drei Behältern verwandt ist.

3.4.4.2 Best-Fit

Im Nachfolgenden zeigen wir die Heuristik Best-Fit (BF), auf die wir im nächsten Abschnitt weiter aufbauen. Die allgemeine Prozedur kann wie folgt beschrieben werden.

Prozedur Best-Fit:

Sei $x \geq 1$ die Anzahl der gerade offenen Behälter.
Die Heuristik fügt das gerade behandelte Objekt a_i in jenen Behälter y_j mit $1 \leq j \leq x$, der nach der Befüllung mit a_i den höchsten Füllgrad aufweist. Besitzt kein y_j mehr ausreichend Platz, so wird ein neuer Behälter geöffnet und a_i darin verstaut. Ist in einem y_j der maximale Füllgrad erreicht, schließe den Behälter y_j.

Bemerkung:

- Da der maximale Füllgrad nahezu nie erreicht wird, bleiben meist alle Behälter bis zum Ende der Prozedur geöffnet, weshalb das Schließen der Behälter oft gar nicht implementiert wird.
- Der Einfachheit halber haben hier und in den nachfolgenden Pseudocodes das zulässige Beladegewicht und das Volumen eines jeden Behälters verändert und diese dann als Restkapazitäten verwendet. Grundsätzlich sollte man diese Werte nicht verändern. Man sollte lieber separate Variablen dafür einführen und mit diesen arbeiten.

Algorithm 3. Best−Fit

Input: Same as in problem Bin−Packing
Output: A solution (x, f)

1. $x = 1$
2. for all objects $i = 1, \ldots, m$ **do**
3. $j_{min} = 0$
4. for all bins $j = 1, \ldots, x$ **do**
5. if $g(a_i) \le \bar{g}(y_j)$ **and** $v(a_i) \le \bar{v}(y_j)$ **then**
6. if $j_{min} = 0$ **or** $(\bar{g}(y_j) - g(a_i) \le \bar{g}_{min}$ **and** $\bar{v}(y_j) - v(a_i) \le \bar{v}_{min})$
7. $j_{min} = j$
8. $\bar{g}_{min} = \bar{g}(y_j) - g(a_i)$, $\bar{v}_{min} = \bar{v}(y_j) - v(a_i)$
9. end if
10. end if
11. end for
12. if $j_{min} = 0$ **then**
13. $x = x + 1$
14. $j_{min} = x$
15. end if
16. $\bar{g}(y_{j_{min}}) = \bar{g}(y_{j_{min}}) - g(a_i)$, $\bar{v}(y_{j_{min}}) = \bar{v}(y_{j_{min}}) - v(a_i)$
17. $f(i) = j_{min}$
18. end for

Algorithmus 3. Best-Fit

Laufzeitanalyse

Für die Bestimmung der Laufzeit haben wir im Abschnitt 3.1.1 bereits gesehen, dass wir hier die Anzahl der elementaren Schritte zählen. Wir beschränken uns dabei aber nur auf die dominanten Terme und lassen konstante Faktoren außer Acht.
Die einfachen Initialisierungen im Schritt 1 und 3 können wir unberücksichtigt lassen. Die Schleife in Schritt 2 ist direkt vom Input abhängig und hat eine Laufzeit von m. Die darin enthaltene Schleife in Schritt 4 hat eine worst case Laufzeit von $x = i$, da wir im schlimmsten Fall jeden Artikel in einen neuen Behälter packen müssen. Danach kommen zwei Vergleiche, die wir zählen müssen und in Schritt 6 sind zwei Subtraktionen und weitere zwei Vergleiche, die wir ebenfalls berücksichtigen. Insgesamt bisher 6 zu zählende Schritte. Schritt 7 ist wieder eine einfache Zuweisung die wir vernachlässigen. Die Subtraktionen in Schritt 8 können wir auch vernachlässigen, wenn wir sie aus Schritt 6 zwischenspeichern. Somit entspricht auch Schritt 8 einer einfachen Zuweisung. Die restlichen Schritte außer Schritt 16 sind auch einfache Zuweisungen oder Inkrementierungen, die wir nicht Zählen müssen. Lediglich die Endberechnung in Schritt 16, wo wir nun die tatsächlichen Restkapazitäten berechnen, muss noch beachtet werden. Damit gilt nun insgesamt:

$$t_{BF}(m) = \sum_{i=0}^{m} (6\,i + 2) = 3\,m^2 + 5\,m + 2$$

Da wir uns nur auf die dominanten Terme konzentrieren folgt:

$$t_{BF}(m) \in O(m^2)$$

3.4.4.3 Best-Fit Decreasing

Dies ist eine modifizierte Variante des zuvor gezeigten BF-Algorithmus. Der Best-Fit Decreasing Algorithmus sortiert die Inputliste zuvor noch absteigend bezüglich dem Gewicht, was einen erheblichen qualitativen Einfluss auf die Lösung hat. Dies kann man sich auch einfach überlegen, denn große Elemente am Schluss verursachen meist Probleme bei der Packung, denn sie benötigen meist eigene Behälter vor allem, wenn es mehrere Objekte sind, die mehr als die Hälfte des Platzes benötigen. Damit würde dann am Ende jeder Artikel in einen eigenen Behälter verpackt werden müssen und diese Behälter wären dann nur mit etwas mehr als 50% gefüllt. Natürlich kann auch weiterhin eine optimale Lösung durch die Sortierung nicht garantiert werden.

Der Vollständigkeit halber wird auch hier der Pseudocode angegeben:

Algorithm 4. Best−Fit Decreasing

Input: Same as in problem Bin−Packing
Output: A solution (*x*, *f*)

1. Sort objects in decreasing order using QuickSort.
2. Apply Best−Fit to the sorted list of objects.

Algorithmus 4. Best-Fit Decreasing

Laufzeitanalyse

Im ersten Schritt benötigen wir für den Quicksort eine Laufzeit von $O(m^2)$. Im zweiten Schritt wird der BF ausgeführt, der $O(m^2)$ benötigt. In Summe benötigen wir also:

$$O(m^2 + m^2) = O(2\,m^2) = O(m^2)$$

Der Best-Fit Decreasing hat also die asymptotisch gleiche Laufzeit wie auch der BF. Also gilt:

$$t_{BFD}(m) \in O(m^2)$$

Bemerkung:

In dem Artikel von [Varsamis, 2017] ist ein Ansatz zu finden, wie man diese Methode parallelisieren kann.

Im nachfolgenden Beispiel zeigen wir, dass der BFD nicht immer das Optimum erreicht. Um das Beispiel nicht unnötig aufzublähen, betrachten wir in den beiden nachfolgenden Beispielen für jeden Behälter nur eine Restkapazität c, anstatt immer das noch restliche Volumen und noch verfügbare Beladegewicht zu berechnen.

Beispiel:

Die Kapazität eines jeden Behälters sei auf 1 gesetzt. Der Einfachheit halber bezeichnen wir das $i-$te Element in der Instanz I_s mit e_i und die Restkapazität des Behälters y_i mit c_i, wobei $1 \leq i \leq z$ und z die Länge der Instanz ist.

Gegebene Instanz : $I = (0.2, 0.5, 0.6, 0.2, 0.3, 0.2, 0.6, 0.4)$

Der BFD sortiert die Instanz bevor die Verpackung beginnt. Damit erhalten wir :

$$I_s = (0.6, 0.6, 0.5, 0.4, 0.3, 0.2, 0.2, 0.2)$$

Nun geht der BFD wie der BF vor. Da die ersten 3 Elemente nicht in einen Behälter passen, öffnet er für jedes einen. Damit sehen wir:

$y_1 = [e_1 \mid \ ...]$ mit $c_1 = 0.4$
$y_2 = [e_2 \mid ...]$ mit $c_2 = 0.4$
$y_3 = [e_3 \mid ...]$ mit $c_3 = 0.5$

Nun wird e_4 betrachtet, dies passt am besten in y_1 sowie in y_2 deshalb wird der BFD dies in y_1 packen, da dieser als Erstes betrachtet wird. Damit erhalten wir:

$y_1 = [e_1 \mid e_4]$ mit $c_1 = 0$
$y_2 = [e_2 \mid ...]$ mit $c_2 = 0.4$
$y_3 = [e_3 \mid ...]$ mit $c_3 = 0.5$

Danach wird e_5 betrachtet, welcher nun am besten in den Behälter y_2 passt. Somit haben wir:

$y_1 = [e_1 \mid e_4]$ mit $c_1 = 0$
$y_2 = [e_2 \mid e_5 ...]$ mit $c_2 = 0.1$
$y_3 = [e_3 \mid ...]$ mit $c_3 = 0.5$

Als Nächstes betrachten wir e_6, dieser passt am besten in y_3. Danach wird e_7 ebenfalls noch in y_3 gepackt und wir erhalten:

$y_1 = [e_1 \mid e_4]$ mit $c_1 = 0$
$y_2 = [e_2 \mid e_5 ...]$ mit $c_2 = 0.1$
$y_3 = [e_3 \mid e_6 \mid e_7 ...]$ mit $c_3 = 0.1$

Als Letztes wird e_8 bearbeitet, dieser passt in keinen der geöffneten Behälter mehr, deshalb wird ein neuer geöffnet und e_8 darin verpackt. Somit ist die ganze Instanz abgearbeitet und wir erhalten:

$y_1 = [e_1 \mid e_4]$ mit $c_1 = 0$
$y_2 = [e_2 \mid e_5]$ mit $c_2 = 0.1$
$y_3 = [e_3 \mid e_6 \mid e_7]$ mit $c_3 = 0.1$
$y_4 = [e_8]$ mit $c_4 = 0.8$

Damit wurden nun 4 Behälter befüllt.

