technische universität
dortmund

Entwicklung und Analyse eines Verfahrens zur effizienten Lösung des Tourenplanungs- und Laderaumoptimierungsproblems

Adalbert Dawid

Algorithm Engineering Report
TR08-1-007
Dez. 2008
ISSN 1864-4503

Fakultät für Informatik
Algorithm Engineering (LS 11)
44221 Dortmund / Germany
http://ls11-www.cs.uni-dortmund.de/

TECHNISCHE UNIVERSITÄT DORTMUND
FAKULTÄT FÜR INFORMATIK

Diplomarbeit

Entwicklung und Analyse eines Verfahrens zur effizienten Lösung des Tourenplanungs- und Laderaumoptimierungsproblems

Adalbert Dawid
11. Februar 2008

INTERNE BERICHTE
INTERNAL REPORTS

Diplomarbeit an der
Fakultät für Informatik
der Technischen Universität Dortmund

Erstgutachter : Prof. Dr. Petra Mutzel
Technische Universität Dortmund
Fakultät für Informatik
Lehrstuhl für Algorithm Engineering
D-44221 Dortmund

Zweitgutachter : Dr.-Ing. Giovanni Prestifilippo
Fraunhofer-Institut für Materialfluss und Logistik
Abteilung Verkehrslogistik
D-44227 Dortmund

Hiermit versichere ich, dass ich meine Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Dortmund, den 11. Februar 2008

Inhaltsverzeichnis

Abbildungsverzeichnis

Algorithmenverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

Das kombinierte Tourenplanungs- und Laderaumoptimierungsproblem setzt sich im Wesentlichen zusammen aus den folgenden Teilproblemen:

- Tourenplanungsproblem
- Laderaumoptimierungsproblem

Das Tourenplanungsproblem (VRP, Vehicle Routing Problem) ist ein klassisches und ausgiebig untersuchtes Optimierungsproblem, das mit dem Traveling Salesman Problem (TSP) verwandt ist. Die Aufgabe besteht darin, eine Menge von Kunden von einem Depot aus mit Gütern zu versorgen, so dass der vorgegebene Bedarf jedes Kunden gedeckt wird. Dabei steht eine Anzahl – die unter Umständen beliebig groß sein kann – von identischen Transportfahrzeugen zur Verfügung, deren Transportkapazität beschränkt ist. Es muss also eine geeignete Partitionierung der Kunden sowie die Bildung von Touren für jede Partition erfolgen, so dass möglichst wenige Fahrzeuge/Touren eingesetzt werden und die Gesamtlänge der Touren minimiert wird. Die Lösung muss alle gegebenen Restriktionen, wie eine maximale Tourlänge und Gewichtsbeschränkungen der Fahrzeuge, berücksichtigen.

Die Erweiterung des VRP, das VRPTW (Vehicle Routing Problem with Time Windows), beinhaltet als zusätzliche Restriktion sogenannte Zeitfenster für alle Kunden, innerhalb derer diese beliefert werden müssen. Das zweite oben genannte Kriterium der Zielfunktion wird hier üblicherweise geändert, so dass statt der gesamten Tourlänge die gesamte Tourdauer minimiert werden soll. Auch das VRPTW – sowie seine zahlreichen Variationen – ist mittlerweile sehr gut untersucht und findet insbesondere im logistischen Umfeld Anwendung. Sowohl das VRP als auch das VRPTW sind NP-harte Probleme. Je nach Problemgröße und Laufzeitanforderung werden unterschiedliche Lösungsansätze angewandt, angefangen bei schnellen Konstruktions- und Verbesserungsheuristiken, über Metaheuristiken und evolutionäre Verfahren bis hin zu exakten Methoden.

Bedarf für gute Tourenplanungsverfahren besteht insbesondere in der Transportlogistik. In der Disposition von Transportunternehmen müssen auf ein- oder mehrtägiger Basis Tourenpläne berechnet und umgesetzt werden, wobei es stets darum geht, die vorhandenen Ressourcen (Anzahl eingesetzter Fahrzeuge und Fahrer, Arbeitszeit, Kraftstoff etc.)

so einzusetzten, dass die Gesamtkosten minimiert werden. Dabei ist es häufig eine Anforderung, dass Ergebnisse der Berechnungen innerhalb kurzer Zeit vorliegen müssen. In der strategischen Planung, hingegen, werden Tourenplanungsalgorithmen zur Unterstützung langfristiger Entscheidungen eingesetzt. Es wird versucht, möglichst präzise Vorhersagen über die Zukunft zu treffen, um die zu erwarteten Kosten abzuschätzen und eine Vorstellung über die benötigten Arbeitsmittel zu erhalten.

Üblicherweise wird in Tourenplanungsalgorithmen die Kapazitätsbeschränkung der Transportfahrzeuge dadurch abgebildet, dass im Zuge der Berechnung von Touren eine Überprüfung auf Überschreitung des maximal zulässigen Transportgewichts (oder auch des Transportvolumens) des eingesetzten Fahrzeugs stattfindet. Diese Methode ist zwar sehr schnell, jedoch hat das so entstehende Modell in vielen Fällen nur eine unzureichende Genauigkeit und ist dadurch für eine logistische Planung ungeeignet. Dies trifft beispielsweise dann zu, wenn die transportierten Güter Gegenstände sind, die auf bestimmte Weise im Frachtraum untergebracht werden müssen und das Gewicht der Waren nur eine untergeordnete Rolle spielt. In solchen Fällen muss der Tourenplanungsalgorithmus zusätzlich die geometrische Form der zu transportierenden Güter kennen, um für jede Tour eine zulässige Anordnung der Waren im Stauraum des eingesetzten Transportfahrzeugs bestimmen und ausweisen zu können.

Die transportierten Gegenstände sind in den meisten Problemstellungen dieser Art als rechtwinklige, zwei- oder dreidimensionale Objekte beschrieben. Als Beispiel für den zweidimensionalen Fall seien Transporte von Küchenmöbeln genannt, bei denen die Möbelstücke nicht übereinander gestapelt werden dürfen und es deshalb genügt, eine passende Anordnung und Orientierung der Möbelstücke auf dem Boden des LKW-Laderaums zu finden. Ein weiteres Beispiel für die Betrachtung der transportierten Waren als zweidimensionale Objekte ist die Beförderung von Paletten, die nicht gestapelt werden dürfen.

Das Ziel dieser Diplomarbeit ist es, ein Verfahren zur Lösung des Tourenplanungs- und zweidimensionalen Laderaumoptimierungsproblems (Vehicle Routing with Time Windows and Container Loading Problem, VRTWLP) zu konstruieren, so dass

- das Tourenplanungsverfahren zufriedenstellende Ergebnisse liefert,
- eine korrekte räumliche Anordnung der transportierten Objekte in den Containern gewährleistet und ausgewiesen wird und
- weitere aus der Logistik stammende Anforderungen erfüllt werden.

Zu diesen Anforderungen gehören beispielsweise die Drehbarkeit der zu verstauenden Objekte, so wie die Einhaltung der sogenannten LIFO-Reihenfolge bei der Bepackung des Containers.

Diese Arbeit gliedert sich wie folgt: In Kapitel 2 wird eine Einführung in die in dieser Arbeit untersuchte Problemstellung gegeben sowie ein konkreter Anwendungsfall aus der logistischen Praxis vorgestellt. Kapitel 3 beschäftigt sich mit dem Tourenplanungsproblem. Hier werden die in dieser Arbeit implementierten Tourenplanungsalgorithmen beschrieben. Kapitel 4 ist dem Problem der Laderaumoptimierung gewidmet. Es beinhaltet die Beschreibung mehrerer Algorithmen, die im Rahmen dieser Arbeit implementiert und untersucht wurden. In Kapitel 5 werden Methoden beleuchtet, mit denen die entwickelten Algorithmen miteinander verknüpft werden können, um eine Lösung für das in

dieser Arbeit zu untersuchende Gesamtproblem der kombinierten Tourenplanung und Laderaumoptimierung zu liefern. In Kapitel 6 werden experimentelle Analysen beschrieben, die durchgeführt wurden, um die implementierten Ansätze sowohl miteinander zu vergleichen als auch bestehenden Ansätzen aus der Literatur gegenüberzustellen. In Kapitel 7 wird schließlich ein Fazit gezogen, sowie ein Ausblick auf mögliche Verbesserungen der implementierten Methoden gegeben.

Kapitel 2

Einführung in die Problemstellung

Das folgende Kapitel dient zur Einführung in das Problem der kombinierten Tourenplanung und Laderaumoptimierung. Dieses komplexe Problem kommt in diversen Variationen im Transportwesen vor. Hier geht es darum, die Betriebskosten durch Reduktion der Anzahl gefahrener Touren sowie der dafür benötigten Fahrstrecke und Arbeitszeit zu minimieren. Eine zusätzliche Schwierigkeit zum Finden optimaler Tourenpläne besteht darin, eine zulässige räumliche Anordnung der transportierten Güter in den Frachträumen der Transportfahrzeuge zu bestimmen und auszuweisen.

2.1 Logistischer Hintergrund

Wie bereits in der Einleitung erwähnt, besteht insbesondere in der Transportlogistik Bedarf nach Tourenplanungsalgorithmen, die zusätzlich die Stauraumbeladung berücksichtigen. Zum einen werden solche kombinierten Algorithmen als Bestandteil strategischer Planung und Analyse eingesetzt, falls das klassische Modell der Tourenplanung als zu ungenau angesehen wird. Ein weiteres Einsatzgebiet für Algorithmen zur Lösung des kombinierten Tourenplanungs- und Laderaumoptimierungsproblems stellt die Disposition von Transportfahrzeugen bei Speditionen, Paket- und Expressdienstleistern, sowie in sonstigen Unternehmen mit einem eigenen Fuhrpark dar. Hier müssen häufig auf täglicher Basis Tourenpläne erstellt werden, wobei kurzfristige Änderungen der Auftragslage (sogenannte „Schnellschüsse“) häufig eher die Regel als die Ausnahme darstellen und binnen kurzer Zeit im Tourenplan Berücksichtigung finden müssen.

Warentransporte können einen nicht unerheblichen Anteil der Gesamtkosten eines Unternehmens ausmachen. Empirische Untersuchungen haben ergeben, dass je nach Wirtschaftszweig die Distributionskosten einen Anteil von 5,0% bis 22,8% der Gesamtkosten eines Unternehmens verursachen können. Dem steht ein enormes Einsparpotential gegenüber, das auf bis zu 40% geschätzt wird (siehe „Konzepte in der Logistik“ (A. Vastag) in [Buchholz, Clausen, und Vastag 1998]). Deshalb ist es ein ständiges Bestreben der Transportunternehmen, ihre logistischen Strukturen immer wieder zu überprüfen und wenn möglich zu konsolidieren und zu optimieren. Die Transportstruktur hat zudem großen Einfluss auf andere Kostenfaktoren, wie die Größe des zu unterhaltenden Fuhrparks, Materialdurchlaufzeiten und die notwendigen Kapazitäten von Depots.

Insbesondere in der Großserienfertigung (bspw. in der Automobilindustrie) wird häufig das Prinzip der „fertigungssynchronen Beschaffung“ eingesetzt. Hierbei wird das Material erst zu Terminen angeliefert, die durch den Produktionsablauf bestimmt werden. Auf diese Weise kann durch geringe Lagerbestände eine Reduktion von Materialdurchlaufzeiten sowie Lagerhaltungs- und Kapitalbindungskosten erzielt werden. Die Kehrseite ist, dass Lieferverzögerungen einen Ausfall oder zumindest eine Verzögerung der Produktion nach sich ziehen können, weshalb eine sehr enge Zusammenarbeit mit den Lieferanten und eine hohe Zuverlässigkeit letzterer erforderlich sind (vgl. „Konzepte in der Logistik“ (A. Vastag) in [Buchholz, Clausen, und Vastag 1998]).

Aus den oben genannten Gründen spielt in der Beschaffungslogistik – aber auch in der Distribution – für die Planung der Transporte die zeitliche Komponente eine große Rolle. Deshalb gehört zu den Anforderungen an Tourenplanungsverfahren die Berücksichtigung von Zeifenstern der Lieferanten bzw. Abnehmer von Gütern. Ein Zeitfenster ist ein festgelegtes Zeitintervall, innerhalb dessen ein Beschaffungs- oder Lieferauftrag abgewickelt werden muss. Genauer gesagt muss die Abwicklung der Beladung oder der Entladung der Güter eines Kunden innerhalb des vorgegebenen Zeitfensters beginnen, damit der Zeitplan einer Tour eingehalten und Verzögerungen bei Anlieferung oder Abholung von Waren vermieden werden.

Bei der Kalkulation von Tourenplänen ist es zudem notwendig, die zeitliche Dauer der Be- oder Entladung von Gütern (im Folgenden „Servicezeit“) zu berücksichtigen, da diese durchaus einen großen Anteil an der gesamten Tourdauer haben können. Um die Servicezeit zu minimieren, müssen Packstücke eines zu beliefernden Kunden üblicherweise so im Frachtraum des Transportfahrzeugs angeordnet sein, dass sie vor Ort direkt zugreifbar sind und nicht von Packstücken solcher Kunden blockiert werden, die erst zu einem späteren Zeitpunkt beliefert werden sollen. Da Transportfahrzeuge häufig nur durch die hintere Ladeöffnung be- und entladen werden können, müssen die Waren bereits zu Beginn der Tour in einer Anordnung im Frachtraum vorliegen, in der sie sich später nicht gegenseitig blockieren. Langwierige Umordnungen beim Kunden sind zu vermeiden, da dies einen unter Umständen großen Zeitverlust sowie zusätzliche technische Schwierigkeiten mit sich bringen kann. Der Disponent – bzw. das seine Arbeit unterstützende Computerprogramm – muss also die Anordnung der Packstücke im Frachtraum von der Reihenfolge der Stopps einer Tour abhängig machen. Im Falle der Distribution müssen sich die an den ersten Kunden der Tour auszuliefernde Gegenstände ganz hinten im Laderaum befinden, um sofort zugreifbar zu sein, dann die des zweiten Kunden der Tour usw. Man spricht in diesem Zusammenhang von der LIFO-Reihenfolge[1] der Bepackung, oder auch von sequentieller Bepackung.

Das kombinierte Problem der Tourenplanung und Stauraumbepackung hat eine enorm hohe Komplexität, da bereits das Tourenplanungsproblem NP-hart (siehe z.B. [Iori, Gonzalez, und Vigo 2007]) und das Laderaumoptimierungsproblem NP-vollständig ist (siehe [Garey und Johnson 1979]). Möglicherweise wurden aus diesem Grund bislang nur sehr wenige Ansätze zur Lösung des gesamten Problems in der Literatur vorgestellt (siehe Abschnitt 5.1). Die Motivation dieser Arbeit besteht deshalb darin, das aus wissenschaftlicher Sicht sehr junge Problem der Tourenplanung und Stauraumbepackung zu untersuchen und einen Algorithmus zu implementieren, der einerseits alle essenziellen, durch Anforderungen der Logistik vorgegebenen Nebenbedingungen berücksichtigt. Andererseits sollten

[1] **l**ast **i**n **f**irst **o**ut

auch große Probleminstanzen aus der Praxis in vernünftiger Laufzeit gelöst werden können.

2.1.1 Eine konkrete Aufgabenstellung

Die Idee für das Thema dieser Diplomarbeit sowie die Rahmenbedingungen der Problemstellung ergaben sich aus einem Projekt am Fraunhofer-Institut für Materialfluss und Logistik (IML) in Dortmund, welches im Auftrag des Verbandes der Automobilindustrie (VDA) durchgeführt wurde (siehe [Vastag, Presifilippo, und Schwarz 2005]). Zur Durchführung des Projekts erhielt das IML Lieferdaten eines deutschen Automobilunternehmens, die den Zeitraum eines repräsentativen Monats umfassen. Diese Sendungsdaten wurden als willkommene Hilfestellung bei der Konzeption, Entwicklung und für ausgiebige Tests der in dieser Diplomarbeit entwickelten Algorithmik verwendet.

Hintergrund

Die maximale Höhe von Fahrzeugen ist auf deutschen Straßen gesetzlich auf 4 Meter beschränkt. Diese Regelung ist in Europa jedoch nicht vereinheitlicht. So gibt es viele Länder, in denen diese maximale Höhe überschritten werden darf. Beispielsweise gilt in Finnland eine Höhenbeschränkung von $4,20$m, in Irland sind es $4,25$m und in Frankreich, Großbritannien sowie Schweden gibt es gar keine gesetztlich festgelegte Maximalhöhe. Der deutsche Gesetzgeber hat jahrelang die Vorschrift nicht konsequent Umgesetzt; Transporte, die mit zu hohen Fahrzeugen durchgeführt wurden, wurden nur mit geringen Verwarngeldern geahndet. Zudem wurde in vielen Fällen durch Sondergenehmigungen die Ausführung solcher Transporte genehmigt. So wurden mit der Zeit von den Unternehmen immer mehr Fahrzeuge mit einer Höhe von bis zu $4,20$m (sogenannte Volumenfahrzeuge) eingesetzt, da diese über eine Innenraumhöhe von ≥ 3m Verfügen, gegenüber einer Innenraumhöhe von etwa $2,80$m im Falle der bis zu 4m hohen LKW. Nun aber hat der Gesetzgeber angedroht, die Vorschrift der Höhenbeschränkung konsequent umzusetzen, so dass Volumenfahrzeuge auf deutschen Straßen nicht weiter hätten eingesetzt werden dürfen.

Die Umsetzung der Vorschrift hätte für deutsche Transportunternehmen sowie für Unternehmen anderer Branchen in vielerlei Hinsicht negative Konsequenzen gehabt. Wegen des Wegfalls an Innenraumvolumen wurde ein Anstieg der Anzahl notwendiger Transporte befürchtet, was sich in steigenden Kosten, einer Erhöhung der CO_2-Emissionen, sowie einem allgemeinen Wettbewerbsnachteil gegenüber Unternehmen in anderen europäischen Ländern ausgedrückt hätte. Besonders schmerzhaft wäre die Umsetzung der Vorschrift für die Automobilbranche gewesen, denn hier werden zu einem großen Teil genormte Behälter (sogenannte Gitterboxen) eingesetzt, die eine Höhe von $1,0$m oder $1,50$m haben und deshalb den Frachtraum von Volumenfahrzeugen optimal ausnutzen. Eine Senkung der Innenraumhöhe auf $2,80$m hätte deshalb bedeutet, dass in vielen Fällen eine gesamte Schicht solcher Gitterboxen nicht mehr hätte transportiert werden können.

Aufgabenstellung und Durchführung

Die Aufgabe der Fraunhofer IML bestand darin, eine möglichst genaue Schätzung des Zuwachses notwendiger Transporte und der damit verbundenen Kosten sowie CO_2-

Emissionen abzugeben für den Fall, dass Volumenfahrzeuge nicht weiter eingesetzt werden dürften.

Mangels einer fertigen Software zur Lösung des kombinierten Tourenplanungs- und Laderaumoptimierungsproblems wurde eine solche Software prototypisch implementiert. Mit dieser Implementierung konnten beiden Szenarien mit und ohne Volumenfahrzeuge simuliert und die entstehende Differenz bezüglich Kosten und CO_2-Emissionen ermittelt werden.

2.2 Problemformulierung

Die in dieser Arbeit zu lösende Aufgabe besteht darin, ein Optimierungsverfahren für das kombinierte Problem der Tourenplanung und Laderaumpotimierung zu finden. Das Tourenplanungsproblem (Vehicle Routing Problem, VRP) ist ein klassisches Optimierungsproblem, welches eine Generalisierung des bekannten Problems des Handlungsreisenden (Traveling Salesman Problem, TSP) darstellt. Dem klassischen TSP liegt ein Graph $G = (V, E)$ zugrunde, wobei $V = \{1, \ldots, n\}$ die Knotenmenge und $E = \{(i, j) \mid i, j \in V,\ i \neq j\}$ die Kantenmenge darstellt, die jedes Knotenpaar miteinender Verbindet. Mit jeder Kante $(i, j) \in E$ ist eine Distanz $d_{ij} \geq 0$ verknüpft, die zurückgelegt werden muss, um vom Start- zum Zielknoten zu gelangen. In der symmetrischen Variante des Problems ist der Graph G ungerichtet, d.h. es gilt $d_{ij} = d_{ji} \forall i, j \in V$. Die Aufgabe besteht nun darin, eine sogenannte Rundtour auf den Kanten aus E über die Knoten aus V zu finden, so dass die Summe der Distanzen aller Kanten minimal ist. Eine Rundtour ist nichts anderes als eine Permutation der Knotenmenge V, insbesondere gilt also, dass jeder Knoten genau einmal besucht werden muss.

Das VRP erweitert das mathematische Modell des TSP in mehreren Punkten. Gegeben ist hierbei ein Graph $G' = (V, A)$ mit Knotenmenge $V = \{0, \ldots, n\}$ und Kantenmenge $A = \{(i, j) \mid i, j \in V,\ i \neq j\}$. Der Knoten 0 repräsentiert ein Depot, die übrigen Knoten entsprechen Kundenstandorten. Jeder Kunde i hat einen Bedarf q_i, mit dem die Warenmenge beschrieben wird, die an diesen Kunden geliefert werden soll. Mit der Kantenmenge A assoziiert ist eine Entfernungsmatrix (d_{ij}), in der die Entfernungen zwischen Standorten aus V eingetragen sind. Man beachte, dass der Graph G' gerichtet ist und die Entfernungen und Fahrzeiten zwischen Knotenpaaren somit unsymmetrisch sein können, es gilt also im Allgemeinen $d_{ij} \neq d_{ji}$. Dies ermöglicht es, Entfernungen eines realen Straßennetzes abzubilden, in welchem die Distanzen zwischen Knotenpaaren in Hin- und Rückrichtung verschieden sein können. Am Depot befindet sich eine Flotte von m identischen Fahrzeugen (wobei m zu Anfang unbekannt und eine Entscheidungsvariable sein kann), die jeweils eine maximale Transportkapazität Q haben. Die Aufgabe besteht darin, eine Menge von höchstens m Touren zu bilden, die

1. jeweils am Depot beginnen und auch dort enden, so dass
2. jeder Kunde genau ein Mal von genau einem Fahrzeug besucht wird,
3. die gesamte Transportmenge jeder Tour den Wert Q nicht überschreitet,
4. die Gesamtlänge jeder Tour eine vorgegebene Länge D nicht überschreitet und dabei

5. die insgesamt gefahrenen Kilometer minimiert werden.

Es existieren diverse Variationen dieser Problemstellung, und auch im Rahmen dieser Arbeit wurde nicht die klassische Problemformulierung betrachtet. Eine in dieser Arbeit berücksichtigte Ergänzung zum VRP stellen Zeitfenster $[e_i, l_i]\,,\ i \in V \setminus \{0\}$ der Kunden dar.[2] Diese erfordern die zusätzliche Berücksichtigung einer Zeitmatrix (t_{ij}), in welcher die Fahrzeiten für alle Knotenpaare $i, j \in V$ eingetragen sind. Außerdem wird mit jedem Kunden $i \in V$ eine Servicezeit s_i verknüpft, die für das Be- oder Entladen von Waren beim Kunden i benötigt wird und die somit eine Standzeit für den LKW bedeutet. In einer zulässigen Lösung müssen alle Zeitfenster beachtet werden, so dass

1. ein LKW den Servicevorgang bei jedem Kunden i innerhalb des gegebenen Zeitfensters $[e_i, l_i]$ beginnt,

2. der LKW erst nach Beginn e_0 des Depotzeitfensters aufbricht und bis zum Ende l_0 dieses Zeitfensters zum Depot zurückkehrt.

Falls ein LKW bereits vor dem Anfang e_i des Zeitfensters beim Kunden eintrifft, muss er bis zum Zeitpunkt e_i warten, bis er damit beginnen kann, die Ware zu entladen. Das Ende der Servicezeit kann durchaus nach dem Ende l_i des Zeitfensters liegen. Wenn das Fahrzeug aber erst nach Ende des Zeitfensters beim Kunden eintrifft, ist die Lösung ungültig. Das Tourenplanungsproblem mit Zeitfenstern wird abgekürzt VRPTW genannt (Vehicle Routing Problem with Time Windows).

Im Kontext des VRPTW wird häufig nach einer anderen Zielfunktion optimiert als der oben genannten. Es kommen hierbei die folgenden Kriterien in Frage (vgl. z.B. „Vehicle Routing“ (G. Laporte) in [Dell'Amico, Maffioli, und Martello 1997]):

- Minimiere die Anzahl m benötigter Fahrzeuge.
- Minimiere die GesamtFahrtdauer.
- Minimiere die Gesamttourlänge.

In der Logistik ist es häufig von primärem Interesse, die Anzahl m benötigter Fahrzeuge und damit auch der Touren zu minimieren (vgl. „Touren- und Routenplanung“ (H.-W. Graf) in [Buchholz, Clausen, und Vastag 1998]). Die Minimierung der gefahrenen Strecke oder der Fahrzeit ist dann ein untergeordnetes Ziel und wird als sekundäres Optimierungskriterium betrachtet. Deshalb wird in dieser Arbeit die folgende, hierarchische Zielfunktion verfolgt:

1. Minimiere die Anzahl m benötigter Fahrzeuge.

2. Minimiere die Gesamtlänge oder die Gesamtdauer aller Touren.

[2] e steht hier für *earliest*, l für *latest*

Dabei wird das erste Kriterium mit höherer Priorität verfolgt als das zweite. Die Frage, ob nach der Tourlänge oder der Tourdauer optimiert werden soll, hängt von der konkreten Aufgabenstellung ab. So wurde bei der Durchführung der in Abschnitt 6.2 beschriebenen Experimente die Tourlänge als zweites Optimierungskriterium eingesetzt, da die dort verwendeten Eingabedaten keine Zeitfenster und keine Informationen zu den zeitlichen Distanzen zwischen den Kunden beinhalten. Die in Abschnitt 6.3 beschriebenen Experimente auf den Real-World-Eingabedaten konnten hingegen unter Anwendung einer Zeitmatrix durchgeführt werden, weshalb hier die Fahrtdauer als zweites Kriterium eingesetzt wurde.

Der in Abschnitt 3.4 vorgestellte Tourenplanungsalgorithmus verfolgt stets das Ziel, eine minimale Anzahl von Touren zu generieren. Dabei kann durch geeignete Parametrisierung sein Verhalten so beeinflusst werden, dass die Touren entweder unter der Prämisse der Tourlängenminimierung oder der Fahrzeitminimierung erstellt werden.

Es ist zu beachten, dass sich die beiden oben genannten Optimierungskriterien 1 und 2 bei Betrachtung mehrerer Lösungen für eine Eingabeinstanz durchaus entgegengesetzt zueinander verhalten können. Lösungen mit geringer Tourenanzahl haben nicht immer eine geringe Gesamtlänge bzw. Gesamtdauer und umgekehrt.

Das bislang beschriebene Modell des Tourenplanungsproblems beinhaltet Gewichte von Kundenbedarfen und Fahrzeugkapazitäten. In der Realität (und insbesondere in der Transportlogistik) genügt dies jedoch nicht immer. Häufig kommt die Anforderung hinzu, bei der Berechnung des Tourenplans die Ausmaße der beförderten Waren mit zu berücksichtigen (vgl. „Tourenplanungssysteme“ (H.-W. Graf) in [Buchholz, Clausen, und Vastag 1998]). Als Beispiel wurden in der Einleitung bereits Möbelspeditionen und Transporte von Paletten genannt. Ein anderes Beispiel sind für die Automobilindustrie durchgeführte Transporte, bei denen die transportierten Güter in sogenannten Gitterboxen verstaut werden, bevor sie im LKW untergebracht und befördert werden. In diesen Beispielen existiert eine geometrische Beschreibung der zu transportierenden Güter, die dazu genutzt werden kann, eine Verteilung letzterer im Transportfahrzeug zu berechnen und auszuweisen. Die Feststellung, ob eine Menge von Gegenständen in einem Container untergebracht werden kann, wird allgemein als das Laderaumoptimierungsproblem oder auch kürzer als das Packproblem bezeichnet. Üblicherweise geht man davon aus, dass die zu transportierenden Gegenstände als dreidimensionale, rechtwinklige Objekte i mit den Ausmaßen (w_i, h_i, d_i) beschrieben sind und der Frachtraum Ausmaße (W, H, D) hat. In vielen Fällen lässt sich jedoch das Problem auf zwei Dimensionen reduzieren. Im Falle von Möbeltransporten ist es üblicherweise so, dass die Möbelstücke nicht übereinander gestapelt werden dürfen. Ähnlich verhält es sich beim Transport von Paletten. Die in der Automobilbranche verwendeten Gitterboxen können zwar durchaus übereinander gestapelt werden, aber nicht etwa versetzt, sondern nur zu wohldefinierten Stapeln. Diese werden dann anschließend auf dem Boden des LKW-Frachtraums verteilt. Das Problem, festzustellen, ob eine Menge I von zweidimensionalen, rechteckigen Gegenständen oder auch „Items“ mit den Ausmaßen $(w_i, h_i),\ i \in I$ in einem rechteckigen Container mit den Ausmaßen $(W, H),\ W \geq w_i, H \geq h_i \forall i \in I$ untergebracht werden kann, wird als das zweidimensionale orthogonale Packproblem (2OPP) bezeichnet.

Das kombinierte Problem, einen kostenminimalen Tourenplan unter Einhaltung von Zeitfenstern zu ermitteln und dabei die Zulässigkeit der geometrischen Verteilung der transportierten Waren in den Fahrzeugen zu berücksichtigen, wird allgemein als VRTWLP (Vehicle Routing with Time Windows and Container Loading Problem) bezeichnet und

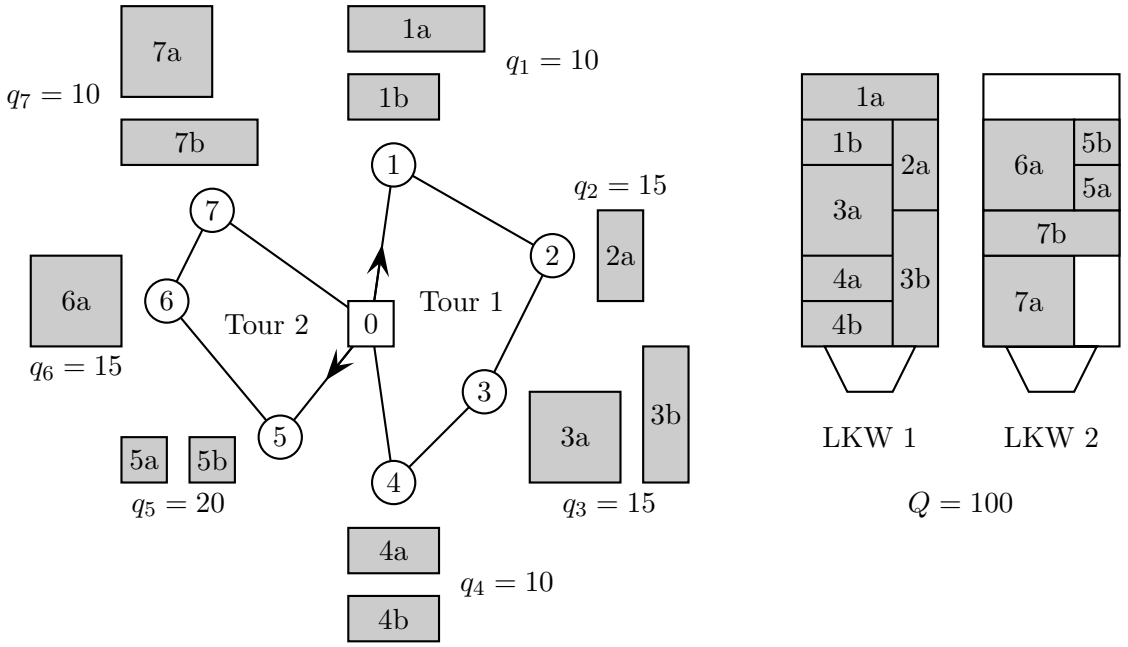


Abbildung 2.1: Beispielinstanz des kombinieren Tourenplanungs- und Laderaumoptimierungsproblems

ist Gegenstand dieser Arbeit. Das Problem soll an einem einfachen Beispiel verdeutlicht werden (siehe Abbildung 2.1). In diesem Beispiel sind das Depot 0 und Kunden $1, \ldots, 7$ gegeben. Jeder Kunde $i = 1, \ldots, 7$ hat einen Bedarf bestehend aus einer Menge $I(i)$ von zweidimensionalen Items. Zudem ist für den Bedarf jedes Kunden i das Gewicht q_i seiner Waren angegeben. Die Kunden werden von zwei Fahrzeugen mit der Gewichtskapazität $Q = 100$ und den Containerausmaßen $(W, H) = (3, 6)$ beliefert. Es ist offensichtlich, dass in diesem Beispiel die angegebene Lösung bezüglich der Anzahl eingesetzter Fahrzeuge optimal ist. Müssten hingegen die geometrischen Gegebenheiten der Fahrzeugbeladung nicht berücksichtigt werden, wäre eine Lösung mit nur einem Fahrzeug zulässig, denn die Gewichtsrestriktion wäre eingehalten: $\sum q_i = 95 \leq Q$.

Zwei weitere, häufig in der Praxis anzutreffende Anforderungen sind zu berücksichtigen: Zum einen muss bei der Beladung der Transportfahrzeuge häufig eine bestimmte Reihenfolge der Bepackung eingehalten werden. In diesem Zusammenhang spricht man von der einzuhaltenden LIFO-Reihenfolge der Bepackung (siehe Abschnitt 4.3). In einigen Fällen wäre es nicht möglich, in anderen würde es viel Zeit kosten, wenn das Entladen der Güter eines Kunden dadurch erschwert würde, dass Packstücke eines anderen Kunden die Packstücke des ersteren blockieren würden. Solche Situationen können vermieden werden, indem die Laderaumoptimierung beim Berechnen der Bepackug eines Fahrzeugs die Reihenfolge der Kunden auf der Tour von vornherein berücksichtigt. Damit die Packstücke sich nicht gegenseitig blockieren, müssen die Güter des letzten Kunden der Tour als erste gepackt werden und die des ersten zuletzt. Die Berücksichtigung der LIFO-Reihenfolge hat zum einen eine Auswirkung auf gewisse Details der Implementierung der Laderaumoptimierungsalgorithmen und muss zum anderen auch bei den Planungen der Kombination der Tourenplanungs- mit den Laderaumoptimierungsalgorithmen berücksichtigt werden.

Die zweite Anforderung an die Laderaumoptimierung besteht darin, optional die Drehbarkeit um 90° der zu packenden Gegenstände zu berücksichtigen. In einigen Fällen kann sehr viel Stauraum eingespart werden, wenn beide Orientierungen der Packstücke berücksichtigt werden. Andererseits erhöht dies die Komplexität und die Laufzeit der Algorithmen.

Das Ziel dieser Arbeit ist es, folgende Algorithmen zu implementieren:

- einen Tourenplanungsalgorithmus mit Berücksichtigung von Zeitfenstern,
- ein Verfahren zur Bidlung von Boxenstapeln für jeden Kunden,
- einen Algorithmus zur Lösung des zweidimensionalen orthogonalen Packproblems mit Einhaltung der LIFO-Reihenfolge und Drehbarkeit der Packstücke.

Diese Algorithmen sollen so miteinander kombiniert werden, dass ein möglichst effizientes Verfahren für das Gesamtproblem VRTWLP entsteht.

2.3 Auswahl von Algorithmen

Nachdem sowohl klassische als auch aktuelle Ansätze aus der Literatur zu den Themen Tourenplanung und Laderaumoptimierung, sowie zum kombinierten Problem weitestgehend in Erfahrung gebracht wurden und ihre Arbeitsweise sowie ihre Vor- und Nachteile bekannt waren, galt es, eine geeignete Auswahl an Algorithmen für diese beiden Problemteile zu treffen und diese Algorithmen anschließend möglichst geschickt miteinander zu verknüpfen.

An die auszuwählenden Verfahren wurden zunächst die folgenden Ansprüche gestellt:

- Die Algorithmen müssen miteinander kombinierbar sein.
- Die Algorithmen müssen – nachdem sie zu einer kombinierten Lösung verknüpft wurden – in der Lage sein, die gegebene Problemstellung zu lösen und sollten im Zusammenspiel über gute Laufzeiteigenschaften verfügen sowie Lösungen akzeptabler Qualität liefern.
- Die Algorithmen sollten für sich allein genommen über gute Laufzeiteigenschaften verfügen, ohne dabei die Lösungsqualität zu vernachlässigen.

Zu den Besonderheiten der gegebenen Problemstellung gehören insbesondere die folgenden Anforderungen:

- Die Boxen (oder Boxenstapel) im Frachtraum der eingesetzten LKW sollten optional um 90° gedreht werden können. In manchen Fällen kann so eine wesentlich bessere Auslastung der Fahrzeuge erzielt werden.
- Es muss die sogenannte LIFO-Reihenfolge der Bepackung eingehalten werden. Sie garantiert, dass die Entladevorgänge vereinfacht werden und ein Minimum an Servicezeit beanspruchen. Der Begriff LIFO-Reihenfolge wird in Abschnitt 4.3 definiert (Definition 4.3.1, Seite 43).

- Falls gegeben, müssen Zeitfenster der Kunden eingehalten werden. Ein Zeitfenster ist ein mit einem Kunden assoziiertes Zeitintervall, innerhalb dessen das Transportfahrzeug bei diesem Kunden ankommen muss.

Die erstgenannte Anforderung muss innerhalb des eingesetzten Laderaumoptimierungsalgorithmus implementiert werden, die letztgenannte liegt in der Domäne des Tourenplanungsalgorithmus. Die Einhaltung der LIFO-Reihenfolge erfordert hingegen die Kenntnis des Packalgorithmus über die Reihenfolge der Kunden auf einer Tour, es ist also ein Austausch von Informationen zwischen den beiden Algorithmenklassen notwendig.

Weitere allgemeine Kriterien für die Auswahl von in der Praxis einzusetzenden Algorithmen wurden von [Cordeau, Gendreau, Laporte, Potvin, und Semet 2002] genannt. Die Autoren beziehen sich dabei speziell auf Tourenplanungsheuristiken. Zu diesen Kriterien gehören

- Lösungsqualität (Accuracy),
- Ausführungsgeschwindigkeit (Speed),
- Einfachheit (Simplicity) und
- Flexibilität (Flexibility).

Mit Lösungsqualität meinen die Autoren zum einen den Abstand des berechneten Ergebnisses zu einem Optimum bezüglich der eingesetzten Zielfunktion (z.B. Anzahl Touren oder die Tourlänge). Da jedoch im Falle des VRP für viele Eingabeinstanzen eine optimale Lösung unbekannt ist, sollten die von einer Heuristik ausgewiesenen Ergebnisse mit den besten bekannten Ergebnissen verglichen werden. Zum anderen bezieht sich das Gütekriterium Lösungsqualität auf die Konsistenz der berechneten Ergebnisse über eine Vielzahl von Eingabeinstanzen hinweg.

Die benötigte Ausführungsgeschwindigkeit eines VRP-Algorithmus hängt stark von seinem Einsatzzweck ab. Als Beispiele für den Bedarf nach sehr kurzen Laufzeiten nennen die Autoren die Tourenplanung bei Express-Kurierdiensten oder für den Einsatz von Krankentransporten. Auf der anderen Seite können VRP-Algorithmen beim Einsatz in der strategischen Planung sehr lange Laufzeiten von mehreren Stunden oder gar Tagen eingeräumt werden.

Als ein weiteres wichtiges Kriterium für den Erfolg von Tourenplanungsalgorithmen in der Praxis sehen Cordeau et al. die Einfachheit der Verfahren an. Als Beispiel wird der Savings-Algorithmus für das VRP von [Clarke und Wright 1964] genannt, welcher sich nach wie vor einer hohen Popularität erfreut, obwohl mittlerweile sehr viel bessere Verfahren existieren. Als Grund nennen Cordeau et al. die leichte Verständlichkeit des Verfahrens, sowie die Tatsache, dass es in sich abgeschlossen ist und in seiner Grundvariante keine Parameter benötigt. Das Problem zu vieler Parameter benennen die Autoren als einen Grund für den mangelnden Erfolg einiger Heuristiken und Metaheuristiken. Als Ausweg schlagen sie vor, die Parameter auf bestimmte, erwartungsgemäß beste Werte festzulegen, oder aber ein Verfahren einzusetzen, das automatisiert nach einer guten Kombination der Parameter sucht.

Schließlich wird die Flexibilität von VRP-Heuristiken als ein wichtiges Gütekriterium genannt. Mit Flexibilität von VRP-Heuristiken ist die Möglichkeit gemaint, diese an neue Problemstellungen und Nebenbedingungen anpassen zu können, ohne dass dabei die Lösungsqualität und die Ausführungsgeschwindigkeit allzu sehr leidet.

2.3.1 Auswahl eines Tourenplanungsalgorithmus

Insebesondere die überdurchschnittliche Größe der in dieser Arbeit zugrundeliegenden Real-World-Eingabedaten (siehe Abschnitt 6.3), sowie die Tatsache, dass ein großer Teil der Laufzeit für die Berechnung der Laderaumbepackung reserviert werden muss, haben die Auswahl eines Tourenplanungsalgorithmus maßgeblich bestimmt. Es sollte ein Verfahren gefunden werden, das sehr große Eingabeinstanzen in möglichst kurzer Laufzeit lösen kann, ohne dabei die Lösungsqualität zu vernachlässigen. Vor dem Hintergrund, dass die integrierte Methode (siehe Abschnitt 5.2) eines der untersuchten Verfahren zur Kombination des Tourenplanungs- mit dem erwartungsgemäß laufzeitintensiven Laderaumoptimierungsalgorithmus sein sollte, ergab sich die Anforderung an das gewählte Tourenplanungsverfahren, möglichst wenige Aufrufe des Moduls zur Stauraumbepackung auszuführen.

Einen guten Kompromiss zwischen hoher Ausführungsgeschwindigkeit und angemessener Lösungsqualität bieten insbesondere Konstruktionsheuristiken für das VRP und die zahlreichen Varianten des Problems. Eine Übersicht über Konstruktionsheuristiken für Tourenplanungsprobleme sowie der Versuch eines Leistungsvergleichs wird in [Bräysy und Gendreau 2005] geboten. Um eine gute Vergleichbarkeit sicherzustellen, wurden nur solche Verfahren in den Leistungsvergleich aufgenommen, für welche die Autoren der Algorithmen geeignete Benchmarks auf Grundlage der Eingabeinstanzen von [Solomon 1987] unter Angabe der eingesetzten Hardware, der Laufzeiten und der Lösungen durchgeführt und dokumentiert hatten. Zu den schnellsten Verfahren in diesem Vergleich gehört der Algorithmus *I1* von [Solomon 1987]. Aus diesem Grund wurde er im Rahmen dieser Arbeit implementiert. Weitere Vorzüge von *I1* bestehen in der relativen Einfachheit (und damit guter Implementierbarkeit) verglichen mit anderen Heuristiken verbunden mit großer Flexibilität des Verfahrens, da die Arbeitsweise des Algorithmus mit Hilfe mehrerer Parameter je nach Problemstellung angepasst werden kann. Zudem berücksichtigt *I1* explizit möglicherweise gegebene Zeitfenster von Kunden, was seinen Einsatz in vielen logistischen Aufgabenstellungen erst ermöglicht. Ein Nachteil des Verfahrens gegenüber neueren Konstruktions- und insbesondere gegenüber Verbesserungsheuristiken liegt sicherlich in der leicht reduzierten Lösungsqualität. So generiert *I1* gemäß des von Bräysy und Gendreau (2005) durgeführten Algorithmenvergleichs Lösungen, die im Durchschnitt etwa 10% mehr Touren aufweisen als das beste untersuchte Verfahren (eine Metaheuristik), welches aber wiederum wesentlich längere Laufzeiten beansprucht.

Ein weiteres Kriterium für die Auswahl des Algorithmus zur Tourenplanung kristallisierte sich erst während der Bearbeitung dieser Arbeit heraus: Auf der Suche nach einem effizienten Verfahren, die Algorithmen zur Tourenplanung und Laderaumoptimierung miteinander zu verknüpfen, entstand die Idee, die spezielle Arbeitsweise von *I1* zu nutzen. Die so entstandene sequentielle Methode (siehe Abschnitt 5.4) basiert auf der Eigenschaft des Tourenplanungsalgorithmus, Touren sequentiell zu konstruieren und wäre weder mit parallelen Konstruktionsverfahren noch mit Verbesserungsverfahren in dieser Form möglich gewesen.

Die Arbeitsweise von *I1* wird in Abschnitt 3.4 beschrieben.

2.3.2 Auswahl von Verfahren zur Laderaumoptimierung

Das durch die vorliegenden Eingabedaten bestimmte Laderaumoptimierungsproblem ist zunächst dreidimensional, da alle zu transportierenden Objekte rechtwinklige, dreidimensionale Gitterboxen sind. Zur Berechnung der Verteilung dreidimensionaler Objekte in einem Containerinnenraum ist es im Allgemeinen notwendig, das dreidimensionale orthogonale Packproblem (3OP) zu lösen, das sehr schwierig ist und üblicherweise nur für kleine Eingabeinstanzen exakt gelöst werden kann (siehe [Martello, Pisinger, und Vigo 2000]). Eine Reduktion der Problemkomplexität ergibt sich durch die Tatsache, dass im Falle der vorliegenden Eingabedaten die Boxen nur in Form von wohldefinierten Stapeln transportiert werden dürfen. Somit kann die Lösung des dreidimensionalen Laderaumoptimierungsproblems in zwei Phasen unterteilt werden. Hierbei werden die Gitterboxen zunächst je Kundenauftrag zu Stapeln zusammengefasst und diese Stapel anschließend im Containerinnenraum passend verteilt.

Für die Stapelbildung genügt es, einen Algorithmus für das eindimensionale Bin-Packing-Problem (BPP) einzusetzen, wobei die Frachtraumhöhe als Kapazität der Bins und die Höhe der einzelnen Boxen als Gewicht der zu verteilenden Items interpretiert werden können. In dieser Arbeit wurde zu diesem Zweck die First-Fit-Decreasing-Heuristik (FFD) für das BPP implementiert, die trotz ihrer hohen Ausführungsgeschwindigkeit bewiesenermaßen Ergebnisse nahe am Optimum liefert. Das Verfahren wird in Abschnitt 4.2 beschrieben.

Das Verteilen von Boxenstapeln auf der Grundfläche des Containers gehört zur Klasse der sogenannten Cutting-&-Packing-Probleme. Hierzu werden alle Probleme gezählt, die mit der effizienten Verteilung von Objekten in einer, zwei oder drei Dimensionen bzw. mit der Aufteilung oder dem Zuschneiden von Flächen oder Räumen zu tun haben. Mehrdimensionale Bin-Packing-Probleme gehören ebenso zur Klasse der Cutting-&-Packing-Probleme wie das mehrdimensionale Knapsack- und Strip-Packing-Problem. Eine erste allgemeine Klassifizierung dieser Problemklasse geht auf [Dyckhoff 1990] zurück; eine detailliertere Klassifizierung der verschiedenen Varianten des zweidimensionalen Bin-Packing-Problems haben [Lodi, Martello, und Vigo 1999] durchgeführt.

Das in dieser Arbeit zu lösende zweidimensionale Laderaumoptimierungsproblem stellt eine Entscheidungsvariante des zweidimensionalen Bin-Packing-Problems (2BPP) dar, bei der die Anzahl zur Verfügung stehender Bins auf 1 beschränkt ist. Das so entstehende Problem wird in der Literatur als zweidimensionales orthogonales Packproblem (2OPP) bezeichnet (siehe z.B. [Clautiaux, Carlier, und Moukrim 2007]). Die Boxenstapel sollen optional um 90° drehbar sein und eine Lösung muss nicht unbedingt durch eine Reihe von sogenannten „Guillotine-Schnitten“[3] erzielbar sein. [Lodi, Martello, und Vigo 1999] schlagen für das 2BPP mit freier Orientierung der Items und ohne den Bedarf an Guillotine-Schnitten eine heuristische Methode namens *Touching-Perimeter* vor, die nach Aussage der Autoren gegenüber anderen untersuchten Heuristiken die besten Ergebnisse in dieser Problemklasse erzielt hat. Dieses Verfahren wurde im Rahmen dieser Arbeit

[3] Die praktische Umsetzung von mehrdimensionalen Packproblemen erfordert es häufig, dass die Lösung durch eine Folge von kantenparallelen Schnitten (Guillotine-Schnitten) erzielt werden kann. Guillotime-Schnitte können z.B. dann erforderlich sein, wenn es sich um das Ausschneiden von rechteckigen Stücken aus Stoffrollen, Metall- oder Holzplatten handelt.

implementiert und durch geringfügige Modifikation an das zu Lösende 2OPP angepasst. Eine Beschreibung des modifizierten Verfahrens TP_{MOD} befindet sich in Abschnitt 4.4.

Um zu untersuchen, inwiefern bessere Verfahren für das 2OPP die Lösungsqualität des gesamten VRTWLP positiv beeinflussen können, wurden zudem zwei exakte Verfahren für die Laderaumoptimierung entwickelt. Das in Abschnitt 4.5 vorgestellte Verfahren MV_{MOD} stellt eine Modifikation des von [Martello und Vigo 1998] vorgestellten, mittlerweile als klassisch bezeichneten, enumerativen Verfahrens für das zweidimensionale Bin-Packing-Problem dar. Eine Beschleunigung dieses Verfahrens versprach das von [Clautiaux, Carlier, und Moukrim 2007] vorgeschlagene Verfahren *LMAO* zu sein, welches in modifizierter Form als $LMAO_{MOD}$ implementiert und in Abschnitt 4.6 beschrieben wurde. Um ein Ausufern der in der Eingabegröße exponentiellen Laufzeiten der exakten Verfahren zu verhindern, können diese nach Ablauf einer vorgegebenen Laufzeit mit negativem Ergebnis abgebrochen werden. Ein experimenteller Vergleich der beiden exakten Verfahren für das 2OPP auf Grundlage von Eingabeinstanzen aus der Literatur wurde durchgeführt. Ergebnisse des Benchmarks werden in Abschnitt 6.1 vorgestellt.

Ein Aufruf eines der potentiell sehr laufzeitintensiven, exakten 2OPP-Verfahren kann in einigen Fällen vermieden werden, wenn im Vorfeld gezeigt werden kann, dass für die gegebene Eingabeinstanz keine Lösung existiert. Ein solcher Beweis kann durch Berechnung einer unteren Schranke für das 2BPP geführt werden, wenn diese Schranke für die Zahl benötigter Container einen Wert größer eins liefert. Neben der trivialen „Continuous Lower Bound“ (L_0) existieren ausgereiftere Ansätze zur Erkennung solcher nicht lösbarer Instanzen. Im Rahmen dieser Arbeit wurden die Schranken L_0 bis L_4 aus [Martello und Vigo 1998] implementiert, die in Abschnitt 4.7 vorgestellt werden.

Schließlich wird in Abschnitt 4.8 das kombinierte Verfahren *Staged-Packer* für das 2OPP vorgestellt. Dieses verwendet die Continuous Lower Bound, die Schranke L_4 und die Heuristik TP_{MOD}, um, wenn möglich, einen teuren Aufruf eines der exakten 2OPP-Verfahren zu vermeiden.

2.3.3 Kombination der Verfahren für das VRTWLP

Neben der Auswahl geeigneter Algorithmen für die beiden Teilprobleme des VRTWLP ist es notwendig, die gewählten und implementierten Verfahren geschickt miteinander zu verknüpfen. In dieser Arbeit wurden drei unterschiedliche Möglichkeiten getestet, dies zu bewerkstelligen. Zusätzlich zu der integrierten und der isolierten Methode, die beide recht naheliegend erscheinen, wurde unter Ausnutzung gewisser Eigenschaften des Tourenplanungsalgorithmus *I1* die sequentielle Methode entwickelt, welche die Nachteile der beiden zuvor genannten Methoden weitestgehend eiliminieren soll.

Die drei implementierten Methoden werden in Kapitel 5 beschrieben.

Kapitel 3

Das Tourenplanungsproblem mit Zeitfenstern

Das Tourenplanungsproblem (Vehicle Routing Problem, VRP) ist ein klassisches und umfassend untersuchtes Optimierungsproblem und wurde ursprünglich von [Dantzig und Ramser 1959] formuliert. Das Problem spielt eine wesentliche Rolle im Transportwesen, in dem es ein wichtiges Ziel ist, durch Optimierung der Tourenpläne die Anzahl benötigter Fahrzeuge bzw. Fahrer und die Länge wie auch Dauer von Touren zu minimieren und so die Betriebskosten zu senken. Eine Definition der Problemstellung wurde bereits in Abschnitt 2.2 gegeben.

Es sind in der Vergangenheit zahlreiche Lösungsansätze für das VRP und das VRPTW (Vehicle Routing Problem with Time Windows) veröffentlicht worden. Die Bandbreite erstreckt sich von Konstruktionsheuristiken über Verbesserungsverfahren bis hin zu exakten Methoden. Abschnitt 3.1 stellt eine Übersicht der in der Vergangenheit vorgeschlagenen Methoden zur Lösung des VRP und einiger seiner Varianten dar.

Beim VRTWP ist es notwendig, neben der Berechnung von Touren und dem Vergleich ihrer Längen auch die Zulässigkeit der Touren bezüglich gegebener Kundenzeitfenster zu überprüfen, sowie ihre Mindestdauer zu kennen. In Abschnitt 3.2 wird ein Verfahren beschrieben, welches diese Aufgabe effizient erfüllt.

In Abschnitt 3.3 wird ein Grundgerüst für Tourkonstruktionsverfahren beschrieben, auf dessen Grundlage bereits mehrere Algorithmen entwickelt wurden. Eine dieser Implementierungen ist der Algorithmus *I1*, der eines der schnellsten heuristischen Verfahren zur Lösung des VRPTW darstellt. *I1* wurde im Rahmen dieser Arbeit implementiert und wird in Abschnitt 3.4 vorgestellt. Die Beschreibung bezieht sich zunächst auf die grundlegende Variante des Tourenplanungsproblems und berücksichtigt dabei nicht die geometrischen Gegebenheiten der transportierten Güter.

Schließlich wird in Abschnitt 3.5 das Verbesserungsverfahren *2-Opt* vorgestellt, welches zur Reduktion der Länge der von *I1* berechneten Touren eingesetzt wird.

3.1 Bestehende Ansätze zur Lösung des VRP

Der folgende Abschnitt bietet – ohne Anspruch auf Vollständigkeit – eine Übersicht existierender Verfahren für das Tourenplanungsproblem.

