TECHNISCHE UNIVERSITÄT WIEN

VIENNA UNIVERSITY OF TECHNOLOGY

DISSERTATION

Combining Metaheuristics and Integer Programming for Solving Cutting and Packing Problems

ausgeführt zum Zwecke der Erlangung des akademischen Grades eines Doktors der technischen Wissenschaften unter der Leitung von

Univ.-Prof. Dr. Günther Raidl
Institut für Computergraphik und Algorithmen E186
Technische Universität Wien

und

ao. Univ.-Prof. Dr. Ulrich Pferschy
Institut für Statistik und Operations Research
Universität Graz

eingereicht an der Technischen Universität Wien
Fakultät für Informatik

von

Jakob Puchinger
Matrikelnummer 9826289
Paulinengasse 9/14/11, 1180 Wien

Wien, am

Jakob Puchinger

Kurzfassung

Thema der vorliegenden Dissertation ist die Kombination von Metaheuristiken und Algorithmen aus der ganzzahligen Programmierung (Integer Programming IP) zum Lösen von zwei verschiedenen $\mathcal{NP}$-schweren Pack- und Verschnittproblemen: Das aus der Glaserzeugung kommende zweidimensionale Bin Packing Problem (2BP) und das mehrdimensionale Rucksackproblem (Multidimensional Knapsack Problem MKP), welches erstmals im Zusammenhang mit Investitionsplanung in der Literatur Erwähnung fand.

Zu Beginn werden Metaheuristiken, im Besonderen evolutionäre Algorithmen und variable Nachbarschaftssuche sowie ganzzahlige Programmierung eingeführt. Metaheuristiken und IP können als komplementär angesehen werden, wenn man die Vor- und Nachteile beider Methoden betrachtet. Daher scheint es umso natürlicher zu sein diese verschiedenen Ansätze zu leistungsfähigeren Lösungstrategien zu vereinen. In der vorliegenden Arbeit werden wir verschiedene moderne Ansätze zur Kombination von exakten Verfahren und Metaheuristiken überblicksmäßig besprechen. Eine Klassifikation der behandelten Ansätze in integrative und kollaborative Kombinationen der oben genannten Verfahren wird vorgestellt. Die Ergebnisse der besprochenen Verfahren sind in den meisten Fällen sehr vielversprechend.

Wir wenden uns dann dem 2BP zu. Es werden neue IP-Formulierungen vorgestellt: Ein Modell für eine eingeschränkte Version und eines für das ursprüngliche Problem. Beide Modelle benötigen nur eine polynomielle Anzahl von Variablen und Nebenbedingungen und können Symmetrien vermeiden. Diese Modelle werden mit dem allgemeinen IP-Löser CPLEX gelöst. Außerdem entwickeln wir einen Branch-and-Price (B&P) Algorithmus der auf eine set-covering Formulierung des Problems angewandt wird. Die Spaltengenerierung wird mittels dual-optimalen Un-

gleichungen stabilisiert. Die Erzeugung der Spalten wird mittels einer aus vier Methoden bestehenden hierarchischen Strategie gelöst: (a) einer schnellen Heuristik, (b) eines evolutionären Algorithmus, (c) des Lösens einer eingeschränkten Version des Pricing-Problems mittels CPLEX und schließlich (d) des Lösens des gesamten Pricing-Problems mittel CPLEX. Die durchgeführten Experimente dokumentieren die Vorteile unseres Ansatzes: Die eingeschränkte Version ermöglicht es rasch hochqualitative Lösungen zu berechnen. Das Lösen des Modells der uneingeschränkten Version benötigt wesentlich mehr Rechenaufwand. Mithilfe der Spaltengenerierung können aber in akzeptabler Zeit sehr gute untere Schranken erreicht werden. Das vorgestellte B&P-Verfahren mit Stabilisierung und der vierstufigen hierarchischen Pricing-Strategie erzielt die besten Ergebnisse in Hinblick auf die benötigten Laufzeiten, die Lösungsgüte und die Anzahl der optimal gelösten Instanzen in limitierter Laufzeit.

Weiters betrachten wir das MKP, dessen Struktur wir theoretisch und empirisch analysieren. Wir stellen auch verschiedene IP-basierte und metaheuristische Verfahren sowie Kombinationen dieser Verfahren vor. Zuerst analysieren wir die Distanzen zwischen optimalen ganzzahligen Lösungen und optimalen Lösungen der LP-Relaxierung des MKP. Dazu benutzen wir die kleineren der üblicherweise verwendeten Benchmark-Instanzen. Danach führen wir das Core-Konzept für das MKP ein und unterziehen es einer ausgiebigen theoretischen und empirischen Analyse. Daraus werden neue, erfolgreiche Ansätze zum Lösen des MKP entwickelt. Wir stellen dann die neue relaxierungsgesteuerte variable Nachbarschaftssuche (Relaxation Guided Variable Neighborhood Search) für allgemeine kombinatorische Optimierungsprobleme und ihre Implemetierung für das MKP vor. Schließlich werden verschiedene kollaborative Kombinationen der vorgestellten Verfahren diskutiert und evaluiert. Die durchgeführten Experimente zeigen, dass unsere Verfahren die besten bisher bekannten Lösungen berechnen können, wobei die Laufzeiten deutlich unter jenen der besten aus der Literatur bekannten Ansätze liegen.

Zusammenfassend lässt sich feststellen, dass die Kombination der Vorteile von IP-basierten Verfahren und Metaheuristiken uns ermöglichen, bessere Problemlösungsstrategien für spezielle kombinatorische Optimierungsprobleme zu entwickeln. Ausserdem sind die Erfolge unserer Ansätze ein weiterer Anhaltspunkt dafür, dass derartige Kombinationen einen vielversprechenden Forschungszweig darstellen.

Abstract

The main topic of this thesis is the combination of metaheuristics and integer programming based algorithms for solving two different cutting and packing problems, belonging to the class of $\mathcal{NP}$-hard combinatorial optimization problems: A problem originating from the glass cutting industry, three-stage two-dimensional bin packing (2BP), and a problem first mentioned in the context of capital budgeting, the multidimensional knapsack problem (MKP).

We begin by introducing metaheuristics, in particular evolutionary algorithms and variable neighborhood search, and integer programming (IP) based methods. IP and metaheuristic approaches each have their particular advantages and disadvantages. In fact, looking at the assets and drawbacks, the approaches can be seen as complementary. It therefore appears to be natural to combine techniques of these two distinct streams into more powerful problem solving strategies. We discuss different state-of-the-art approaches of combining exact algorithms and metaheuristics to solve combinatorial optimization problems. Some of these hybrids mainly aim at providing optimal solutions in shorter time, while others focus primarily on obtaining better heuristic solutions. The two main categories into which we divide the approaches are collaborative versus integrative combinations. We further classify the different techniques in a hierarchical way. As a whole, the work on combinations of exact algorithms and metaheuristics surveyed, documents the usefulness and strong potential of this research direction.

As first application we present an integrative combination for solving three-stage 2BP. New integer linear programming formulations are developed: models for a restricted version and the original version of the problem are given. Both only involve polynomial numbers of variables and constraints and effectively avoid symmetries.

These models are solved using the general-purpose IP-solver CPLEX. Furthermore, a branch-and-price (B&P) algorithm is presented for a set covering formulation of the unrestricted problem. We consider column generation stabilization in the B&P algorithm using dual-optimal inequalities. Fast column generation is performed by applying a hierarchy of four methods: (a) a fast greedy heuristic, (b) an evolutionary algorithm, (c) solving a restricted form of the pricing problem using CPLEX, and finally (d) solving the complete pricing problem using CPLEX. Computational experiments on standard benchmark instances document the benefits of the new approaches: The restricted version of the integer linear programming model can be used to obtain near-optimal solutions quickly. The unrestricted version is computationally more expensive. Column generation provides a strong lower bound for 3-stage 2BP. The combination of all four pricing algorithms and column generation stabilization in the proposed B&P framework yields the best results in terms of the average objective value, the average run-time, and the number of instances solved to proven optimality.

We then study the MKP, present some theoretical and empirical results about its structure, and evaluate different ILP-based, metaheuristic, and collaborative approaches for it. A short introduction to the multidimensional knapsack problem is followed by an empirical analysis of widely used benchmark instances. First the distances between optimal solutions to the LP-relaxation and the original problem are studied. Second we introduce the new core concept for the MKP and conduct an extensive study of it. The empirical analysis is then used to develop new concepts for solving the MKP using ILP-based and memetic algorithms. We then describe the newly developed Relaxation Guided Variable Neighborhood Search in general, and its implementation for the MKP in particular. Different collaborative combinations of the algorithms presented are discussed and evaluated. Further computational experiments with longer run-times are also performed in order to compare the solutions of our approaches to the best known solutions for the MKP. Several of the collaborative and core based approaches were able to reach many of the best known results from the literature in substantially shorter times.

In summary, we can note that using the advantages of IP-based methods and metaheuristics by combining them in different ways improves our problem solving capacities for specific problems. The successful results of our approaches suggest, that these types of combinations point to a promising research direction for the solution of $\mathcal{NP}$-hard combinatorial optimization problems.

Acknowledgments

First of all I want to thank Prof. Günther Raidl, who was a great thesis supervisor, boss and colleague. He introduced me to the world of combinatorial optimization, metaheuristics and integer programming, and gave me irreplaceable guidance and advice. I further want to thank Prof. Ulrich Pferschy, my second supervisor. He gave me valuable advice and hints for all the parts of my thesis, especially for the MKP. It was really nice and enriching to have the opportunity to work with Ulrich.

I also want to say thank you to my colleagues at the Algorithms and Data Structures Group: Gabriele Koller was the co-supervisor of my master thesis and helped me with my first publication. Ivana Ljubic was my roommate at the University until she finished her PhD, and I had many interesting professional and private discussions with her. Martin Schönhacker was an important advisor in teaching and university issues. Martin Gruber was a great co-worker and colleague, and also an important technical help. Philipp Neuner was of invaluable help in technical questions and we had great private and scientific discussions. Bin Hu was my always obliging and supportive new roommate at the University. Matthias Prandtstetter and Daniel Wagner were nice colleagues with whom I had fruitful discussions. Gunnar Klau and René Weiskircher are former colleagues with whom I had interesting scientific discussions and who also gave me support with teaching issues. I also want to thank Prof. Petra Mutzel, the former head of our group, for her support. Many thanks to Stefanie Wogowitsch, who was always of great help with all administrative purposes and with whom I also had great private discussions.

Special thanks to my family, in particular to my parents to whom I owe everything.

Last but not least I want to thank Miriam for her endless support and love.

Contents

Chapter 1

Introduction

Cutting and packing problems are *combinatorial optimization problems* (COPs) usually occurring in a multitude of industrial applications such as

- glass, paper and steel cutting,
- container or pallet loading,
- VLSI design,
- portfolio optimization
- and many others.

Usually items have to be cut from raw material such that the resulting waste should be kept minimal, or the number of bins/containers needed to pack given items has to be minimized. Another kind of packing problem is the so called knapsack problem, where items with an associated profit have to be selected and packed into one or more knapsacks, and the resulting total profit has to be maximized. There are several variants of the knapsack problem, such as the classical knapsack problem, the multidimensional knapsack problem, the multiple knapsack problem, and many others.