Das Optimum wäre jedoch nur $OPT(I) = 3$ Behälter.
$y_1 = [e_1 \mid e_4]$ mit $c_1 = 0$.
$y_2 = [e_2 \mid e_6 \mid e_7]$ mit $c_2 = 0$.
$y_3 = [e_3 \mid e_5 \mid e_8]$ mit $c_3 = 0$.

Im nachfolgenden Beispiel verdeutlichen wir, wie sich der BFD gegenüber dem BF verhält.

Beispiel:
Es gelten dieselben Voraussetzungen wie im vorherigen Beispiel.
Gegeben sei die folgende Instanz:

$$I = (0.5, 0.2, 0.1, 0.4, 0.3, 0.7, 0.8)$$

Der BF packt die ersten 3 Elemente in den ersten Behälter das vierte und fünfte Element in den nächsten und die letzten beiden in einen eigenen. Damit erhalten wir:

$y_1 = [e_1, e_2, e_3]$ mit $c_1 = 2$
$y_2 = [e_4, e_5]$ mit $c_2 = 3$
$y_3 = [e_6]$ mit $c_3 = 3$
$y_4 = [e_7]$ mit $c_4 = 2$

Wie wir hier sehen können, sind schwere Objekte am Ende ein Problem, da sie meist einen eigenen Behälter benötigen.

Im Gegensatz zum BFD. Als Erstes wird sortiert:

$$I_s = (0.8, 0.7, 0.5, 0.4, 0.3, 0.2, 0.1)$$

Nun öffnet der BFD für die ersten 3 Elemente einen eigenen Behälter, da sie aufgrund des Gewichtes stets einen eigenen benötigen. Erst e_4 wird in denselben Behälter gegeben wie auch e_3. Damit erhalten wir:

$y_1 = [e_1 \mid \ldots]$ mit $c_1 = 0.2$

$y_2 = [e_2 \mid ...]$ mit $c_2 = 0.3$
$y_3 = [e_3 \mid e_4]$ mit $c_3 = 0.1$

e_5 wird nun als Nächstes bearbeitet und passt genau in y_2, danach wird noch e_6 verarbeitet, welcher exakt in y_1 passt. Der letzte Artikel e_7 kommt dann noch in y_3, wo er ebenfalls exakt hineinpasst. Damit ist das Resultat:

$y_1 = [e_1 \mid e_6]$ mit $c_1 = 0$
$y_2 = [e_2 \mid e_5]$ mit $c_2 = 0$
$y_3 = [e_3 \mid e_4 \mid e_7]$ mit $c_3 = 0$

In diesem Fall hat der BFD sogar das Optimum gefunden.

3.4.4.4 First-Fit Decreasing

Ein weitere ähnlicher Algorithmus zu BFD ist First-Fit Decreasing (FFD). Hier ist wieder der erste Schritt das Sortieren der Inputliste, genau wie beim BFD. Danach wird der sogenannte First-Fit (FF) angewendet. Die Prozedur des First-Fit lässt sich folgendermaßen beschreiben:

> Prozedur First-Fit:
>
> Sei $x \geq 1$ die Anzahl der gerade offenen Behälter.
> Die Heuristik fügt das gerade behandelte Objekt a_i in den ersten Behälter y_j mit $1 \leq j \leq x$, in dem a_i noch passt. Besitzt kein y_j mehr ausreichend Platz, so wird ein neuer Behälter geöffnet und a_i darin verstaut. Ist in einem y_j der maximale Füllgrad erreicht, schließe den Behälter y_j.

Auch hier wird das Schließen der Behälter in der Praxis nicht implementiert. (siehe BFD)

Im Folgenden der Pseudocode:

Algorithm 5. First−Fit

Input: Same as in problem Bin−Packing
Output: A solution (x, f)

1. $x = 1$
2. for all objects $i = 1, \ldots, m$ **do**
3. $k = 0$
4. for all bins $j = 1, \ldots, x$ **do**
5. if $g(a_i) \leq \overline{g}(y_j)$ **and** $v(a_i) \leq \overline{v}(y_j)$ **then**
6. $k = j$
7. $f(i) = j$ (**pack object** i **in bin** j)
8. $\overline{g}(y_j) = \overline{g}(y_j) - g(a_i)$, $\overline{v}(y_j) = \overline{v}(y_j) - v(a_i)$
9. break loop.
10. end if
11. end for
12. if $k = 0$ **then**
13. $x = x + 1$,
14. $f(i) = x$
15. $\overline{g}(y_x) = \overline{g}(y_x) - g(a_i)$, $\overline{v}(y_x) = \overline{v}(y_x) - v(a_i)$
16. end if
17. end for

Algorithmus 5. First-Fit

Laufzeitanalyse

Wir werden hier nur die zu zählenden Schritte aufzeigen. Alle nicht erwähnten Schritte sind damit zu vernachlässigen. (Einfache Inkrementierungen, Zuweisungen etc.)
Beginnen wir mit Schritt 2 diese ist eine vom Input abhängige Schleife und hat eine Laufzeit von m. Die darin enthaltene Schleife in Schritt 4 hat eine worst case Laufzeit von $x = i$, da wir möglicherweise jeden Artikel in einen neuen Behälter geben müssen. Die beiden Vergleiche in Schritt 5 sowie das Berechnen der neuen Restkapazitäten (Schritt 8 und Schritt 15) müssen wir zählen. Damit ergibt sich insgesamt:

$$t_{\mathrm{FF}}(m) = \sum_{i=1}^{m} (4\,i + 2) = 2\,m^2 + 4\,m$$

Der First-Fit hat also die asymptotisch gleiche Laufzeit wie auch der BF:

$$t_{\mathrm{FF}}(m) \in O(m^2)$$

Wie oben bereits beschrieben, wird nun noch die Inputliste absteigend sortiert und man erhält den First-Fit Decreasing. Der Vollständigkeit halber auch hier der Pseudocode:

Algorithm 6. First−Fit−Decreasing

Input: Same as in problem Bin−Packing
Output: A solution (*x*, *f*)

1. Sort objects in decreasing order using QuickSort.
2. Apply Fest−Fit to the sorted list of objects.

Algorithmus 6. First-Fit-Decreasing

Laufzeitanalyse

Schritt 1 benötigt $O(m^2)$. Schritt 2 benötigt $O(m^2)$. In Summe also:

$$O(\,m^2 + m^2) = O(2\,m^2) = O(m^2)$$

Damit gilt also für den First-Fit Decreasing:

$$t_{\mathrm{FFD}}(m) \in O(m^2)$$

Wir werden hier kurz einige Eigenschaften des FFD zeigen, da dieses Verfahren in unserer finalen Konfiguration auch eine Rolle spielen wird.

Satz 8
Der FFD-Algorithmus hat eine relative Approximationsgüte von $k = \frac{3}{2}$ für das Bin-Packing Problem.

Beweis:

Sei o.B.d.A. die Kapazität pro Behälter auf $b = 1$ gesetzt. Weiters sei I eine Instanz, $k := \text{FFD}(I)$ und $j := \left\lceil \frac{2}{3} k \right\rceil$, wobei $\lceil x \rceil$ die kleinste ganze Zahl größer oder gleich x symbolisiert. Wir betrachten nun den Behälter B_j.