3.1.1 Konstruktionsheuristiken

Konstruktionsheuristiken werden insbesondere dann eingesetzt, wenn die Eingabeinstanzen relativ groß sind und die Laufzeiten eine kritische Rolle spielen. Sie haben jedoch den Nachteil, im Allgemeinen keine sehr guten Lösungen zu produzieren, obwohl ihre Güte im Laufe der Zeit durch den Einsatz ausgereifter Techniken deutlich zugenommen hat.

Im Fall des VRP(TW) starten Konstruktionsheuristiken entweder mit einer leeren oder einer sehr einfachen Lösung, um dann iterativ einzelne Kunden zum bestehenden Tourenplan hizuzufügen. Die Auswahl des oder der jeweils nächsten Kunden erfolgt in der Regel basierend auf einem festgelegten Auswahlkriterium. Dabei unterscheidet man zwischen sequentiellen und parallelen Konstruktionsverfahren. Sequentielle Verfahren verlängern eine „offene“ Tour so lange wie es die Nebenbedingungen erlauben, um sie dann als „geschlossen“ zu markieren und eine neue Tour zu öffnen. Parallele Verfahren, hingegen, bauen mehrere Touren gleichzeitig auf, d.h. es sind stets mehrere Touren als offen gekennzeichnet.

Eines der einfachsten Konstruktionsverfahren für das VRP ist die *Nearest-Neighbor* Heuristik. Das Verfahren beginnt mit einer Stichtour[1] zu einem beliebigen Kunden. Im Verlauf wird iterativ derjenige Kunde, zu dem die Entfernung am geringsten ist, an das Ende der aktuell offenen Tour gehängt. Das geschieht so lange, bis die Tour aufgrund der Verletzung von Restriktionen nicht mehr erweitert werden kann.

In [Solomon 1987] werden für eine Reihe von VRP-Heuristiken Erweiterungen um Zeitfensterrestriktionen vorgeschlagen und es werden anschließend diese neuen Verfahren miteinander verglichen. Zu den von Solomon verglichenen Verfahren gehören angepasste Versionen der klassischen Algorithmen Savings [Clarke und Wright 1964], Sweep [Gillett und Miller 1974] sowie eine um Zeitfenster bereicherte Version der Nearest-Neighbor Heuristik. Zudem schlägt er drei neue, auf Zeitfenstern basierende, Algorithmen auf Grundlage des Insertion-Frameworks vor, von denen *I1* auch noch viel später von Wissenschaftlern als Grundlage für weitere Entwicklungen verwendet wird. Eine weitere Errungenschaft von Solomon besteht darin, dass er in seiner Veröffentlichung für den Vergleich der VRPTW-Algorithmen eine Reihe von 56 Eingabeinstanzen vorstellt. Diese Instanzen werden noch bis heute als Standard-Testinstanzen für Benchmarks von VRPTW-Algorithmen verwendet.

[Bräysy und Gendreau 2005] stellen eine Reihe von Konstruktionsheuristiken vor und versuchen diese qualitativ miteinander zu vergleichen. Da ein fairer Vergleich von Algorithmen sehr schwierig ist, verwenden sie ein Pareto-Diagramm und tragen dabei die Kriterien Lösungsqualität und Laufzeit auf die beiden Achsen des Diagramms auf.[2] Zu den inter-

[1] Eine Stichtour ist eine Tour, die nur einen Kunden enthält.

[2] Es ist nicht immer klar, anhand welcher Kriterien Algorithmen miteinander verglichen werden sollten. In Frage kommen beispielsweise Lösungsqualität, Laufzeit, Flexibilität, Robustheit, Skalierbarkeit und Implementierbarkeit.

essantesten Algorithmen aus dieser Analyse zählen die Verfahren von [Russell 1995], [Antes und Derigs 1995] und [Ioannou, Kritikos, und Prastacos 2001]. Sie basieren alle auf Solomons Insertion-Framework und unterscheiden sich hauptsächlich in den Kriterien der Auswahl von Kunden, mit denen Touren initialisiert und erweitert werden. Die Verfahren von Russell und Antes et al. stellen parallele Tourkonstruktionsverfahren dar, während der Ansatz von Ioannou et al. ein sequentieller ist.

3.1.2 Verbesserungsverfahren

Die Funktionsweise von Verbesserungsverfahren ist es, zunächst eine gültige Lösung zu generieren (dies geschieht häufig mit Hilfe einer Konstruktionsheuristik), um diese dann schrittweise zu modifizieren. Die Modifikationen sind häufig randomisiert und erfolgen durch den Einsatz von vorher festgelegten Operatoren, die einen kleinen Teil der aktuellen Lösung verändern und hoffentlich zu einer Verbesserung der Gesamtlösung führen. Zur Überwindung lokaler Optima bedient man sich häufig Methoden der Diversifikation. Um hingegen die Nachbarschaft einer gegebenen Lösung nach Verbesserungen zu durchsuchen, wird das Mittel der Intensivierung eingesetzt.

Das Ziel von Verbesserungsverfahren ist es, ein möglichst breites Spektrum des Lösungsraums zu untersuchen und so viele verschiedene Lösungen zu testen. Dabei orientieren sich diese Verfahren mehr oder weniger stark an der Struktur des Lösungsraums und steuern so ihre weiteren Züge. Im Allgemeinen gilt, dass die Lösungsqualität von Verbesserungsverfahren stark von der ihnen eingeräumten Laufzeit und der Größe der Eingabeinstanzen abhängt.

Zu der Klasse der Verbesserungsverfahren gehören unter anderem populationsbasierte Ansätze (z.B. evolutionäre Algorithmen) sowie diverse Metaheuristiken (wie z.B. Tabu Search, Simulated Annealing, Ant Colonization Optimization und Variable Neighborhood Search). Einige sogenannte hybride Verfahren bedienen sich mehrerer unterschiedlicher Lösungstechniken und lassen sich somit nicht eindeutig in eine der oben genannten Kategorien einordnen.

Evolutionäre Algorithmen

[Homberger und Gehring 1999] stellen zwei (μ, λ)-Evolutionsstrategien zur Lösung des VRPTW vor. Fünf Jahre später veröffentlichen sie ein hybrides Verfahren, das zunächst eine Evolutionsstrategie zur Minimierung der Anzahl Touren verwendet, um anschließend mit Hilfe eines Tabu-Search-Algorithmus die Gesamttourlänge zu minimieren [Homberger und Gehring 2004]. Beide Verfahren stellen sich als sehr konkurrenzfähig heraus.

Tabu Search

Tabu Search (TS) gilt als eine der effizientesten Metaheuristiken zur Lösung des VRP und seiner zahlreichen Varianten. Beispielhaft seien an dieser Stelle die Algorithmen *Taburoute* von [Gendreau, Hertz, und Laporte 1994], *Unified Tabu Search* von [Cordeau, Laporte, und Mercier 2001] sowie das Verfahren von [Ho und Haugland 2004] erwähnt.

In *Taburoute* werden Nachbarschaften von Lösungen gebildet, indem einzelne, randomisiert gewählte Kunden einer Tour in eine benachbarte Tour bewegt werden, die dem zu bewegenden Kunden geometrisch nahe gelegene Kunden enthält. Einer solchen Umplatzierung eines Kunden folgt stets eine lokale Reoptimierung mittels des *GENI* Mechanismus für das TSP (siehe [Gendreau, Hertz, und Laporte 1992]).

Der Algorithmus *Unified Tabu Search* ähnelt in einigen Aspekten dem Ansatz von *Taburoute*, jedoch zeichnet er sich durch eine verbesserte Flexibilität und Einfachheit aus. Aus diesem Grund wurde er bereits im Kontext vieler verschiedener Varianten des VRP erfolgreich eingesetzt, ohne dass eine besondere Anpassung seiner Parameter notwendig gewesen wäre.

Der Tabu-Search-Algorithmus von Ho und Haugland (2004) wurde speziell für den Fall entwickelt, dass die Waren eines Kunden auf mehrere LKW aufgeteilt werden dürfen („split deliveries“). Dadurch können im Allgemeinen kürzere Touren erreicht werden, mit dem Nachteil, dass eine Tour in der Regel mehr Zwischenstopps enthält.

Eine gute Übersicht über den Forschungsstand bezüglich Tabu Search im VRP-Kontext bietet [Cordeau und Laporte 2002].

Simulated Annealing und Large Neighborhood Search

Weitere metaheuristische Verfahren, die zur Lösung des VRP eingesetzt wurden sind Simulated Annealing (SA) sowie Large Neighborhood Search (LNS). [Bent und Hentenryck 2001] stellen einen zweistufigen Ansatz vor, bestehend aus einem Simulated-Annealing-Algorithmus und einer nachgeschalteten Large-Neighborhood-Suche. In der ersten Phase (SA) wird eine Ausgangslösung erstellt, die aus möglichst wenigen Touren bestehen soll. Die zweite Phase (LNS) hat das Ziel, die bestehende Lösung so zu modifizieren, dass die Anzahl Touren nicht steigt und die Gesamttourlänge minimiert wird. Hierfür versucht der LNS-Algorithmus, Kunden möglichst günstig innerhalb der vorgegebenen Lösung neu zu positionieren.

[Bräysy 2003] schlägt eine deterministische Metaheuristik zur Lösung des VRPTW vor, die auf der Variable-Neighborhood-Search-Methode (VNS) von [Mladenovic und Hansen 1997] basiert und aus vier Phasen besteht. Zunächst wird mittels einer Konstruktionsheuristik eine Ausgangslösung generiert, die in der zweiten Phase einer Tour-Eliminierungs-Prozedur unterzogen wird, um die Anzahl Touren zu minimieren. Anschließend wird versucht, die Gesamttourlänge mit Hilfe von lokalen Suchverfahren zu verkleinern, um schließlich in der vierten Phase die bisherige Lösung durch geschickte Modifikationen der Zielfunktion zu verbessern und so aus einem möglichen lokalen Optimum zu entkommen.

[Pisinger und Ropke 2007] verwenden Adaptive Large Neighborhood Search (ALNS), eine Erweiterung von LNS um eine selbst anpassende Komponente, zur Lösung des VRP in zahlreichen Ausprägungen. Hierfür definieren sie zunächst ein umfassendes Modell, auf das sich mehrere VRP-Varianten abbilden lassen. Dann wird die in ihrem Modell abgebildete Probleminstanz mittels ALNS gelöst.

Ameisenalgorithmen

Als letzte Ausprägung der Verbesserungsverfahren sollen noch einige Ansätze vorgestellt werden, die auf dem Prinzip der Ant Colonization Optimization (ACO) beruhen. Die sogenannten Ameisenalgorithmen wurden ursprünglich von [Colorni, Doringo, und Maniezzo 1991] entwickelt und zum Lösen des Handlungsreisendenproblems (TSP) vorgeschlagen. Die Idee dieser Verfahren besteht darin, die kollektive Intelligenz einer Ameisenkolonie beim Auffinden kürzester Wege zu imitieren, indem auf den Kanten des zugrunde liegenden Netzwerkgraphen eine Pheromonspur in Form einer digitalen Markierung hinterlassen wird sobald diese Kanten in einer Iteration betreten wurden. In nachfolgenden Iterationen steigt dann die Wahrscheinlichkeit, dass markierte Wege wiederverwendet werden, mit der aktuellen Intensität ihrer Pheromonspur. Im Verlauf der Zeit verblasst die Spur jedoch und wird nur dann wieder aufgefrischt, wenn der jeweilige Weg neu begangen wird. Zudem erhalten kürzere Wege eine stärkere Pheromonspur als längere.

Die Anwendung der Ameisenalgorithmen auf das klassische VRP wurde zum ersten Mal in einem hybriden Ansatz von [Bullnheimer, Hartl, und Strauss 1998] vorgeschlagen. Touren werden in sequentieller Form auf die gleiche Weise konstruiert wie es bei der Anwendung der Ameisenalgorithmen auf das TSP der Fall ist. Sobald eine Tour nicht weiter verlängert werden kann, wird sie geschlossen und eine neue Tour geöffnet. Sobald eine komplette Lösung errechnet wurde, erfolgt eine lokale Optimierung aller Touren mit Hilfe des lokalen Suchverfahrens *2-Opt*.

[Gambardella, Taillard, und Agazzi 1999] wenden das ACO-Konzept auf das Tourenplanungsproblem mit Zeitfenstern (VRPTW) unter Verwendung einer hierarchischen Zielfunktion an. Sie lassen zwei unterschiedlich konfigurierte Ameisenkolonien parallel arbeiten, wobei die erste Kolonie die Anzahl eingesetzter Fahrzeuge und die zweite die Gesamttourlänge minimieren soll. Sobald die erstgenannte Kolonie eine verbesserte Lösung finden kann, wird diese Information an die zweitgenannte Kolonie weitergegeben. Dies geschieht so lange, bis ein vorgegebenes Abbruchkriterium erfüllt ist.

[Reimann, Doerner, und Hartl 2003] verwenden eine Kombination des ACO-Konzepts mit der Konstruktionsheuristik *I1* von [Solomon 1987] für das Lösen des VRP mit Zeitfenstern und Rücktransporten. Für die Auswahl von Kunden zur Tour-Konstruktion verwenden sie eine modifizierte Variante der Kostenfunktion aus *I1*, wobei weitere Komponenten hinzukommen, um Kunden mit Rücktransporten in die Touren einzufügen. Randomisierung der Zielfunktion sowie die Berücksichtigung von Pheromonspuren beim Einfügen von Kunden in Touren sind Konzepte aus der Welt der Ameisenalgorithmen, die hier Verwendung finden. Nach der Generierung einer Lösung wird diese durch einen lokalen Suchalgorithmus weiter verbessert.

3.2 Handhabung des zeitlichen Ablaufs von Touren

Bevor der im Rahmen dieser Arbeit implementierte Tourenplanungsalgorithmus erläutert wird, sollen einige Konzepte zum Umgang mit dem zeitlichen Ablauf einer Tour sowie Zeitfenstern erläutert werden, welche in [Solomon 1987] vorgestellt und in dieser Arbeit umgesetzt wurden.

Die Bestimmung der Länge einer Tour ist eine triviale Aufgabe. Gegeben sei eine Tour $r = (i_0, i_1, \ldots, i_k, i_{k+1}),\ i_0, \ldots i_{k+1} \in V$, wobei $i_1, \ldots, i_k$ die auf der Tour besuchten Kunden und $i_0 = i_{k+1} = 0$ das Depot darstellen, sowie eine Entfernungsmatrix (d_{ij}). Dann kann die Länge $D(r)$ der Tour durch Summierung der einzelnen Teilabschnitte emittelt werden:

$$D(r) = \sum_{p=0}^{k} d_{i_p i_{p+1}} \tag{3.1}$$

Als schwieriger stellen sich die Ermittlung der kleinstmöglichen Tourdauer $T(r)$ und die Überprüfung der zeitlichen Zulässigkeit einer Tour dar, wenn Kundenzeitfenster ins Spiel kommen. Solomon schlägt das folgende Modell vor, um diese Aufgaben effizient zu lösen.

Im Folgenden gehen wir davon aus, dass jedem Knoten $i \in V$ ein Zeitfenster $[e_i, l_i]$ sowie eine Servicezeit $s_i \geq 0$ zugeordnet sind. Des weiteren sei (t_{ij}) eine Zeitmatrix und b_i der Zeitpunkt, an dem der Service beim Kunden i beginnt. Für die zeitliche Zulässigkeit der Tour müssen die folgenden Bedingungen erfüllt sein:

$$e_0 \leq b_{i_1} - t_{0i_1} \tag{3.2}$$

$$b_k + s_k + t_{k,k+1} \leq l_0 \tag{3.3}$$

$$e_{i_p} \leq b_{i_p} \leq l_{i_p}\ \forall p \in \{1, \ldots, k\} \tag{3.4}$$

Bedingungen (3.2) und (3.3) gewährleisten, dass das Fahrzeug nach Beginn e_0 des Depotzeitfensters die Tour beginnt und vor seinem Ende l_0 wieder beendet. Bedingung (3.4) stellt sicher, dass alle Servicezeiten innerhalb der zugehörigen Kundenzeitfenster beginnen.

Aus Bedingung (3.4) folgt, dass ein Fahrzeug, das vom Kunden i_p zum Kunden i_{p+1} fährt und vor Beginn e_{p+1} seines Zeitfensters eintrifft, eine Wartezeit $w_{p+1} = e_{p+1} - b_{i_p} - s_{i_p} - t_{i_p i_{p+1}}$ einlegen muss.

Sei der Beginn der Servicezeit b_{i_p} beim Kunden i_p bekannt. Dann lässt sich der Beginn der Servicezeit $b_{i_{p+1}}$ beim nachfolgenden Kunden i_{p+1} wie folgt berechnen:

$$b_{i_{p+1}} = \max\left\{e_{i_{p+1}}, b_{i_p} + s_{i_p} + t_{i_p i_{p+1}}\right\}. \tag{3.5}$$

Wenn das Fahrzeug zum frühestmöglichen Zeitpukt – d.h. mit Beginn e_0 des Depotzeitfensters – abfährt, kommt es zum Zeitpunkt $e_0 + t_{0i_1}$ beim ersten Kunden an, muss aber mindestens bis zum Anbruch e_{i_1} des Kundenzeitfensters mit dem Service warten. Hieraus folgt der Beginn der Servicezeit für den ersten Kunden:

$$b_{i_1} = \max\left\{e_{i_1}, e_0 + t_{0i_1}\right\}. \tag{3.6}$$

Eine nach den Gleichungen (3.5) und (3.6) berechnete Liste $b = (b_{i_1}, \ldots, b_{i_k})$ von Service-Anfangszeiten repräsentiert eine zeitlich zulässige Tour, falls für alle Werte $b_p,\ p = 1, \ldots, k$

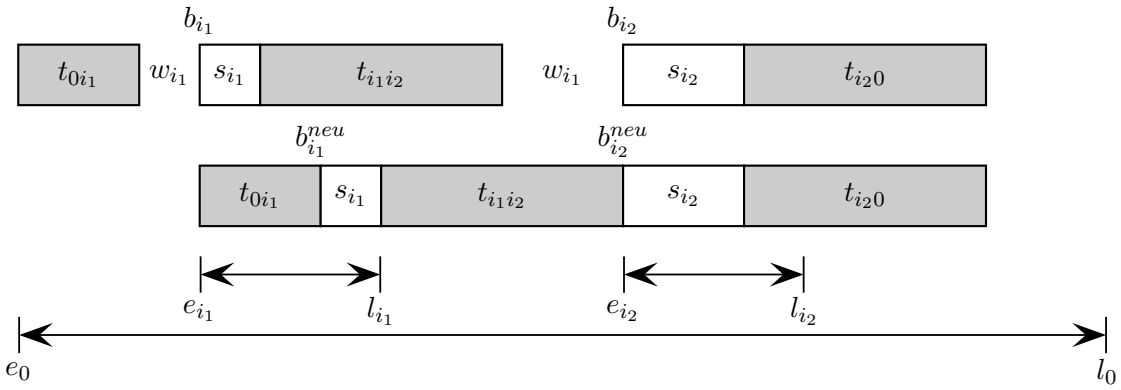


Abbildung 3.1: Zeitleiste Nach Erstellung (oben) und Kompaktierung (unten)

Bedingungen (3.2)–(3.4) erfüllt sind. Offenbar lässt sich also durch Berechnung von b die Zulässigkeit einer Tour in Zeit $O(k)$ feststellen, wenn die Tour k Kunden enthält.

In der wie oben dargestellten Tour werden alle Kundenbesuche so früh wie möglich ausgeführt, weshalb keine Verschiebungen nach links möglich sind. Diese Situation ist am Beispiel einer Tour mit zwei Kunden in Abbildung 3.1 (oben) dargestellt.

Um die minimale Länge einer Tour festzustellen, genügt es, alle Servicezeiten $b_{i_1}, \ldots, b_{i_{k-1}}$ bis auf die letzte so weit wie möglich nach rechts zu schieben und damit ihren Wert zu maximieren. Auf diese Weise erhält man eine Liste $b^{neu} = (b^{neu}_{i_1}, \ldots, b^{neu}_{i_k})$ von modifizierten Servicezeiten, so dass sich der gesuchte Wert als Differenz zwischen Ankunftszeit und Abfahrtzeit ergibt. b^{neu} kann wie folgt berechnet werden:

$$b^{neu}_{i_k} = b_{i_k} \tag{3.7}$$

$$b^{neu}_{i_p} = \min\left\{l_p,\ b_{i_{p+1}} - t_{i_p,i_{p+1}} - s_{i_p}\right\},\ p = 1, \ldots, k-1 \tag{3.8}$$

Das Ergebnis einer solchen „Kompaktierung“ des zeitlichen Tourverlaufs ist in Abbildung 3.1 (unten) beispielhaft dargestellt.

Schließlich lässt sich die Tourdauer leicht bestimmen:

$$T(r) = b^{neu}_{i_k} + s_{i_k} + t_{i_k0} - b^{neu}_{i_1} + t_{0i_1} \tag{3.9}$$

Eine Operation, die der in den folgenden Abschnitten vorgestellte Algorithmus *I1* sehr häufig ausführt, ist das Einfügen eines Kunden u in eine bestehende Tour r (siehe Abschnitt 3.4). Wir gehen davon aus, dass der nicht kompaktierte Vektor $b = (b_{i_1}, \ldots, b_{i_k})$ für r bekannt ist und $r' = (i_0, \ldots, i_{p-1}, u, i_p, \ldots, i_{k+1})$ die Tour nach der Einfügung von u zwischen die Positionen $p-1$ und p, $1 \leq p \leq k+1$ beschreibt. Unter anderem wird für die von *I1* durchgeführte Berechnung der Kostenfunktion c_1 eine durch die Einfügung verursachte Verschiebung der Servicezeit beim Kunden i_p benötigt. Gleichzeitig muss die Zulässigkeit der neu entstandenen Tour r' überprüft werden. Diese Aufgaben können mit dem hier vorgestellten Modell sehr effizient gelöst werden.

Für die Überprüfung der Zulässigkeit von r' muss der neue Vektor $b' = (b'_{i_1}, \ldots, b'_{i_{p-1}}, b_u, b'_{i_p}, \ldots, b'_{i_k})$ berechnet werden. Da alle Werte von b unter Berücksichtigung der Zeitfenster minimal sind, ändert sich offensichtlich der zeitliche Ablauf der Tour nicht für Kunden, die sich vor dem eingefügten Kunden in der Tour befinden. Es gilt also $b'_i = b_i \; \forall i \in \{i_1, \ldots, i_{p-1}\}$. Der Wert b_u lässt sich wie folgt berechnen:

$$b_u = \max\left\{e_u, b_{i_{p-1}} + s_{i_{p-1}} + t_{i_{p-1}u}\right\} \tag{3.10}$$

Entsprechend gilt für b'_{i_p}

$$b'_{i_p} = \max\left\{e_{i_p}, b_u + s_u + t_{ui_p}\right\} \tag{3.11}$$

und für die restlichen Werte $b'_{i_{p+1}}, \ldots, b'_{i_k}$

$$b'_{i_q} = \max\left\{e_{i_q}, b'_{i_{q-1}} + s_{i_{q-1}} + t_{i_{q-1}i_q}\right\}, \; q = p+1, \ldots, k \tag{3.12}$$

Die neue Tour r' ist zeitlich zulässig, falls b' die Zeitfensterrestriktionen (3.2)–(3.4) erfüllt. Die durch die Einfügung verursachte Verschiebung der Servicezeit beim Kunden i_p ergibt sich als $b'_{i_p} - b_{i_p}$. Die Berechnung und Überprüfung auf Zulässigkeit der neuen Servicezeiten kann offenbar in Zeit $O(k)$ durchgeführt werden, wenn k die Anzahl Kunden in einer Tour ist.

3.3 Das Insertion-Framework

Das Insertion-Framework von [Solomon 1987] ist ein simples und flexibles Grundgerüst für Tourenplanungsalgorithmen. Es werden in sequentieller Weise Touren erstellt, wobei zu jedem Zeitpunkt eine offene Tour $r = (i_0, i_1, \ldots, i_k, i_{k+1}), \; i_0, \ldots, i_{k+1} \in V, \; i_0 = i_{k+1} = 0$, die Menge R aller abgeschlossener Touren sowie die Menge U aller noch nicht in eine Tour eingefügten („ungerouteten“) Kunden existieren. Dabei sind einige Entscheidungskriterien nicht näher spezifiziert und bleiben somit implementierenden Algorithmen überlassen.

Das Verfahren generiert iterativ einzelne Touren. Im ersten Schritt einer Iteration wird ein sogenannter Seed-Kunde aus U extrahiert, mit dem eine neue offene Tour initialisiert wird. Anschließend fügt das Verfahren so lange einzelne Kunden aus U in die offene Tour ein bis die Tour aufgrund vorgegebener Restriktionen – wie beispielsweise Gewichts- oder Volumenrestriktionen des eingesetzten Fahrzeugs oder auch Längenrestriktionen der Tour – um keinen weiteren Kunden erweitert werden kann. Nun wird die Tour als geschlossen deklariert und zur Menge R hinzugefügt. Es beginnt eine neue Iteration mit dem Festlegen des nächsten Seed-Kunden.

Das Framework lässt folgende Entscheidungen offen:

- Welcher Kunde aus der Menge U wird nächster Seed-Kunde?
- Welcher Kunde aus U wird in die offene Tour eingefügt?
- An welcher Stelle wird dieser Kunde in die Tour eingefügt?

Für die Auswahl eines Seed-Kunden $u^* \in U$ schlägt Solomon drei mögliche Kriterien vor:

1. Es eröffnet derjenige Kunde eine neue Tour, der im gegebenen Iterationsschritt am weitesten vom Depot entfernt ist und für den damit gilt:

$$d_{0u^*} = \max \{d_{0u} : u \in U \text{ und zulässig}\} \tag{3.13}$$

2. Der Seed-Kunde wird nach dem Kriterium der frühesten Deadline ausgewählt, d.h. es muss gelten:

$$l_{u^*} = \min \{l_u : u \in U \text{ und zulässig}\} \tag{3.14}$$

3. Der Kunde mit minimaler, ungewichteter Summe aus Entfernung und Fahrzeit zum Depot und zurück wird Seed-Kunde. Es muss also gelten:

$$d_{0u^*} + d_{u^*0} + t_{0u^*} + t_{u^*0} = \min \{d_{0u} + d_{u0} + t_{0u} + t_{u0} : u \in U \text{ und zulässig}\} \tag{3.15}$$

.

Sei $r = (i_0, i_1, \ldots, i_k, i_{k+1}),\ i_0 = i_{k+1} = 0$ die aktuell offene Tour. Die Entscheidung, welcher Kunde aus U als nächstes in r eingefügt werden soll, wird durch die Berechnung einer Kostenfunktion in zwei Schritten getroffen. Zunächst werden für jeden Kunden u die Kosten c_1 einer Einfügung an allen möglichen Position in die Tour berechnet:

$$c_1(i(u), u, j(u)) = \min[c_1(i_{p-1}, u, i_p)],\ p = 1, \ldots, k+1. \tag{3.16}$$

Anschließend ist also für jeden Kunden bekannt, welche Position in der Tour für ihn bezüglich der Kostenfunktion c_1 am günstigsten ist. Der tatsächlich einzufügende Kunde u^* wird nun anhand einer weiteren Kostenfunktion c_2 bestimmt:

$$c_2(i(u^*), u^*, j(u^*)) = \text{optimum}[c_2(i(u), u, j(u))],\ u \in U \text{ und zulässig}. \tag{3.17}$$

Es sei darauf hingewiesen, dass die Funktionen c_1 und c_2 unterschiedlich sein können, jedoch an dieser Stelle nicht näher spezifiziert sind. Wie schon das Kriterium zur Auswahl des Seed-Kunden wird die Definition dieser Funktionen vielmehr dem das Framework implementierenden Algorithmus überlassen. Ebenfalls abhängig von der Implementierung und insbesondere von der Definition der Funktion c_2 ist die genaue Bedeutung der Funktion *optimum* in der letztgenannten Formel. Auch wenn Solomon in seinem Artikel auf diese Frage nicht weiter eingeht, wird in seinen vorgeschlagenen Implementierungen deutlich, um welche Funktion (*min* oder *max*) es sich in jeweils handeln muss.

[Solomon 1987] gibt die drei Referenzimplementierungen *I1*, *I2* und *I3* an, von denen sich in empirischen Untersuchungen *I1* als die beste erwiesen hat. Aus diesem Grund ist *I1* im Rahmen dieser Diplomarbeit implementiert worden.

3.4 Der Algorithmus I1

Der Tourenplanungsalgorithmus *I1* stellt eine Implementierung und eine Parametrisierung des Insertion-Frameworks von Solomon dar (siehe Abschnitt 3.3). Er definiert die Kriterien, nach denen ein Seed-Kunde zur Initialisierung einer neuen Tour ausgewählt wird, wie die

Auswahl von einzelnen Kunden stattfinden soll, die in eine offene Tour eingefügt werden, und an welche Stelle in der Tour die jeweiligen Kunden einzufügen sind.

Wie im vorherigen Abschnitt beschrieben, wird mit Hilfe der Kostenfunktion c_1 die beste Einfügeposition eines Kunden u in eine Tour r berechnet. Im Algorithmus *I1* ist diese Funktion als eine gewichtete Summe der beiden Funktionen c_{11} und c_{12} definiert mit

$$c_{11}(i,u,j) = d_{iu} + d_{uj} - \mu d_{ij},\ \mu \geq 0 \tag{3.18}$$

und

$$c_{12}(i,u,j) = b'_j - b_j \tag{3.19}$$

wobei b_j den Anfang der Entladezeit beim Kunden j bezeichnet und b'_j den Anfang der Entladezeit bei j, nachdem u zwischen i und j eingefügt wurde.

Der durch c_{11} berechnete Wert drückt also den Umweg aus, der gefahren werden muss, wenn das Transportfahrzeug auf dem Weg vom Kunden i zum Kunden j noch einen Zwischenstopp beim in die Tour eingefügten Kunden u einlegt, statt die direkte Verbindung von i nach j zu nehmen. Das Ergebnis von c_{12}, hingegen, beschreibt die Verschiebung der Entladezeit beim Kunden j nach hinten. Für eine Tour mit k Kunden können die Werte b_j und b'_j mit Hilfe der in Abschnitt 3.2 beschriebenen Methode zur Berechnung des zeitlichen Ablaufs einer Tour in Zeit $O(k)$ ermittelt werden.

Insgesamt ergibt sich die erste Kostenfunktion:

$$c_1(i,u,j) = \alpha c_{11}(i,u,j) + (1-\alpha)c_{12}(i,u,j),\ 0 \leq \alpha \leq 1. \tag{3.20}$$

Es werden durch c_1 sowohl der räumliche als auch der zeitliche Aspekt der Einfügung eines Kunden an eine Position in der offenen Tour berücksichtigt. Eine Gewichtung dieser beiden Kriterien kann durch die Variable α vorgenommen werden.

Die Funktion c_2 dient zur Festlegung des einzufügenden Kunden. Sie ist in *I1* als

$$c_2(i,u,j) = \lambda d_{0u} - c_1(i,u,j),\ \lambda \geq 0 \tag{3.21}$$

definiert. Aus dem negativen Vorzeichen des in dieser Gleichung auftauchenden Summanden c_1 lässt sich schließen, dass der Funktionswert von c_2 maximiert werden sollte.[3] c_2 besteht somit aus den beiden Komponenten $(-c_1)$ und der gewichteten Entfernung des einzufügenden Kunden zum Depot. Weiter vom Depot entfernte Kunden werden bevorzugt, weil sie potenziell schwieriger zu integrieren sind und deshalb in einem möglichst frühen Stadium des Algorithmus einer Tour zugewiesen werden sollten.

Das Verhalten von *I1* wird durch vier Parameter an mehreren Stellen beeinflusst:

- μ ist in der Kostenfunktion c_{11} enthalten (Gleichung (3.18)). Falls $\mu = 0$, wird der durch Einfügung von u zwischen i und j enstehende Zuwachs an Fahrstrecke absolut gewertet. Falls $\mu = 1$, wird dieser Zuwachs in Relation zu der Länge der direkten Verbindung zwischen i und j gesetzt.

- α ist Bestandteil der Funktion c_1 (Gleichung (3.20)) und gewichtet den Einfluss der beiden Summanden c_{11} und c_{12}.

[3] Dies ist im Artikel von Solomon nicht explizit beschrieben.

- λ ist in c_2 enthalten (Gleichung (3.21)). Mit steigendem Wert von λ werden Kunden bevorzugt, deren Entfernung zum Depot groß ist.
- Die Auswahl des Seed-Kunden kann, wie im letzten Abschnitt beschrieben, nach einem von mehreren verschiedenen Kriterien stattfinden. Für den Algorithmus *I1* empfiehlt Solomon die Kriterien 1 und 2.

Das so entstehende Quadrupel von Parametern entscheidet maßgeblich über die Arbeitsweise des Algorithmus. Da für ein gegebenes Problem nicht klar ist, welche Parametrisierung die besten Ergebnisse liefern wird, empfiehlt Solomon das Ausprobieren mehrerer vielversprechender Parameter-Kombinationen. Die von ihm in der empirischen Analyse eingesetzten Kombinationen für den Vektor $(\mu, \lambda, \alpha, \text{Seed-Kriterium})$ sind $(1,1,1,1)$, $(1,1,1,2)$, $(1,2,1,1)$, $(1,2,1,2)$, $(1,1,0,1)$, $(1,1,0,2)$, $(1,2,0,1)$ und $(1,2,0,2)$.

Algorithmus 1 stellt eine mögliche Implementierung von *I1* dar. Es soll zunächst der Fall des einfachen VRP (mit oder ohne Zeifenster) vorgestellt werden, in dem das maximale Transportgewicht w_{max} des Fahrzeugs sowie eine gegebenenfalls vorgeschriebene maximale Tourlänge l_{max} und Tourdauer t_{max} die einzigen einzuhaltenden Restriktionen sind. Für den Fall des kombinierten Problems (VRTWLP) müssen unter Umständen Modifikationen am Algorithmus vorgenommen werden. Diese werden in den Abschnitten 5.2, 5.3 und 5.4 gesondert behandelt.

Die äußere while-Schleife des Algorithmus (Zeilen 4-40) terminiert erst dann, wenn alle Kunden einer Tour zugeordnet worden sind. In jeder Iteration wird zunächst eine neue Tour initialisiert (Zeilen 5-15). Diese wird dann so lange erweitert (Zeilen 17-39), bis entweder keine Kunden mehr in U sind oder bis die Einfügung jedes der Kandidaten aus U eine der gegebene Restriktionen w_{max}, l_{max} oder t_{max} verletzen würde. Schließlich wird die Tour in die Menge R eingefügt (Zeile 40).

Die hier vorgestellte Implementierung ist speziell darauf ausgerichtet, die potentiell laufzeitintensive Überprüfung der Zulässigkeit einer Tour so selten wie möglich durchzuführen.[4] Deshalb werden zunächst bei gegebener Tour $r = (i_0, i_1, \ldots, i_k, i_{k+1}),\ i_0 = i_{k+1} = 0$ die Kosten c_2 für jede der möglichen Einfügekombinationen $(u, p),\ u \in U,\ p \in 1, \ldots k+1$ eines Kunden an eine Position p in die Tour berechnet (Zeile 22) und in der Menge I als Tripel (u, p, c) abgelegt. Dabei wird vorerst nicht die Zulässigkeit der Tour geprüft. Es folgt ein Durchlauf durch I in nicht-aufsteigender Reihenfolge der Kostenkomponente der Tripel (Zeilen 26-39), mit anderen Worten werden die günstigsten Einfügekombination zuerst betrachtet. Sei (u, p) die aktuelle Einfügekombination und r' die Tour, die aus der aktuellen Tour r entsteht, nachdem die Einfügung (u, p) durchgeführt wurde. In Zeilen 29 sowie 33 werden alle Restriktionen für r' überprüft. Die erste Einfügekombination (u, p), bei der alle Nebenbedingungen erfüllt bleiben, wird durchgeführt, d.h. Kunde u wird an Position p in die aktuelle Tour eingefügt und aus der Menge U der ungerouteten Kunden extrahiert (Zeilen 35 und 36).

Zwischen den Restriktionen Gewicht und Tourlänge/-dauer besteht ein wesentlicher Unterschied: Das Gesamtgewicht w_r der Waren aller Kunden einer Tour r ist unabhängig von der Reihenfolge der Kunden in dieser Tour, im Gegensatz zu den Größen Tourlänge $D(r)$ und Tourdauer $T(r)$. Diese Tatsache wird dazu genutzt, eine weitere Reduktion der Anzahl

[4] Dies trifft zwar im Falle des VRP(TW) nicht zu, wohl aber in der integrierten Methode für das VRTWLP, in der die Fahrzeugpebackung berechnet werden muss (siehe Abschnitt 5.2).

Algorithmus 1: I1

1: **procedure** I1$(U, w_{max}, l_{max}, t_{max})$
2: $R \leftarrow \emptyset$ ▷ Rückgabemenge aller Touren
3: $\bar{U} \leftarrow \emptyset$ ▷ Rückgabemenge aller nicht routbaren Kunden
4: **while** $U \neq \emptyset$ **do**
▷ Phase 1: Bestimme den Seed-Kunden u^*
5: $u^* \leftarrow$ **null**
6: **for all** $u \in U$ **do**
7: Berechne die Wertigkeit von u bzgl. des Seed-Kriteriums
8: **if** u bzgl. des Seed-Kriteriums besser als u^* **then**
9: $u^* \leftarrow u$
10: $U \leftarrow U \setminus \{u^*\}$ ▷ Entferne den Seed-Kunden aus U
11: $r \leftarrow (0, u^*, 0)$ ▷ Erstelle neue Tour mit u^* als einzigem Kunden
12: **if** r verletzt eine der Restriktionen w_{max}, l_{max}, t_{max} **then**
13: $\bar{U} \leftarrow \bar{U} \cup \{u^*\}$
14: **goto** 4
15: $w_r \leftarrow w_{u^*}$ ▷ Gewicht aller Güter auf der Tour
▷ Phase 2: Füge weitere Kunden in die Tour ein
16: $routeExtended \leftarrow$ wahr
17: **while** $U \neq \emptyset$ **and** $routeExtended$ **do**
18: $routeExtended \leftarrow$ falsch
19: $I \leftarrow \emptyset$ ▷ Liste von Tripeln (u, p, c)
20: **for all** $u \in U$ **do**
21: **for all** Einfügepositionen p in r **do**
22: $c \leftarrow c_2(r, u, p)$ ▷ Berechne Kostenfunktion c_2
23: $I \leftarrow I \cup (u, p, c)$
24: Sortiere die Tripel (u, p, c) in I nicht-aufsteigend nach den Kosten c
25: $T \leftarrow \emptyset$ ▷ Tabu-Liste von Kunden
26: **for all** $(u, p, c) \in I$ **do**
27: **if** $u \in T$ **then** ▷ Ist Kunde u tabu?
28: **goto** 26
29: **if** $w_r + w_u > w_{max}$ **then** ▷ Gewichtsrestriktion
30: $T \leftarrow T \cup \{u\}$
31: **goto** 26
32: Berechne Tourlänge $D(r')$ und Tourdauer $T(r')$ der Tour r', die durch Einfügung von u an p in r entstehen würde
33: **if** $D(r') > l_{max} \vee T(r') > t_{max}$ **then** ▷ Längenrestriktionen
34: **goto** 26
35: Füge u in r an Position p ein
36: $U \leftarrow U \setminus u$
37: $w_r \leftarrow w_r + w_u$
38: $routeExtended \leftarrow$ wahr
39: **goto** 17
40: Füge r zu R hinzu
41: **return** $(R, \bar{U})$

von Einfügekombinationen, deren Restriktionen geprüft werden müssen, zu erreichen. In der Menge I befinden sich bei gegebener Tour $r = (0, i_1, \ldots, i_k, 0)$ für jeden Kunden $k+1$ Tripel. Wenn aber für einen Kunden bereits bekannt ist, dass er die Gewichtsrestriktion verletzt, müssen die restlichen k Einfügekombinationen nicht mehr betrachtet werden. Um dies zu erreichen, wird eine Hashtabelle T mit Tabu-Kunden gepflegt, in die jeder Kunde eingetragen wird, der die Gewichtsrestriktion verletzt (Zeile 30). Ein solcher Kunde wird in Zukunft übersprungen (Zeile 27).

Satz 3.4.1. *Sei $n = |V \setminus \{0\}|$ die Anzahl der Kunden und l die Anzahl Kunden der längsten Tour. Dann ist Laufzeit des Algorithmus* I1 *durch $O(l^2 n^2)$ nach oben beschränkt.*

Beweis. In jeder Iteration der äußeren while-Schleife (Zeilen 4-40 in Algorithmus 1) wird mindestens ein Kunde aus der Menge U extrahiert. Die Festlegung eines Seed-Kunden und die Bildung einer initialen Tour (Zeilen 5-15) schlagen mit dem Faktor $O(n)$ zu Buche. Jeder Durchlauf (außer der jeweils letzte) der Schleife in den Zeilen 17-39 zieht die Extraktion eines Kunden aus U nach sich (Zeile 36), weshalb durch diese Schleife keine weiteren Kosten entstehen. Die for-Schleife in den Zeilen 20-23 iteriert über $O(n)$ Kunden; die darin eingebettete Schleife in den Zeilen 21-23 iteriert über $O(l)$ Positionen. Da die Kostenfunktion c_2 in Zeit $O(l)$ berechnet werden kann, ergibt sich für diesen Teil des Algorithmus die Laufzeitschranke $O(l^2 n^2)$.

Das gleiche gilt für den Code in Zeilen 26-39, denn es gibt höchstens $O(nl)$ Tripel (u, p, c) in I über die iteriert wird. Die teuersten Operationen im Rumpf der for-Schleife sind die Berechnungen der Tourlänge $D(r')$ und Tourdauer $T(r')$ der durch Einfügung des Kunden u an Position p neu entstehenden Tour r'. Diese Operationen werden in Zeit $O(l)$ durchgeführt. □

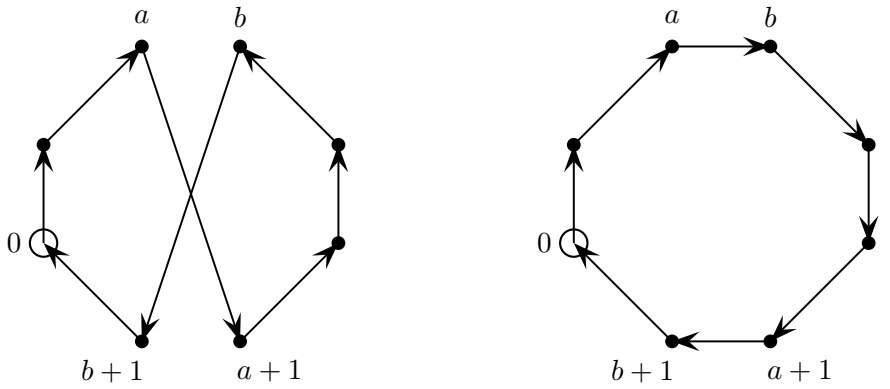
3.5 Reduktion der Tourlänge mittels 2-Opt

Eine Möglichkeit, die Tourlänge einer heuristisch berechneten Ausgangslösung f für das Handlungsreisendenproblem (TSP) oder das Tourenplanungsproblem (VRP) weiter zu verkürzen besteht darin, diese Lösung mit Hilfe eines Algorithmus zur lokalen Suche nachträglich zu verbessern. Verfahren zur lokalen Suche kennzeichnen sich dadurch, dass sie durch einzelne, zuvor definierte Regeln zur Veränderung einer Lösung f Verbesserungen dieser Lösung zu erzielen versuchen und zwar so lange, bis keine Verbesserung mehr möglich ist (dann ist ein *lokales Optimum* erreicht) oder bis eine obere Schranke an Iterationen oder Laufzeit überschritten wurde. Die Menge $N(f)$ von Lösungen, die sich durch einmalige Anwendung der zuvor festgelegten Operation auf f bilden lassen, wird die Nachbarschaft von f genannt. Im Kontext des VRP ist f eine Menge von Touren, die zuvor durch einen Tourenplanungsalgorithmus berechnet wurde.

Für das TSP wurde in [Croes 1958] das lokale Optimierungsverfahren *2-Opt* vorgeschlagen. Sei ein Graph $G = (V, E)$ gegeben mit der Knotenmenge $V = \{1, \ldots, n\}$ und der Kantenmenge $A = \{(i, j),\ i, j \in V\}$ sowie eine Entfernungsmatrix $(d_{ij}),\ i, j \in V$.

2-Opt versucht für eine gegebene Tour $r = (i_0, \ldots, i_k),\ i_0, \ldots, i_k \in V$ durch das Entfernen von zwei Kanten (i_a, i_{a+1}) und (i_b, i_{b+1}), $0 \leq a, b \leq k-1,\ a+1 < b$ der Tour r und das Einfügen der neuen Kanten (i_a, i_b) und (i_{a+1}, i_{b+1}), eine kürzere Tour r' zu bilden. Konkret entsteht bei gegebenen Indizes a und b aus der Tour $r = (i_0, \ldots, i_a, i_{a+1}, \ldots, i_b, i_{b+1}, \ldots, i_k)$

die neue Tour $r' = (i_0, \ldots, i_a, i_b, \ldots, i_{a+1}, i_{b+1}, \ldots, i_k)$. Abbildung 3.2 stellt das Prinzip des durch *2-Opt* vorgenommenen Kantenaustausches dar.



(a) Tour vor Anwendung eines *2-Opt* Kantenaustausches

(b) Tour nach dem Kantenaustausch

Abbildung 3.2: Prinzip der Arbeitsweise von 2-Opt

Man beachte, dass eine solche Austauschoperation die Umkehrung der Reihenfolge der Kunden $i_{a+1}, \ldots, i_b$ auf der Tour nach sich zieht. Dies kann, falls der Graph G gerichtet ist und somit die Entfernungen zwischen Knotenpaaren asymmetrisch sein können (d.h. wenn im Allgemeinen gilt $d_{ij} \neq d_{ji}$), die Konsequenz haben, dass sich die Länge des Teilstücks $i_{a+1}, \ldots, i_b$ der Tour verändern und insbesondere wachsen kann.

Zudem kann die neue Tour r' durch die Austauschoperation aus zwei Gründen unzulässig werden, nämlich

1. falls Zeitfenster der Kunden im Spiel sind und diese durch die neue Reihenfolge der Kunden auf der Tour nicht eingehalten werden können,
2. falls der Algorithmus im Kontext des VRTWLP mit LIFO-Reihenfolge der LKW-Bepackung ausgeführt wird und durch die neue Reihenfolge der Kunden die Bepackung nicht durchgeführt werden kann.

Es muss also nach jeder Modifikation einer Tour die Tourlänge und Tourdauer neu bestimmt werden. Zudem muss gegebenenfalls die Zulässigkeit der Tour bezüglich Zeitfensterrestriktionen überprüft und – im Falle geforderter LIFO-Reihenfolge – die Bepackung des LKW neu berechnet werden.

Nachdem ein Kantenaustausch vorgenommen wurde, kann die neu entstandene Tour r' auf weitere mögliche Verbesserungen hin untersucht werden. Dies geschieht so lange, bis keine weitere Verkürzung der Tour durch einen Austausch von zwei Kanten mehr möglich ist. In diesem Fall hat das Verfahren ein (lokales) Optimum gefunden und die resultierende Tour nennt sich *2-optimal*.

Eine 2-optimale Tour kann durch die Anwendung einer von zwei Strategien ermittelt werden:

1. *First Improvement*: Iteriere über die Kantenpaare der Tour und führe den ersten Austausch durch, der zu einer Verkürzung der Tour führt. Untersuche anschließend die neu entstandene Tour mittels *First Improvement*.

2. *Best Improvement*: Iteriere zunächst über alle Kantenpaare der Tour und identifiziere denjenigen Austausch, welcher zu der besten Lösung, d.h. zur maximalen Verkürzung der Tour führt. Führe anschließend diesen Austausch durch und untersuche die neu entstandene Tour mittels *Best Improvement*.

Man beachte, dass diese beiden Strategien zu unterschiedlichen Resultaten führen können, wenn auch beide stets eine 2-optimale Tour ausweisen. [Hansen und Mladenovic 2006] haben beide Strategien hinsichtlich Laufzeit und Lösungsqualität untersucht. Die Ergebnisse zeigen, dass *first improvement* insbesondere dann schneller ist und qualitativ bessere lokale Optima findet, wenn eine „schlechte“ (zufällig generierte) Ausgangslösung verbessert werden soll. Wenn aber die Ausgangslösung qualitativ „gut“ (durch ein heuristisches Verfahren generiert) ist, haben die Autoren Vorteile auf Seiten der Best-Improvement-Strategie festgestellt, sowohl hinsichtlich Laufzeit als auch Lösungsqualität.

Im Rahmen dieser Arbeit wurde die Best-Improvement-Strategie implementiert und zur Verbesserung der durch den Algorithmus *I1* generierten Touren eingesetzt. Algorithmus 2 stellt die Implementierung dar. Als Datenstruktur für eine Tour $r = (i_0, i_1, \ldots, i_k, i_{k+1}),\ i_0 = i_{k+1} = 0$ wird ein Array der Kunden $(i_1, \ldots, i_k)$ verwendet.

Algorithmus 2: 2-Opt

```
1:  procedure 2-Opt(r = (i_0, i_1, ..., i_k, i_{k+1}), t_max)
2:      r* ← r                                          ▷ Beste gefundene Tour
3:      for u ← 1 to k − 1 do
4:          for v ← u + 1 to k do
5:              for c ← 1 to u − 1 do                   ▷ Kopiere Anfangssequenz von r
6:                  j_c ← i_c
7:              for c ← v + 1 to k do                   ▷ Kopiere Endsequenz von r
8:                  j_c ← i_c
9:              idx ← 0
10:             for c ← u to v do                       ▷ Kopiere und invertiere Mittelteil von r
11:                 j_c ← i_{v−idx}
12:                 idx ← idx + 1
13:             r' ← (i_0, j_1, ..., j_k, i_{k+1})
14:             if r' zulässig bzgl. Zeitfensterrestriktionen und t_{r'} ≤ t_max then
15:                 if l_{r'} < l_{r*} then
16:                     r* ← r'                         ▷ Verbesserung gefunden
17:     if r* = r then                                  ▷ Keine Verbesserung gefunden
18:         return r
19:     else
20:         return 2-Opt(r*, t_max)
```

Die Parameter von Algorithmus 2 sind die zu verbessernde Tour r in Form des Kundenarrays $(i_1, \ldots, i_k)$ und des Depots 0 und die maximale Tourdauer t_{max}. Der Kantenaustausch wird dadurch vollzogen, dass zunächst ein Anfangsstück und ein Endstück des Arrays von Kunden in r in das neue Array $(j_1, \ldots, j_k)$ kopiert werden (Zeilen 5-6 und 7-8). Anschließend wird das Mittelstück des Arrays invertiert in das entsprechende Mittelstück des Zielarrays Kopiert (Zeilen 10-12). Auf diese Weise erhält man bei gegebenen Indizes u und $v,\ u < v$ das neue Array $(j_1, \ldots, j_k) = (i_1, \ldots, i_{u-1}, i_v, \ldots, i_u, i_{v+1}, \ldots, i_k)$, welches der Kundenreihenfolge in der Tour r entspricht, nachdem die Kanten (i_{u-1}, i_u) und (i_v, i_{v-1}) durch die Kanten (i_{u-1}, i_v) und (i_u, i_{v+1}) ausgetauscht wurden. Anschließend wird die so neu entstandene Tour r' hinsichtlich Zeitfensterrestriktionen und maximaler Tourdauer geprüft (Zeile 14). Falls diese eingehalten sind und die Tourlänge $l_{r'}$ kleiner als die Länge l_{r^*} der bis dahin kürzesten Tour ist, ist eine neue kürzeste Tour gefunden worden (Zeile 16). Falls innerhalb des aktuellen Durchlaufs eine verbesserte Tour r^* gefunden werden konnte, wird diese in einem rekursiven Aufruf weiter verbessert (Zeile 20).

Die Laufzeit eines Aufrufs des wie in Algorithmus 2 implementierten *2-Opt*-Verfahrens ist durch $O(n^3)$ beschränkt. Dies ergibt sich aus der Tatsache, dass $n(n-1)/2 = O(n^2)$ Kanten ausgetauscht werden und die Überprüfung der Restriktionen in Zeit $O(n)$ vollzogen wird (siehe Abschnitt 3.2). Es ist allerdings unklar, wie viele rekursive Aufrufe (= Tourverbesserungen) notwendig sind, bis ein lokales Optimum erreicht wird. [Lueker 1976] konnte zeigen, dass Eingabeinstanzen existieren, für die *2-Opt* nicht weniger als $\Theta(2^{n/2})$ Tourverbesserungen durchläuft, bevor es ein lokales Optimum findet. Theoretisch kann das Verfahren also eine exponentielle Laufzeit haben, in der Praxis erweist es sich jedoch als „schnell“. Die im Rahmen dieser Arbeit durchgeführten Experimente haben gezeigt, dass die Laufzeiten des Verfahrens vernachlässigbar gering sind.

Kapitel 4

Das Laderaumoptimierungsproblem

Wie in Kapitel 2 erwähnt wurde, kommen für das Laderaumoptimierungsproblem grundsätzlich zwei verschiedene Variationen und damit auch zwei Klassen von Lösungsansätzen in Frage:

1. Die Packstücke können frei im Container verteilt werden.
2. Aus den einzelnen Packstücken werden zunächst Stapel gebildet, die dann auf dem Boden des Containers verteilt werden.

Die algorithmischen Ansätze für diese beiden Problemklassen unterscheiden sich grundsätzlich voneinander. Im ersten Fall hat man es mit einem dreidimensionalen orthogonalen Packproblem (3OP) zu tun, bei dem prinzipiell eine freie Anordnung der Packstücke innerhalb des Containers möglich ist. Im zweiten Fall hat man es hingegen mit zwei in der Komplexität reduzierten Teilproblemen zu tun, die sequentiell gelöst werden können.

In der Transportlogistik hat man es häufig mit dem zweiten der oben genannten Fälle zu tun (vgl. [Vastag, Presifilippo, und Schwarz 2005]) und auch die in dieser Arbeit verwendeten Real-World-Eingabedaten sehen eine Stapelbildung der Gitterboxen vor.[1] Deshalb wird in dieser Arbeit der zweite der genannten Fälle betrachtet.

