Lehrstuhl für Informatik I
Institut für Informatik
Julius-Maximilians-Universität Würzburg

Bachelorarbeit

Packalgorithmen für quaderförmige Objekte

Ilia Belozerov
Eingereicht am 26. März 2012

Betreuer:
Prof. Dr. Alexander Wolff
Dipl.-Inf. Martin Fink

Inhaltsverzeichnis

1 Einleitung

Die Effizienz vieler Prozesse der Distributionslogistik hängt unmittelbar davon ab, wie gut die Waren in Versand- und Lagerbehälter gepackt werden können. Bereits geringe Verbesserungen der Packvorgänge führen direkt zu einer deutlichen Reduktion der Prozesskosten. Die Lösung der dazu äquivalenten Zuschnittprobleme reduziert den Materialverschnitt und führt so zu Einsparungen in der Papier- oder Glasindustrie. Neben den großen wirtschaftlichen Vorteilen wirken sich gute Lösungen der Pack- und Zuschnittprobleme durch den reduzierten Rohstoffverbrauch und die verringerten Transportvorgänge positiv auf die Umweltbelastung aus.

Pack- und Zuschnittprobleme lassen sich meistens gut als ganzzahlige lineare Programme beschreiben wie zum Beispiel das klassische eindimensionale Zuschnittproblem von Gomory und Gilmore [GG61]. Diese linearen Programme liefern zwar optimale Lösungen, sind allerdings sehr zeitintensiv und eignen sich außerdem nicht für Online-Probleme, bei denen zu Beginn der Berechnung nicht alle Daten bekannt sind. Aus diesem Grund existiert auch eine Reihe von Ansätzen, die mit Hilfe von Approximationsalgorithmen annähernd optimale Lösungen sehr effizient finden können. Lodi et al. haben bereits Heuristiken für zwei- [LMM02] und dreidimensionale [LMV02] Packprobleme betrachtet, die in verschiedenen Näherungsalgorithmen Verwendung finden. Die am weitesten verbreiteten Methoden sind dabei das rekursive Aufteilen des Problems in kleinere Unterprobleme oder schrittweises Packen der vermutlich besten Objektschichten in den Behälter.

Zum Einstieg in das Thema wird im nächsten Unterabschnitt zunächst eine allgemeine Typologie betrachtet, die die verschiedenen Ausprägungen der Probleme kategorisiert. Im Anschluss konzentrieren wir uns auf einen konkreten Problemtyp, nämlich das Packen maximaler Anzahl von identischen quaderförmigen Objekten in einen quaderförmigen Behälter fester Größe, und führen dafür einige wichtige Definitionen und Sätze ein.

Im Hauptteil dieser Arbeit werden zunächst ganzzahlige lineare Programme eingeführt, die zum Finden von optimalen Lösungen für die qualitative Bewertung der heuristischen Ansätze verwendet werden (siehe Abschnitt 2). Die verschiedenen bereits bekannten Näherungsalgorithmen werden dabei im darauf folgenden Abschnitt 3 und einige eigene Ansätze in Abschnitt 4 vorgestellt. In Abschnitt 5 werden dann die ausgewerteten Vergleiche der Heuristiken auf die Laufzeit der Algorithmen und die Qualität der Lösungen vorgestellt, um in Abschnitt 6 schließlich ein Fazit über den Praxisbezug der verschiedenen Ansätze für unterschiedliche Aufgabenbereiche zu ziehen.

1.1 Problemtypologie

Pack- und Zuschnittprobleme sind schon intensiv untersucht worden und es wurden bereits verschiedene Lösungsansätze für die vielfältigen Formulierungen der Probleme vorgestellt.

Um einen Überblick über all diese Ausprägungen zu bekommen haben Wäscher et al. [WHS07] eine verbesserte Typologie vorgestellt (siehe Abbildungen 1.1 und 1.2), die auf der Typologie von Dyckhoff [Dyc90] aufbaut.

Wie in der ursprünglichen Typologie von Dyckhoff gruppiert auch die Typologie von Wäscher et al. Pack- und Zuschnittprobleme anhand ihrer Dimensionalität, ihres Aufgabentyps, der Menge der Objekte und der Menge der Behälter. Als Grundproblemtypen werden dabei nur ein-, zwei- und dreidimensionale Probleme betrachtet, Probleme mit mehr als drei Dimensionen werden als Sonderfälle angesehen und nicht typisiert.

Bei der Aufteilung nach Aufgabentyp gibt es in der Typologie die zwei Kategorien *Output-Maximierung* und *Input-Minimierung*, die unterscheiden, ob die maximale Anzahl von Objekten in die vorgegebenen Behälter gepackt werden soll, oder die minimale Anzahl der Behälter gesucht ist, die ausreichen, um die vorgegebenen Objekte zu verpacken. Die Begriffe Input und Output kommen dabei aus dem Umfeld der Zuschnittprobleme: der Input entspricht den großen Objekten, also zum Beispiel einer großen Rolle Papier, dieser Input wird in die kleineren Objekte zerschnitten, zum Beispiel DIN A4 Blätter, die wiederum den Output darstellen. Bei Packproblemen kann diese Notation allerdings anfangs etwas irritierend sein, da die fertig gepackten Behälter als Input und die Objekte, die gepackt werden sollen, als Output bezeichnet werden.

Das nächste Kriterium der Typologie ist die Menge der zu packenden Objekte. Dabei wird unterschieden, ob die Objekte identisch, schwach heterogen oder stark heterogen sind. Schwach heterogen bedeutet, dass die Anzahl der verschiedenen Formen der Objekte im Vergleich mit der Gesamtzahl der Objekte klein ist, während bei einer stark heterogenen Objektmenge keine oder nur wenige Objekte zueinander kongruent sind.

Bei der Menge der Behälter wird zunächst unterschieden, ob es sich um nur einen oder mehrere Behälter handelt. Ein einziger großer Behälter kommt nur bei Problemen vor, die den Output maximieren, und kann entweder eine feste Größe haben oder in einer oder mehreren Dimensionen offen sein. Behältermengen mit mehreren Behältern eignen sich sowohl für Input-Minimierung als auch für Output-Maximierung und können analog zu der Objektmenge identisch, schwach oder stark heterogen sein.

Zusätzlich führen Wäscher et al. noch die Form der Objekte als Kriterium ein, um zum Beispiel zwischen dem Packen von Rechtecken und Kreisen unterscheiden zu können. Die beiden grundverschiedenen Probleme würden nach Dyckhoff zur selben Problemgruppe gehören.

Das in dieser Arbeit behandelte Problem ist dreidimensional und zielt darauf ab, den Output, also die Anzahl der gepackten Objekte zu maximieren. Die zu packenden Objekte sind identisch, quaderförmig und müssen in *einen* großen Behälter fester Größe gepackt werden. Nach der Typologie von Wäscher et al. wird dieses Problem als *3-dimensional regular Identical Item Packing Problem* oder kurz *3D-IIPP* bezeichnet (siehe Abbildung 1.2).

characteristics of large objects \ assortment of small items		weakly heterogeneous	strongly heterogeneous
all dimensions fixed	identical	Single Stock Size Cutting Stock Problem SSSCSP	Single Bin Size Bin Packing Problem SBSBPP
	weakly heterogeneous	Multiple Stock Size Cutting Stock Problem MSSCSP	Multiple Bin Size Bin Packing Problem MBSBPP
	strongly heterogeneous	Residual Cutting StockProblem RCSP	Residual Bin Packing Problem RBPP
one large object variable dimension(s)		Open Dimension Problem ODP	

Abb. 1.1: Input-Minimierung Problemtypen aus [WHS07]

characteristics of the large objects \ assortment of the small items		identical	weakly heterogeneous	strongly heterogeneous
all dimensions fixed	one large object	Identical Item Packing Problem IIPP	Single Large Object Placement Problem SLOPP	Single Knapsack Problem SKP
	identical		Multiple Identical Large Object Placement Problem MILOPP	Multiple Identical Knapsack Problem MIKP
	heterogeneous		Multiple Heterogeneous Large Object Placement Problem MHLOPP	Multiple Heterogeneous Knapsack Problem MHKP

Abb. 1.2: Output-Maximierung Problemtypen aus [WHS07]

1.2 Problembeschreibung und Definitionen

Bevor man nun aber damit anfängt, verschiedene Lösungen vorzustellen, ist eine formelle Beschreibung des Problems notwendig. In diesem Abschnitt werden einige wichtige Definitionen und Sätze eingeführt, die im Verlauf der Arbeit immer wieder benötigt werden.

Wie soeben erwähnt geht es darum, möglichst viele identische quaderförmige Objekte der Länge l, Breite w und Höhe h in einen großen quaderförmigen Behälter der Länge L, Breite W und Höhe H orthogonal und ohne Überlappungen zu packen. Orthogonal bedeutet dabei, dass jede Dimension eines Objekts parallel zu jeweils einer Dimension des Behälters sein muss. Da sowohl die Objekte als auch der Behälter quaderförmig sind, bedeutet das, dass jede Dimension eines Objekts zu genau einer Dimension des Behälters parallel und zu den anderen beiden senkrecht ist. Damit ist jede gültige Ausrichtung der Objekte im Behälter eine Bijektion $o : \{l, w, h\} \to \{L, W, H\}$. Eine Objektausrichtung im Behälter wird im Folgenden verkürzt durch o_{pqr} dargestellt, wobei p, q und r jeweils die Objektdimensionen sind, die zur Länge, Breite und Höhe des Behälters parallel sind. Die sechs möglichen Ausrichtungen eines Objekts sind in Abbildung 1.3 dargestellt.

Aus dieser Problembeschreibung ergibt sich unmittelbar die folgende Definition:

Definition 1.1 (Packproblem)**.** Ein *(3D-IIPP-)Packproblem* wird durch das Tupel (L, W, H, l, w, h) repräsentiert, wobei ohne Beschränkung der Allgemeinheit $L \geq W \geq H$, $l \geq w \geq h$ und $L, W, H, l, w, h \in \mathbb{R}^+$ gilt.

Für einen gegebenen Behälter (L, W, H) nimmt man an, dass die untere linke vordere Ecke des Behälters sich im Ursprung des $\mathbb{R}^3$ befindet und die drei Dimensionen des Behälters entlang der Achsen des Koordinatensystems ausgerichtet sind.

Definition 1.2 (Packung)**.** Eine *Packung* von N Objekten für ein Packproblem ist eine Menge von Tupeln (x_i, y_i, z_i, o_i) mit $i = 1, \ldots, N$ und $x_i, y_i, z_i \in \mathbb{R}_0^+$ und $o_i \in \{o_{lwh}, o_{lhw}, o_{wlh}, o_{whl}, o_{hlw}, o_{hwl}\}$. Dabei ist (x_i, y_i, z_i) die Koordinate der unteren linken vorderen Ecke des Objekts und o_i die Ausrichtung des Objekts im Behälter. Einzelne Objekte einer Packung dürfen sich nicht überlappen und müssen sich innerhalb des Behälters befinden.

Die Koordinatenmenge der gepackten Objekte wurde soeben als eine unendliche Teilmenge des $\mathbb{R}^3$ definiert. Für die später verwendeten Verfahren ist es allerdings von Vorteil, eine endliche und möglichst kleine Koordinatenmenge zur Verfügung zu haben, die dennoch alle möglichen Packungen erzeugen kann.

Definition 1.3 (Konische Linearkombinationen)**.** Eine *konische Linearkombination* ist eine Linearkombination, bei der alle Koeffizienten größer oder gleich null sind.