Most COPs, and therefore most cutting and packing problems, are, in general, difficult to solve. In theoretical computer science, this is captured by the fact that many such problems are $\mathcal{NP}$-hard [42]. Because of the inherent difficulty and the enormous practical importance of $\mathcal{NP}$-hard COPs, a large number of techniques for solving such problems have been proposed over the last decades.

The available techniques for solving difficult COPs can be roughly classified into two main categories: *exact* and *heuristic* algorithms. Exact algorithms are guaranteed to find an optimal solution for every instance of a COP. The run-time, however, often increases dramatically with the instance size, and in practice only small or medium-sized instances can be solved to proven optimality. In this case, the only possibility for larger instances is to trade optimality for run-time, yielding heuristic algorithms. In other words, the guarantee of finding an optimal solution is sacrificed for the sake of getting a good solution in a limited available time.

Two realizations of these categories have had significant success: *integer (linear) programming* (IP), as an exact approach, and local search with various extensions and independently developed variants, in the following called *metaheuristics* (MH), as a heuristic approach.

Some well known IP methods are branch-and-bound, Lagrangian relaxation based methods, and cutting-plane techniques based on linear programming, such as branch-and-cut, branch-and-price, and branch-and-cut-and-price [92, 132]. In recent years remarkable improvements have been reported for IP when applied to particular problems (see for example [3] for the traveling salesman problem).

The main advantage of IP methods is that they yield proven optimal solutions or solutions with bounds or guarantees on the quality of the objective function. However, for many of the available IP algorithms the size of the instances that can be solved in practice is still relatively small, and the computation time increases in general exponentially with increasing instance size. Often, memory requirements are also a limiting factor to applicability.

Metaheuristic methods include, among others, simulated annealing, tabu search, variable neighborhood search and various population-based models such as evolutionary algorithms and memetic algorithms. See [1, 49, 62] for more general introductions to local search based approaches and metaheuristics. These methods have shown significant success in achieving near-optimal (and sometimes optimal) solutions to difficult practical COPs.

Specific advantages of metaheuristics are that in many cases they are found to be the best performing algorithms for large problems in practice, they can examine a large number of possible solutions in a relatively short computation time, and they are typically easier to understand and implement than advanced exact methods. However, disadvantages of metaheuristics are that they cannot prove optimality, they do not give tight quality guarantees for approximate solutions, and they cannot provably reduce the search space.

IP and metaheuristic approaches each have their particular advantages and disadvantages. In fact, looking at the assets and drawbacks, the approaches can be seen

as complementary. It therefore appears to be natural to combine the techniques of these two distinct streams into more powerful problem solving strategies.

Apart from obvious combinations such as applying local search based heuristics to get tighter bounds for e.g. branch-and-bound algorithms and approaches in the context of preprocessing, combining exact optimization techniques and metaheuristics is still an underrepresented research area. In Chapter 4 we give a survey and classification of different combination approaches, which we mainly classify into *integrative* and *cooperative* combinations.

The main subject of this thesis is the combination of exact and metaheuristic algorithms for solving cutting and packing problems. Looking at their assets and drawbacks, the approaches can be seen as complementary. It appears to be natural to combine ideas from both streams. In the last years there have been several hybridization approaches that are often significantly more effective in terms of running time and/or solution quality since they benefit from synergy. On the one hand, we developed an integrative combination applied to two-dimensional bin packing, in which the property of exact algorithms of being able both to find the global optimum and to give quality guarantees for feasible solutions identified during optimization was retained. On the other hand, we developed a cooperative framework for combining exact algorithms and metaheuristics, which was tested mainly on the multidimensional knapsack problem.

We will now formally introduce combinatorial optimization problems and further describe the two problem classes, bin packing and knapsack problems, addressed throughout this thesis.

Combinatorial Optimization Problems

Combinatorial Optimization Problems are optimization problems where an optimum object is sought from a finite collection of objects. A formal definition of COPs as we will treat them here is given in [1]:

Definition 1 *A combinatorial optimization problem is specified by a set of problem instances and is either a minimization or a maximization problem.*

Definition 2 *An instance of a combinatorial optimization problem is a pair $(\mathcal{S}, f)$, where the solution set $\mathcal{S}$ is the set of feasible solutions and the cost/profit function f is a mapping $f : \mathcal{S} \to \mathbb{R}$. The problem is to find a globally optimal solution, i.e. an $x^* \in \mathcal{S}$ such that $f(x^*) \leq f(x)\ \forall x \in \mathcal{S}$ in case of a minimization problem and $f(x^*) \geq f(x)\ \forall x \in \mathcal{S}$ in case of a maximization problem. Furthermore, $f^* = f(x^*)$*

denotes the optimal cost/profit, and $\mathcal{S}^* = \{x \in \mathcal{S} \mid f(x) = f^*\}$ *denotes the set of optimal solutions.*

Difficult to solve COPs are said to be $\mathcal{NP}$-hard. It is assumed that they cannot be solved by algorithms having their run-times bounded by a polynomial in the size of the input. We give a formal definition of $\mathcal{NP}$-hardness below. For a detailed treatment of computational complexity we refer to [42].

A decision problem P_d is a problem that takes an input (an instance of the problem) and requires as output either 0 or 1 (yes or no). If there exists an algorithm capable to produce the correct answer for any input of length n in a polynomially bounded number of steps, P_d is said to be solvable in polynomial time. $\mathcal{P}$ denotes the class of all decision problems for which there exists a polynomial-time algorithm. The class of problems for which answers can be verified by an algorithm, with run time polynomial in the size of the input is denoted $\mathcal{NP}$. A decision problem $P_d \in \mathcal{NP}$ is said to be $\mathcal{NP}$-complete, if it is possible to polynomially transform every problem from $\mathcal{NP}$ to P_d. A decision problem is called $\mathcal{NP}$-hard, if it is at least as hard as every problem in $\mathcal{NP}$. An optimization problem is therefore already $\mathcal{NP}$-hard if its underlying decision problem (deciding whether the optimization problem has a solution with objective better than a given constant) is $\mathcal{NP}$-complete. In this thesis we consider $\mathcal{NP}$-hard COPs only, for an in-depth comprehensive overview on efficiently solvable COPs we refer to [111].

Bin Packing Problems

Bin packing problems occur in applications such as glass manufacturing, container and vehicle loading, or scheduling. The main differences occurring in different bin packing applications are the dimension and the type of allowed packing patterns. Many exact and heuristic approaches have been developed in the last decades for solving different variants of bin-packing problems. An annotated bibliography on cutting and packing problems is given in [31]. A recent survey on two-dimensional bin-packing is given in [75].

In one-dimensional bin packing, n items with given lengths l_i $i = 1, \cdots, n$ have to packed without overlapping into bins with length L. The goal is to pack all the items while minimizing the total number of bins used. This principle also applies for two-dimensional bin packing where n two-dimensional items with given widths w_i and heights h_i $i = 1, \cdots, n$ have to be packed into a minimal number of bins having width W and height H.

Knapsack Problems

Knapsack Problems also occur in a number of important real-world applications. Many very different algorithmic approaches have been developed for the different variants of the knapsack problem. A comprehensive treatment of knapsack problems is given in [66].

In the classical 0/1-knapsack problem a set of n items with associated profits p_j and weights $w_j \quad j = 1, \cdots, n$ is given. A subset has to be selected and packed into a knapsack having a maximum weight capacity c. The total profit of the items in the knapsack has to be maximized. In this thesis we will tackle the multidimensional knapsack problem where n items are given and each item has an associated profit $p_j \; j = 1, \cdots, n$. We are further given m resources with limited amounts $c_i \; i = 1, \cdots, m$ Each of the items consumes w_{ij} of each resource. The objective is to maximize the profit while not violating the resource constraints.

Overview of the thesis

First we will give a short introduction to metaheuristics and IP-based algorithms in Chapters 2 and 3. Our main goal is to enable the reader to become familiar with the terms and notations used throughout this thesis.

In Chapter 4 we discuss different state-of-the-art approaches of combining exact algorithms and metaheuristics to solve combinatorial optimization problems. Some of these hybrids mainly aim at providing optimal solutions in shorter time, while others focus primarily on getting better heuristic solutions. The two main categories into which we divide the approaches are collaborative versus integrative combinations. We further classify the different techniques in a hierarchical way. As a whole the surveyed work on combinations of exact algorithms and metaheuristics documents the usefulness and strong potential of this research direction. An earlier version of this chapter was published in:

> Jakob Puchinger and Günther R. Raidl. Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification. In *Proceedings of the First International Work-Conference on the Interplay Between Natural and Artificial Computation*, volume 3562 of LNCS, pages 41-53. Springer, 2005.

In Chapter 5 an integrative combination for solving a two-dimensional bin packing problem is presented.

We consider the three-stage two-dimensional bin packing problem (2BP) which occurs in real-world applications such as glass, paper, or steel cutting. We present new integer linear programming formulations: models for a restricted version and the original version of the problem are developed. Both only involve polynomial numbers of variables and constraints and effectively avoid symmetries. Those models are solved using the general-purpose IP-solver CPLEX.

Furthermore, a branch-and-price (B&P) algorithm is presented for a set covering formulation of the unrestricted problem. We consider column generation stabilization in the B&P algorithm using dual-optimal inequalities. Fast column generation is performed by applying a hierarchy of four methods: (a) a fast greedy heuristic, (b) an evolutionary algorithm, (c) solving a restricted form of the pricing problem using CPLEX, and finally (d) solving the complete pricing problem using CPLEX. Computational experiments on standard benchmark instances document the benefits of the new approaches: The restricted version of the integer linear programming model can be used to quickly obtain near-optimal solutions quickly. The unrestricted version is computationally more expensive. Column generation provides a strong lower bound for 3-stage 2BP.

The combination of all four pricing algorithms and column generation stabilization in the proposed B&P framework yields the best results in terms of the average objective value, the average run-time, and the number of instances solved to proven optimality.

An earlier version of this chapter appeared in:

> Jakob Puchinger and Günther R. Raidl. Models and algorithms for three-stage two-dimensional bin packing. *European Journal of Operational Research*, feature issue on Cutting and Packing. Accepted for publication 2005.

Preliminary results of the presented algorithms have been published by the authors in:

> Jakob Puchinger and Günther R. Raidl. An evolutionary algorithm for column generation in integer programming: an effective approach for 2D bin packing. In X. Yao et. al, editor, *Parallel Problem Solving from Nature - PPSN VIII*, volume 3242 of *LNCS*, pages 642-651. Springer, 2004.

In Chapter 6, we study the multidimensional knapsack problem, present some theoretical and empirical results about its structure, and evaluate different ILP-based, metaheuristic and collaborative approaches for it.