Zunächst wird ein Algorithmus benötigt, der unter Einhaltung der gegebenen Stapelbarkeitsvorschriften geeignete Boxenstapel für jeden Kunden bildet. Das Ziel dabei ist es, die Packstücke unter Berücksichtigung der Höhe H des Transportraums möglichst hoch zu stapeln und dadurch die Anzahl entstehender Boxenstapel für jeden Kunden zu minimieren. Dadurch soll der vertikale Raum im Frachtraum des Transportfahrzeugs optimal genutzt und gleichzeitig die zum Platzieren der Stapel benötigte Grundfläche minimiert werden. Die Bildung der Boxenstapel für jeden Kunden kann in einem Preprocessingschritt stattfinden.

In Abschnitt 4.2 wird ein heuristischs Verfahren zur Bildung von Boxenstapeln vorgestellt, das trotz seiner hohen Ausführungsgeschwindigkeit eine sehr gute Lösungsqualität garantiert.

[1] Für eine genaue Beschreibung der Eingabedaten siehe Abschnitt 6.3.

Des Weiteren wird ein Algorithmus zur Lösung des zweidimensionalen, orthogonalen Packproblems (2OPP) benötigt, welcher die vorab berechneten Boxenstapel möglichst geschickt im Stauraum des Transportfahrzeugs platziert. Da das 2OPP zur Klasse der NP-vollständigen Probleme gehört[2], sollte ein Verfahren eingesetzt werden, das einen guten Kompromiss zwischen benötigter Laufzeit und Lösungsqualität bietet. Abhängig von der Art und Weise, wie der Algorithmus zur Lösung des 2OPP mit dem Tourenplanungsalgorithmus kombiniert wird (siehe hierzu Kapitel 5) und der Zeit, die dem Gesamtalgorithmus zur Verfügung gestellt werden kann, sind die Ansprüche an den Algorithmus entweder kurze Laufzeiten oder aber eine hohe Lösungsqualität. Deshalb wurden im Rahmen dieser Arbeit mehrere Algorithmen implementiert, deren Fokus auf jeweils einer dieser beiden Anforderungen liegt. In Abschnitt 4.4 wird eine schnelle Heuristik zur Lösung des 2OPP vorgestellt; Abschnitte 4.5 und 4.6 beschreiben zwei exakte Verfahren zur Lösung des Problems.

Es sei an dieser Stelle angemerkt, dass in den Abschnitten 4.4, 4.5 und 4.6 die Ausmaße der zweidimensionalen Items als Breite und Höhe (w, h) bezeichnet werden. Dies ist zwar eine Inkonsistenz zu der Tatsache, dass es sich bei den Items eigentlich um dreidimensionale Boxenstapel handelt, die die Ausmaße (w, h, d) haben. Die Stapelhöhe h spielt in den genannten Abschnitten jedoch keine Rolle, also bleiben die Breite und Tiefe (w, d) als relevante Größen. Diese werden in der Literatur zum Themengebiet der zweidimensionalen, orthogonalen Packprobleme jedoch als Breite und Höhe (w, h) bezeichnet, weshalb auch in dieser Arbeit so verfahren wird.

Alle implementierten Verfahren berechnen bei Bedarf Lösungen, die bezüglich der LIFO-Reihenfolge der berechneten Packmuster korrekt sind. Darüber hinaus beherrschen die Verfahren optional die Drehbarkeit der Items um 90°.

Es wird stets davon ausgegangen, dass alle Items in einem Container kantenparallel bezüglich der Wände des Containers orientiert sind.

Der Begriff Packmuster kann für den zweidimensionalen Fall wie folgt definiert werden (vgl. [Schepers 1997]):

Definition 4.0.1 (Packmuster)**.** Gegeben sei ein rechtwinkliger Container mit den Ausmaßen (W, H) und eine Menge $I = \{i_1, \ldots, i_n\}$ von ebenfalls rechtwinkligen Objekten (*Items*) mit Ausmaßen (w_i, h_i), $i = 1, \ldots, n$. Sei $P : I \to \mathbb{Q}^2$, $P(i) = (x_i, y_i)$ eine Abbildung, welche jedem Item $i \in I$ eine Position (x_i, y_i) im zweidimensionalen Koordinatensystem zuordnet. P wird genau dann ein Packmuster für I genannt, wenn die folgenden Bedingungen erfüllt sind:

1. Die Items überlappen sich nicht gegenseitig.

$$x_i \geq x_j + w_j \vee y_i \geq y_j + h_j \vee x_j \geq x_i + w_i \vee y_j \geq y_i + h_i \; \forall i, j \in I, \; i \neq j \tag{4.1}$$

2. Die Items ragen nicht aus dem Container.

$$x_i, y_i \geq 0 \wedge x_i + w_i \leq W \wedge y_i + h_i \leq H \; \forall i \in I \tag{4.2}$$

[2]Ein Beweis für die NP-Vollständigkeit des 2OPP wurde von [Garey und Johnson 1979] geführt.

Eine Itemmenge I und das zugehörige Packmuster $P(I)$ definieren eindeutig die Bepackung $(I, P(I))$ eines Containers. Zur Vereinfachung wird in den folgenden Abschnitten der Begriff Packmuster als Synonym für eine Itemmenge verwendet, deren Elementen zulässige Platzierungen innerhalb eines Containers zugeordnet sind. Es wird also davon ausgegangen, dass mit einem Packmuster P eine Menge von Items i mit Ausmaßen (w_i, h_i) und zugehörigen Platzierungen (x_i, h_i) bezeichnet wird.

Das Konzept der Korrektheit eines Packmusters bezüglich der LIFO-Reihenfolge wird in Abschnitt 4.3 erläutert.

Da es sich beim 2OPP um eine spezielle Entscheidungsvariante des zweidimensionalen Bin-Packing-Problems (2BPP) handelt („Kann die Menge I von Items in einem Container mit Ausmaßen (W, H) unter den gegebenen Restriktionen untergebracht werden?“), kommt es üblicherweise nicht auf die Qualität einer Lösung an, sondern nur auf ihre Zulässigkeit und die für die Berechnung nötige Laufzeit.[3] In diesem Fall ist es sinnvoll, Eingaben für das 2OPP, für die keine zulässige Lösung existiert, frühzeitig zu identifizieren. Diesen Zweck kann die Berechnung einer unteren Schranke L für die Anzahl benötigter Container erfüllen. Falls $L > 1$, existiert offenbar keine gültige Lösung und eine solche muss gar nicht erst gesucht werden. In Abschnitt 4.7 werden untere Schranken für das 2BPP beschrieben, die in dieser Diplomarbeit implementiert und eingesetzt wurden. Schließlich beschreibt Abschnitt 4.8 ein gestaffeltes Verfahren für das 2OPP, in dem zunächst die unteren Schranken überprüft, dann das heuristische Verfahren und erst zum Schluss die exakte Methode verwendet wird.

4.1 Algorithmen für das Cutting-&-Packing

Ähnlich wie für das Tourenplanungsproblem existieren im Zusammenhang mit dem Laderaumoptimierungsproblem viele verschiedene Aufgabenstellungen und Anforderungen, so dass im Laufe der Zeit diverse Lösungsansätze entwickelt und getestet wurden. Hauptsächlich unterscheidet man zwischen zweidimensionalen und dreidimensionalen Packungsproblemen, wobei die zweidimensionalen Varianten immer dann zum Einsatz kommen, wenn die zu packenden Gegenstände entweder nicht stapelbar sind oder die entsprechenden Stapel bereits im Vorfeld gebildet werden können und nur noch auf der Grundfläche des Transportbehälters zu verteilen sind. Zusätzlich kommt eine Fülle weiterer Nebenbedingungen wie die Zerbrechlichkeit der Items, die Stabilität der Bepackung, spezielle Vorschriften zur Stapelbarkeit und die gleichmäßige Gewichtsverteilung im Container in Betracht, so dass entsprechend viele Lösungsansätze für die Laderaumoptimierung existieren. Auch hier gilt, dass der passende Algorithmus immer abhängig von der konkreten Aufgabenstellung und den gestellten Anforderungen mit Bedacht gewählt werden muss, da bislang kein allgemein bestes Verfahren bekannt ist.

Im Folgenden soll ohne Anspruch auf Vollständigkeit eine Übersicht über die existierenden Verfahren zur Lösung des Laderaumoptimierungsproblems gegeben werden.

[3] Diese Aussage ist zwar im Allgemeinen korrekt, trifft aber nicht immer zu. Wie in Kapitel 5 gezeigt wird, genügt es im Fall der isolierten und der sequentiellen Methode für das VRTWLP sehr wohl, eine qualitativ möglichst gute Lösung des 2OPP zu finden, die nicht unbedingt alle Packstücke im Container enthalten muss. In diesem Fall erübrigt sich der Einsatz der unteren Schranke für das 2BPP.

4.1.1 Heuristiken

Die einfachste Möglichkeit das zweidimensionale Bin-Packing-Problem (2BPP) heuristisch zu lösen besteht darin, die zu packenden Items zunächst ihrer Höhe nach absteigend zu sortieren und sie dann in regalförmiger Anordnung sequentiell in einem Strip[4] unterzubringen. Anschließend wird versucht, die Regale auf eine möglichst kleine Anzahl Container zu verteilen. Diese Möglichkeit wurde bereits mehrfach implementiert und getestet, wobei verschiedene Autoren unterschiedliche Strategien zur Anordnungen der Items in den Regalen sowie zur anschließenden Verteilung der Regale auf möglichst wenige Container untersucht haben.

[Chung, Garey, und Johnson 1982] entwickelten den Algorithmus *Hybrid-First-Fit*, der die zur Lösung des eindimensionalen Bin-Packing-Problems (BPP) bekannte *First-Fit* Heuristik sowohl für die Generierung als auch die Verteilung der Regale verwendet.

Der Algorithmus *Finite-Best-Strip* von [Berkey und Wang 1987] ist hingegen an die *Best-Fit* Strategie für das BPP angelehnt und verwendet diese für die beiden oben genannten Phasen.

[Frenk und Galambos 1987] entwickelten den Algorithmus *Hybrid-Next-Fit*, welcher wiederum die *Next-Fit* BPP-Lösungsstrategie für beide Phasen des zweidimensionalen Falls umsetzt.

Eine weitere Optimierung bei der regalförmigen Anordnung von Items im Container wurde von [Lodi, Martello, und Vigo 1999] in Form des Algorithmus *Floor-Ceiling* vorgeschlagen. Die Verbesserung besteht darin, dass die Regale nicht nur von links unten nach rechts unten, sondern zusätzlich auch von rechts oben nach links oben mit Items gefüllt werden falls die räumlichen Gegebenheiten dies zulassen. Auf diese Weise kann eine bessere Ausnutzung der zur Verfügung stehenden Ladefläche erzielt werden, was aber mit schlechteren Laufzeiteigenschaften einhergeht.

Ein weiterer, auf dem Regal-Konzept basierender Ansatz ist der Algorithmus *Knapsack-Packing* von [Lodi, Martello, und Vigo 1999]. Hierbei wird jedes neue Regal vom höchsten verbliebenen Item initialisiert. Der verbleibende Raum wird optimal von den bislang nicht-platzierten Items belegt, indem ein Knapsack-Problem mit den Breiten der Items als Gewichte und ihren Flächen als Profite gelöst wird. Auch wenn das Knapsack-Problem in der Theorie NP-hart ist, haben Experimente gezeigt, dass im Kontext von Knapsack Packing der Algorithmus häufig bereits nach kurzer Laufzeit ein optimales Ergebnis für das zu lösende Regal finden kann.

Eine 2BPP-Heuristik mit der Bezeichnung *Finite-Bottom-Left*, welche nicht auf dem Regal-Konzept basiert, wurde von [Berkey und Wang 1987] vorgeschlagen. Hierbei werden zunächst alle Items absteigend nach ihrer Breite sortiert. Anschließend wird ein Item nach dem nächsten in den aktuell offenen Container platziert, und zwar an die unterste linke Position, in die es hinein passt.

Ein anderer, nicht auf dem Regal-Konzept basierender Algorithmus wurde von [Lodi, Martello, und Vigo 1999] entwickelt. Er trägt den Namen *Alternate-Directions* und hält anfangs eine Menge von L Containern offen. Hierbei ist L eine untere 2BPP-Schranke.

[4] Mit „Strip“ bezeichnet man einen nach oben hin offenen, also in der Höhe unbeschränkten, Container. Die Breite des Strips entspricht der des Original-Containers.

Die Items werden absteigend nach ihrer Höhe sortiert, um anschließend gemäß des *Best-Fit* Verfahrens von links nach rechts zeilenweise in den Containern platziert zu werden. Sobald jeder Container mit einer Item-Zeile gefüllt ist, wird das Verfahren für alle Container in umgekehrter Richtung wiederholt, d.h. es werden Zeilen gebildet, die von rechts nach links verlaufen und auf den im ersten Schritt gebildeten Item-Zeilen aufliegen. Dieses Verfahren wird dann so oft wiederholt, bis keiner der Container weitere Items aufnehmen kann. Dann werden so lange einzelne neue Container geöffnet, bis alle Items platziert wurden.

[Lodi, Martello, und Vigo 2002] stellen den heuristischen Algorithmus *Height-first-Area-second* zur Lösung des dreidimensionalen Bin-Packing-Problems (3BPP) vor. Dieser füllt den Container von unten nach oben schichtweise mit Items, wobei jede Schicht das Lösen eines 2BPP erfordert. Diese Heuristik wird im Rahmen eines Tabu-Search-Verfahrens zur Lösung des 3BPP verwendet.

Eine Heuristik zur Lösung des dreidimensionalen orthogonalen Rucksackproblems (3KP) wird von [Bischoff 2006] vorgeschlagen. Falls gewünscht berücksichtigt dieses Verfahren die Gewichte und die Tragekapazitäten der Items. Gute experimentelle Ergebnisse, eine parametrisierbare Bewertungsfunktion sowie sehr einfache und kompakte Datenstrukturen lassen das Verfahren sehr attraktiv erscheinen.

4.1.2 Untere Schranken

Untere Schranken im Kontext des mehrdimensionalen Bin-Packing-Problems dienen dazu, die Anzahl benötigter Container für eine Menge zu packender Items nach unten abzuschätzen. Sie können beispielsweise innerhalb von Branch-and-Bound-Algorithmen verwendet werden, um an einigen Stellen teure Enumerationen einzusparen und so die Laufzeit drastisch zu reduzieren.

Die naheliegendste untere Schranke besteht in der sogenannten *Continuous Lower Bound*. Sie kann sehr schnell berechnet werden und basiert auf einer simplen Abschätzung der Anzahl benötigter Container auf Grundlage der Volumina (bzw. Flächen im zweidimensionalen Fall) der zu packenden Items und des Containers. Eine Analyse der Worst Case Performance der Continuous Lower Bound wurde von [Dell'Amico 1999] durchgeführt.

Bessere untere Schranken beruhen auf einer geschickten Partitionierung der Items in Größenklassen und der Beobachtung, dass beispielsweise zwei relativ breite (oder auch hohe bzw. tiefe) Items nicht nebeneinander (übereinander, hintereinander) im selben Container platziert werden können. [Martello und Vigo 1998] haben solche unteren Schranken untersucht; [Boschetti 2004] stellt weiter fortgeschrittene untere Schranken dieser Kategorie für das 3BPP vor.

Eine weitere Klasse von unteren Schranken ergibt sich durch die sogenannte konservative Skalierung der zu packenden Items mittels dualzulässiger Funktionen. Diese haben die Eigenschaft, dass sie die Fläche bzw. das Volumen der Items vergrößern, aber trotzdem die Lösbarkeit des Ausgangsproblems nicht verändern. Falls also nach der Anwendung einer dualzulässigen Funktion auf eine Menge von Items die Gesamtfläche dieser Items größer ist als die eines Containers, weiß man sofort, dass diese Items nicht im Container untergebracht werden können. Aufgrund der oben genannten Eigenschaft der dualzulässigen Funktionen weiß man aber auch, dass die ursprünglichen, nicht skalierten Items nicht in den Container passen. Dualzulässige Funktionen wurden für untere Schranken des

mehrdimensionalen Bin Packing Problems erstmals von [Fekete und Schepers 2001] eingesetzt.

4.1.3 Exakte Verfahren

Das mehrdimensionale Laderaumoptimierungsproblem ist ein sehr schwieriges Optimierungsproblem, weshalb exakte Lösungsverfahren nur dann angemessen sind, wenn die Eingabeinstanzen erwartungsgemäß weniger als 100 Items umfassen oder wenn die Laufzeit keine kritische Rolle spielt. Ein enumeratives Verfahren zur Lösung des 2BPP wurde von [Martello und Vigo 1998] vorgestellt. Es handelt sich um einen Branch-and-Bound-Algorithmus, der ein zweischichtiges Branching-Schema beinhaltet. In einem äußeren B&B-Baum wird ein Item jedem offenen Container zugeordnet, sowie ein neuer Container für dieses Item geöffnet. Die anschließende Platzierung des Items im jeweiligen Container erfolgt dann in einem inneren B&B-Baum. Dies kann einen hohen Rechenaufwand erfordern, da dabei möglicherweise alle Packmuster des jeweiligen Containers durchgerechnet werden müssen. Ein Versuch diesen Rechenaufwand zu vermeiden ist es, zunächst die Bepackung des Containers heuristisch zu lösen. Ein weiteres Mittel zur Reduzierung des Rechenaufwands besteht in der Berechnung einer unteren Schranke für die Anzahl benötigter Container für die jeweilige Menge der zu platzierenden Items.

Ein exakter Algorithmus für das 3BPP wird von [Martello, Pisinger, und Vigo 2000] vorgeschlagen. Er basiert auf der Idee, eine Menge von sogenannten Eckpunkten (*Corner Points*) zunächst in zwei, dann in drei Dimensionen zu bestimmen. Die Eckpunkte werden als potentielle Kandidaten für das Platzieren von Items verwendet. Anschließend werden – ähnlich wie in [Martello und Vigo 1998] – im Rahmen eines zweischichtigen Branch-and-Bound-Algorithmus ungepackte Items in den Eckpunkten positioniert (innerer B&B-Baum) und einzelnen, aktiven Containern zugeordnet (äußerer B&B-Baum). Auch hier werden untere Schranken sowie ein heuristisches Verfahren eingesetzt, um den Rechenaufwand zu reduzieren.

Ein exakter Algorithmus für das mehrdimensionale orthogonale Packungsproblem (d-OPP) wurde von [Fekete, Schepers, und van der Veen 2007] vorgestellt. Er ist sowohl auf den zwei- als auch für den dreidimensionalen Fall anwendbar und bedient sich fortgeschrittener Ideen wie isomorpher Packungsklassen und der konservativen Skalierung mittels dualzulässiger Funktionen. Den Kern des Algorithmus bildet ein Branch-and-Bound-Verfahren.

[Clautiaux, Carlier, und Moukrim 2007] stellen zwei exakte, Branch-and-Bound-basierte Verfahren (*LMAO* und *TSBP*) zur effizienten Lösung des 2OPP vor. Dabei verwenden sie zunächst zwei Reduktionsmethoden, um im Vorfeld die Problemgröße zu verkleinern und unter Umständen nicht lösbare Instanzen zu erkennen. Des Weiteren werden fortgeschrittene untere Schranken in Form konservativer Skalierungen eingesetzt, um nicht lösbare Instanzen im Vorfeld zu identifizieren. Besondere Aufmerksamkeit wird der Vermeidung von Redundanzen in der Enumeration von Packmustern gewidmet.

4.2 Die Heuristik First-Fit-Decreasing zur Bildung von Boxenstapeln

Die im Rahmen dieser Diplomarbeit untersuchten und implementierten Algorithmen zur Lösung des Laderaumoptimierungsproblems sind auf das zweidimensionale orthogonale Packproblem (2OPP) ausgerichtet. Im Bereich der Transportlogistik gibt es zwei Szenarien, die eine Lösung des 2OPP erfordern:

1. Die transportierten Gegenstände dürfen nicht gestapelt werden.
2. Die Stapelbarkeit der Gegenstände unterliegt gewissen Regeln, die ausschließlich die Bildung von wohldefinierten Stapeln zulassen. Wohldefiniert ist ein Stapel nur dann, wenn die Grundflächen aller seiner Bausteine kongruent sind. Zudem wird in einigen Fällen das Transportgut in sogenannten Gitterboxen verstaut, die nur dann übereinander gestapelt werden können, wenn die jeweiligen Verankerungsmechanismen an der unteren und oberen Fläche zueinander kompatibel sind. Ein versetztes Stapeln von Gitterboxen ist in keinem Fall zulässig.

Im zweiten der beiden Fälle ist es notwendig, zunächst für jeden Kunden aus allen zu transportierenden Boxen unter Berücksichtigung der gegebenen Stapelbarkeitsvorschriften möglichst wenige Stapel zu bilden, damit diese dann auf der Grundfläche des Stauraums des eingesetzten Transportfahrzeugs angeordnet werden können.

Da in der Praxis das Entladen der Ware eines Kunden ohnehin schon sehr zeitintensiv und schwierig sein kann, ist die hier vorgestellte Algorithmik auf den Fall zugeschnitten, dass die Boxen in jedem Boxenstapel immer genau einem Kunden zugeordnet sind, es also keine Stapel mit Boxen verschiedener Kunden gibt. Dies führt zu der Vereinfachung, dass die Bildung der Boxenstapel für den Bedarf jedes Kunden unabhängig voneinander stattfinden kann und sich das Problem somit auf die Lösung des eindimensionalen Bin-Packing-Problems reduzieren lässt. Dabei fungiert die Höhe H des Stauraums als Größe eines Bins und die Höhe h_b einer Box b als die Größe eines Items.

Der Algorithmus *First-Fit-Decreasing-Stackbuilder* (Algorithmus 3) verwendet zur Bildung von Boxenstapeln die *First-Fit-Decreasing*-Heuristik (*FFD*) für das eindimensionale Bin-Packing-Problem. Eingabeparameter für den Algorithmus sind die Menge B von Boxen und die maximale Stapelhöhe H. Eine Box b hat die Ausmaße (w_b, d_b, h_b) und eine Stapeltechnik st_b. Zwei Boxen b_1, b_2 sind zueinander kompatibel und können somit übereinander gestapelt werden, wenn gilt: $w_{b_1} = w_{b_2}, d_{b_1} = d_{b_2}$ und $st_{b_1} = st_{b_2}$.

Zunächst wird die Boxenmenge B in Kompatibilitätsklassen partitioniert (Zeilen 4-11 in Algorithmus 3). Anschließend wird für jede Menge C kompatibler Boxen die Prozedur *BuildStacks* aufgerufen. Diese Prozedur implementiert die *FFD*-Heuristik, welche zunächst alle Boxen absteigend ihrer Höhe nach sortiert (Zeile 17). Anschließend werden die Boxen nacheinander auf Stapel verteilt (Zeilen 18-24), und zwar nach der Regel, dass jede Box dem ersten Stapel zugeordnet wird, auf den sie gestapelt werden kann, ohne die Fahrzeughöhe zu überschreiten (Zeile 20). Falls eine Box keinem Stapel zugeordnet werden kann, wird ein neuer Stapel erstellt (Zeile 23).

In [Yue 1991] wurde gezeigt, dass *FFD* eine erstaunlich gute Lösungsqualität liefert. Es gilt nämlich für jede Eingabe l: $FFD(l) \leq \frac{11}{9}OPT(l) + 1$.

Algorithmus 3: First-Fit-Decreasing-Stackbuilder

```
1:  procedure FFDStackBuilder(B, H)
2:      Ŝ ← ∅                                  ▷ Rückgabewert: Menge aller Boxenstapel
3:      Ĉ ← ∅                                  ▷ Menge von Mengen kompatibler Boxen
4:      for all b ∈ B do                       ▷ Iteriere über alle Boxen
5:          for all C ∈ Ĉ do                   ▷ Iteriere über alle Kategorien
6:              if b kompatibel zu den Boxen in C then
7:                  C ← C ∪ {b}
8:                  goto 4
9:          C ← new BoxCategory(w_b, d_b, st_b)
10:         C ← C ∪ {b}
11:         Ĉ ← Ĉ ∪ {C}
12:     for all C ∈ Ĉ do
13:         Ŝ ← Ŝ ∪ BuildStacks(C, H)
14:     return Ŝ

Vorbedingung: Alle Boxen in C sind bezüglich Stapelbarkeit paarweise kompatibel
15: procedure BuildStacks(C, H)
16:     Ŝ ← ∅                                  ▷ Rückgabewert: Menge aller Boxenstapel
17:     Sortiere C absteigend nach der Boxenhöhe
18:     for all b ∈ C do
19:         for all S ∈ Ŝ do
20:             if h_S + h_b ≤ H then
21:                 S ← S ∪ {b}
22:                 goto 18
23:         S ← {b}                            ▷ Erstelle neuen Stapel S mit b als einziger Box
24:         Ŝ ← Ŝ ∪ {S}
25:     return Ŝ
```

Satz 4.2.1. FFD *hat eine Laufzeit von* $O(n^2)$.

Beweis. Im Worst Case müssen für das i-te Item $i-1$ Bins getestet und ein neuer angelegt werden. Daraus folgt die Behauptung. □

4.3 LIFO-Reihenfolge der Bepackung

In diesem Abschnitt werden die Konzepte der LIFO-Reihenfolge und der umgekehrten LIFO-Reihenfolge der Bepackung eines Transportfahrzeugs erläutert. Die Einhaltung der LIFO-Reihenfolge ist eine Anforderung, die sich bei der Distribution von Gütern ergeben kann. Die umgekehrte LIFO-Reihenfolge sollte hingegen eingehalten werden, wenn es um das Einsammeln von Gütern geht.

Definition 4.3.1 (LIFO-Reihenfolge der Bepackung)**.** Gegeben sei ein Packmuster $P = \{i_1, \ldots, i_n\}$ sowie eine Menge $V = \{v_1, \ldots, v_l\}$ von Kunden, deren Waren gemäß der Reihenfolge einer Tour sequentiell entladen werden müssen. Des Weiteren sei mit jedem Item $i \in P$ ein Wert $prio_i \in \{1, \ldots, l\}$ assoziiert, der einerseits das Item i eines Kunden aus V über den Index dem Kunden zuordnet, und andererseits eine Priorität für die Reihenfolge des Entladens der Items darstellt.[5] Das Packmuster P nennt man korrekt bezüglich LIFO-Reihenfolge, wenn alle Items aus P in der Reihenfolge ihrer Prioritäten entladen werden können, ohne dass laterale Verschiebungen durchgeführt werden müssen. Es dürfen also Items mit hoher Priorität nicht von Items mit niedriger Priorität blockiert werden. Formal ausgedrückt muss gelten:

$$\forall i, j \in P \text{ mit } prio_i < prio_j : x_i + w_i \leq x_j \vee x_i \geq x_j + w_j \vee y_i \geq y_j + h_j \tag{4.3}$$

Definition 4.3.2 (Umgekehrte LIFO-Reihenfolge der Bepackung)**.** Gegeben seien ein Packmuster P, eine Menge V von Kunden sowie die Prioritäten $prio_i,\ i \in P$ der Packstücke wie in Definition 4.3.1. Wir nennen das Packmuster P korrekt bezüglich der umgekehrten LIFO-Reihenfolge, wenn es durch sukzessives Beladen der Packstücke in nicht absteigender Reihenfolge ihrer Prioritäten in den zugehörigen Container generiert werden kann, ohne dass laterale Verschiebungen durchgeführt werden müssen. Es dürfen also in diesem Fall Items mit niedriger Priorität nicht von Items mit hoher Priorität blockiert werden, was sich durch folgender Bedingung ausdrücken lässt:

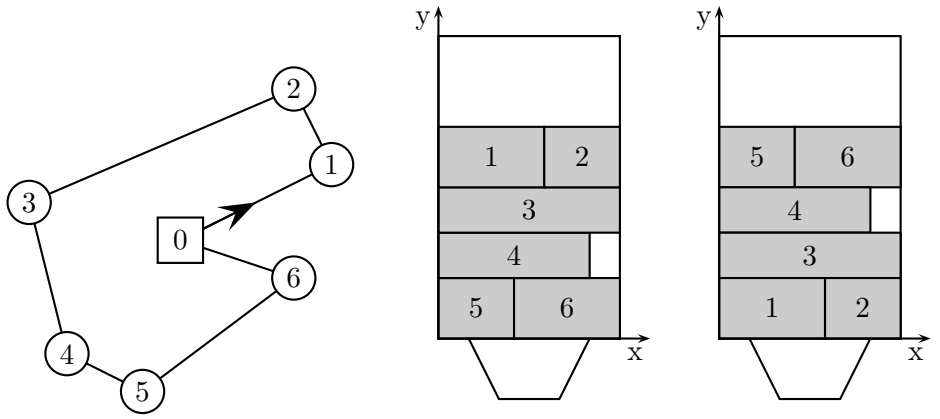
$$\forall i, j \in P \text{ mit } prio_i < prio_j : x_i + w_i \leq x_j \vee x_i \geq x_j + w_j \vee y_i + h_i \leq y_j \tag{4.4}$$

Der Unterschied zwischen der korrekten Reihenfolge der Packstücke in den Fällen der Verteilung und des Einsammelns von Gütern wird in Abbildung 4.1 verdeutlicht.

In Abbildung 4.1(a) ist eine Beispieltour zu sehen, die am Standort 0 (Depot) beginnt, sukzessive die Kunden 1 bis 6 besucht und wieder am Depot endet. Jedem Kunden sei ein Packstück zugeordnet, das die Bezeichnung $1, \ldots, 6$ des jeweiligen Kunden trägt.

Zuerst wird der Fall betrachtet, dass die Packstücke an die Kunden ausgeliefert werden sollen. Um das Entladen der Packstücke bei den Kunden zu vereinfachen, wird dabei die

[5] Im Kontext von Prioritäten kennzeichnen kleinere Werte üblicherweise höhere Prioritäten. Diese Konvention wird in dieser Arbeit eingehalten.



(a) Beispieltour mit 6 Kunden

(b) LIFO-Reihenfolge beim Entladen von Gütern

(c) Umgekehrte LIFO-Reihenfolge beim Beladen von Gütern

Abbildung 4.1: Beispieltour und die zugehörige Fahrzeugbepackung mit LIFO-Reihenfolge und umgekehrter LIFO-Reihenfolge

Einhaltung der LIFO-Reihenfolge gefordert. Abbildung 4.1(b) zeigt eine zulässige Bepackung des Fahrzeugstauraums, wenn man davon ausgeht, dass das Fahrzeug in der Abbildung von oben beladen wird. Die Packstücke müssen zuvor in umgekehrter Reihenfolge des Auftretens der Kunden in der Tour in den LKW gepackt worden sein, damit sie in korrekter Reihenfolge entladen werden können.

Als nächstes wird der Fall betrachtet, dass die Packstücke bei den Kunden eingesammelt werden sollen. Die Packstücke sind dabei dieselben wie im vorangehenden Beispiel. Wenn das Umordnen bereits früher eingesammelter Packstücke nicht zugelassen ist, kann sich eine Bepackung ergeben, wie sie in Abbildung 4.1(c) zu sehen ist. Offensichtlich ist die Packreihenfolge hierbei genau entgegengesetzt zum erstgenannten Fall.

Die in dieser Arbeit implementierten Algorithmen sind auf die Auslieferung von Waren und die damit verbundene Einhaltung der LIFO-Reihenfolge zugeschnitten. Dadurch wird die Korrektheit der Verfahren für den Fall des Einsammelns von Waren jedoch nicht beeinträchtigt, denn ein Packmuster P, das bezüglich der LIFO-Reihenfolge korrekt ist, kann leicht in ein Packmuster P' überführt werden, welches bezüglich der umgekehrten LIFO-Reihenfolge korrekt ist und umgekehrt. Hierzu genügt die Spiegelung aller Items an der Geraden $y = H/2$. Dies geschieht durch folgende Berechnung neuer Platzierungen (x'_i, y'_i) für alle Items $i \in P$: $x'_i = x_i$ und $y'_i = (H - y_i - h_i)$ (siehe [Iori, Gonzalez, und Vigo 2007]).

4.4 Die Heuristik TP_{MOD} für die Laderaumoptimierung

Der Algorithmus *Touching-Perimeter* (TP_{RF}[6]) ist eine Heuristik, die ursprünglich zur Lösung des zweidimensionalen Bin-Packing-Problems (2BPP) entwickelt und in [Lodi, Martello, und Vigo 1999] vorgestellt wurde. Im Falle des 2BPP sind eine zunächst beliebig große Menge von rechtwinkligen Containern der Größe (W, H), sowie eine Menge I von ebenfalls rechtwinkligen Items $i_1, \ldots, i_n$ mit den Ausmaßen $(w_1, h_1), \ldots, (w_n, h_n)$ gegeben. Die Aufgabe besteht darin, die Items so in den Containern zu platzieren, dass

1. die Items sich nicht gegenseitig überschneiden,
2. die Containergrenzen nicht überschritten werden und
3. die Anzahl verwendeter Container minimiert wird.

Im Kontext dieser Diplomarbeit ist jedoch das zweidimensionale orthogonale Packproblem (2OPP) von Interesse, welches eine eingeschränkte Version des 2BPP darstellt. Beim 2OPP gibt es nur einen Container und man löst das Problem, alle Items aus I in diesem unterzubringen. Aus diesem und einigen anderen Gründen, die später näher erläutert werden, musste das für das 2BPP entwickelte Verfahren TP_{RF} leicht modifiziert werden. Dies resultierte in dem neuen Algorithmus TP_{MOD}, der jedoch sehr stark an das ursprüngliche Verfahren TP_{RF} angelehnt ist.

Im Folgenden wird davon ausgegangen, dass die Fläche des Containers in den oberen rechten Quadranten eines zweidimensionalen Koordinatensystems abgebildet ist, so dass sich die untere linke Ecke des Containers im Ursprung des Koordinatensystems befindet. Des Weiteren wird festgelegt, dass der Container „von oben“ beladen wird und die ersten Packstücke „unten“, d.h. bei $y = 0$, platziert werden.

Die Grundidee hinter TP_{RF} besteht darin, Items nach und nach in *normalen Positionen* zu platzieren. Der Begriff normale Position stammt aus [Christofides und Whitlock 1977] und bezeichnet Platzierungen, an denen ein Item nicht weiter nach unten oder links im Koordinatensystem geschoben werden kann, ohne ein benachbartes Item zu überlappen oder aus dem Container herauszuragen. Um im Container möglichst wenig Verschnitt, d.h. nicht weiter nutzbare Flächen, zu erzeugen, sortiert TP_{RF} im ersten Schritt die Items absteigend nach ihrer Fläche. Im zweiten Schritt werden die Items in dieser Reihenfolge an normalen Positionen platziert, wobei für ein Item stets diejenige Position und Orientierung gewählt wird, an der das Item die größte Berühroberfläche zu anderen Items oder den Containergrenzen hat. Dieser recht simple Algorithmus ist schnell und hat zudem im Fall mit drehbaren Items in experimentellen Untersuchungen bessere Ergebnisse erzielt als die beiden zum Vergleich herangezogenen Heuristiken FC_{RF} (*floor-ceiling*) und AD_{RF} (*alternate directions*) (vgl. [Lodi, Martello, und Vigo 1999]).

Im Rahmen dieser Arbeit wurde der Algorithmus TP_{MOD} entwickelt, der gegenüber TP_{RF} die folgenden Modifikationen aufweist:

[6] Die tiefgestellten Buchstaben RF stammen aus einer Klassifizierung von Packproblemen und Packalgorithmen, die von [Lodi, Martello, und Vigo 1999] vorgenommen wurde. Das R (*rotate*) kennzeichnet Probleme, in denen die Items gedreht werden können. Das F steht für *free* und kennzeichnet Probleme, die nicht der Einschränkung unterliegen, durch eine Folge von sogenannten „Guillotine-Schnitten“ lösbar sein zu müssen.

1. TP_{MOD} berücksichtigt optional die LIFO-Reihenfolge der Bepackung.
2. Der Algorithmus kennt die Reihenfolge der Kunden auf der Tour. Auch wenn keine LIFO-Reihenfolge der Bepackung gefordert ist, findet eine Vorsortierung der Kunden statt, die dazu beitragen soll, die Güter möglichst vieler Kunden vollständig zu packen (siehe Kapitel 5).
3. Der Algorithmus packt Kunden atomar, d.h. es werden entweder alle Packstücke eines Kunden gepackt oder keine. Nur teilweise gepackte Kunden können ohnehin nicht verwendet werden. Daher werden Packstücke solcher Kunden frühzeitig aus dem Container entfernt, um Platz für die Packstücke anderer Kunden freizugeben.
4. Da in [Lodi, Martello, und Vigo 1999] keine Vorschrift angegeben ist, wie für ein gegebenes Packmuster die Menge der normalen Positionen berechnet werden kann, wurde hierfür ein eigenes Verfahren entwickelt.

4.4.1 Vorsortierung der Packstücke

Um die Effizienz des Verfahrens zu steigern, werden die Packstücke vor Ausführung des Algorithmus nach gewissen Kriterien sortiert. Zunächst wird der Fall betrachtet, in dem keine LIFO-Reihenfolge benötigt wird. In diesem hängt die Sortierreihenfolge davon ab, ob vom Algorithmus gefordert wird, die gegebene 2OPP-Eingabeinstanz vollständig zu lösen (1) oder ob es genügt, eine möglichst gute Lösung zu finden (2).

Fall 1 tritt insbesondere dann auf, wenn die Laderaumoptimierung von einem Tourenplanungsalgorithmus aufgerufen wird, der im integrierten Modus (siehe Abschnitt 5.2) ausgeführt wird. Im Kontext des integrierten Modus hängt die Sortierreihenfolge der Items nicht davon ab, welchen Kunden diese zugeordnet sind. Falls die Instanz nicht vollständig gepackt werden kann, spielt keine Rolle, welche Packstücke nicht im Container untergebracht werden können. Deshalb kann hier die gleiche Sortierung der Items vorgenommen werden, die auch im ursprünglichen Verfahren TP_{RF} verwendet wird. Die Packstücke werden gemäß ihrer Fläche $w_i h_i$ absteigend sortiert; dabei werden Items mit gleicher Fläche absteigend nach der Länge ihrer kürzeren Seite, also dem Wert $\min\{w_i, h_i\}$ geordnet. Es werden also zunächst große Items gepackt, um später eventuell entstehende Freiräume mit kleineren Packstücken füllen zu können. Im Falle mehrerer Items mit identischen Ausmaßen (w, h) sorgt das zweite Kriterium dafür, dass solche Items direkt hintereinander gepackt werden. Auf diese Weise erhält der Algorithmus potentiell die Möglichkeit, identische Items „en bloc“ zu verstauen und dabei möglichst wenige Lücken in der Bepackung zu hinterlassen. Falls für ein beliebiges Item im Verlauf des Algorithmus keine geeignete Platzierung gefunden wird, kann das Verfahren mit negativem Rückgabewert abgebrochen werden.

Fall 2 tritt auf, wenn die Tourenplanung im isolierten oder im sequentiellen Modus ausgeführt wird (siehe Abschnitte 5.3 und 5.4). In diesen beiden Modi werden zunächst Touren ohne Berücksichtigung der Stauraumbepackung berechnet und erst anschließend gepackt. Können die Kunden einer Tour nicht vollständig gepackt werden, sollen möglichst wenige Kunden von der Tour entfernt werden. Es muss also unabhängig davon, ob die gegebene 2OPP-Instanz vollständig gelöst werden konnte, eine Lösung mit allen Packstücken möglichst vieler Kunden ausgewiesen werden. Um das zu bewerkstelligen wird erwartet, dass mit jedem zu packenden Item i eine ganzzahlige Priorität $prio_i$ assoziiert ist, die

kennzeichnet, an welchem Stopp der Tour das Item entladen werden soll. Alle Items eines Kunden haben die gleiche Priorität; kleinere *prio*-Werte (sprich: größere Prioritäten) bedeuten frühere Positionen in der Tour. Die *prio*-Werte stellen nun das erste Sortierkriterium dar, und zwar werden die Items in eine nicht-absteigende Reihenfolge ihrer *prio*-Werte gebracht. Items mit gleichem *prio*-Wert werden gemäß der im letzten Absatz beschriebenen Kriterien $w_i h_i$ und $\min\{w_i, h_i\}$ geordnet. Falls ein Packstück j nicht im Container untergebracht werden kann, muss der Algorithmus nicht abgebrochen werden. Da nur eine ausreichend gute Lösung ausgewiesen werden soll und ohnehin kein Interesse an teilweise gepackten Kunden besteht, werden zunächst alle Packstücke k mit $prio_k = prio_j$ aus dem Container entfernt, um die von ihnen eingenommenen Plätze wieder freizugeben. Zudem werden im weiteren Verlauf des Algorithmus keine weiteren Packstücke mit der Priorität von j gepackt.

In Algorithmus 5 entscheidet der boolsche Eingabeparameter *findBestSolution* darüber, ob TP_{MOD} versucht, die Eingabeinstanz vollständig zu lösen (*findBestSolution* = falsch) oder nach einer möglichst guten Lösung mit atomar gepackten Kunden sucht (*findBestSolution* = wahr).

4.4.2 LIFO-Reihenfolge der Bepackung

Falls gewünscht, stellt TP_{MOD} sicher, dass die sich ergebende Bepackung bezüglich LIFO-Reihenfolge korrekt ist (siehe Definition 4.3.1). Hierzu werden die im letzten Absatz beschriebenen *prio*-Werte der Items herangezogen. Für jedes Paar (j, k) von Items, für das gilt, dass j früher auf der Tour entladen werden muss als k ($prio_j < prio_k$), muss sichergestellt sein, dass j beim Entladen des Containers nicht von k blockiert wird. Es muss also für alle Items aus P Eigenschaft 4.3 aus Definition 4.3.1 gelten.

Um Rechenzeit zu sparen, wird die LIFO-Bedingung im Zuge der Berechnung des Berührumfangs (touching perimeter, *tp*) überprüft, da diese Routine ohnehin über alle Items der aktuellen Bepackung P iterieren muss. Im Falle einer Verletzung der LIFO-Reihenfolge wird ein negativer *tp*-Wert zurückgegeben.

TP_{MOD} wurde gegenüber TP_{RF} an zwei weiteren Stellen modifiziert, um besser mit der Anforderung der LIFO-Reihenfolge der Bepackung umgehen zu können.

Die erste Anpassung betrifft die Art, wie die Items anfangs sortiert werden. Bei geforderter LIFO-Reihenfolge wird jedes Item i gemäß das Tupels $(prio_i, w_i h_i, \min\{w_i, h_i\})$ in die Menge der zu platzierenden Items einsortiert, wobei Items mit hohen *prio*-Werten – also geringer Priorität – an den Anfang kommen. Mit anderen Worten werden die Items verglichen mit der Sortierung ohne LIFO-Anforderung in umgekehrter Reihenfolge ihrer *prio*-Werte sortiert. Der Grund dafür ist, dass bei geforderter LIFO-Reihenfolge Packstücke von Kunden, die gegen Ende der Tour beliefert werden sollen (d.h. geringe Priorität haben), als erste gepackt werden müssen und solche, die zu Anfang der Tour entladen werden (d.h. hohe Priorität haben), als letzte gepackt werden müssen.

Die zweite Anpassung von TP_{MOD} betrifft die Auswahl der Platzierung eines Items im Container. Erstes Kriterium ist, wie auch im originalen Algorithmus TP_{RF}, der Wert *tp* des an jeder potentiellen Platzierung erzielten Berührumfangs. Die Modifikation betrifft Situationen, in denen für ein Item mehrere Platzierungen mit gleichem *tp*-Wert existieren. Eine solche Situation ist in Abbildung 4.2 zu sehen.

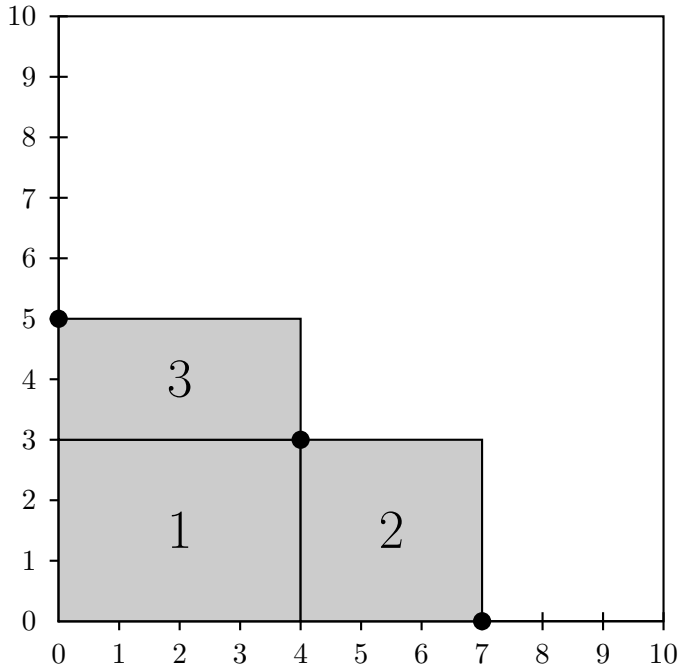


Abbildung 4.2: Potentielle Platzierungen für ein Item

Hier besteht das Packmuster bereits aus drei Items und ein weiteres, drehbares Item i mit $(w_i, h_i) = (1, 2)$ soll platziert werden. Dafür kommen die drei Orte $p_1 = (0, 5), p_2 = (4, 3), p_3 = (7, 0)$ und beide Orientierungen von i in Frage, denn alle Kombinationen würden für das Packstück i den gleichen tp-Wert von 3 ergeben. An Punkt p_1 würde i die linke Seitenfläche des Containers und die obere Kante von Item 3 berühren; an p_2 würde i die rechte Kante von Packstück 3 und die obere Kante von Packstück 2 berühren; an p_3 würde i die untere Containerwand und die rechte Seite von Packstück 2 berühren. Welche Platzierung und Orientierung sollte i also zugewiesen werden? Da der Container von oben be- und entladen wird, ist es sinnvoll, zunächst die unteren freien Plätze zu besetzen, denn falls diese in frühen Iterationen ungenutzt bleiben, kann es sein, dass sie später für Packstücke mit höherer Priorität aufgrund der Vorgabe der LIFO-Reihenfolge nicht mehr nutzbar sind. Des Weiteren scheint es intuitiv vorteilhaft zu sein, jedes Item i so zu orientieren, dass im Container möglichst viel Platz oberhalb von i verbleibt, dass also $y_i + h_i$ minimal ist. Aus diesen Überlegungen ergibt sich die Vorschrift für TP_{MOD} zur Auswahl der Platzierung eines Items:

Sei $C_P = \{(x_1, y_1), \ldots, (x_m, y_m)\}$ die Menge aller potentiellen Platzierungen eines Packmusters P und $O = \{normal, gedreht\}$ die Menge der möglichen Orientierungen eines Items. Sei $\Phi_P(i) \subseteq C_P \times O$ die Menge aller für das Item i zulässigen Kombinationen $x = (c, o), c \in C, o \in O$ und $\Phi'_P(i) \subseteq \Phi_P(i)$ die Auswahl derjenigen Kombinationen, für das Item i maximalen tp-Wert erreicht. Wähle nun aus $\Phi'_P(i)$ das Element $x^* = (c_{x^*}, o_{x^*})$ so, dass das Item i an Position c_{x^*} und mit der Orientierung o_{x^*} minimalen Wert $y_{c_{x^*}} + h_i$ erreicht. Platziere i an Position c_{x^*} mit Orientierung o_{x^*}.

Gemäß dieser Vorschrift würde in Abbildung 4.2 das Item i mit den Ausmaßen $(1, 2)$ an

Position $(7, 0)$ platziert werden, und zwar gedreht, d.h. mit $(w_i, h_i) = (2, 1)$.

4.4.3 Berechnung potentieller Platzierungen im Container

Die Frage, wie die Positionen für potentielle Platzierungen berechnet werden können, blieb in [Lodi, Martello, und Vigo 1999] unbeantwortet. Deshalb wurde für diesen Zweck eine eigene Routine entwickelt.

Beobachtung 4.4.1. Sei $\bar{X}(P) = \{0\} \cup \{x_i + w_i \mid i \in P\}$ die Menge der x-Koordinaten der rechten Enden aller Items aus P zuzüglich der linken Containerwand und $\bar{Y}(P) = \{0\} \cup \{y_i + h_i \mid i \in P\}$ die Menge der y-Koordinaten der oberen Enden aller Items aus P zuzüglich der unteren Containerwand. Dann ist das kartesische Produkt $\bar{X}(P) \times \bar{Y}(P)$ eine Obermenge der Menge C_P aller potentiellen Platzierungen für weitere Items.

Basierend auf dieser Beobachtung genügt es, in jeder Iteration die Menge $\bar{X}(P) \times \bar{Y}(P)$ von Positionen zu berechnen, um anschließend für das aktuelle Item jede der Positionen auszuprobieren und die beste zu wählen. Zudem kann der Algorithmus beschleunigt werden, indem die Menge T der im aktuellen Packmuster P bereits besetzten Positionen – sogenannter Tabu-Punkte – nicht für weitere Platzierungen berücksichtigt wird. Algorithmus 4 beschreibt das sich so ergebende Verfahren zur Berechnung normaler Positionen für ein Packmuster P.

Algorithmus 4: Algorithmus zur Berechnung der Itemplatzierungen für $\mathrm{TP}_{\mathrm{MOD}}$

```
1: procedure PLACEMENTS(P, T)
2:     X̄(P) ← {0} ∪ {x_i + w_i | i ∈ P}
3:     Ȳ(P) ← {0} ∪ {y_i + h_i | i ∈ P}
4:     for all x̄ ∈ X̄ do
5:         for all ȳ ∈ Ȳ do
6:             if (x̄, ȳ) ∉ T then
7:                 C_P ← C_P ∪ {(x̄, ȳ)}
8:     return C_P
```

Beobachtung 4.4.2. Wenn die Menge T der Tabu-Punkte in einer Hashtabelle mit erwarteter Look-Up-Zeit von $O(1)$ gehalten wird, hat Algorithmus 4 eine erwartete Laufzeit von $O(n^2)$.

4.4.4 Der Algorihmus TP_{MOD}

Nachdem erläutert wurde, wie die Menge der potentiellen Platzierungen für ein Item berechnet werden kann und nach welcher Vorschrift eine der Platzierungen ausgewählt wird, ist es leicht, im Folgenden die Heuristik TP_{MOD} zu beschreiben (Algorithmus 5).

Der Algorithmus erfordert die folgenden Parameter:

- V: Eine Liste von Kunden einer Tour in der Reihenfolge, wie sie in der Tour vorkommen. Insbesondere ist jedem Kunden eine Menge von rechteckigen Items zugeordnet, die im Container untergebracht werden sollen.

Algorithmus 5: TP_{MOD}

1: **procedure** $\text{TP}_{MOD}(V, W, H,$ *lifo*, *turnable*, *findBestSolution*$)$
2: Initialisiere Datenstrukturen: P, I, T
3: Füge alle zu platzierenden Items aus V in I ein
4: Sortiere alle $i \in I$ nach einer Vorschrift, die von der Belegung der Parameter *lifo* und *findBestSolution* abhängt.
5: *success* $\leftarrow$ wahr
6: **for all** $i \in I$ **do**
7: $C_P \leftarrow \textsc{Placements}(P, T)$ ▷ potenzielle Platzierungen
8: $tp^* \leftarrow -\infty,\ \bar{y}^* \leftarrow \infty$ ▷ Beste Lösung für i
9: $p^*, o^* \leftarrow$ **null** ▷ Position und Orientierung der besten Lösung
10: **for all** $p \in C_P$ **do**
11: Platziere Item i an Position (x_p, y_p)
12: **if** *turnable* **then**
13: $O \leftarrow \{normal, gedreht\}$
14: **else**
15: $O \leftarrow \{normal\}$
16: **for all** $o \in O$ **do**
17: Orientiere i gemäß dem Wert von o
18: **if** i ragt aus dem Container **or** i überlappt mit einem Item aus P **or** $P \cup \{i\}$ verletzt LIFO-Reihenfolge **then goto** 16
19: Berechne den Berührumfang tp von i
20: **if** $tp > tp^*$ **or** $(tp = tp^* \wedge y_p + h_i < \bar{y}^*)$ **then**
21: $tp^* \leftarrow tp$
22: $\bar{y}^* \leftarrow y_p + h_i$
23: $p^* \leftarrow p$
24: $o^* \leftarrow o$
25: **if** $tp^* > 0$ **then**
26: $p_i \leftarrow p^*,\ o_i \leftarrow o^*$
27: $P \leftarrow P \cup \{i\}$
28: $T \leftarrow T \cup p_i$
29: **else**
30: *success* $\leftarrow$ falsch
31: **if** $\neg$ *findBestSolution* **then**
32: **goto** 34
33: Entferne alle Items j mit $prio_j = prio_i$ aus P und aus I
34: **return** *success*

- W, H: Breite und Höhe des Containers.
- *lifo*: Boolscher Wert, der angibt, ob die LIFO-Reihenfolge eingehalten werden muss.
- *turnable*: Boolscher Wert, der angibt, ob die Items um 90° drehbar sind.
- *findBestSolution*: Boolscher wert, der festlegt, ob TP_{MOD} nach einer bestmöglichen Lösung suchen (wahr) oder nur feststellen soll, ob es die Eingabeinstanz lösen kann (falsch).

Zunächst werden die Datenstrukturen P (Liste bereits platzierter Items), I (Liste der zu platzierenden Items) und T (Hash-Tabelle der Tabu-Positionen) initialistert (Zeile 2). Anschließend werden die Items aus der Menge V aller Kunden in die Liste I eingefügt und diese wird gemäß der Belegung der Parameter *lifo* und *findBestSolution* sortiert (Zeile 4).

Nun werden die Items aus I einzeln abgearbeitet (Zeilen 6-33). In Zeile 7 wird für das aktuelle Packmuster P unter Berücksichtigung der Tabu-Positionen T die Menge der potentiellen Platzierungen für i berechnet. In den Zeilen 10-24 findet die Iteration über alle von *Placements* berechneten Positionen $p \in C_P$ statt, wobei das aktuelle Item gegebenenfalls in normaler und in rotierter Orientierung betrachtet wird (Zeilen 16-24). Nachdem das aktuelle Item an Position p platziert (Zeile 11) und gemäß der Orientierung o ausgerichtet wurde (Zeile 17), wird in Zeile 18 die Zulässigkeit dieser Platzierung überprüft. Gegebenenfalls kann nun die Größe tp des Berührumfangs berechnet werden (Zeile 19). In dem Fall, dass tp größer als der Berührumfang tp^* der bislang besten Lösung ist, oder $tp = tp^*$ und $y_p + h_i < \bar{y}^*$ (gleicher Berührumfang, aber Platzierung ist „tiefer“ im Container), wird die aktuelle Lösung als die bisher beste vorgemerkt (Zeilen 21-24). Falls für das Item i eine zulässige Platzierung existiert (Zeile 25), wird i gemäß dieser Lösung positioniert, orientiert und zum Packmuster P hinzugefügt; zudem wird die Platzierung p_i von i zur Menge T der Tabu-Positionen hinzugefügt. Falls für i keine zulässige Platzierung existiert, wird im Falle von *findBestSolution* = falsch die Prozedur abgebrochen (Zeile 32). Ansonsten wird das Verfahren fortgesetzt, nachdem alle Items der aktuellen Priorität $prio_i$ aus der Menge gepackter Items P und der Menge der noch zu packenden Items I entfernt wurden (Zeile 33).