Die endlichen Mengen von konischen Linearkombinationen S_L, S_W, S_H eines Packproblems (L, W, H, l, w, h) sind wie folgt definiert:

$$
\begin{aligned}
S_L &= \{x \in \mathbb{R}_0^+ \mid x = rl + sw + th, 0 \leq x \leq L, r, s, t \in \mathbb{N}_0\}, \\
S_W &= \{y \in \mathbb{R}_0^+ \mid y = rl + sw + th, 0 \leq y \leq W, r, s, t \in \mathbb{N}_0\}, \\
S_H &= \{z \in \mathbb{R}_0^+ \mid z = rl + sw + th, 0 \leq z \leq H, r, s, t \in \mathbb{N}_0\}.
\end{aligned}
$$

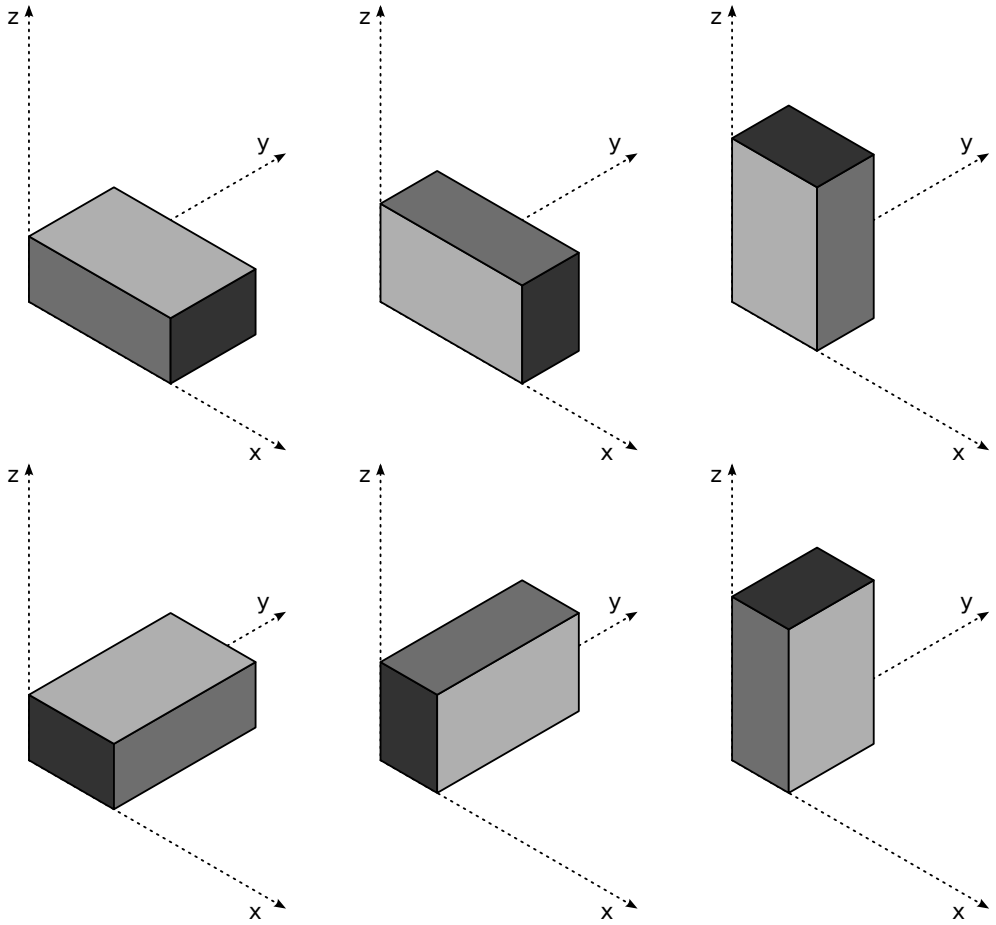

Abb. 1.3: Die sechs möglichen Ausrichtungen eines Objekts im Behälter

Der folgende Satz zeigt, dass die endliche Koordinatenmenge $S_L \times S_W \times S_H$, die aus diesen konischen Linearkombinationen erzeugt wurde, ausreichend ist, um alle möglichen Packungen zu repräsentieren:

Satz 1.4. *Sei* $(x_i'', y_i'', z_i'', o_i), i = 1, \ldots, N$ *eine beliebige Packung eines Packproblems* (L, W, H, l, w, h). *Dann existiert eine Packung* $(x_i', y_i', z_i', o_i), i = 1, \ldots, N$ *mit* $(x_i', y_i', z_i') \in S_L \times S_W \times S_H$.

Beweis. Die x-Koordinate (y-Koordinate, z-Koordinate) eines Objekts ist falsch, wenn sie nicht in S_L (S_W, S_H) liegt. Sei τ die Anzahl aller falschen Koordinaten. Bei $\tau = 0$ hat kein Objekt eine falsche Koordinate und die Annahme ist erfüllt. Nun betrachten wir eine Packung mit $\tau > 0$. Sei Q ein Objekt dieser Packung mit der kleinsten falschen x-Koordinate, x_{inf}.

Gibt es keine Objekte mit einer falschen x-Koordinate, dann gibt es ein Objekt mit der falschen y- oder z-Koordinate, da $\tau > 0$ ist, und der folgende Beweis ist analog anwendbar.

Da x_{inf} eine Koordinate eines Objekts einer Packung ist, muss x_{inf} kleiner gleich der größten Linearkombination aus S_L sein, da das Objekt sonst aus dem Behälter hinausragen würde und die Packung ungültig wäre. Analog muss x_{inf} größer gleich der kleinsten Linearkombination aus S_L sein. Somit muss x_{inf} echt zwischen zwei konischen Linearkombinationen liegen, das heißt, es existiert ein k, so dass x_{inf} im offenen Intervall $(\lambda_k, \lambda_{k+1})$ liegt, wobei $\lambda_i \in S_L$ und $\lambda_1 < \lambda_2 < \cdots < \lambda_{|S_L|}$. Nun kann man das Objekt Q nach links entlang der x-Achse verschieben, bis seine x-Koordinate mit λ_k übereinstimmt.

Dies ist genau dann unmöglich wenn in der Packung ein zweites Objekt Q' existiert, dessen rechte Seite sich im Intervall $(\lambda_k, x_{\text{inf}})$ befindet. In diesem Fall ist allerdings die x-Koordinate von Q' nicht in S_L und damit falsch, was der Annahme widerspricht, dass das gewählte x_{inf} die kleinste falsche x-Koordinate ist. Also kann man das Objekt Q verschieben und τ verkleinern, womit der Induktionsschritt abgeschlossen ist.

Nach endlich vielen Induktionsschritten ist τ schließlich 0 und die Behauptung bewiesen. □

Satz 1.4 und sein Beweis sind lediglich Verallgemeinerungen eines analogen Satzes und seines Beweises für zweidimensionale Probleme von Lins et al. [LLM03].

Das kartesische Produkt der konischen Linearkombinationen S_L, S_W und S_H ist zwar ausreichend um jede mögliche Packung darzustellen, es ist jedoch möglich, die notwendige Koordinatenmenge noch weiter zu verkleinern.

Definition 1.5 (Rasterpunkte)**.** Für ein Problem (L, W, H, l, w, h) sind die Mengen R_L, R_W, R_H, die als Mengen von Rasterpunkten [ST96] bekannt sind, wie folgt definiert:

$$
\begin{aligned}
R_L &= \{x \in \mathbb{R}_0^+ \mid x = \max\{\tilde{x} \in S_L \mid \tilde{x} \leq L - \hat{x}\}, \hat{x} \in S_L\}, \\
R_W &= \{y \in \mathbb{R}_0^+ \mid y = \max\{\tilde{y} \in S_W \mid \tilde{y} \leq W - \hat{y}\}, \hat{y} \in S_W\}, \\
R_H &= \{z \in \mathbb{R}_0^+ \mid z = \max\{\tilde{z} \in S_H \mid \tilde{z} \leq H - \hat{z}\}, \hat{z} \in S_H\}.
\end{aligned}
$$

Aus der Definition ist es leicht ersichtlich, dass die Anzahl der Rasterpunkte kleiner gleich der Anzahl der konischen Linearkombinationen ist. Der Unterschied zwischen den beiden Punktemengen wird in Abbildung 1.4 am Beispiel des zweidimensionalen Packproblems $(16, 15, 5, 3)$ verdeutlicht. Im Folgenden wird nun gezeigt, dass die Rasterpunktemengen ebenfalls ausreichen, um jede mögliche Packung zu repräsentieren.

Satz 1.6. *Sei $(x_i'', y_i'', z_i'', o_i), i = 1, \ldots, N$ eine beliebige Packung eines Packproblems (L, W, H, l, w, h). Dann existiert eine Packung $(x_i, y_i, z_i, o_i), i = 1, \ldots, N$ mit $(x_i, y_i, z_i) \in R_L \times R_W \times R_H$.*

Beweis. Nach Satz 1.4 existiert eine Packung $(x_i', y_i', z_i', o_i), i = 1, \ldots, N$ mit $(x_i', y_i', z_i') \in S_L \times S_W \times S_H$. Nun betrachten wir die Packung $(L - x_i' - d_L(o_i), W - y_i' - d_W(o_i), H - z_i' - d_H(o_i), o_i), i = 1, \ldots, N$, wobei $d_k(o_i), k \in \{L, W, H\}$ die Dimension des Quaders i ist, die bei der Ausrichtung o_i parallel zur Dimension k des Behälters ist. Anschaulich ist diese Packung eine dreifache Spiegelung der Packung (x_i', y_i', z_i', o_i). Auf diese neue gespiegelte Packung wendet man das im Beweis von Satz 1.4 beschriebene Verfahren an und erhält dadurch die Packung $(x_i, y_i, z_i, o_i), i = 1, \ldots, N$ mit $(x_i, y_i, z_i) \in R_L \times R_W \times R_H$. □

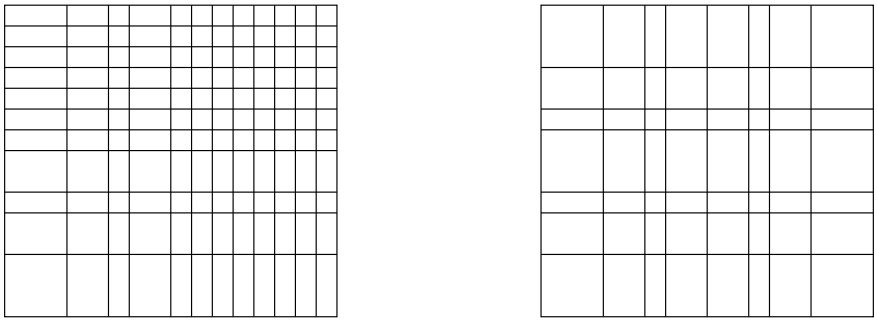

Abb. 1.4: Vergleich zwischen den konischen Linearkombinationen links und den Rasterpunkten rechts am Beispiel des Packproblems $(16, 15, 5, 3)$

Satz 1.6 ist die auf den dreidimensionalen Fall verallgemeinerte Version eines Satzes von Birgin et al. für zweidimensionale Probleme [BLM09].

2 Ganzzahlige lineare Programme

Um die Qualität der Lösungen verschiedener Ansätze testen zu können ist es vorteilhaft, ein Verfahren zur Verfügung zu haben, das in der Lage ist, eine optimale Lösung für ein gegebenes Problem zu finden. Dabei kann das Verfahren durchaus höhere Laufzeiten haben, da es nur zum Testen verwendet wird. In diesem Kapitel werden ganzzahlige lineare Programme vorgestellt, die zum Finden von optimalen Lösungen verwendet werden können.

Bei einem *linearen Programm* handelt es sich um ein Optimierungsproblem, bei dem es darum geht, den Wert einer linearen Zielfunktion unter Berücksichtigung einer Menge von Nebenbedingungen zu minimieren oder zu maximieren. Die Nebenbedingungen eines linearen Programms werden dabei als lineare Gleichungen oder Ungleichungen dargestellt. Können alle Variablen des linearen Programms beliebige reelle Werte annehmen und ist die Länge der Darstellung der Koeffizienten in den Nebenbedingungen und der Zielfunktion beschränkt, so gibt es Lösungsverfahren wie zum Beispiel die Ellipsoidmethode [Kha79] oder das Innere-Punkte-Verfahren [Kar84], die eine optimale Lösung in Polynomialzeit finden können, sofern eine Lösung existiert.

Anders verhält es sich bei *ganzzahligen* linearen Programmen, bei denen es für einige der Variablen zusätzlich die Ganzzahligkeit gefordert ist. In diesem Fall ist das Finden einer optimalen Lösung des linearen Programms NP-schwer und ist somit vermutlich nicht effizient lösbar. Mit Hilfe von spezieller für ganzzahlige lineare Programme konzipierter Problemlösesoftware ist es jedoch möglich bei Problemgrößen, die klein genug ausfallen, eine optimale Lösung in akzeptabler Zeit zu finden. Zum Lösen der im Folgenden beschriebenen linearen Programme wurde der *Gurobi Optimizer 4.6.1*[1] verwendet.

2.1 Auf kontinuierlichen Objektkoordinaten basierendes Modell

Bei der ersten Implementierung des dreidimensionalen Packens als lineares Programm betrachten wir eine Menge von Objekten, die sich frei im Behälter bewegen können, und suchen die Positionen der maximalen Teilmenge der Objekte, die sich paarweise nicht überlappen.

In diesem Modell werden für jedes Objekt i die drei kontinuierlichen Variablen x_i, y_i und z_i, die die Koordinate der linken vorderen unteren Ecke des Objekts repräsentieren, sowie eine binäre, und damit insbesondere ganzzahlige, Sichtbarkeitsvariable v_i, die aussagt, ob das Objekt gepackt ist oder nicht, angelegt. Alle Objekte werden als heterogen betrachtet und haben jeweils ihre eigenen Abmessungen, die von ihrer fest gesetzten Ausrichtung abhängen. Damit genug Objekte mit jeder Ausrichtung zur Verfügung stehen, werden von

[1] http://www.gurobi.com/

jeder Ausrichtung so viele Objekte angelegt, wie mit der maximalen homogenen Packung dieser Ausrichtung gepackt werden können.

Die *lineare* Zielfunktion, die in diesem Modell maximiert werden soll ist die Summe der binären Sichtbarkeitsvariablen:

$$\max \sum_i v_i$$

Die nötigen Nebenbedingungen fallen dagegen etwas komplizierter aus und werden im Folgenden beschrieben.

Da die Objekte sich in diesem Modell frei im Behälter bewegen können ist es notwendig für jedes Objekt die Bereiche der Positionsvariablen x_i, y_i und z_i auf die Abmessungen des Behälters zu beschränken, hierfür sind sechs Bedingungen pro Objekt notwendig. Außerdem müssen alle Objekte paarweise auf Überlappung geprüft werden, da sie sich überall innerhalb des Behälters befinden können. Für jedes Paar von Objekten sind sechs Überlappungsbedingungen und drei binäre Steuervariablen b_1, b_2 und b_3 notwendig. Jede der Bedingungen ist so mit den Steuervariablen verknüpft, dass für jede mögliche Belegung der Steuervariablen, genau eine der Bedingungen aktiviert ist und die anderen ausgeschaltet bleiben.