First, we give a short introduction to the multidimensional knapsack problem, followed by an empirical analysis of widely used benchmark instances. First the distances between optimal solutions to the LP-relaxation and the original problem are studied. Second we introduce the new core concept for the MKP, which we then study extensively. The empirical analysis is then used to develop new concepts for solving the MKP using ILP-based and memetic algorithms. We then describe the newly developed Relaxation Guided Variable Neighborhood Search in general, and its implementation for the MKP. Different collaborative combinations of the algorithms presented are discussed and evaluated. Further computational experiments with longer run-times are also performed in order to compare the solutions of our approaches to the best known solutions for the MKP. Several of the collaborative and core based approaches were able to reach many of the best known results from the literature in substantially shorter times.

Earlier versions of parts of this chapter have been published in:

> Jakob Puchinger, Günther R. Raidl, and Martin Gruber. Cooperating memetic and branch-and-cut algorithms for solving the multidimensional knapsack problem. In *Proceedings of MIC2005, the 6th Metaheuristics International Conference*, pages 775-780, Vienna, Austria, 2005.
>
> Jakob Puchinger and Günther R. Raidl. Relaxation guided variable neighborhood search. In *Proceedings of the XVIII Mini EURO Conference on VNS*, Tenerife, Spain, 2005.
>
> Jakob Puchinger, Günther R. Raidl, and Ulrich Pferschy. The Core Concept for the Multidimensional Knapsack Problem. To appear in *Evolutionary Computation in Combinatorial Optimization - EvoCOP 2006*, Budapest, Hungary. LNCS, Springer 2006.

We conclude this thesis in Chapter 7 by summarizing the work presented and giving a look ahead to future tasks and development.

Chapter 2

Metaheuristics

Local search based *metaheuristics* have proven to be highly useful in practice. This category of problem solving techniques includes, among others, simulated annealing [68], tabu search [51], iterated local search [79], variable neighborhood search [57], various population-based models such as evolutionary algorithms [4], scatter search [52] and memetic algorithms [90], and estimation of distribution algorithms such as ant colony optimization [29]. See also [1, 50, 62] for more general introductions to local search and metaheuristics.

In this chapter we will give an introduction to evolutionary algorithms and variable neighborhood search, which are the metaheuristics mainly used and referred to throughout this thesis. Before describing these metaheuristics in detail, we will introduce some basic notions of heuristics and local search. In the remainder of this chapter we will consider minimization problems only, since maximization problems can be treated analogously.

2.1. Constructive Heuristics and Local Search

Constructive heuristics are able to find feasible solutions to COPs without guaranteeing their optimality. Having a potentially suboptimal solution, one could try to improve it. Local Search, a basis of most metaheuristics, is a possible way for achieving this goal. In the following we will introduce these basic concepts in detail.

2.1.1. Constructive Heuristics

Constructive heuristics usually build a solution from scratch incrementally. A common scheme is to start from an "empty" solution and add an atomic solution component at each step. Constructive heuristics always enlarge partial solutions and never reconsider the choices made during the construction process.

Considering the classical 0/1 single-knapsack problem as an example. A simple constructive heuristic would try to add one item after the other until no item can be added anymore. If the locally best item, according to its profit or any other measure, is chosen, the heuristic is said to be *greedy*.

Performance guarantees can often be provided for constructive heuristics. More generally, the theoretical framework of approximation algorithms [129] permits absolute and relative performance guarantees of heuristics to be given. Furthermore it allows the inherent algorithmic difficulty of specific COPs to be analyzed, by providing positive or negative approximability results.

2.1.2. Local Search

Local search [1] is a simple way for improving feasible solutions of a COP. Having a solution, local search iteratively looks for better neighboring solutions and stops if no such solutions can be found.

The main component of local search is therefore the neighborhood structure. Furthermore, a definition of the search space, i.e. a representation of solutions x, and an objective function $f(x)$ associating an objective value to each feasible solution are needed to define a local search.

Definition 3 *A neighborhood structure $\mathcal{N} : \mathcal{S} \rightarrow 2^{\mathcal{S}}$ is a function associating a set of neighbors, called neighborhood $\mathcal{N}(x) \subseteq \mathcal{S}$ to every solution $x \in \mathcal{S}$.*

Actual neighborhood structures are often defined by a specific *move*. A move is an operation applied to a solution x yielding another solution x'. Considering a solution represented by a bit-string of a given length n, a move could be a flip of a single bit. The resulting neighborhood of a given solution is then the set of solutions which can be generated by flipping a single bit of x.

The pseudocode description of basic local search is given in Algorithm 1.

Algorithm 1: Basic local search

$x =$ start solution

repeat
  choose $x' \in \mathcal{N}(x)$
  if $f(x') \leq f(x)$ **then**
    $x = x'$

until *termination condition*

The given pseudocode leaves the choice of neighbor x' open. Usually, one of the following strategies for choosing x', called *step-functions*, is used:

- Random Neighbor: Randomly pick a solution from $\mathcal{N}(x)$.
- Next Improvement: Search through $\mathcal{N}(x)$ in some order and use the first solution at least as good as x
- Best Improvement: Use a best neighbor solution from $\mathcal{N}(x)$

If basic local search is applied with the next or best improvement strategies and no further improvements are possible, a local optimum with respect to $\mathcal{N}$ is reached:

Definition 4 *A solution $\hat{x}$ of a minimization problem is locally optimal with respect to a neighborhood structure $\mathcal{N}$ if*

$$f(\hat{x}) \leq f(x) \forall x \in \mathcal{N}(\hat{x}).$$

The success of local search strongly depends on the choice of the starting solution, the neighborhood structure and the step function. Local search is a very simple and often successful algorithmic framework. There are many problems (e.g. linear programming) where local and global optima coincide which can be solved to optimality by local search approaches. In $\mathcal{NP}$-hard COPs local and global optima do not coincide, i.e. there are several local optima different from the global optima. In order to overcome this drawback of simple local search, several extensions and variants such as multi-start and iterated local search have been developed. Furthermore, algorithms such as tabu search, variable neighborhood search, memetic algorithms and many others incorporate the principles of local search.

2.2. Evolutionary and Memetic Algorithms

The biologically inspired field of evolutionary algorithms has its origin in Charles Darwin's theory of evolution [22]. As early as 1948 Turing proposed a "genetical or evolutionary search", see [123]. Three different implementations of the basic idea were developed in the 1960s; evolutionary programming by Fogel, Owens and Walsh [36], the genetic algorithm by Holland[61], and evolution strategies by Rechenberg and Schwefel [110, 112]. Over the last decades these strategies have been extended, adapted, and combined in various ways and are now mainly referred to under the umbrella term evolutionary algorithms [4]. For a more detailed historic overview and a comprehensive treatment of evolutionary computation we refer to [4, 33].

2.2.1. Principles

The term evolutionary algorithm (EA) represents a much wider class of algorithms than the original class of genetic algorithms, since in its general form almost all aspects can be freely defined and adapted to the problem at hand. In Algorithm 2 pseudocode for a generic evolutionary algorithm is given.

Algorithm 2: Generic evolutionary algorithm

$P \leftarrow$ set of initial solutions

Evaluate(P)

repeat

  $Q \leftarrow$ SelectSolutionsForVariation(P)

  $Q \leftarrow$ GenerateNewSolutionsByVariation(Q)

  Evaluate(Q)

  $P \leftarrow$ SelectSurvivingSolutions(P,Q)

until *termination condition*

The depicted EA acts on a set of, usually different, solutions P also called *population*. P is usually initialized by random constructive heuristics. The measure for the number of different solutions in a population is called its *diversity*. Note that there is no single measure for diversity, for example either the number of individuals with different fitness values or the number of really different individuals could be used.

P is then evaluated for the first time, i.e. the *fitness* of each candidate solution or *individual* or *chromosome* is calculated. The fitness of an individual is a measure

for its quality and is closely linked to the objective value of the solution it represents. The population then undergoes the artificial evolutionary process until a termination condition is met. First a certain number of individuals is selected to undergo variation. Then the biologically inspired *variation operators*, *recombination* and *mutation*, are applied to these candidate solutions, generating new individuals, which are again evaluated. Finally a second selection takes place to decide which of the newly created individuals are incorporated into the population, and which of the old individuals are replaced.

In the remainder of this section we will introduce the different aspects of modern evolutionary algorithms in more detail.

Representation

When designing evolutionary algorithms, one has to decide about the representation (*genotype*) of solutions (*phenotypes*). A genotype can consist of a binary string, but also of a permutation encoding, real valued numbers, or any other possibility. Furthermore a representation can be *direct* if the genotype directly corresponds to the phenotype, and *indirect* if an algorithm has to be applied to the genotype in order to decode the phenotype.

Consider the knapsack problem. On the one hand solutions can be directly represented by a bit string corresponding to solution vector x, where each bit represents the status (packed or not) of an item. On the other hand a permutation of the items can represent a candidate solution, if it is decoded by a first-fit heuristic, which packs the items into the knapsack according to the order given by the genotype, may be used to represent a solution [60].

Evaluation

The evaluation function $F(x)$ is needed to calculate the fitness of each individual x. The objective value can be directly used as fitness value, but it can often be advantageous to use a scaled objective value. In case of a constrained problem, if infeasible individuals appear in the population, a penalty term can be introduced in order to control infeasibility.

Selection Mechanisms

Parent Selection The selection of the parents that undergo variation is typically done in a probabilistic way, but giving fitter individuals a higher chance to be chosen

than those with lower quality. Frequently used selection mechanisms are fitness proportional [61], rank-based [5], and tournament [33] selection.

In *fitness-proportional* or *roulette wheel* selection, the probability of selecting an individual x from a population P is given by:

$$p(x) = \frac{F(x)}{\sum_{y \in P} F(y)}.$$

If the objective value is chosen as fitness, this type of selection can lead to premature convergence, i.e. the effect of losing population diversity too quickly and getting trapped in a local optimum, since outstanding individuals have too high selection probabilities. On the other side if the fitness values of the different individuals are too similar to one another, better individuals are not significantly preferred and the evolutionary algorithm tends to perform not better than random search. How better individuals are preferred to other ones is generally expressed by the term *selection-pressure*, for which different specific definitions exist in the literature. Using a scaled objective function as fitness value can overcome those problems. It is also used for solving minimization problems, or problems with possibly negative objective function values. Linear scaling is a linear transformation of the objective value (a and b are constant parameters):

$$F(x) = af(x) + b.$$

In order to avoid negative outliers one can use sigma-scaling [54], where information about the mean ($\overline{f}$) and the standard deviation (σ) of objective values in the population is incorporated (c is a constant parameter):

$$F(x) = \begin{cases} f(x) - (\overline{f} - c\sigma) & \text{if } f(x) > \overline{f} - c\sigma \\ 0 & \text{else} \end{cases}$$

Furthermore, other selection types were developed. In *rank-based* selection mechanisms, individuals are sorted according to their fitness values and their probability of being selected is then calculated depending on their rank. The association of rank-number and selection probability can be done in various ways, such as linearly or exponentially decreasing.

For both of the selection strategies presented here, knowledge of the whole population is needed and it has to be possible to evaluate each chromosome individually, which is, for example, not possible in a game-theoretic context. In *tournament* selection, k individuals are randomly chosen from the population and the best is selected from them. The parameter k allows selection pressure to be controlled, since the larger k is, the higher the probability of selecting above-average individuals becomes. Tournament selection can be implemented very efficiently and is therefore a widely used selection strategy in modern EAs [33].