Lemma 4.4.1. *Algorithmus* $\mathrm{TP}_{\mathrm{MOD}}$ *hat eine Laufzeit von* $O(n^4)$.

Beweis. Die Funktion *Placements* wird n mal aufgerufen, was bei guter Hash-Verteilung in T eine Laufzeit von $O(n^3)$ ergibt. Zur Überprüfung der Zulässigkeit einer Platzierung (Zeile 18 in Algorithmus 5) sowie die Berechnung des Berührumfangs tp für ein Item i (Zeile 19) genügt eine Iteration über alle Items aus P. Da *Placements* bis zu n^2 Positionen zurückliefert, werden die Zeilen 18 und 19 bis zu n^3 mal betreten, was somit eine Gesamtlaufzeit von $O(n^4)$ zur Folge hat. □

Es bleibt anzumerken, dass in [Lodi, Martello, und Vigo 1999] für das Verfahren TP_{RF} eine Laufzeit von $O(n^3)$ angegeben wurde, was auf die dort angegebene maximale Anzahl $O(n)$ von normalen Positionen zurückzuführen ist.

4.5 Exaktes Verfahren MV_{MOD} für die Laderaumoptimierung

Aufgrund der NP-Vollständigkeit des zweidimensionalen, orthogonalen Packproblems (2OPP) muss man bei exakten Algorithmen mit einer Laufzeit rechnen, die in der Anzahl zu packender Items exponentiell steigt. Trotzdem wurden bereits einige Algorithmen entwickelt, die das 2OPP exakt lösen.

[Martello und Vigo 1998] beschreiben ein solches Verfahren – von nun an *MV* genannt. Es handelt sich hierbei um ein enumeratives Branch-and-Bound-Verfahren, das alle möglichen *normalen Packmuster* durchprobiert, um auf diese Weise festzustellen, ob eine zulässige Bepackung der gegebenen Items möglich ist. Der Begriff *normales Packmuster* wurde zum ersten Mal von [Christofides und Whitlock 1977] verwendet.

Definition 4.5.1 (Normales Packmuster)**.** Ein zweidimensionales Packmuster P bezeichnet man als normal, wenn für jedes Item $i \in P$ gilt, dass eine Verschiebung von i entlang einer der Koordinatenachsen zum Ursprung hin zu einer Überlappung von Items oder zu einer Überschreitung der Containergrenzen führen würde.

Wenn also ein Packmuster eine beliebige, zulässige Anordnung von Items im Container bezeichnet (siehe Definition 4.0.1), so kann dieses Packmuster normalisiert werden, indem alle Items so weit wie möglich hin zur Koordinate $(0, 0)$ des Containers geschoben werden. In einer solchen Anordnung berührt jedes Item mit seiner linken Seite entweder die linke Seitenfläche des Containers oder die rechte Seitenfläche eines anderen, benachbarten Items. Entsprechend berührt jedes Item mit seiner unteren Seite entweder die untere Seitenfläche des Containers oder die obere Seitenfläche eines Nachbaritems.

Eine Vorschrift, aus einem beliebigen Packmuster P ein normales Packmuster $\hat{P}$ zu erzeugen, kann beispielsweise wie folgt aussehen:

1. Betrachte alle Items von P in nicht-absteigender Reihenfolge der x-Koordinaten ihrer Platzierungen im Container. Verschiebe jedes Item, beginnend mit dem ersten bezüglich dieser Reihenfolge, so weit wie möglich zur y-Achse.
2. Nun betrachte die Items in nicht-absteigender Reihenfolge der y-Koordinaten der Bepackung. Verschiebe jedes Item, beginnend mit dem ersten dieser Reihenfolge, so weit wie möglich zur x-Achse.
3. Wiederhole Schritte 1 und 2 so lange, bis kein Item mehr verschoben werden kann.

Lemma 4.5.1. *Gegeben sei eine deterministische Vorschrift zur Normalisierung von Packmustern. Dann definiert diese Vorschrift auf der Menge aller Packmuster eine Äquivalenzrelation. Dabei kommt in jeder Äquivalenzklasse genau ein normales Packmuster als Repräsentant vor.*

Beweis. Zu zeigen sind die Eigenschaften Reflexivität, Symmetrie und Transitivität:

Reflexivität: Der Determinismus der Vorschrift sorgt dafür, dass ein Packmuster immer in das gleiche normale Packmuster verwandelt wird, und somit in die gleiche Äquivalenzklasse fällt.

Symmetrie: Aufgrund der Tatsache gegeben, dass zwei äquivalente Packmuster immer auf das gleiche normale Packmuster abgebildet werden.

Transitivität: Falls die beiden Packmuster P_1 und P_2 auf das gleiche normale Packmuster $\hat{P}_1$ und die beiden Packmuster P_2 und P_3 auf das gleiche normale Packmuster $\hat{P}_2$ abgebildet werden, ergibt sich aus dem Determinismus der Abbildungsvorschrift und aus dem Vorkommen von P_2 in beiden Paaren, dass $\hat{P}_1 = \hat{P}_2$ und somit auch P_1 äquivalent zu P_3. □

Korollar 4.5.1. *Aus den obigen Überlegungen folgt, dass es für die exakte Lösung eines zweidimensionalen orthogonalen Packproblems genügt, alle normalen Packmuster zu enumerieren, denn jedes nicht-normale Packmuster fällt in eine der Äquivalenzklassen, die durch die normalen Packmuster definiert werden und wird somit implizit betrachtet.*

Im Rahmen dieser Diplomarbeit wurde das Verfahren MV_{MOD} entwickelt, das dem Algorithmus MV stark ähnelt, aber einige Anpassungen und Verbesserungen aufweist:

1. In jedem Suchknoten von MV wird eine Menge von Knotenpunkten benötigt, an denen weitere Items platziert werden können. Die Berechnung dieser Knotenmenge wird jedoch in [Martello und Vigo 1998] nicht näher spezifiziert, obwohl das eine nicht-triviale Aufgabe ist. Vielmehr wird diesbezüglich auf [Hadjiconstantinou und Christofides 1995] verwiesen; diese Quelle beschreibt das Verfahren jedoch ebenfalls nicht weiter. Aus diesem Grund wird in MV_{MOD} die Menge der möglichen Platzierungen so berechnet, wie es von [Martello, Pisinger, und Vigo 2000] als Subroutine eines Algorithmus zur Lösung des dreidimensionalen Bin-Packing-Problems beschrieben wird.

2. Um die Anzahl von Kinderknoten eines Branch-and-Bound-Suchknoten i zu reduzieren, setzt MV_{MOD} nicht jedes Item aus der Menge $U = I \setminus P$ der noch zu platzierenden Items an jede mögliche Position. Stattdessen wird U zunächst in Teilmengen („Item-Cluster“) von zueinander äquivalenten Items partitioniert, um anschließend in Suchbaumknoten nur ein Item aus jedem Cluster betrachten zu müssen. Dies trägt zu einer deutlichen Reduktion der Suchbaumgröße bei.

3. Es wird eine untere Schranke verwendet, um frühzeitig Äste des Suchbaums mit nicht lösbaren Teilpackungen zu erkennen und zu verlassen.

4. MV_{MOD} kann die LIFO-Reihenfolge der Bepackung beachten.

5. Die Items sind optional um 90° drehbar.

Wie auch MV enumeriert MV_{MOD} alle normalen Packmuster in einem Branch-and-Bound-Suchbaum. In jedem Knoten i des Suchbaumes gibt es das aktuelle Packmuster P_i und die Menge $U_i = I \setminus P_i$ der noch nicht platzierten Items. Das Branching erfolgt, indem gewisse Items aus U_i an zu P_i gehörenden *Eckpunkten* platziert werden, an denen sie nicht über die Containergrenzen hinausragen. Bevor jedoch die genaue Arbeitsweise von MV_{MOD} beschrieben werden kann (Algorithmus 8 auf Seite 60), werden einige Konzepte und Ideen erläutert, die für das Verständis der genauen Arbeitsweise des Algorithmus notwendig sind.

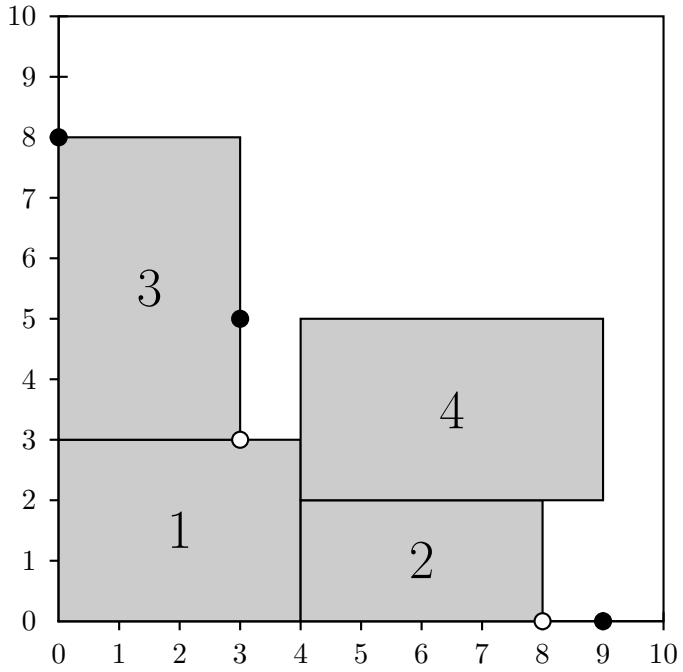


Abbildung 4.3: Normales Packmuster

Das Konzept der Eckpunkte (*Corner Points*) stammt aus [Martello, Pisinger, und Vigo 2000].

Definition 4.5.2 (Eckpunkte)**.** Mit Eckpunkten werden Punkte (x, y) im Container bezeichnet, die folgende Eigenschaften haben:

1. Es befindet sich kein gepacktes Item über und zugleich rechts neben dem Punkt (x, y), d.h. es gilt:
$$x \geq x_j + w_j \vee y \geq y_j + h_j \; \forall j \in P \tag{4.5}$$
2. Alle Punkte (x', y'), für die entweder $x' < x \wedge y' \leq y$ oder $y' < y \wedge x' \leq x$ gilt, sind keine Eckpunkte.

Es sei darauf hingewiesen, dass die Menge von Eckpunkten nicht gleichzusetzen ist mit der Menge derjenigen Punkte, die sowohl links als auch unten an ein platziertes Item oder eine Seitenfläche des Containers angrenzen. Dies soll in Abbildung 4.3 verdeutlicht werden, in der ein Packmuster bestehend aus vier Items zu sehen ist. Die gefüllten Kreise sind die sich ergebenden Eckpunkte; die nicht gefüllten Kreise sind hingegen keine Eckpunkte.

Lemma 4.5.3 zeigt, dass es genügt, in jedem Suchbaumknoten von MV_{MOD} nur die Eckpunkte als potentielle Platzierungskandidaten für Items zu betrachten. Zuvor wird aber noch die folgende Eigenschaft von Packmustern benötigt.

Lemma 4.5.2. *In jedem Packmuster P existiert mindestens ein Item, das weder über noch rechts neben sich ein weiteres Item hat:*

$$\forall P \exists j \in P : \forall \bar{j} \in P \setminus \{j\} : x_j \geq x_{\bar{j}} + w_{\bar{j}} \wedge y_j \geq y_{\bar{j}} + h_{\bar{j}} \tag{4.6}$$

Beweis. Diese Aussage wurde von [Martello, Pisinger, und Vigo 2000] für den dreidimensionalen Fall bewiesen. Dieser Beweis kann durch Vernachlässigung der dritten Dimension auch auf den zweidimensionalen Fall übertragen werden. □

Lemma 4.5.3. *Für jedes normale Packmuster $\hat{P}$ existiert ein Pfad im Suchbaum von* $\mathrm{MV}_{\mathrm{MOD}}$*, so dass $\hat{P}$ erreicht wird.*

Beweis. Der Beweis erfolgt induktiv. In jedem Packmuster P_k gibt es nach Lemma 4.5.2 mindestens ein Item j, das weder nach oben noch nach rechts von einem weiteren Item blockiert wird. Sei P_{k-1} das Packmuster, das entsteht, wenn Item j aus P_k entfernt wird. Insbesondere gilt für den Fall, in dem P_k ein normales Packmuster ist, dass die Platzierung (x_j, y_j) von j einem Eckpunkt in P_{k-1} entspricht. Man kann also von jedem Packmuster P_k zu dem leeren Packmuster $P_0 = \emptyset$ gelangen, indem in jedem Teilpackmuster $P_i, i \leq k$ das Item entfernt wird, welches die Eigenschaft aus Lemma 4.5.2 erfüllt und somit in einem Eckpunkt von P_{i-1} platziert ist.

Da MV_{MOD} in jedem Suchknoten alle Kombinationen aus Eckpunkten und zu platzierenden Items testet, gibt es also einen Pfad im Suchbaum, in dem die beschriebene Folge von Entfernungen in umgekehrter Reihenfolge beschritten wird. Hierbei entspricht jeder Entfernung eines Items j von Position (x_j, y_j) eine Einfügung von j an (x_j, y_j). Deshalb befindet sich jedes normale Packmuster $\hat{P}$ im Suchbaum von MV_{MOD}. □

Satz 4.5.1. $\mathrm{MV}_{\mathrm{MOD}}$ *ist ein korrektes Verfahren zur exakten Lösung des zweidimensionalen, orthogonalen Packproblems.*

Beweis. Die Korrektheit der Aussage folgt aus Korollar 4.5.1 und Lemma 4.5.3. □

4.5.1 Berechnung von Eckpunkten

Es bleibt zu klären, wie die Menge der Eckpunkte für ein gegebenes Packmuster berechnet werden kann. [Martello, Pisinger, und Vigo 2000] beschreiben den Algorithmus *2D-Corners*, der diese Aufgabe effizient löst. Im Rahmen dieser Arbeit wurde der Algorithmus *2D-Corners*$_{MOD}$ implementiert, der gegenüber *2D-Corners* eine Modifikation enthält: Statt zunächst alle Eckpunkte zu berechnen und sie anschließend daraufhin zu überprüfen, ob sie zulässige Platzierungen für mindestens eines der Items aus U_i darstellen, werden hier solche unzulässigen Platzierungen direkt verworfen.

Algorithmus 6 beschreibt die Arbeitsweise von *2D-Corners*$_{MOD}$. Als Eingabe werden ein Packmuster P, die Ausmaße (W, H) des Containers sowie die minimale Breite $w_{min} = \min_{i \in U} w_i$ und Höhe $h_{min} = \min_{i \in U} h_i$ über alle Elemente aus der Menge U der ungepackten Items benötigt. Genau wie in *2D-Corners*, müssen in *2D-Corners*$_{MOD}$ die Items in der Eingabemenge P nach ihren Endpunkten $(x + w, y + h)$ so sortiert sein, dass die Werte $y + h$ der Items nicht aufsteigend sind und Elemente mit gleich großer y-Komponente nicht absteigend nach ihrer x-Komponente geordnet sind.

Definition 4.5.3 (Extreme Items)**.** Für jedes extreme Item e eines Packmusters P gilt, dass sich kein Item aus $P \setminus \{e\}$ über oder rechts neben der oberen rechten Ecke von e befindet. Anders ausgedrückt gilt für die Koordinate $(x_e + w_e, y_e + h_e)$ jedes extremen Items e die folgende Eigenschaft:

Algorithmus 6: 2D-Corners$_{\text{MOD}}$

Vorbedingung: Die Items in P sind nach ihren Endkoordinaten (x_j+w_j, y_j+h_j) sortiert, so dass die Werte $y_j + h_j$ nicht aufsteigend sind und bei Gleichheit der größere Wert $x_j + w_j$ gewinnt.

```
1:  procedure 2D-CORNERS_MOD(P, W, H, w_min, h_min)
2:      C ← ∅
3:      if P = ∅ then
4:          return {(0, 0)}
                                  ▷ Schritt 1: Identifiziere extreme Items E = {e_1, ..., e_m}
5:      x̄ ← −∞
6:      m ← 0
7:      for all i ∈ P do
8:          if x_i + w_i > x̄ then
9:              x̄ ← x_i + w_i
10:             m ← m + 1
11:             e_m ← i
                                  ▷ Schritt 2: Bestimme Eckpunkte
12:     if y_{e_1} + h_{e_1} + h_min ≤ H then
13:         C ← {(0, y_{e_1} + h_{e_1})}
14:     for i ← 2 to m do
15:         if x_{e_{i−1}} + w_{e_{i−1}} + w_min ≤ W ∧ y_{e_i} + h_{e_i} + h_min ≤ H then
16:             C ← C ∪ {(x_{e_{i−1}} + w_{e_{i−1}}, y_{e_i} + h_{e_i})}
17:     C ← C ∪ {(x_{e_m} + w_{e_m}, 0)}
18:     return C
```

$$x_e + w_e \geq x_j + w_j \vee y_e + h_e \geq y_j + h_j \; \forall j \in P \setminus \{e\} \tag{4.7}$$

In Abbildung 4.3 sind 3 und 4 extreme Items.

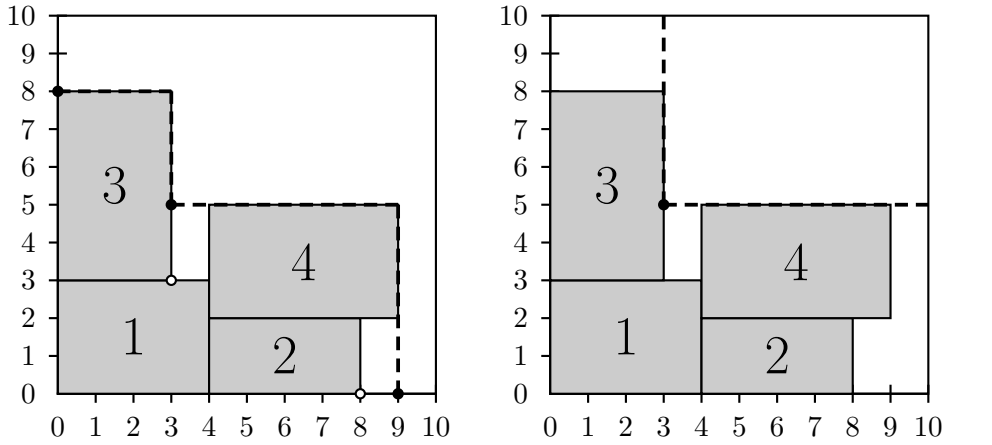
Der Algorithmus arbeitet in zwei Phasen, wobei die erste Phase (Zeilen 5-11) dazu dient, die Menge E der extremen Items $e_1, \ldots, e_m$ zu berechnen. Hierzu werden alle Items des Packmusters P von oben nach unten (d.h. in nicht aufsteigender Reihenfolge der Werte $y+h$) und bei Gleichheit der y-Komponente von rechts nach links (d.h. in nicht aufsteigender Reihenfolge der Werte $x + w$) durchlaufen. Für diese Reihenfolge sorgt die geforderte Sortierung von P. Items mit bis dahin maximalem Wert $(x + w, y + h)$ werden zu E hizugefügt. In der zweiten Phase (Zeilen 12-17) werden die Eckpunkte bestimmt. Hierzu genügt ein Durchlauf durch die nach der y-Koordinate aufsteigend sortierten Items der Menge E (Zeilen 14-16). Für zwei aufeinander folgende extreme Items e_{i-1}, e_i wird der Eckpunkt $(x_{e_{i-1}} + w_{e_{i-1}}, h_{e_i} + h_{e_i})$ gebildet (Zeile 16). Der Eckpunkt wird verworfen, falls ohnehin kein Item an dieser Position platziert werden kann ohne aus dem Container herauszuragen. Da das erste Element aus E keinen Vorgänger hat, wird die x-Koordinate des entsprechenden Eckpunktes auf den Wert 0 gesetzt (Zeile 13). Da das letzte Element aus E keinen Nachfolger hat, wird die y-Koordinate des entsprechenden Eckpunktes auf 0

gesetzt (Zeile 17).

Beobachtung 4.5.1. *2D-Corners*$_{MOD}$ benötigt für die Berechnung der Eckpunkte eine Laufzeit von $O(n)$.

4.5.2 Untere Schranke

Die Eckpunkte definieren eine Hülle (envelope) um die gegebene Bepackung, die den Bereich, in dem Items platziert werden können, von dem Bereich abgrenzt, in dem dies nicht mehr möglich ist. In Abbildung 4.4(a) liegt diese Hülle unterhalb der gestrichelten Linie.



(a) Durch Eckpunkte erzeugte Hülle

(b) Vergrößerte Hülle aufgrund der Eliminierung von Eckpunkten

Abbildung 4.4: Hülle einer Bepackung

Es fällt auf, dass es durch Items unbesetzte Bereiche geben kann, die innerhalb der Hülle liegen. Diese Bereiche können jedoch unterhalb des aktuellen Suchbaumknotens nicht mehr erschlossen werden, denn Items werden nie unter der Demarkationslinie platziert, die durch die Eckpunkte definiert ist.

Diese Tatsache ermöglicht die schnelle Berechnung der unteren Schranke L'_0 für die Anzahl benötigter Container. L'_0 entspricht der Continuous Lower Bound L_0 (siehe Abschnitt 4.7), stärkt diese aber um die ungenutzten Bereiche innerhalb der Hülle. Seien A' die Größe der ungenutzten Fläche innerhalb der Hülle, (W, H) die Ausmaße des Containers, P das aktuelle Packmuster und U die Menge bislang unplatzierter Items. Dann lässt sich die neue untere Schranke als

$$L'_0 = \left\lceil \frac{A' + \sum_{j \in P} w_j h_j + \sum_{j \in U} w_j h_j}{W \times H} \right\rceil \tag{4.8}$$

beschreiben. Wenn in einem Knoten des Suchbaums der Wert der unteren Schranke L'_0 größer eins ist, kann dieser Branch erfolglos verlassen werden.

Für die Berechnung der unteren Schranke kann die Tatsache genutzt werden, dass die Summe $A' + \sum_{j \in P} w_j h_j$ gerade der Gesamtfläche der Hülle entspricht. Algorithmus 7

berechnet für eine Menge von Eckpunkten diese Fläche in Zeit $O(n)$. Als Parameter werden die Menge C von Eckpunkten sowie die Höhe W und Breite H des Containers übergeben.

Algorithmus 7: EnvelopeArea

Vorbedingung: Die Eckpunkte in der Menge C sind aufsteigend nach ihren x-Koordinaten sortiert.

```
1: procedure EnvelopeArea(C, W, H)
2:     a, x̄ ← 0
3:     ȳ ← H
4:     for all c ∈ C do
5:         a ← a + ȳ(x_c − x̄)
6:         x̄ ← x_c
7:         ȳ ← y_c
8:     a ← a + ȳ(W − x̄)
9:     return a
```

Die Vorbedingung ist deshalb erfüllt, weil *2D-Corners*$_{MOD}$ die Eckpunkte bereits in der benötigten Reihenfolge ausgibt.

Eine Stärkung der unteren Schranke erhält man dadurch, dass *2D-Corners*$_{MOD}$ all diejenigen Eckpunkte verwirft, an denen keine weiteren Items platziert werden können. Man betrachte z.B. die Situation, in der das Packmuster und die Hülle den in Abbildung 4.4(a) dargestellten entsprechen und die Menge U der zu platzierenden Items nur noch Items j enthält, die alle mindestens 2 Längeneinheiten breit und 3 Längeneinheiten hoch sind. Es kann also weder im Eckpunkt $(0, 8)$ noch in $(9, 0)$ ein Item aus U platziert werden. Durch Eliminierung dieser Eckpunkte kommen weitere ungenutzte Flächen unter die Demarkationslinie der Hülle, die somit an Fläche zunimmt. Abbildung 4.4(b) stellt die Situation nach Entfernung der Eckpunkte $(0, 8)$ und $(9, 0)$ dar. Da *2D-Corners*$_{MOD}$ Eckpunkte, an denen keine weiteren Items platziert werden können, von vornherein verwirft (dafür sorgt die Bedingung in Zeile 15 in Algorithmus 6), müssen diese jedoch nicht im Nachhinein eliminiert werden.

4.5.3 LIFO-Reihenfolge der Bepackung

Falls gewünscht, generiert MV_{MOD} nur soche Bepackungen, die bezüglich LIFO-Reihenfolge korrekt sind (siehe Definition 4.3.1). Hierzu wird beim Platzieren jedes Items $j \in U$ in ein Packmuster P überprüft, ob für jedes Paar $(j, k),\ k \in P$ von Items die Eigenschaft (4.3) (Seite 43) erfüllt ist. Da hierzu eine Iteration über alle Items aus P genügt, beläuft sich die zusätzliche Laufzeit auf $O(n)$ pro Verzweigung im Suchbaum.

4.5.4 Drehbarkeit der Items

Falls die zu platzierenden Items um 90° drehbar sein sollen, versucht MV_{MOD} jedes Item sowohl in normaler als auch in rotierter Position in den jeweiligen Eckpunkten unterzubringen. Dies erfordert einen konstanten Mehraufwand pro Suchbaumknoten, denn es müssen

höchstens doppelt so viele Platzierungen überprüft werden wie es ohne Drehbarkeit der Fall ist. Für Items j mit $w_j = h_j$ wird die Drehbarkeit vernachlässigt.

4.5.5 Clustering von Items

Die Idee, Items zu klassifizieren und in äquivalente Mengen („Cluster“) $\hat{i}_1, \ldots, \hat{i}_m$ zu verteilen resultiert aus der Überlegung, dass es bei einem gegebenen Packmuster P nicht notwendig ist zwei Items j und k mit identischen Ausmaßen und gleicher Priorität beide an einem Eckpunkt c zu platzieren. Falls die Platzierung von j in c zu keiner gültigen Lösung des Gesamtproblems führt, wird auch die Platzierung von k in c kein anderes Ergebnis erzeugen. Deshalb wird in einem Suchbaumknoten i von MV_{MOD} nicht jedes Item aus der Menge U_i mit allen Eckpunkten kombiniert, sondern nur ein Repräsentant aus jedem der Cluster $\hat{i}_1, \ldots, \hat{i}_m$.

4.5.6 Der Algorithmus MV_{MOD}

Im Folgenden wird eine detaillierte Beschreibung der Arbeitsweise von MV_{MOD} gegeben. Diese wird von Pseudocode 8 ergänzt.

Der Algorithmus erfordert die folgenden Parameter:

- V: Eine Liste von Kunden einer Tour in der Reihenfolge, wie sie in der Tour vorkommen. Insbesondere ist jedem Kunden eine Menge von zweidimensionalen, rechteckigen Items zugeordnet, die im Container untergebracht werden sollen.
- W, H: Breite und Höhe des Containers.
- *lifo*: Boolscher Wert, der angibt, ob die LIFO-Reihenfolge eingehalten werden muss.
- *turnable*: Boolscher Wert, der angibt, ob die Items um 90° drehbar sind.
- *timeout*: Zeit in Millisekunden, nach der der Algorithmus die Berechnung abbricht, falls bis dahin keine vollständige Bepackung gefunden werden konnte.
- *findBestSolution*: Boolscher Wert, der festlegt, ob MV_{MOD} in jedem Suchbaumknoten die Qualität der aktuellen Lösung mit der bislang besten vergleichen und die beste Lösung zwischenspeichern soll. Auch wenn die Instanz nicht gelöst werden kann, kann so die beste gefundene Lösung später abgefragt werden.

Zunächst werden die globalen Variablen W, H, *lifo*, *turnable* und *timeout* gesetzt (Zeile 2) und die folgenden Datenstrukturen initialisiert (Zeile 3):

- P: Liste von bereits platzierten Items (das Packmuster).
- *wc*: Datenstruktur zur Beantwortung der Frage nach der Anzahl von ungepackten Items, die eine bestimmte Breite haben. Insbesondere benötigt MV_{MOD} die Information, welche Breite der Items minimal ist. Für diesen Zweck wird eine `java.util.TreeMap` verwendet, wobei als Schlüssel die Breite und als Wert die Anzahl Items mit der gegebenen Breite verwendet werden. Alle hier verwendeten Operationen (`get`, `put`, `removeKey`, `firstEntry`) werden von der `java.util.TreeMap` in Zeit $O(\log n)$ durchgeführt.

Algorithmus 8: MV_{MOD}

```
1: procedure MV_MOD(V, W, H, lifo, turnable, timeout, findBestSolution)
2:     Setze globale Variablen: W, H, lifo, turnable, timeout
3:     Initialisiere globale Datenstrukturen: P, wc, hc, Î
4:     Sortiere Î nach Entladepriorität, Fläche, Breite gemäß der Belegung von lifo und findBest-
       Solution
5:     Berechne Fläche A_I aller Items in I
6:     DeterminePlacements

7: procedure DeterminePlacements
8:     if keine weiteren Items then
9:         return wahr
10:    if Timeout überschritten then
11:        return falsch
12:    w_min ← firstEntry(wc); h_min ← firstEntry(hc)
13:    C ← 2D-Corners_MOD(P, W, H, w_min, h_min)                    ▷ Berechne Eckpunkte
14:    if ¬findBestSolution then
15:        A ← EnvelopeArea(C, W, H)
16:        if Fläche der noch nicht gepackten Items > WH − A then    ▷ Untere Schranke
17:            return falsch
18:    if turnable then
19:        O ← {normal, gedreht}
20:    else
21:        O ← {normal}
22:    for all o ∈ O do
23:        for c ← |C| downto 1 do   ▷ Iteriere die Eckpunkte nach aufsteigender y-Koordinate
24:            for all î ∈ Î do                                     ▷ Iteriere über alle Item-Cluster
25:                i ← getFirst(î)                                  ▷ Betrachte einen Repräsentanten aus î
26:                if o = gedreht then                                                  ▷ Drehe Item
27:                    if w_i = h_i then                                    ▷ Item ist quadratisch
28:                        goto 24                                                  ▷ Nächstes Item
29:                    Drehe i um 90°
30:                if x_c + w_i > W or y_c + h_i > H then          ▷ Item ragt aus dem Container
31:                    goto 24                                              ▷ Nächstes Item
32:                Setze i an Position c
33:                if lifo and P ∪ {i} verletzt LIFO-Reihenfolge then
34:                    goto 24                                              ▷ Nächstes Item
35:                removeFirst(î)
36:                P ← P ∪ {i}
37:                decrement(i, wc); decrement(i, hc)
38:                success ← DeterminePlacements                ▷ Rekursiver Aufruf
39:                if success then
40:                    return wahr
41:                Entferne i aus P und füge es wieder in î ein.
42:                increment(i, wc); increment(i, hc)
43:    return falsch
```

- *hc*: Zu *wc* analoge Datenstruktur für die Höhe der Items.
- $\hat{I}$: Liste von Item-Clustern $\hat{i}_1, \ldots, \hat{i}_m$. Alle Items eines Clusters haben die gleiche Entladepriorität $prio_i$ und identische Ausmaße (w_i, h_i). Die Cluster in $\hat{I}$ werden lexikografisch sortiert (Zeile 4) nach den folgenden drei Eigenschaften:

  1. Entladepriorität (optional)
  2. Fläche $w_i \times h_i$ der Items im Cluster (absteigend)
  3. Breite w_i der Items (absteigend)

  Ob die Reihenfolge der Entladeprioritäten der Item-Cluster absteigend oder aufsteigend ist, oder ob die Entladepriorität gar nicht als Kriterium herangezogen wird, hängt von den Werten der Parameter *lifo* und *findBestSolution* ab. Es werden hierbei die gleichen Regeln angewandt, wie für die Vorsortierung der Packstücke im Algorithmus TP_{MOD}. Siehe Abschnitt 4.4 für eine ausführliche Beschreibung.

Die Gesamtfläche A_I aller zu platzierenden Items, die in Zeile 5 berechnet wird und mit jedem Hinzufügen eines Items zu P und Entfernen eines Items aus P aktualisiert werden muss, wird für die Berechnung der unteren Schranke benötigt.

Nach der Initialisierung erfolgt in Zeile 6 der Aufruf der rekursiven Subroutine *DeterminePlacements*. *DeterminePlacements* überprüft zunächst die Abbruchkriterien „alle Items gepackt“ und „Timeout überschritten“ (Zeilen 8-11). Nun werden mit Hilfe der Datenstrukturen *wc* und *hc* die minimale Breite und Höhe aller noch ungepackten Items ermittelt (Zeile 12), welche als Parameter zur Berechnung der Eckpunkte benötigt werden (Zeile 13). Mit Hilfe der Eckpunkte wird die Fläche der Packmusterhülle berechnet (Zeile 15), die wiederum für die Berechnung der unteren Schranke benötigt wird. Ob die untere Schranke berechnet wird, hängt von der Belegung des Parameters *findBestSolution* ab. Falls dieser Parameter den Wert „wahr“ hat, ist das primäre Ziel, eine möglichst gute Lösung zu finden, selbst wenn die Instanz nicht gelöst werden kann. In diesem Fall wird die untere Schranke nicht eingesetzt, da durch ihren Einsatz Äste des Suchbaums abgeschnitten werden könnten, die zu einer guten Lösung geführt hätten.

Abhängig davon, ob die Items drehbar sind oder nicht, wird in den Zeilen 18-21 eine Liste O von Orientierungen der Länge 2 oder 1 erstellt, über welche dann iteriert wird (Zeilen 22-42). Innerhalb dieser Schleife wird jede Kombination aus Item-Cluster und Eckpunkt betrachtet. In Zeile 25 wird das erste Item aus dem aktuellen Cluster $\hat{i}$ der Variablen i zugewiesen und gedreht, falls es nicht ohnehin quadratisch ist (Zeile 29). Falls das Item am gegebenen Eckpunkt nicht die Containergrenzen überschreitet, wird seine Platzierung festgelegt (Zeile 32). In Zeile 33 wird $P \cup \{i\}$ auf Korrektheit der LIFO-Reihenfolge überprüft. Falls diese eingehalten ist, wird das Item aus seinem Cluster entfernt (Zeile 35) und mit gegebener Platzierung und Orientierung zum Packmuster hinzugefügt (Zeile 36). In Zeile 37 werden die Datenstrukturen *wc* und *hc* aktualisiert. Im Falle von *wc* bedeutet das einen Lookup mit der Itembreite w_i als Schlüssel und eine Dekrementierung des so erhaltenen Wertes. Falls letzterer nach der Dekrementierung gleich 0 ist, wird der Schlüssel w_i aus der Datenstruktur entfernt. Im Falle von *hc* wird die gleiche Operation mit der Höhe h_i als Schlüssel durchgeführt. Falls die Items drehbar sind, muss die *decrement*-Operation das Lookup in beiden Datenstrukturen mit $\min\{w_i, h_i\}$ als Schlüssel durchführen. Anschließend erfolgt in Zeile 38 ein rekursiver Aufruf von *DeterminePlacements*. Falls dieser

erfolgreich ist, wird der Suchbaumknoten mit positivem Rückgabewert verlassen. Ansonsten wird i wieder aus dem Packmuster P entfernt und zum Cluster i hinzugefügt (Zeile 41). Zudem müssen die Datenstrukturen wc und hc aktualisiert werden, indem jeweils der Wert, der durch den Schlüssel w_i (im Falle von wc) bzw. h_i (im Falle von hc) bzw. $\min\{w_i, h_i\}$ (in beiden Datenstrukturen, falls die Items drehbar sind) inkrementiert wird. Falls der Eintrag mit dem gegebenen Schlüssel nicht existiert, wird er mit dem Wert 1 hinzugefügt. Nach erfolgloser Überprüfung aller Kombinationen aus Item, Eckpunkt und Orientierung wird der Suchbaumknoten mit negativem Rückgabewert verlassen (Zeile 43).

Lemma 4.5.4. *Sei P ein normales Packmuster bestehend aus n Items. Dann gibt es in P höchstens $n + 1 = O(n)$ Eckpunkte.*

Beweis. Das Leere Packmuster $P = \emptyset$ enthält genau einen Eckpunkt. Durch Hinzufügen eines Items i zu einem normalen Packmuster wird genau ein Eckpunkt entfernt (derjenige, an dem i platziert wurde) und es entstehen höchstens zwei neue Eckpunkte. □

Satz 4.5.2. $\mathrm{MV}_{\mathrm{MOD}}$ *hat eine Laufzeit von $O(n^{2n+3})$.*

Beweis. Da es zu jedem Zeitpunkt höchstens n ungepackte Items, zwei Orientierungen für jedes Item und $n + 1$ Eckpunkte gibt, erzeugt jeder Suchbaumknoten nicht mehr als $2n(n+1) = O(n^2)$ Kinderknoten. Da auf jeder Ebene des Baumes ein Item platziert wird, gibt es höchstens $n + 1$ Ebenen (mit der Wurzel) und somit insgesamt nicht mehr als $(2n(n + 1))^{n+1} = O((n^2)^{n+1}) = O(n^{2n+2})$ Suchbaumknoten. Die aufwändigsten Operationen in einem Knoten sind das Berechnen der Eckpunkte sowie die Überprüfung der LIFO-Reihenfolge. Beide können in Zeit $O(n)$ durchgeführt werden. Somit ergibt sich eine Gesamtlaufzeit von $O(n \times n^{2n+2}) = O(n^{2n+3})$. □

4.6 Exaktes Verfahren $\mathrm{LMAO}_{\mathrm{MOD}}$ für die Laderaumoptimierung

Das in Abschnitt 4.5 vorgestellte Verfahren MV_{MOD} enumeriert in einem Suchbaum alle normalen Packmuster. Ein großes Problem des Verfahrens besteht darin, dass unter Umständen gewisse Packmuster und – von diesen Packmustern ausgehend – ganze Äste des Suchbaums mehrfach untersucht werden, was sich sehr stark auf die Laufzeit auswirken kann. Die Ursache hierfür liegt darin, dass MV_{MOD} (wie auch der ursprünglich von [Martello und Vigo 1998] vorgestellte Algorithmus MV) an jedem Suchbaumknoten für jede Kombination aus Eckpunkt und zu verstauendem Packstück eine Verzweigung (Branch) bildet. Dies soll an einem Beispiel verdeutlicht werden.

Angenommen, es gibt an einem Knoten in der i-ten Suchbaumebene $k \geq 2$ Eckpunkte und $l \geq 2$ zu platzierende Items. Der Algorithmus platziert das erste Item am ersten Eckpunkt und verzweigt in die Suchbaumebene $i + 1$. Hier wird nun das zweite Item am zweiten Eckpunkt platziert, die Bepackung stellt sich jedoch im weiteren Verlauf als nicht lösbar heraus. Durch Backtracking gelangen wir zurück zum Ausgangsknoten in Ebene i, wo irgendwann Item zwei an Eckpunkt zwei platziert wird. Am Nachfolgeknoten in Ebene $i + 1$ wird nun Item eins an Eckpunkt eins platziert, was zum zweiten Mal die gleiche Situation ergibt, nämlich:

- Item 1 befindet sich in Eckpunkt 1,
- Item 2 befindet sich in Eckpunkt 2 und
- alle Items $3, \ldots, l$ sind unplatziert.

[Clautiaux, Carlier, und Moukrim 2007] stellen den Algorithmus *LMAO* vor, der *MV* dahingehend verbessern soll, dass mehrfache Betrachtungen von Packmustern von vornherein ausgeschlossen werden. Auf diese Weise soll die Suchbaumgöße und damit auch die Laufzeit stark verringert werden. Die entscheidende Modifikation gegenüber *MV* besteht darin, dass in einem Suchbaumknoten nur noch genau eine Position im Container als Kandidat betrachtet wird, um dort ein Item zu platzieren (oder aber um kein Item dort zu platzieren). Falls die Platzierung jedes der unplatzierten Items in dieser festgelegten Position zu keiner Lösung des Gesamtproblems führt, wird diese Position als inaktiv markiert und verbleibt dies innerhalb des gesamten durch den aktuellen Knoten initiierten Teilbaums. Bei k Positionen und l zu platzierenden Items entstehen somit nur noch $l + 1 = O(n)$ Verzweigungen, gegenüber $lk = O(n^2)$ Verzweigungen, die *MV* in derselben Situation erzeugen würde.

Die Wahl der Position erfolgt in *LMAO* nach einer festgelegten Vorschrift, nämlich dem sogenannten „leftmost-downward“-Kriterium. Es wird stets diejenige noch verfügbare Platzierungsposition eines Packmusters gewählt, welche die kleinste y-Koordinate unter allen Positionen mit minimaler x-Koordinate hat. [Clautiaux, Carlier, und Moukrim 2007] zeigen, dass *LMAO* jedes normale Packmuster durch eine eindeutige Abfolge von Item-Platzierungen erreicht, wenn jedes Item jeweils in der „leftmost-downward“-Position platziert wird. Daraus folgt, dass das Verfahren ein Packmuster niemals mehr als einmal enumeriert.

Das im Rahmen dieser Arbeit implementierte Verfahren $LMAO_{MOD}$ basiert auf den wichtigsten Ideen von *LMAO*, bringt aber gewisse Modifikationen mit sich, die angesichts der an den Algorithmus gestellten Anforderungen notwendig sind oder zumindest sinnvoll erscheinen. Zu diesen Modifikationen gehören:

1. Die optionale Überprüfung der LIFO-Reihenfolge der Bepackung,
2. die optionale Drehbarkeit der Items um 90°,
3. Clustering von gleichartigen Items,
4. eine eigene Methode zur Berechnung der nächsten Platzierungsposition und
5. eine untere Schranke, um frühzeitig nicht lösbare Äste des Suchbaums zu erkennen und zu verlassen.

Die Punkte 1–3 wurden so umgesetzt wie im Algorithmus MV_{MOD} und wurden bereits ausführlich in Abschnitt 4.5 behandelt. Es werden daher im Folgenden die Punkte 4 und 5 näher beleuchtet. Es sei angemerkt, dass sich die in Punkt 5 genannte untere Schranke nach näherer Untersuchung als nicht zulässig erwiesen hat und deshalb nicht angewendet werden kann.

4.6.1 Berechnung von Platzierungen

Wie bereits erwähnt wurde, wird im ursprünglichen Algorithmus *LMAO* für die nächsten Position unter den aktiven, d.h. unter den nicht als inaktiv markierten Positionen diejenige gewählt, die

1. minimale x-Koordinate (primäres Kriterium) und
2. minimale y-Koordinate (sekundäres Kriterium)

hat. Dieses Auswahlkriterium wird von [Clautiaux, Carlier, und Moukrim 2007] als „leftmost-downward"-Kriterium bezeichnet.

Für $LMAO_{MOD}$ wird dieses Kriterium geändert, indem die Bedeutung der beiden Koordinatenachsen vertauscht wird. Mit anderen Worten wählt $LMAO_{MOD}$ als nächste Platzierung die Position, deren x-Koordinate minimal ist aus der Menge der Punkte mit minimaler y-Koordinate. Wenn der Container also im ersten Quadranten des Koordinatensystems liegt, wird er zunächst „von links nach rechts" und dann „von unten nach oben" gepackt. In Anlehnung an die von Clautiaux et al. gewählte Bezeichnung „LMD-Eckpunkt" (leftmost-downward) werden von nun an das modifizierte Auswahlkriterium als „RLMD-Kriterium" (reverse leftmost-downward) und die nach diesem Kriterium gewählten Positionen als „RLMD-Eckpunkte" bezeichnet.

Diese Änderung wird durchgeführt, weil $LMAO_{MOD}$ gegebenenfalls die LIFO-Reihenfolge der Bepackung einhalten muss. Auch wenn beide Reihenfolgen prinzipiell zum gleichen Ergebnis führen (alle Packmuster werden korrekt enumeriert), erscheint es sinnvoll, die „umgekehrte leftmost-downward"-Reihenfolge anzuwenden, da diese eher der Vorgehensweise beim tatsächlichen Beladen eines Transportfahrzeugs unter Berücksichtigung der LIFO-Reihenfolge entspricht. Ein Mensch würde höchstwahrscheinlich zunächst alle Packstücke des zuletzt zu beliefernden Kunden vorne im LKW unterbringen, dann alle Packstücke des vorletzten Kunden der Tour direkt dahinter usw. Diese Vorgehensweise ahmt $LMAO_{MOD}$ durch die Verwendung des RLMD-Kriteriums nach.

Es bleibt immer noch die Frage, wie der jeweils nächste zulässige RLMD-Eckpunkt effizient berechnet werden kann. Leider wird diese Frage in [Clautiaux, Carlier, und Moukrim 2007] nicht beantwortet. Vielmehr wird hier auf die Methode verwiesen, die in [Martello und Vigo 1998] für den Algorithmus MV verwendet wird. Da jedoch auch in dieser Quelle keine weitere Beschreibung des Verfahrens zur Berechnung von Eckpunkten angegeben ist, erscheint es zunächst naheliegend, den Algorithmus *2D-Corners*$_{MOD}$ (Algorithmus 6, Seite 55) zu verwenden, welcher sich für MV_{MOD} als schnell und korrekt erwiesen hat. Leider stellt sich *2D-Corners*$_{MOD}$ im Zusammenhang mit *LMAO* bzw. $LMAO_{MOD}$ als unzulänglich heraus. Um dies zu verdeutlichen, sei das folgende Beispiel aufgeführt:

Gegeben sei ein Container mit Ausmaßen $(W, H) = (10, 10)$ und die Itemmenge $I = \{1, \ldots, 5\}$ mit Ausmaßen $(w_i, h_i) = (6, 2), (4, 7), (3, 8), (3, 5), (7, 3)$. Bis auf Drehungen in 90°-Schritten und Spiegelungen stellt Abbildung 4.5 die einzige Möglichkeit dar, wie die dargestellten Items im gegebenen Container untergebracht werden können. MV_{MOD} hätte unter Verwendung von *2D-Corners*$_{MOD}$ keine Probleme, das Packmuster zu finden. Die Platzierungsreihenfolge der Items wäre 1, 3, 4, 2, 5.

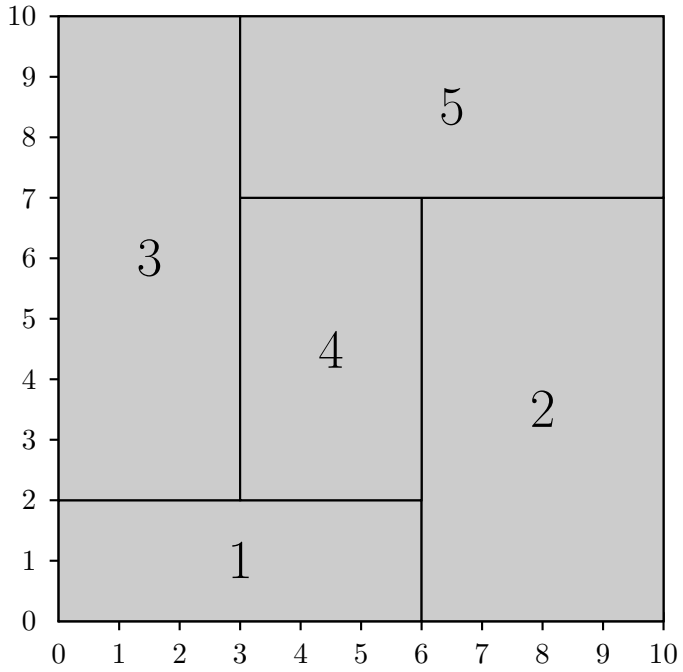


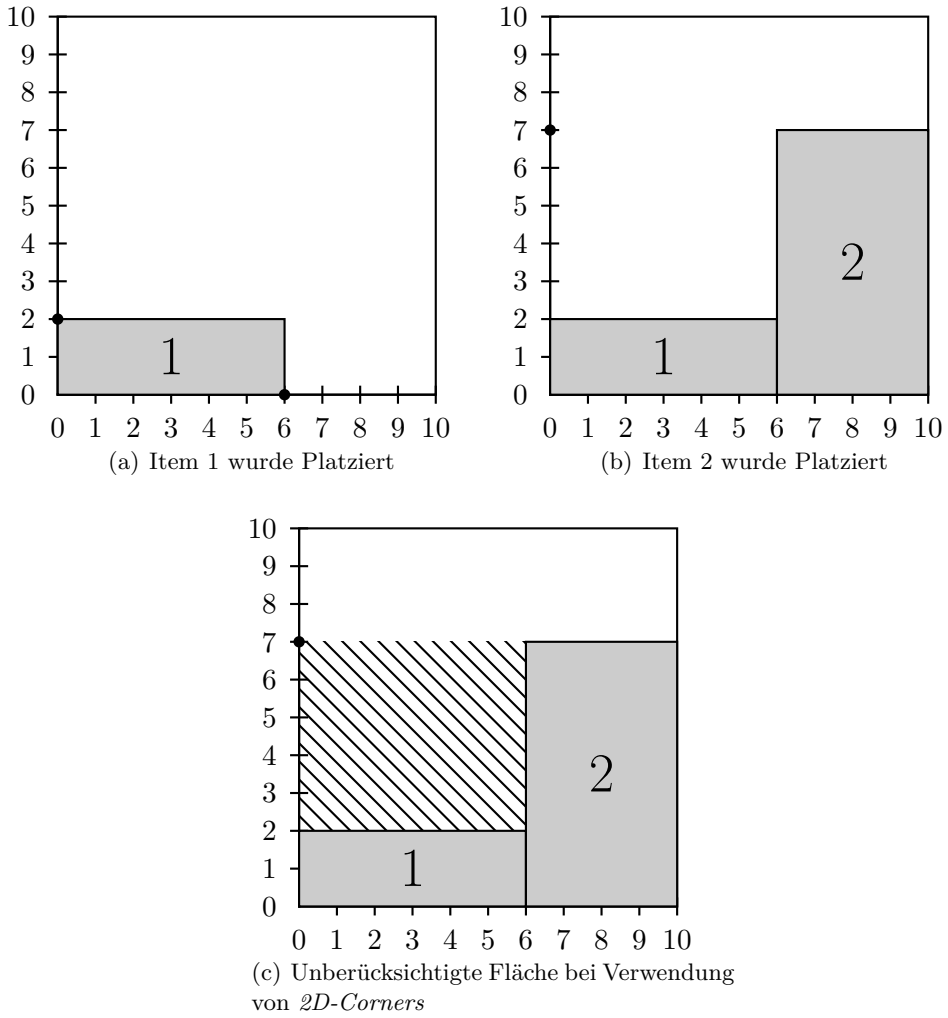
Abbildung 4.5: Eine Lösung, die von $LMAO_{MOD}$ mit *2D-Corners*$_{MOD}$ nicht gefunden wird

Anders verhält es sich mit *LMAO* bzw. $LMAO_{MOD}$. In diesem Beispiel wird davon ausgegangen, dass wir es mit dem Algorithmus $LMAO_{MOD}$ zu tun haben und deshalb die RLMD-Reihenfolge für die Item-Platzierung verwendet wird. Das Beispiel ist jedoch auch auf *LMAO* und die dort verwendete LMD-Reihenfolge anwendbar.

Angenommen, $LMAO_{MOD}$ hätte bereits Item 1 an Position $(0, 0)$ platziert, wie in Abbildung 4.6(a) zu sehen. *2D-Corners*$_{MOD}$ weist nun die beiden Eckpunkte $(6, 0)$ und $(0, 2)$ aus, von denen nach der RLMD-Strategie $(6, 0)$ zuerst betrachtet wird. Das einzige Item, das in diesem Eckpunkt zu einer korrekten Gesamtlösung führt, ist Item 2. Nachdem also Item 2 an Position $(6, 0)$ platziert wurde (Abbildung 4.6(b)), verzweigt $LMAO_{MOD}$ und berechnet mittels *2D-Corners*$_{MOD}$ die Eckpunkte neu. Aufgrund seiner Arbeitsweise würde *2D-Corners*$_{MOD}$ nun aber nur den Eckpunkt $(0, 7)$ ausweisen, was jedoch den Bereich zwischen den Punkten $(0, 7)$ und $(6, 2)$ brach liegen ließe (schraffierte Fläche in Abbildung 4.6(c)), da dieser unter die Hülle der Eckpunkte gelangen würde. Somit wäre das Gesamtproblem für $LMAO_{MOD}$ nicht mehr lösbar, obwohl eine zulässige Lösung existiert.

In einer anderen Verzweigung des Suchbaums würde $LMAO_{MOD}$ – nachdem Item 1 an Position $(0, 0)$ platziert wurde – den Punkt $(6, 0)$ als inaktiv markieren. Dies hätte allerdings zur Folge, dass diese Position innerhalb des aktuellen Teilbaums nie wieder für eine Item-Platzierung zur Verfügung stünde. Insbesondere könnte Item 2 nicht an dieser Position platziert werden und das in Abbildung 4.5 dargestellte Packmuster würde nie erreicht werden.

In beiden Fällen würde $LMAO_{MOD}$ versagen, obwohl eine zulässige Lösung existiert. Ein Ausweg besteht darin, die Routine *2D-Corners*$_{MOD}$ zur Berechnung von Platzierungs-



(a) Item 1 wurde Platziert

(b) Item 2 wurde Platziert

(c) Unberücksichtigte Fläche bei Verwendung von *2D-Corners*

Abbildung 4.6: Vorgehensweise von $LMAO_{MOD}$ unter Verwendung von *2D-Corners*$_{MOD}$

kandidaten durch eine andere Routine zu ersetzen, die alle normalen Positionen nach dem (R)LMD-Kriterium findet und nicht nur Eckpunkte nach Definition 4.5.2, wie es *2D-Corners*$_{MOD}$ tut. Beispielsweise müsste das neue Verfahren in Abbildung 4.6(b) den Punkt $(0, 2)$ ausweisen, damit Item 3 an dieser Position platziert werden kann.