Da es sich bei den Bedingungen um lineare Ungleichungen handelt, wird eine Bedingung durch das Aufaddieren einer genügend großen Konstante auf einer Seite der Ungleichung erfüllt und damit ausgeschaltet. Multipliziert man diese Konstante nun mit einer binären Variable b, so ist die Bedingung bei $b = 0$ ein- und bei $b = 1$ ausgeschaltet.

Zur Verdeutlichung wird im Folgenden die Bedingung für den Fall, dass das Objekt i links vom Objekt j liegt, beschrieben. Dabei ist o_i die Ausrichtung des Objekts i und $d_k(o)$ die Objektdimension, die bei der Ausrichtung o zur Behälterdimension k parallel ist.

$$x_i + d_L(o_i) \leq x_j + \underbrace{L(b_1 + b_2 + b_3 + 1 - v_i + 1 - v_j)}_{\text{gleich null, wenn } b_1{=}0, b_2{=}0, b_3{=}0, v_i{=}1, v_j{=}1}$$

Diese Bedingung wird genau dann aktiviert, wenn die nötigen Voraussetzungen erfüllt sind und die Klammer gleich null ist. Wenn die Klammer größer null ist, ist die Ungleichung immer erfüllt, da das Objekt i echt innerhalb des Behälters liegen muss, so dass die x-Koordinate des rechten Randes von i kleiner gleich L sein muss. Die Variablen v_i und v_j schalten die Bedingung aus, wenn sie auf null gesetzt sind, wenn also eines der Objekte gar nicht gepackt ist. Außerdem ist die Bedingung nur aktiv, wenn alle drei Steuervariablen auf null gesetzt sind. Die anderen Bedingungen lassen sich analog formulieren, wobei der Inhalt der Klammer an die Steuervariablenbelegung angepasst wird, die die Bedingung aktiviert.

Die Steuervariablen verhindern es, dass zwei sich widersprechende Bedingungen gleichzeitig aktiv sind und stellen ein logisches Oder für eine Menge der Bedingungen des linearen Programms dar. Im obigen Beispiel ist es unwichtig, ob die anderen fünf Bedingungen erfüllt wären oder nicht, da sich die zwei Objekte nicht überlappen können, wenn sie in einer Dimension disjunkt sind. Da mit den drei Steuervariablen bis zu acht verschiedene Zustände kodiert werden können, aber nur sechs notwendig sind, gibt es für zwei Bedingungen jeweils zwei aktivierende Belegungen:

b_1	b_2	b_3	aktivierte Bedingung
0	0	0	Objekt i ist links von Objekt j
0	0	1	Objekt i ist rechts von Objekt j
0	1	0	Objekt i ist vor Objekt j
0	1	1	Objekt i ist hinter Objekt j
1	0	*	Objekt i ist unter Objekt j
1	1	*	Objekt i ist über Objekt j

Für die beiden unteren Bedingungen ist die Belegung von b_3 irrelevant, so dass die Variable gar nicht erst in der Bedingung vorkommt und diese damit ein wenig einfacher macht.

Der Aufbau dieses Modells macht es möglich, die Anzahl der Variablen und Bedingungen exakt zu bestimmen: Die Anzahl der Objekte N_{obj}, die modelliert werden müssen ist die Summe der maximalen homogenen Packungen aller möglichen Objektausrichtungen:

$$N_{\text{obj}} = \sum_{s \in O} \left\lfloor \frac{L}{d_L(s)} \right\rfloor \cdot \left\lfloor \frac{W}{d_W(s)} \right\rfloor \cdot \left\lfloor \frac{H}{d_H(s)} \right\rfloor, O = \{o_{lwh}, o_{lhw}, o_{wlh}, o_{whl}, o_{hlw}, o_{hwl}\}$$

Für jedes Paar von Objekten sind sechs Überlappungsbedingungen notwendig:

$$\begin{aligned} N_{\text{constr}} &= 6 \cdot \frac{(N_{\text{obj}} - 1) \cdot N_{\text{obj}}}{2} \\ &= 3 \cdot (N_{\text{obj}}^2 - N_{\text{obj}}) \end{aligned}$$

Für jedes Objekt müssen vier und für jedes Paar von Objekten drei Variablen angelegt werden:

$$\begin{aligned} N_{\text{var}} &= 4 \cdot N_{\text{obj}} + 3 \cdot \frac{(N_{\text{obj}} - 1) \cdot N_{\text{obj}}}{2} \\ &= 1{,}5 \cdot N_{\text{obj}}^2 + 2{,}5 \cdot N_{\text{obj}} \end{aligned}$$

Für das konkrete Problem $(10, 10, 10, 5, 3, 2)$ mit einer optimalen Packmenge von 33 Objekten müssten 180 Objekte modelliert werden, was zu 49 050 Variablen und 96 660 Bedingungen führen würde. Praxistest haben gezeigt, dass dieses Modell selbst bei relativ kleinen Problemgrößen zu sehr hohen Laufzeiten führt.

2.2 Auf Rasterkoordinaten basierendes Modell

Wegen den relativ hohen Laufzeiten des ersten Modells, wurde zum Vergleich ein zweites lineares Programm entwickelt. Bei diesem Modell handelt es sich um ein binäres ganzzahliges lineares Programm, bei dem die Entscheidungsvariablen ausdrücken, ob ein Objekt sich mit einer bestimmten Ausrichtung auf einem bestimmten Koordinatenpunkt befindet oder nicht. Dabei darf höchstens eine der Variablen von zwei sich überlappenden Objekten mit 1 belegt sein.

Sei A die Menge aller gültigen Entscheidungsvariablen eines Packproblems, so dass jede Variable aus A einer orthogonalen Objektplatzierung entspricht, bei der das Objekt

nicht aus dem Behälter hinausragt. Die Entscheidungsvariablen $a_{xyzo} \in A$ mit $x \in R_L$, $y \in R_W$, $z \in R_H$ und $o \in \{o_{lwh}, o_{lhw}, o_{wlh}, o_{whl}, o_{hlw}, o_{hwl}\}$ sind damit wie folgt definiert:

$$a_{xyzo} = \begin{cases} 1, & \text{falls sich auf der Koordinate } (x,y,z) \text{ ein Objekt mit der} \\ & \text{Ausrichtung } o \text{ befindet.} \\ 0, & \text{sonst} \end{cases}$$

Sei $A_{xyzo} \subseteq A$ die Menge aller Entscheidungsvariablen, die die Objekte repräsentieren, die sich mit dem durch a_{xyzo} repräsentierten Objekt überlappen. Dann kann das gesamte Programm wie folgt formuliert werden:

$$\begin{aligned} \text{maximiere} \quad & \sum_{a_{xyzo} \in A} a_{xyzo} \quad \text{unter den Nebenbedingungen:} \\ & \forall\, a_{xyzo} \in A \colon \forall\, a_{pqrs} \in A_{xyzo} \colon a_{xyzo} + a_{pqrs} \leq 1 \\ & a_{xyzo} \in \{0,1\} \\ & x \in R_L, y \in R_W, z \in R_H \\ & o \in \{o_{lwh}, o_{lhw}, o_{wlh}, o_{whl}, o_{hlw}, o_{hwl}\} \end{aligned}$$

Zwei Objekte a_{xyzo} und a_{pqrs} überlappen sich *nicht* wenn sie in einer ihrer Dimensionen disjunkt sind, also wenn mindestens eine der folgenden Ungleichungen erfüllt ist:

$$\begin{aligned} x + d_L(o) \leq p &\iff a_{xyzo} \text{ ist links von } a_{pqrs} \\ p + d_L(s) \leq x &\iff a_{xyzo} \text{ ist rechts von } a_{pqrs} \\ y + d_W(o) \leq q &\iff a_{xyzo} \text{ ist vor } a_{pqrs} \\ q + d_W(s) \leq y &\iff a_{xyzo} \text{ ist hinter } a_{pqrs} \\ z + d_H(o) \leq r &\iff a_{xyzo} \text{ ist unter } a_{pqrs} \\ r + d_H(s) \leq z &\iff a_{xyzo} \text{ ist über } a_{pqrs} \end{aligned}$$

Damit überlappen sich zwei Objekte a_{xyzo} und a_{pqrs} genau dann, wenn keine der Ungleichungen erfüllt ist. Sei $A_{xyzo}(s)$ die Menge aller Entscheidungsvariablen a_{pqrs}, die sich mit a_{xyzo} Überlappen und die Ausrichtung s haben. Diese Menge lässt sich leicht mit Hilfe der oben aufgezählten Ungleichungen beschreiben:

$$\begin{aligned} A_{xyzo}(s) = \{a_{pqrs} \in A \mid & x - d_L(s) < p < x + d_L(o) \wedge y - d_W(s) < q < y + d_W(o) \\ & \wedge z - d_H(s) < r < z + d_H(o)\} \end{aligned}$$

Dabei ist $d_k(o)$ diejenige Objektdimension, die bei der Ausrichtung o zur Behälterdimension k parallel ist. Dann ist die Menge A_{xyzo} die Vereinigung der Mengen $A_{xyzo}(s)$ für alle Ausrichtungen s:

$$A_{xyzo} = \bigcup_{s \in O} A_{xyzo}(s),\ O = \{o_{lwh}, o_{lhw}, o_{wlh}, o_{whl}, o_{hlw}, o_{hwl}\}$$

Für jede Entscheidungsvariable a_{xyzo} in A wird also die Menge A_{xyzo} bestimmt und eine Überlappungsbedingung für jedes $a_{pqrs} \in A_{xyzo}$ zum linearen Programm hinzugefügt. Anschließend wird a_{xyzo} aus A entfernt, um doppelte Bedingungen auszuschließen.

Die Komplexität des linearen Programms hängt also ausschließlich von der Zahl der möglichen Objektkoordinaten ab. Eine größere Anzahl an Koordinatenpunkten führt zu einer größeren Menge von Entscheidungsvariablen, da für jede Koordinate bis zu sechs Variablen für die sechs möglichen Ausrichtungen angelegt werden. Die größere Zahl von Variablen führt wiederum zu einer deutlich größeren Anzahl an Nebenbedingungen, da sich bei einer höheren Koordinatendichte im Behälter auch mehr Objekte paarweise überlappen.

Um die Anzahl der Koordinatenpunkte klein zu halten, werden die Koordinaten aus den Mengen der Rasterpunkte R_L, R_W und R_H, wie in Definition 1.5 beschrieben, erzeugt. Die Anzahl der Rasterkoordinaten ist kleiner als die Zahl der Koordinaten, die aus den Mengen der konischen Linearkombinationen S_L, S_W und S_H erzeugt werden würden, wodurch die Laufzeit des linearen Programms deutlich reduziert wird.

Bei Packproblemen mit einer zu hohen Zahl an Koordinaten ist die Laufzeit dieses Modells extrem hoch, so dass das lineare Programm zum Test dieser Packprobleme ungeeignet ist. Dabei kommt es nicht so sehr darauf an, wie klein die Objekte sind und wie viele in den Behälter passen würden, sondern viel mehr darauf, in welchem Verhältnis die Abmessungen der Objekte zueinander stehen. Der beste Fall tritt dabei dann ein, wenn es darum geht Würfel zu verpacken: alle drei Dimensionen sind gleich lang und die Koordinatenpunkte liegen so im Behälter, dass sich Objekte mit verschiedenen Koordinaten nicht überschneiden können. Sind dagegen die drei Objektdimensionen paarweise teilerfremd, so ist die Menge der Koordinatenpunkte, und damit auch die Koordinatendichte im Behälter, sehr groß. In diesem Fall ist das lineare Programm nur dann in annehmbarer Zeit lösbar, wenn die Objektabmessungen im Verhältnis zu den Behälterabmessungen sehr groß sind. Tests mit verschiedenen Packproblemen (L, W, H, l, w, h) haben ergeben, dass Probleme, bei denen die Objektabmessungen l, w und h prim und die Behälterabmessungen L, W und H größer gleich $l + w + h$ sind, in diesem Modell wohl zu den schwersten gehören und vermutlich nicht in akzeptabler Zeit zu lösen sind.