Fall 1:

Ist in B_j ein Element a der Größe $> \frac{1}{2}$, so hatte jeder andere Behälter mit Index $i < j$ nicht ausreichend Platz für a. Da die Elemente absteigend nach dem Gewicht sortiert sind, gibt es folglich mindestens j Elemente der Größe $> \frac{1}{2}$. Damit folgt $OPT(I) \geq j \geq \frac{2}{3} k$.

Fall 2:

Anderenfalls enthält B_j als auch B_x mit $x > j$ kein Element der Größe $> \frac{1}{2}$. Demnach enthalten die Behälter B_i mit $j \leq i \leq k$ mindestens $2(k-j)+1$ Elemente, von denen nicht eines in die Behälter $B_1, \ldots, B_{j-1}$ passt. Damit folgt:

$$\sum_{i=1}^{n} \gamma_i > \min\{j-1, 2(k-j)+1\} \geq \min\left\{\left\lceil \frac{2}{3} k \right\rceil - 1, 2\left(k - \left(\frac{2}{3} k + \frac{2}{3}\right)\right) + 1\right\} = \left\lceil \frac{2}{3} k \right\rceil - 1 \text{ und}$$

$$OPT(I) > \left\lceil \frac{2}{3} k \right\rceil - 1.$$

Damit $\text{OPT}(I) \geq \left\lceil \frac{2}{3} k \right\rceil$.

■

Nach Satz 7 ist dies sogar die bestmögliche relative Approximationsgüte. Die asymptotische Approximationsgüte ist jedoch besser: Beispielsweise wurde in [Johnson, 1973] gezeigt, dass gilt:

$$FFD(I) \leq \frac{11}{9} OPT(I) + 4$$

[Baker, 1985] hat diese Schranke verbessert zu 3. [Yue, 1990] machte es noch besser mit 1. Das stärkste und bestmögliche Resultat lieferte [Dósa, 2013] und ist der folgende Satz:

Satz 9 (Dósa, 2013)
Für allen Instanzen I des Bin-Packing Problems gilt:

$$\text{FFD}(I) \leq \frac{11}{9} OPT(I) + \frac{2}{3}$$

und diese Schranke ist bestmöglich.

Dieser Satz wird hier nicht bewiesen, da er ca. 30 Seiten umfasst und im Wesentlichen aus einer umfangreichen Fallunterscheidung besteht.

■

3.4.4.5 Modified Best-Fit Decreasing

Laut [Korf, 2002, S.1] und [Kratika Chandra, et al., 2014] ist der BFD im Allgemeinen etwas besser als der FFD, weshalb wir auch den BFD als Grundlage verwendet haben. Weiters wollen wir aber auch alle theoretischen Resultate, die für den FFD gelten, ebenfalls verwenden können. Aus diesem Grund haben wir für unsere Aufgabenstellung eine ergänzende Version des Best-Fit Decreasing Algorithmus verwendet. Dieser wird nun im Folgenden Modified Best-Fit-Decreasing (MBFD) genannt. Im allgemeinen Best-Fit Verfahren wird vor dem Packen eines Artikels *a* die Restkapazität von allen offenen Behältern mit *a* berechnet und *a* dann in jenen Behälter gegeben, der mit *a* die geringste Restkapazität aufweist. Diese ständigen Kapazitätsberechnungen sind sehr teuer und dies ist ebenfalls ein Grund, warum wir den Best-Fit Algorithmus wie folgt etwas abgeändert haben.
Wir haben zusätzlich eine Sortierung der Behälter nach jedem Packvorgang eingefügt, sodass die Behälter absteigend nach ihrem aktuellen Füllgrad sortiert sind. Also so, dass der Behälter mit dem aktuell höchsten Füllgrad als Erstes vom Algorithmus betrachtet wird. Nun können wir uns die Kapazitätsberechnungen sparen und jeden Artikel sofort in den ersten Behälter packen, in den der Artikel passt, denn dieser ist nun sicherlich jener Behälter, der mit dem aktuell behandelten Artikel die geringste Restkapazität aufweist. Mit dieser Sortierung wählt also der BFD den selben Behälter wie der FFD. Für den MBFD gelten also alle theoretischen Resultate, die wir für den FFD im letzten Abschnitt kennengelernt haben. Insbesondere gilt auch Satz 9, der eine Schranke für die maximale Anzahl an benötigten Behälter angibt.

In der Praxis ist natürlich $OPT(I)$ aus Satz 9 nicht bekannt. Daher haben wir hierfür Folgendes angenommen:

$$OPT(I) = max\left\{\frac{\text{Volumen aller Artikel}}{\text{Behältervolumen}}, \frac{\text{Gewicht aller Artikel}}{\text{max Beladegewicht der Behälter}}\right\}.$$

In Anlehnung an diese Schranke haben wir

$$\tilde{x} = \frac{11}{9} * \text{OPT}(I) * 1.2 + \frac{2}{3}$$

als einen Schätzwert für die benötigten Behälter des MBFD verwendet. Dabei ist 1.2 ein empirisch festgestellter Wert.

Nachstehend zeigen wir die Prozedur unseres Modified Best-Fit Algorithmus (MBF):

Algorithm 7. Modified Best−Fit

Input: Same as in problem Bin−Packing
Output: A solution (x, f)

1. $x = 1$
2. for all objects $i = 1, \ldots, m$ do
3. $\quad k = 0$
4. $\quad$ for all bins $j = 1, \ldots, x$ do
5. $\quad\quad$ if $g(a_i) \le \overline{g}(y_j)$ and $v(a_i) \le \overline{v}(y_j)$ then
6. $\quad\quad\quad k = j$
7. $\quad\quad\quad f(i) = j \quad$ (pack object i in bin j)
8. $\quad\quad\quad \overline{g}(y_j) = \overline{g}(y_j) - g(a_i)\ ,\ \overline{v}(y_j) = \overline{v}(y_j) - v(a_i)$
9. $\quad\quad\quad$ break loop.
10. $\quad\quad$ end if
11. $\quad$ end for
12. $\quad$ if $k = 0$ then (no such bin exists)
13. $\quad\quad x = x + 1 \quad$ (open a new one)
14. $\quad\quad f(i) = x \quad$ (pack object i in bin x)
15. $\quad\quad \overline{g}(y_x) = \overline{g}(y_x) - g(a_i)\ ,\ \overline{v}(y_x) = \overline{v}(y_x) - v(a_i)$
16. $\quad$ end if
17. $\quad$ for all Bins backwards $l = k, \ldots, 2$
18. $\quad\quad$ if $\overline{g}(y_l) < \overline{g}(y_{l-1})$ then
19. $\quad\quad\quad$ swap bin l and bin $l-1$
20. $\quad\quad$ else
21. $\quad\quad\quad$ break
22. $\quad\quad$ end if
23. $\quad$ end for
24. end for

Algorithmus 7. Modifizierter Best-Fit

Man achte darauf, dass die Sortierung der Behälter so gestaltet ist, dass lediglich der zuletzt behandelte neu sortiert wird. Er wird solange nach vorne verschoben, bis die ganze Liste wieder richtig sortiert ist. Weiters sei bemerkt, dass wir die Behälter bzgl. dem Beladegewicht und nicht bzgl. dem Volumen sortieren, da wir aus den Daten wissen, dass das Gewicht meistens der limitierende Faktor ist.

Laufzeitanalyse

Den Schritt 1 können wir vernachlässigen. Schritt 2 ist eine vom Input abhängige Schleife und hat eine Laufzeit von m. Die darin enthaltene Schleife in Schritt 4 hat eine worst case Laufzeit von $x = i$, da wir möglicherweise jeden Artikel in einen neuen Behälter geben müssen. Die nächsten beiden Vergleiche in Schritt 5 sind jedenfalls zu zählen. Ebenso das Berechnen der neuen Restkapazitäten, was den zwei

Subtraktionen in Schritt 8 entspricht. Damit haben wir hier insgesamt 4 zu zählende Schritte. Auch in Schritt 15 haben wir nach dem Packen eines Artikels in einen neuen Behälter die neuen Restkapazitäten zu berechnen, was weiteren 2 Subtraktionen entspricht. Weiter geht es mit Schleife 3 in Schritt 17. Diese Schleife läuft rückwärts und hat im schlimmsten Fall immer $i-1$ Vergleiche, da im worst case $k = x = i$ gilt, was bedeutet, dass der letzte Behälter stets bis ganz am Anfang nach vor verschoben werden muss.

Damit gilt nun insgesamt:

$$t_{\text{MBF}}(m) = \sum_{i=1}^{m}(4\,i + 2 + (i-1)) \;=\; \sum_{i=1}^{m}(5\,i+1) = \frac{5\,m^2 + 7\,m}{2}$$

Damit gilt nun für den MBF:

$$t_{\text{MBF}}(m) \in O(m^2)$$

Um nun unseren Modified Best-Fit Decreasing zu bekommen, sortieren wir zuvor noch die Eingabeliste .