Im Rahmen dieser Arbeit wurden zwei Verfahren entwickelt, die die geforderte Aufgabe auf unterschiedliche Weisen erfüllen, für ein gegebenes Packmuster diejenige nicht inaktive normale Position zurückzuliefern, die unter allen Positionen mit minimaler y-Koordinate minimale x-Koordinate hat. Dies sind die beiden Verfahren *RLMD-Corner* und ein zweites Verfahren, das auf der speziellen Datenstruktur *RLMD-Corner-DS* basiert (siehe Abschnitt 4.6.4).

4.6.2 Der Algorithmus RLMD-Corner

RLMD-Corner ist eine Routine zur Berechnung von potentiellen Platzierungspositionen für die bislang unplatzierten Items. Das Verfahren ist eine modifizierte Fassung von Algorithmus 4 (Seite 49). Die wesentlichen Unterschiede sind:

1. *RLMD-Corner* berechnet nur genau einen Punkt nach dem RLMD-Kriterium.
2. *RLMD-Corner* weist keine Positionen aus, die als inaktiv markiert sind.
3. *RLMD-Corner* verwirft sofort Positionen, an denen wegen Überschreitung der Containergrenzen ohnehin kein Item aus der Menge der unplatzierten Items platziert werden kann.

Als Eingabe benötigt der Algorithmus eine Menge P von bereits platzierten Items i mit Koordinaten (x_i, y_i) und Ausmaßen (w_i, h_i), $i \in P$, eine Menge T von als inaktiv markierten Positionen (Tabu-Punkten), die Containerausmaße (W, H), sowie die minimale Breite und Höhe w_{min} und h_{min} der Elemente aus der Menge $U = I \setminus P$ aller unplatzierten Items. Als Ausgabe genügt es, die aktive Position $\hat{c}$ zurückzugeben, die in der gegebenen Konfiguration dem RLMD-Kriterium entspricht.

Offensichtlich ist die gesuchte Position ein Element aus der Menge $(\bar{X} \times \bar{Y}) \setminus T$, wobei $\bar{X}(P) = \{0\} \cup \{x_i + w_i \mid i \in P\}$ und $\bar{Y}(P) = \{0\} \cup \{y_i + h_i \mid i \in P\}$ ist. *RLMD-Corner* bildet zunächst die beiden nicht-absteigend sortierten Listen $\bar{X}$ und $\bar{Y}$ und durchläuft diese anschließend, um den gesuchten Platzierungskandidaten zu ermitteln. Die genaue Arbeitsweise von *RLMD-Corner* wird von Algorithmus 9 beschrieben.

Algorithmus 9: RLMD-Corner

```
1:  procedure RLMDCorner(P, T, W, H, w_min, h_min)
2:      X̄(P) ← {0} ∪ {x_i + w_i | i ∈ P}
3:      Ȳ(P) ← {0} ∪ {y_i + h_i | i ∈ P}
4:      Sortiere X̄(P) und Ȳ(P) nicht-absteigend.
5:      for all ȳ ∈ Ȳ do
6:          if H − ȳ < h_min then          ▷ Alle Items würden nach oben hinausragen
7:              goto 13
8:          for all x̄ ∈ X̄ do
9:              if W − x̄ < w_min then      ▷ Alle Items würden nach rechts hinausragen
10:                 goto 5
11:             if (x̄, ȳ) ∉ T then                ▷ Prüfe, ob (x̄, ȳ) tabu ist
12:                 return (x̄, ȳ)
13:     return null
```

Lemma 4.6.1. *Sei $n = |P|$ die Anzahl gepackter Items. Wenn die Datenstruktur zur Speicherung der Menge T der Tabu-Positionen eine Anfrage nach der Existenz einer Position in T in Zeit $O(1)$ beantworten kann, dann hat* RLMD-Corner *eine Laufzeit von $O(n^2)$.*

4.6.3 Das Verfahren LMAO_{MOD}

Unter Verwendung von *RLMD-Corner* ist die Struktur des gesamten Verfahrens $LMAO_{MOD}$ zur Enumeration von Packmustern der Struktur des in Abschnitt 4.5 vorgestellten Verfahrens MV_{MOD} sehr ähnlich. Es muss im Unterschied zu MV_{MOD} eine weitere Datenstruktur gepflegt werden: die Menge T der als inaktiv markierten Positionen. Diese wurde als Hashtabelle realisiert, um konstante Zugriffszeiten für die Operationen `add`, `remove` und `contains` zu ermöglichen. Algorithmus 10 beschreibt die Arbeitsweise von $LMAO_{MOD}$ unter Verwendung der Routine *RLMD-Corner*.

Die Parameter V, W, H, *lifo*, *turnable*, *timeout* und *findBestSolution* von $LMAO_{MOD}$, die eingesetzten Datenstrukuren P, *wc*, *hc* und $\hat{I}$ sowie die Sortierreihenfolge der Item-Cluster $\hat{I}$ entsprechen denen von MV_{MOD}. Die einzige zusätzliche Datenstruktur, die von $LMAO_{MOD}$ benötigt wird, ist die Menge T der inaktiven Platzierungspostionen. Ein weiterer wesentlicher Unterschied zu MV_{MOD} besteht darin, dass $LMAO_{MOD}$ pro Suchbaumknoten nur genau eine Position berechnet (Zeile 12 in Algorithmus 10), die anschließend als inaktiv markiert (Zeile 13) und für die Platzierung aller noch vorhandenen Items verwendet wird (Zeile 32). Zuvor findet eine Überprüfung statt, ob die berechnete Position innerhalb eines der Items aus dem Packmuster P liegt, so dass ohnehin kein Item in dieser Position untergebracht werden kann (Zeile 17). Wenn das der Fall ist, kann die Platzierung von Items in dieser Position übersprungen werden.

Falls für eine Position $\hat{c}$ der weitere Verlauf des Algorithmus für alle in $\hat{c}$ platzierten Items scheitert, wird $\hat{c}$ zunächst in der Menge T der inaktiven Punkte belassen und es erfolgt ein rekursiver Aufruf von DeterminePlacements (Zeile 43). Falls auch dies scheitert, wird $\hat{c}$ wieder aktiviert (Zeile 46) und der aktuelle Suchbaumknoten mit Rückgabewert „falsch“ verlassen.

4.6.4 Beschleunigung der Berechnung von Platzierungen

Um die quadratische Worst-Case-Laufzeit von *RLMD-Corner* zu verbessern und somit das gesamte Verfahren $LMAO_{MOD}$ zu beschleunigen, wurde eine Methode zum Auffinden zulässiger Platzierungen entwickelt, welche die spezielle Datenstruktur *RLMD-Corner-DS* zur Verwaltung der Informationen über alle aktiven und inaktiven potentiellen Platzierungen des aktuellen Packmusters verwendet. Diese bietet die Möglichkeit, jede Anfrage nach dem nächsten RLMD-Punkt in Zeit $O(n)$ beantworten. Der Nachteil des neuen Verfahrens gegenüber *RLMD-Corner* besteht darin, dass bei Veränderungen des unterliegenden Packmusters ein gewisser Verwaltungsaufwand notwendig ist, um die Datenstruktur aktuell zu halten. Welche der beiden Methoden in der Realität schneller ist, soll durch Benchmarks mit 2OPP-Instanzen aus der Literatur untersucht werden (siehe Abschnitt 6.1).

Grundlage der Datenstruktur *RLMD-Corner-DS* bildet die Klasse `Coordinate`, deren Instanzen entweder x- oder y-Koordinaten repräsentieren. Sie beinhaltet die folgenden Attribute:

- `value`: den eigentlichen, ganzzahligen Wert der Koordinate,
- `neighbours`: eine Menge von Referenzen auf Instanzen vom Typ `Coordinate`, die aufsteigend nach den *value*-Werten dieser Instanzen sortiert ist, und

Algorithmus 10: LMAO_{MOD}

```
1: procedure LMAO_MOD(V, W, H, lifo, turnable, timeout, findBestSolution)
2:     Setze globale Variablen: W, H, lifo, turnable, timeout
3:     Initialisiere globale Datenstrukturen: P, wc, hc, Î, T
4:     Sortiere Î nach Entladepriorität, Fläche, Breite gemäß der Belegung von lifo und findBest-
   Solution
5:     DeterminePlacements

6: procedure DeterminePlacements
7:     if keine weiteren Items then
8:         return wahr
9:     if Timeout überschritten then
10:        return falsch
11:    w_min ← firstEntry(wc); h_min ← firstEntry(hc)
12:    ĉ ← RLMDCorner(P, T, W, H, w_min, h_min)          ▷ Berechne nächsten RLMD-Punkt
13:    T ← T ∪ {ĉ}                                        ▷ Markiere ĉ als inaktiv
14:    if ĉ = null then
15:        return falsch
16:    for all i ∈ P do
17:        if x_i < x_ĉ and x_i + w_i > x_ĉ and y_i < y_ĉ and y_i + h_i > y_ĉ then
18:            goto 43
19:    if turnable then
20:        O ← {normal, gedreht}
21:    else
22:        O ← {normal}
23:    for all o ∈ O do
24:        for all î ∈ Î do                               ▷ Iteriere über alle Item-Cluster
25:            i ← getFirst(î)                            ▷ Betrachte einen Repräsentanten aus î
26:            if o = gedreht then                        ▷ Drehe Item
27:                if w_i = h_i then                      ▷ Item ist quadratisch
28:                    goto 24                            ▷ Nächstes Item
29:                Drehe i um 90°
30:            if x_ĉ + w_i > W or y_ĉ + h_i > H then      ▷ Item ragt aus dem Container
31:                goto 24                                ▷ Nächstes Item
32:            Setze i an Position ĉ
33:            if (i überlappt ein Item aus P) or (lifo and P ∪ {i} verletzt LIFO-Reihenfolge)
   then
34:                goto 24                                ▷ Nächstes Item
35:            removeFirst(î)
36:            P ← P ∪ {i}
37:            decrement(i, wc); decrement(i, hc)
38:            success ← DeterminePlacements              ▷ Rekursiver Aufruf
39:            if success then
40:                return wahr
41:            Entferne i aus P und füge es zurück in î ein.
42:            increment(i, wc); increment(i, hc)
43:    success ← DeterminePlacements                      ▷ Rekursiver Aufruf
44:    if success then
45:        return wahr
46:    T ← T \ {ĉ}                                        ▷ ĉ wird wieder aktiv
47:    return falsch
```

- `itemCount`: die Anzahl von Items i im aktuellen Packmuster, für die gilt: $x_i + w_i = value$ bzw. $y_i + h_i = value$

RLMD-Corner-DS verwaltet die beiden sortierten Mengen `xCoordinates` und `yCoordinates` mit Objekten vom Typ `Coordinate`. Die `Coordinate`-Objekte, die x-Koordinaten repräsentieren, sind in der Menge `xCoordinates` eingetragen. Ensprechend enthält die Menge `yCoordinates` alle `Coordinate`-Objekte, die y-Koordinaten repräsentieren.

Die Elemente aus der `neighbours`-Menge einer `Coordinate`-Instanz repräsentieren Koordinaten des jeweils anderen Typs, d.h. ein Objekt, das eine x-Koordinate repräsentiert, enthält in seiner `neighbours`-Menge Referenzen auf Objekte aus der Menge `yCoordinates` und umgekehrt. Die Idee dieses Konstrukts ist, dass alle Platzierungspositionen eines Packmusters stets durch die beiden Mengen `xCoordinates` und `yCoordinates` gespeichert werden können. Soll dann beispielsweise für eine gegebene x-Koordinate die Menge aller aktiven Eckpunkte mit dieser Koordinate ermittelt werden, genügt ein Durchlauf durch die `neighbours`-Menge der entsprechenden `Coordinate`-Instanz aus der Menge `xCoordinates`.

Sowohl für die sortierte Menge `neighbours` eines `Coordinate`-Objekts als auch für die sortierten Mengen `xCoordinates` und `yCoordinates` werden balancierte Bäume eingesetzt. Konkret werden Instanzen der Klasse `java.util.TreeSet` bzw. `java.util.TreeMap` verwendet, welche die Datenstruktur Red-Black-Tree (auch „Rot-Schwarz-Baum“ genannt) implementieren. Red-Black-Trees erlauben das Einfügen, Entfernen und Suchen eines Elements in Zeit $O(\log n)$ und die Iteration über alle Elemente in linearer Zeit.

Für die korrekte Verwaltung der aktiven Positionen einer Container-Bepackung muss die Datenstruktur während des Ablaufs des Algorithmus $LMAO_{MOD}$ mehrere Operationen unterstützen. Diese werden im Folgenden aufgelistet und näher erläutert.

- `addPoint(x,y)`: Fügt einen neuen Punkt (x, y) in die Datenstruktur ein oder reaktiviert einen zurzeit inaktiven Punkt. Falls (x, y) neu ist, müssen entsprechende Koordinaten-Objekte in eine oder beide der Mengen `xCoordinates` und `yCoordinates` eingefügt und durch Eintragung aller existierenden `Coordinate`-Objekte in die `neighbours`-Mengen initialisiert werden. Falls (x, y) zuvor in der Datenstruktur existiert hat und nur aktiviert werden muss, genügt es, die entsprechenden Einträge in den `neighbours`-Mengen der entsprechenden x- und y-Koordinaten vorzunehmen. Laufzeit: $O(n)$.

- `removePoint(x,y)`: Deaktiviert einen Punkt (x, y) in der Datenstruktur. Hierzu genügt es, die x-Koordinate aus der `neighbours`-Menge der y-Koordinate zu entfernen und ensprechend die y-Koordinate aus der `neighbours`-Menge der x-Koordinate. Laufzeit: $O(\log n)$.

- `addXCoordinate(x)`: Fügt eine neue `Coordinate`-Instanz zur Menge `xCoordinates` hinzu. Alle Elemente aus `yCoordinates` werden in die `neighbours`-Menge der neuen Koordinate aufgenommen und umgekehrt wird die neue Koordinate zu den `neighbours`-Mengen der Elemente in `yCoordinates` hinzugefügt. Falls bereits eine `Coordinate`-Instanz existiert, deren `value`-Attribut dem Wert von x entspricht, genügt es, den `itemCount` dieser Instanz zu inkrementieren. Laufzeit: $O(n)$.

- `addXCoordinate(y)`: Analog zu `addXCoordinate`.
- `removeXCoordinate(x)`: Falls der Wert von `itemCount` der `Coordinate`-Instanz, die x repräsentiert, gleich eins ist, wird die Instanz aus `xCoordinates` und aus den `neighbour`-Mengen aller Instanzen in `yCoordinates` entfernt. Ansonsten genügt es, den `itemCount` zu dekrementieren. Laufzeit: $O(n \log n)$.
- `removeYCoordinate(y)`: Analog zu `removeXCoordinate`.
- `rlmdActivePoint()`: Ermittelt die erste nicht inaktive Position (x, y) nach dem „umgekehrten leftmost-downward“-Kriterium. Diese kann gefunden werden, indem zunächst die Menge `yCoordinates` in aufsteigender Reihenfolge so lange durchlaufen wird, bis ein `Coordinate`-Objekt mit nicht-leerer `neighbours`-Menge gefunden wird. Das erste und wegen der aufsteigenden Sortierung kleinste Element aus der `neighbours`-Menge dieses Objekts stellt dann die passende x-Koordinate dar. Laufzeit: $O(n)$.

Die Operationen `addXCoordinate(`$x_i + w_i$`)` und `addYCoordinate(`$y_i + h_i$`)` kommen dann zum Einsatz, wenn ein Item i zum aktuellen Packmuster P hinzugefügt wird. Ensprechend werden die Operationen `removeXCoordinate(`$x_i + w_i$`)` und `removeYCoordinate(`$y_i + h_i$`)` immer dann eingesetzt, wenn das Item i aus P entfernt wird. Somit befinden sich in den Mengen `xCoordinates` und `yCoordinates` immer genau diejenigen x- und y-Koordinaten, zu denen mindestens ein Item aus P mit entsprechendem rechten bzw. oberen Ende existiert. Eine Ausnahme bilden die `Coordinate`-Objekte mit dem Wert 0, welche die linke und untere Containerwand identifizieren und sich daher unabhängig von der aktuellen Bepackung in den Mengen `xCoordinates` und `yCoordinates` befinden.

Die Operationen `removePoint(x,y)` und `addPoint(x,y)` werden ausgeführt, um eine Position als inaktiv zu markieren, bzw. um eine solche Markierung wieder aufzuheben. Man beachte, dass bei Verwendung von *RLMD-Corner-DS* – im Gegensatz zum Verfahren *RLMD-Corner* – der Bedarf nach der Hashtabelle T entfällt, in der die Tabu-Punkte vermerkt sind.

Die Operation `rlmdActivePoint()` liefert die aktuelle nicht-inaktive Position nach dem umgekehrten leftmost-downward-Kriterium.

4.6.5 Die Idee für eine untere Schranke

Die Verwendung von *2D-Corners*$_{MOD}$ zur Berechnung von Eckpunkten im Rahmen des Algorithmus MV_{MOD} führt dazu, dass in jedem Suchbaumknoten eine naheliegende und schnell zu berechnende untere Schranke geprüft werden kann. Diese basiert auf der Beobachtung, dass die von *2D-Corners*$_{MOD}$ berechneten Eckpunkte eine Hülle um alle Items der aktuellen Bepackung P definieren, innerhalb welcher kein weiteres Item platziert wird (siehe Abschnitt 4.5.2, Seite 56). Im Falle von $LMAO_{MOD}$ wird jedoch statt *2D-Corners*$_{MOD}$ entweder die Routine *RLMD-Corner* oder die Datenstruktur *RLMD-Corner-DS* zur Berechnung potentieller Platzierungen eingesetzt, die beide keine entsprechende Hülle aufspannen und somit die in MV_{MOD} verwendete untere Schranke nicht zulassen.

Die Suche nach einer anderen unteren Schranke für $LMAO_{MOD}$ ergab, dass es möglicherweise zulässig ist, die gesamte Fläche unterhalb des zuletzt platzierten Items als nicht

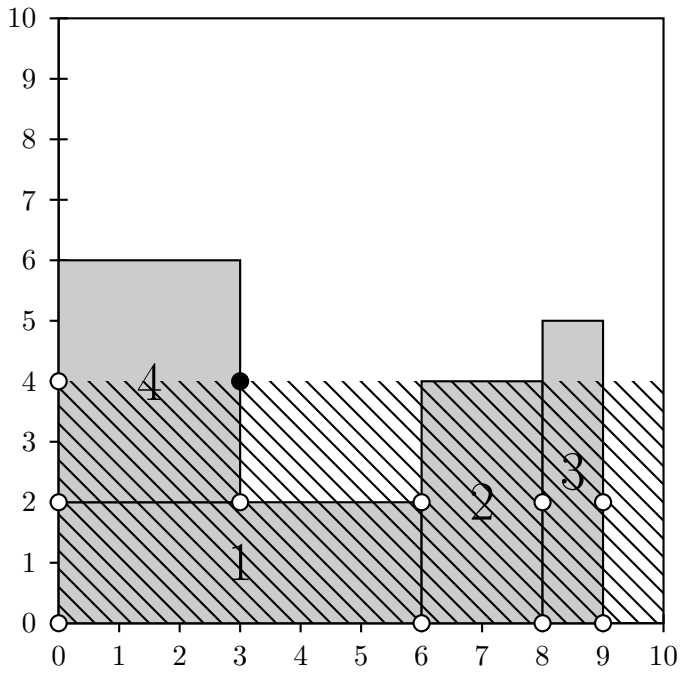


Abbildung 4.7: Idee für eine untere Schranke: Nicht nutzbare Fläche unterhalb des zuletzt berechneten RLMD-Eckpunktes

weiter nutzbar zu deklarieren und somit die noch verbleibende nutzbare Fläche im Container womöglich stark zu reduzieren. Im Folgenden wird diese Idee genauer beschrieben.

Sei P das aktuelle Packmuster in einem Suchbaumknoten i von $LMAO_{MOD}$ und $\hat{c} = (\hat{x}, \hat{y})$ der zugehörige RLMD-Eckpunkt. Falls gezeigt werden kann, dass innerhalb des Teilbaums mit Wurzel i keine weiteren Items aus $I \setminus P$ an irgendeiner Position $(x, y),\ y < \hat{y}$ platziert werden, kann die Fläche $W \times \hat{y}$ innerhalb dieses Teilbaums nicht mehr genuzt werden. Zur nicht-nutzbaren Fläche kommt die Summe der Flächen aller Items aus P hinzu, die – zumindest teilweise – oberhalb von $\hat{y}$ platziert sind. Die noch nutzbare Fläche A kann dann beschrieben werden als:

$$A = W(H - \hat{y}) - \sum_{i \in P} \min \{w_i h_i, \max \{0, w_i(y_i + h_i - \hat{y})\}\} \tag{4.9}$$

Unter diesen Voraussetzungen würde es also genügen, die Fläche A mit der Fläche $\sum_{i \in I \setminus P} w_i h_i$ aller noch zu packenden Items zu vergleichen. Falls A kleiner ist, greift die untere Schranke und der aktuelle Suchbaumknoten kann mit dem Ergebnis „nicht lösbar“ verlassen werden.

Zum besseren Verständnis von Gleichung (4.9) betrachte man Abbildung 4.7. Hier ist ein Packmuster zu sehen, dessen Items in der Reihenfolge $1, 2, 3, 4$ von $LMAO_{MOD}$ platziert wurden. Die Positionen $(0, 0)$, $(6, 0)$, $(8, 0)$, $(9, 0)$, $(0, 2)$, $(3, 2)$, $(6, 2)$, $(8, 2)$, $(9, 0)$ und $(0, 4)$ sind in dieser Reihenfolge als potentielle Platzierungskandidaten (RLMD-Eckpunkte) betrachtet und entweder zur Platzierung eines Items verwendet oder verworfen worden.

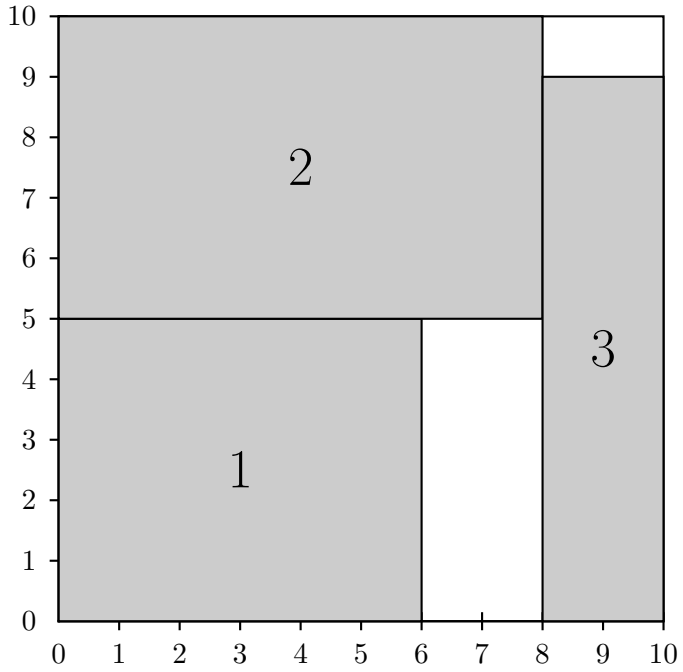


Abbildung 4.8: Packmuster, für das die neue untere Schranke versagt

In beiden Fällen sind diese Positionen als inaktiv markiert. Der nächste ausgewiesene RLMD-Eckpunkt ist $(3, 4) = (x_4+w_4, y_2+h_2)$. Wenn man von der Voraussetzung ausgeht, dass $LMAO_{MOD}$ die Platzierungen bezogen auf ihre y-Koordinaten in nicht-absteigender Reihenfolge ausweist, dann ist es klar, dass die gesamte in Abbildung 4.7 schraffierte Fläche unterhalb des aktuellen Suchbaumknotens nicht für weitere Items verwendet werden wird. Die gesamte nicht-schraffierte Fläche darüber entspricht dem Term $W(H-\hat{y})$ in Gleichung (4.9). Von dieser Fläche subtrahiert wird noch die Flächensumme der Items, die über $\hat{y}$ hinausragen.

Sei der Rang $r : P \to \{1, \ldots, |P|\}$ eine bijektive Funktion, die die Reihenfolge der Platzierung von Items $i \in P$ ausdrückt. Um die oben beschriebene untere Schranke einsetzen zu können, muss gezeigt werden, dass jedes normale Packmuster P von $LMAO_{MOD}$ so zusammengesetzt werden kann, dass für jedes Paar $i, j \in P$ von Packstücken mit $r(i) < r(j)$ gilt, dass $y_i \leq y_j$. Leider kann diese notwendige Eigenschaft nicht immer gewährleistet werden, was am in Abbildung 4.8 dargestellten Beispiel verdeutlicht werden soll.

Das dargestellte normale Packmuster kann durch $LMAO_{MOD}$ unter Verwendung der oben beschriebenen unteren Schranke nicht gefunden werden. Dies widerspricht dem Anspruch eines exakten Verfahrens für das 2OPP, alle normalen Packmuster zu enumerieren. Die zur Platzierung von Item 3 notwendige Position $(8, 0)$ entsteht erst, nachdem Item 2 platziert wurde. Die y-Komponente der Platzierung von Item 2 ist aber 5 und insbesondere > 0, so dass die Position $(8, 0)$ nicht mehr als potentielle Platzierung betrachtet werden dürfte, um die Voraussetzung für die Schranke zu erfüllen. Aus diesem Grund muss die Schranke im Kontext von $LMAO_{MOD}$ als inkorrekt und deshalb nicht anwendbar angesehen werden.

4.7 Untere Schranken für das 2BPP

Bevor ein exaktes Verfahren zur Lösung des zweidimensionalen, orthogonalen Packproblems (2OPP) bemüht wird, ist es sinnvoll, mit Hilfe einer unteren Schranke für das zweidimensionale Bin-Packing-Problem (2BPP) die Anzahl mindestens benötigter Container für die gegebene 2OPP-Eingabeinstanz zu berechnen. Falls die untere Schranke einen Wert > 1 annimmt, ist der Einsatz des erwartungsgemäß laufzeitintensiven exakten Verfahrens für das 2OPP überflüssig, weil a priori klar ist, dass für die Eingabeinstanz keine zulässige Lösung existiert.

Die unteren Schranken L_0–L_4 für das 2BPP wurden in [Martello und Vigo 1998] vorgestellt. Sie wurden im Rahmen dieser Arbeit implementiert und sollen im Folgenden beschrieben werden. Es wird von einer Menge gleichartiger Container mit den Ausmaßen (W, H) und einer Menge $J = \{1, \ldots, n\}$ rechtwinkliger Items mit den Ausmaßen (w_j, h_j) und der Fläche $a_j = w_j \times h_j$, $j \in J$ ausgegangen.

4.7.1 Die Continuous Lower Bound L_0

Die Continuous Lower Bound L_0 ist eine triviale untere Schranke für das 2BPP. Sie ergibt sich aus dem Quotienten aus Summe der Flächen aller Items und Fläche des Containers:

$$L_0 = \left\lceil \frac{\sum_{j=1}^{n} a_j}{WH} \right\rceil \tag{4.10}$$

Die Schranke L_0 kann offensichtlich in Zeit $O(n)$ berechnet werden. [Martello und Vigo 1998] haben bewiesen, dass für L_0 die Worst-Case Performance $\frac{1}{4}$ beträgt.

Definition 4.7.1 (Worst-Case Performance)**.** Sei I eine Instanz eines beliebigen Minimierungsproblems P, $z(I)$ der Wert einer optimalen Lösung für I und $L(I)$ der von der unteren Schranke L berechnete Wert für I. Mit Worst-Case Performance von L bezeichnet man den größten Wert $\rho \in \mathbb{R}$, so dass gilt:

$$\frac{L(I)}{z(I)} \geq \rho \text{ für alle Instanzen } I \text{ des Problems } P. \tag{4.11}$$

4.7.2 Die untere Schranke L_1

Die Schranke L_1 stellt eine Generalisierung der unteren Schranken dar, die von [Martello und Toth 1990] für das eindimensionale Bin-Packing-Problem (BPP) vorgestellt wurden.

Sei $J^W = \left\{ j \in J \mid w_j > \frac{1}{2}W \right\}$ die Teilmenge von J, so dass keine zwei Elemente aus J^W nebeneinander im Container untergebracht werden können. Ferner sei p ein beliebiger ganzzahliger Wert mit $1 \leq p \leq \frac{1}{2}H$, dann sind die Teilmengen J_1, J_2, J_3 von J^W wie folgt definiert:

$$J_1 = \left\{ j \in J^W \mid h_j > H - p \right\}, \tag{4.12}$$

$$J_2 = \left\{ j \in J^W \mid H - p \geq h_j > \frac{1}{2}H \right\}, \tag{4.13}$$

$$J_3 = \left\{ j \in J^W \mid \frac{1}{2}H \geq h_j \geq p \right\}. \tag{4.14}$$

Da keine zwei Items aus $J_1 \cup J_2$ in einem Container untergebracht werden können, stellt $|J_1 \cup J_2|$ bereits eine gültige untere Schranke für z dar, die unabhängig von p ist. Diese Schranke kann weiter verstärkt werden durch die Tatsache, dass kein Item aus J_3 in einem Container untergebracht werden kann, in dem sich bereits ein Item aus J_1 befindet. Aufgrund dieser Beobachtung treffen [Martello und Vigo 1998] die folgende Aussage:

Satz 4.7.1. *Sei p ein ganzzahliger Wert mit $1 \leq p \leq \frac{1}{2}H$ und seien J_1, J_2, J_3 definiert wie in (4.12)–(4.14), dann gilt die folgende untere Schranke für z:*

$$L_1^W(p) = \max\left\{ L_\alpha^W(p), L_\beta^w(p) \right\}, \tag{4.15}$$

Dabei sind $L_\alpha^W(p)$ und $L_\beta^w(p)$ wie folgt definiert:

$$L_\alpha^W(p) = |J_1 \cup J_2| + \max\left\{ 0, \left\lceil \frac{\sum_{j \in J_3} h_j - \left(|J_2|\, H - \sum_{j \in J_2} h_j \right)}{H} \right\rceil \right\}, \tag{4.16}$$

$$L_\beta^W(p) = |J_1 \cup J_2| + \max\left\{ 0, \left\lceil \frac{|J_3| - \sum_{j \in J_2} \left\lfloor \frac{H - h_j}{p} \right\rfloor}{\left\lfloor \frac{H}{p} \right\rfloor} \right\rceil \right\}. \tag{4.17}$$

Ein Beweis für Satz 4.7.1 wird in [Martello und Vigo 1998] geführt.

Korollar 4.7.1. *Eine untere Schranke für z ist*

$$L_1^W = \max_{1 \leq p \leq \frac{1}{2}H} \left\{ L_1^W(p) \right\}. \tag{4.18}$$

L_1^W kann in Zeit $O(n^2)$ berechnet werden.

Beweis. Die Korrektheit der Schranke ergibt sich aus Satz 4.7.1. Die Laufzeit ergibt sich aus den folgenden Beobachtungen:

1. $L_\alpha^W(p)$ und $L_\beta^W(p)$ können in Zeit $O(n)$ berechnet werden;
2. Für $L_1^W(p)$ müssen nur alle $p = h_j$, $j \in J$ betrachtet werden, d.h. nur höchstens n verschiedene Werte (für einen Beweis siehe [Martello und Toth 1990]). □

Sei $J^H = \left\{ j \in J \mid h_j > \frac{1}{2}H \right\}$. Wenn in (4.12)-(4.18) J^W, h_j und H durch J^H, w_j und W ersetzt werden, erhält man die zu (4.18) analoge untere Schranke

$$L_1^H = \max_{1 \leq p \leq \frac{1}{2}W} \left\{ L_\alpha^H(p), L_\beta^H(p) \right\}. \tag{4.19}$$

Schließlich lassen sich (4.18) und (4.19) zusammenfassen zu der Schranke

$$L_1 = \max\left\{L_1^W, L_1^H\right\}, \tag{4.20}$$

welche in Zeit $O(n^2)$ berechnet werden kann. [Martello und Vigo 1998] haben gezeigt, dass zwischen den Schranken L_0 und L_1 keine Dominanzbeziehung besteht und dass die Worst-Case Performance von L_1 beliebig schlecht sein kann.

Die im Folgenden vorgestellten Schranken berücksichtigen explizit beide Dimensionen der zu packenden Items. Sie dominieren die Schranken L_0 und L_1, haben jedoch schlechtere Laufzeiteigenschaften.

4.7.3 Die untere Schranke L_2

Satz 4.7.2. *Gegeben sei ein ganzzahliger Wert q mit $1 \leq q \leq \frac{1}{2}W$ und seien*

$$K_1 = \{j \in J \mid w_j > W - q\}, \tag{4.21}$$

$$K_2 = \left\{j \in J \mid W - q \geq w_j > \frac{1}{2}W\right\}, \tag{4.22}$$

$$K_3 = \left\{j \in J \mid \frac{1}{2}W \geq w_j \geq q\right\}. \tag{4.23}$$

Dann ist

$$L_2^W(q) = L_1^W + \max\left\{0, \left\lceil \frac{\sum_{j\in K_2\cup K_3} h_j w_j - (HL_1^W - \sum_{j\in K_1} h_j)W}{HW} \right\rceil\right\} \tag{4.24}$$

eine gültige untere Schranke für z.

Beweis. Ein Beweis für die Korrektheit von $L_2^W(q)$ wird in [Martello und Vigo 1998] geführt. □

Korollar 4.7.2.

$$L_2^W = \max_{1\leq q\leq \frac{1}{2}W}\left\{L_2^W(q)\right\} \tag{4.25}$$

ist eine gültige untere Schranke für z und kann in Zeit $O(n^2)$ berechnet werden.

Beweis. Die Korrektheit von L_2^W ergibt sich direkt aus Satz 4.7.2. Die Laufzeitkomplexität ergibt sich einerseits aus der Komplexität $O(n^2)$ der einmaligen Berechnung von L_1^W und andererseits aus der Berechnung des rechten Terms in (4.24) für jedes q. Da dieser Term in Zeit $O(n)$ für festes q berechnet werden kann, bleibt nur noch zu zeigen, dass es genügt, alle $q = w_j$, $j \in J$ zu betrachten. Für einen Beweis dieser Aussage siehe [Martello und Vigo 1998]. □

Korollar 4.7.3. *Eine untere Schranke für z ist*

$$L_2 = \max\left\{L_2^W, L_2^H\right\}. \tag{4.26}$$

Dabei ergibt sich L_2^H durch Ersetzung von W, w_j und L_1^W durch H, h_j und L_1^H in (4.21)–(4.25).

Auch wenn die asymptotische Laufzeitkomplexität $O(n^2)$ der Schranke L_2 der Komplexität von L_1 entspricht, sollte angemerkt werden, dass die Berechnung von L_2 die Berechnung von L_1 erfordert. Da zusätzlich der rechte Term in (4.24) bis zu n mal berechnet werden muss (und zwar sowohl für L_2^W als auch für L_2^H), kann für L_2 in der Praxis von einer etwa zwei mal so großen Laufzeit wie für L_1 ausgegangen werden.

[Martello und Vigo 1998] zeigen, dass sowohl L_0 als auch L_1 von L_2 dominiert werden.

4.7.4 Die untere Schranke L_3

Die im Folgenden beschriebene untere Schranke L_3 kann in einer asymptotisch schlechteren Laufzeit als die bisher in diesem Abschnitt vorgestellten Schranken L_0, L_1 und L_2 berechnet werden, aber dafür in einigen Fällen zu einem besseren Ergebnis führen.

Seien p und q ganzzahlige Werte mit $1 \leq p \leq \frac{1}{2}H$ und $1 \leq q \leq \frac{1}{2}W$ und seien die Mengen I_1, I_2, I_3 wie folgt definiert:

$$I_1 = \{j \in J \mid h_j > H - p \wedge w_k > W - q\}, \tag{4.27}$$

$$I_2 = \left\{j \in J \setminus I_1 \mid h_j > \frac{1}{2}H \wedge w_j > \frac{1}{2}W\right\}, \tag{4.28}$$

$$I_3 = \left\{j \in J \mid \frac{1}{2}H \geq h_j \geq p \wedge \frac{1}{2}W \geq w_j \geq q\right\} \tag{4.29}$$

Man beachte, dass unabhängig von p und q keine zwei Packstücke aus $I_1 \cup I_2$ in einem Container untergebracht werden können. Deshalb ist $|I_1 \cup I_2|$ bereits eine gültige untere Schranke für z. Des Weiteren gilt, dass ein Packstück aus I_3 niemals in einen Container gepackt werden kann, in dem sich bereits ein Item aus I_1 befindet. Es kann aber durchaus möglich sein, in Containern mit jeweils einem Item aus I_2 je ein oder mehrere Items aus I_3 zu platzieren. Für diejenigen Items aus I_3, für die das nicht möglich ist, werden dann ein oder mehrere weitere Container benötigt. Diese Anzahl an zusätzlich benötigten Containern gilt es zu berechnen und zur unteren Schranke $|I_1 \cup I_2|$ hinzuzuaddieren.

Es wird im Folgenden eine relaxierte Instanz des Problems betrachtet, in der alle Items $j \in I_3$ minimale Ausmaße annehmen, nämlich $h_j = p$ und $w_j = q$.

Lemma 4.7.1. *Gegeben sei ein Container $W \times H$, der ein Item der Größe (w_j, h_j) enthält. Die Anzahl Items der Größe (p, q), die zusätzlich in diesem Container untergebracht werden können, kann durch den folgenden Term nach oben abgeschätzt werden:*

$$m(j,p,q) = \left\lfloor \frac{H}{p} \right\rfloor \left\lfloor \frac{W - w_j}{q} \right\rfloor + \left\lfloor \frac{W}{q} \right\rfloor \left\lfloor \frac{H - h_j}{p} \right\rfloor - \left\lfloor \frac{H - h_j}{p} \right\rfloor \left\lfloor \frac{W - w_j}{q} \right\rfloor \tag{4.30}$$

Beweis. Der Beweis wird in [Martello und Vigo 1998] geführt. □

Satz 4.7.3. *Seien p und q ganzzahlige Werte mit $1 \leq p \leq \frac{1}{2}H$ und $1 \leq q \leq \frac{1}{2}W$. Ferner seien I_1, I_2, I_3 und $m(j,p,q)$ definiert wie in (4.27)–(4.30). Eine untere Schranke für z ist gegeben durch*

$$L_3(p,q) = |I_1 \cup I_2| + \max\left\{0, \left\lceil \frac{|I_3| - \sum_{j \in I_2} m(j,p,q)}{\left\lfloor \frac{H}{p} \right\rfloor \left\lfloor \frac{W}{q} \right\rfloor} \right\rceil \right\}. \tag{4.31}$$

Beweis. Die Schranke ergibt sich direkt aus den Eigenschaften von I_1, I_2, I_3 und aus Lemma 4.7.1. □

Korollar 4.7.4. *Eine untere Schranke für z ist*

$$L_3 = \max_{1 \leq p \leq \frac{1}{2}H,\, 1 \leq q \leq \frac{1}{2}W} \{L_3(p,q)\}. \tag{4.32}$$

Diese Schranke kann in Zeit $O(n^3)$ berechnet werden.

Beweis. Die Korrektheit der Schranke folgt unmittelbar aus Satz 4.7.3. Die Laufzeit ergibt daraus, dass $L_3(p,q)$ für ein festes Paar von Werten (p,q) in Zeit $O(n)$ berechnet werden kann und es zudem genügt, solche Wertepaare (p,q) zu betrachten, für die $p = h_j$, $q = w_k$ für beliebige $j,k \in J$ gilt. Für einen Beweis letzterer Aussage siehe [Martello und Vigo 1998]. □

4.7.5 Die untere Schranke L_4

[Martello und Vigo 1998] haben gezeigt, dass zwischen den unteren Schranken L_2 und L_3 keine Dominanzbeziehung besteht. Deshalb ist die sich insgesamt ergebende untere Schranke:

$$L_4 = \max\{L_2, L_3\} \tag{4.33}$$

Korollar 4.7.5. *L_4 kann in Zeit $O(n^3)$ berechnet werden.*

Beweis. Die Aussage folgt aus den Korollaren 4.7.3 und 4.7.4. □

4.7.6 Berücksichtigung der Drehbarkeit von Items

Die „Continuous Lower Bound“ L_0 gilt unabhängig davon, ob Items um 90° drehbar sind oder nicht. Anders verhält es sich jedoch mit den Schranken L_1-L_4. Diese verwenden für die Berechnungen explizit die Breite und Höhe der Items sowie des Containers. Die beiden erstgenannten Werte können jedoch bei der Drehung eines Items ihre Werte vertauschen. Das würde dazu führen, dass die Schranken falsche und im schlimmsten Fall zu hohe Ergebnisse liefern würden.

Um dies zu verhindern, schlagen [Boschetti und Mingozzi 2003] vor, die Ausmaße der Items vor Berechnung der unteren Schranken zu modifizieren. Ein Item j mit den Ausmaßen (w_j, h_j) erhält vorübergehend die Ausmaße $(\bar{w}_j, \bar{h}_j)$, die wie folgt definiert sind:

$$\bar{w}_j = \begin{cases} \min\{w_j, h_j\}, & \text{falls } w_j \leq H,\ h_j \leq W \text{ und } j \text{ drehbar} \\ w_j & \text{sonst} \end{cases} \tag{4.34}$$

$$\bar{h}_j = \begin{cases} \min\{w_j, h_j\}, & \text{falls } w_j \leq H,\ h_j \leq W \text{ und } j \text{ drehbar} \\ h_j & \text{sonst} \end{cases} \tag{4.35}$$

Die Dimensionen (W, H) des Containers bleiben hingegen unverändert.

Die unteren Schranken L_0, L_1, L_3 und L_4, sowie die Anpassung der Ausmaße der Items bei eingeschalteter Drehbarkeit wurden im Rahmen dieser Arbeit implementiert. Sie sind Bestandteil des gestaffelten Verfahrens zur exakten Lösung des 2OPP, welches im folgenden Abschnitt beschrieben wird.

4.8 Das Verfahren Staged-Packer

Die Ausführung eines exakten Verfahrens für das zweidimensionale, orthogonale Packproblem (2OPP) kann mit einer sehr hohen Laufzeit verbunden sein, was die Ergebnisse der in Abschnitt 6.1 durchgeführten Untersuchungen belegen. Das Laufzeitproblem betrifft insbesondere den integrierten Ausführungsmodus (siehe Abschnitt 5.2), in welchem die Anzahl von Aufrufen des Packalgorithmus stark von der Arbeitsweise des eingesetzten Tourenplanungsalgorithmus abhängt und potentiell sehr groß ist. Eine Möglichkeit, die Laufzeit des Packverfahrens einzuschränken besteht darin, eine maximale Laufzeit für jede zu lösende Instanz festzulegen und nach Ablauf dieser Zeitvorgabe die Instanz als nicht lösbar anzusehen. Es erscheint sinnvoll, sowohl relativ leicht lösbare als auch klar unlösbare Instanzen so weit wie möglich im Vorfeld zu erkennen und somit den exakten Algorithmus gar nicht erst anwenden zu müssen. Zu diesem Zweck wurde das gestaffelte Verfahren *Staged-Packer* implementiert, welches eine 2OPP-Instanz in drei Schritten abarbeitet. Algorithmus 11 skizziert den Ablauf das Verfahrens.

Algorithmus 11: Gestaffeltes Verfahren für das 2OPP

```
1: procedure STAGED-PACKER(V, W, H, lifo, turnable, timeout, findBestSolution)
2:     if ¬ findBestSolution then
3:         if L_4 > 1 then                          ▷ Berechne die untere Schranke
4:             return falsch
5:     Versuche die Instanz mittels der Heuristik TP_MOD zu lösen. Falls die Instanz heu-
       ristisch nicht gelöst werden kann:
6:     Versuche die Instanz mittels eines exakten Verfahrens zu lösen.
```

Der Algorithmus benötigt, wie jedes andere in dieser Arbeit vorgestellte Verfahren zur Lösung des 2OPP auch, die folgenden Parameter:

- V: Eine Liste von Kunden einer Tour in der Reihenfolge, wie sie in der Tour vorkommen. Jedem Kunden ist eine Menge von Items zugeordnet, die im Container untergebracht werden sollen.
- W, H: Breite und Höhe des Containers
- *lifo*: Boolscher Wert der festlegt, ob bei die Bepackung die LIFO-Reihenfolge eingehalten werden muss
- *turnable*: Boolscher Wert der angibt, ob die Packstücke um 90° drehbar sind
- *findBestSolution*: Boolscher Wert der festlegt, ob es genügt, das beste gefundene Ergebnis auszuweisen (wahr) oder ob das Entscheidungsproblem vollständig gelöst werden muss (falsch)
- *timeout*: Laufzeit, nach der die Berechnungen abgebrochen werden müssen

Im ersten Schritt berechnet *Staged-Packer* den Wert der unteren Schranke L_4 für das zweidimensionale Bin-Packing-Problem, um womöglich die Nicht-Lösbarkeit der Instanz festzustellen (Zeile 3 in Algorithmus 11). Es sei darauf hingewiesen, dass die Schranke geschwächt wird, falls die Packstücke gedreht werden dürfen, d.h. wenn der Parameter *turnable* wahr ist (siehe hierzu Abschnitt 4.7). Die Nebenbedingung der LIFO-Reihenfolge muss von der unteren Schranke nicht berücksichtigt werden, denn sequentielle Bepackung kann die Lösbarkeit des Ladeproblems nur verschlechtern und nie verbessern.

Wenn $L_4 = 1$ ist und somit die Nicht-Lösbarkeit der Instanz mittels der unteren Schranke nicht nachgewiesen werden kann, wird versucht, eine Lösung mit Hilfe der Heuristik TP_{MOD} zu finden. Erst wenn diese keine Lösung für die Instanz finden kann, wird die Instanz mittels eines exakten Verfahrens gelöst. Hierfür kommen prinzipiell die Verfahren MV_{MOD} (siehe Abschnitt 4.5) sowie $LMAO_{MOD}$ (siehe Abschnitt 4.6) in Frage, wobei letzteres in Kombination mit dem Verfahren *RLMD-Corner* oder unter Verwendung der Datenstruktur *RLMD-Corner-DS* eingesetzt werden kann. Die Ergebnisse der experimentellen Vergleiche der exakten Verfahren haben es jedoch nahegelegt, stets MV_{MOD} einzusetzen.

Die Parameter V, W, H, *lifo*, *turnable* und *findBestSolution* werden an die beiden eingesetzten 2OPP-Verfahren weitergereicht. Es hängt vom Wert von *findBestSolution* ab, ob die untere Schranke L_4 eingesetzt wird. Falls es genügt, ein möglichst gutes Ergebnis für das 2OPP zu finden (*findBestSolution* = wahr), wäre ein Ergebnis $L_4 > 1$ wertlos, da die nachfolgenden Schritte ohnehin ausgeführt werden müssten. Des Weiteren wird, falls *findBestSolution* gesetzt ist, die bessere der beiden Lösungen vorgemerkt, die die Heuristik und das exakte Verfahren gefundenen haben. Die vorgemerkte Lösung kann später über eine Zugriffsmethode abgefragt werden.

Der Parameter *timeout* wird ausschließlich an das exakte 2OPP-Verfahren weitergeleitet, denn üblicherweise ist die Ausführungsgeschwindigkeit der unteren Schranke und der Heuristik im Vergleich zu der des exakten Verfahrens so hoch, dass es keinen Sinn ergeben würde, ihre Laufzeit zu beschränken.

Die Effektivität der drei Stufen von *Staged-Packer* wird in den Abschnitten 6.2 und 6.3 untersucht.

Kapitel 5

Das kombinierte Problem

In den vorangehenden Kapiteln wurden der Algorithmus *I1* zur Lösung des Tourenplanungsproblems (Abschnitt 3.4), das Verfahren *2-Opt* zur Reduktion der Tourlänge (Abschnitt 3.5), sowie mehrere Algorithmen zur Lösung des zweidimensionalen, orthogonalen Packproblems (Kapitel 4) vorgestellt. Dabei wurden für das 2OPP das schnelle heuristische Verfahren TP_{MOD} (Abschnitt 4.4) sowie verschiedene exakte Anstätze implementiert (Abschnitte 4.5 und 4.6). Für die Bildung von Boxenstapeln wird die sehr schnelle und effektive Heuristik *FirstFitDecreasing* zur Lösung des eindimensionalen Bin-Packing-Problems eingesetzt (Abschnitt 4.2).

Es verbleibt die Aufgabe, durch Kombination dieser Algorithmen ein möglichst effektives Verfahren für das kombinierte Tourenplanungsproblem mit Zeitfenstern und Laderaumbepackung (VRTWLP) zu konstruieren. Es werden drei verschiedene Herangehensweisen zur Kombination der Tourenplanung mit einem Algorithmus für das 2OPP getestet. Es handelt sich dabei zum einen um zwei relativ naheliegende Möglichkeiten der Kombination: die integrierte Methode und die isolierte Methode. Zum anderen wurde unter Ausnutzung gewisser Eigenschaften des Tourenplanungsalgorithmus *I1* eine weitere vielversprechende Methode implementiert: die sogenannte sequentielle Methode. Diese drei Methoden der Kombination werden in den folgenden Abschnitten näher erläutert. Sie alle haben gemeinsam, dass die Bildung der Boxenstapel bereits vor Ausführung des kombinierten VRTWLP-Algorithmus stattfinden kann und deshalb unabhängig von den ansonsten eingesetzten Algorithmen ist. Somit hat man es stets nur noch mit dem zweidimensionalen Packproblem zu tun. Die Schnittstellen aller Packalgorithmen wurden so gestaltet, dass die Algorithmen beliebig ausgetauscht werden können. So kann an jeder Stelle jedes der in den Abschnitten 4.4, 4.5, 4.6 und 4.8 beschriebenen Verfahren für das zweidimensionale Packen eingesetzt werden. Anders verhält es sich mit dem Tourenplanungsalgorithmus *I1*, der für die integrierte und die sequentielle Methode für das VRTWLP (siehe Abschnitte 5.2 und 5.4) gesondert angepasst wurde.

Abhängig davon, welche der drei Methoden zur Kombination der Algorithmen angewandt wird, treten der Tourenplanungsalgorithmus und das Laderaumoptimierungsverfahren auf unterschiedliche Weisen miteinander in Interaktion. Eine Gemeinsamkeit dabei ist, dass zunächst eine oder mehrere Touren konstruiert werden und die Bepackung dieser Tour(en) danach überprüft wird. Die Arbeitsweise und das jeweils berechnete Ergebnis des Packverfahrens können einen negativen Einfluss auf die Tourenplanung und damit auch auf

die Qualität des Tourenplans haben. Um diesen Einfluss zu minimieren, passen die in dieser Arbeit implementierten Packalgorithmen ihre Arbeitsweise abhängig von der Belegung der Parameter *lifo* und *findBestSolution* den gegebenen Anforderungen an. Falls die LIFO-Reihenfolge der Bepackung eingehalten werden soll, müssen die Items in umgekehrter Reihenfolge ihres Vorkommens in der Tour gepackt werden, um später in der korrekten Reihenfolge wieder entpackt werden zu können. Das ist unabhängig davon, ob die integrierte, isolierte oder die sequentielle Methode zur Kombination der Algorithmen eingesetzt wird. Der Fall, in dem die LIFO-Reihenfolge nicht eingehalten werden muss, wird in den Abschnitten 5.2, 5.3 und 5.4 gesondert betrachtet.

5.1 Algorithmen für das kombinierte Tourenplanungs- und Laderaumoptimierungsproblem

Das kombinierte Tourenplanungs- und Laderaumoptimierungsproblem ist trotz zahlreicher Forschungsarbeiten zu den beiden einzelnen Teilproblemen bisher nur sehr wenig untersucht worden. Die einzigen bekannten Veröffentlichungen stammen aus den Jahren 2006 und später.

[Gendreau, Iori, Laporte, und Martello 2006] stellen einen Tabu-Search-Algorithmus zur Lösung des kombinierten Tourenplanungsproblems und dreidimensionalen Packproblems vor. Hierbei werden jeweils ein Tabu-Search-Verfahren für die Tourenplanung und die Stauraumbepackung eingesetzt. Das Verfahren benötigt für Probleminstanzen in der Größenordnung von bis zu 100 Kunden und weniger als 200 Packstücken Laufzeiten von bis zu zwei Stunden. Da im Anschluss an jede Einfügung eines oder mehrerer Kunden in eine Tour eine Überprüfung der Fahrzeugbepackung stattfindet, entspricht die Verknüpfung der Tourenplanung mit der Laderaumoptimierung der in dieser Arbeit eingesetzten integrierten Methode (siehe Abschnitt 5.2).

[Iori, Gonzalez, und Vigo 2007] schlagen einen exakten Algorithmus zur Lösung des Tourenplanungsproblems mit zweidimensionaler Laderaumoptimierung vor. Das VRP wird zunächst in Form eines ganzzahligen linearen Programms (ILP) formuliert und anschließend mittels eines Branch-and-Cut Verfahrens gelöst. Das Packproblem wird als untergeordnetes Problem behandelt und zunächst heuristisch, wenn notwendig aber auch exakt mittels Branch-and-Bound gelöst. Die experimentellen Ergebnisse zeigen, dass die Laufzeit des Verfahrens mit der Größe der Eingabeinstanz explosionsartig ansteigt und daher nur für sehr kleine Probleminstanzen mit Kundenanzahlen im unteren zweistelligen Bereich eingesetzt werden kann.

[Moura und Oliveira 2007] stellen zwei Verfahren zur Lösung des VRTWLP vor. Das Laderaumoptimierungsproblem wird in seiner dreidimensionalen Form mit einem heuristischen Ansatz nach dem sogenannten Wall Building Approach gelöst. Für das Tourenplanungsproblem wird eine GRASP-Heuristik verwendet. Die Autoren untersuchen zwei Methoden, die beiden Teilprobleme miteinander zu kombinieren. Die erste Methode baut zunächst einen Tourenplan auf ohne dabei die Validität der Containerbepackung zu überprüfen. Diese wird erst zum Schluss für alle Touren berechnet und Items, die nicht mehr in den Container passen, werden aus der Tour entfernt und anschließend neu geroutet. Die zweite Methode betrachtet die Tourenplanung als übergeordnetes und die Containerbeladung als untergeordnetes Problem. Immer wenn ersteres eine Tour konstruiert,

wird zweiteres aufgerufen, um die Zulässigkeit der Bepackung zu überprüfen. Die erstgenannte Methode entspricht im Wesentlichen der isolierten Methode (siehe Abschnitt 5.3). Die zweitgenannte Methode entspricht der integrierten Methode. Die Autoren vergleichen beide Ansätze experimentell miteinander, aber sie geben leider keine Laufzeiten für ihre Experimente an. Auch muss die Vergleichbarkeit der Ansätze in Frage gestellt werden, da sie unterschiedliche Nebenbedingungen einhalten.