Da für jeden Koordinatenpunkt im Behälter eine Variable für jede mögliche Objektausrichtung existiert, sofern die Ausrichtung an dieser Koordinate zulässig ist und das Objekt nicht aus dem Behälter hinausragt, gilt die folgende obere Schranke für die Anzahl der Variablen im Modell:

$$N_{\text{var}} \leq 6 \cdot |R_L| \cdot |R_W| \cdot |R_H|$$

Die Anzahl der Überlappungsbedingungen lässt sich nur schwer bestimmen, da sie von verschiedenen Faktoren abhängt, die wiederum voneinander abhängig sind. Aus diesem Grund werden an dieser Stelle lediglich lockere Schranken für die Anzahl der Bedingungen angegeben. Eine untere Schranke für die Zahl der Überlappungen ist durch $N_{\text{constr}} = 5 \cdot N_{\text{var}}$ gegeben. Sie liegt genau dann vor, wenn Würfel gepackt werden, da jeder Würfel sich nur mit den fünf anderen Würfeln überlappt, die dieselbe Koordinate, aber andere Ausrichtungen haben. Die obere Schranke repräsentiert den Fall, dass sich jedes Objekt mit jedem anderen Objekt im Behälter überlappt, dann ist die Anzahl der Überlappungen durch

$$N_{\text{constr}} \leq \frac{(N_{\text{var}} - 1) \cdot N_{\text{var}}}{2}$$

gegeben. Die Mächtigkeit der Mengen der Rasterpunkte hängt wie bereits erwähnt von den Objekt- und Behälterabmessungen ab und kann bei ungünstigem Seitenverhältnis der Objekte sehr groß werden. Da man bei praxisbezogenen Packproblemen annehmen kann, dass die Objektdimensionen größer gleich 1 sind und die Behälterabmessungen ganzzahlig, tritt der schlechteste Fall dann ein, wenn die kürzeste Objektabmessung gleich 1 ist. In diesem Fall gilt $|R_L| = L$, $|R_W| = W$ und $|R_H| = H$, so dass die oberen Schranken für die Variablen und Bedingungen wie folgt umformuliert werden können:

$$N_{\text{var}} \leq 6 \cdot L \cdot W \cdot H$$

$$\begin{aligned} N_{\text{constr}} &\leq \frac{((6 \cdot L \cdot W \cdot H) - 1) \cdot (6 \cdot L \cdot W \cdot H)}{2} \\ &\leq 18 \cdot L^2 \cdot W^2 \cdot H^2 - 3 \cdot L \cdot W \cdot H \end{aligned}$$

Damit hat man selbst für einen kleinen Behälter mit den Abmessungen $10 \times 10 \times 10$ im schlimmsten Fall bis zu sechstausend Variablen und 18 Millionen Bedingungen, was für das lineare Programm nicht in akzeptabler Zeit zu lösen ist. Es ist jedoch zu beachten, dass diese Schranke sehr großzügig ist und diese Extremfälle in der Praxis nur sehr selten vorkommen, nichtsdestotrotz demonstriert sie gut, wie stark die Komplexität des linearen Programms für ungünstige Eingaben wachsen kann.

3 Bekannte Verfahren

Dieser Abschnitt zielt darauf ab, einige der bereits bekannten heuristischen Packalgorithmen vorzustellen. Neben dreidimensionalen Ansätzen wird auch eine zweidimensionale Heuristik vorgestellt, die als optimal vermutet wird und in dreidimensionalen Schichtenheuristiken verwendet werden kann.

3.1 Rekursive Aufteilung

Die zweidimensionale Version des Identical Item Packing Problem (2D-IIPP) wurde bereits oft und intensiv untersucht und ist unter anderem als *(Manufacturer's) Pallet Loading Problem* oder *Woodpulp Stowage Problem* bekannt. Die Lösung des 2D-IIPP liefert eine optimale Möglichkeit Objekte mit einer rechteckigen Grundfläche auf einer größeren rechteckigen Fläche zu positionieren, zum Beispiel um möglichst viel rechteckige Kartons auf einer Palette zu platzieren oder den Schiffsladeraum möglichst effizient mit rechteckigen Behältern zu füllen.

Birgin et al. [BLM09] haben ein Verfahren vorgestellt, das dieses zweidimensionale Problem für praktische Problemgrößen in akzeptabler Zeit löst. Dabei werden keine grundlegend neuen Methoden eingeführt, sondern lediglich ältere bekannte Verfahren optimiert und geschickt miteinander kombiniert, so dass das Ergebnis gegenüber den einzelnen Algorithmen jeweils in gewisser Hinsicht überlegen ist. Da bisher noch kein Gegenbeispiel gefunden wurde, wird angenommen, dass der *rekursive Aufteilungsalgorithmus* für jede Eingabe eine optimale Lösung findet. Es existiert jedoch noch kein Optimalitätsbeweis.

Der kombinierte Algorithmus ist in zwei Phasen eingeteilt: in der ersten Phase wird die *5-Block-Heuristik* von Morabito und Morales [MM98] angewandt und anschließend, falls keine nachweislich optimale Lösung gefunden wurde, der *L-Ansatz* von Lins et al. [LLM03] als zweite Phase durchgeführt. Die verwendeten Algorithmen haben dabei jeweils einige Verbesserungen gegenüber ihren ursprünglich eingeführten Varianten erhalten und sind darauf optimiert, die Synergien untereinander auszunutzen.

Die Worst-Case-Laufzeit des kombinierten Algorithmus wird dabei durch die Laufzeit des L-Ansatzes bestimmt, dessen Zeitkomplexität über der 5-Block-Heuristik liegt. Birgin et al. haben gezeigt, dass die Worst-Case-Komplexität des L-Ansatzes für ein zweidimensionales Problem (L, W, l, w) in $O(|R_L|^2|R_W|^2(|R_L||R_W| + |S_L||S_W|) + L)$ liegt.

Phase Eins: 5-Block-Heuristik

Die eben erwähnte 5-Block-Heuristik zerschneidet das große zu packende Rechteck rekursiv in kleinere Rechtecke bis es sich optimal mit einer homogenen Packung füllen lässt, oder eine vorgegebene Rekursionstiefe erreicht ist. Nach Abschluss der Rekursion wird die

Anzahl der gepackten Rechtecke als Lösung zurückgegeben. Der Name der Heuristik leitet sich dabei von der Tatsache ab, dass das Rechteck in bis zu fünf kleinere Rechtecke aufgeteilt wird wie Abbildung 3.1 verdeutlicht.

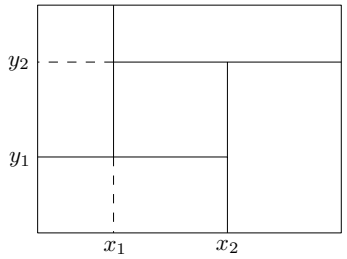

Abb. 3.1: Ein Schnitt der 5-Block-Heuristik

Ein Schnitt des 5-Block-Algorithmus wird eindeutig durch das Tupel (x_1, y_1, x_2, y_2) bestimmt, wobei ohne Beschränkung der Allgemeinheit $0 \leq x_1 \leq x_2 \leq L$ und $0 \leq y_1 \leq y_2 \leq W$ gilt. Anschaulich ist (x_1, y_1) die Koordinate der unteren linken Ecke des inneren Rechtecks und (x_2, y_2) die Koordinate der oberen rechten Ecke. Da auch Aufteilungen in weniger als fünf Rechtecke möglich sind, also zum Beispiel Schnitte mit x_1 gleich x_2, kann es vorkommen, dass das innere Rechteck in der Aufteilung nicht existiert oder leer ist.

Die Rekursionstiefe beschränkt die Anzahl der rekursiven Aufrufe des Algorithmus und damit auch die Tiefe des Rekursionsbaums. Ohne eine Beschränkung würde der Algorithmus im schlimmsten Fall das große Rechteck so lange aufteilen, bis alle entstandenen Unterrechtecke nur noch ein Objekt enthalten. Das wäre allerdings nicht effizient, da so jede mögliche Packung des Problems bestimmt werden müsste.

Ein Aufruf mit Rekursionstiefe null liefert die Anzahl der Rechtecke, die mit der besten homogenen Packung in das große Rechteck passen, zurück. Beim Aufruf mit der Rekursionstiefe k wird das Rechteck für jede gültige Belegung (x_1, y_1, x_2, y_2) in Unterrechtecke zerschnitten und der Algorithmus rekursiv mit Tiefe $k - 1$ auf jeden der Unterrechtecke angewandt. Das beste der gefundenen Ergebnisse wird schließlich als Lösung zurückgegeben.

Die von der Heuristik verwendeten Schnitte und daraus geformte Muster sind wie folgt definiert:

Definition 3.1 (Schnittmuster)**.** Ein Schnitt ist ein *Guillotine-Schnitt*, wenn er aus einem gegebenen Rechteck zwei Rechtecke erzeugt, sonst ist der Schnitt ein *Nicht-Guillotine-Schnitt*. Ein *Guillotine-Muster* ist ein Schnittmuster das durch mehrfaches Anwenden von Guillotine-Schnitten erzeugt werden kann. *Nicht-Guillotine-Muster* sind dagegen Schnittmuster, die durch wiederholte Guillotine- und Nicht-Guillotine-Schnitte erzeugt werden können. Somit sind Guillotine-Muster besondere Nicht-Guillotine-Muster.

Ein Schnitt ist *Nicht-Guillotine-Schnitt erster Ordnung*, wenn er ein gegebenes Rechteck so in fünf Rechtecke aufteilt, dass diese Rechtecke kein Guillotine-Muster darstellen (siehe Abb. 3.1). Ein Schnittmuster ist ein *Nicht-Guillotine-Muster erster Ordnung*, wenn es durch mehrfaches Anwenden von Guillotine-Schnitten und Nicht-Guillotine-Schnitten

erster Ordnung erzeugt werden kann. Nicht-Guillotine-Muster, die keine Nicht-Guillotine-Muster erster Ordnung sind, werden *Nicht-Guillotine-Muster höherer Ordnung* genannt.

Der von Morabito und Morales vorgestellte Algorithmus findet das beste Nicht-Guillotine-Muster erster Ordnung für ein gegebenes 2D-IIPP, das heißt dass kein anderes Nicht-Guillotine-Muster erster Ordnung existiert, mit dem mehr Objekte verpackt werden können, als mit dem gefundenen Muster. Der Algorithmus ist nicht optimal, da die optimale Lösung eines Packproblems unter Umständen nur als ein Muster höherer Ordnung dargestellt werden kann. Nichtsdestotrotz hat die 5-Block-Heuristik nur in 16 aus über 20 000 Beispielfällen, die von den Autoren getestet wurden, nicht die optimale Lösung gefunden und ist damit, auch vor allem aufgrund ihrer relativ kurzen Laufzeit, durchaus praxistauglich.

Birgin et al. haben den ursprünglichen 5-Block-Algorithmus noch um folgende drei Punkte verbessert:

- Verwenden von Rasterpunkten von Scheithauer und Terno [ST96], die analog zur Definition 1.5 definiert sind, statt konischen Linearkombinationen wie in Definition 1.3
- Besseres Erkennen von Symmetrien zwischen verschiedenen Schnittmustern, um unnötige mehrfache Berechnungen zu vermeiden
- Verwenden der besseren oberen Schranke von Barnes [Bar79] statt dem trivialen Flächenquotienten zum Finden der maximal möglichen Anzahl von Objekten die gepackt werden können

Barnes stellt eine Methode vor, mit der man eine untere Schranke für die ungenutzte Fläche bei einem gegebenen zweidimensionalen Problem bestimmen kann. Daraus ergibt sich unmittelbar eine obere Schranke für die tatsächlich verwendete Fläche. Der abgerundete Quotient aus der oberen Schranke der verwendeten Fläche des Behälters und der Fläche des Objekts liefert schließlich eine obere Schranke für die maximale Anzahl der Rechtecke, die gepackt werden können. Laut Dowsland [Dow85] ist die Schranke von Barnes in etwa 4,2 Prozent der Fälle dem trivialen Flächenquotienten überlegen.

Zwar erfordert die bessere Schranke zusätzlichen Berechnungsaufwand, jedoch lohnt sich ihre Verwendung, da die Schranke nur ein mal für ein Unterproblem berechnet werden muss. Sobald eine optimale Lösung für ein Unterproblem gefunden wurde, können äquivalente Unterprobleme einfach die gespeicherte Lösung abrufen und müssen nicht neu berechnet werden.

Falls die Lösung, die vom verbesserten 5-Block-Algorithmus gefunden wurde, mit der berechneten oberen Schranke übereinstimmt, ist die Lösung optimal und der kombinierte Algorithmus terminiert. Kann die Optimalität nicht nachgewiesen werden, so geht der Algorithmus in die zweite Phase.

Phase Zwei: L-Ansatz

Der L-Ansatz von Lins et al. [LLM03] ist, wie auch die 5-Block-Heuristik, ein rekursives Verfahren, bei dem größere Flächen in kleinere zerschnitten werden. Der Unterschied

besteht dabei darin, dass der L-Ansatz sich nicht nur auf Rechtecke beschränkt, sondern auch mit L-Figuren arbeitet. Eine Beispielhafte Aufteilung des zu packenden Rechtecks in L-Figuren wird in Abbildung 3.2 dargestellt. Die Autoren haben gezeigt, dass jedes Nicht-Guillotine-Muster erster Ordnung durch wiederholtes Aufteilen von Rechtecken oder L-Figuren in zwei kleinere Rechtecke oder L-Figuren erzeugt werden kann, das Gegenteil aber nicht der Fall ist. Somit ist die Lösungsmenge des 5-Block-Algorithmus eine echte Teilmenge der Lösungsmenge des L-Ansatzes.

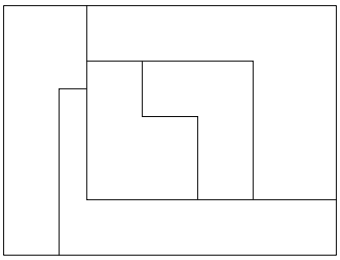

Abb. 3.2: Eine mögliche Aufteilung eines Rechtecks durch den L-Approach

Zwar ist der L-Ansatz in Hinsicht auf die Qualität der gefunden Lösung dem 5-Block-Algorithmus überlegen, jedoch hat er deutlich höhere Speicher- und Laufzeitanforderungen, weswegen er im kombinierten Algorithmus von Birgin et al. auch nur dann aufgerufen wird, falls die Optimalität der Lösung in Phase Eins nicht nachgewiesen werden konnte und man ein gründlicheres Verfahren benötigt.