Algorithm 8. Modified Best−Fit Decreasing (MBFD)

Input: Same as in problem Bin−Packing
Output:A solution (*x*, *f*)

1. Sort objects in decreasing order using QuickSort.
2. Apply Modified Best−Fit to the sorted list of objects.

Algorithmus 8. Modifizierter Best-Fit Decreasing

Laufzeitanalyse

Schritt 1 benötigt $O(m^2)$. Schritt 2 benötigt $O(m^2)$. In Summe also:

$$O(\,m^2 + m^2) = O(2\,m^2) = O(m^2)$$

Damit gilt also für den Modified Best-Fit Decreasing:

$$t_{\text{MBFD}}(m) \in O(m^2)$$

Bemerkung:

Damit hat der MBFD die asymptotisch gleiche Laufzeit wie der BFD, in der Praxis zeigt sich jedoch, dass der MBFD merklich schneller als der BFD ist.

4. Erweiterung der Praxistauglichkeit

Hier werden wir kurz noch einige umgesetzte Zusätze erwähnen, die die Praxistauglichkeit unsere Software noch etwas erweitern.

Verdichtung von Aufträgen

Wie wir in der Einleitung kurz erwähnt hatten, ist es wichtig die Behältnisse so dicht wie möglich zu packen. Dies ist am besten zu realisieren, wenn wir erlauben, dass mehrere verschiedene Aufträge in einem Behälter untergebracht werden dürfen. Praktisch umgesetzt haben wir das über die Dimension der Behälter. Ist es Beispielsweise erlaubt bis zu zwei Aufträge pro Behälter zuzulassen, lässt man die Algorithmen die Aufträge für die halbe Behältergröße lösen und gibt im Nachhinein die Aufträge wieder zusammen. In diesem Beispiel wären also zwei Behälter in Wirklichkeit einer und durch das Zusammenrücken der halben Behälter im Nachhinein, ist es nun möglich, dass sich auch 2 Aufträge in einem Behälter befinden. Dies ist natürlich nicht nur für zwei Aufträge beschränkt, man kann jede beliebige Teilung nehmen. Natürlich müssen alle Parameter eines Behälters auch dementsprechend angepasst werden. In unserem Beispiel oben muss auch das Gewicht, Volumen etc. halbiert werden.

Nextloading

Ebenfalls von Vorteil ist es, wenn wir automatisch mehrere Aufträge nacheinander abarbeiten können. Wie in Abschnitt 2.1 erwähnt haben, wissen wir nicht wissen, mit wie vielen Files die Aufträge als Input in unser System eingehen. Aus diesem Grund werden mehrere Files auf ein File zusammengeschrieben und mit einem Indikator versehen, der angibt, in welchem Auftrag wir uns befinden. Damit können wir nun mehrere Aufträge, wie einen einzelnen Auftrag lösen. Man könnte dies nun auch als die automatische Hintereinanderausführung des Algorithmus betrachten.

LKW Beladung

Hier ist ein ganz entscheidender Vorteil, dass wir annehmen dürfen, dass die Reihenfolge in der die Aufträge ankommen, für die weitere Verarbeitung bereits die richtige ist. Das bedeutet wir müssen für die LKW Beladung nicht die geographische Lage oder die Auslieferungszeit beachten. Wir wissen vom LKW die maximale Kapazität, die er pro Warengruppe aufnehmen kann. Damit müssen wir nur Nextloading solange anwenden bis alle Aufträge im LKW untergebracht sind oder die Kapazität einer Warengruppe erreicht ist. Ist nicht ausreichen Platz für einen Auftrag im LKW vorhanden aber der LKW noch nicht völlig ausgelastet, wird der Auftrag an das System mit einer Benachrichtigung: "Auftrag x konnte nicht im LKW untergebracht werden" zurückgegeben. Danach wird versucht, den nächsten Auftrag in den LKW zu verladen.

5. Programmaufbau

In diesem Abschnitt zeigen wir kurz, den Aufbau des Programms. Zeigen welche Klassen wir implementiert und das Programm strukturiert haben. Im Zuge dessen erläutern wir die einzelnen Klassen bei ihrer Erwähnung. An dieser Stelle sei nochmals erwähnt, dass das gesamte Programm in C# umgesetzt wurde, da dies dem Unternehmensstandard des Auftraggebers entspricht.

Die Projektmappe gliedert sich in die folgenden 6 Projekte:

- Tests:

Das Testprojekt enthält alle Tests, die im Laufe der Entwicklung des Programms gemacht wurden. Darunter sind Funktionstests und die dafür verwendeten Hilfsfunktionen. Außerdem sind hier alle Testaufträge, die zur Erzeugung der Testdaten gemacht wurden, gespeichert. Das Wichtigste in diesem Projekt sind die sogenannten Unittests. Wir haben für viele Szenarien einen solchen Unittest geschrieben und können alle gleichzeitig nacheinander ausführen lassen, um zu überprüfen, ob das Programm noch ordnungsgemäß funktioniert. (Sehr hilfreich, wenn viele Änderungen vorgenommen wurden.) Ferner haben wir zu Beginn der Testklasse eine Methode “TestInitialize”. In dieser können wir alle Einstellungen für alle Unittests an nur einem Ort verändern. Siehe Abbildung 14.

```
29          [TestInitialize]
30          public void TestInitialize()
31          {
32              packer = new BestFitAlgorithm(true);
33              //packer = new LinearAlgorithm();
34              //packer = new SimulatedAnnealingAlgorithm(1);
35
36              solutionList = new List<Solution>();
37
38              amountOfBagsInOneBin = 1;
39
40
41              binDimension = new Dimension(600,400,400);  //
42
43
44              maxBinTour = new Tour();
45
46              maxBinTour.MaxBin.Add(1000);    //Dry
47              maxBinTour.MaxBin.Add(1000);    //Chem
48              maxBinTour.MaxBin.Add(1000);    //Chill
49              maxBinTour.MaxBin.Add(1000);    //Froz
50
51          }
```

Abbildung 14. Initialisierungsmethode

Wie hier zu sehen ist, kann man den zu verwendeten Algorithmus auswählen und auch den Behälter in die gewünschte Größe unterteilen (amountOfBagsInOneBin). Im Programmcode wurde die Unterteilung der Behälter so interpretiert als würde man die Aufträge in kleinere Tüten packen und diese dann im Nachhinein in die Behälter geben. Es wurde deshalb die Bezeichnung Tüten gewählt, damit man im Programmcode sofort erkennt, ob es sich nun um eine Unterteilung des Behälters oder um den tatsächlichen (realen) Behälter handelt. Außerdem kann auch die Dimension der Behälter angegeben werden sowie die Anzahl der Behälter in einem LKW. Ferner kann sogar in Zeile 34 per

Konstruktor die Anfangstemperatur des Simulated Annealing eingestellt werden.

- BestFit.Algorithm
- Linear.Algorithm
- SimulatedAnnealing.Algorithm

In diesen drei Projekten befinden sich unsere Ansätze, die wir gemäß der vorherigen Kapitel umgesetzt haben. Dabei beinhaltet jeder Ansatz auch eine eigene Klasse "*Helpers*", die alle für die jeweiligen Algorithmen unmittelbaren Funktionen noch einmal auslagert.

- DataImport

Dieses Projekt haben wir für die Inputfiles geschrieben. Darin befindet sich unser eigens für dieses Projekt entwickelter CSV-Reader. Dieser liest die Files und speichert die Artkiel dann Zeile für Zeile in ein Element unserer Klasse *Article*.

- Domain

Im letzten Projekt Domain sind alle Klassen sowie Hilfsklassen, die für unsere Algorithmen nötig sind, untergebracht. Im Detail sind das: *Article*, *Bin*, *Dimension*, *IPackAlgorithm*, *Order*, *OrderLine*, *Solution*, *Sorting* und *Tour.*

Article; *Bin* und *Dimension* sind die Klassen, die alle Eigenschaften besitzen, um ihr dazugehöriges Wesen vollständig beschreiben zu können. So benötigt man beispielsweise für die eindeutige Charakterisierung eines Artikels für unsere Problemstellung einen Namen, ein Gewicht, seine Identifikationsnummer, die Klassifikation, also in welche Warengruppe S_i er fällt, die Stabilität sowie seine Dimension und das daraus resultierende Volumen. Hier sei bemerkt, dass die einzelnen Klassen ineinander verschachtelt sein können. So besitzt ein Artikel eine Instanz der Klasse *Dimension*.