Survivor Selection – Replacement Strategy The replacement strategy depends on the chosen population model. The classical model, coming from the genetic algorithm, is the generational EA. The model we mainly use for the EAs presented in this thesis is the steady-state EA.

Generational EA Every generation begins with a population of size μ, from which μ parents (if the recombination generates 2 offspring individuals) are selected for recombination and mutation. Variation operators are applied to the selected individuals, generating λ ($= \mu$) new individuals, the offspring, who replace the entire old generation. The drawback of this classical model, is that the entire old population is thrown away, and potentially very high quality solutions are lost.

Steady-State EA Here, the whole population is not changed at once, but only a part of it [131]. λ ($< \mu$) old individuals are replaced by new ones, the offspring. Usually a single individual ($\lambda = 1$) is created per iteration and it replaces the worst of the population. Furthermore one can ensure that no duplicate exist in the population, increasing the populations diversity. One of the main advantages of this model, is that big population sizes can be handled more easily, and that at each generation only a single individual needs to be evaluated. The steady-state EA exhibits faster convergence than the generational EA, which can be a drawback, but this can be countered with strategies such as duplicate-elimination, where individuals with the same genotype are not allowed.

Variation Operators

The main variation operators are recombination (also called crossover), which derives new solutions from the attributes of two parents, and mutation, where small changes are applied to a single individual.

Recombination Recombination is a variation operator, combining attributes of two parent individuals into one or several new offspring solutions. The one-point crossover operator, usually used with a bit-string representation, cuts two parental chromosomes in two parts at a randomly determined crossover point. Then it recombines the first part of the first chromosome with the second part of the second chromosome. This crossover type can be generalized to two point and multipoint crossover in a natural way; see Figure 2.1.

In the uniform crossover operator, it is decided randomly for every gene whether it is taken from the first or the second parent. Partially Mapped Crossover [55],

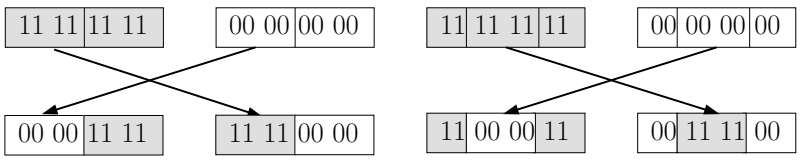

Figure 2.1.: One point and two point crossover.

Order Crossover [23], and Uniform Order Based Crossover [117] are examples of permutation based representation operators.

Mutation The mutation operator is a variation operator usually performing a small change on a single individual. A typical mutation operator is the bit-flip mutation for binary-string representation: Each bit of a binary-string with length l is flipped with a certain small probability p_m, where p_m could be chosen, for example, as $1/l$ with l. In case of a permutation based representation one could use a simple swap of two genes as a mutation operator.

2.2.2. Memetic Algorithms

Many successful EAs for combinatorial optimization use hybridization, where some problem specific knowledge or algorithm is introduced into the EA, permitting it to achieve better solutions in shorter running-times than simple EAs. In 1989 Pablo Moscato [89] introduced the term memetic algorithm (MA) for local-search and problem-specific knowledge enhanced EAs.

Dawkins [24] presented the term meme as a unit of imitation in cultural transmission. Using martial-arts as an example, Moscato writes in [89]:

> In the case of martial arts, those undecomposable movements (...) should be considered as memes. A defensive movement is composed by the coordinated action of many of these memes.

And about the main concept of MAs:

> While Genetic Algorithms have been inspired in trying to emulate biological evolution, Memetic Algorithms (MA) would try to mimic cultural evolution. (...) Memetic algorithms are a marriage between a population-based global search and the heuristic local search made by each of the individuals.

Problem specific knowledge can and should be used at any point of the algorithm, for example at the initialization, inside the variation operators, at postprocessing the results of the variation operators, or in genotype to phenotype mapping.

2.3. Variable Neighborhood Search

Variable neighborhood search (VNS) is a relatively new and successful metaheuristic framework, developed by Hansen and Mladenovic [58, 59]. It systematically exploits the idea of neighborhood change. On the one hand better local optima are found since several neighborhood structures are searched. On the other hand valleys containing these local optima are escaped by exploring parts of the search space chosen randomly.

Hansen and Mladenovic present the following observations for motivating their algorithmic ideas:

- A local optimum with respect to one neighborhood structure is not necessarily so for another.
- A global optimum is a local optimum with respect to all possible neighborhood structures.
- For many problems local optima with respect to one or several neighborhoods are relatively close to each other.

The last, empirical observation implies that information about global optima can often be gathered by using the local ones.

2.3.1. Variable Neighborhood Descent

In Variable Neighborhood Descent (VND) the first of the above observations is exploited. A kind of local search using several systematically changing neighborhoods $N_1, \ldots, N_{k_{\max}}$ is performed. The pseudocode for VND is given in Algorithm 3.

In [59] the following questions to be taken into consideration arise:

1. What is the time-complexity of searching the different neighborhoods?
2. What is the best order in applying them?
3. Are the neighborhoods considered sufficient to ensure a thorough exploration of the search space containing x?

Algorithm 3: VND(x)

Input: x denotes the initial solution
Given neighborhoods N_k, for $k = 1, \ldots, k_{\max}$

$k \leftarrow 1$
repeat
  find $x' \in N_k$ with $f(x') < f(y),\ \forall y \in N_k(x)$
  if $f(x') < f(x)$ **then**
    $x \leftarrow x'$
    $k \leftarrow 1$
  else
    $k \leftarrow k + 1$
until $k = k_{\max}$
return x

Usually neighborhoods are sorted according to increasing complexity of searching for the best moves, which often, but not always, corresponds to the size of neighborhoods. In Chapter 6 we will present a new dynamic approach, in which the neighborhoods are ordered according to an estimated improvement potential.

The third question points to the fact that larger neighborhoods or some escape mechanisms have to be available in order to obtain good results for difficult problems. This question leads us to the VNS metaheuristic.

2.3.2. Variable Neighborhood Search

Variable Neighborhood Search (VNS) can extend the previously described VND algorithm and is able to escape local optima. Furthermore it is a very simple but general algorithmic framework permitting to incorporate any local search algorithm.

A set of different potentially nested neighborhoods $\mathcal{N}_1, \ldots, \mathcal{N}_{l_{\max}}$ of increasing size is considered here. Based on the observation that local optima often lie near to both each other and the global optima, random steps are performed in those neighborhoods in order to escape valleys or hills in the optimization landscape. These random moves called *shaking* are performed in systematically changing neighborhoods. At the beginning shaking creates solutions near to the current local optimum. Later, when larger neighborhoods are considered as the search goes on, more varied solutions are devised. A more formal description is given in Algorithm 4, where we present the *Basic Variable Neighborhood Search*. In *General Variable Neighborhood Search*, VND is used instead of the basic local search, in order to improve solutions after shaking.

Algorithm 4: Basic VNS(x)

Input: x denotes the initial solution
Given neighborhoods $\mathcal{N}_l$, for $l = 1, \ldots, l_{\max}$

```
repeat
    u ← 1
    repeat
        x ← Shake(l, x), i.e. choose random solutions from N_l(x)
        x ← basicLocalSearch(x)
        if f(x') < f(x) then
            x ← x'
            l ← 1
        else
            l ← u + 1
    until l = l_max
until stopping conditions are met
return x
```

Chapter 3

Exact Algorithms – Integer Programming

In this chapter we will give a short introduction to integer linear programming and the algorithms emerging from this field of optimization. It is intended to provide an overview, with references to more in-depth material for the interested reader, and a notational and conceptional basis for the remainder of this thesis. This chapter is based on the books on linear optimization by Bertsimas and Tsitsiklis [10], and on combinatorial and integer optimization of Nemhauser and Wolsey [92] and Wolsey [132].

First linear and integer programming will be introduced, then algorithmic concepts for solving integer programs will be presented.

3.1. Linear and Integer Programming

Many optimization problems, especially combinatorial optimization problems, can be formulated as an Integer (Linear) Program (IP / ILP). The ideas and theory of integer programming are based on the concepts of general linear programming, where the decision variables do not have to be integral. Modern linear programming emerged from a research project undertaken by the US Air Force to coordinate troop supplies more effectively. In 1947 George Dantzig, a member of the project, proposed the simplex method for efficiently solving linear programs (LP) [21].

3.1.1. Linear Programs

The linear programming problem can be stated as:

$$z_{LP} = \min\{cx \mid Ax \geq b, x \in \mathbb{R}^n_+\} \tag{3.1}$$

where the data are rational and are given by the n-dimensional row vector c, defining the *objective function* cx, the $m \times n$ matrix A, and the m-dimensional column vector b defining the *constraints*.

It should be noted that equality constraints can be given by two inequalities, and that a minimization problem can be transformed into a maximization by simply multiplying the objective by -1. In the remainder of this chapter, for the sake of simplicity, we consider minimization problems only.

The LP

$$z_{LP} = \min\{cx \mid Ax = b, x \in \mathbb{R}^n_+\} \tag{3.2}$$

is said to be in *standard form*.

Duality

One of the main theoretical concepts of linear programming is duality, which deals with pairs of linear programs and the relationship between their solutions. This concept is also very important for more advanced topics such as primal/dual or column generation algorithms.

The *primal* problem is stated as in (3.1). Its *dual* is then defined as the LP:

$$w_{LP} = \max\{ub \mid uA \leq c, u \in \mathbb{R}^m_+\} \tag{3.3}$$

In the following we provide some major LP-duality results.

Proposition 1 *The dual of the dual problem is the primal problem.*

Proposition 2 (Weak Duality) *If x is primal feasible and u is dual feasible, then*

$$cx \leq ub.$$

The following theorem is a fundamental result of LP duality. It states that if the primal and the dual are both feasible, their optimal values are equal.

Theorem 1 (Strong Duality) *If z_{LP} or w_{LP} is finite, then both (3.1) and (3.3) have the same finite optimal value*

$$z_{LP} = w_{LP}.$$

Another important relation between the primal and the dual optimal solutions is given by the so called *complementary slackness* conditions.

Proposition 3 (Complementary slackness) *If x^* and u^* are feasible solutions for the primal (3.1) and the dual (3.3) problem respectively, then x^* and u^* are optimal solutions if and only if*

$$\begin{aligned} u_i(b - Ax)_i &= 0, \quad \forall i, \\ x_j(uA - c)_j &= 0, \quad \forall j. \end{aligned}$$

3.1.2. Integer Programs

If we add the restriction that all variables have to be integral to the Linear Program 3.1, we get an Integer (Linear) Program (IP),

$$z_{IP} = \min\{cx \mid x \in X\} \tag{3.4}$$

with $X = P \cap \mathbb{Z}^n$ and $P = \{x \mid Ax \geq b, x \in \mathbb{R}^n\}$, which can also be written as

$$\begin{aligned} \min \quad & cx \\ \text{subject to} \quad & Ax \geq b \\ & x \,\text{integer} \end{aligned} \tag{3.5}$$

and if all variables are restricted to 0/1 values, we have a 0/1 or Binary Integer Program.