Um die Laufzeit des L-Ansatzes zu reduzieren haben Birgin et al. einige Anpassungen vorgenommen. Wie auch beim 5-Block-Algorithmus, werden Rasterpunkte statt der konischen Linearkombinationen verwendet. Allerdings können die Autoren, im Gegensatz zu der Verwendung der Rasterpunkte beim 5-Block-Algorithmus, in diesem Fall keinen Beweis dafür geben, dass die Schnittmuster, die durch Schnitte an den Rasterpunkten erzeugt werden, zu den Schnittmustern äquivalent sind, die mit Schnitten an den konischen Linearkombinationen erzeugt werden.

Es wurden zwei neue Unterteilungen einer L-Figur in zwei andere L-Figuren eingeführt, die in der ursprünglichen Arbeit von Lins et al. nicht berücksichtigt wurden. Als obere Schranke wird wie auch beim 5-Block-Algorithmus, die Schranke von Barnes verwendet. Die wichtigste Verbesserung für die Laufzeit ist jedoch die Tatsache, dass die Lösungen der Unterprobleme, die in Phase Eins gespeichert wurden, in Phase Zwei nicht mehr berechnet werden müssen und somit auch einige Rekursionszweige früher abgebrochen werden können.

3.2 Dreidimensionale Schichtenheuristik

Trotz der großen wirtschaftlichen Bedeutung des Problems gibt es nur wenige Arbeiten zur dreidimensionalen Variante des IIPP. George [Geo92] stellt einen einfachen heuristischen Ansatz vor, der einen gegebenen Behälter so mit parallelen Schichten von Objekten füllt, dass das verfügbare Volumen möglichst effektiv ausgenutzt wird:

1. Finde die Lösung für jede Behälterdimension, bei der alle Schichten in diese Dimension gepackt sind:
   a) Bestimme für jede Objektdimensionen die Packmenge der Schicht, die diese Objektdimension als Dicke hat.
   b) Finde die Schichtenkombination mit der maximalen Packmenge, die noch in den Behälter passt.
   c) Fülle den verbleibenden Platz des Behälters über mehrere Schichten hinweg mit Objekten.
   d) Die gesamte gepackte Objektmenge ist die Lösung für diese Behälterdimension.
2. Gebe die beste der drei Lösungen als Ergebnis des Algorithmus zurück.

Der Algorithmus erlaubt es einen beliebigen 2D-IIPP Algorithmus zum Finden der Packmenge einer Schicht in Punkt 1a) zu verwenden. Es ist jedoch zu beachten, dass sich nicht alle Algorithmen gleich gut dafür eignen, da die Objekte der von ihnen erzeugten Packungen im dreidimensionalen Raum eventuell nicht vollständig gestützt sind. Der Autor präsentiert einen einfachen Algorithmus, der eine stabile Schlichtung der Objekte garantiert:

1. Finde die beiden möglichen homogenen Packungen.
2. Ersetze bei beiden Packungen die Objektstapel mit Stapeln mit der jeweils anderen Objektausrichtung bis die beste Kombination gefunden wurde.
3. Entferne bei beiden Packungen reihenweise Objekte aus den Stapeln, die noch in der ursprünglichen Ausrichtung sind, und ersetze sie mit Reihen von Objekten der anderen Ausrichtung bis die beste Kombination gefunden wurde.
4. Wähle die bessere der beiden Packungen.

Aufgrund des Aufbaus des Algorithmus ist es also gesichert, dass die Packung stabil ist und die Objekte mit ihrem Rand nicht über die darunterliegenden Objekte hinausragen.

Um die Qualität der gefundenen Lösung zu testen verwendet George statt dem einfachen Volumenquotienten eine verbesserte obere Schranke. Dabei wird die Tatsache ausgenutzt, dass jede beliebige Packung in eine Packung umgeformt werden kann, bei der die Objekte auf konischen Linearkombinationen liegen (siehe Satz 1.4). Damit kann man die Abmessungen des Behälters vor dem Bilden des Volumenquotienten reduzieren und so in manchen Fällen bessere Schranken erzeugen.

Sei $L' = \max S_L$, $W' = \max S_W$ und $H' = \max S_H$, dann ist das Packproblem (L', W', H', l, w, h) äquivalent zu (L, W, H, l, w, h). Da $L' \leq L$, $W' \leq W$ und $H' \leq H$ gilt, ist der Volumenquotient des verkleinerten Problems kleiner gleich dem Volumenquotienten des Ausgangsproblems und stellt damit eine bessere obere Schranke dar.

Diese Schranke lässt sich nun aber noch weiter verbessern, indem man prüft, ob die drei maximalen Linearkombinationen L', W' und H' zu einer gültigen Packung führen können und falls das nicht der Fall ist, die nächstbeste Kombination nimmt. Es ist

nicht möglich ein Objekt mit der selben Dimension in alle drei Behälterdimensionen gleichzeitig zu packen, so dass jede der drei Objektdimensionen in mindestens einer der drei Behälterdimensionen vorkommen muss, damit die Packung gültig sein kann.

Aus diesem Grund wird für jede Behälterdimension nicht die maximale Linearkombination, sondern die drei maximalen Linearkombinationen, in denen jeweils eine der Objektdimensionen mindestens ein mal vorkommt, gesucht. Für die Behälterlänge findet man die drei maximalen Linearkombinationen L_l, L_w, L_h, die jeweils mindestens ein mal die Objektlänge, -breite oder -höhe enthalten. Analog findet man die Linearkombinationen W_l, W_w, W_h für die Behälterbreite und H_l, H_w, H_h für die Behälterhöhe. Das effektiv nutzbare Behältervolumen lässt sich dann auf das Maximum der Folgenden sechs Produkte reduzieren:

$$\begin{array}{lll} L_l W_w H_h & L_w W_l H_h & L_h W_l H_w \\ L_l W_h H_w & L_w W_h H_l & L_h W_w H_l \end{array}$$

Teilt man nun das reduzierte Behältervolumen durch das Objektvolumen, so erhält man oft eine deutlich bessere Schranke, als die, die man bei Verwendung des gesamten Behältervolumens erhalten würde. Bei dem konkreten praxisbezogenen Problem $(1200, 800, 785, 520, 171, 171)$ würde der einfache Volumenquotient eine obere Schranke von 49 Objekten liefern. Wendet man allerdings das oben beschriebene Verfahren auf dieses Problem an, so führt es zu einer um fast 25% besseren oberen Schranke von nur 37 Objekten, die in diesem Fall auch die optimale Lösung des Packproblems darstellt.

3.3 Dreidimensionale 9-fach-Aufteilungsheuristik

Lins et al. [LLM99] haben eine Heuristik vorgestellt, die eine Abwandlung der 5-Block-Heuristik von Morabito und Morales [MM98] auf den dreidimensionalen Fall darstellt. Statt einem Rechteck, das in fünf kleinere Rechtecke aufgeteilt wird, teilt man einen gegebenen Quader rekursiv so in bis zu neun Unterquader auf, dass diese Aufteilung ein einfaches dreidimensionales Nicht-Guillotine-Muster erzeugt. Wie auch bei der 5-Block-Heuristik erzeugt diese Rekursion zwar nicht alle möglichen Lösungen eines Packproblems, deckt jedoch eine große Menge von Mustern ab, die oft auch eine optimale Lösung enthalten. In der Abbildung 3.3 wird diese Aufteilung eines Quaders in neun Unterquader zur Verdeutlichung schrittweise dargestellt.

Analog zur 5-Block-Heuristik ist einer der neun Unterquader, die bei einem Schnitt entstehen, komplett von den anderen Quadern eingeschlossen und bestimmt durch seine Position und Abmessungen die entstandene Aufteilung eindeutig. Der innere Quader wird durch die Koordinaten seiner linken unteren vorderen Ecke (x_1, y_1, z_1) und der rechten oberen hinteren Ecke (x_2, y_2, z_2) definiert, es gibt also sechs statt den vier Variablen des zweidimensionalen Falls, was die Laufzeit des Algorithmus deutlich erhöht.

Wegen der höheren Komplexität ist es unausweichlich, die Laufzeit des Algorithmus wie auch bei der 5-Block-Heuristik durch Beschränkung der maximal erlaubten Rekursionstiefe akzeptabel klein zu halten. Ein Aufruf des Algorithmus mit Rekursionstiefe null gibt die Anzahl von Objekten, die mit der besten homogenen Packung in den Quader passen,

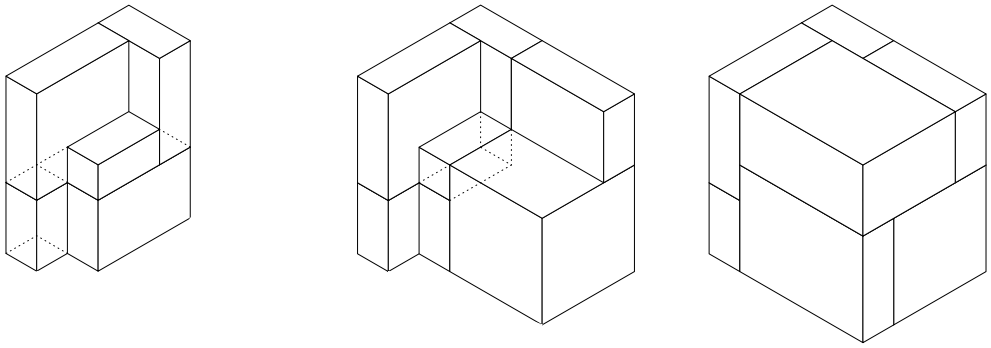

Abb. 3.3: Aufteilung eines Quaders in neun Unterquader

zurück. Beim Aufruf mit der Rekursionstiefe k wird der Quader für jede gültige Belegung von $(x_1, y_1, z_1, x_2, y_2, z_2)$ in bis zu neun Unterquader aufgeteilt und der Algorithmus rekursiv mit Tiefe $k - 1$ auf jeden der Unterquader angewandt. Anschließend werden die ermittelten Packmengen hoch propagiert und das Beste der ermittelten Ergebnisse schließlich als die gefundene Lösung zurückgegeben.

Später haben Lins et al. [LLM02] dieses Verfahren auf den n-dimensionalen Fall mit einer heterogenen Objektmenge verallgemeinert (*n-dimensional regular Single Large Object Packing Problem* nach Wäscher et al.). Außerdem haben sie in dieser Arbeit auch einige Rechenergebnisse der dreidimensionalen 9-fach-Aufteilungsheuristik vorgestellt und gezeigt, dass diese den damals bekannten Verfahren überlegen ist. Diese besseren Ergebnisse wurden jedoch mit einer Rekursiontiefe von 3 erzielt und erforderten dadurch bis zu einem Tag Rechenzeit für ein Problem. Aufgrund dieser langen Laufzeiten wurde der Ansatz nicht implementiert und getestet.

4 Eigene Ansätze

In diesem Abschnitt werden einige eigene Ansätze vorgestellt, die später mit den anderen bereits erwähnten Ansätzen auf ihre Lösungsqualität sowie Laufzeit verglichen werden.

4.1 Beste-Schicht-Zuerst-Heuristik

Im Rahmen eines Praktikums bei ZF Services wurde ein Algorithmus zum Packen von identischen quaderförmigen Objekten in einen quaderförmigen Behälter entwickelt, der auch als Motivation für diese Arbeit diente. Dieser Algorithmus wird im Folgenden vorgestellt.

Bei diesem Algorithmus handelt es sich um einen heuristischen Ansatz, der ein Guillotine-Schnitt-Muster erzeugt, indem er nach und nach Scheiben vom Behälter abschneidet und diese mit den zu packenden Objekten füllt. Jeder Schnitt ist senkrecht auf genau eine der drei Behälterdimensionen und hat eine Dicke, die genau einer der drei Objektdimensionen entspricht. In jedem Algorithmusschritt gibt es also insgesamt neun verschiedene Schnitte, aus denen der Beste gewählt wird. Die Auswahl der Schnitte erfolgt dabei anhand des ungenutzten Restvolumens jedes Schnittes, so dass immer der Schnitt angewandt wird, der das kleinste ungenutzte Restvolumen im Verhältnis zu der in das Schnittvolumen gepackten Objektmenge hat:

$$\text{Volumenverlustwert} = \frac{\text{Schnittvolumen} - \text{Objektvolumen} \cdot \text{Objektanzahl}}{\text{Objektanzahl}}$$

Handelt es sich bei einem Schnitt um den letzten Schnitt, so dass das Restvolumen des Behälters nach dem Schnitt nicht mehr ausreichen würde, um weitere Objekte zu verpacken, so wird bei der Berechnung des Volumenverlustwertes nicht das Schnittvolumen, sondern das gesamte restliche Behältervolumen verwendet, da es natürlich auch ungenutztes Volumen darstellt. Dadurch wird der letzte Schnitt stärker gewichtet und es wird verhindert, dass ein vermeintlich besserer Schnitt das gesamte restliche Volumen beansprucht und so zu einer schlechteren Lösung führt.

Um die Anzahl der Objekte zu bestimmen, die in einen gegebenen Schnitt passen, wird der Schnitt zunächst homogen mit jeder möglichen Objektausrichtung gepackt und anschließend das ungenutzte Restvolumen innerhalb des Schnittes, das selbst wiederum einen Quader darstellt, durch einen rekursiven Aufruf des Algorithmus gepackt. Das Maximum der ermittelten Packmengen wird dann tatsächlich verpackt.