Erst während der Bearbeitungszeit dieser Diplomarbeit ist der Artikel [Gendreau, Iori, Laporte, und Martello 2008] veröffentlicht worden. Die Autoren setzen für das Lösen des kombinierten Tourenplanungs- und zweidimensionalen Laderaumoptimierungsproblems auf einen hybriden Ansatz. Das VRP wird mit Hilfe eines Tabu-Search-Algorithmus gelöst, während für das 2OPP eine modifizierte Form der Heuristik *Touching-Perimeter* zum Einsatz kommt, welche die LIFO-Reihenfolge der Bepackung berücksichtigt. Der späte Veröffentlichungszeitpunkt dieses prinzipiell sehr interessanten Ansatzes hat es leider unmöglich gemacht, die Ergebnisse des Artikels in dieser Arbeit zu berücksichtigen. Einzig die in dieser Veröffentlichung angegebenen Eingabeinstanzen wurden verwendet, um die in dieser Arbeit implementierten Ansätze miteinander zu vergleichen und den in der Veröffentlichung angegebenen Ergebnissen gegenüberzustellen. Die hier verwendete Kombination des Tourenplanungsalgorithmus mit der Überprüfung der Stauraumbepackung ähnelt der aus [Gendreau, Iori, Laporte, und Martello 2006] und entspricht daher ebenfalls der integrierten Methode.

5.2 Die integrierte Methode

Die integrierte Methode zur Kombination eines Tourenplanungsalgorithmus mit einem Verfahren zur Laderaumoptimierung ist sicherlich die naheliegendste, wenn auch erwartungsgemäß die laufzeitintensivste der drei untersuchten Methoden. Die Idee dabei ist es, den Tourenplanungsalgorithmus wie gewohnt auszuführen und dabei in jedem Fall, in dem eine Tour konstruiert bzw. um einen Kunden erweitert werden soll, zusätzlich zur Überprüfung der Restriktionen Gewicht, Tourlänge und Tourdauer das zuvor gewählte Verfahren zur Stauraumbepackung auszuführen. Falls der Packalgorithmus die Packstücke der Kunden auf der Tour (in der vorgegebenen Ausführungszeit) nicht im Container unterbringen kann, muss diese Tour als nicht zulässig angesehen und verworfen werden.

Die Implementierung des Tourenplanungsalgorithmus *I1* (siehe Abschnitt 3.4) ist bereits darauf ausgerichtet, die Zulässigkeit einer Tour so selten wie möglich zu überprüfen. Um das zu erreichen, werden zunächst die potentiellen Einfügekombinationen samt entstehender Kosten in der Liste I gespeichert. Anschließend wird I in einer Reihenfolge durchlaufen, in der die günstigsten Einfügekombinationen zuerst untersucht werden. Der Durchlauf wird sofort abgebrochen sobald eine Einfügung als zulässig erkannt wurde. Für eine zusätzliche Überprüfung der Fahrzeugbepackung müssen daher nur geringfügige Änderungen an diesem Algorithmus vorgenommen werden.

Die Bestimmung des Seed-Kunden (Zeilen 5-15 in Algorithmus 3.4) bleibt unverändert, bis auf die Tatsache, dass zusätzlich zu den Restriktionen Transportgewicht, Tourlänge und Tourdauer (Zeile 12) die Fahrzeugbepackung berechnet werden muss. Falls die Boxenstapel des Seed-Kunden nicht gepackt werden können, wird dieser Kunde als nicht packbar markiert und verworfen.

Als ähnlich unproblematisch erweist sich die Erweiterung der zweiten Phase des Algorithmus. Diese besteht darin, zusätzlich zu den Gewichts- (Zeile 29) und Längenrestriktionen (Zeile 33) eventuell die Fahrzeugbepackung durchzuführen. Falls keine LIFO-Reihenfolge der Bepackung gefordert ist, ist der Erfolg der Laderaumbepackung unabhängig davon, an welcher Position p der offenen Tour ein Kunde u eingefügt werden soll. In diesem Fall kann also u zur Kunden-Tabuliste hinzugefügt werden sobald der Packalgorithmus für die Einfügung von u an einer beliebigen Position p scheitert. Anders verhält es sich bei geforderter LIFO-Reihenfolge. Da hier der Erfolg der Bepackung zusätzlich von der Einfügeposition p abhängt, können Kunden nach erstmaligem Scheitern des Packalgorithmus nicht zur Tabuliste hinzugefügt werden. Die dadurch beabsichtigte Laufzeitreduktion kann in diesem Fall nicht erzielt werden.

Der eingesetzte Packalgorithmus wird innerhalb der integrierten Methode stets mit dem Parameter *findBestSolution* = falsch aufgerufen, denn in diesem Modus ist es nur von Interesse, vollständige Lösungen zu erhalten.

Da insbesondere beim Einsatz exakter Verfahren zur Laderaumbepackung mit extrem langen Laufzeiten pro Eingabeinstanz gerechnet werden muss, sollten solche Algorithmen unbedingt mit einer beschränkten Laufzeit (einem Timeout) sowie in der Form eines gestaffelten Verfahrens (siehe Abschnitt 4.8 für eine genauere Beschreibung) ausgeführt werden.

Nachdem die Touren und die zugehörigen LKW-Bepackungen berechnet wurden, wird versucht, die Touren mittels *2-Opt* zu verbessern (siehe Abschnitt 3.5). Falls die LIFO-Reihenfolge der Bepackung gefordert ist, muss nach einer eventuellen Umordnung der Tourreihenfolge die Fahrzeugbepackung neu berechnet werden.

5.3 Die isolierte Methode

Die isolierte Methode zur Kombination der Algorithmen zur Tourenplanung und zur Laderaumoptimierung verfolgt einen zur integrierten Methode entgegengesetzten Ansatz. Hierbei werden die beiden Algorithmen sequentiell und losgelöst voneinander ausgeführt.

Wie im Folgenden gezeigt wird, erfordert die isolierte Methode die Möglichkeit, dass für den Fall des Scheiterns des eingesetzten 2OPP-Algorithmus trotzdem die beste erzielte Bepackung ausgewiesen wird. Deshalb wurden alle in dieser Arbeit implementierten 2OPP-Algorithmen um die Fähigkeit erweitert, das nach gewissen Kriterien beste erzielte Packergebnis zwischenzuspeichern und zu einem späteren Zeitpunkt abrufbar zu machen. Diese Fähigkeit wird duch das Setzen des Parameters *findBestSolution* aktiviert. Die Güte einer solchen (Zwischen-)lösung des 2OPP wird anhand zweier Kriterien bestimmt:

1. Die Anzahl Kunden, für die alle Boxenstapel gepackt werden konnten (je mehr Kunden desto besser die Lösung). Bei zwei Lösungen mit gleicher Anzahl Kunden entscheidet

2. die von den Boxenstapeln eingenommene Grundfläche (wieder: je größer die Fläche desto besser die Lösung).

Die erste Idee hinter diesen Kriterien ist es, Touren mit möglichst vielen Kunden zu konstruieren. Damit wird das angestrebte Ziel verfolgt, so wenige Touren wie möglich zu generieren. Zweitens werden bei gleich vielen vollständig gepackten Kunden Lösungen bevorzugt, deren verbrauchte Grundfläche des LKW-Frachtraums groß ist. Damit soll eine möglichst hohe Auslastung der LKW erzielt werden.

Der Ablauf des Verfahrens wird mit Hilfe von Algorithmus 12 erläutert.

Algorithmus 12: Isolierte Methode für das VRTWLP

```
1:  procedure ISOLATED(U)
2:      R' ← ∅                                   ▷ Menge der Touren mit gepackten Kunden
3:      Ū ← ∅      ▷ Menge der Kunden, die nicht in einer Tour untergebracht oder nicht
    gepackt werden können
4:      repeat
5:          R ← Führe den Tourenplanungsalgorithmus ohne Berücksichtigung der Stau-
    raumbepackung für alle Kunden in U aus
6:          for all r ← R do
7:              Wende 2-Opt auf die Tour r an
8:              Packe alle Kunden der Tour r
9:              Erfrage bestes erzieltes Packergebnis vom 2OPP-Algorithmus
10:             if Kein Kunde aus r konnte gepackt werden then
11:                 if Das eingesetzte 2OPP-Verfahren ist TP_MOD then
12:                     Entferne alle Kunden in r aus U
13:                     Füge alle Kunden in r zu Ū hinzu
14:                 else
15:                     Ermittle denjenige Kunden i ∈ r, dessen Packsücke vom eingesetzen
    2OPP-Verfahren als erste gepackt werden
16:                     Entferne i aus r und aus U und versuche die zugehörigen Boxenstapel
    in einen leeren LKW zu packen
17:                     if s konnte gepackt werden then
18:                         Bilde Stichtour r' ← (0, i, 0) und füge sie zu R' hinzu
19:                     else
20:                         Füge i zu Ū hinzu
21:             else
22:                 Bilde neue Tour r' mit allen Kunden des besten Ergebnisses
23:                 Füge r' zu R' hinzu
24:                 Entferne alle Kunden in r' aus U
25:     until U = ∅
26:     return R'
```

Der Parameter U der isolierten Methode ist die Menge aller zu verarbeitenden Kunden. Es wird davon ausgegangen, dass die Packstücke jedes Kunden bereits im Vorfeld gestapelt worden sind. In Zeilen 2 und 3 werden die Rückgabemengen R' aller berechneten Touren sowie $\bar{U}$ aller nicht-routbaren Kunden initialisiert. Solange U nicht leer ist, werden nun iterativ die Schritte in Zeilen 5-24 ausgeführt. Zunächst generiert der eingesetzte Touren-

planungsalgorithmus die Menge R aller Touren, ohne dabei die LKW-Bepackung durchzuführen (Zeile 5). Jede Tour $r \in R$ wird einer Nachoptimierung mittels *2-Opt* unterzogen (Zeile 7). Nun wird versucht, die Boxenstapel der Kunden der Tour r mit einem der Algorithmen zur Stauraumbeladung im LKW zu verteilen (Zeile 8). Da man davon ausgehen kann, dass die Packstücke einiger oder sogar aller Kunden einer Tour in der für den 2OPP-Algorithmus vorgegebenen Zeitspanne nicht im LKW untergebracht werden können, wird das Laderaumoptimierungsverfahren mit dem Parameter *findBestSolution* = wahr aufgerufen, damit das nach den beiden oben genannten Kriterien beste erzielte Packergebnis anschließend zur Verfügung steht (Zeile 9). Dieses beste Ergebnis kann entweder die Packstücke

1. keines Kunden der Tour r, oder
2. aller $|r|$ Kunden der Tour, oder
3. einer Anzahl $k < |r|$ von Kunden der Tour

beinhalten.

Im ersten Fall (Zeilen 10-20) muss unterschieden werden, ob als Laderaumoptimierungsalgorithmus TP_{MOD} oder eines der in den Abschnitten 4.5 und 4.6 vorgestellten exakten Verfahren für das 2OPP eingesetzt wird. TP_{MOD} hat im Gegensatz zu den anderen Verfahren die Eigenschaft, dass es Kunden atomar behandelt, d.h. es können entweder alle Packstücke eines Kunden gepackt werden, oder es werden alle Stapel dieses Kunden aus dem Container entfernt und der Algorithmus fährt mit dem nächsten Kunden der Tour fort. In dem Fall, dass das Verfahren die Packstücke keines Kunden einer Tour packen kann, hat es (in sequentieller Reihenfolge) versucht, die Güter jedes Kunden in einem leeren Container unterzubringen. Dies erlaubt den Schluss, dass der Algorithmus auch keinen der Kunden dieser Tour packen könnte, wenn dieser Kunde allein gepackt werden sollte. Deshalb können alle Kunden der Tour aus der Menge U entfernt und zur Menge $\bar{U}$ der nicht routbaren Kunden hinzugefügt werden (Zeilen 12 und 13).

Die Algorithmen MV_{MOD} und $LMAO_{MOD}$ haben diese Eigenschaft von TP_{MOD} nicht, da sie trotz einer Vorsortierung der Packstücke gemäß der Reihenfolge der Tour diese Packstücke nicht strikt sequentiell abarbeiten. Somit kann die Verarbeitung von Kunden nicht als atomar bezeichnet werden. Die Güter eines Kunden, die im Kontext weiterer Kunden in der Tour nicht gepackt werden können, könnten deshalb unter Umständen doch gepackt werden, wenn sie allein betrachtet würden.

Falls also nicht das Verfahren TP_{MOD} eingesetzt wird, kann erst dann mit Gewissheit gesagt werden, dass die Güter eines Kunden nicht packbar sind, wenn diese allein für sich betrachtet wurden. Es ist naheliegend anzunehmen, dass in einer Tour, deren Kunden alle nicht gepackt werden konnten, derjenige Kunde das Problem verursacht, dessen Packstücke vom Packalgorithmus als erste betrachtet werden und das Packen der Packstücke anderer Kunden unter Umständen unmöglich machen. Der zuerst betrachtete Kunde i wird in Zeile 15 ermittelt. Welcher der $|r|$ Kunden der Tour dies ist, hängt von der Vorsortierung der Packstücke innerhalb des 2OPP-Verfahrens ab: Falls die LIFO-Reihenfolge eingehalten werden muss, werden die Packstücke des letzten Kunden der Tour zuerst gepackt. Ansonsten sind es wegen des im isolierten Modus gesetzten Flags *findBestSolution* die Güter des ersten Kunden der Tour. Der Kunde i wird nun einer Sonderbehandlung

unterzogen, indem versucht wird, die Packstücke dieses einen Kunden allein in einem leeren LKW unterzubringen (Zeile 16). Falls dies gelingt, wird eine Stichtour $r' = (0, i, 0)$ gebildet und zur Ergebnismenge R' aller gebildeten Touren hinzugefügt (Zeile 18). Sonst wird der Kunde i zur Menge $\bar{U}$ der nicht-routbaren Kunden hinzugefügt (Zeile 20). In jedem der beiden Fälle wird i aus der Eingabemenge U entfernt und somit im weiteren Verlauf des Algorithmus nicht noch einmal betrachtet.

Weniger kompliziert gestalten sich die oben genannten Fälle zwei und drei. In diesen Fällen konnten die Packstücke von k Kunden, $1 \leq k \leq |r|$ der Tour r vollständig im LKW verstaut werden. Es wird eine neue Tour r' der Länge k gebildet (Zeile 22), indem die Packstücke aller nicht vollständig gepackten Kunden aus r entfernt werden. Diese nicht gepackten Kunden verbleiben in der Menge U, um möglicherweise später in anderen Touren untergebracht zu werden, während die neue Tour r' zur Ergebnismenge R' hinzugefügt wird (Zeile 23). Die k erfolgreich gepackten Kunden aus r' werden aus U entfernt (Zeile 24).

5.4 Die sequentielle Methode

Die Idee für die sequentielle Methode zur Lösung des VRTWLP entstand durch die Identifikation der Schwächen der integrierten und der isolierten Methode.

Die große Schwäche der integrierten Methode (Abschnitt 5.2) besteht darin, dass das Verfahren Probleme hat, Ergebnisse in akzeptabler Laufzeit zu liefern, falls ein 2OPP-Algorithmus mit langer Laufzeit und zu großzügig gewähltem Timeout eingesetzt wird. Die direkte Verknüpfung des Tourenplanungsalgorithmus mit der Laderaumoptimierung sorgt dafür, dass die Anzahl Aufrufe des 2OPP-Verfahrens direkt von der Arbeitsweise der Tourenplanung abhängt. Bereits der schnelle und speziell an die integrierte Methode angepasste Tourenplanungsalgorithmus *I1* ruft im Kontext der integrierten Methode das eingesetzte Packverfahren $O(ln^2) = O(n^3)$ mal auf.

Fortgeschrittene Tourenplanungsalgorithmen wie Metaheuristiken und evolutionäre Verfahren sind bestrebt, den Lösungsraum intensiv abzusuchen. Es werden viele Lösungen generiert, hinsichtlich Zulässigkeit überprüft und gegebenenfalls wieder verworfen. Bei Verwendung der integrierten Methode würde sich dieses Vorgehen in einer erhöhten Anzahl von Aufrufen des 2OPP-Moduls und damit erwartungsgemäß in einem großen Laufzeitzuwachs niederschlagen. Hier wäre die isolierte Methode (Abschnitt 5.2) hinsichtlich der benötigten Rechenzeit im Vorteil. Die Isolation der Tourenplanung von der Stauraumbepackung sorgt für Unabhängigkeit der Algorithmen voneinander. Das bedeutet, dass aufwändige Tourenplanungsalgorithmen nicht notwendigerweise zu einem erhöhten Laufzeitbedarf des Packalgorithmus führen müssen.

Die isolierte Methode hat gegenüber der integrierten Methode den Nachteil einer reduzierten Lösungsqualität. Diese lässt sich damit erklären, dass die von der Tourenplanung ausgewiesenen, potentiell guten Lösungen regelmäßig zerrissen werden, indem einzelne Kunden aus dem berechneten Tourenplan entfernt und der Tourenplanung für einen weiteren Durchlauf neu zugeführt werden. Die Entscheidung, welche Kunden aus Touren entfernt werden, basiert zudem nur auf den Ergebnissen der Stauraumbepackung – es werden weder die örtliche Lage noch die Zeitfenster der Kunden in Betracht gezogen. Deshalb kann es leicht passieren, dass nach einem Durchlauf der Tourenplanung Kunden aus meh-

reren Touren entfernt werden, die entweder örtlich oder zeitlich zueinander inkompatibel sind und deshalb die Bildung eigener, kurzer Touren erfordern.

Unter Ausnutzung der Besonderheit, dass *I1* ein sequentielles Tourkonstruktionsverfahren ist, wurde die sequentielle Methode für das VRTWLP entwickelt. Die Idee besteht darin, *I1* die einzelnen Touren – wie im Falle des isolierten Verfahrens – zunächst ohne Überprüfung der Stauraumbepackung berechnen zu lassen. Unmittelbar nachdem eine Tour von *I1* erstellt und mittels *2-Opt* nachoptimiert wurde (und nicht wie bei der isolierten Methode, erst nachdem alle Touren der Instanz berechnet wurden), werden die Boxenstapel aller Kunden dieser Tour gepackt. Falls nicht die Packstücke aller Kunden der Tour gepackt werden können, wird genau wie in der isolierten Methode das vom eingesetzten 2OPP-Algorithmus bislang beste erzielte Packergebnis abgefragt. Um dies tun zu können, wird der jeweils eingesetzte Laderaumoptimierungsalgorithmus mit dem Parameter *findBestSolution* = wahr aufgerufen. Alle Kunden, deren Packstücke nicht vollständig beladen werden können, werden unverzüglich wieder der Menge U der noch zu verarbeitenden Kunden (siehe Algorithmus 1, Seite 30) zugeführt.

Der Vorteil dieser Vorgehensweise gegenüber der isolierten Methode besteht darin, dass Kunden, die nicht gepackt werden können, sofort wieder in die Menge aller noch nicht gerouteten Kunden eingefügt werden. Dies soll dem Tourenplanungsalgorithmus größere Spielräume bezüglich der Integration dieser Kunden in später erstellte Touren bringen. Der Vorteil gegenüber der integrierten Methode ist eine große Laufzeitersparnis, da der Packalgorithmus nur ein Mal pro generierter Tour aufgerufen wird, statt ein Mal pro potentieller Einfügung eines Kunden an eine der möglichen Positionen einer offenen Tour.

Die sequentielle Methode wurde durch Anpassung des Algorithmus *I1* gemäß oberer Beschreibung umgesetzt. Im Pseudocode von *I1* (Algorithmus 1) genügt hierfür eine geringfügige Modifikation: Es muss Zeile 40 durch die in Algorithmus 13 angegebenen Zeilen ersetzt werden.

Man beachte, dass Algorithmus 13 – bis auf kleine Unterschiede – den Zeilen 8 bis 24 in Algorithmus 12 entspricht. Auch hier wurde dem Fall besondere Aufmerksamkeit gewidmet, dass kein einziger Kunde einer Tour gepackt werden kann. Das Vorgehen in diesem Fall (Zeilen 4-14 in Algorithmus 13) entspricht der Vorgehensweise in der isolierten Methode (Zeilen 10-20 in Algorithmus 12).

Algorithmus 13: Anpassung von I1 an die sequentielle Methode für das VRTWLP

1: Wende *2-Opt* auf die Tour r an
2: Packe alle Kunden der Tour r
3: Erfrage bestes erzieltes Packergebnis vom 2OPP-Algorithmus
4: **if** Kein Kunde aus r konnte gepackt werden **then**
5: **if** Das eingesetzte 2OPP-Verfahren ist TP_{MOD} **then**
6: Entferne alle Kunden in r aus U
7: Füge alle Kunden in r zu $\bar{U}$ hinzu
8: **else**
9: Ermittle denjenige Kunden $i \in r$, dessen Packsücke vom eingesetzen 2OPP-Verfahren als erste gepackt werden
10: Entferne i aus r und aus U und versuche die zugehörigen Boxenstapel in einen leeren LKW zu packen
11: **if** i konnte gepackt werden **then**
12: Bilde Stichtour $r' = \{i\}$ und füge sie zu R' hinzu
13: **else**
14: Füge i zu $\bar{U}$ hinzu
15: **else**
16: Bilde neue Tour r' mit allen Kunden des besten Ergebnisses
17: Füge r' zu R hinzu

Kapitel 6

Experimentelle Evaluation

Nachdem in den Kapiteln 3, 4 und 5 Algorithmen für die Tourenplanung, Laderaumoptimierung und für die Kombination dieser beiden Problemklassen zu einem Gesamtverfahren für das VRTWLP beschrieben wurden, werden in diesem Kapitel die vorgestellten Ansätze hinsichtlich ihrer Lösungsqualität und Laufzeit getestet, miteinander verglichen und Ansätzen aus der Literatur gegenübergestellt.

Zunächst werden in Abschnitt 6.1 Benchmarks beschrieben, die durchgeführt wurden, um die implementierten exakten Verfahren MV_{MOD} (Abschnitt 4.5) und $LMAO_{MOD}$ (Abschnitt 4.6) in Kombination mit der Routine *RLMD-Corner* bzw. der speziellen Datenstruktur *RLMD-Corner-DS* zur Lösung des zweidimensionalen orthogonalen Packproblems (2OPP) miteinander zu vergleichen. Auf diese Weise soll das schnellste exakte Verfahren bestimmt werden, um in den weiteren Tests innerhalb der Ansätze für das Gesamtproblem eingesetzt zu werden. Für diese Benchmarks werden Eingabeinstanzen aus der Literatur eingesetzt.

In Abschnitt 6.2 werden die integrierte (Abschnitt 5.2), isolierte (Abschnitt 5.3) und sequentielle (Abschnitt 5.4) Methode zur Lösung des gesamten VRTWLP unter Verwendung des heuristischen Verfahrens TP_{MOD} (Abschnitt 4.4) sowie des gestaffelten 2OPP-Verfahrens (Abschnitt 4.8) verglichen. Das Ziel ist es, eine oder mehrere Kombinationen der eingesetzten Algorithmen zu identifizieren, mit denen besonders gute Ergebnisse erzielt werden können. Zu diesem Zweck werden Benchmark-Eingabeinstanzen aus der Literatur verwendet, die es ebenfalls erlauben, einen Vergleich der hier vorgestellten Algorithmen mit einem Verfahren aus der Literatur zum kombinierten Tourenplanungs- und Laderaumoptimierungsproblem vorzunehmen.

Zum Abschluss des Kapitels werden in Abschnitt 6.3 ausgewählte Kombinationen und Parametrisierungen der in dieser Arbeit implementierten Verfahren auf Real-World-Sendungsdaten getestet. Um den Einfluss der Drehbarkeit von Items auf die Laufzeit und die Lösungsqualität zu überprüfen, werden die Benchmarks sowohl mit festgelegter als auch mit freier Orientierung durchgeführt.

6.1 Ermittlung des schnellsten exakten 2OPP-Verfahrens

Im Folgenden werden Experimente beschrieben, die in erster Linie dem Zweck dienen, das schnellste der in Kapitel 4 beschriebenen exakten Verfahren für das 2OPP zu ermitteln. Hierzu gehören die Verfahren MV_{MOD} (Abschnitt 4.5) und $LMAO_{MOD}$ (Abschnitt 4.6), welches zur Ermittlung von potentiellen Platzierungen in einem gegebenen Packmuster entweder die Prozedur *RLMD-Corner* oder die spezielle Datenstrukktur *RLMD-Corner-DS* einsetzt. Zur Unterscheidung der beiden letztgenannten Ansätze wird $LMAO_{MOD}$ in Kombination mit *RLMD-Corner-DS* im Folgenden $LMAO_{MOD}$+*DS* genannt, bei Verwendung von *RLMD-Corner* wird hingegen die Bezeichnung $LMAO_{MOD}$ beibehalten.

6.1.1 Die Eingabedaten

Für die in diesem Abschnitt beschriebenen Experimente wurden Benchmarkinstanzen aus [Clautiaux, Carlier, und Moukrim 2007] verwendet. Diese wurden von den Autoren eigens generiert, um einen Vergleich zwischen den Algorithmen *MV*, *LMAO* und *TSBP* (einem weiteren, in dieser Arbeit nicht beschriebenen Algorithmus) zu ermöglichen. Die Daten umfassen 42 Eingabeinstanzen, die sich hinsichtlich der Itemanzahl und Packungsdichte voneinander unterscheiden. Die Containerausmaße sind in allen Instanzen gleich, nämlich $(W, H) = (20, 20)$, die Anzahl zu packender Items bewegt sich im Bereich 10 bis 23. 13 der 42 Instanzen sind lösbar, 22 sind nicht lösbar; für die restlichen 7 Instanzen machen die Autoren keine Angaben zur Lösbarkeit. Da Clautiaux et al. einen Effekt abhängig von der gegebenen Packungsdichte (d.h. abhängig von dem Wert $\frac{\sum_{i \in I} w_i h_i}{WH}$, wobei I die Menge aller Items einer Eingabeinstanz ist) erwarten, geben sie für ihre Instanzen die Packungsdichte in Form eines Wertes ϵ an. Für diesen gilt:

$$\sum_{i \in I} w_i h_i = (1 - \epsilon) WH \tag{6.1}$$

Einfach ausgedrückt gilt also: je kleiner ϵ desto höher die Packungsdichte, bei $\epsilon = 0$ entspricht die Grundfläche des Containers der Summe der Flächen der Items.

Die Eingabeinstanzen können online von der Seite `http://www2.lifl.fr/~clautiau/pmwiki/pmwiki.php?n=Research.Benchmarks` bezogen werden.

6.1.2 Experimentelles Setup

Alle Experimente wurden auf einem AMD Athlon XP 2000+ (Taktfrequenz 1667 MHz) mit 1 GB Hauptspeicher unter der Linux-Distribution Debian 4.0 (Etch) durchgeführt. Zur Übersetzung der Quelltexte in Bytecode und zur Ausführung wurde das Sun JDK 6 eingesetzt.

Um so weit wie möglich eine Vergleichbarkeit der Ergebnisse mit denen aus [Clautiaux, Carlier, und Moukrim 2007] sicherzustellen, wurde zunächst – genau wie im genannten Artikel – für jede Eingabeinstanz eine maximale Laufzeit von 15 Minuten veranschlagt. Nach Ablauf dieser Zeitspanne wurde eine Instanz als vom eingesetzten Algorithmus nicht

lösbar angesehen. Anschließend wurde das beste der Verfahren zusätzlich mit einer maximalen Laufzeit von einer Stunde pro Instanz getestet, um einen Einfluss der zusätzlichen Laufzeit auf die Menge der gelösten Instanzen zu überprüfen.

6.1.3 Ergebnisse

Die nach 15 Minuten Laufzeit von $LMAO_{MOD}$ und $LMAO_{MOD}$+DS berechneten Ergebnisse sind enttäuschend. Da sich die in Abschnitt 4.6.5 beschriebene Idee für eine untere Schranke letztendlich als unzulässig erwiesen hat, konnte diese nicht eingesetzt werden. Ohne untere Schranke konnten beide Verfahren in der vorgegebenen Zeit keine einzige der 42 Eingabeinstanzen lösen.

Deutlich besser fallen die Ergebnisse der Benchmarks für MV_{MOD} aus. Der Algorithmus kann innerhalb von 15 Minuten immerhin 8 der 42 Probleminstanzen lösen. Dies entspricht genau dem Ergebnis, welches das Verfahren MV (auf welchem MV_{MOD} beruht) in experimentellen Untersuchungen von Clautiaux et al. erzielen konnte. Nach einer Erhöhung der maximalen Laufzeit von 15 Minuten auf eine Stunde pro Instanz konnte MV_{MOD} bereits 11 Instanzen lösen. Tabelle 6.1 fasst die Benchmarkergebnisse von MV_{MOD} zusammen. Die Spalten 1 bis 4 geben die Eigenschaften der jeweiligen Instanz an: den Namen der Instanz, die Anzahl zu platzierender Items, den Wert ϵ, sowie eine Angabe, ob die Instanz lösbar ist. Instanzen, für die Clautiaux et al. keine Angabe zur Lösbarkeit machen, sind hier mit dem Kürzel „k. A.“ gekennzeichnet. Die Spalten rechts vom senkrechten Trennstrich beinhalten die Berechnungsergebnisse von MV_{MOD}: die Anzahl erzeugter Suchbaumknoten, die Anzahl aufgrund der unteren Schranke verworfener Suchbaumknoten, die Angabe, ob die Instanz gelöst wurde (wobei nicht lösbare Instanzen gelöst sind, sobald der gesamte Suchbaum enumeriert wurde), und schließlich die benötigte Laufzeit.

Wie bereits erwähnt wurde, konnten nach 15 Minuten 8 der 42 Instanzen gelöst werden, nämlich alle 6 Instanzen mit 10 Items (die alle nicht lösbar sind) und zusätzlich die beiden lösbaren Instanzen E05F18 und E04F19, welche 18 bzw. 19 Items enthalten. Durch die Ausdehnung der Laufzeit auf eine Stunde konnte MV_{MOD} 3 weitere Instanzen mit je 15 Items lösen, nämlich E00N15, E07F15 und E20F15.

Auffällig ist die Effektivität der unteren Schranke von MV_{MOD}. Offensichtlich können aufgrund der Schranke viele Suchbaumknoten frühzeitig verlassen werden, was die gute Performanz von MV_{MOD} gegenüber $LMAO_{MOD}$ erklärt. Sporadische Tests von $LMAO_{MOD}$ und $LMAO_{MOD}$+DS unter Einbeziehung der in Abschnitt 4.6.5 vorgestellten und leider inkorrekten unteren Schranke haben diese Vermutung bestätigt. $LMAO_{MOD}$ konnte mit MV_{MOD} in etwa gleichziehen und nach 15 Minuten sogar eine Instanz mehr lösen als MV_{MOD} (wenn auch die Laufzeiten im Allgemeinen etwas schlechter waren).

Zur Prüfung der Korrektheit der beiden Verfahren $LMAO_{MOD}$ und $LMAO_{MOD}$+DS sowie zur Überprüfung des Verhaltens der Algorithmen bei Verwendung der leider im Allgemeinen nicht zulässigen unteren Schranke wurden alle Instanzen mit 10 Items zusätzlich mit einem Timeout von 10 Stunden durchgerechnet und haben die Algorithmen $LMAO_{MOD}$ und $LMAO_{MOD}$+DS als korrekt arbeitend bestätigt. Tabelle 6.2 fasst die Ergebnisse dieser ausgewählten Berechnungen zusammen.

Wenn auch $LMAO_{MOD}$ und $LMAO_{MOD}$+DS mangels unterer Schranke wesentlich weniger performant sind als MV_{MOD}, zeigen die in Tabelle 6.2 zusammengefassten Ergebnisse

Instanz	Items	ϵ	Lösbar?	Knoten	Untere Schranke	Gelöst?	Laufzeit (s)
E00N10	10	00	Nein	52.093	45.398	Ja	0,5
E03N10	10	03	Nein	1.492.183	1.087.287	Ja	9,6
E07N10	10	07	Nein	2.180.426	1.086.180	Ja	14,3
E10N10	10	10	Nein	8.499.805	4.170.782	Ja	56,0
E13N10	10	13	Nein	18.651.990	5.620.589	Ja	123,9
E15N10	10	15	Nein	37.118.812	12.204.229	Ja	246,0
E00N15	15	00	Nein	355.438.872	326.611.558	Ja	2.616,9
E03N15	15	03	Nein	468.702.012	378.438.885	Nein	3.600,0
E04F15	15	04	Ja	422.352.458	295.271.663	Nein	3.600,0
E04N15	15	04	Nein	467.021.508	375.559.513	Nein	3.600,0
E05F15	15	05	Ja	452.275.140	289.227.930	Nein	3.600,0
E05N15	15	05	Nein	458.162.243	288.127.235	Nein	3.600,0
E05X15	15	05	k. A.	443.069.175	279.737.251	Nein	3.600,0
E07F15	15	07	Ja	451.302.068	349.505.127	Ja	3.436,4
E07N15	15	07	Nein	473.875.040	372.058.182	Nein	3.600,0
E07X15	15	07	k. A.	464.270.419	329.270.399	Nein	3.600,0
E08F15	15	08	Ja	454.468.201	326.846.119	Nein	3.600,0
E08N15	15	08	Nein	472.689.571	345.057.238	Nein	3.600,0
E10N15	15	10	Nein	480.136.415	264.960.843	Nein	3.600,0
E10X15	15	10	k. A.	481.048.306	271.493.172	Nein	3.600,0
E13N15	15	13	Nein	515.444.970	376.378.696	Nein	3.600,0
E13X15	15	13	k. A.	447.970.620	195.298.166	Nein	3.600,0
E15N15	15	15	Nein	445.173.299	328.472.930	Nein	3.600,0
E20F15	15	20	Ja	242.117.253	193.276.134	Ja	1.945,4
E20X15	15	20	k. A.	442.203.659	298.485.833	Nein	3.600,0
E03N16	16	03	Nein	433.649.466	365.523.550	Nein	3.600,0
E02F17	17	02	Ja	480.088.708	431.599.567	Nein	3.600,0
E03N17	17	03	Nein	412.228.796	363.271.708	Nein	3.600,0
E04F17	17	04	Ja	423.231.169	351.997.988	Nein	3.600,0
E04N17	17	04	Nein	429.078.703	362.055.969	Nein	3.600,0
E05N17	17	05	Nein	500.835.050	435.803.070	Nein	3.600,0
E03X18	18	03	k. A.	405.185.203	370.363.439	Nein	3.600,0
E04N18	18	04	Nein	448.704.325	402.061.802	Nein	3.600,0
E05F18	18	05	Ja	2.163.214	1.744.265	Ja	16,9
E04F19	19	04	Ja	35.847.837	23.895.239	Ja	317,0
E02F20	20	02	Ja	407.456.026	353.757.994	Nein	3.600,0
E02N20	20	02	Nein	404.095.742	353.321.662	Nein	3.600,0
E04F20	20	04	Ja	410.282.155	309.504.841	Nein	3.600,0
E05F20	20	05	Ja	398.838.758	342.481.105	Nein	3.600,0
E02F22	22	02	Ja	400.422.520	358.061.612	Nein	3.600,0
E00N23	23	00	Nein	411.605.883	378.744.379	Nein	3.600,0
E00X23	23	00	k. A.	401.953.039	358.658.199	Nein	3.600,0

Tabelle 6.1: Benchmarkergebnisse MV_{MOD}: 42 Instanzen mit 10 bis 23 Items, maximale Laufzeit eine Stunde pro Instanz

Algorithmus	ϵ	Gelöst?	Knoten	Untere Schranke	Laufzeit (s)
MV_{MOD}	0	Ja	52.093	45.398	0,5
MV_{MOD}	3	Ja	1.492.183	1.087.287	9,6
MV_{MOD}	7	Ja	2.180.426	1.086.180	14,3
MV_{MOD}	10	Ja	8.499.805	4.170.782	56,0
MV_{MOD}	13	Ja	18.651.990	5.620.589	123,9
MV_{MOD}	15	Ja	37.118.812	12.204.229	246,0
$LMAO_{MOD}$	0	Ja	538.739.572	–	6.510,6
$LMAO_{MOD}$	3	Nein	2.867.070.431	–	36.000,0
$LMAO_{MOD}$	7	Ja	797.717.750	–	9.625,1
$LMAO_{MOD}$	10	Nein	3.000.146.330	–	36.000,0
$LMAO_{MOD}$	13	Nein	2.836.950.200	–	36.000,0
$LMAO_{MOD}$	15	Nein	2.897.631.461	–	36.000,0
$LMAO_{MOD}$+DS	0	Ja	538.739.572	–	2.549,6
$LMAO_{MOD}$+DS	3	Ja	3.170.413.706	–	18.846,2
$LMAO_{MOD}$+DS	7	Ja	797.717.750	–	3.772,4
$LMAO_{MOD}$+DS	10	Ja	5.049.478.418	–	26.347,8
$LMAO_{MOD}$+DS	13	Ja	3.836.291.607	–	19.245,1
$LMAO_{MOD}$+DS	15	Nein	6.852.750.982	–	36.000,0
$LMAO_{MOD}$+LB	0	Ja	219.478	58.310	2,3
$LMAO_{MOD}$+LB	3	Ja	998.800	344.870	9,7
$LMAO_{MOD}$+LB	7	Ja	9.348.946	1.491.774	85,8
$LMAO_{MOD}$+LB	10	Ja	48.172.623	7.588.969	471,5
$LMAO_{MOD}$+LB	13	Ja	30.350.733	5.176.746	261,2
$LMAO_{MOD}$+LB	15	Ja	75.512.588	9.937.399	643,1
$LMAO_{MOD}$+DS+LB	0	Ja	219.478	58.310	2,3
$LMAO_{MOD}$+DS+LB	3	Ja	998.800	344.870	9,5
$LMAO_{MOD}$+DS+LB	7	Ja	9.348.946	1.491.774	86,4
$LMAO_{MOD}$+DS+LB	10	Ja	48.172.623	7.588.969	473,9
$LMAO_{MOD}$+DS+LB	13	Ja	30.350.733	5.176.746	261,5
$LMAO_{MOD}$+DS+LB	15	Ja	75.512.588	9.937.399	651,3

Tabelle 6.2: Benchmark der exakten 2OPP-Algorithmen: 6 nicht lösbare Instanzen mit jeweils 10 Items, maximale Laufzeit 10 Stunden pro Instanz

die Effektivität der speziellen Datenstruktur *RLMD-Corner-DS* zur Verwaltung von Platzierungen. Im Durchschnitt hat $LMAO_{MOD}$ 80.796 Suchbaumknoten pro Sekunde beim Berechnen der sechs Testinstanzen mit jeweils 10 Items erzeugt. $LMAO_{MOD}$+*DS* war mehr als doppelt so schnell: Es hat durchschnittlich 189.633 Knoten pro Sekunde erzeugt. Des Weiteren zeigen die Ergebnisse, dass die Effektivität der Verfahren wesentlich gesteigert würde falls die untere Schranke eingesetzt werden könnte. Die Verfahren $LMAO_{MOD}$ und $LMAO_{MOD}$+*DS* mit unterer Schranke sind in Tabelle 6.2 durch den Suffix „+*LB*“ gekennzeichnet. Doch auch $LMAO_{MOD}$+*LB* und $LMAO_{MOD}$+*DS*+*LB* sind dem Algorithmus MV_{MOD} unterlegen, was sich mit der höheren Effektivität der Berechnung von Itemplatzierungen mittels *2D-Corners* in MV_{MOD} begründen lässt. Für ein Packmuster P mit k Items liefert *2D-Corners* stets $O(k)$ Eckpunkte. Hingegen liefern *RLMD-Corner* und *RLMD-Corner-DS* zwar nur einen RLMD-Eckpunkt pro Packmuster, dieser stammt jedoch aus einer Menge von bis zu k^2 Punkten, von denen viele irrelevant sind, da sie nicht zur Menge der normalen Positionen gehören und deshalb von anderen Eckpunkten dominiert werden.

Zusammenfassend kann festgestellt werden, dass sich MV_{MOD} als das schnellste der vorgestellten exakten Packverfahren erwiesen hat. Deshalb wird dieses Verfahren in den folgenden Analysen der Verfahren für das kombinierte Problem innerhalb des gestaffelten Packalgorithmus *Staged-Packer* eingesetzt.

6.2 Vergleich der Verfahren mittels Eingabeinstanzen aus der Literatur

Die im Folgenden vorgestellten experimentellen Analysen dienen dazu, das Verhalten und die Leistungsfähigkeit der implementierten Ansätze für das kombinierte Tourenplanungs- und Laderaumoptimierungsproblem zu untersuchen.

6.2.1 Die Eingabedaten

Die Experimente wurden unter Zuhilfenahme von 180 Eingabeinstanzen durchgeführt, die in [Gendreau, Iori, Laporte, und Martello 2008] zur Beurteilung des dort vorgestellten Tabu-Search-Verfahrens zur Lösung des Tourenplanungs- und zweidimensionalen Laderaumoptimierungsproblems verwendet wurden. Die Datensätze wurden von den Autoren aus 36 VRP-Instanzen generiert, von denen jede als Basis für 5 unterschiedliche Instanzen für das kombinierte Tourenplanungs- und Laderaumoptimierungsproblem dient. Jede der 36 VRP-Instanzen enthält einen Depotstandort und eine Anzahl n von Kundenstandorten, die sich zwischen 15 und 255 bewegt. Für jeden Standort i ist eine Position (x_i, y_i) im zweidimensionalen Koordinatensystem angegeben; die Entfernungsmatrix (d_{ij}) wird durch Berechnung euklidscher Distanzen zwischen allen Paaren von Standorten ermittelt und ist somit symmetrisch.

Jedem Depot ist eine Menge von v gleichartigen Fahrzeugen mit zweidimensionalen Ausmaßen (W, H) des Containers und einem maximal zulässigen Transportgewicht Q zugeordnet. Die Anzahl zur Verfügung stehender Fahrzeuge bewegt sich zwischen 3 und 51 pro Eingabeinstanz. Jeder Kunde i hat einen Bedarf, der durch das Gewicht q_i beschrieben wird und muss durch genau einen Fahrzeugbesuch beliefert werden.

		Vertikal		*Homogen*		*Horizontal*	
Klasse	m_i	*Breite*	*Höhe*	*Breite*	*Höhe*	*Breite*	*Höhe*
2	$[1,2]$	$\left[\frac{W}{10}, \frac{2W}{10}\right]$	$\left[\frac{4H}{10}, \frac{9H}{10}\right]$	$\left[\frac{2W}{10}, \frac{5W}{10}\right]$	$\left[\frac{2H}{10}, \frac{5H}{10}\right]$	$\left[\frac{4W}{10}, \frac{9W}{10}\right]$	$\left[\frac{H}{10}, \frac{2H}{10}\right]$
3	$[1,3]$	$\left[\frac{W}{10}, \frac{2W}{10}\right]$	$\left[\frac{3H}{10}, \frac{8H}{10}\right]$	$\left[\frac{2W}{10}, \frac{4W}{10}\right]$	$\left[\frac{2H}{10}, \frac{4H}{10}\right]$	$\left[\frac{3W}{10}, \frac{8W}{10}\right]$	$\left[\frac{H}{10}, \frac{2H}{10}\right]$
4	$[1,4]$	$\left[\frac{W}{10}, \frac{2W}{10}\right]$	$\left[\frac{2H}{10}, \frac{7H}{10}\right]$	$\left[\frac{W}{10}, \frac{4W}{10}\right]$	$\left[\frac{H}{10}, \frac{4H}{10}\right]$	$\left[\frac{2W}{10}, \frac{7W}{10}\right]$	$\left[\frac{H}{10}, \frac{2H}{10}\right]$
5	$[1,5]$	$\left[\frac{W}{10}, \frac{2W}{10}\right]$	$\left[\frac{H}{10}, \frac{6H}{10}\right]$	$\left[\frac{W}{10}, \frac{3W}{10}\right]$	$\left[\frac{H}{10}, \frac{3H}{10}\right]$	$\left[\frac{W}{10}, \frac{6W}{10}\right]$	$\left[\frac{H}{10}, \frac{2H}{10}\right]$

Tabelle 6.3: Itemklassen 2 – 5

Diese 36 VRP-Instanzen wurden zu 180 Instanzen für das Tourenplanungs- und zweidimensionale Laderaumoptimierungsproblem weiterentwickelt, indem mit jedem Kunden i eine Menge von m_i rechteckigen Items mit den Ausmaßen $(w_k, h_k),\ k = 1, \ldots, m_i$ assoziiert wurde. Die Anzahl der Items und ihre Dimensionen wurden gemäß der folgenden 5 Klassen gebildet:

- *Klasse 1*: Jedem Kunden $i = 1, \ldots, n$ wird ein Item k mit $w_k = h_k = 1$ zugeordnet.
- *Klassen 2 bis 5*: Die Itemanzahl m_i für einen Kunden i wird uniform zufällig aus einem gegebenen Intervall bestimmt (siehe Spalte 2 in Tabelle 6.3). Anschließend wird jedem der Items uniform zufällig eine der drei Grundformen *vertikal*, *homogen* und *horizontal* zugeordnet. Die Grundform entscheidet darüber, aus welchen Intervallen die Breite und Höhe des jeweiligen Items uniform zufällig gewählt werden. Die Spalten 3-8 in Tabelle 6.3 enthalten die Intervalle für die Bestimmung der Breite und der Höhe eines Items, nachdem diesem eine der drei Grundformen zugewiesen wurde.

Pro Instanz müssen zwischen 15 und 786 Packstücke zugestellt werden. Die Containerausmaße sind in allen Instanzen auf die Werte $W = 20$ und $H = 40$ festgelegt. Da alle Items in Klasse 1 die Ausmaße $(1, 1)$ haben, stellen die Instanzen dieser Klasse im Wesentlichen reine VRP-Probleme dar; bei der Berechnung dieser Instanzen mit der Packheuristik TP_{MOD} kam kein einziger Fall vor, in dem ein Fahrzeug nicht hätte gepackt werden können. Die Instanzen der Klassen 2-5 erfordern hingegen die explizite Berücksichtigung der Fahrzeugbepackung.

Die Datensätze geben weder Fahrzeiten noch Zeitfensterrestriktionen vor. Aus diesem Grund wird im Folgenden die Minimierung der Tourlänge als sekundäres Optimierkriterium verwendet.

Die Eingabeinstanzen können online von `http://www.or.deis.unibo.it/research.html` bezogen werden.

6.2.2 Experimentelles Setup

Die Berechnungen wurden auf einem AMD Athlon XP 2000+ (1667 MHz) mit 1GB Hauptspeicher unter Debian 4.0 (Etch) durchgeführt. Als Laufzeitumgebung wurde das Sun JDK

6 eingesetzt. Die Eingabeinstanzen wurden mittels der integrierten, der isolierten und der sequentiellen Methode (siehe Abschnitte 5.2, 5.3 und 5.4) unter Verwendung der Packalgorithmen TP_{MOD} (Abschnitt 4.4) sowie des gestaffelten Verfahrens *Staged-Packer* (Abschnitt 4.8) gelöst. Innerhalb von *Staged-Packer* (kurz: *SP*) wurden die untere Schranke L_4 (Abschnitt 4.7), die Heuristik TP_{MOD} sowie das exakte Verfahren MV_{MOD} (Abschnitt 4.5) eingesetzt. Letzteres hat sich in experimentellen Analysen als das schnellste der in dieser Arbeit implementierten exakten Verfahren für das Packproblem erwiesen (siehe Abschnitt 6.1).

Da das Verhalten des Tourenplanungsalgorithmus *I1* von den Parametern α, λ, μ und *Seed-Kriterium* abhängt (siehe Abschnitt 3.4) und für eine gegebene Eingabeinstanz nicht bekannt ist, welche Parameter die besten Ergebnisse liefern werden, muss für jede Instanz ein guter Satz von Parametern ermittelt werden. Dies wurde – abhängig vom eingesetzten Laderaumoptimierungsalgorithmus – auf eine der folgenden zwei Arten umgesetzt:

1. (*Die Brute-Force-Methode*) Wähle eine Menge $\pi = \{\pi_1, \ldots \pi_N\}$ von Parameterkombinationen, wobei $\pi_i = (\alpha_i, \lambda_i, \mu_i, \textit{Seed-Kriterium}_i),\ i = 1, \ldots, N$ eine vollständige Parametrisierung von *I1* darstellt. Führe die Berechnung der Instanz unter Einsatz jeder Parameterkombination aus π durch und weise das beste Ergebnis bezüglich der eingesetzten Zielfunktion aus.

2. (*Die Auswahl-Methode*) Wähle eine Menge $\pi = \{\pi_1, \ldots \pi_N\}$ von Parameterkombinationen. Führe für jede dieser Kombinationen die Tourenplanung ohne Berücksichtigung der Stauraumbepackung aus (für die Zuslässigkeit der Bepackung wird nur die untere Schranke L_0 geprüft), um den besten Parametersatz $\pi_{i^*} \in \pi$ für die gegebene Eingabeinstanz zu ermitteln. Führe nun das kombinierte Tourenplanungs- und Laderaumoptimierungsverfahren unter Verwendung der Parameter π_{i^*} aus und weise dieses Ergebnis aus.

Es wurde in beiden Methoden eine Menge von 24 Parametrisierungen zugrundegelegt, die sich als Kombination der folgenden Werte ergeben:

- $\alpha = 0, 1$
- $\lambda = 0, 1, 2$
- $\mu = 0, 1$
- *Seed-Kriterium* = maximale Entfernung zum Depot (1), früheste Deadline (2)

Die Brute-Force-Methode wurde unter Verwendung des schnellen Packverfahrens TP_{MOD} eingesetzt, da hier die Rechenzeiten ohnehin sehr kurz sind. Die Auswahl-Methode wurde im Zusammenhang mit dem wesentlich laufzeitintensiveren gestaffelten Verfahren *Staged-Packer* verwendet. Zudem wurde der Algorithmus MV_{MOD} innerhalb des gestaffelten Verfahrens mit einem Timeout T versehen, um ein Ausufern der Rechenzeit zu verhindern. Innerhalb der integrierten Methode wurde ein Timeout von einer Sekunde pro Aufruf des Packalgorithmus gesetzt, da hier sehr viele Aufrufe möglich sind. In der isolierten und der sequentiellen Methode wurde die maximale Laufzeit pro Aufruf (und damit pro Tour) auf 60 Sekuden begrenzt.

Um den Einfluss der LIFO-Reihenfolge der Bepackung auf die Rechenzeit und die Qualität der Lösungen festzustellen, wurden alle Benchmarks sowohl mit als auch ohne LIFO-Reihenfolge durchgeführt. Der Einfluss von Kundenzeitfenstern und der Drehbarkeit der Items wird in Abschnitt 6.3 untersucht. Für die hier beschriebenen Tests wurde die Orientierung der Items fixiert und für jeden Kunden i das unbeschränkte Zeitfenster $[e_i, l_i] = [-\infty, +\infty]$ festgelegt.

Dieses Setup erlaubt – zumindest in einem gewissen Rahmen – einen Vergleich der vorgestellten Ansätze mit dem Tabu-Search-Algorithmus aus [Gendreau, Iori, Laporte, und Martello 2008]. Die Autoren weisen experimentelle Ergebnisse mit und ohne Einhaltung der LIFO-Reihenfolge aus. Dabei werden die Packstücke stets in der vorgegebenen Orientierung belassen. Eine Vergleichbarkeit ist jedoch nicht in vollem Umfang gegeben, denn das Verfahren von Gendreau et al. ist bestrebt, die Gesamtlänge der Touren zu minimieren und nicht deren Anzahl, was nicht mit der in dieser Arbeit verfolgten Zielfunktion übereinstimmt. So gehen die Autoren von einer maximalen, zuvor heuristisch ermittelten Anzahl v von Fahrzeugen aus, während sich diese im Falle des Tourenplanungsalgorithmus *I1* erst zur Laufzeit ergibt.

Die wichtigsten Kennzahlen zu den eingesetzten Eingabeinstanzen können Tabelle 6.4 entnommen werden. Die ersten beiden Spalten der Tabelle geben die Nummer sowie den Namen der jeweiligen VRP-Instanz an. In den folgenden Auswertungen werden die Instanznummern zur Identifikation der Instanzen verwendet. Die dritte Spalte beinhaltet die Anzahl n von Kunden, die in der jeweiligen Instanz zu beliefern sind. Schließlich beinhalten die Spalten 4 bis 13 Informationen zu den fünf Itemklassen jeder Instanz. Hierbei kennzeichnet der Wert v die Anzahl zu verwendender Fahrzeuge und der Wert M die Anzahl Packstücke in der jeweiligen Klasse.

6.2.3 Ergebnisse

Die Ergebnisse der experimentellen Untersuchungen sind in den Tabellen 6.5, 6.6, 6.7 und 6.8 zusammengefasst. Die Tabellen beinhalten die Anzahl ausgewiesener Touren ($\#v_{2-5}$), die Tourlänge (z_{2-5}) sowie die Rechenzeit in Sekunden für die 36 VRP-Instanzen. Dabei handelt es sich stets um Durchschnittswerte, die für jede Instanz über die Itemklassen 2-5 gebildet wurden. Die Tabellen 6.5 und 6.6 beinhalten Benchmarkergebnisse mit TP_{MOD} als Packalgorithmus, wobei die Ergebnisse in der erstgenannten Tabelle ohne und in der zweitgenannten mit Berücksichtigung der LIFO-Reihenfolge ermittelt wurden. Analog enthalten die Tabellen 6.7 und 6.8 Ergebnisse unter Verwendung des gestaffelten Verfahrens – erstere ohne und letztere mit LIFO-Reihenfolge.

Zusätzlich wurden in den Spalten 2-4 in Tabelle 6.5 die Ergebnisse der Berechnung der 36 Eingabeinstanzen unter Verwendung der Itemklasse 1 eingetragen. Da alle Items dieser Klasse für jede Instanz problemlos in einem Fahrzeug untergebracht werden können, hat man es hier im Wesentlichen mit dem einfachen VRP ohne Stauraumbepackung zu tun. Deshalb sind diese Ergebnisse unabhängig von der verwendeten Methode (integriert, isoliert oder sequentiell), dem eingesetzten Packalgorithmus und davon, ob die LIFO-Reihenfolge der Bepackung gefordert ist.