IPackAlgorithm ist eigentlich ein Interface, das alle 3 Ansätze teilen. Das Interface enthält die Signatur der Klassen BestFit.Algorithm, Linear.Algorithm und SimulatedAnnealing.Algorithm und stellt damit eine verallgemeinerte Klasse dar. (Beispielsweise wie aus C++ bekannt die abstrakten Klassen)
Beachte: Hier sind keine konkreten Implementierungen zu finden lediglich die Signatur! Also alle Methoden und Members, die die obigen drei Klassen haben. Man kann ein Interface also als eine Schablone für die davon abgeleiteten Klassen ansehen.

Die Klasse *Order* beinhaltet eine Liste von *OrderLines* und beschreibt damit einen kompletten Auftrag. Eine *OrderLine* hingegen enthält nur einen einzigen Auszug eines Artikels. Das bedeutet, eine Variable der Klasse *OrderLine* enthält nur einen Artkiel und die Stückzahl des Artikels in dem jeweiligen Auftrag.

Solution enthält zu einem gegebenen Auftrag die Lösung, die einer unserer Ansätze generiert hat. Dazu enthält sie auch noch die Inputliste, die Stückzahl der benötigten Behälter sowie die Position aller Artikel in jedem Behälter. Für den letzten Punkt wurde in der Klasse *Solution* eine weitere Klasse *BinResult* implementiert. Diese enthält eigentlich die Position der Artkiel in den jeweiligen Behältern.

Sorting ist die Klasse, die alle Funktionen rund um das Sortieren enthält, ebenfalls wird hier das Vertauschen zweier Artikel im Simulated Annealing durchgeführt.

Die Klasse *Tour* ist für die Verpackung mehrere Aufträge in einem LKW zuständig. Deshalb enthält sie natürlich eine Liste von Lösungen für die jeweiligen Aufträge. Daneben enthält sie auch einen Zähler, der angibt, wie viele Aufträge nicht gepackt wurden, also nicht genügend Platz im restlichen LKW hatten. In diesem Zusammenhang wird auch die Auftragsnummer gespeichert, dies allerdings als String, da wir nicht wissen, ob so eine ID nur aus Zahlen besteht. Darüber hinaus sind hier die Werte des LKW´s gespeichert d.h. wie viele Behälter er generell transportieren kann, sowie für die einzelnen Temperaturzonen.

6. Numerische Resultate

In diesem Kapitel werden wir die verschiedenen Ansätze auf reale Aufträge anwenden. Wir werden Aufträge testen, wie sie von einzelnen Personen bestellt werden bis hin zu theoretischen Aufträgen, um festzustellen, bis welcher Größe die Ansätze einsetzbar sind. Dabei werden wir die benötigten Behälter jedes Verfahrens sowie die dazugehörige Laufzeit aufzeigen.

Die Messzeiten für die nachfolgenden ausgeführten Tests wurden für einen einheitlichen Vergleich auf derselben Rechenmaschine berechnet, welche durch die folgenden technischen Daten charakterisiert ist:

- Windows 7 Service Pack 1 64 Bit
- Prozessor: Intel Core i5-6200U CPU @ 2.3GHz
- RAM: 16GB DDR3
- GPU: Intel(R) HD Graphics 520
- HDD: 256 GB LITEON L8H-256V2G-HP

6.1 Aufträge ohne Verdichtung

Als Erstes werden wir nur normale einzelne Aufträge testen. Hier wird nur verglichen, wie lange und wie viele Behälter ein Ansatz für einen einzelnen Auftrag benötigt. Im Weiteren Verlauf nennen wir unser

6.1.1 Kleine Aufträge

Wir fangen klein an und testen zuerst Aufträge, wie sie beispielsweise einzelne Personen bestellen würden.

Kleiner Auftrag 1: (Order1)

Artikelanzahl ~60

Methode	Laufzeit	Benötigte Behälter	TW	KW	TKW	Chem
–	–	–	–	–	–	–
BPP – LP	301 mSek	3	1	1	0	1
Simulated Annealing	153 mSek	3	1	1	0	1
MBFD	132 mSek	3	1	1	0	1

Tabelle 2. Order1

Kleiner Auftrag 2: (Order2)

Artikelanzahl ~72

Methode	Laufzeit	Benötigte Behälter	TW	KW	TKW	Chem
–	–	–	–	–	–	–
BPP – LP	350 mSek	5	2	1	1	1
Simulated Annealing	157 mSek	5	2	1	1	1
MBFD	135 mSek	5	2	1	1	1

Tabelle 3. Order2

Kleiner Auftrag 3: (Order3)

Artikelanzahl ~77

Methode	Laufzeit	Benötigte Behälter	TW	KW	TKW	Chem
–	–	–	–	–	–	–
BPP – LP	361 mSek	5	2	1	1	1
Simulated Annealing	155 mSek	5	2	1	1	1
MBFD	137 mSek	5	2	1	1	1

Tabelle 4. Order3

Wie wir hier sehen können, liefern für diese kleinen Aufträge alle Ansätze die gleiche Lösung! Auch die Laufzeit ist bei diesen Problemen noch zu vernachlässigen.

6.1.2 Mittlere Aufträge

Als Nächstes betrachten wir Aufträge mit einer Größe von 120 - 150 Artikeln. Diese werden durchschnittlich von zwei bis vier Personen bestellt.

Mittlerer Auftrag 1: (MOrder1)

Artikelanzahl ~120

Methode	Laufzeit	Benötigte Behälter	TW	KW	TKW	Chem
–	–	–	–	–	–	–
BPP – LP	585 mSek	6	2	2	1	1
Simulated Annealing	206 mSek	6	2	2	1	1
MBFD	192 mSek	6	2	2	1	1

Tabelle 5. MOrder1

Mittlerer Auftrag 2: (MOrder2)

Artikelanzahl ~135

Methode	Laufzeit	Benötigte Behälter	TW	KW	TKW	Chem
–	–	–	–	–	–	–
BPP – LP	688 mSek	6	2	2	1	1
Simulated Annealing	221 mSek	6	2	2	1	1
MBFD	141 mSek	6	2	2	1	1

Tabelle 6. MOrder2

Mittlerer Auftrag 3: (MOrder3)

Artikelanzahl ~150

Methode	Laufzeit	Benötigte Behälter	TW	KW	TKW	Chem
–	–	–	–	–	–	–
BPP – LP	2 Sek	6	3	1	1	1
Simulated Annealing	232 mSek	6	3	1	1	1
MBFD	145 mSek	6	3	1	1	1

Tabelle 7. MOrder3

Auch hier liefern alle Ansätze die gleichen Lösungen. Das BPP-LP benötigt jedoch nun deutlich länger als die beiden Heuristiken.

6.1.3 Große Aufträge

Nun erreichen die Aufträge Größen von bis zu 700 Artikeln.

Großer Auftrag 1: (BOrder1)

Artikelanzahl ~300

Methode	Laufzeit	Benötigte Behälter	TW	KW	TKW	Chem
–	–	–	–	–	–	–
BPP – LP	8 Min	10	6	2	1	1
Simulated Annealing	500 mSek	10	6	2	1	1
MBFD	140 mSek	10	6	2	1	1

Tabelle 8. BOrder1

Großer Auftrag 2: (BOrder1.1)

Artikelanzahl ~360

Methode	Laufzeit	Benötigte Behälter	TW	KW	TKW	Chem
–	–	–	–	–	–	–
BPP – LP	11 Min	12	7	3	1	1
Simulated Annealing	513 mSek	12	7	3	1	1
MBFD	155 mSek	12	7	3	1	1

Tabelle 9. BOrder1.1

Großer Auftrag 3: (BOrder1.2)

Artikelanzahl ~400

Methode	Laufzeit	Benötigte Behälter	TW	KW	TKW	Chem
–	–	–	–	–	–	–
BPP – LP	33 Min	12	7	3	1	1
Simulated Annealing	481 mSek	12	7	3	1	1
MBFD	170 mSek	12	7	3	1	1

Tabelle 10. BOrder1.2

Großer Auftrag 4: (BOrder2)

Artikelanzahl ~500

Methode	Laufzeit	Benötigte Behälter	TW	KW	TKW	Chem
–	–	–	–	–	–	–
BPP – LP	/	/	/	/	/	/
Simulated Annealing	464 mSek	25	19	3	1	2
MBFD	144 mSek	25	19	3	1	2

Tabelle 11. BOrder2

Großer Auftrag 5: (BOrder3)

Artikelanzahl ~700

Methode	Laufzeit	Benötigte Behälter	TW	KW	TKW	Chem
–	–	–	–	–	–	–
BPP – LP	/	/	/	/	/	/
Simulated Annealing	633 mSek	26	13	6	1	6
MBFD	145 mSek	27	14	6	1	6

Tabelle 12. BOrder3

Im "Großen Auftrag 3" macht sich die exponentielle Laufzeit schon deutlich bemerkbar, denn hier benötigt das BPP-LP bereits 33 Minuten. Aus diesem Grund wurden die nachfolgenden Tests ohne diesen Ansatz durchgeführt. Wir können aber mit Sicherheit sagen, dass die beiden Heuristiken keine zu "schlechten" Lösungen liefern werden, auch wenn wir diese nun nicht mehr mit dem BPP-LP vergleichen können. Es bleibt allerdings festzuhalten, dass selbst bei solch großen Aufträgen alle Ansätze dennoch die gleichen Lösungen liefern! Die einzige Ausnahme ist der letzte Auftrag in diesem Abschnitt, hier hat zum ersten Mal das Simulated Annealing eine bessere Lösung gefunden als unser MBFD.