Der gesamte Ablauf des Algorithmus kann damit wie folgt skizziert werden:

1. Berechnen des Volumenverlustwertes für alle möglichen Schnitte.
2. Packen des Schnittes mit dem kleinsten ungenutzten Volumen pro verpacktem Teil.

3. Reduzieren des Behältervolumens um den verpackten Schnitt.

4. Falls das Restvolumen für weiteres Packen ausreicht zurück zu Schritt 1.

5. Zurückgeben der Packmenge.

Diese Aufteilung eines Quaders in einzelne Schnitte wird in Abbildung 4.1 beispielhaft dargestellt.

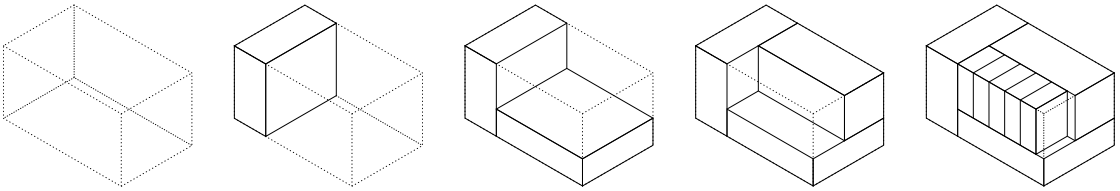

Abb. 4.1: Schrittweise Entwicklung einer Packung durch die Heuristik

Es ist leicht zu erkennen, dass die Zeitkomplexität des Algorithmus von der Anzahl der durchgeführten Schnitte abhängt. Da jeder Schnitt eine Behälterdimension um eine Objektdimension reduziert ist die maximale mögliche Anzahl von Schnitten $(L+W+H)/h$, womit der Algorithmus eine gesamte Zeitkomplexität von $O((L+W+H)/h)$ hat. Zwar ist der Algorithmus sehr schnell, jedoch liefert er nicht immer eine optimale Lösung, weswegen andere Verfahren berücksichtigt werden müssen.

4.2 Einfache Greedy Heuristik

Während die meisten Ansätze zum Lösen von Packproblemen darauf basieren den Behälter in kleinere Quader zu zerschneiden und diese dann zu packen, hat Dowsland [Dow87] einen etwas anderen, nämlich auf einem Graphen basierenden, Ansatz vorgestellt.

Dabei wird ein Graph aufgestellt, dessen Knoten die Objekte repräsentieren, die an einer bestimmten Position mit einer bestimmten Ausrichtung im Behälter gepackt werden können. Zwei Knoten sind durch eine Kante verbunden, wenn sich die durch die Knoten repräsentierten Objekte im Behälter überlappen. Für die Knotenmenge ist es dabei ausreichend nur die Rasterkoordinaten als mögliche Objektpositionen zu betrachten, da diese nach Satz 1.6 ausreichen um jede gültige Packung zu repräsentieren. Die größte unabhängige Knotenmenge des so erzeugten Graphen ist dann eine optimale Lösung des dem Graphen zugrunde liegenden Packproblems.

Da das Finden der größten unabhängigen Knotenmengen eines Graphen ein NP-schweres Problem darstellt und die Überlappungsgraphen sehr groß werden können ist es nicht zu erwarten mit dieser Methode dreidimensionale Packprobleme in der Praxis effizient lösen zu können. Dennoch kann diese Graphendarstellung dazu verwendet werden einen einfachen heuristischen Lösungsansatz zu implementieren, indem man lediglich eine maximale unabhängige Knotenmenge und damit eine gültige, wenn auch nicht optimale, Packung findet.

Dafür wird für ein gegebenes Packproblem zunächst wie beschrieben ein Überlappungsgraph aufgestellt, wobei die Überlappungen zwischen Objekten und damit auch die Kanten des Graphen mit derselben Methode bestimmt werden können, die dazu verwendet wurde die Überlappungsbedingungen für das auf Rasterkoordinaten basierte lineare Programm in Abschnitt 2.2 zu finden.

So lange dieser Graph noch Kanten hat, wird ein Knoten ausgewählt, dessen Grad im Graphen minimal ist, und seine Nachbarn aus dem Graphen entfernt, so das der gewählte Knoten isoliert wird. Nach endlich vielen Schritten hat der Graph schließlich keine Kanten mehr und alle übrig gebliebenen Knoten sind isoliert. Die Menge dieser isolierten Knoten liefert eine gültige überlappungsfreie Packung für das gegebene Packproblem. Die erzeugte Packung hängt von der Reihenfolge ab, in der die Knoten ausgewählt und entfernt werden und ist oft nicht optimal.

5 Vergleich der Methoden an Beispielen aus der Praxis

Zum Generieren der Praxisinstanzen wurden die Abmessungen der Versand- und Lagerbehälter sowie der möglichen zu packenden Objekte aus dem SAP-System von ZF Services verwendet. Anschließend wurden alle Objekte mit Seitenlängen größer gleich 50 Millimeter mit den 15 am häufigsten verwendeten Behältern kombiniert. Die Instanzen, bei denen die Objekte nicht in den Behälter passen würden, wurden heraus gefiltert. Dies ergab insgesamt 2976 Instanzen mit sehr unterschiedlichen Charakteristiken. Die kleinsten Instanzen konnten nur mit einem einzigen Objekt gepackt werden, während bei den größten über 1800 Objekte in den Behälter passten.

Zusätzlich wurden 14 Beispielinstanzen von Lins et al. [LLM02] betrachtet, in denen ein $50 \times 50 \times 50$ Würfel mit unterschiedlich großen Objekten gepackt wurde, wobei die oberen Schranken der Instanzen zwischen 28 und 394 Objekten lagen. Der Durchschnitt der unteren Schranken aller Instanzen, die den Packmengen der besten homogenen Packung entsprechen, lag bei 93, während die obere Schranke, die mit dem in Abschnitt 3.2 beschriebenen Verfahren bestimmt wurde, im Schnitt bei 99 Objekten lag.

Die Instanzen, bei denen keine der heuristischen Lösungen die obere Schranke erreichen konnte, wurden nach der Laufzeit des Greedy-Algorithmus sortiert und mit dem auf Rasterkoordinaten basierenden linearen Programm gelöst, um die optimalen Packmengen dieser Instanzen zu finden. Da die Laufzeit des linearen Programms, wie schon in Abschnitt 2.2 beschrieben, von den selben Faktoren wie die Laufzeit des Greedy-Algorithmus abhängt, wird mit einer Vorsortierung der Instanzen erreicht, dass mehr Instanzen in der verfügbaren Zeit optimal gelöst werden können, da das lineare Programm mit den einfachen Instanzen anfängt und nicht gleich bei einem sehr schweren Problem hängen bleibt. Bei den 857 Instanzen, bei denen auch das lineare Programm in der verfügbaren Zeit keine optimale Lösung finden konnte, wurde zur Bewertung der Heuristiken die obere Schranke statt des Optimums verwendet.

Definition 5.1 (Qualität)**.** Die Qualität eines Packalgorithmus für ein Packproblem ist wie folgt definiert.

$$\text{Qualität} = \frac{\text{Anzahl der durch die Heuristik gepackten Objekte}}{\text{beste obere Schranke}}$$

Die beste obere Schranke ist dabei die kleinste bekannte obere Schranke. Sofern das Optimum des Packproblems bekannt ist, ist dieses die beste mögliche obere Schranke.

Für die grafische Auswertung der Ergebnisse wurde die Menge der Instanzen nach ihrer besten oberen Schranke sortiert und in 26 Cluster von jeweils 115 Instanzen gruppiert.

Für alle Instanzen wurde die Qualität aller Algorithmen wie in definition 5.1 beschrieben berechnet. Für jedes Cluster wurde der Durchschnitt der besten oberen Schranken der Instanzen, der Qualitäten und der Laufzeiten der Algorithmen bestimmt. Insgesamt entstanden so 26 Datensätze, die sich jeweils aus dem Durchschnitt der besten oberen Schranken, den Durchschnitten der Qualitäten der Algorithmen und den durchschnittlichen Laufzeiten der Algorithmen zusammensetzen. Da die durchschnittliche beste obere Schranke der meisten Datensätze eher klein ausfällt, es aber dennoch einige wenige Datensätze mit sehr hohen durchschnittlichen besten oberen Schranken gibt, wurde die x-Achse der Diagramme zur Übersichtlichkeit logarithmisch skaliert. Bei Laufzeitdiagrammen, die ebenfalls große Differenzen zwischen den verschiedenen Algorithmen aufweisen, wurde zusätzlich die y-Achse logarithmisch skaliert. Bei den qualitativen Vergleichen bleibt die y-Achse dagegen linear.

Die getesteten Algorithmen wurden in Java implementiert und auf einem Linux System mit einen Intel Xeon X5550 2,67 GHz Prozessor und 19 GiB (etwa 20,4 GB) Arbeitsspeicher ausgeführt. Die spezifischen Implementierungsdetails der verschiedenen Algorithmen werden im Folgenden beschrieben.

5.1 Greedy-Algorithmus

Der Greedy-Algorithmus erzeugt zunächst einen Knoten für jede Position und Ausrichtung eines Objekts und ordnet anschließend jedem Knoten die Menge seiner adjazenten Knoten, die wie in Abschnitt 2.2 bestimmt wird, zu. Die so erzeugten Graphen können sehr groß werden, so dass das Generieren des Graphen sehr lange dauert oder der verfügbare Speicher nicht ausreicht, um den Graphen zu halten. Aus diesem Grund wurde die Laufzeit des Greedy-Algorithmus auf zehn Minuten beschränkt und die zurückgegebene Lösung bei Überschreiten dieses Zeitlimits auf -1 gesetzt. Für den Fall, dass der Speicher nicht ausreicht wurde eine Behelfslösung implementiert, die einen `OutOfMemoryError` abfängt und behandelt, ohne dass die Java-Virtual-Machine terminiert. Um die Instanzen im Fall eines `OutOfMemoryError` korrekt auswerten zu können, wird beim Abfangen des Fehlers sowohl die zurückgegebene Lösung als auch die benötigte Zeit auf -1 gesetzt.

5.2 Beste-Schicht-Zuerst

Die Beste-Schicht-Zuerst-Heuristik wurde in drei Varianten getestet, wobei bei jeder ein anderes zweidimensionales Verfahren verwendet wurde um die Packmengen der einzelnen Schichten zu bestimmen.

Zunächst wurde die ursprüngliche Version dieser Heuristik getestet so wie sie von mir während des Praktikums bei ZF Services geschrieben wurde. Diese Version verwendet einen sehr einfachen Algorithmus zum Packen der einzelnen Schichten und ist aufgrund unnötiger zusätzlicher Funktionalitäten auch etwas langsamer als neuere Implementierungen. Diese Version der Heuristik wird im Folgenden mit $\mathsf{BSZ}_{\mathsf{Original}}$ bezeichnet.

Für die zweite und dritte Version wurde eine neue leichtgewichtige Implementierung der Heuristik geschrieben, da die ursprüngliche Version sich nur schwer mit anderen

Methoden kombinieren ließ. Bei einer der neuen Implementierungen wurde der bessere zweidimensionale Algorithmus von George (Abschnitt 3.2) verwendet, um die Packungen der einzelnen Schichten zu bestimmen. Diese Variante wird im Folgenden mit $\mathsf{BSZ}_{\mathsf{George}}$ bezeichnet.

Die letzte der drei Varianten verwendet die als optimal vermutete und in Abschnitt 3.1 beschriebene rekursive Aufteilungsheuristik von Birgin et al. [BLM09]. Durch die Verwendung der komplexeren rekursiven Aufteilung erhöht sich die Laufzeit der Beste-Schicht-Zuerst-Heuristik. Diese Variante des Algorithmus wird im Folgenden mit $\mathsf{BSZ}_{\mathsf{RecPart}}$ bezeichnet.

Zum Berechnen der Lösungen der rekursiven Aufteilung wurde dabei das C++-Programm von Brigin und Lobato, das unter `http://lagrange.ime.usp.br/~lobato/packing/` erhältlich ist, verwendet. Bei jedem Aufruf des zweidimensionalen Algorithmus wurde über `Runtime.exec()` ein neuer nativer Prozess gestartet und sein Ausgabestrom vom aufrufenden Java-Prozess eingelesen.

Abbildung 5.1 stellt die Qualität der drei getesteten Varianten der Beste-Schicht-Zuerst-Heuristik dar, während Abbildung 5.2 die Laufzeiten der Methoden vergleicht.