Die Auswertung der Experimente deckte ein Laufzeitproblem eines der Verfahren auf. Es zeigte sich, dass die integrierte Methode unter Anwendung des gestaffelten Packverfahrens relativ lange Rechenzeiten benötigt, wenn große Eingabeinstanzen unter Einhaltung der

			Klasse 1		*Klasse 2*		*Klasse 3*		*Klasse 4*		*Klasse 5*	
Nr.	*Instanz*	n	M	v	M	v	M	v	M	v	M	v
1	E016-03m	15	15	3	24	3	31	3	37	4	45	4
2	E016-05m	15	15	5	25	5	31	5	40	5	48	5
3	E021-04m	20	20	4	29	5	46	5	44	5	49	5
4	E021-06m	20	20	6	32	6	43	6	50	6	62	6
5	E022-04g	21	21	4	31	4	37	4	41	4	57	5
6	E022-06m	21	21	6	33	6	40	6	57	6	56	6
7	E023-03g	22	22	3	32	5	41	5	51	5	55	6
8	E023-05s	22	22	5	29	5	42	5	48	5	52	6
9	E026-08m	25	25	8	40	8	61	8	63	8	91	8
10	E030-03g	29	29	3	43	6	49	6	72	7	86	7
11	E030-04s	29	29	4	43	6	62	7	74	7	91	7
12	E031-09h	30	30	9	50	9	56	9	82	9	101	9
13	E033-03n	32	32	3	44	7	56	7	78	7	102	8
14	E033-04g	32	32	4	47	7	57	7	65	7	87	8
15	E033-05s	32	32	5	48	6	59	6	84	8	114	8
16	E036-11h	35	35	11	56	11	74	11	93	11	114	11
17	E041-14h	40	40	14	60	14	73	14	96	14	127	14
18	E045-04f	44	44	4	66	9	87	10	112	10	122	10
19	E051-05e	50	50	5	82	11	103	11	134	12	157	12
20	E072-04f	71	71	4	104	14	151	15	178	16	226	16
21	E076-07s	75	75	7	114	14	164	17	168	17	202	17
22	E076-08s	75	75	8	112	15	154	16	198	17	236	17
23	E076-10e	75	75	10	112	14	155	16	179	16	225	16
24	E076-14s	75	75	14	124	17	152	17	195	17	215	17
25	E101-08e	100	100	8	157	21	212	21	254	22	311	22
26	E101-10c	100	100	10	147	19	198	20	247	20	310	20
27	E101-14s	100	100	14	152	19	211	22	245	22	320	22
28	E121-07c	120	120	7	183	23	242	25	299	25	384	25
29	E135-07f	134	134	7	197	24	262	26	342	28	422	28
30	E151-12b	150	150	12	225	29	298	30	366	30	433	30
31	E200-16b	199	199	16	307	38	402	40	513	42	602	42
32	E200-17b	199	199	17	299	38	404	39	497	39	589	39
33	E200-17c	199	199	17	301	37	407	41	499	41	577	41
34	E241-22k	240	240	22	370	46	490	49	604	50	720	50
35	E253-27k	252	252	27	367	45	507	50	634	50	762	50
36	E256-14k	255	255	14	387	47	511	51	606	51	786	51

Tabelle 6.4: Eingabeinstanzen

LIFO-Reihenfolge berechnet werden. Im Durchschnitt über alle Instanzen hat das Verfahren bei Einhaltung der LIFO-Reihenfolge etwa $9,3$ mal mehr Rechenzeit benötigt als ohne LIFO-Reihenfolge (vgl. die Laufzeiten *LZ* in den Zeilen 8 und 12 in Tabelle 6.10). Der Grund liegt in der stark erhöhten Anzahl von Aufrufen des Packalgorithmus, wenn mit LIFO-Reihenfolge optimiert wird (im Durschschnitt 1.934,4 Aufrufe gegenüber 261,8 ohne LIFO-Bepackung). Die Erklärung hierfür liegt in der Art der Implementierung des Tourenplanungsalgorithmus *I1*. Der Algorithmus sammelt zunächst alle möglichen Einfügekombinationen in der Liste I und sortiert diese anschließend nach den durch die Einfügungen verursachten Kosten. Dann iteriert das Verfahren über alle Kombinationen (u, p, c) in I und versucht, die Items des Kunden u zusammen mit den Items der sonstigen Kunden der Tour im Fahrzeug unterzubringen. Falls das misslingt, wird u als tabu markiert und später nicht noch einmal betrachtet – dies gilt jedoch nur, falls keine LIFO-Reihenfolge gefordert ist. Unter Berücksichtigung der LIFO-Reihenfolge ist es möglich, dass die Items dieses Kunden gepackt werden können falls sich die Reihenfolge der Tour ändert. Wenn n die Anzahl Kunden in der Menge U und l die Anzahl Kunden in der offenen Tour ist, wird im schlimmsten Fall das Packverfahren $\Theta(nl)$ mal aufgerufen. Es wurde eine naheliegende Lösung für dieses Problem gewählt, nämlich, die Anzahl Iterationen über die Liste I durch einen festgelegten Wert γ zu beschränken. Als günstig hat sich der Wert $\gamma = 10$ erwiesen, da er einerseits sehr große Laufzeitersparnisse bringt (90% Ersparnis bei Beachtung der LIFO-Reihenfolge, knapp 60% bei deren Nichtbeachtung) und andererseits eine nur geringe Verschlechterung des primären Zielfunktionskriteriums nach sich zieht. Es wurden mit LIFO-Bepackung nur $0,4\%$ mehr Touren generiert. Ohne LIFO-Bepackung waren es $4,6\%$ mehr Touren. Ein positiver Nebeneffekt der Beschränkung der Packversuche auf den Wert $\gamma = 10$ ist, dass dadurch die durchschnittliche Gesamtlänge der Touren teilweise drastisch gesenkt wird. So sinkt mit der Einschränkung $\gamma = 10$ die Tourenlänge mit LIFO-Bepackung um ganze $11,6\%$. Ohne LIFO-Bepackung sind es immerhin noch 2%. Die Reduktion der Tourlänge ist kein Zufall, denn die vorderen Einfügekombinationen in der Liste I sind bezüglich Tourlänge und/oder Tourdauer günstiger als die hinteren. Ohne Einschränkung der Anzahl Iterationen fügt der Algorithmus deshalb manchmal relativ ungünstige Kunden in die Tour ein, was jedoch aus der Sicht der Zielfunktion, primär die Anzahl Touren zu minimieren, völlig korrekt ist. Wäre die Tourlänge das primäre Kriterium der Zielfunktion, wäre es sicherlich lohnenswert, stets einen relativ kleinen Wert γ zu verwenden.

Die Ergebnistabellen 6.7 und 6.8 wurden um die Benchmarkergebnisse der integrierten Methode mit $\gamma = 10$ erweitert.

Für eine bessere Gesamtübersicht wurden die Ergebnisse der Tabellen 6.5-6.8 in den Tabellen 6.9 und 6.10 zusammengefasst und um weitere Kennzahlen ergänzt. Die Spalten dieser beiden Tabellen haben folgende Bedeutung:

- $\#v$: Anzahl Touren
- z: Tourlänge
- $\%w$: Gewichtsauslastung
- $\%a$: Nutzung der Frachtraum-Fläche
- *LZ*: Laufzeit in Sekunden

	VRP			*Integriert*			*Isoliert*			*Sequentiell*		
Instanz	v_1	z_1	LZ	$\#v_{2-5}$	z_{2-5}	LZ	$\#v_{2-5}$	z_{2-5}	LZ	$\#v_{2-5}$	z_{2-5}	LZ
1	3	286,4	0,7	3,8	304,5	1,2	4,2	337,2	0,4	4,0	315,4	0,4
2	5	363,6	0,3	5,0	374,1	0,4	5,5	379,3	0,3	5,5	363,4	0,3
3	4	362,4	0,4	4,8	418,1	0,8	5,5	453,7	0,5	5,0	454,2	0,6
4	6	469,1	0,4	6,0	494,5	0,5	6,8	507,5	0,4	6,2	540,3	0,5
5	4	424,0	0,4	4,0	420,4	0,9	4,8	522,4	0,5	4,8	436,7	0,5
6	6	514,6	0,3	6,0	623,6	0,6	6,5	547,0	0,4	6,5	560,9	0,5
7	3	600,7	1,4	4,5	830,1	1,8	5,0	886,0	0,6	4,8	901,2	0,6
8	3	600,7	1,5	4,0	818,0	2,9	5,0	861,9	0,6	4,8	810,4	0,6
9	8	711,0	0,4	8,0	713,8	0,7	8,8	766,1	0,5	8,8	746,9	0,5
10	3	566,7	1,3	5,5	810,5	3,6	6,5	1.057,2	0,7	6,5	951,4	0,9
11	3	566,7	1,3	6,2	904,6	3,6	7,8	1.024,2	0,7	7,5	928,1	0,6
12	9	636,3	0,5	9,5	644,8	0,9	10,0	713,4	0,5	9,8	714,1	0,9
13	3	2.184,8	2,6	6,2	3.007,7	7,7	7,0	3.512,9	0,8	7,0	3.405,4	0,9
14	4	887,2	1,0	5,5	1.157,4	5,5	6,8	1.343,6	0,8	6,0	1.243,6	0,7
15	4	887,2	0,9	6,5	1.335,0	3,7	8,0	1.523,1	0,7	7,5	1.434,5	0,7
16	11	767,4	0,4	11,0	886,6	0,8	11,8	931,8	0,5	11,8	900,1	0,5
17	15	912,8	0,5	14,8	1.015,3	0,6	15,0	947,0	0,5	15,0	935,0	0,5
18	4	898,0	3,0	8,2	1.456,7	7,2	10,2	1.400,7	0,9	10,0	1.422,0	0,9
19	5	592,2	1,6	10,2	947,2	8,3	12,8	1.072,7	1,0	12,5	1.042,7	1,0
20	4	320,1	20,4	14,8	640,9	15,7	18,2	836,1	1,4	17,2	729,5	1,4
21	7	793,3	3,0	14,5	1.247,7	18,0	17,8	1.673,2	1,5	17,2	1.530,4	1,4
22	8	882,8	2,1	15,2	1.306,5	18,6	19,0	1.476,1	1,5	18,0	1.519,3	1,4
23	10	962,7	1,5	15,0	1.288,9	12,5	18,5	1.654,2	1,4	18,0	1.564,3	1,3
24	14	1.145,5	1,0	16,5	1.558,0	5,9	19,2	1.887,1	1,2	19,2	1.481,7	1,2
25	8	990,4	5,4	20,2	1.755,8	30,4	24,5	2.125,5	2,1	24,0	2.112,2	2,0
26	10	890,4	3,7	19,5	1.776,9	33,9	23,8	2.199,2	2,2	22,8	2.267,7	2,0
27	14	1.270,1	2,0	19,8	2.109,1	19,9	24,2	2.263,9	2,0	24,0	2.021,9	1,9
28	7	1.103,7	10,5	23,0	3.261,3	46,5	28,8	3.898,2	2,6	27,8	3.898,5	2,5
29	7	1.395,6	44,5	25,2	2.710,4	52,2	31,2	3.286,8	3,0	30,5	3.234,1	2,8
30	12	1.226,5	8,9	28,2	2.377,6	65,9	34,8	3.125,7	3,4	34,2	3.264,4	3,3
31	16	1.522,9	12,1	38,0	3.004,5	110,3	46,8	4.713,4	5,7	46,2	3.030,6	5,1
32	16	1.522,9	11,6	36,8	3.800,9	112,6	45,2	3.860,7	6,0	44,8	3.882,9	5,3
33	16	1.592,6	12,7	37,8	3.452,6	101,4	47,8	3.548,9	4,7	46,8	3.670,7	5,1
34	22	788,1	15,3	46,5	1.748,6	130,7	57,5	2.214,1	7,9	57,2	1.716,0	6,8
35	26	1.000,4	15,9	46,8	2.513,4	169,3	59,8	2.150,4	7,0	58,2	2.350,7	7,6
36	14	712,6	64,9	46,8	2.341,6	175,2	59,5	2.448,7	7,4	58,0	2.339,8	7,9

Tabelle 6.5: *I1 + 2-Opt +* TP_{MOD}, freie Stauraumbepackung

- $\%^{LZ}_{2\text{-}Opt}$: Anteil der *2-Opt*-Nachoptimierung an der Gesamtlaufzeit
- $\%^{Dist}_{2\text{-}Opt}$: Reduktion der Tourlänge, die durch *2-Opt* erzielt wurde
- $\%^{LZ}_{Packer}$: Anteil des eingesetzten Packalgorithmus an der Gesamtlaufzeit
- $\#_{Packer}$: Anzahl Aufrufe des Packalgorithmus
- $\%^{Erfolg}_{L_4}$: Anteil erfolgreicher Aufrufe der unteren Schranke L_4
- $\%^{Erfolg}_{TP}$: Anteil erfolgreicher Aufrufe der Heuristik TP_{MOD}

Alle Zahlen in diesen beiden Tabellen sind Mittelwerte über die 144 Eingabeinstanzen der Klassen 2-5 (dies gilt für die VRTWLP-Algorithmen, Zeilen 2-15 sowie 17 und 18) bzw. über die 36 Instanzen der Klasse 1 (dies gilt für die Zeilen 1 und 16). Um zu zeigen, dass die implementierten Algorithmen vernünftige Ergebnisse liefern, enthalten die letzten drei Zeilen beider Tabellen Informationen zum Tabu-Search-Verfahren aus [Gendreau, Iori, Laporte, und Martello 2008] sofern diese aus dem Artikel entnommen werden konnten. Die Werte $\%a$, $\%t_{Packer}$ und $\#_{Packer}$ sind bezogen auf die Itemklasse 1 (reines VRP) offensichtlich uninteressant, da die Fahrzeugauslastung ohnehin allein von den Gewichten der Güter abhängt; der Vollständigkeit halber wurden sie trotzdem in die Tabellen aufgenommen. Die Werte $\%^{Erfolg}_{L_4}$ und $\%^{Erfolg}_{TP}$ beziehen sich ausschließlich auf das gestaffelte Verfahren, wobei die untere Schranke L_4 nur in Kombination mit der integrierten Methode eingesetzt wird.

	Integriert			*Isoliert*			*Sequentiell*		
Instanz	$\#v_{2-5}$	z_{2-5}	LZ	$\#v_{2-5}$	z_{2-5}	LZ	$\#v_{2-5}$	z_{2-5}	LZ
1	3,8	335,6	1,6	4,2	334,0	1,0	4,0	319,3	0,4
2	5,0	370,9	0,9	5,2	387,8	0,9	5,2	388,9	0,4
3	4,8	469,3	1,5	5,2	505,8	1,2	5,2	463,9	0,5
4	6,0	495,1	1,2	6,8	507,3	1,0	6,8	501,4	0,4
5	4,0	521,5	1,8	4,8	515,8	1,1	4,8	456,1	0,5
6	6,0	563,1	1,3	6,5	661,3	1,1	6,5	598,0	0,4
7	4,8	854,6	4,8	5,2	919,7	1,3	4,8	895,4	0,6
8	4,2	893,8	6,7	5,0	887,5	1,3	5,0	858,2	0,6
9	8,2	708,7	1,5	9,0	771,8	1,2	8,8	759,1	0,5
10	5,8	867,0	6,9	7,0	959,1	1,5	6,2	878,5	0,7
11	6,5	1.094,1	7,2	8,0	1.100,9	1,5	7,5	997,6	0,7
12	9,5	669,2	1,3	10,5	670,2	1,2	10,2	647,6	0,5
13	6,2	3.348,7	10,2	7,5	3.697,6	1,6	7,0	3.645,4	0,8
14	5,5	1.268,8	9,7	7,0	1.424,7	1,7	6,2	1.281,0	0,8
15	6,5	1.440,6	8,2	8,5	1.609,2	1,7	7,5	1.499,8	0,7
16	11,2	788,3	1,3	12,0	927,2	1,2	11,8	866,1	0,5
17	14,2	1.080,9	1,2	15,0	943,8	1,1	15,0	930,0	0,5
18	8,8	1.318,5	15,3	10,2	1.464,5	2,0	10,2	1.469,2	1,0
19	10,8	1.049,1	22,7	13,5	1.137,7	2,2	12,8	1.034,2	1,0
20	14,8	724,7	33,1	18,5	887,2	2,8	17,8	772,1	1,5
21	14,5	1.624,0	42,9	18,5	1.477,6	2,8	17,5	1.451,9	1,5
22	15,2	1.842,5	43,5	19,2	1.600,0	2,9	19,0	1.415,0	1,5
23	15,2	1.700,0	24,3	19,5	1.577,0	2,7	18,8	1.464,9	1,4
24	16,8	1.590,1	8,9	20,0	1.859,5	2,8	20,0	1.471,8	1,3
25	20,5	1.932,4	91,3	25,8	1.982,2	4,4	25,0	1.965,3	2,1
26	19,2	2.423,6	71,4	24,2	2.280,4	3,6	23,8	1.944,5	2,1
27	20,2	2.084,1	43,8	25,2	1.985,7	3,5	24,5	1.879,9	2,0
28	23,2	3.951,1	121,5	29,5	4.176,6	4,2	28,5	4.016,4	2,6
29	25,8	3.427,6	120,6	32,0	3.563,8	4,6	31,5	3.355,0	3,1
30	28,8	2.667,6	164,6	36,8	2.758,2	5,0	36,0	2.908,3	3,6
31	38,2	3.851,5	267,5	49,0	3.539,3	6,6	48,2	3.233,9	5,5
32	37,0	4.208,0	271,5	47,5	3.599,7	6,8	46,5	3.324,1	5,7
33	38,2	3.628,0	241,7	49,2	3.622,4	6,5	48,0	3.260,1	5,3
34	46,8	1.824,0	300,7	61,0	2.014,2	8,0	59,5	1.749,5	7,2
35	47,8	2.185,7	366,7	62,0	2.474,0	8,9	61,0	2.069,6	8,1
36	47,8	2.206,5	435,8	61,8	2.987,8	9,0	61,2	2.334,4	8,4

Tabelle 6.6: *I1 + 2-Opt +* TP_{MOD}, LIFO-Reihenfolge der Bepackung

	Integriert			*Integriert* ($\gamma = 10$)			*Isoliert*			*Sequentiell*		
Inst.	$\#v_{2-5}$	z_{2-5}	LZ	$\#v_{2-5}$	z_{2-5}	LZ	$\#v_{2-5}$	z_{2-5}	LZ	$\#v_{2-5}$	z_{2-5}	LZ
1	4,0	313,5	1,8	4,0	313,5	1,8	4,0	352,5	93,3	4,2	346,2	108,5
2	5,2	370,6	0,6	5,2	370,6	0,6	5,2	369,8	9,5	5,2	369,1	9,6
3	4,8	412,3	2,2	4,8	412,3	2,2	5,0	435,2	45,8	5,0	434,1	50,3
4	6,2	470,7	1,5	6,2	470,7	1,5	6,0	470,9	2,9	6,0	470,9	2,9
5	4,2	413,2	2,2	4,2	413,2	2,3	5,0	467,4	62,4	4,5	451,7	95,7
6	6,2	530,3	1,4	6,2	530,3	1,4	6,8	570,6	44,3	6,8	548,0	43,8
7	4,0	850,2	6,7	4,0	850,2	6,7	5,0	870,6	103,4	5,0	854,6	152,1
8	4,8	827,8	12,4	4,8	827,8	11,9	4,8	896,4	107,1	4,8	857,1	151,4
9	8,5	728,1	1,7	8,5	728,1	1,7	8,8	756,6	65,2	8,8	748,3	75,4
10	5,2	830,0	18,9	5,2	830,0	18,9	6,8	969,3	213,5	6,2	934,7	250,9
11	6,2	889,7	13,0	6,2	908,2	14,0	7,0	966,1	225,9	6,8	978,2	226,0
12	9,5	632,8	1,5	9,5	632,8	1,5	9,8	670,8	36,9	9,8	657,1	52,1
13	6,2	2.803,6	20,4	6,2	2.803,6	19,4	7,2	3.488,7	242,7	6,8	3.307,0	255,9
14	5,2	1.115,4	15,0	5,2	1.115,4	14,0	6,2	1.258,6	230,5	6,0	1.227,0	256,3
15	6,5	1.264,5	15,4	6,5	1.264,5	15,4	7,5	1.464,1	295,1	7,2	1.381,9	284,6
16	11,5	792,4	1,0	11,5	792,4	1,0	11,8	820,7	30,5	11,8	800,8	30,5
17	15,0	931,2	0,4	15,0	931,2	0,4	15,2	938,6	0,4	15,2	926,0	0,4
18	8,2	1.133,1	36,4	8,2	1.108,1	35,6	10,0	1.404,6	355,6	9,5	1.372,7	428,1
19	9,8	854,1	37,4	9,8	854,3	37,9	12,2	1.001,6	383,1	11,2	944,0	425,7
20	14,0	608,9	69,5	14,0	602,5	60,6	16,5	711,3	612,5	16,8	707,4	691,8
21	14,0	1.192,4	103,6	14,0	1.155,7	77,0	17,2	1.412,4	660,9	16,8	1.303,0	777,5
22	14,8	1.139,4	81,0	14,8	1.156,0	67,1	18,0	1.400,2	627,6	17,5	1.363,3	714,8
23	14,5	1.200,9	79,5	14,5	1.182,6	67,1	18,0	1.400,1	584,4	17,0	1.364,8	636,2
24	16,2	1.377,6	30,8	16,2	1.377,6	30,2	18,2	1.536,4	432,7	18,0	1.499,4	479,0
25	19,5	1.566,0	120,0	19,8	1.511,6	97,9	23,0	1.812,9	848,7	23,0	1.770,6	984,2
26	18,5	1.544,6	156,9	18,5	1.516,3	117,4	22,5	2.016,7	882,3	22,0	1.882,4	919,0
27	19,2	1.557,6	92,3	19,2	1.559,3	81,1	22,8	1.842,0	905,0	23,0	1.849,3	909,8
28	22,2	2.840,8	233,8	22,2	2.771,9	153,8	27,2	3.511,8	1.154,5	26,8	3.484,3	1.209,6
29	24,2	2.532,6	243,5	24,2	2.449,4	173,1	29,2	3.194,1	1.234,9	29,0	2.999,0	1.384,2
30	27,0	2.050,0	329,2	27,5	1.937,9	197,1	33,0	2.410,6	1.323,3	33,2	2.391,9	1.433,3
31	36,5	2.648,1	516,0	36,8	2.530,4	268,5	44,2	3.147,9	1.922,2	44,2	3.080,6	2.098,1
32	35,8	2.556,2	502,9	36,0	2.440,0	269,9	44,0	3.080,9	1.893,1	43,0	2.960,3	1.889,0
33	36,2	2.540,4	472,7	36,8	2.476,7	279,1	44,8	3.071,5	1.868,1	44,2	3.014,8	2.000,5
34	44,5	1.345,8	641,3	44,8	1.289,8	340,1	53,8	1.897,0	2.239,9	54,0	1.538,7	2.447,9
35	45,8	1.816,5	746,3	45,8	1.672,0	355,6	55,8	2.096,6	2.421,9	56,2	2.108,0	2.564,9
36	45,5	1.773,0	764,8	46,0	1.756,4	388,0	56,2	2.182,0	2.443,2	55,5	2.130,0	2.627,0

Tabelle 6.7: *I1 + 2-Opt + Staged-Packer*, freie Stauraumbepackung

	Integriert			*Integriert* ($\gamma = 10$)			*Isoliert*			*Sequentiell*		
Inst.	$\#v_{2-5}$	z_{2-5}	LZ	$\#v_{2-5}$	z_{2-5}	LZ	$\#v_{2-5}$	z_{2-5}	LZ	$\#v_{2-5}$	z_{2-5}	LZ
1	4,0	335,6	8,5	3,8	335,8	6,3	4,0	351,5	75,7	4,0	336,8	80,1
2	5,2	373,9	1,0	5,2	373,9	0,8	5,5	382,2	1,0	5,5	389,1	1,0
3	4,8	448,7	19,7	4,8	431,9	8,5	5,8	461,2	92,0	5,5	471,7	65,0
4	6,2	464,0	3,2	6,2	464,0	3,2	6,0	470,9	1,5	6,0	470,9	1,5
5	4,2	415,0	7,5	4,2	415,0	7,4	5,0	475,6	40,7	4,5	440,1	52,2
6	6,5	569,6	14,2	6,5	548,7	5,1	6,8	547,1	32,9	6,5	531,6	32,6
7	4,8	973,4	70,2	4,8	877,1	24,5	5,0	939,4	77,7	5,0	915,3	142,3
8	4,8	931,6	70,9	4,8	849,6	28,4	5,0	871,5	105,6	5,0	907,3	126,4
9	8,5	744,6	7,2	8,5	744,6	7,3	9,0	779,5	77,9	9,0	749,3	60,9
10	5,5	897,5	173,9	5,8	836,7	47,9	7,0	996,2	208,3	6,2	889,7	236,2
11	6,2	1.003,0	100,9	6,8	892,8	46,0	7,8	1.086,0	214,8	7,2	1.072,0	213,5
12	9,5	647,1	7,3	9,5	647,1	5,5	9,8	686,2	32,8	10,0	667,2	63,0
13	6,2	3.399,6	131,7	6,5	3.115,6	50,7	8,0	3.660,3	224,0	7,0	3.483,8	210,9
14	5,5	1.331,3	184,2	5,8	1.255,3	44,0	6,8	1.357,3	216,4	6,2	1.327,5	204,9
15	6,8	1.381,8	124,9	6,5	1.310,2	58,7	8,5	1.645,9	325,1	7,2	1.489,9	270,6
16	11,8	789,3	3,4	11,8	789,3	3,3	11,8	830,3	30,5	11,8	809,8	38,2
17	15,0	933,5	0,7	15,0	933,5	0,7	15,5	945,2	15,5	15,2	934,5	15,4
18	8,2	1.410,5	317,2	9,0	1.176,6	75,2	10,5	1.425,1	317,2	9,8	1.368,2	337,4
19	10,8	953,7	329,4	11,0	877,5	82,0	13,0	1.029,0	352,3	12,0	1.016,5	397,3
20	14,2	725,1	538,5	15,2	640,6	133,8	17,2	743,2	585,9	17,8	759,8	684,4
21	14,5	1.502,7	751,0	15,0	1.210,6	127,8	17,5	1.427,5	600,8	17,5	1.416,3	644,1
22	15,2	1.384,8	845,7	16,0	1.251,7	136,1	18,0	1.453,2	570,8	18,0	1.397,3	619,5
23	15,0	1.355,2	575,1	15,5	1.229,2	131,6	18,2	1.468,3	589,3	18,2	1.423,1	640,7
24	16,8	1.462,3	169,4	17,0	1.412,5	78,1	19,2	1.593,3	433,8	18,8	1.488,3	492,3
25	20,5	1.902,2	1.298,6	21,2	1.727,0	183,2	24,5	1.947,9	831,5	23,5	1.790,1	858,1
26	18,8	1.946,6	1.229,4	20,2	1.653,0	183,8	23,0	1.988,0	796,9	22,2	1.914,1	857,0
27	19,5	1.799,9	736,6	20,8	1.602,4	176,9	24,0	1.972,1	820,0	23,5	1.804,0	846,2
28	22,8	3.626,9	2.099,8	24,8	3.032,9	240,6	29,2	3.719,9	1.026,3	27,8	3.545,3	1.072,5
29	25,2	3.165,3	2.473,3	26,5	2.588,0	266,3	31,2	3.255,1	1.212,0	30,2	3.102,4	1.265,8
30	28,2	2.296,4	2.759,9	29,5	2.017,5	268,3	34,8	2.546,5	1.247,6	33,5	2.396,4	1.217,3
31	38,0	3.121,7	4.598,4	39,8	2.611,9	398,1	46,2	3.257,9	1.713,2	45,2	3.084,0	1.844,0
32	36,8	3.258,7	4.803,4	38,5	2.610,1	379,6	44,8	3.171,0	1.684,3	45,2	3.075,6	1.857,4
33	37,8	3.041,7	5.043,6	39,5	2.551,3	382,3	46,2	3.190,3	1.651,4	45,8	3.057,1	1.787,9
34	46,0	1.638,6	6.598,6	48,5	1.380,5	476,3	56,8	1.953,9	2.092,9	55,5	1.576,1	2.174,1
35	46,2	2.121,1	6.511,0	50,0	1.752,9	472,9	56,5	2.123,4	2.093,8	56,2	2.082,4	2.138,9
36	46,5	1.984,7	7.701,2	50,0	1.874,3	486,4	56,8	2.222,0	2.121,2	57,0	2.183,3	2.213,0

Tabelle 6.8: *I1 + 2-Opt + Staged-Packer*, LIFO-Reihenfolge der Bepackung

	Packer	*Methode*	*LIFO*	$\#v$	z	$\%w$	$\%a$
1	VRP (I1)		–	8,72	870,9	93,5%	1,1%
2	TP_{MOD}	integriert	nein	16,51	1.501,6	59,6%	76,3%
3	TP_{MOD}	isoliert	nein	20,12	1.726,4	51,3%	63,9%
4	TP_{MOD}	sequentiell	nein	19,67	1.631,1	52,7%	65,7%
5	TP_{MOD}	integriert	ja	16,72	1.666,9	59,0%	75,2%
6	TP_{MOD}	isoliert	ja	20,86	1.717,0	50,2%	62,0%
7	TP_{MOD}	sequentiell	ja	20,34	1.586,3	51,6%	64,1%
8	*SP*	integriert	nein	16,11	1.290,4	59,5%	77,3%
9	*SP* ($\gamma = 10$)	integriert	nein	16,18	1.265,1	59,4%	77,1%
10	*SP*	isoliert	nein	19,13	1.524,9	53,1%	66,7%
11	*SP*	sequentiell	nein	18,92	1.473,8	54,0%	67,9%
12	*SP*	integriert	ja	16,57	1.509,4	58,1%	75,1%
13	*SP* ($\gamma = 10$)	integriert	ja	17,34	1.333,9	56,9%	72,6%
14	*SP*	isoliert	ja	19,85	1.582,6	51,2%	63,9%
15	*SP*	sequentiell	ja	19,44	1.510,2	52,7%	66,0%
16	VRP (Gendreau et al.)		–	8,78	792,3	–	–
17	Gendreau et al.		nein	16,46	1.216,1	–	76,7%
18	Gendreau et al.		ja	16,65	1.266,0	–	75,5%

Tabelle 6.9: Lösungsqualität der Verfahren im Vergleich

Man betrachte zunächst die Ergebnisse in Tabelle 6.9. Bezüglich des primären Optimierkriteriums – der Anzahl von Touren – ist die integrierte Methode den anderen beiden Methoden klar überlegen. Diese Methode generiert durchschnittlich zwischen $15,3\%$ und $18,8\%$ weniger Touren als die isolierte und die sequentielle Methode – der genaue Wert hängt vom eingesetzten Packalgorithmus ab und davon, ob die LIFO-Reihenfolge der Bepackung gefordert ist. Die beiden letztgenannten Methoden liefern im Durchschnitt ähnliche Tourenanzahlen, wobei die sequentielle Methode stets leicht im Vorteil ist. Die Auswirkungen des Wertes $\gamma = 10$ auf die Tourenzahl wurden bereits genannt.

Die Bedingung der LIFO-Bepackung erhöht die durchschnittliche Tourenanzahl. Im Falle der integrierten Methode bewirkt sie einen Zuwachs um $1,3\%$ (TP_{MOD}) bzw. $2,9\%$ (gestaffeltes Packverfahren), im Falle der isolierten Methode um $3,7\%$ (beide Packalgorithmen) und im Falle der sequentiellen Methode um $3,4\%$ bzw. $2,7\%$. Einzig der Zuwachs in der Variante mit $\gamma = 10$ fällt größer aus: Er beträgt $7,2\%$. Das Diagramm in Abbildung 6.1 stellt den durch die LIFO-Reihenfolge verursachten mittleren Zuwachs an Touren aufgeschlüsselt nach verwendetem Algorithmus dar.

Verglichen mit den Ergebnissen von Gendreau et al. generiert die integrierte Methode eine sehr ähnliche durchschnittliche Anzahl an Touren. Unter Verwendung von TP_{MOD} werden $0,3\%$ (ohne LIFO-Bedingung) bzw. $0,4\%$ (mit LIFO-Bedingung) mehr Touren generiert, unter Einsatz des gestaffelten Verfahrens werden sogar $2,1\%$ (ohne LIFO) bzw. $0,5\%$ (mit LIFO) weniger Touren generiert als es der Tabu-Search-Algorithmus von Gendreau et al. tut. Beim Vergleich der hier erzielten Benchmarkergebnisse mit den Ergebnissen von Gendreau et al. sollte stets beachtet werden, dass die Autoren des Tabu-Search-Verfahrens eine andere Zielfunktion verfolgten als die in dieser Arbeit vorgestellten Verfahren. Bei ihnen steht die Gesamttourlänge und nicht die Tourenanzahl im Vordergrund. Es ist daher nicht überraschend, dass das Tabu-Search-Verfahren im Allgemeinen Lösungen mit einer geringeren Tourlänge ausweist als es die hier vorgestellten Verfahren tun. Andererseits wird der Tabu-Search-Ansatz in einigen Fällen von den hier vorgestellten Verfahren hinsichtlich der Tourenanzahl geschlagen. Die geringsten Tourlängen der hier vorgestellten Verfahren werden unter Einsatz des Parameters $\gamma = 10$ erzielt. Ein Vergleich mit den Ergebnissen des Verfahrens von Gendreau et al. zeigt, dass in diesem Fall durchschnittlich nur um $4,0\%$

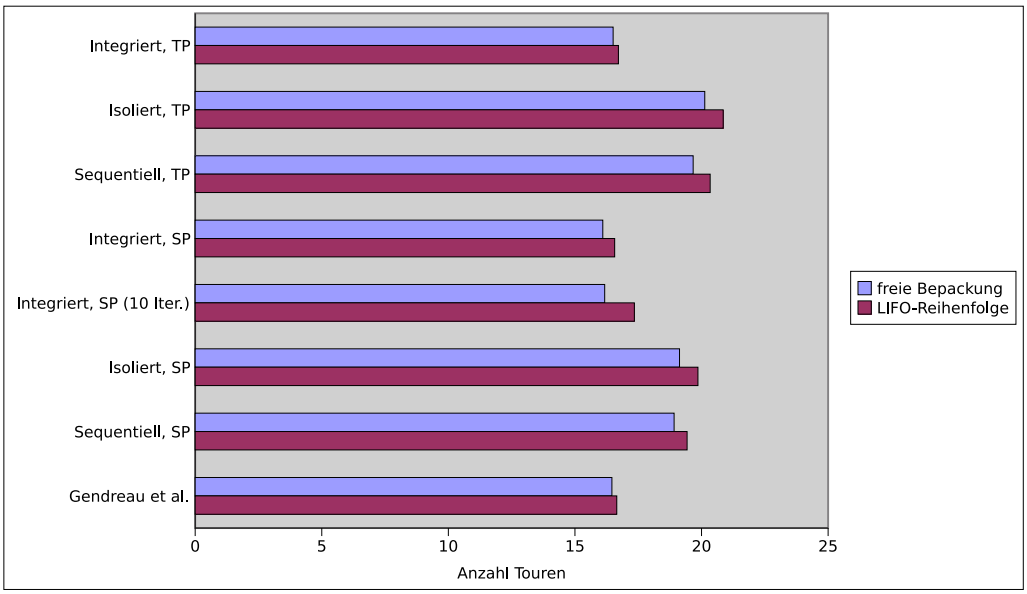


Abbildung 6.1: Anzahl Touren mit und ohne LIFO-Bepackung

(ohne LIFO-Bedingung) bzw. um $5,4\%$ (mit LIFO-Bedingung) längere Touren ausgewiesen wurden.

In Tabelle 6.10 werden die Laufzeiten und das technische Verhalten der Algorithmen beleuchtet. Man betrachte zunächst den oberen Abschnitt der Tabelle (Zeilen 1-7), in dem die Ergebnisse unter Anwendung des Packalgorithmus TP_{MOD} zusammengefasst sind. Es zeigt sich, dass die isolierte und die sequentielle Methode wesentlich kürzere durchschnittliche Gesamtlaufzeiten LZ aufweisen als die integrierte Methode, wenn weder die Laufzeit des Packalgorithmus noch die Anzahl seiner Aufrufe künstlich manipuliert werden (diese Voraussetzung ist im Falle von TP_{MOD} gegeben, da hier kein Timeout eingesetzt und der Wert $\gamma = \infty$ verwendet wurde). Die Erklärung findet sich in der stark erhöhten Anzahl $\#_{Packer}$ von Aufrufen des Packalgorithmus. So waren es über 50.000 Aufrufe im integrierten Modus gegenüber nicht einmal 600 Aufrufen in den anderen beiden Modi. Dies spiegelt sich in der Laufzeitverteilung wider, in der der Anteil $\%t_{Packer}$ des Packalgorithmus an der Gesamtlaufzeit im integrierten Modus bei $94,0\%$ (ohne LIFO) bzw. $97,1\%$ (mit LIFO) liegt. In den anderen beiden Modi beträgt der Anteil weniger als 30%. Die Gesamtlaufzeit LZ bei der Berechnung der Instanzen der Itemklasse 1 liegt sogar höher als die der isolierten und der sequentiellen Methode unter Anwendung von TP_{MOD} für die Itemklassen 2 bis 5. Dies liegt daran, dass in den Problemen der Klasse 1 die durchschnittliche Anzahl Kunden in einer Tour wesentlich höher ist als beim kombinierten Tourenplanungs- und Laderaumoptimierungsproblem. Hieraus ergeben sich erhöhte Laufzeiten für die Tourenplanung (die mögliche Anzahl an Einfügekombinationen in eine Tour nimmt zu) sowie für die Nachoptimierung *2-Opt*. Wenn das VRTWLP gelöst wird, steigt der Anteil $\%LZ_{2\text{-}Opt}$ von *2-Opt* an der Gesamtlaufzeit von maximal $13,4\%$ auf über 40% im Falle des VRP. Entsprechend steigt auch die Erfolgsquote $\%Dist_{2\text{-}Opt}$ der Nachoptimierung von maximal $8,6\%$ Tourlängenreduktion im Falle des VRTWLP auf $9,9\%$ im Falle des VRP. Ein Einbruch der Effektivität von *2-Opt* ist im integrierten Modus mit LIFO-Bepackung zu beobachten. Die Erklärung ist, dass im integrierten Modus nur bereits fertig gepackte

	Packer	*Methode*	*LIFO*	*LZ*	$\%^{LZ}_{2\text{-}Opt}$	$\%^{Dist}_{2\text{-}Opt}$	$\%^{LZ}_{Packer}$	$\#_{Packer}$	$\%^{Erfolg}_{L_4}$	$\%^{Erfolg}_{TP}$
1	VRP (I1)		–	5,5	40,2%	9,9%	1,6%	82,0	–	–
2	TP_{MOD}	integriert	nein	32,5	2,8%	8,0%	94,0%	50.752,7	–	–
3	TP_{MOD}	isoliert	nein	2,0	13,4%	8,2%	28,8%	512,5	–	–
4	TP_{MOD}	sequentiell	nein	2,0	13,0%	8,6%	28,1%	497,3	–	–
5	TP_{MOD}	integriert	ja	76,5	0,6%	1,4%	97,1%	67.761,5	–	–
6	TP_{MOD}	isoliert	ja	3,1	11,6%	8,1%	29,8%	533,6	–	–
7	TP_{MOD}	sequentiell	ja	2,1	12,5%	7,9%	28,9%	516,6	–	–
8	*SP*	integriert	nein	149,3	0,2%	6,1%	99,1%	261,8	1,5%	24,1%
9	*SP* ($\gamma = 10$)	integriert	nein	89,2	0,3%	5,9%	98,5%	188,1	1,4%	33,9%
10	*SP*	isoliert	nein	683,4	0,0%	6,1%	99,8%	19,1	–	17,2%
11	*SP*	sequentiell	nein	740,8	0,0%	5,9%	99,8%	18,9	–	12,5%
12	*SP*	integriert	ja	1.397,5	0,0%	2,3%	99,9%	1.934,4	1,7%	3,0%
13	*SP* ($\gamma = 10$)	integriert	ja	139,6	0,2%	3,1%	99,1%	271,4	1,0%	23,3%
14	*SP*	isoliert	ja	625,4	0,0%	6,1%	99,8%	19,9	–	16,6%
15	*SP*	sequentiell	ja	660,1	0,0%	6,1%	99,8%	19,4	–	12,0%
16	VRP (Gendreau et al.)		–	1.757,4	–	–	–	–	–	–
17	Gendreau et al.		nein	1.837,4	–	–	–	–	–	–
18	Gendreau et al.		ja	1.831,9	–	–	–	–	–	–

Tabelle 6.10: Leistung der Verfahren im Vergleich

Touren nachoptimiert werden. Selbst wenn eine Verkürzung der Tour durch Umstellung der Reihenfolge ihrer Kunden gefunden wird, muss anschließend noch die Bepackung unter Berücksichtigung der LIFO-Reihenfolge neu berechnet werden. Dies schlägt in vielen Fällen fehl, weshalb die potentielle Tourlängenreduktion nicht genutzt werden kann.

Nun betrachte man die Zeilen 8-15 in Tabelle 6.10, in denen Kennzahlen unter Verwendung des gestaffelten Packverfahrens *Staged-Packer* aufgeführt sind. Auffällig sind die stark reduzierten durchschnittlichen Anzahlen $\#_{Packer}$ an Aufrufen des Packalgorithmus pro Eingabeinstanz gegenüber den Benchmarks mit TP_{MOD}. Die Erklärung hierfür ist, dass die Benchmarks mit dem gestaffelten Verfahren unter Anwendung der Auswahl-Methode durchgeführt wurden, während für die Berechnungen mit TP_{MOD} die Brute-Force-Methode zum Einsatz kam. Die Brute-Force-Methode berechnet das gesamte VRTWLP 24 Mal pro Instanz (ein Mal pro *I1*-Parametersatz), während die Auswahl-Methode dies nur ein Mal tut. Wie bereits erwähnt, kann durch Setzen der Iterationsbeschränkung $\gamma = 10$ eine deutliche Reduktion der Aufrufe des Packalgorithmus in der integrierten Methode erzielt werden, aus welcher eine entsprechende Reduktion der Gesamtlaufzeit LZ resultiert. Im Allgemeinen sind die Gesamtlaufzeiten unter Einsatz des gestaffelten Verfahrens jedoch deutlich höher als unter Einsatz von TP_{MOD}. Die Tatsache, dass die Gesamtlaufzeiten der isolierten und der sequentiellen Methode hier teilweise höher sind als die der integrierten Methode ergibt sich aus den unterschiedlichen für den Packalgorithmus vorgegebenen maximalen Laufzeiten pro Aufruf. So wurde der gestaffelte Packalgorithmus in der isolierten und der sequentiellen Methode mit einem Timeout von 60 Sekunden ausgeführt, während das Timeout in der integrierten Methode auf eine Sekunde gesetzt war. Bei Anwendung des gestaffelten Packalgorithmus verschieben sich die Laufzeitanteile klar in Richtung des Packalgorithmus und weg von der Tourenplanung. So zeigt sich in den Zeilen 8-15 von Tabelle 6.10, dass mindestens $98,5\%$ der Laufzeit für die Stauraumbepackung aufgewendet wurden, während der Anteil der Nachoptimierung *2-Opt* an der Laufzeit nicht mehr als $0,3\%$ ausgemacht hat. Die durch *2-Opt* erzielte Tourlängenreduktion bewegt sich hier im Bereich $2,3\%$ bis $6,1\%$, wobei die Ausreißer nach unten ($2,3\%$ und $3,1\%$ in den Zeilen 12 und 13) wieder im Fall der integrierten Methode unter Beachtung der LIFO-Reihenfolge auftreten.

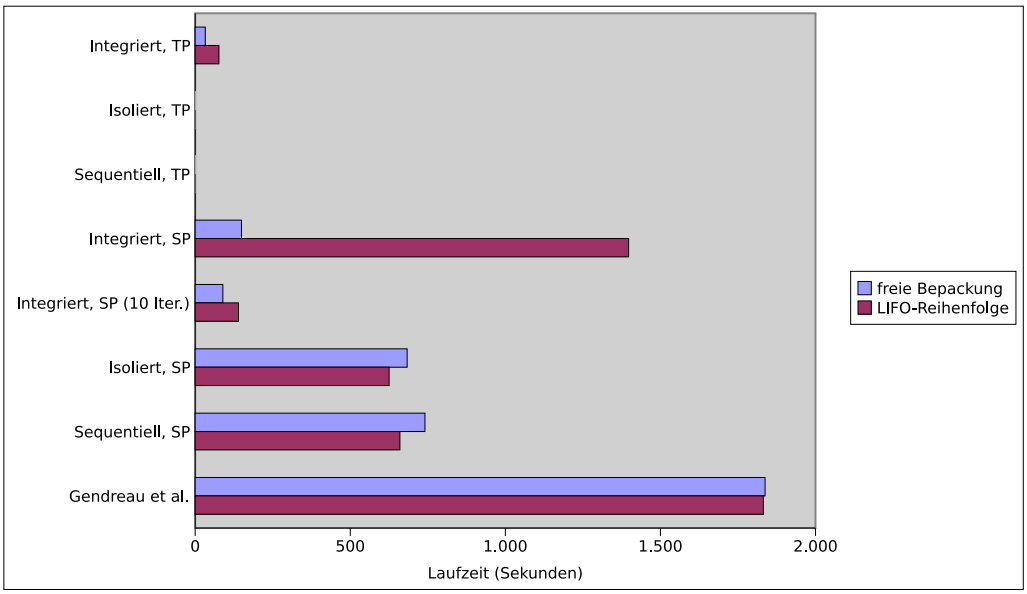


Abbildung 6.2: Laufzeiten mit und ohne LIFO-Bepackung

Die letzten beiden Spalten in Tabelle 6.10 beschreiben die durchschnittliche Effektivität $\%_{L_4}^{Erfolg}$ der unteren Schranke L_4 und die Effektivität $\%_{TP}^{Erfolg}$ der Heuristik TP_{MOD} innerhalb des gestaffelten Packverfahrens. Der Wert $\%_{L_4}^{Erfolg}$ gibt den Anteil von Aufrufen des gestaffelten Verfahrens *Staged-Packer* an, in denen L_4 einen Wert > 1 berechnet hat. Der Wert $\%_{TP}^{Erfolg}$ drückt den Anteil an Aufrufen von *Staged-Packer* aus, in denen TP_{MOD} erfolgreich war. In beiden Fällen muss der exakte Algorithmus MV_{MOD} nicht aufgerufen werden, was mit einer Laufzeitersparnis verbunden ist. Die untere Schranke L_4 ist wenig effektiv: es konnten nur zwischen zwischen $1,0\%$ und $1,7\%$ Packinstanzen als nicht lösbar identifiziert werden. Als wesentlich effektiver erweist sich die Heuristik TP_{MOD}, die bis zu $33,9\%$ der Aufrufe des exakten Packverfahrens im Vorfeld verhindern konnte. Die Bedingung der LIFO-Reihenfolge der Bepackung beeinflusst die Effektivität der Heuristik negativ, so dass im schlimmsten Fall eine Effektivität von nur $3,0\%$ erreicht werden konnte (Zeile 12).

Diagramm 6.2 stellt die Auswirkung der LIFO-Bedingung auf die Laufzeiten der einzelnen Verfahren dar. Dargestellt ist die mittlere Laufzeit pro Instanz jedes der Verfahren, wobei nur Instanzen der Itemklassen 2 bis 5 einbezogen wurden. Man beachte insbesondere den erheblichen Laufzeitzuwachs der integrierten Methode unter Verwendung des gestaffelten Verfahrens wenn die LIFO-Reihenfolge der Bepackung beachtet werden muss.

Ein Vergleich der Laufzeiten der implementierten Verfahren mit denen des Tabu-Search-Algorithmus von Gendreau et al. zeigt, dass die hier vorgestellten Verfahren in allen Fällen schneller sind – teilweise sogar um ein Vielfaches.

Die Diagramme 6.3 und 6.4 stellen die Laufzeiten der Testläufe dar, die im integrierten Modus durchgeführt wurden. Für die 144 Instanzen mit den Itemklassen 2 bis 5 wurden – genau wie in den Tabellen 6.5, 6.6, 6.7 und 6.8 – 36 Mittelwerte ermittelt (einer für jede der 36 VRP-Instanzen) und im Diagramm eingetragen. Auf der x-Achse sind folglich die VRP-Instanzen aufgetragen und auf der y-Achse die mittleren Laufzeiten in Sekunden.

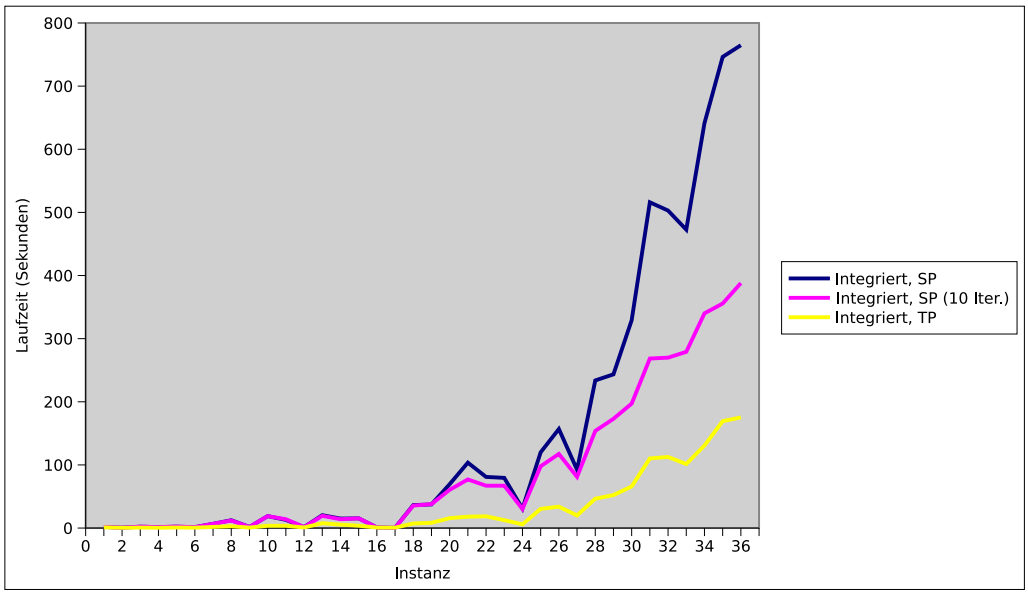


Abbildung 6.3: Laufzeiten der integrierten Methode für die 36 VRP-Instanzen, Mittelwerte für die Itemklassen 2 bis 5 (ohne LIFO-Bepackung)

Diagramm 6.3 veranschaulicht die Rechenzeiten ohne LIFO-Reihenfolge; Diagramm 6.4 zeigt die Rechenzeiten mit LIFO-Reihenfolge.

Bei Betrachtung von Diagramm 6.4 wird deutlich, dass die Laufzeit der integrierten Methode mit *Staged-Packer* unter Beachtung der LIFO-Reihenfolge für die Instanzen 28 bis 36 explosionsartig ansteigt, also für die Instanzen ab 120 Kunden und durchschnittlich mehr als 250 Items. Dieser Effekt wird durch das Setzen des Parameters $\gamma = 10$ wirkungsvoll eliminiert: die Laufzeiten des Verfahrens mit diesem Parameter sind durchgehend reduziert und bewegen sich knapp über den Laufzeiten der Methode mit TP_{MOD}. Ohne Beachtung der LIFO-Reihenfolge ist der Laufzeitzuwachs bei Berechnung der großen Instanzen sehr viel schwächer, wie in Diagramm 6.3 zu sehen ist. Hier hat der Parameter γ eine viel kleinere Auswirkung auf die Laufzeit.

Einen Abschluss der Analysen bildet eine Klassifikation der implementierten Verfahren bezüglich ihrer durchschnittlichen Lösungsqualität $\#v$ (Anzahl benötigter Fahrzeuge bzw. generierter Touren) und Laufzeit LZ. Hierfür wurden die Verfahren in zwei Pareto-Diagrammen eingetragen, wobei das Diagramm in Abbildung 6.5 die Ergebnisse ohne Beachtung der LIFO-Reihenfolge der Bepackung darstellt während das Diagramm in Abbildung 6.6 die Ergebnisse mit LIFO-Bepackung präsentiert.