6.1.4 Transport Aufträge

Nun bearbeiten wir Aufträge, die ganze Transporter füllen würden. Wir bearbeiten nun Artikel bis zu ~8200 Stück.

Transport Auftrag 1: (TOrder1)

Artikelanzahl ~2000

Methode	Laufzeit	Benötigte Behälter	TW	KW	TKW	Chem
–	–	–	–	–	–	–
BPP – LP	/	/	/	/	/	/
Simulated Annealing	2 Sek	69	48	9	1	11
MBFD	374 mSek	71	49	10	1	11

Tabelle 13. TOrder1

Transport Auftrag 2: (TOrder2)

Artikelanzahl ~3067

Methode	Laufzeit	Benötigte Behälter	TW	KW	TKW	Chem
–	–	–	–	–	–	–
BPP – LP	/	/	/	/	/	/
Simulated Annealing	5 Sek	103	81	9	2	11
MBFD	401 mSek	104	82	9	2	11

Tabelle 14. TOrder2

Transport Auftrag 3: (TOrder3)

Artikelanzahl ~3119

Methode	Laufzeit	Benötigte Behälter	TW	KW	TKW	Chem
–	–	–	–	–	–	–
BPP – LP	/	/	/	/	/	/
Simulated Annealing	4 Sek	102	71	20	2	9
MBFD	402 mSek	105	73	21	2	9

Tabelle 15. TOrder3

Transport Auftrag 4: (TOrder4)

Artikelanzahl ~8166

Methode	Laufzeit	Benötigte Behälter	TW	KW	TKW	Chem
–	–	–	–	–	–	–
BPP – LP	/	/	/	/	/	/
Simulated Annealing	27 Sek	335	192	76	6	61
MBFD	435 mSek	352	209	76	6	61

Tabelle 16. TOrder4

Ab dieser Größe macht sich nun das Simulated Annealing richtig bezahlt, denn es erzielt durchweg bessere Lösungen als unser MBFD. Je größer die Aufträge werden, desto besser performt das Simulated Annealing im Gegensatz zum MBFD. Jedoch wird die bessere Lösung mit der Laufzeit aufgewogen.

6.1.5 Hypothetische Aufträge

Nun betrachten wir Aufträge mit bis zu ~400 000 Artikeln. Hier werden wir versuchen, die Ansätze bis zu ihrer Belastbarkeit auszureizen.

Riesiger Auftrag 1: (HOrder1)

Artikelanzahl ~90 000

Methode	Laufzeit	Benötigte Behälter	TW	KW	TKW	Chem
–	–	–	–	–	–	–
BPP – LP	/	/	/	/	/	/
Simulated Annealing	40 Min	2671	1854	446	37	334
MBFD	7 Sek	2818	1927	508	37	346

Tabelle 17. HOrder1

Riesiger Auftrag 2: (HOrder2)

Artikelanzahl ~130 000

Methode	Laufzeit	Benötigte Behälter	TW	KW	TKW	Chem
–	–	–	–	–	–	–
BPP – LP	/	/	/	/	/	/
Simulated Annealing	/	/	/	/	/	/
MBFD	15 Sek	4043	2711	733	55	504

Tabelle 18. HOrder2

Riesiger Auftrag 3: (HOrder3)

Artikelanzahl ~400 000

Methode	Laufzeit	Benötigte Behälter	TW	KW	TKW	Chem
–	–	–	–	–	–	–
BPP – LP	/	/	/	/	/	/
Simulated Annealing	/	/	/	/	/	/
MBFD	1 Min	10 873	7360	1923	147	1443

Tabelle 19. HOrder3

Hier wurde selbst das Simulated Annealing an seine Grenze gebracht, denn für den ersten Auftrag wurden satte 40 Minuten benötigt, wohingegen der MBFD-Algorithmus nur knappe 7 Sekunden dafür brauchte. Jedoch würde das Simulated Annealing 147 Behälter im Vergleich zum MBFD einsparen.
Sogar für ~400 000 Aufträge benötigt der MBFD-Algorithmus lediglich eine Minute, womit wir seine Belastungsgrenze bei weitem nicht erreicht haben.

6.2 Aufträge mit Verdichtung

In diesem Abschnitt werden wir einige der oberen Aufträge miteinander verdichten. Hier achten wir besonders darauf, ob sich ein qualitativer Benefit aus dieser Methodik ergibt. Aus praktischen Gründen wurden diese Tests mit 2 Tüten pro Behälter getestet. Wir schreiben für x wird mit y verdichtet kurz $x \triangleright y$.

2x MOrder 1 ▷ 2x MOrder2

Artikelanzahl ~ 510

Methode	Laufzeit	Benötigte Behälter	TW	KW	TKW	Chem
–	–	–	–	–	–	–
BPP – LP	42 Sek	17	7	6	2	2
Simulated Annealing	1 Sek	17	7	6	2	2
MBF	665 mSek	17	7	6	2	2

Tabelle 20. 2x MOrder1 ▷ 2x MOrder2

Ohne Verdichtung benötigte:

$$\text{MOrder1} = 6 \text{ Behälter je Verfahren}$$
$$\text{MOrder2} = 6 \text{ Behälter je Verfahren}$$

Somit wären für

$$2 * \text{MOrder1} + 2 * \text{MOrder2} = 24$$

Behälter nötig gewesen.

Damit haben alle Ansätze 7 Behälter eingespart.

Insgesamt gilt also:

Methode	Behälter eingespart	TW	KW	TKW	Chem
–	–	–	–	–	–
BPP – LP	7	1	2	2	2
Simulated Annealing	7	1	2	2	2
MBFD	7	1	2	2	2

Tabelle 21. Einsparungstabelle 2x MOrder1 ▷ 2x MOrder2

Bemerkung:

Hier wurden alle Tüten in die optimale Anzahl an Behältern gepackt, das bedeutet, es mussten keine halbleeren Behälter transportiert werden.

MOrder1 ▷ MOrder2 ▷ MOrder3

Artikelanzahl ~ 405

Methode	Laufzeit	Benötigte Behälter	TW	KW	TKW	Chem
–	–	–	–	–	–	–
BPP – LP	1 Min	15	7	4	2	2
Simulated Annealing	756 mSek	15	7	4	2	2
MBF	600 mSek	15	7	4	2	2

Tabelle 22. MOrder1 ▷ MOrder2 ▷ MOrder3

Ohne Verdichtung benötigte:

MOrder1 = 6 Behälter je Verfahren
MOrder2 = 6 Behälter je Verfahren
MOrder3 = 6 Behälter je Verfahren

Somit wären für:

MOrder1 + MOrder2 + MOrder3 = 18

Behälter nötig gewesen.
Damit haben alle Ansätze 3 Behälter eingespart.

Insgesamt gilt also:

Methode	Behälter eingespart	TW	KW	TKW	Chem
–	–	–	–	–	–
BPP – LP	3	0	1	2	2
Simulated Annealing	3	0	1	2	2
MBFD	3	0	1	2	2

Tabelle 23. Einsparungstabelle MOrder1 ▷ MOrder2 ▷ MOrder3

Bemerkung:

Hier wurden leider einige halbleere Behälter transportiert, denn es wurde für die jeweiligen Warengruppen folgende Anzahl an Tüten verwendet:

Trockenware 13 Tüten
Tiefkühlware 3 Tüten
Chemikalien 3 Tüten

Order1 ▷ Order2 ▷ BOrder1

Artikelanzahl ~717

Methode	Laufzeit	Benötigte Behälter	TW	KW	TKW	Chem
–	–	–	–	–	–	–
BPP – LP	50 Min	17	9	4	2	2
Simulated Annealing	630 mSek	17	9	4	2	2
MBFD	350 mSek	17	9	4	2	2

Tabelle 24. Order1 ▷ Order2 ▷ BOrder1

Ohne Verdichtung benötigte:

Order1 = 3 Behälter je Verfahren
Order2 = 5 Behälter je Verfahren
BOrder1 = 10 Behälter je Verfahren

Somit wären für:

Order1 + Order2 + BOrder1 = 18

Behälter nötig gewesen.
Damit haben alle Ansätze einen Behälter eingespart.