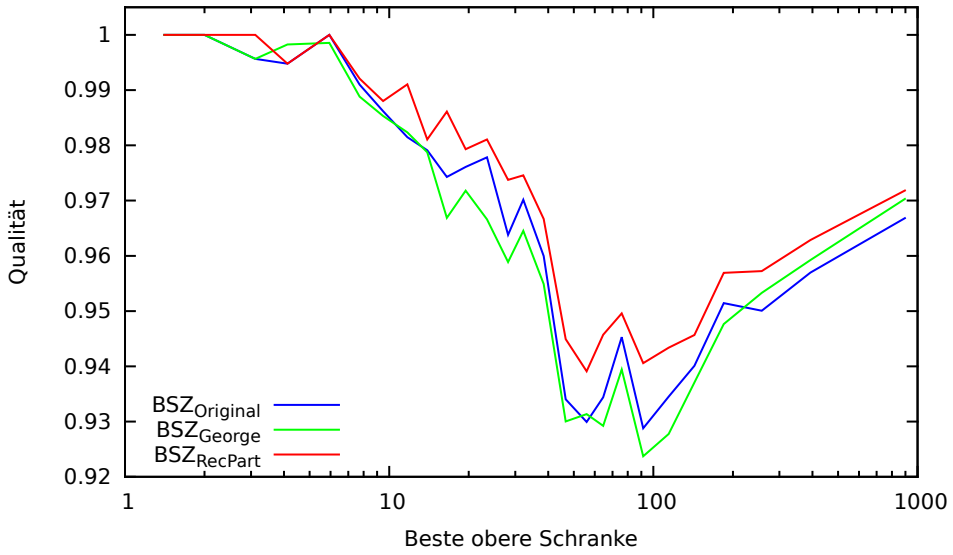

Abb. 5.1: Ergebnisse der Beste-Schicht-Zuerst-Heuristik mit verschiedenen zweidimensionalen Algorithmen

5.3 Dreidimensionale Schichtenheuristik von George

Von der Schichtenheuristik von George wurden analog zur Beste-Schicht-Zuerst-Heuristik zwei Varianten untersucht. Die erste Variante stellt den unveränderten Algorithmus von George, so wie in Abschnitt 3.2 beschrieben, dar und wird abkürzend mit $\mathsf{George}_{\mathsf{Original}}$ bezeichnet. Die zweite Variante verwendet die rekursive Aufteilung als zweidimensionalen

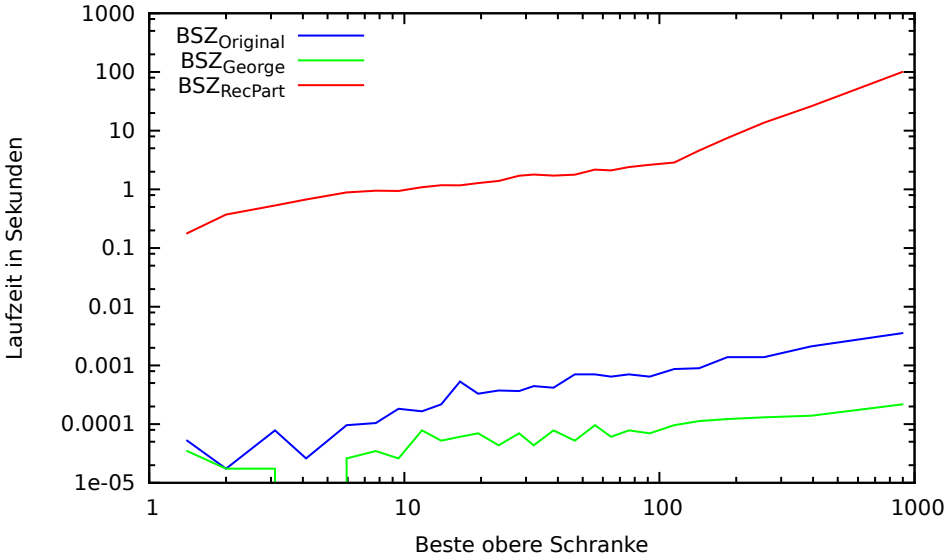

Abb. 5.2: Laufzeiten der Beste-Schicht-Zuerst-Heuristik mit verschiedenen zweidimensionalen Algorithmen

Algorithmus, was wie auch bei $\mathsf{BSZ}_{\mathsf{RecPart}}$ die Zeitkomplexität des Algorithmus erhöht. Diese zweite Variante wird bei den folgenden Tests abkürzend mit $\mathsf{George}_{\mathsf{RecPart}}$ bezeichnet. Auch hier wurde der rekursive Aufteilungsalgorithmus als externes C++-Programm aufgerufen.

Da diese Schichtenheuristik alle möglichen Kombinationen aus parallelen Schichten ausprobiert um schließlich die beste daraus zu wählen, wurde dieser Aspekt als ein einfaches lineares ganzzahliges Programm implementiert und mit Hilfe des *Gurobi Optimizer* gelöst. Die Verwendung des linearen Programms erleichtert in diesem Fall in erster Linie die Implementierung und wirkt sich nicht nennenswert auf die Laufzeit aus, da der Zeitgewinn aus der Verwendung der spezialisierten Software sich durch die höheren Initialisierungskosten weitgehend ausgleicht.

Die Schichtenheuristik von George wurde nicht in Kombination mit dem zweidimensionalen Algorithmus der ursprünglichen Beste-Schicht-Zuerst-Heuristik verwendet, da dieser sich aufgrund seines Aufbaus nicht aus dem Rest der Implementierung lösen ließ. In den Abbildungen 5.3 und 5.4 werden die beiden getesteten Varianten bezüglich ihrer Qualität und Laufzeit verglichen.

5.4 Vergleich der Algorithmen

In Tabelle 5.1 wird die Qualität und die Laufzeit der Algorithmen zusammenfassend für alle 2990 getesteten Instanzen dargestellt. Da die Berechnung der größten homogenen Packung konstante Komplexität hat und garantiert schneller ist als jeder andere Algorithmus, wurde auf das Messen der Laufzeiten in diesem Fall verzichtet. Aus der

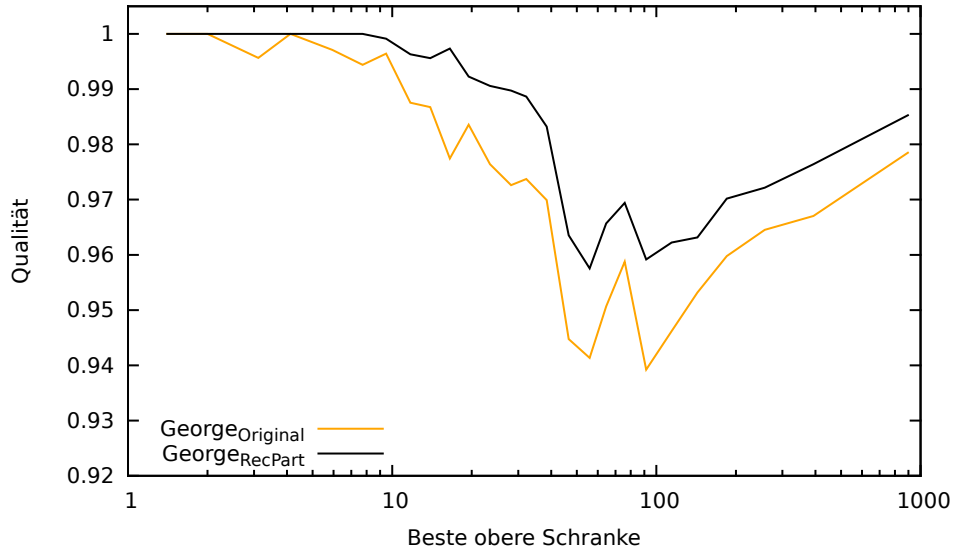

Abb. 5.3: Ergebnisse der Schichtenheuristik von George im Original und mit rekursiver Aufteilung als zweidimensionalem Algorithmus

Tabelle kann man sofort erkennen, dass die mit der rekursiven Aufteilung kombinierte Version der Schichtenheuristik von George die besten Ergebnisse liefert, während die neue leichtgewichtige Implementierung der Beste-Schicht-Zuerst-Heuristik neben der homogenen Packung die mit Abstand niedrigsten Laufzeiten hat. Die Laufzeiten der Algorithmen wurden bei den Tests auf die Millisekunde genau bestimmt, weswegen der berechnete Durchschnitt von 0,07 ms je Problem zwar nicht besonders genau ist, jedoch demonstriert er eindrucksvoll, wie viele Instanzen in unter einer Millisekunde gelöst werden konnten.

In Abbildung 5.5 wird die Qualität aller Algorithmen in Abhängigkeit von der besten oberen Schranke dargestellt. Eine wichtige Beobachtung ist dabei die Tatsache, dass zum

Algorithmus	Optimum gefunden	Anteil am Optimum	Durchschnittliche Laufzeit je Problem in ms	Standardabweichung der Laufzeit
Homogene Packung	44,8%	90,5%	-	-
Greedy	49,9%	95,0%	64826,84	183329,07
BSZ	55,9%	96,6%	0,65	1,27
BSZ_{George}	53,6%	96,4%	**0,07**	**0,26**
$BSZ_{RecPart}$	57,7%	97,2%	6989,76	44947,77
George	61,1%	97,4%	2,98	3,81
$George_{RecPart}$	**67,0%**	**98,4%**	2110,62	12604,68

Tab. 5.1: Vergleich aller Algorithmen über alle Testinstanzen

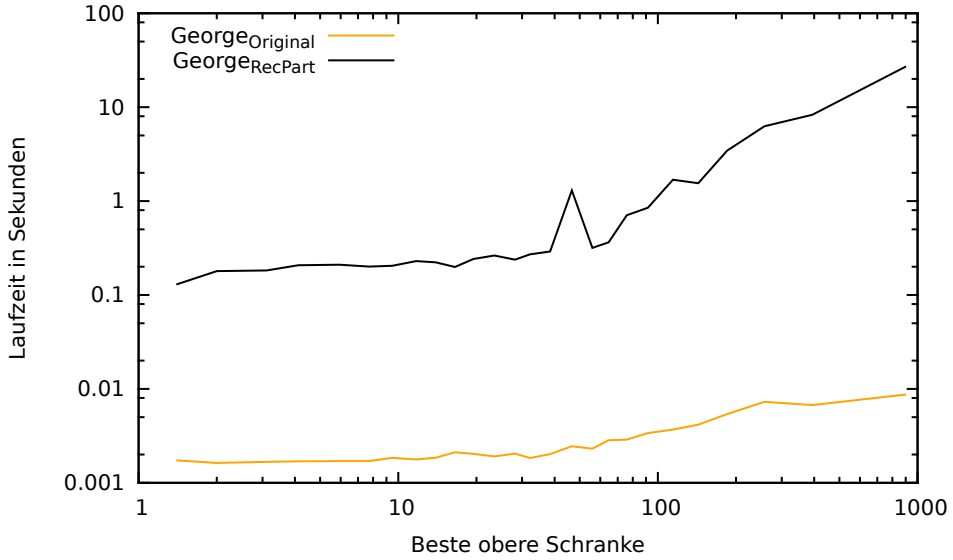

Abb. 5.4: Laufzeiten der Schichtenheuristik von George im Original und mit rekursiver Aufteilung als zweidimensionalem Algorithmus

Bestimmen der Qualität des Greedy-Algorithmus nur diejenigen Instanzen betrachtet wurden, in denen der Algorithmus eine Lösung vor Ablauf des Zeitlimits von 10 Minuten gefunden hat. Abbildung 5.6 stellt den Anteil der gelösten Instanzen in Abhängigkeit von der besten oberen Schranke dar.

Eine andere Beobachtung, die aus dieser Abbildung hervorgeht, ist die Tatsache, das die größten homogenen Packungen in allen Problemclustern von den Lösungen aller anderen Algorithmen dominiert werden. Es besteht eine wesentliche Differenz zwischen der größten homogenen Packung (gelb in der Abbildung 5.5) und der Lösung des ebenfalls sehr schnellen Algorithmus $\mathsf{BSZ}_{\mathsf{George}}$ (grün), so das die größte homogene Packung nicht ein mal in sehr zeitkritischen Umgebungen als Lösungsmethode für das 3D-IIPP verwendet werden sollte.

Wie schon in den Abbildungen 5.1 und 5.3 zu erkennen ist, wirkt sich das bei den Schichtenheuristiken verwendete zweidimensionale Verfahren stark auf die gefundene dreidimensionale Lösung aus. Die rekursive Aufteilung von Birgin et al. [BLM09] war sowohl bei der Heuristik von George als auch bei der Beste-Schicht-Zuerst-Heuristik den anderen getesteten Varianten deutlich überlegen. Aus Abbildung 5.5 ist leicht zu erkennen, dass $\mathsf{George}_{\mathsf{RecPart}}$ alle anderen Verfahren auf der getesteten Menge von Instanzen dominiert, wie schon die Tabelle 5.1 vermuten ließ.