3.1.3. Geometric interpretation and the simplex algorithm

In this section some important geometrical aspects of linear and integer programming are described. Firstly we will give some important definitions, secondly we will present the simplex algorithm and finally we will give some results of great importance for solving IPs.

Definition 5 *A* polyhedron *is a set that can be described in the form*

$$P = \{x \in \mathbb{R}^n \mid Ax \geq b\} \tag{3.6}$$

where $A \in \mathbb{R}^{m \times n}$ *and* $b \in \mathbb{R}^n$.

The feasible set of any linear programming problem is therefore a polyhedron. In our case, P corresponds to the feasible set of (3.1).

Definition 6 *A polyhedron* $P \subset \mathbb{R}^n$ *is* bounded *if there exists a constant* k *such that* $|x_i| < k \forall x \in P, i = 1, \ldots, n$. *Such a polyhedron is called a* polytope.

Definition 7 *A set* $S \subset \mathbb{R}^n$ *is* convex *if we have* $\lambda x + (1 - \lambda)y \in S,\ \forall x, y \in S,\ \lambda \in [0, 1]$.

Definition 8 *Given* $X = \{x^1, \ldots, x^k\}$, *with* $x^i \in \mathbb{R}^n,\ \lambda_i \geq 0,\ i = 1, \ldots, k$ *and* $\sum_{i=1}^{k} \lambda_i = 1$. *Then*

(i) the vector $\sum_{i=1}^{k} \lambda_i x^i$ *is called a* convex combination *of* X;

(ii) the convex hull *of* X *(*$conv(X)$*) is the set of all convex combinations of* X.

Note that all polyhedra are convex.

Definition 9 *Consider a polyhedron* P *defined by linear equality and inequality constraints, and let* $x^* \in \mathbb{R}^n$.

(a) The vector x^* *is a* basic solution *if:*

(i) All equality constraints are satisfied;

(ii) There are n linearly independent constraints that are active (i.e. that hold with equality) at x^*.

(b) If x^* is a basic solution that satisfies all of the constraints, it is called a basic feasible solution.

One of the most important properties of LP is that basic feasible solutions correspond to the vertices of the LP's polyhedron.

Theorem 2 *Let P be a nonempty polyhedron and let $x \in P$.*
Then the following are equivalent:

(a) x is a vertex;

(b) x is a basic feasible solution.

A fundamental theorem of LP establishes the existence of basic feasible solutions. This theorem, together with the previous one are the basis for the simplex algorithm, sketched in the next Section.

Theorem 3 *Given the linear programming problem (3.1) the following is true:*

(1) If the polyhedron P in (3.6) is nonempty, there exists a basic feasible solution.

(2) If (3.1) has an optimal solution, then there is an optimal basic feasible solution.

The simplex algorithm

The simplex algorithm is the best known and a very efficient LP algorithm. Its main idea consists of moving from one vertex of the polyhedron (basic feasible solution) to an adjacent one, improving the solution at each step. The simplex method consists of two phases. In the first phase an initial basic feasible solution is computed, and in the second phase the solution is iteratively improved following adjacent edges. The method terminates if none of the adjacent edges improves the objective function, or if an unbounded adjacent edge is found.

Given a LP in standard form, a basis of it is defined by any set of m linear independent column vectors of the constraint matrix A. The equation system $Ax = b$ can then be solved (with e.g. the Gaussian elimination method), yielding a feasible solution. Let x be a basic feasible solution to the standard form, $B(1), \ldots, B(m)$ be the indices of the basic variables and $B = [A_{B(1)} \ldots A_{B(m)}]$ be the corresponding basic matrix.

The move from one basic feasible solution to the next one is also called pivoting, since a new column (and therefore a new variable) enters the basis and another one has to leave the basis.

Definition 10 *Let x be a basic solution, B be an associated basis matrix, and c_B be the vector of costs of the basic variables. For each j, we define the* reduced cost $\overline{c_j}$ *of the variable x_j according to*

$$\overline{c}_j = c_j - c_B B^{-1} A_j. \tag{3.7}$$

Since the reduced cost $\overline{c_j}$ corresponds to the per unit cost change for variable x_j, by only bringing variables with negative reduced costs (in case of a minimization problem) into the basis, basic feasible solutions will be improved. If no such variables exist anymore, the solution at hand is optimal.

A thorough description of the simplex method can be found, for example, in [10]. One problem of the simplex algorithm is that it has an exponential worst-case runtime, although in practice it is a very competitive algorithm.

Nevertheless the LP problem is solvable in polynomial-time, which was first shown by Khachiyan [67] in 1979 using the so-called *ellipsoid-method*. Other polynomial-time algorithms of more practical interest are the *interior-point* methods, introduced by Karmakar [65] in 1984. A great number of different highly-competitive algorithms emerged from this starting point. Most of the modern commercial LP-solvers incorporate interior-points method such as barrier or primal-dual algorithms.

Minimal Description of Polyhedra

Of great importance for solving an IP is having a minimal description of its underlying polyhedron. We begin by giving some basic results.

Proposition 4 *Let $X \subseteq \mathbb{R}^n$, then*

(i) $conv(X)$ is a polyhedron and

(ii) the extreme points of $conv(X)$ all lie in X.

With these two results, the IP $\min\{cx \mid x \in X\}$ can be represented by the equivalent LP $\min\{cx \mid x \in conv(X)\}$. Using this reduction we could apply an LP algorithm to solve this IP. However, in most cases, the number of inequalities needed to describe $conv(X)$ is exponential.

In order to search for minimal descriptions of polyhedra, we need the following theoretical basis.

Definition 11 *A polyhedron $P \in \mathbb{R}^n$ is of* dimension k *($dim(P) = k$) if the maximum number of affinely independent points in P is $k+1$. P is said to be* full-dimensional *if $dim(P) = n$.*

Definition 12 *Let P be a polyhedron where $(A^=, b^=)$ designates the rows $\{a_i \in A \mid a_i x = b_i \; \forall x \in P\}$ and $(A^\leq, b^\leq)$ the other rows.*

We now come to a basic concept in describing polyhedra which is of great importance for the development of algorithms for solving IPs.

Definition 13 *An inequality $\pi x \leq \pi_0$ is a* valid inequality *for $P \subseteq \mathbb{R}^n$ if $\pi x \leq \pi_0 \; \forall x \in P$.*

Definition 14 *If $\pi x \leq \pi_0$ is a valid inequality for P, $F = \{x \in P \mid \pi x = \pi_0\}$ is called a* face *of P. A face F is said to be a $facet$ of P if $dim(F) = dim(P) - 1$.*

Theorem 4

(i) If P is a full-dimensional polyhedron, it has a unique (up to scalar multiplication) minimal description $P = \{x \in \mathbb{R}^n \mid a_i x \leq b_i$ for $i = 1, \ldots, m\}$, consisting of one inequality representing each facet of P.

(ii) If $dim(P) = n - k$ with $k > 0$, then P is described by a maximal set of linearly independent rows of $(A^=, b^=)$ and one inequality representing each facet of P.

The "best", or "strongest" valid inequalities are therefore the facet defining inequalities. Hence, algorithms trying to strengthen IP formulations will be based on the generation of such inequalities.

3.2. LP-based Branch-and-Bound

Solving $\mathcal{NP}$-hard problems is, in general, a difficult task. One possible way for solving such problems consists in applying the divide and conquer approach. In Branch-and-Bound (B&B) methods, bounds are calculated for the problem, which is divided into two or more distinct subproblems. New bounds are generated for the subproblems and only those potentially holding an optimal solution are kept for further processing, whereas the other ones can be pruned from the B&B tree.

The main idea in LP-based B&B is to use an LP-relaxation of the IP being solved in order to derive a lower bound of the objective function.

Definition 15 *The* linear programming relaxation *for the IP* $z_{IP} = min\{cx \mid x \in P \cap \mathbb{Z}^n\}$ *is the LP* $z_{LP} = min\{cx \mid x \in P\}$.

Proposition 5 *If a LP is the relaxation of an IP, then* $z_{LP} \leq z_{IP}$.

Furthermore every feasible solution of (3.4) provides an upper bound of the objective function z_{IP}.

Algorithm 5: LP-based Branch-and-Bound

Input: Initial problem S with formulation P
Initialization: List L with problem P; Upper bound $\overline{z} =$ Infinity
while *List L not empty* **do**
  Choose and remove problem S^i with formulation P^i from list L
  Solve LP-relaxation over P^i yielding solution x^i_{LP}, with objective value $\underline{z}^i$
  if P^i *is empty* **then**
    Prune by infeasibility
  else if $\underline{z}^i \geq \overline{z}$ **then**
    Prune by bound
  else if x^i_{LP} *integer* **then**
    $\overline{z} = \underline{z}^i$
    Incumbent $x^* = x^i_{LP}$
    Prune by optimality
  else
    Branch: put subproblems S^i_1 and S^i_2
    With formulations P^i_1 and P^i_2 into list L
Incumbent x^* is optimal solution to S.

Using the bounding information, subtrees of the branch and bound tree can be pruned if they are solved to optimality, if their local lower bound is greater than or equal to the global upper bound, or if they are infeasible. In Algorithm 5 we give a pseudocode of LP-based B&B, which leaves two open issues.

Branching, i.e. generating the subproblems

Branching is usually performed by partitioning the search space into two parts by choosing an integer variable that has a fractional value (x^f_i) in the LP solution. The two resulting branches are then defined by: $x_i \leq \lfloor x^f_i \rfloor$ and $x_i \geq \lceil x^f_i \rceil$. A common choice is to branch on *the most fractional variable*, i.e. the variable with fractional

part closest to 0.5. In many commercial systems there are other rules based on *estimating the cost* of forcing variable x_i to be integer. Another, more sophisticated and costly, rule for choosing branching variables is *strong branching* [132]. A set of integer variables that are fractional in the LP solution is chosen. For a specified number of simplex iterations, all branches (up and down on each of these variables) are reoptimized resulting in bounds for each of those branches. The variable having the largest effect (largest decrease of the local upper bound) is then chosen, and the real branching is performed on this variable.

Choosing next problem S^i

Significant pruning of the B&B tree is only possible if a feasible solution giving a strong upper bound is available. A *Depth-First Search* strategy, where the next processed node is an immediate descendant of the current one, often quickly results in such feasible solutions. Furthermore it is easy to resolve the LP when a single constraint is added and an optimal basis is available. On the other hand, it would is advantageous to minimize the total number of evaluated nodes, which can be achieved by choosing the active node with the largest lower bound. This leads to so called *Best-First Search* strategy. In commercial ILP-solvers more sophisticated combinations of the presented strategies are used.

3.3. Branch-and-Cut

The bounds obtained from the LP-relaxations are often weak, which may causes standard B&B algorithms to fail in practice. It is therefore of crucial importance to tighten the formulation of the problem to be solved. The idea of dynamically adding so called cutting planes to the problem is one way of obtaining stronger bounds. Combining the cutting plane algorithm with B&B results in the very powerful class of *Branch-and-Cut* (B&C) algorithms.