Insgesamt gilt also:

Methode	Behälter eingespart	TW	KW	TKW	Chem
–	–	–	–	–	–
BPP – LP	1	0	0	1	0
Simulated Annealing	1	0	0	1	0
MBFD	1	0	0	1	0

Tabelle 25. Einsparungstabelle Order1 ▷ Order2 ▷ BOrder1

Bemerkung:

Auch hier wurden einige halb leere Behälter transportiert. Die Anzahl der Tüten ist im Folgenden aufgelistet:

Trockenware 17 Tüten
Kühlware 7 Tüten
Tiefkühlware 3 Tüten

Order1 ▷ MOrder1 ▷ BOrder1 ▷ TOrder1

Artikelanzahl ~2480

Methode	Laufzeit	Benötigte Behälter	TW	KW	TKW	Chem
–	–	–	–	–	–	–
BPP – LP	/	/	/	/	/	/
Simulated Annealing	4 Sek	86	57	14	3	12
MBFD	550 mSek	88	58	15	3	12

Tabelle 26. Order1 ▷ MOrder1 ▷ BOrder1 ▷ TOrder1

Ohne Verdichtung benötigte Simulated Annealing:

Order1 = 3 Behälter
MOrder1 = 6 Behälter
BOrder1 = 10 Behälter
TOrder1 = 69 Behälter

MBFD benötigte dafür:

Order1 = 3 Behälter
MOrder1 = 6 Behälter
BOrder1 = 10 Behälter
TOrder1 = 71 Behälter

Somit wären für Simulated Annealing:

Order1 + MOrder1 + BOrder1 + TOrder1 = 88

Behälter nötig gewesen.
Für MBFD wären es 90 gewesen.

Damit hat Simulated Annealing und der MBFD jeweils 2 Behälter eingespart.

Insgesamt gilt also:

Methode	Behälter eingespart	TW	KW	TKW	Chem
–	–	–	–	–	–
BPP – LP	/	/	/	/	/
Simulated Annealing	2	0	0	0	2
MBFD	2	0	0	0	2

Tabelle 27. Einsparungstabelle Order1 ▷ MOrder1 ▷ BOrder1 ▷ TOrder1

Bemerkung:

Auch hier mussten einige halb leere Behälter transportiert werden.
Für Simulated Annealing:

Trockenware 113 Tüten
Tiefkühlware 5 Tüten

Für MBFD:

Trockenware 115 Tüten
Kühlware 29 Tüten
Tiefkühlware 5 Tüten

TOrder1 ▷ TOrder2

Artikelanzahl ~5067

Methode	Laufzeit	Benötigte Behälter	TW	KW	TKW	Chem
–	–	–	–	–	–	–
BPP – LP	/	/	/	/	/	/
Simulated Annealing	13 Sek	171	129	18	3	21
MBFD	522 mSek	175	131	19	3	22

Tabelle 28. TOrder1 ▷ TOrder2

Ohne Verdichtung benötigte Simulated Annealing:

TOrder1 = 69 Behälter
TOrder2 = 103 Behälter

MBFD benötigte dafür:

TOrder1 = 71 Behälter
TOrder2 = 104 Behälter

Somit wären für Simulated Annealing:

TOrder1 + TOrder2 = 172

Behälter nötig gewesen.
Für MBFD wären es 175 gewesen.

Damit hat nur Simulated Annealing einen Behälter einsparen können. Der MBFD hat lediglich eine Tüte im Vergleich zu der Methode ohne Verdichtung einsparen können.

Insgesamt gilt also:

Methode	Behälter eingespart	TW	KW	TKW	Chem
–	–	–	–	–	–
BPP – LP	/	/	/	/	/
Simulated Annealing	1	0	0	0	1
MBFD	0	0	0	0	0

Tabelle 29. Einsparungstabelle TOrder1 ▷ TOrder2

Bemerkung:

Auch hier wurden nicht alle Behälter vollständig mit Tüten gefüllt.
Für Simulated Annealing:

Trockenware 257 Tüten
Tiefkühlware 5 Tüten

Für MBFD:

Tiefkühlware 5 Tüten

Bemerkung:

Es sollte nicht verwunderlich sein, dass hier beinahe nichts eingespart wurde. Denn TOrder1 und TOrder2 sind sehr große Aufträge, was natürlich die Behälter nahe an ihre maximale Füllgrenze bringt. Deshalb wird auch tendenziell mehr eingespart, wenn viele kleine Aufträge verdichtet werden.

TOrder1 ▷ TOrder2 ▷ 2xMOrder1

Artikelanzahl ~5307

Methode	Laufzeit	Benötigte Behälter	TW	KW	TKW	Chem
–	–	–	–	–	–	–
BPP – LP	/	/	/	/	/	/
Simulated Annealing	14 Sek	179	132	21	4	22
MBFD	879 mSek	183	134	22	4	23

Tabelle 30. TOrder1 ▷ TOrder2 ▷ 2xMOrder1

Ohne Verdichtung benötigte Simulated Annealing:

TOrder1 = 69 Behälter
TOrder2 = 103 Behälter
MOrder1 = 6 Behälter

MBFD benötigte dafür:

TOrder1 = 71 Behälter
TOrder2 = 104 Behälter
MOrder1 = 6 Behälter

Somit wären für Simulated Annealing:

$$\text{TOrder1} + \text{TOrder2} + 2 * \text{MOrder3} = 184$$

Behälter nötig gewesen.
Für MBFD wären es 187 gewesen.

Damit hat Simulated Annealing 5 und der MBFD 4 Behälter eingespart.

Insgesamt gilt also:

Methode	Behälter eingespart	TW	KW	TKW	Chem
–	–	–	–	–	–
BPP – LP	/	/	/	/	/
Simulated Annealing	5	1	1	1	2
MBFD	4	1	1	1	1

Tabelle 31. Einsparungstabelle TOrder1 ▷ TOrder2 ▷ 2xMOrder1

Bemerkung:

Auch hier ging es sich nicht perfekt aus.
Für Simulated Annealing:

Trockenware 263 Tüten
Tiefkühlware 7 Tüten

Für MBFD:

Tiefkühlware 7 Tüten

Fazit

Der Mechanismus der Verdichtung liefert in den Tests durchweg bessere Resultate als ohne. Jedoch gilt festzuhalten, dass die Verdichtung von einigen Aufträgen gepaart mit kleinen Aufträgen grundsätzlich bessere Ergebnisse erzielt, als wenn nur große Aufträge verdichtet werden.

7. Resümee

Dieses Kapitel fasst zunächst die Hauptziele zusammen und gibt im Anschluss noch eine Handlungsempfehlung für den Auftraggeber ab. Weiters werden dann einige Gedanken besprochen, wie die Methoden noch erweitert bzw. verbessert werden können.

7.1 Zusammenfassung

Diese Abschlussarbeit hat die Ziele eine Programmpaket zu entwickeln, dass das Bin-Packing Problem für unsere Problemstellung so gut wie möglich löst und praktisch eingesetzt werden kann. Das Programmpaket enthält 3 vollkommen verschiedene Ansätze. Es enthält ein lineares Optimierungsproblem, wofür wir eigens ein mathematisches Modell entwickelt haben. Dieser Ansatz hat den Vorteil, immer die optimale Lösung zu finden. Jedoch steht dem der Nachteil der exponentiellen Laufzeit gegenüber, womit dieser Ansatz in erster Linie als Vergleich für die übrigen gedient hat. Dem berühmten Simulated Annealing, dass sehr gute (aber nicht optimale) Resultate erzielt, jedoch nicht mit Sicherheit eine gute Lösung findet. Und unserem modifizierten Best-Fit Decreasing Algorithmus, der sehr schnell Lösungen findet, die jedoch im Vergleich zu den anderen Ansätzen die schlechtesten sind.
Weiters haben wir die Praxistauglichkeit des Programmpakets noch etwas verbessert, indem wir noch die Verdichtung das Nextloading sowie eine LKW-Beladung implementiert haben.