Ein anderer wichtiger Aspekt der Tests, nämlich die Laufzeitbetrachtungen der verwendeten Verfahren, wird in der Abbildung 5.7 grafisch präsentiert. Die Grafik stellt dabei sehr deutlich die durchschnittlichen Laufzeiten der Algorithmen für jedes Problem in Abhängigkeit von der besten bekannten oberen Schranke dar. Besonders gut ist dabei zu erkennen, wie schnell die Laufzeit des Greedy-Algorithmus ansteigt, vor allem wenn man bedenkt, dass die y-Achse in der Darstellung logarithmisch skaliert ist. Die Laufzeitun-

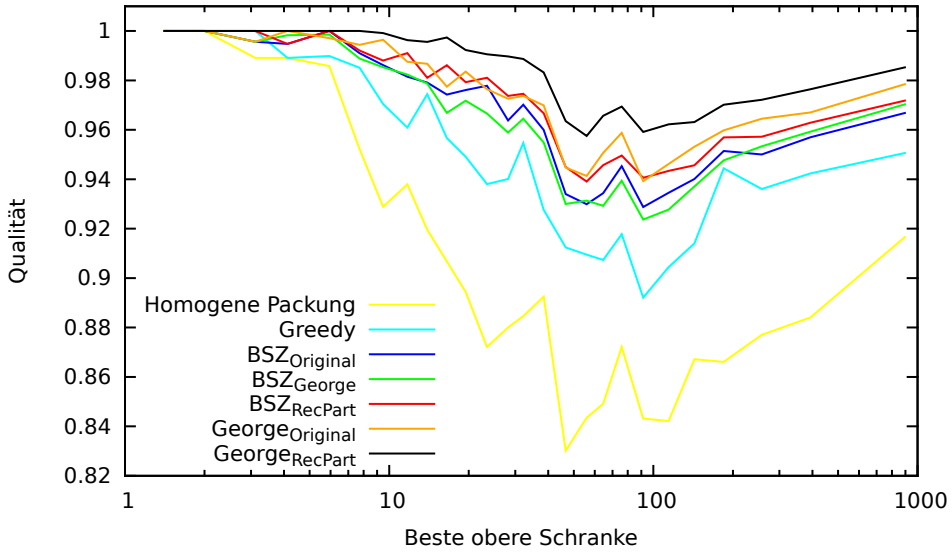

Abb. 5.5: Übersicht über alle getesteten Methoden

terschiede der anderen Methoden verhalten sich wie erwartet und entsprechen den in der Tabelle 5.1 vorgestellten Gesamtdurchschnitten.

Vergleicht man die Laufzeitdifferenzen zwischen den ursprünglichen Versionen der Schichtenheuristiken und ihren mit der rekursiven Aufteilung kombinierten Varianten, so erkennt man, dass sich die Laufzeit des BSZ-Algorithmus bei Verwendung der rekursiven Aufteilung deutlich stärker erhöht, als im Fall der George-Heuristik. Das liegt daran, dass bei George die rekursive Aufteilung immer konstante 9 mal je Problem aufgerufen werden muss, während bei der Beste-Schicht-Zuerst-Heuristik bis zu $(L + W + H)/h$ Aufrufe pro Problem notwendig sind.

Die bereits erwähnte sehr niedrige Laufzeit von $\mathrm{BSZ}_{\mathrm{George}}$ ist auch in Abbildung 5.7 sehr gut zu erkennen.

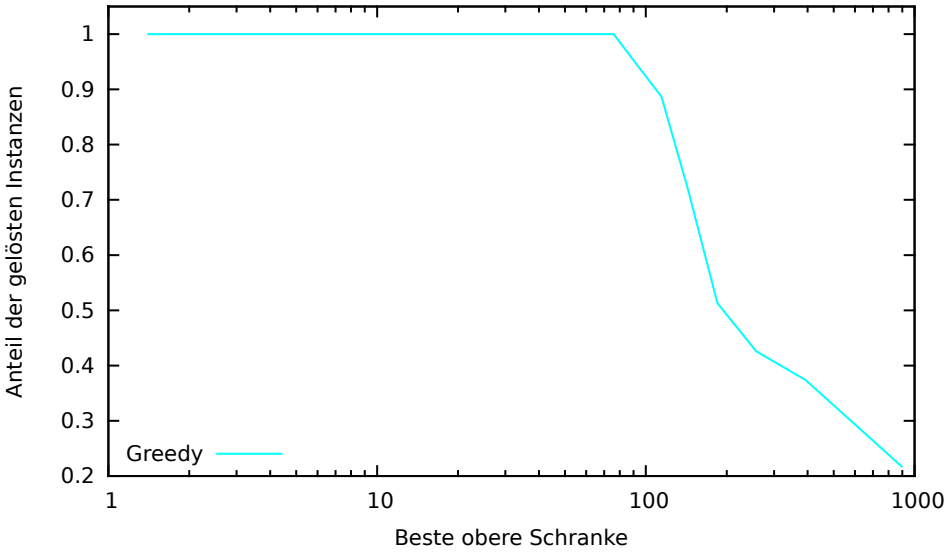

Abb. 5.6: Anteil der vom Greedy-Algorithmus gelöster Instanzen in Abhängigkeit von der besten oberen Schranke

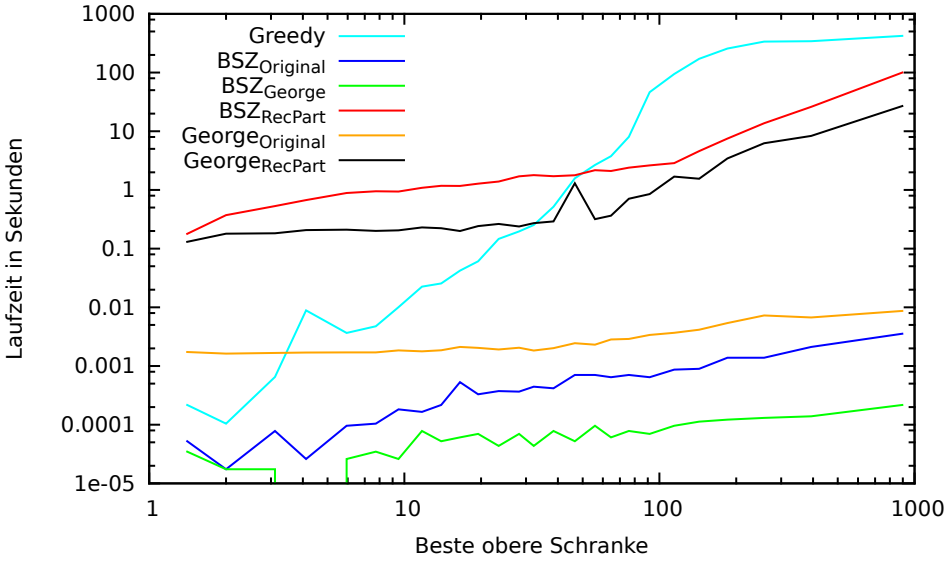

Abb. 5.7: Übersicht über die Laufzeiten der getesteten Methoden

6 Fazit

Aus den verschiedenen getesteten Methoden haben sich im Verlauf dieser Arbeit zwei besonders hervorgehoben. Dabei handelt es sich ein mal um die Schichtenheuristik von George kombiniert mit der rekursiven Aufteilungsheuristik von Birgin et al. zum Lösen der zweidimensionalen Unterprobleme. Dieses Verfahren konnte in über 67% der Fälle eine optimale Lösung finden und packte im Schnitt über 98% der optimalen Packmenge. Dabei ist zu beachten, dass für über ein Viertel der Testinstanzen die optimale Lösung nicht bekannt ist, und die obere Schranke für die Bewertung der Qualität verwendet wurde. Aus diesem Grund stellen die gefundenen Werte gewissermaßen eine untere Schranke für die Qualität der Heuristik für diese spezielle Testmenge dar.

Bei der zweiten Methode handelt es sich um die Beste-Schicht-Zuerst-Heuristik, die Aufgrund der niedrigen Komplexität sehr schnell ist. In zeitkritischen Problemstellungen ist diese Methode den anderen vorzuziehen, auch wenn sie etwas schlechtere Ergebnisse liefert als der eben erwähnte $\mathsf{George}_{\mathsf{RecPart}}$-Algorithmus. In Abbildung 6.1 werden diese

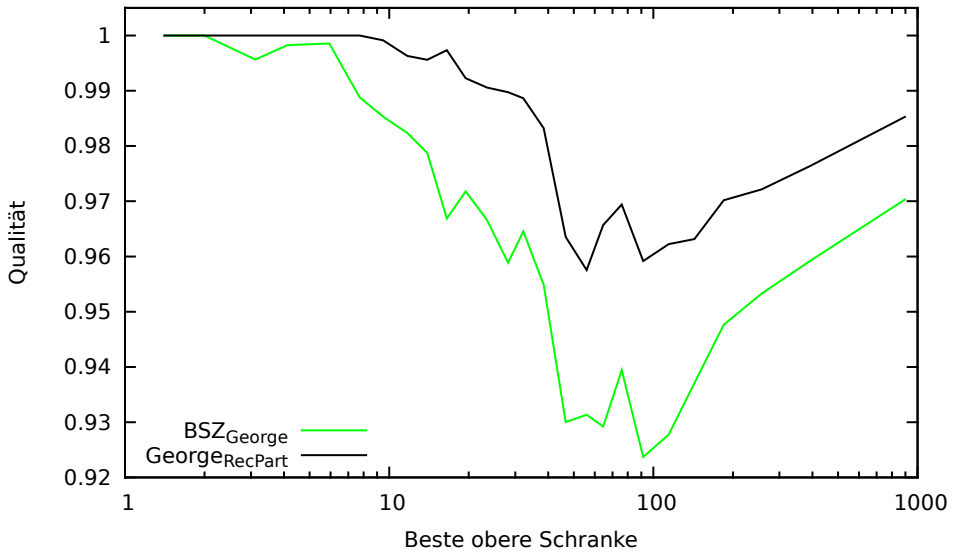

Abb. 6.1: Vergleich zwischen dem schnellsten und dem effektivsten Packalgorithmus

beiden Methoden miteinander verglichen. Trotz der allgemein schlechteren Lösungen hat der $\mathsf{BSZ}_{\mathsf{George}}$-Algorithmus dennoch selbst im schlechtesten Datensatz im Schnitt über 92% der optimalen Packmenge erreicht und ist somit zweifelsohne praxistauglich.

Literaturverzeichnis

[Bar79] F. W. Barnes: Packing the maximum number of $m \times n$ tiles in a large $p \times q$ rectangle. *Discrete Mathematics*, 26(2):93–100, 1979.

[BLM09] Ernesto G. Birgin, Rafael D. Lobato und Reinaldo Morabito: An effective recursive partitioning approach for the packing of identical rectangles in a rectangle. *J. Oper. Res. Soc.*, 61(2):306–320, 2009.

[Dow85] Kathryn A. Dowsland: Determining an upper bound for a class of rectangular packing problems. *Comput. Oper. Res.*, 12(3):201–205, 1985.

[Dow87] Kathryn A. Dowsland: An exact algorithm for the pallet loading problem. *Europ. J. Oper. Res.*, 31(1):78–84, 1987.

[Dyc90] Harald Dyckhoff: A typology of cutting and packing problems. *Europ. J. Oper. Res.*, 44(2):145–159, 1990.

[Geo92] John A. George: A Method for Solving Container Packing for a Single Size of Box. *J. Oper. Res. Soc.*, 43(4):307–312, 1992.

[GG61] Paul C. Gilmore und Ralph E. Gomory: A Linear Programming Approach to the Cutting-Stock Problem. *Oper. Res.*, 9(6):849–859, 1961.

[Kar84] Narendra Karmarkar: A new polynomial-time algorithm for linear programming. In: *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, STOC '84, Seiten 302–311. ACM, 1984.

[Kha79] Leonid G. Khachiyan: A polynomial Algorithm in Linear Programming, 1979.

[LLM99] Lauro Lins, Sostenes Lins und Reinaldo Morabito: A 9-fold Partition Heuristic for Packing Boxes into a Container. *Investigacion Operativa*, 7(3):69–82, 1999.

[LLM02] Lauro Lins, Sostenes Lins und Reinaldo Morabito: An n-tet graph approach for non-guillotine packings of n-dimensional boxes into an n-container. *Europ. J. Oper. Res.*, 141(2):421–439, 2002.

[LLM03] Lauro Lins, Sostenes Lins und Reinaldo Morabito: An L-approach for packing (l, w)-rectangles into rectangular and L-shaped pieces. *J. Oper. Res. Soc.*, 54(7):777–789, 2003.

[LMM02] Andrea Lodi, Silvano Martello und Michele Monaci: Two-dimensional packing problems: A survey. *Europ. J. Oper. Res.*, 141(2):241–252, 2002.

[LMV02] Andrea Lodi, Silvano Martello und Daniele Vigo: Heuristic algorithms for the three-dimensional bin packing problem. *Europ. J. Oper. Res.*, 141(2):410–420, 2002.

[MM98] Reinaldo Morabito und Silvia Regina Morales: A Simple and Effective Recursive Procedure for the Manufacturer's Pallet Loading Problem. *J. Oper. Res. Soc.*, 49(8):819–828, 1998.

[ST96] Guntram Scheithauer und Johannes Terno: The G4-Heuristic for the Pallet Loading Problem. *J. Oper. Res. Soc.*, 47(4):511–522, 1996.

[WHS07] Gerhard Wäscher, Heike Hausner und Holger Schumann: An improved typology of cutting and packing problems. *Europ. J. Oper. Res.*, 183(3):1109–1130, 2007.

Die vorliegende Bachelorarbeit mit dem Titel „Packalgorithmen für quaderförmige Objekte“ enthält vertrauliche Informationen der ZF Friedrichshafen AG. Veröffentlichungen, Vervielfältigungen bzw. eine Weitergabe – auch auszugsweise – sowie das Einstellen in eine öffentliche Bibliothek sind ohne vorherige schriftliche Zustimmung der ZF Friedrichshafen AG nicht gestattet. Ausgenommen von diesem Zustimmungserfordernis ist die Verwendung, die aufgrund einer Prüfungs- oder Studienordnung zu wissenschaftlichen Zwecken erfolgt.

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen Hilfsmittel und Quellen als die angegebenen benutzt habe. Weiterhin versichere ich, die Arbeit weder bisher noch gleichzeitig einer anderen Prüfungsbehörde vorgelegt zu haben.

Ort, Datum	(Ilia Belozerov)