3.3.1. Cutting Plane Algorithm

In order to understand the strength of B&C algorithms, we will first present the main concepts of cutting plane algorithms.

From Section 3.1.3 we already know that an IP can be reformulated as a LP. However, in the case of $\mathcal{NP}$-hard problems, it is, in practice, nearly impossible to find such a formulation, or if such a formulation can be found it would usually need an

exponential number of constraints. With this in mind, one can try to generate only those additional constraints that are violated by a LP-relaxation and strengthen the original formulation. Such constraints are valid inequalities, which are also called cutting planes in this context.

Definition 16 *The* separation problem *associated with IP (3.4) is the problem: Given $\hat{x} \in \mathbb{R}^n$, is $\hat{x} \in conv(X)$? If not find a valid inequality $\pi x \leq \pi_0$ violated by $\hat{x}$.*

We can now describe a basic cutting plane algorithm for solving the IP (3.4) in Algorithm 6, leaving open how to solve the separation problem. Different systematic ways of generating valid inequalities and strong valid inequalities such as Gomory mixed integer cuts or knapsack cover cuts can be found in [92].

Algorithm 6: Cutting plane algorithm

Initialization: $t = 0$ and $P^0 = P$
loop
  Solve the LP $\overline{z}^t = \min\{cx : x \in P^t\}$. yielding solution x^t
  if $x^t \in \mathbb{Z}^n$ **then**
    Stop, x^t is an optimal solution for the IP.
  else
    Solve the separation problem for x^t.
    if *a valid inequality $\pi^t x \leq \pi_0^t$ cutting off x^t is found* **then**
      $P^{t+1} = P^t \cap \{x : \pi^t x \leq \pi_0^t\}$
    else
      Stop, no integer solution found
  $t = t + 1$

If, due to run-time reasons for example, not all the necessary cutting planes are generated, the cutting plane algorithm can terminate without finding an integral solution for the IP. Nevertheless, the improved formulation it has produced leads to a tighter bound and can be used as a better starting point of a classical B&B algorithm.

3.3.2. Branch-and-Cut

In order to find integral solutions to an IP, the cutting plane algorithm can be hybridized with B&B, resulting in the Branch-and-Cut (B&C) algorithm. The idea is to generate cutting planes throughout the B&B tree of a standard B&B algorithm, in order to get tight bounds at each node.

In practice, many issues such as cut pool management, different cut generation strategies, B&B strategies and many more have to be considered. Commercial IP-solvers such as CPLEX are based on the basic scheme explained here. However, they have implemented additionally a multitude of sophisticated strategies and methods which address the issues mentioned above.

3.4. Branch-and-Price

In Branch-and-Price, the concept of column generation is combined with a Branch-and-Bound algorithm.

3.4.1. Column Generation

The simplex algorithm is at the origin of the column generation concept, where only variables with negative reduced costs are allowed to enter the basis in each iteration. Given a LP model with a huge number of variables, possibly depending exponentially on the instance size, it would be efficient to consider only the variables potentially improving the objective function. The main idea is to efficiently determine a variable with negative reduced costs to enter the basis, add it to the problem, resolve it and iteratively repeat this process until no variable with negative reduced costs exists anymore.

In general, the method of Dantzig-Wolfe decomposition is often used for obtaining LP/ILP models with an exponential number of variables, which provide tighter bounds than the original compact LP/ILP pair. See [92, 132] for a description of Dantzig-Wolfe decomposition.

We will explain the principles of column generation using a small example. Consider the one-dimensional cutting stock problem. We are given stock-pieces of a given length L, which have to be cut into smaller pieces $i = 1, \ldots, m$ having lengths l_i with minimal waste, that is to say, using the minimal number of stock-pieces. For each piece i a demand d_i is given. Gilmore and Gomory [45, 46] gave an ILP formulation of this problem with an exponential number of variables: A variable x_j is introduced for each possible cutting layout of one stock piece. Since there is an exponential number of cutting patterns, the number of variables is also exponential in the size of the input. Matrix $A \in \mathbb{N}^{m \times n}$ contains the information about the n different patterns. Each column of matrix A represents a cutting pattern, each row

represents a piece which has to be cut out, and each entry A_{ij} represents how many instances of piece i are considered in pattern j.

$$\min\{x \mid Ax \geq d, x \in \mathbb{Z}^+ \} \tag{3.8}$$

The original approach of Gilmore and Gomory was to solve the LP-relaxation of (3.8) only. Rounding up the LP solution yields an integer feasible solution with objective value within $m-1$ of the optimum. In order to tackle the huge number of variables they used delayed column generation, whereby variables/patterns are only generated when they are needed in order to find the optimal solution.

Starting from a basic feasible solution the simplex algorithm needs, at each iteration, only a single additional variable with negative reduced costs (see Section 3.1.3). This property is used in the following. We define the so called Restricted Master Problem (RMP), a restricted LP-relaxation of the original problem (3.8):

$$\min\{x \mid A'x \geq d, x \in \mathbb{R}^+ \} \tag{3.9}$$

where A' is a $m \times n'$-matrix containing only a reduced set of variables/patterns. This set can be trivially initialized e.g. by introducing a variable for each item. The reduced costs of a variable for (3.9) with cutting pattern a are given by $1-ua$ where $u \in \mathbb{R}^m_+$ are the optimal dual variable values of (3.9) . The optimal solution of the *pricing problem*:

$$\max\{ua \mid al \leq L, a \in \mathbb{Z}^n_+\} \tag{3.10}$$

is iteratively added to the RMP if its objective value exceeds one and therefore has negative reduced costs. If no such variables exist anymore, the LP-relaxation of (3.8), the so called *Master Problem*, is optimally solved.

A more formal overview of the column generation method is presented in Algorithm 7.

Algorithm 7: Column generation

Start with a subset of the variables: RMP
Solve RMP
while *variable with negative reduced costs* $\overline{c_j}$ *exists* **do**
    Determine such a variable
    Add it to the RMP
    Resolve RMP

In order to solve the original problem to integer optimality, column generation has to be hybridized with branch-and-bound as described in the next section.

3.4.2. Branch-and-Price

Since column generation is an algorithm for solving LPs, it has to be combined with another method in order to solve IPs to optimality. The so called branch-and-price (B&P) algorithm [7] is the result of combining column generation with B&B. In each node of the B&B tree, column generation is performed to solve the LP-relaxation. Branching is usually performed on original variables or by other strategies to partition the remaining search space in a balanced way.

An important point is that the column generation algorithm used has to be aware of branching decisions and may only generate solutions respecting them. Another interesting question is whether the column generation algorithm should search for optimal solutions of the pricing problem or not. For a detailed review of column generation and B&P methods we refer to the recent book [26].

From a theoretical point of view, B&C and B&P are closely related, since column generation in the primal problem corresponds to cut generation in the dual and vice-versa. Furthermore, B&C and B&P can be combined to so called branch-and-cut-and-price algorithms, where both cuts and variables are dynamically generated.

Chapter 4

Combining Metaheuristics and Exact Algorithms: A Survey and Classification

4.1. Introduction

Hard combinatorial optimization problems (COPs) appear in a multitude of real-world applications, such as routing, assignment, scheduling, cutting and packing, network design, protein alignment, and many other fields of utmost economic, industrial and scientific importance. The available techniques for COPs can roughly be classified into two main categories: *exact* and *heuristic* methods. Exact algorithms are guaranteed to find an optimal solution and to prove its optimality for every instance of a COP. The run-time, however, often increases dramatically with the instance size, and in practice often only small or moderately-sized instances can be solved to provable optimality. In this case, the only possibility for larger instances is to trade optimality for run-time, yielding heuristic algorithms. In other words, the guarantee of finding an optimal solution is sacrificed for the sake of getting a good solutions in a limited time.

An earlier version of this chapter appeared in [101].

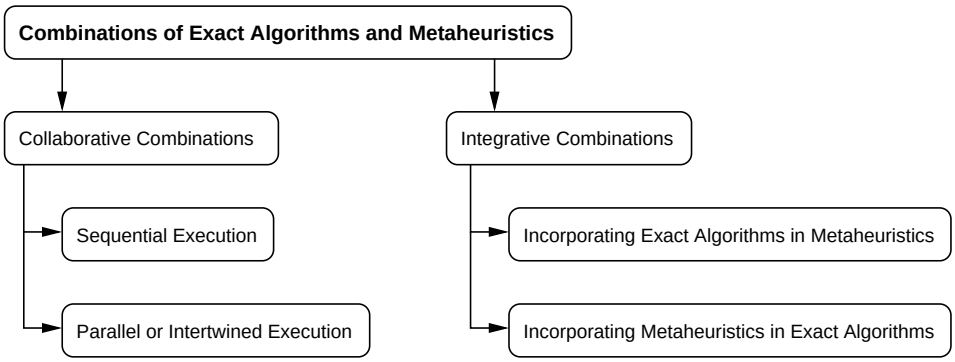

Figure 4.1.: Major classification of exact/metaheuristic combinations.

Two independent heterogeneous streams, coming from very different scientific communities, had significant success in solving COPs:

- *Integer Programming (IP)* as an exact approach, based on the concepts of linear programming [21].

- Local search with various extensions and independently developed variants, in the following called *metaheuristics*, as a heuristic approach.

Recently there have been very different attempts to combine ideas and methods from these two scientific streams. Dumitrescu and Stützle [30] describe existing combinations which focus on local search approaches that are strengthened by the use of exact algorithms. In their survey they concentrate on integration and exclude obvious combinations such as preprocessing.

Here, we present a more general classification of existing approaches combining exact and metaheuristic algorithms for combinatorial optimization. We distinguish the following two main categories:

- *Collaborative Combinations*: By collaboration we mean that the algorithms exchange information, but are not part of each other. Exact and heuristic algorithms may be executed sequentially, intertwined or in parallel.

- *Integrative Combinations*: By integration we mean that one technique is a subordinate embedded component of another technique. Thus, there is a distinguished master algorithm, which can be either an exact or a metaheuristic algorithm, and at least one integrated slave.

In the following sections this classification is further refined and examples are presented from the literature, which reflect developments at the forefront of current research. Figure 4.1 gives an overview of this classification.

4.2. Collaborative Combinations

The different algorithms and approaches described in this section have in common that they are top-level combinations of metaheuristics and exact techniques; no algorithm is contained within another. We further distinguish whether the algorithms are executed sequentially or in an intertwined or even parallel way.

4.2.1. Sequential Execution

Either the exact method is executed as a kind of preprocessing before the metaheuristic, or vice-versa. Sometimes it is difficult to say whether the first technique is used as an initialization of the second, or whether the second is a postprocessing of the solution(s) generated by the first.

Clements et al. [16] propose a column generation approach in order to solve a production-line scheduling problem. Each feasible solution of the problem consists of a line-schedule for each production line. First, the squeaky wheel optimization (SWO) heuristic is used to generate feasible solutions to the problem. SWO is a heuristic using a greedy algorithm to construct a solution, which is then analyzed in order to find the problematic elements. Higher priorities, which mean that these elements are considered earlier by the greedy algorithm, are assigned to them and the process restarts until a termination condition is reached. SWO is called several times in a randomized way in order to generate a set of diverse solutions. In the second phase, the line-schedules contained in these solutions are used as columns of a set-partitioning formulation for the problem, which is solved using the mixed-integer programming solver MINTO[2]. This process always provides a solution which is at least as good as, but usually better than the best solution devised by SWO. Reported results indicate that SWO performs better than a tabu-search algorithm.