Man betrachte zunächst das Diagramm 6.5. Die Werte der x-Achse repräsentieren durchschnittliche Laufzeiten in Sekunden pro Instanz und die der y-Achse kennzeichnen die durchschnittliche Anzahl generierter Touren. Ein Verfahren ist um so schneller je weiter links es sich im Diagramm befindet und es generiert um so bessere Ergebnisse je weiter unten es sich befindet. Ein Verfahren ist *pareto-optimal*, falls es kein anderes Verfahren gibt, das sowohl weiter unten als auch weiter links im Diagramm liegt. Ohne Beachtung der LIFO-Reihenfolge erweisen sich die integrierten Verfahren als insgesamt am besten. Sie weisen eine hohe Lösungsqualität auf und benötigen relativ geringe Laufzeiten. Schneller

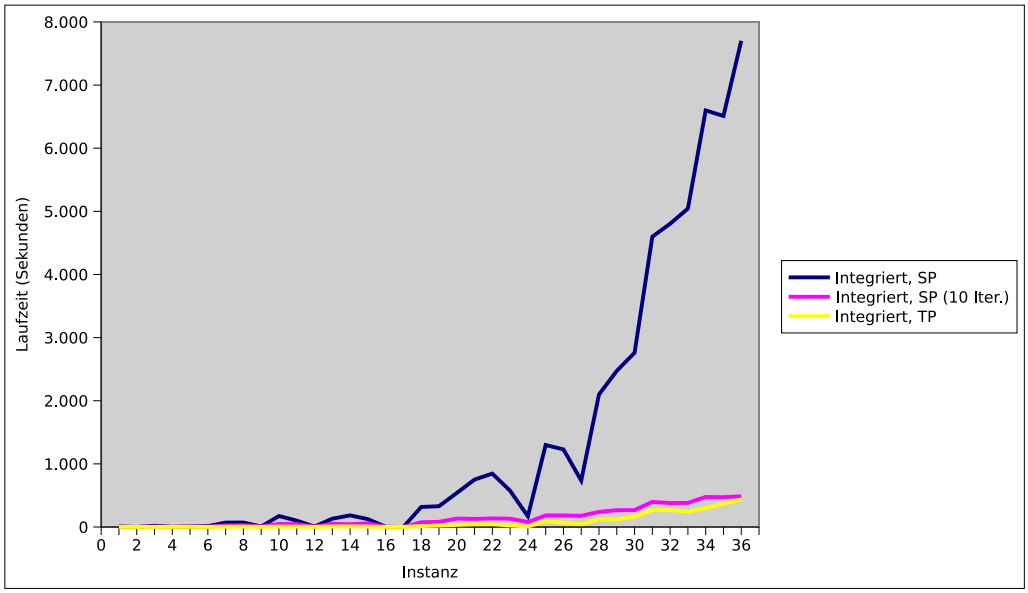


Abbildung 6.4: Laufzeiten der integrierten Methode für die 36 VRP-Instanzen, Mittelwerte für die Itemklassen 2 bis 5 (mit LIFO-Bepackung)

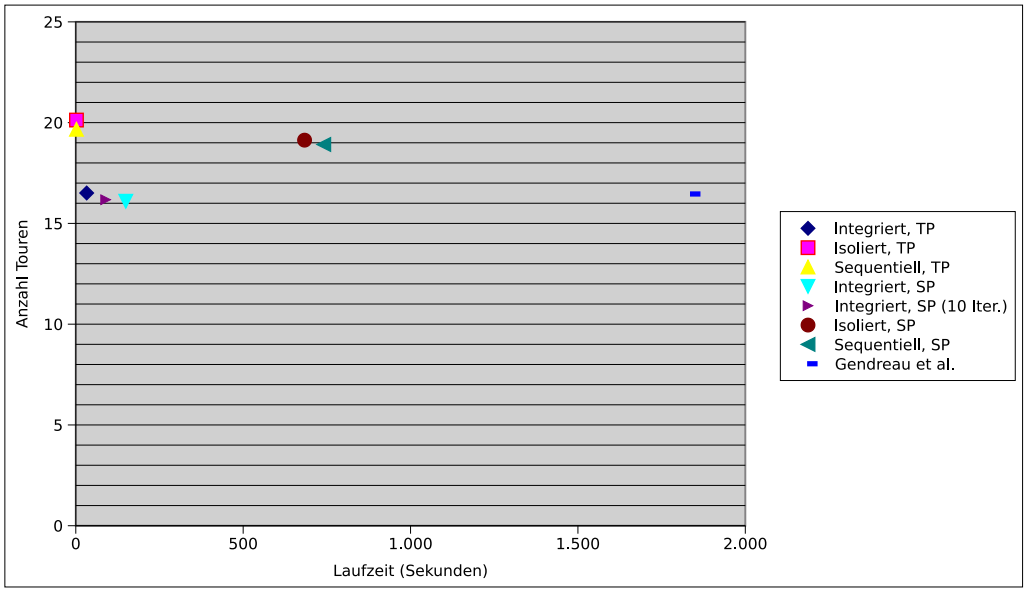


Abbildung 6.5: Pareto-Vergleich der implementierten Verfahren (ohne LIFO-Bepackung)

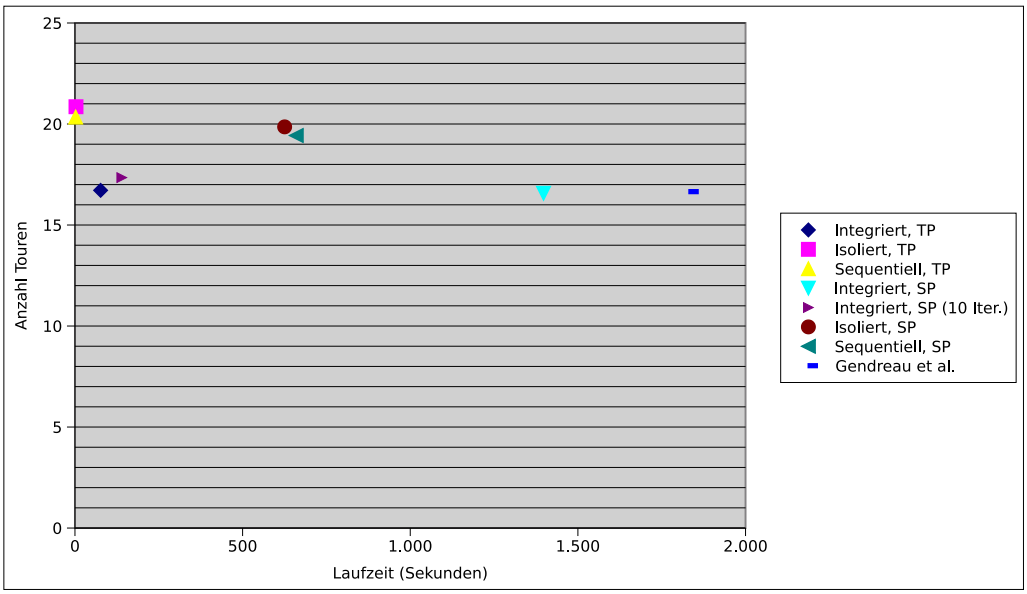


Abbildung 6.6: Pareto-Vergleich der implementierten Verfahren (mit LIFO-Bepackung)

sind nur die isolierte und die sequentielle Methode unter Verwendung des heuristischen Packalgorithmus. Diese sind jedoch bezüglich der Lösungsqualität nicht konkurrenzfähig. Das Tabu-Search-Verfahren von Gendreau et al. generiert bezüglich der Tourenanzahl ähnlich gute Ergebnisse wie die integrierten Verfahren, wobei es jedoch deutlich langsamer ist. Als im Allgemeinen nicht konkurrenzfähig erweisen sich das isolierte und das integrierte Verfahren in Verbindung mit *Staged-Packer* als Laderaumoptimierungsverfahren.

Diagramm 6.6 stellt die Situation unter Beachtung der LIFO-Reihenfolge der Bepackung dar. Abgesehen davon, dass die Verfahren im Durchschnitt mehr Touren generieren, ist das Bild ähnlich wie bei Nicht-Beachtung der LIFO-Reihenfolge. Eine Ausnahme stellt hier die integrierte Methode in kombination mit dem gestaffelten Verfahren dar, deren Laufzeit verglichen mit der Situation ohne LIFO-Reihenfolge deutlich zunimmt. Diese Eigenschaft kann durch Einschränkung der Iterationen ($\gamma = 10$) verbessert werden, auch wenn die Lösungsqualität etwas darunter leidet.

6.2.4 Fazit

Die implementierten Verfahren wurden ausführlichen Benchmarks unterzogen und anschließend miteinander sowie mit einem aktuellen Verfahren aus der Litaratur verglichen. Die Analysen zeigen, dass die isolierte und die sequentielle Methode zur Kombination des Tourenplanungsalgorithmus mit den implementierten Laderaumoptimierungsverfahren zwar sehr gute Laufzeiten liefern können, jedoch bezüglich der Lösungsqualität nicht mit der integrierten Methode konkurrieren können. Es wurde ein Laufzeitproblem der integrierten Methode festgestellt, wenn diese mit der Bedingung ausgeführt wird, die LIFO-Reihenfolge zu berücksichtigen. Das Problem konnte im Wesentlichen eliminiert werden, indem eine maximale Anzahl γ von Packversuchen pro Einfügung eines ungerouteten Kunden in eine Tour festgelegt wurde. Es wurde zur Überprüfung der Plausibilität der Ergebnisse ein Vergleich mit einem aktuellen Verfahren aus der Literatur durchgeführt.

Dabei haben sich die hier implementierten Verfahren als sehr schnell und zudem durchaus konkurrenzfähig bezüglich der Lösungsqualität erwiesen, auch wenn aufgrund der unterschiedlichen Zielfunktionen der Vergleich mit Vorsicht zu genießen ist.

Zusammenfassend kann man sagen, dass sich die integrierte Methode – gegebenenfalls unter Anwendung eines relativ kleinen Wertes γ – als die beste der vorgestellten Methoden erwiesen hat. Daher wird im folgenden Abschnitt, in dem Experimente mit großen Real-World-Eingabeinstanzen beschrieben werden, ausschließlich diese Methode eingesetzt.

6.3 Verhalten der Verfahren auf Real-World-Eingabeinstanzen

Für die in diesem Abschnitt beschriebenen experimentellen Analysen wurden Instanzen aus dem in Abschnitt 2.1.1 beschriebenen Projekt eingesetzt. Diese Eingabedaten sind dafür geeignet, die in dieser Arbeit implementierten Algorithmen realitätsnahen Tests zu unterziehen und ihre Leistungsfähigkeit bei besonders großen Problemstellungen für das kombinierte Tourenplanungs- und Laderaumoptimierungsproblem zu untersuchen. Der Fokus der hier beschriebenen Untersuchungen liegt auf den folgenden Fragestellungen:

- Welche Laufzeiten benötigen die implementierten Algorithmen für die Berechnung sehr großer Eingabeinstanzen?
- Wie verhalten sich Laufzeit und Lösungsqualität im Falle der Vorgabe, Kundenzeitfenster zu berücksichtigen?
- Inwiefern werden Laufzeit und Lösungsqualität beeinflusst, wenn die zu transportierenden Gegenstände um 90° im Frachtraum gedreht werden können?

6.3.1 Die Eingabedaten

Die Daten beinhalten Informationen über Transporte zu 9 Werken eines Automobilherstellers, welche innerhalb eines Monats von insgesamt 1.981 Zulieferern bedient wurden. Innerhalb dieses Zeitraums wurden 100.263 einzelne Sendungen[1] transportiert, wobei 26.632 dieser Sendungen an das größte und 2.580 Sendungen an das kleinste der 9 Werke geliefert wurden. Der maximale Umschlag des größten Werks an einem Tag belief sich auf 1.412 Sendungen, die von 515 verschiedenen Lieferanten stammten.

Auch wenn die Transporte unter Verwendung von stundengenauen Zeitfenstern getätigt wurden, sind die Zeitfensterinformationen leider nicht in den Daten enthalten – einzig das Anlieferdatum jeder Sendung wurde festgehalten.

Zu jeder Sendung existiert eine Gewichtsangabe, sowie eine Beschreibung der Anzahl und der Art der transportierten Objekte. Bei diesen Objekten handelt es sich um Gitterboxen, in denen die eigentlichen Güter untergebracht sind. Bei den Transporten des betrachteten Monats wurden insgesamt 1.338 unterschiedliche Arten solcher Gitterboxen verwendet. Sie unterscheiden sich in Breite, Höhe, Tiefe sowie in ihrer speziellen Stapeltechnik. Mehrerere Gitterboxen dürfen übereinander gestapelt werden, wenn (1) ihre Grundflächen identische

[1] Der Begriff Sendung bezeichnet einen Lieferauftrag zu einem Kunden (im Falle der Distribution von Gütern) bzw. von einem Lieferanten (im Falle der Beschaffung von Gütern).

Ausmaße (Breite und Tiefe) haben und (2) die Verankerungsmechanismen dieser Gitterboxen zueinander kompatibel sind, sie also über eine identische Stapeltechnik verfügen. Eine Sendung besteht aus bis zu 19 verschiedenen Boxentypen und bis zu 2.106 Boxen.

Zudem ethalten die Daten genaue Informationen zu 13 Fahrzeugtypen (insbesondere zu den Ausmaßen des Stauraums und zum maximal zulässigen Transportgewicht), wobei sowohl Volumenfahrzeuge mit einer Innenraumhöhe von 3 Metern und einem maximalen Transportgewicht von 24 Tonnen als auch kleinere LKW mit geringeren Ausmaßen und zulässigen Tonnagen beschrieben sind.

In dem angegebenen Zeitraum wurden insgesamt 23.253 Touren gefahren, bei denen eine durchschnittliche Volumenauslastung von $43,85\%$ und eine durchschnittliche Gewichtsauslastung von $41,07\%$ der Fahrzeuge erzielt wurde. Es wurden durchschnittlich $4,31$ Boxentypen und $39,42$ Boxen pro Tour transportiert.

Schließlich sind Informationen über die genauen Standorte der Kunden und der Werke in einem digitalen Straßennetz verfügbar, auf deren Grundlage eine Entfernungs- und eine Zeitmatrix berechnet wurde.

6.3.2 Generierung von Benchmarkinstanzen

Angesichts beschränkter Zeitressourcen und Rechenkapazitäten musste eine enge Auswahl von Benchmarkinstanzen getroffen werden. Es wurden drei Instanzen gebildet, von denen jede die Sendungen eines der drei Tage umfasst, an denen die meisten Sendungen an das Werk mit dem größten Umschlag geliefert wurden. Anders ausgedrückt werden drei Tage des größten der neun Werke betrachtet, an denen die meiste Last entstanden ist.

Die gewählten Instanzen umfassen zwischen 1.412 und 1.338 Sendungen und jeweils zwischen 12.713 und 12.212 Packstücke (Gitterboxen). Die Sendungen müssen unabhängig voneinander betrachtet werden, da in der Ausgangssituation jedem Lieferauftrag ein eigenständiges Zeitfenster zugeordnet war und somit nicht davon ausgegangen werden kann, dass mehrere Sendungen eines Lieferanten zwangsläufig innerhalb einer Tour eingesammelt und dem Werk zugestellt wurden. Somit muss im unterliegenden Modell jede Sendung wie ein eigenständiger Kunde behandelt werden, dem gegebenenfalls ein eigenes Zeitfenster zugeordnet ist. Im Folgenden werden deshalb die Begriffe Sendung und Kunde synonym verwendet.

Im Gegensatz zu den in Abschnitt 6.2 eingesetzten Daten aus [Gendreau, Iori, Laporte, und Martello 2008], in denen die zu transportierenden Objekte als zweidimensionale Items beschrieben sind, werden hier dreidimensionale Boxen beschrieben, aus denen vor Ausführung des Algorithmus Boxenstapel gebildet werden können. So entstehen durch Ausführung des Algorithmus *FFDStackBuilder* (siehe Abschnitt 4.2) zwischen 4.317 und 4.437 Boxenstapel, die im Verlauf der Optimierung in den Fahrzeugen verteilt werden müssen.

Tabelle 6.11 fasst die wichtigsten Kennzahlen zu den gewählten Eingabeinstanzen zusammen. Zu diesen gehören die Anzahl Kunden (bzw. Sendungen), die Anzahl Boxen und die Anzahl Boxenstapel der jeweiligen Instanz.

Da die Daten die ursprünglich einzuhaltenden Zeitfenster nicht enthalten, wurde für jede Sendung ein Zeitfenster künstlich generiert. Da neben dem Einfluss von engen Zeitfenstern

Instanz	*Kunden*	*Boxen*	*Boxenstapel*
1	1.412	12.713	4.395
2	1.389	12.417	4.317
3	1.338	12.212	4.437

Tabelle 6.11: Ausgewählte Real-World-Instanzen

auch die Auswirkung der Drehbarkeit der transportierten Objekte auf das Verhalten des Algorithmus untersucht werden sollte, wurde jede der Instanzen unter vier Bedingungen durchgerechnet:

- Itemorientierung fix, Tageszeitfenster
- Items drehbar, Tageszeitfenster
- Itemorientierung fix, enge Zeitfenster
- Items drehbar, enge Zeitfenster

Mit *Tageszeitfenster* wird der 24-Stunden-Zeitraum bezeichnet, der um 0:00 Uhr des ursprünglichen Zustellungsdatums einer Sendung beginnt und um 0:00 des nachfolgenden Tages endet. Sei $TZF(i) = \left[e_{TZF(i)}, l_{TZF(i)}\right]$ das Tageszeitfenster einer Sendung i. Das *enge Zeitfenster* $ZF(i)$ wird auf Grundlage des Tageszeitfensters wie folgt generiert:

1. Bestimme die Dauer $T_{ZF(i)}$ des Zeitfensters uniform zufällig im Bereich 2 Stunden bis 4 Stunden
2. Bestimme den Beginn $e_{ZF(i)}$ des neuen Zeitfensters innerhalb des Intervalls $\left[e_{TZF(i)}, l_{TZF(i)} - T_{ZF(i)}\right]$ uniform zufällig.
3. Das enge Zeitfenster von i ergibt sich dann als $ZF(i) = \left[e_{ZF(i)}, e_{ZF(i)} + T_{ZF(i)}\right]$

6.3.3 Experimentelles Setup

Die Experimente wurden auf einem AMD Athlon XP 2000+ (1667 MHz) mit 1GB RAM unter der Linux-Distribution Debian 4.0 (Etch) durchgeführt. Als Algorithmus für das kombinierte Tourenplanungs- und Laderaumoptimierungsproblem wurde die integrierte Methode (siehe Abschnitt 5.2) eingesetzt, die sich in den in Abschnitt 6.2 beschriebenen Analysen als qualitativ am besten und zudem als genügend schnell herausgestellt hat. Diese Analysen haben ebenfalls ergeben, dass eine Einschränkung der Packversuche innerhalb des Tourenplanungsalgorithmus *I1* auf einen kleinen Wert γ eine große Laufzeitreduktion bei nur geringfügig verschlechtertem Zielfunktionswert nach sich zieht. Da die in diesem Abschnitt verwendeten Instanzen wesentlich mehr Kunden enthalten, wurde für alle Testläufe der Wert $\gamma = 20$ statt – wie zuvor – 10 gesetzt. Diese Erhöhung des γ-Wertes soll es dem Algorithmus ermöglichen, einen größeren Teil der möglichen Einfügekombinationen von Kunden in eine offene Tour auszuprobieren und so potentiell bessere Lösungen zu

Inst.	*Zeitfenster*	$\#v$	z	*LZ*
1	nein	109	5.015.841	571,4
2	nein	104	4.987.601	532,3
3	nein	119	5.240.260	409,2
1	ja	110	8.298.712	128,0
2	ja	105	8.223.123	119,5
3	ja	120	9.414.445	101,9

Tabelle 6.12: Ergebnisse VRPTW

finden. Der genaue Einfluss von γ auf die Laufzeit und die Lösungsqualität bleibt eine zu näher zu untersuchende Fragestellung.

Als Packalgorithmen wurden sowohl die Heuristik TP_{MOD} als auch das gestaffelte Verfahren *Staged-Packer* (kurz: *SP*) eingesetzt. Dem exakten Algorithmus MV_{MOD} innerhalb von *SP* wurde eine Rechenzeit von maximal einer Sekunde pro Aufruf zugeteilt. Die Testläufe mit TP_{MOD} wurden mit der Brute-Force-Methode (siehe Abschnitt 6.2) durchgeführt, für die Läufe mit dem gestaffelten Verfahren wurde die Auswahl-Methode eingesetzt. Um die Auswirkung der Fahrzeugbepackung auf die Zielfunktion zu beobachten, wurden die Instanzen zusätzlich als reines VRP gelöst. Zu diesem Zweck wurden die Ausmaße aller Boxen auf $(w, h, d) = (1, 1, 1)$ festgelegt, so dass alle Boxen einer Instanz problemlos in einem LKW untergebracht werden können.

Als Transportfahrzeug wurde ein Großvolumenfahrzeug mit dem zulässigen Transportgewicht von 24t und den Innenraumausmaßen $W = 2,4\text{m}$, $H = 3,0\text{m}$ und $D = 13,6\text{m}$ gewählt. Als sekundäres Kriterium der Zielfunktion wurde die Fahrtdauer verwendet. Alle Testläufe wurden unter Einhaltung der LIFO-Reihenfolge durchgeführt. Die Nachoptimierung *2-Opt* kam hier nicht zum Einsatz, denn dieses Verfahren dient zur Minimierung der Tourlänge, nicht der Fahrtdauer.

6.3.4 Ergebnisse

Die Ergebnisse der Benchmarks sind in den Tabellen 6.12, 6.13 und 6.14 zusammengefasst.

Tabelle 6.12 enthält die Ergebnisse der Läufe ohne Überprüfung der Fahrzeugbepackung, mit und ohne Einhaltung von Zeitfenstern. In der dritten Spalte ist für jede Instanz die Anzahl $\#v$ von Touren bzw. Fahrzeugen angegeben; Spalte 4 enthält die Gesamt-Fahrzeit z in Sekunden; in Spalte 5 ist die Laufzeit des Algorithmus in Sekunden angegeben.

Tabelle 6.13 enthält die entsprechenden Ergebnisse der Testläufe, in denen TP_{MOD} zur Berechnung der Bepackung eingesetzt wurde; in Tabelle 6.14 sind die Ergebnisse unter Verwendung des gestaffelten Packverfahrens aufgeführt. In diesen beiden Tabellen wird zusätzlich zwischen fixer und variabler Orientierung der Boxenstapel im Frachtraum unterschieden.

Für eine bessere Übersicht der Ergebnisse betrachte man Tabelle 6.15, in der Durchschnittswerte der Ergebnisse für die drei eingesetzten Testinstanzen aufgeführt sind.

Es werden die folgenden Bezeichner verwendet:

		Items fix			*Items drehbar*		
Inst.	*Zeitfenster*	*#v*	*z*	*LZ*	*#v*	*z*	*LZ*
1	nein	233	11.533.761	520,9	225	6.848.961	588,9
2	nein	231	11.283.761	506,6	224	10.410.521	550,1
3	nein	240	7.197.220	423,9	231	7.041.580	481,8
1	ja	234	13.875.997	179,2	225	12.245.880	238,5
2	ja	232	13.252.529	170,2	224	13.161.835	213,7
3	ja	241	12.637.342	160,8	231	11.644.047	200,2

Tabelle 6.13: Ergebnisse TP_{MOD}

		Items fix			*Items drehbar*		
Inst.	*Zeitfenster*	*#v*	*z*	*LZ*	*#v*	*z*	*LZ*
1	nein	235	7.684.821	1.050,6	209	7.411.041	1.829,6
2	nein	233	6.981.221	855,3	210	6.681.221	2.045,6
3	nein	240	7.136.320	766,9	216	6.952.780	1.916,8
1	ja	235	12.475.693	491,6	210	12.024.718	1.460,8
2	ja	232	13.271.087	662,9	211	12.961.951	1.664,2
3	ja	240	12.767.564	544,4	217	12.860.169	1.623,0

Tabelle 6.14: Ergebnisse gestaffeltes Packverfahren

	Packer	*drehbar*	*Zeitfenster*	*#v*	*z*	*%w*	*%vol*	*%a*	*#b*
1	VRPTW		nein	110,7	5.081.234,0	99,6%	–	–	–
2	VRPTW		ja	111,7	8.645.426,7	98,7%	–	–	–
3	TP_{MOD}	nein	nein	234,7	10.004.914,0	47,0%	66,4%	85,4%	52,3
4	TP_{MOD}	nein	ja	235,7	13.255.289,3	46,8%	66,1%	85,1%	52,3
5	TP_{MOD}	ja	nein	226,7	8.100.354,0	48,6%	68,7%	88,4%	54,3
6	TP_{MOD}	ja	ja	226,7	12.350.587,3	48,6%	68,7%	88,4%	54,3
7	*SP*	nein	nein	236,0	7.267.454,0	46,7%	66,0%	85,0%	52,3
8	*SP*	nein	ja	235,7	12.838.114,7	46,8%	66,1%	85,1%	52,3
9	*SP*	ja	nein	211,7	7.015.014,0	52,1%	73,6%	94,7%	58,3
10	*SP*	ja	ja	212,7	12.615.612,7	51,8%	73,2%	94,3%	58,0

Tabelle 6.15: Lösungsqualität: Durchschnittswerte über die Instanzen 1-3

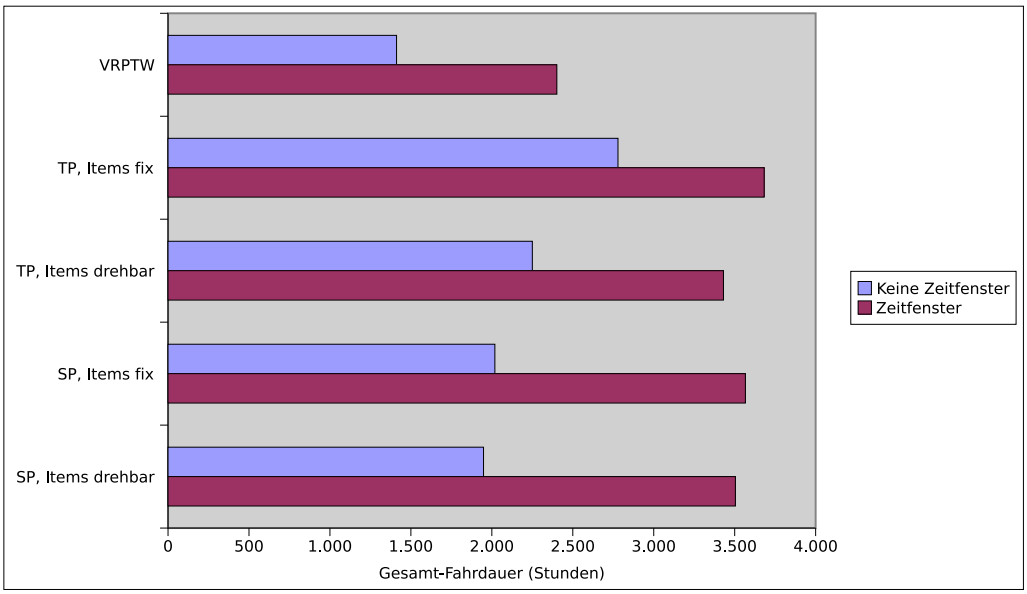


Abbildung 6.7: Einfluss von Zeitfenstern auf die Fahrzeit

- $\#v$: Anzahl Touren
- z: Gesamt-Tourdauer in Sekunden
- $\%w$: Gewichtsauslastung
- $\%vol$: Nutzung des Frachtraumvolumens
- $\%a$: Nutzung der Frachtraumfläche
- $\#b$: Anzahl Gitterboxen pro Fahrzeug

Die Ergebnisse zeigen, dass die Berücksichtigung der Fahrzeugbepackung zu einer deutlichen Erhöhung der Tourenanzahl $\#v$ führt. Der Zuwachs beträgt in den meisten Fällen mehr als 100% mit Ausnahme der Fälle, in denen das gestaffelte Verfahren eingesetzt wurde und die Boxenstapel gedreht werden durften. Interessant ist, dass die Drehbarkeit von Items eine signifikante Reduktion der generierten Touren mit sich bringt. Diese Reduktion fällt bei Einsatz des gestaffelten Algorithmus mit $11,5\%$ (ohne Zeitfenster) bzw. $10,8\%$ (mit Zeitfenstern) stärker aus als bei Verwendung der Heuristik TP_{MOD} ($3,5\%$ bzw. $4,0\%$). Das Diagramm in Abbildung 6.8 stellt den Zusammenhang zwischen der Drehbarkeit von Items und der Anzahl Touren dar.

Die Zeitfenster wirken sich hingegen kaum auf die Tourenanzahl aus. Vielmehr verursachen diese einen deutlichen Zuwachs an Fahrtdauer z: Bei Verwendung von TP_{MOD} beträgt der Zuwachs $32,5\%$ (fixe Itemorientierung) bzw. $52,5\%$ (drehbare Items) und bei Verwendung von *Staged-Packer* sogar $76,7\%$ (Items fix) bzw. $79,8\%$ (Items drehbar). Dieser Zusammenhang wird in Abbildung 6.7 für die fünf untersuchten Konfigurationen verdeutlicht.

Die mittlere Gewichtsauslastung $\%w$ der Fahrzeuge sinkt bei Berechnung der Fahrzeugbepackung auf $46,7\%$ bis $52,1\%$ gegenüber einer Auslastung von mindestens $98,7\%$, falls das

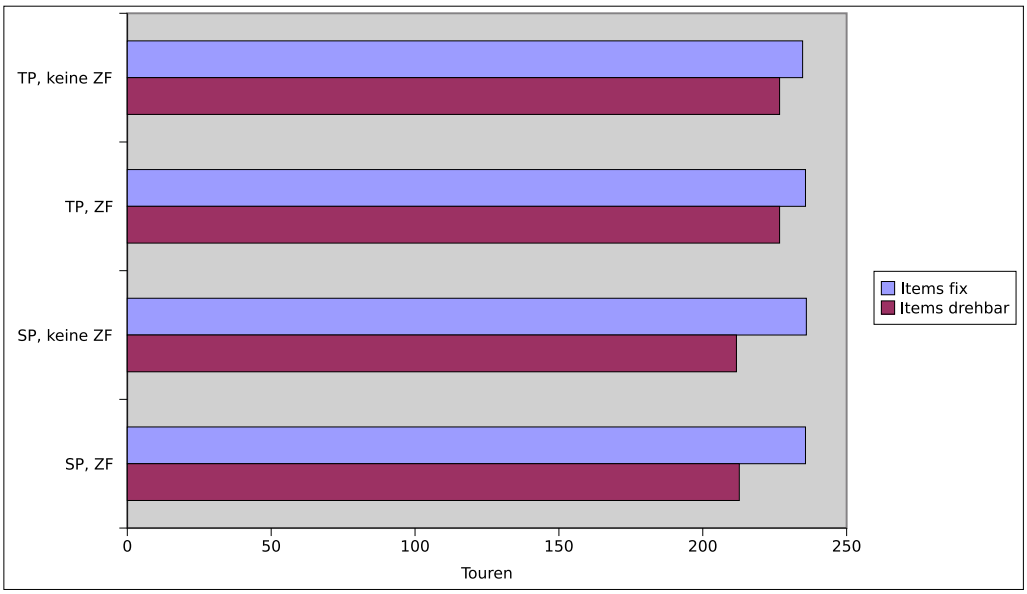


Abbildung 6.8: Einfluss der Drehbarkeit von Items auf die Tourenanzahl

reine VRPTW gelöst wird. Es werden eine mittlere Auslastung $\%vol$ des Fahrzeugvolumens von $66,0\%$ bis $73,6\%$ und eine mittlere Auslastung $\%a$ der Stellfläche des Fahrzeuginnenraums von $85,0\%$ bis $94,7\%$ erreicht. Die Anzahl $\#b$ durchschnittlich transportierter Boxen pro Fahrzeug beträgt zwischen 52,3 und 58,0.

Zum Abschluss betrachte man Tabelle 6.16, die einige Kennzahlen zum Verhalten der Algorithmen aufführt. Die hier verwendeten Spaltenbezeichnungen entsprechen denen in Tabelle 6.10 (Abschnitt 6.2).

Die maximal benötigte mittlere Laufzeit LZ pro Instanz beträgt 1.930,6 Sekunden im Falle des gestaffelten Verfahrens *Staged-Packer* mit drehbaren Items und ohne Zeitfenster (Zeile 9). Interessant ist, dass die Berechnung des reinen VRPTW mit dem einheitlichen Tageszeitfenster für alle Kunden mehr Rechenzeit beansprucht als drei der vier Setups mit TP_{MOD} als Packalgorithmus. Der Grund dafür ist, dass der VRTWLP-Algorithmus

	Packer	*drehbar*	*Zeitfenster*	LZ	$\%^{LZ}_{Packer}$	$\#_{Packer}$	$\%^{Erfolg}_{L_4}$	$\%^{Erfolg}_{TP}$
1	VRPTW		nein	504,3	–	–	–	–
2	VRPTW		ja	116,5	–	–	–	–
3	TP_{MOD}	nein	nein	483,8	15,0%	182.395,0	–	–
4	TP_{MOD}	nein	ja	170,1	38,4%	168.910,7	–	–
5	TP_{MOD}	ja	nein	540,3	22,5%	180.160,7	–	–
6	TP_{MOD}	ja	ja	217,5	51,2%	166.526,0	–	–
7	*SP*	nein	nein	890,9	50,1%	7.742,0	34,7%	17,6%
8	*SP*	nein	ja	566,3	81,1%	7.453,7	37,5%	18,3%
9	*SP*	ja	nein	1.930,6	76,3%	6.909,7	1,9%	17,5%
10	*SP*	ja	ja	1.582,7	93,0%	6.551,7	1,7%	18,4%

Tabelle 6.16: Verhalten der Algorithmen: Durchschnittswerte über die Instanzen 1-3

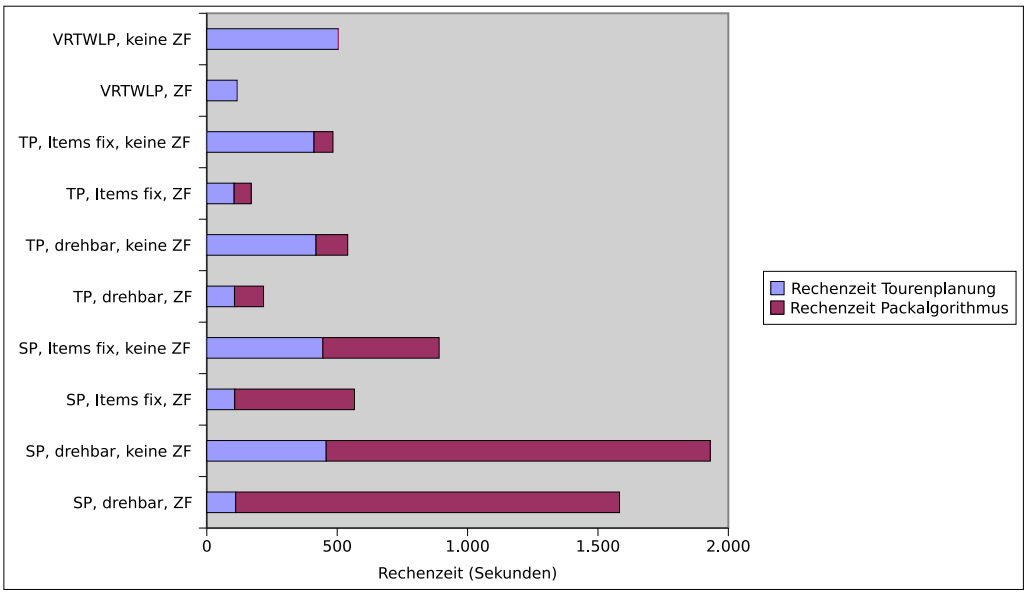


Abbildung 6.9: Verteilung der Rechenzeit auf die Algorithmen

unter Verwendung von TP_{MOD} (wie auch von *Staged-Packer*) pro Einfügung eines Kunden in eine Tour nur maximal $\gamma = 20$ Kombinationen ausprobiert, während im Falle des reinen VRPTW $O(n^2)$ Kombinationen aus der Menge der ungerouteten Kunden und der Menge der Einfügepositionen in der Tour getestet werden. Hinzu kommt, dass die Touren im Falle des VRPTW im Schnitt mehr Kunden enthalten und somit die Anzahl potentieller Einfügekombinationen steigt. Einen Überblick über die Anteile der Algorithmen an den Gesamtrechenzeiten bietet das Diagramm in Abbildung 6.9. Es fällt auf, dass der Laufzeitanteil des Tourenplanungsalgorithmus regelmäßig steigt, wenn keine Zeitfenster berücksichtigt werden. Eine weitere Beobachtung ist, dass *Staged-Packer* im Schnitt eine wesentlich höhere Laufzeit beansprucht, wenn die Boxenstapel gedreht werden dürfen.

Interessant ist zudem, dass die untere Schranke L_4 bei diesen Instanzen eine weitaus bessere mittlere Erfolgsquote $\%_{L_4}^{Erfolg}$ aufweist, als es beim Berechnen der Instanzen von Gendreau et al. der Fall war. Diese Quote bricht drastisch ein, wenn die Boxenstapel gedreht werden dürfen. Der Anteil erfolgreicher Aufrufe der Heuristik TP_{MOD} aus dem gestaffelten Verfahren heraus ist hingegen geringer als bei den Experimenten, die in Abschnitt 6.2 beschrieben werden.

6.3.5 Fazit

Die implementierten Algorithmen wurden Benchmarks auf Real-World-Eingabedaten unterzogen, deren Größe die der synthetisch generierten Instanzen aus der Literatur um ein Vielfaches übersteigt. Die für diese Tests ausgewählten Algorithmen haben bei recht kurzen Laufzeiten stabile Ergebnisse mit zufriedenstellender Auslastung der Fahrzeuge geliefert. Die Experimente haben bestätigt, dass eine signifikante Erhöhung der Auslastung des Frachtraums erzielt werden kann, wenn Boxenstapel gedreht werden dürfen. Schließlich wurde ein großer Einfluss von engen Zeitfenstern auf die Tourdauer nachgewiesen.

Kapitel 7

Fazit und Ausblick

Die Zielsetzung dieser Arbeit war es, ein leistungsfähiges Verfahren zur Lösung des kombinierten Tourenplanungs- und Laderaumoptimierungsproblems zu konstruieren. Dieses Ziel wurde durch Implementierung und Verknüpfung von geeigneten Algorithmen für die Teilprobleme der Tourenplanung und der Laderaumoptimierung erreicht.

Als Tourenplanungsalgorithmus wurde das Verfahren *I1* gewählt, welches einen guten Kompromiss aus hoher Ausführungsgeschwindigkeit und guter Lösungsqualität bietet. Der Algorithmus war wegen seiner zielgerichteten Arbeitsweise ein sehr guter Kandidat für die Bewältigung der vorliegenden Aufgabenstellung. Das Verfahren zeichnet sich zudem dadurch aus, dass es durch explizite Berücksichtigung von Zeitfenstern eine der wichtigsten Anforderungen aus der logistischen Praxis erfüllt. Durch geeignete Parametrisierung kann das Verhalten des Algorithmus in Richtung kurzer Fahrzeiten oder kurzer Fahrstrecken beeinflusst werden. Eine weitere Reduktion der Tourlängen konnte durch den Einsatz der Intratour-Nachoptimierung *2-Opt* erreicht werden.

Das in seiner Grundform dreidimensionale Laderaumoptimierungsproblem wurde in seiner Komplexität reduziert. Die Reduktion besteht darin, dass zunächst durch Lösung des eindimensionalen Bin-Packing-Problems aus der Menge der zu transportierenden Boxen geeignete Stapel gebildet werden. Diese werden anschließend durch Lösung des zweidimensionalen, orthogonalen Packproblems auf der Grundfläche des LKW-Frachtraums verteilt. Für die Bildung von Boxenstapeln wurde die Bin-Packing-Heuristik *First-Fit-Decreasing* implementiert, welche bei hoher Ausführungsgeschwindigkeit qualitativ hochwertige Lösungen liefert. Für das zweidimensionale Packproblem wurden mehrere Algorithmen implementiert und getestet. Die Heuristik TP_{MOD} erwies sich dabei als sehr schnell und lieferte bereits gute Ergebnisse. Eine qualitative Steigerung der Ergebnisse der Stauraumbepackung konnte durch Einsatz des exakten Branch-and-Bound-Verfahrens MV_{MOD} erzielt werden. Schließlich wurde ein zweites exaktes Packvervahren implementiert, nämlich der Algorithmus $LMAO_{MOD}$, welcher eine Verbesserung gegenüber MV_{MOD} darstellen sollte. Es zeigte sich jedoch, dass $LMAO_{MOD}$ mangels einer zulässigen unteren Schranke MV_{MOD} unterlegen ist. Deswegen musste $LMAO_{MOD}$ schließlich verworfen werden. Um Aufrufe des potentiell laufzeitintensiven Algorithmus MV_{MOD} zu vermeiden, wurde das gestaffelte Verfahren *Staged-Packer* konstruiert. Dieses prüft vor Aufruf des exakten Algorithmus den Wert der unteren Schranke L_4 und führt gegebenenfalls die Heuristik TP_{MOD} aus. Erst wenn weder die Nicht-Lösbarkeit einer Eingabeinstanz durch die untere Schranke noch

ihre Lösbarkeit durch die Heuristik festgestellt werden können, wird das exakte Verfahren ausgeführt. Alle implementierten Verfahren zur Stauraumbepackung berücksichtigen die Anforderung der Einhaltung der sogenannten LIFO-Reihenfolge, welche in der logistischen Praxis üblich ist. Zudem kann die Bepackung mit fixer oder mit variabler Orientierung der Packstücke durchgeführt werden.

Es wurden drei Möglichkeiten implementiert, die Tourenplanung und die Stauraumbepackung miteinander zu kombinieren: die integrierte, die isolierte und die sequentielle Methode. In der integrierten Methode werden die Tourenplanung und die Laderaumoptimierung eng miteinander verknüpft – die Stauraumbepackung wird bei jeder Einfügung eines Kunden in eine Tour überprüft. Die isolierte Methode führt die Verfahren getrennt voneinander aus, wobei die Tourenplanung und die Bepackung der resultierenden Touren in abwechselnder Reihenolge durchgeführt werden. Die sequentielle Methode beschreitet einen Mittelweg zwischen den ersten beiden Methoden. Hierbei wird die Eigenschaft des Algorithmus *I1* ausgenutzt, Touren sequentiell zu generieren. So wird für jede generierte Tour die Laderaumbepackung durchgeführt. Es werden diejenigen Kunden aus der Tour entfernt und wieder der Menge der zu bearbeitenden Kunden zugeführt, deren Packstücke nicht im LKW untergebracht werden konnten.

Die implementierten Verfahren wurden ausführlichen experimentellen Analysen unterzogen, in denen sie hinsichtlich ihrer Funktionsfähigkeit und Leistungsfähigkeit überprüft wurden. Zunächst wurden auf Grundlage von Benchmarkinstanzen aus der Literatur die Laufzeiten der beiden exakten Packalgorithmen MV_{MOD} und $LMAO_{MOD}$ verglichen. In diesen Tests wurde eine klare Überlegenheit von MV_{MOD} deutlich, weshalb dieser Algorithmus, neben der Heuristik TP_{MOD}, in den nachfolgenden Analysen der Verfahren für das kombinierte Problem eingesetzt wurde.

Es wurden Analysen für das kombinierte Verfahren auf der Grundlage von zwei verschiedenen Datensätzen durchgeführt. Zum Vergleich des Verhaltens der integrierten, der isolierten und der sequentiellen Methode, sowie zur Überprüfung der Auswirkungen der LIFO-Reihenfolge der Bepackung auf das Verhalten der Algorithmen wurden Benchmarkinstanzen aus der Literatur herangezogen. Zur Analyse des Verhaltens der Algorithmen bei der Bewältigung sehr großer Eingabeinstanzen sowie bei Berücksichtigung von Kundenzeitfenstern wurden Real-World-Lieferdaten eines Unternehmens aus der Automobilbranche herangezogen.

Die auf Grundlage der Literaturdaten durchgeführten Experimente ergaben eine qualitative Überlegenheit der integrierten gegenüber der sequentiellen und der isolierten Methode, wobei die beiden letzteren im Allgemeinen geringere Laufzeiten aufweisen. Es konnte auch gezeigt werden, dass der Einsatz des exakten Packverfahrens mit Laufzeitbeschränkung verglichen mit dem heuristischen Verfahren eine signifikante Verbesserung der erzielten Lösungen nach sich zieht. Untersuchungen unter Einhaltung der LIFO-Reihenfolge der Bepackung enthüllten Laufzeitprobleme der integrierten Methode in Kombination mit dem exakten Packalgorithmus. Diese konnten behoben werden, indem innerhalb des Tourenplanungsalgorithmus *I1* die Anzahl von Packversuchen pro Einfügung eines Kunden in eine Tour durch den Parameter γ beschränkt wurde.

Die auf Grundlage der Real-World-Daten durchgeführten Analysen haben bestätigt, dass die integrierte Methode auch große Probleminstanzen in kurzer Laufzeit lösen kann. Zudem ergaben die Experimente, dass die Berücksichtigung von engen Zeitfenstern einen Zuwachs der Gesamt-Fahrtdauer nach sich zieht, während das Zulassen freier Orientie-

rung der Packstücke zu einer besseren Auslastung der Fahrzeuge und somit zu einer Reduktion der Tourenanzahl führt. Die untere Schranke L_4 hat in diesen Tests eine weitaus größere Effektivität gezeigt als in den Experimenten auf den Benchmarkinstanzen aus der Literatur.

Es bleiben einige offene Fragen und Verbesserungsmöglichkeiten. Zum einen bedürfte es genauerer Analysen um festzustellen, welchen Einfluss der Parameter γ auf das Laufzeitverhalten und die Lösungsqualität des integrierten Verfahrens hat. Zum anderen wäre es interessant zu untersuchen, inwiefern ein Verfahren zur Intertour-Nachoptimierung[1] im Kontext des kombinierten Tourenplanungs- und Laderaumoptimierungsproblems die Anzahl benötigter Fahrzeuge senken kann. Auch wäre es eine interessante Herausforderung, ein Verbesserungsverfahren zur Tourenplanung – wie beispielsweise einen Tabu-Search-Algorithmus – effizient in das bestehende Framework zu integrieren.

Das kombinierte Tourenplanungs- und Laderaumoptimierungsproblem stellt durch seine hohe Komplexität und die große Vielfalt möglicher Varianten und Nebenbedingungen ein großes und interessantes Forschungsgebiet dar. Zudem besteht in der logistischen Praxis ein großer Bedarf nach guten Algorithmen für diese Problemstellung – sei es in der strategischen Planung oder in der Disposition, die in vielen Unternehmen immer noch ohne Unterstützung durch einen Computer durchgeführt wird. Deshalb sind in der Zukunft weitere Neuentwicklungen und Verbesserungen auf dem Gebiet der kombinierten Tourenplanung und Laderaumoptimierung zu erwarten.

[1] Im Gegensatz zu Verfahren für die *Intra*tour-Optimierung, welche Tourlängen oder Tourdauern minimieren, reduzieren Algorithmen zur *Inter*tour-Optimierung ebenfalls die Anzahl von Touren.

Literaturverzeichnis

Antes, J. und U. Derigs (1995). A New Parallel Tour Construction Algorithm for The Vehicle Routing Problem with Time Windows. Technischer report, Universität zu Köln, Lehrstuhl für Wirtschaftsinformatik und Operations Research.

Bent, R. und P. V. Hentenryck (2001). A Two-Stage Hybrid Local Search for the Vehicle Routing Problem with Time Windows. Technischer Report CS–01–06, Brown University, Providence, Department of Computer Science.

Berkey, J. und P. Wang (1987). Two dimensional finite bin packing algorithms. *Journal of the Operational Research Society 38*(5), 423–429.

Bischoff, E. (2006). Three-dimensional packing of items with limited load bearing strength. *European Journal of Operational Research 168*(3), 952–966.

Boschetti, M. A. (2004). New lower bounds for the three-dimensional finite bin packing problem. *Discrete Applied Mathematics 140*(1-3), 241–258.

Boschetti, M. A. und A. Mingozzi (2003). The Two-Dimensional Finite Bin Packing Problem. Part II: New lower and upper bounds. *4OR 1*(2), 135–147.

Bräysy, O. (2003). A Reactive Variable Neighborhood Search for the Vehicle-Routing Problem with Time Windows. *INFORMS Journal on Computing 15*(4), 347–368.

Bräysy, O. und M. Gendreau (2005). Vehicle Routing Problem with Time Windows, Part I: Route Construction and Local Search Algorithms. *TRANSPORTATION SCIENCE 39*(1), 104–118.

Buchholz, J., U. Clausen, und A. Vastag (Eds.) (1998). *Handbuch der Verkehrslogistik*. Logistik in Industrie, Handel und Dienstleistungen. Springer.

Bullnheimer, B., R.-F. Hartl, und C. Strauss (1998). *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, Chapter Applying the Ant System to the Vehicle Routing Problem, S. 109–120. Kluwer Academics.

Christofides, N. und C. A. Whitlock (1977). An algorithm for two-dimensional cutting problems. *Operations Research 25*, 30–44.

Chung, F., M. Garey, und D. Johnson (1982). On packing two-dimensional bins. *SIAM Journal of Algebraic and Discrete Methods 3*, 66–76.

Clarke, G. und W. Wright (1964). Scheduling of Vehicles from a Central Depot to a Number of Delivery Points. *Operations Research 12*(4), 568–581.

Clautiaux, F., J. Carlier, und A. Moukrim (2007, Dezember). A new exact method for the two-dimensional orthogonal packing problem. *European Journal of Operational Research 127*(3), 1196–1211.

Colorni, A., M. Doringo, und V. Maniezzo (1991). Distributed Optimization by Ant Colonies. In F. Varela und P. Bourgine (Eds.), *Proceedings of the First European Conference on Artificial Life*, S. 134—142. Elsevier Publishing.

Cordeau, J., M. Gendreau, G. Laporte, J. Potvin, und F. Semet (2002). A guide to vehicle routing heuristics. *Journal of the Operational Research Society 53*(5), 512–522.

Cordeau, J.-F. und G. Laporte (2002). Tabu search heuristics for the vehicle routing problem. Technischer Report G–2002–15, Les Cahiers du GERAD.

Cordeau, J.-F., G. Laporte, und A. Mercier (2001). A unified tabu search heuristic for vehicle routing problems with time windows. *Journal of the Operational Research Society 52*(8), 928–936.

Croes, G. A. (1958). A method for solving traveling salesman problems. *Operations Research 6*, 791–812.

Dantzig, G. und J. Ramser (1959). The truck dispatching problem. *Management Science 6*, 80–91.

Dell'Amico, M., F. Maffioli, und S. Martello (Eds.) (1997). *Annotated Bibliographies in Combinatorial Optimization*. John Wiley & Sons Canada, Ltd.

Dell'Amico, M. (1999). On the continuous relaxation of packing problems. Technischer report, Università di Modena, Dipartimento di Economia Politica.

Dyckhoff, H. (1990). A typology of cutting and packing problems. *European Journal of Operational Research 44*(2), 145–159.

Fekete, S. und J. Schepers (2001). New classes of fast lower bounds for bin packing problems. *Mathematical Programming 91*(1), 11–31.

Fekete, S., J. Schepers, und J. van der Veen (2007). An Exact Algorithm for Higher-Dimensional Orthogonal Packing. *Operations Research 55*(3), 569–587.

Frenk, J. und G. Galambos (1987). Hybrid next-fit algorithm for the two-dimensional rectangle bin-packing problem. *Computing 39*(3), 201–217.

Gambardella, L., E. Taillard, und G. Agazzi (1999). MACS-VRPTW: A Multiple Ant Colony System for Vehicle Routing Problems with Time Windows. Technischer Report IDSIA-06-99, Istituto Dalle Molle di Studi sull'Intelligenza Artificiale.

Garey, M. R. und D. S. Johnson (1979). *Computers and Intractability : A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)*. W. H. Freeman.

Gendreau, M., A. Hertz, und G. Laporte (1992). New Insertion and Postoptimization Procedures for the Traveling Salesman Problem. *Operations Research 40*(6), 1086–1094.

Gendreau, M., A. Hertz, und G. Laporte (1994). A tabu search heuristic for the vehicle routing problem. *Management Science 40*(10), 1276–1290.

Gendreau, M., M. Iori, G. Laporte, und S. Martello (2006). A Tabu Search Algorithm for a Routing and Container Loading Problem. *Transportation Science 40*(3), 342–350.

Gendreau, M., M. Iori, G. Laporte, und S. Martello (2008). A Tabu search heuristic for the vehicle routing problem with two-dimensional loading constraints. *Networks 51*(1), 4–18.

Gillett, B. E. und L. R. Miller (1974). A Heuristic Algorithm for the Vehicle Dispatching Problem. *Operations Research 22*(2), 340–349.

Hadjiconstantinou, E. und N. Christofides (1995). An exact algorithm for general, orthogonal, two-dimensional knapsack problems. *European Journal of Operational Research 83*(1), 39–56.

Hansen, P. und N. Mladenovic (2006). First vs. best improvement: An empirical study. *Discrete Applied Mathematics 154*, 802–817.

Ho, S. und D. Haugland (2004). A Tabu Search Heuristic for the Vehicle Routing Problem with Time Windows and Split Deliveries. *Computers & Operations Research 31*(12), 1947–1964.

Homberger, J. und H. Gehring (1999). Two Evolutionary Metaheuristics for the Vehicle Routing Problem with Time Windows. Technischer report, Fernuniversität Hagen, Lehrstuhl Wirtschaftsinformatik.

Homberger, J. und H. Gehring (2004). A two-phase hybrid metaheuristic for the vehicle routing problem with time windows. *European Journal of Operational Research 162*(1), 220–238.

Ioannou, G., M. Kritikos, und G. Prastacos (2001). A greedy look-ahead heuristic for the vehicle routing problem with time windows. *Journal of the Operational Research Society 52*(5), 523–537.

Iori, M., J. S. Gonzalez, und D. Vigo (2007). An exact approach for the vehicle routing problem with two dimensional loading constraints. *Transportation Science 41*(2), 253–264.

Lodi, A., S. Martello, und D. Vigo (1999). Heuristic and Metaheuristic Approaches for a Class of Two-Dimensional Bin Packing Problems. *INFORMS J. on Computing 11*(4), 345–357.

Lodi, A., S. Martello, und D. Vigo (2002). Heuristic algorithms for the three-dimensional bin packing problem. *European Journal of Operational Research 127*(2), 410–420.

Lueker, G. (1976). Manuskript, Department of Computer Science, Princeton University.

Martello, S., D. Pisinger, und D. Vigo (2000). The three-dimensional bin packing problem. *Operations Research 48*(2), 256–267.

Martello, S. und P. Toth (1990). Lower bounds and reduction procedures for the bin packing problem. *Discrete Appl. Math. 28*(1), 59–70.

Martello, S. und D. Vigo (1998). Exact solution of the two-dimensional finite bin packing problem. Technical report, DEIS-OR-96-3.

Mladenovic, N. und P. Hansen (1997). Variable neighborhood search. *Computers and Operations Research 24*, 1097–1100.

Moura, A. und J. Oliveira (2007). An integrated approach to the Vehicle Routing and Container Loading Problems. Technischer report, Universidade de Coimbra, Portugal, Instituto de Engenharia de Sistemase Computadores de Coimbra.

Pisinger, D. und S. Ropke (2007). A general heuristic for vehicle routing problems. *Comput. Oper. Res. 34*(8), 2403–2435.

Reimann, M., K. Doerner, und R. Hartl (2003). *EvoWorkshops 2003*, Chapter Analyzing a Unified Ant System for the VRP and Some of Its Variants, S. 300–310. Lecture Notes in Computer Science. Springer.

Russell, R. (1995). Hybrid Heuristics for The Vehicle Routing Problem with Time Windows. *Transportation Science 29*(2), 156–166.

Schepers, J. (1997). *Exakte Algorithmen fur orthogonale Packungsprobleme.* Doktorarbeit, Universität zu Köln.

Solomon, M. M. (1987). Algorithms for the vehicle routing and scheduling problems with time window constraints. *Oper. Res. 35*(2), 254–265.

Vastag, A., G. Presifilippo, und F. Schwarz (2005). Wirtschaftliche und ökologische Effizienz des Einsatzes von Volumenfahrzeugen in der Transportlogistik am Automobilstandort Deutschland. Studie des Fraunhofer-Institut für Materialfluss und Logistik, Dortmund.

Yue, M. (1991). A simple proof of the inequality $\mathrm{ffd}(l) \leq \frac{11}{9}\mathrm{opt}(l) + 1\ \forall l$, for the FFD bin-packing algorithm. *Acta Math. App. Sinica 7*, 321–331.