7.2 Handlungsempfehlung

Wenn wir nun die Vor- und Nachteile der Ansätze gegeneinander abwägen, kommen wir zum Schluss, dass für die praktische Anwendung unser modifizierter Best-Fit Decreasing Algorithmus der am besten geeignetste Ansatz für unsere Aufgabenstellung ist. Zwar sind die Resultate gegenüber den anderen Ansätzen qualitativ im Nachteil, jedoch ist der qualitative Unterschied zu vernachlässigen und für die praktische Anwendung mehr als ausreichend. Aus diesem Grund kann der MBFD mit bestem Gewissen für den praktischen Einsatz empfohlen werden.

7.3 Ausblick

Es gibt eine Vielzahl an Verbesserungen. Es wäre beispielsweise denkbar, nur kleinere Aufträge mit dem BPP-LP zu lösen und den Rest mit einem der übrigen Ansätze.
Außerdem könnte versucht werden, auf unterschiedliche Art und Weise Nachbarlösungen im Simulated Annealing zu erzeugen. Dadurch könnten andere (möglicherweise bessere) Lösungen erzeugt werden. Weiters könnte die Verdichtung dahingehend optimiert werden, dass die Schnittbehälter zweier verschiedener Aufträge die gleichen oder ähnliche Produkte beinhalten, um so noch schneller packen und ausliefern zu können. Außerdem könnte in Zukunft auch die Routenplanung mit betrachtet werden. Das bedeutet, dass auch die geographische Lage der einzelnen Lieferadressen mit berücksichtigt werden. (Problem des Handlungsreisenden) Dies wird derzeit auch des Öfteren mit dem Simulated Annealing gelöst. Damit wäre auch für dieses Problem der Grundstein bereits gelegt.

Literaturverzeichnis

[Aarts&Korst, 1989]

Emile Aarts and Jan Korst, *Simulated Annealing and Bolzmann Machines, A Stochastic Approach to Combinatorial Optimization and Neural Compution,* ISBN 0 471 92146 7

[Baker, 1985]

Baker, B.S. [1985]: *A new proof for the First-Fit Decreasing bin-packing algorithm*. Journal of Algorithms 6 (1985), 49–70

[Blömer, 2010]

Johannes Blömer, Skript zur Vorlesung *Komplexitätstheorie*, Universität Paderborn, SS 2010.

[Bockenhauer et al., 2003]

Hans-Joachim Bockenhauer. Dirk Bongartz, *Algorithmische Grundlagen der Bioinformatik, Modelle, Methoden und Komplexitat,* 1. Auflage Mai 2003, B. G. Teubner Verlag / GWV Fachverlage GmbH, Wiesbaden 2003, (S. 35-42)

[C. A. R. Hoare, 1962]

C. A. R. Hoare, *Quicksort,* doi:10.1093/comjnl/5.1.10

[Černy, 1985]

Černy, V., *Thermodynamical Approach to the Traveling Salesman Problem: An Efficient Simulation Algorithm*, J. Opt. Theory Appl., 45(1985)41-51.

[Dósa, 2013]

Dósa, György (2007), *"The Tight Bound of First Fit Decreasing Bin-Packing Algorithm Is FFD(I)≤ (11/9)OPT(I)+6/9"*, in Chen, Bo; Paterson, Mike; Zhang, Guochuan, Combinatorics, Algorithms, Probabilistic and Experimental Methodologies, 4614/2007, Springer Berlin /Heidelberg, pp. 1–11, doi:10.1007/978-3-540-74450-4, ISBN 978-3-540-74449-8, ISSN 0302-9743

[Gritzmann, 2013]

Peter Gritzmann, *Grundlagen der Mathematischen Optimierung: Diskrete Strukturen, Komplexitätstheorie, Konvexitätstheorie, Lineare Optimierung, Simplex-Algorithmus, Dualität*, Springer Fachmedien Wiesbaden 2013

[Hajek et al., 1989]

Hajek, Bruce und Galen Sasaki: *Simulated annealing — to cool or not*. Systems & Control Letters, 1989, [S.443 – 447].

[Hromkovič, 1998]

Juraj Hromkovič, *Algorithmics for Hard Problems, Introduction to Combinatorial Optimization, Randomization, Approximation and Heuristics*, Springer-Verlag 2001, [S.389 - 397].

[Hüftle, 2006]

Mike Hüftle, *Heuristiken* Vorlesungsskirpt 2006

[Johnson, 1973]

Johnson, D.S. [1973]: *Near-Optimal Bin Packing Algorithms*. Doctoral Thesis, Dept. of Mathematics, MIT, Cambridge, MA, 1973

[Joswig&Theobald, 2008]

Michael Joswig, Thorsten Theobald, *Algorithmische Geometrie : Polyedrische und algebraische Methoden*, 1. Auflage 2008, Friedr. Vieweg & Sohn Verlag | GWV Fachverlage GmbH, Wiesbaden 2008 [S.245 - S.250]

[Kallrath, 2013]

Josef Kallrath, *Gemischt-ganzzahlige Optimierung: Modellierung in der Praxis : Mit Fallstudien aus Chemie, Energiewirtschaft, Papierindustrie, Metallgewerbe, Produktion und Logistik*, 2., überarbeitete und erweiterte Auflage, Springer FachmedienWiesbaden 2002, 2013

[Kirkpatrick et al., 1982]

Kirkpatrick, S., C.D. Gelatt Jr. and M.P. Vecchi, *Optimization by Simulated Annealing*, IBM Research Report RC 9955, 1982.

[Koop&Mook, 2018]

Andreas Koop, Hardy Mook, *Lineare Optimierung - eine anwendungsorientierte Einführung in Operations Research*, 2.Auflage , Springer-Verlag GmbH Deutschland 2008, 2018

[Kopp, 1999]

Herbert Kopp, *Algorithmen und Datenstrukturen*, SS99, [S.33 - S.38]

[Korf, 2002]

Richard E. Korf, *A New Algorithm for Optimal Bin Packing*,Computer Science Department University of California, Los Angeles, Los Angeles, CA 90095 korf@cs.ucla.edu

[Korte&Vygen, 2008]

Bernhard Korte, Jens Vygen *Kombinatorische Optimierung,Theorie und Algorithmen* 2.Auflage 2012 Springer Verlag Berlin Heidelberg 2008.

[Kratika et al., 2014]

Kratika Chandra , Sudhir Singh, *Firefly Algorithm to Solve Two Dimensional Bin Packing Problem*, Kratika Chandra et al, / (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 5 (4), 2014, 5368-5373 ISSN:0975-9646

[M.Sipser, 1997]

M.Sipser, *Introduction to the Theory of Computation*, 2.Edition 1997

[Ottmann&Widmayer, 1996]

T.Ottmann/P-Widmayer, *Algorithmen und Datenstrukturen*, 3.Auflage, Spektrum Verlag 1996. [S.76 - S.96]

[Varsamis, 2017]

Dimitris Varsamis, *On the Parallel Implementation of Best Fit Decreasing Algorithm in Matlab*, Contemporary Engineering Sciences, Vol. 10, 2017, no. 19, 945 - 952 HIKARI Ltd, www.m-hikari.-com
https://doi.org/10.12988/ces.2017.79120

[Wegner, 2005]

Ingo Wegner, *Theoretische Informatik -eine algorithmenorientierte Einführung*, 3.Auflage, B.G. Teubner Verlag/ GWV Fachverlage GmbH, Wiesbaden 2005 [S.37 - S.86]

[Yue, M. ,1990]

A simple proof of the inequality $FFD(L) \leq \frac{11}{9} OPT(L) + 1, \ \forall L$ *for the FFD bin-packing algorithm*. Report No. 90665, Research Institute for Discrete Mathematics, University of Bonn, 1990

[1]

Kantar Worldpanel, Europanel, Intage

[2]

Ocado Homepage: https://risnews.com/ocado-tesla-grocery (Zugriff: 15.12.2018)

[3]

TGW page : https://www.tgw-group.com/en/products/carton-and-tote-conveyor-system/workstations/workstations (Zugriff: 15.12.2018)

[4]

Mengendiagramm: https://de.wikipedia.org/wiki/NP-Vollst%C3%A4ndigkeit (Zugriff : 20.12.2018)

Eidesstattliche Erklärung

Ich Ingolf Neumüller erkläre an Eides statt, dass ich die vorliegende Masterarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.

Die vorliegende Masterarbeit ist mit dem elektronisch übermittelten Textdokument identisch.

Ort, Datum

Ingolf Neumüller