Applegate et al. [3] propose an approach for finding near-optimal solutions to the traveling salesman problem. They derive a set of diverse solutions by multiple runs of an iterated local search algorithm. The edge-sets of these solutions are merged and the traveling salesman problem is finally solved to optimality on this strongly

[2] http://www.isye.gatech.edu/faculty/Martin_Savelsbergh/software

restricted graph. In this way a solution is achieved that is typically superior to the best solution of the iterated local search.

Klau et al. [69] follow a similar idea and combine a memetic algorithm with integer programming to heuristically solve the prize-collecting Steiner tree problem. The proposed algorithmic framework consists of three parts: extensive preprocessing, a memetic algorithm, and an exact branch-and-cut algorithm applied as post-optimization procedure to the merged final solutions of the memetic algorithm.

Plateau et al. [99] combine interior point methods and metaheuristics for solving the multiconstrained knapsack problem. The first part is an interior point method with early termination. By rounding and applying several different ascent heuristics, a population of different feasible candidate solutions is generated. This set of solutions is then used as the initial population for a path-relinking (scatter search) algorithm. Extensive computational experiments are performed on standard multiconstrained knapsack benchmark instances. Obtained results show that the presented combination is a promising research direction.

Sometimes, a relaxation of the original problem is solved to optimality and the obtained solution is repaired to act as a promising starting point for a subsequent metaheuristic. Often, the linear programming (LP) relaxation is used for this purpose, and only a simple rounding scheme is needed. For example, Raidl and Feltl [108] solve the generalized assignment problem using a hybrid genetic algorithm (GA). The LP-relaxation of the problem is solved using CPLEX[3] and its solution is used by a randomized rounding procedure to create a population of promising integral solutions. These solutions are, however, often infeasible; therefore, randomized repair and improvement operators are additionally applied, yielding an even more meaningful initial population for the GA. Reported computational experiments suggest that this type of LP-based initialization is effective.

Vasquez and Hao [127] heuristically solve the multiconstrained knapsack problem by reducing and partitioning the search space via additional constraints that fix the total number of items to be packed. The bounds for these constraints are calculated by solving a modified LP-relaxation of the multiconstrained knapsack problem. For each remaining part of the search space, parallel tabu-search is finally performed starting with a solution derived from the LP-relaxation of the partial problem. This hybrid algorithm yields excellent results, even for large benchmark instances with up to 2 500 items and 100 constraints.

Lin et al. [72] describe an exact algorithm for generating the minimal set of affine functions that describes the value function of the finite horizon partially observed Markov decision process. In the first step a GA is used to generate a set Γ of

[3] http://www.ilog.com

witness points, which is as large as possible. In the second step a component-wise domination procedure is performed in order to eliminate redundant points in Γ. The set generated so far does not, in general, fully describe the value function. Therefore, a Mixed Integer Program (MIP) is solved to generate the missing points in the final third step of the algorithm. Reported results indicate that this approach requires less time than some other numerical procedures.

Another kind of sequential combination of B&B and a GA is described by Nagar et al. [91] for a two-machine flowshop scheduling problem in which solution candidates are represented as permutations of jobs. Prior to running the GA B&B is executed down to a predetermined depth k and suitable bounds are calculated and recorded at each node of the explicitly stored B&B tree. During the execution of the GA the partial solutions up to position k are mapped onto the correct tree node. If the bounds indicate that no path below this node can lead to an optimal solution, the permutation is subjected to a mutation operator that has been specifically designed to change the early part of the permutation in a favorable way.

Tamura et al. [120] tackle a job-shop scheduling problem and start from its IP formulation. For each variable, they take the range of possible values and partition it into a set of subranges, which are then indexed. The chromosomes of the GA are defined so that each position represents a variable, and its value corresponds to the index of one of the subranges. The fitness of a chromosome is calculated using Lagrangian relaxation to obtain a bound on the optimal solution subject to the constraints that the values of the variables fall within the correct ranges. When the GA terminates, an exhaustive search of the region identified as the most promising is carried out to produce the final solution.

4.2.2. Parallel or Intertwined Execution

Instead of a strictly sequential batch approach, exact and heuristic algorithms may also be executed in a parallel or intertwined way. Such peer-to-peer combinations of exact/heuristic techniques are less frequent. An interesting framework for this purpose was proposed by Talukdar et al. [118, 119] with the so-called *asynchronous teams* (A-Teams). An A-Team is a problem solving architecture consisting of a collection of agents and memories connected into a strongly cyclic directed network. Each of these agents is an optimization algorithm and can work on the target problem, on a relaxation—i.e., a superclass—of it, or on a subclass of the problem. The basic idea of A-Teams is that these agents work asynchronously and autonomously on a set of shared memories. These shared memories consist of trial solutions for some problem (the target problem, a superclass, or a subclass as mentioned before), and the action of an agent consists of modifying the memory by adding a solution,

deleting a solution, or altering a solution. A-Teams have been successfully utilized in a variety of combinatorial optimization problems, see e.g. [13, 119].

Denzinger and Offerman [25] present a similar multi-agent based approach for achieving cooperation between search-systems with different search paradigms. The TECHS (TEams for Cooperative Heterogenous Search) approach consists of teams of one or more agents using the same search paradigm. The communication between the agents is controlled by so-called send- and receive-referees, in order to filter the exchanged data. Each agent is in a cycle between searching and processing received information. In order to demonstrate the usefulness of TECHS, a GA and a B&B based system for job-shop scheduling is described. The GA and B&B agents exchange only positive information (solutions), whereas the B&B agents can also exchange negative information (closed subtrees). Computational experiments show that the cooperation results in finding better solutions given a fixed time-limit and in finding solutions comparable to the ones of the best individual system alone in less time.

Gallardo, Cotta and Fernández [41] present a hybridization of an evolutionary algorithm and a branch-and-bound approach, evaluated on the multidimensional knapsack problem. The algorithms are executed in an intertwined way and are cooperating by exchanging information. The EA provides bounds for the B&B, while B&B provides best and partial solutions to the EA. First the EA is executed until a certain convergence criterion is reached, providing an initial bound for the exact algorithm. Then the B&B is executed until it finds an improved solution, and gives control back to the EA, which incorporates the new better solution into its population as well as some promising partial solution from the ongoing B&B search. Control is switched between the algorithms until a run-time limit is reached. The experimental results presented in this article, show that the collaborative approach yields better results than the individual techniques executed on their own.

4.3. Integrative Combinations

In this section we discuss approaches of combining exact algorithms and metaheuristics in an integrative way such that one technique is a subordinate embedded component of another technique.

4.3.1. Incorporating Exact Algorithms in Metaheuristics

We start by considering techniques where exact algorithms are incorporated into metaheuristics.

Exactly Solving Relaxed Problems

The usefulness of solutions to relaxations of an original problem has already been mentioned in Section 4.2.1. Besides exploiting them to derive promising initial solutions for a subsequent algorithm, they can be of great benefit for heuristically guiding neighborhood search, recombination, mutation, repair and/or local improvement. Examples where the solution of the LP-relaxation and its dual were exploited in such ways are the hybrid genetic algorithms for the multiconstrained knapsack problem from Chu and Beasley [14] and Raidl [107].

Exactly Searching Large Neighborhoods

A common approach is to search neighborhoods in local search based metaheuristics by means of exact algorithms. If the neighborhoods are chosen appropriately, they can be relatively large and nevertheless an efficient search for the best neighbor is still reasonable. Such techniques are known as Very Large-Scale Neighborhood (VLSN) search [2].

Burke et al. [12] present an effective local and variable neighborhood search heuristic for the asymmetric traveling salesman problem in which they have embedded an exact algorithm in the local search part, called HyperOpt, in order to search relatively large promising regions of the solution space exhaustively. Moreover, they propose a hybrid of HyperOpt and 3-opt which benefits from the advantages of both approaches and gains better tours overall. Using this hybrid within the variable neighborhood search metaheuristic framework also allows local optima to be overcomed and tours of high quality to be created.

Dynasearch [17, 18] is another example where exponentially large neighborhoods are explored. The neighborhood where the search is performed consists of all possible combinations of mutually independent simple search steps, and one Dynasearch move consists of a set of independent moves that are executed in parallel in a single local search iteration. Independence in the context of Dynasearch means that the individual moves do not interfere with each other; in this case, dynamic programming can be used to find the best combination of independent moves. Dynasearch is restricted to problems where the single search steps are independent, and it has so far only been applied to problems where solutions are represented by permutations.

For the class of partitioning problems, Thompson et al. [121, 122] defined the concept of a cyclic exchange neighborhood, which is the transfer of single elements between several subsets in a cyclic manner; for example, a 2–exchange move can be seen as a cyclic exchange of length two. Thompson et al. showed that for any current solution

to a partitioning problem a new, edge-weighted graph can be constructed, where the set of nodes is split into subsets according to a partition induced by the current solution of the partitioning problem. A cyclic exchange for the original problem corresponds to a cycle in this new graph that uses at most one node of each subset. Exact and heuristic methods that solve the problem of finding the most negative-cost subset-disjoint cycle (which corresponds to the best improving neighbor of the current solution) have been developed.

Puchinger et al. [105] describe a combined GA/B&B approach for solving a real-world glass cutting problem. The GA uses an order-based representation, which is decoded using a greedy heuristic. The B&B algorithm is applied with a certain probability enhancing the decoding phase by generating locally optimal subpatterns. Reported results indicate that the approach of occasionally solving subpatterns to optimality may increase the overall solution quality.

The work of Klau et al. [69] has already been mentioned in Section 4.2.1 in the context of collaborative sequential combinations. When looking at the memetic algorithm we encounter another kind of exact/heuristic algorithm combination. An exact subroutine for the prize-collecting Steiner tree problem on trees is used to improve candidate solutions locally.

Merging Solutions

Subspaces defined by the merged attributes of two or more solutions can, like the neighborhoods of single solutions, also be searched by exact techniques. The algorithms by Clements et al. [16], Applegate et al. [3], and Klau et al. [69], which were already discussed in Section 4.2.1, also follow this idea, but are of sequential collaborative nature. Here, we consider approaches where merging is iteratively applied within a metaheuristic.

Cotta and Troya [19] present a framework for hybridizing B&B with evolutionary algorithms. B&B is used as an operator embedded in the evolutionary algorithm. The authors recall the necessary theoretical concepts on forma analysis (formae are generalized schemata), such as the dynastic potential of two chromosomes x and y, which is the set of individuals that only carry information contained in x and y. Based on these concepts the idea of dynastically optimal recombination is developed. This results in an operator exploring the potential of the recombined solutions using B&B, providing the best possible combination of the ancestors' features that can be attained without introducing implicit mutation. Extensive computational experiments on different benchmark sets comparing different crossover operators with the new hybrid one show the usefulness of the presented approach.

Marino et al. [82] present an approach where a GA is combined with an exact method for the Linear Assignment Problem (LAP) to solve the graph coloring problem. The LAP algorithm is incorporated into the crossover operator and generates the optimal permutation of colors within a cluster of nodes, thereby preventing the offspring from being less fit than its parents. The algorithm does not outperform other approaches, but provides comparable results. The main conclusion is that solving the LAP in the crossover operator strongly improves the performance of the GA compared to the GA using crossover without LAP.

Exact Algorithms as Decoders

In evolutionary algorithms, candidate solutions are sometimes only incompletely represented in the chromosome, and an exact algorithm is used as decoder for determining the missing parts in an optimal way.

Staggemeier et al. [116], for example, present a hybrid genetic algorithm to solve a lot-sizing and scheduling problem minimizing inventory and backlog costs of multiple products on parallel machines. Solutions are represented as product subsets for each machine at each period. Corresponding optimal lot sizes are determined when the solution is decoded by solving a linear program. The approach outperforms a MIP formulation of the problem solved using CPLEX.

4.3.2. Incorporating Metaheuristics in Exact Algorithms

We now turn to techniques where metaheuristics are embedded within exact algorithms.

Metaheuristics for Obtaining Incumbent Solutions and Bounds

In general, heuristics and metaheuristics are often used to determine bounds and incumbent solutions in B&B approaches. For example, Woodruff [133] describes a chunking-based selection strategy to decide at each node of the B&B tree whether or not reactive tabu search is called in order to eventually find a better incumbent solution. The chunking-based strategy measures a distance between the current node and nodes already explored by the metaheuristic in order to bias the selection toward distant points. Reported computational results indicate that adding the metaheuristic improves the B&B performance.

Metaheuristics for Column and Cut Generation

In branch-and-cut and branch-and-price algorithms, the dynamic separation of cutting-planes and the pricing of columns respectively is sometimes done by means of heuristics including metaheuristics, in order to speed up the whole optimization process.

Filho and Lorena [34] apply a heuristic column generation approach to graph coloring. They describe the principles of their constructive genetic algorithm and give a column generation formulation of the problem. The GA is used to generate the initial columns and to solve the slave problem (the weighted maximum independent set problem) at every iteration. Column generation is performed as long as the GA finds columns with negative reduced costs. The master problem is solved using CPLEX. Some encouraging results are shown.

Puchinger and Raidl [100, 102] (see also chapter 5) propose new integer linear programming formulations for the three-stage two-dimensional bin packing problem. Based on these formulations, a branch-and-price algorithm was developed in which fast column generation is performed by applying a hierarchy of four methods: (a) a greedy heuristic, (b) an evolutionary algorithm, (c) solving a restricted form of the pricing problem using CPLEX, and finally (d) solving the complete pricing problem using CPLEX. Computational experiments on standard benchmark instances document the benefits of the new approach. The combination of all four pricing algorithms in the proposed branch-and-price framework yields the best results in terms of the average objective value, the average run-time, and the number of instances solved to proven optimality.

Metaheuristics for Strategic Guidance of Exact Search

French et al. [37] present a GA/B&B hybrid to solve feasibility and optimization IP problems. Their hybrid algorithm combines the generic B&B of the MIP-solver XPRESS-MP[4] with a steady-state GA. It starts by traversing the B&B tree. During this phase, information from nodes is collected in order to suggest chromosomes to be added to the originally randomly initialized GA-population. When a certain criterion is fulfilled, the GA is started using the augmented initial population. When the GA terminates, its fittest solution is passed back and grafted onto the B&B tree. Full control is given back to the B&B-engine, after the newly added nodes had been examined to a certain degree. Reported results on MAX-SAT instances show that this hybrid approach yields better solutions than B&B or the GA alone.

[4] http://www.dashoptimization.com/

Kotsikas and Fragakis [70] determine improved node selection strategies within B&B for solving MIPs by using genetic programming (GP). After running B&B for a certain amount of time, information is collected from the B&B tree and used as a training set for GP, which is performed to find a node selection strategy more appropriate for the specific problem at hand. The following second B&B phase then uses this new node selection strategy. Reported results show that this approach has potential, but needs to be enhanced in order to be able to compete with today's state-of-the-art node selection strategies.

Applying the Spirit of Metaheuristics

Last but not least, there are a few approaches where the spirit of local search based techniques is incorporated into B&B. The main idea is to first search some neighborhood of incumbent solutions more intensively before turning to a classical node selection strategy. However, there is no explicit metaheuristic, and B&B itself is used for performing the local search. The metaheuristic may be seen to be executed in a "virtual" way.

Fischetti and Lodi [35] introduced local branching, an exact approach combining the spirit of local search metaheuristics with a generic MIP-solver (CPLEX). They consider general MIPs with 0-1 variables. The idea is to iteratively solve a local subproblem corresponding to a classical k-OPT neighborhood using the MIP-solver. This is achieved by introducing a local branching constraint based on an incumbent solution $\overline{x}$, which partitions the search space into the k-OPT neighborhood and the rest: $\Delta(x, \overline{x}) \leq k$ and $\Delta(x, \overline{x}) \geq k + 1$, respectively, with Δ being the Hamming distance of the 0-1 variables. The first subproblem is solved, and if an improved solution could be found, a new subproblem is devised and solved; this is repeated as long as an improved solution is found. If the process stops, the rest of the problem is solved in a standard way. This basic mechanism is extended by introducing time limits, automatically modifying the neighborhood size k and adding diversification strategies in order to improve the performance. Reported results are promising.

Danna et al. [20] present an approach called Relaxation Induced Neighborhood Search (RINS) in order to explore the neighborhoods of promising MIP solutions more intensively. The main idea is to occasionally devise a sub-MIP at a node of the B&B tree that corresponds to a certain neighborhood of an incumbent solution: First, variables having the same values in the incumbent and in the current solution of the LP-relaxation are fixed. Second, an objective cutoff based on the objective value of the incumbent is set. Third, a sub-MIP is solved on the remaining variables. The time for solving this sub-MIP is limited. If a better incumbent could be found during this process, it is passed to the global MIP-search which is resumed after

the sub-MIP termination. CPLEX is used as MIP-solver. The authors experimentally compare RINS to standard CPLEX, local branching, combinations of RINS and local branching, and guided dives. Results indicate that RINS often performs best.

4.4. Conclusions

We gave a survey on very different existing approaches for combining exact algorithms and metaheuristics. The two main categories in which we divided these techniques were collaborative and integrative combinations. Some of the combinations are dedicated to very specific combinatorial optimization problems, whereas others were designed to be more generally useful. Altogether, the existing work documents that both, exact optimization techniques and metaheuristics have specific advantages which complement each other. Suitable combinations of exact algorithms and metaheuristics can benefit greatly from synergy and often exhibit significantly higher performance with respect to solution quality and time. Some of the presented techniques are well developed, whereas others are still in their infancy and need substantial further research in order to develop them fully. Future work on such hybrid systems is highly promising.

Chapter 5

Models and Algorithms for Three-Stage Two-Dimensional Bin Packing

5.1. Introduction

The two-dimensional bin packing (2BP) problem occurs in different variants of important real-world applications such as glass, paper, and steel cutting. In almost every variant of the *2BP* problem, we are given a set of n rectangular items having individual heights h_i and widths w_i, $i = 1, ..., n$. The objective is to pack them into a minimum number of rectangular bins or cut them out of a minimum number of sheets of some raw-material, each having height H and width W. Items may not overlap and we do not allow rotation. We assume $0 < w_i \leq W,\ 0 < h_i \leq H$. Recent surveys on 2D packing problems are given in Lodi et al. [75] and Lodi et al. [73], an annotated bibliography on cutting and packing is presented by Dyckhoff et al. [31].

In practice, there are often special requirements on the cutting/packing patterns. Here, we consider in particular orthogonal *guillotine* cuts; i.e., pieces are always

An earlier version of this chapter appeared in [102].

rectangular and may only be cut horizontally or vertically from one border of a given or produced rectangle to the opposite one. Furthermore, real-world cutting machines are often constructed such that the sheets are processed in a certain number of *stages*. In each stage either horizontal or vertical cuts can be performed, but never both. Pieces having passed a stage may not be put back to a previous stage. These conditions limit the nesting of horizontal and vertical cuts and thus the maximum height of the slicing-tree of each bin. In this article we focus on three-stage problems, where the first stage is only able to perform horizontal cuts, the second only vertical cuts, and the third again only horizontal cuts; see Figure 5.1 (on page 53) for an example of a feasible cutting pattern. The three-stage restriction is, for example, typical for glass manufacturing [40, 105].

The next section gives a short overview on previous work related to the considered problem. The combination of exact and heuristic methods for solving difficult problems is a central concern of the work presented here. Section 5.3 defines the three-stage two-dimensional bin packing (3-stage 2BP) problem in a more formal way. We then develop an integer linear programming (ILP) model for a restricted version of 3-stage 2BP and extend it to a model for the unrestricted case in section 5.4. These models involve only $O(n^2)$ and $O(n^3)$ variables, the number of constraints is bounded by $O(n)$ and $O(n^3)$ respectively, and symmetries are effectively avoided. To our knowledge, this is the first polynomially-sized ILP for 3-stage 2BP.

In section 5.5, we describe an alternative approach based on a Dantzig-Wolfe decomposition [126]: the 3-stage 2BP problem is reformulated as a set covering problem and fast column generation is performed. We also introduce dual subset inequalities in order to derive dual constraints to stabilize the column generation process. Section 5.6 describes how branching is performed in order to obtain optimal integer solutions. Furthermore, the column generation process is described in detail: Columns are generated by using a hierarchy of four methods, namely a greedy heuristic, an evolutionary algorithm, and a restricted as well as an unrestricted ILP for the pricing problem. In section 5.7, computational experiments are described and analyzed. Finally we summarize our work and draw conclusions in section 5.8.

5.2. Previous Work

In this section we will present the previous work we consider relevant for this chapter, for more detailed information we refer to the surveys by Lodi et al. [75] and Lodi et al. [73] and the annotated bibliography by Dyckhoff et al. [31].

Approximation Algorithms

Most of the simple algorithms for 2BP are of greedy nature. Items are placed one by one and never reconsidered again. One- and two-phase algorithms are presented in the literature [75]. In one-phase algorithms, the items are directly placed into the bins, whereas the two-phase algorithms first partition the items into levels whose widths do not exceed W and which aim to minimize the total height. In the second phase, levels are then assigned to bins by solving a one-dimensional bin packing problem.

An example of a two-phase algorithm is Hybrid First-Fit (HFF) described by Chung et al. [15]. In the first phase, the first fit decreasing height algorithm for strip packing is applied. The items are sorted by decreasing heights, and are then placed, one after the other, onto the first level they fit. If there is no such level, a new level as high as the item requiring it is created. The levels are then assigned to bins by applying the one-dimensional first fit decreasing algorithm, which firstly sorts the levels according to decreasing height, and secondly places the levels, one after the other, into the first bin they fit. If no such bin exists, a new empty bin is created were the level can be packed. In [15] it is shown that if the heights are normalized to one $\mathrm{HFF(I)} \leq \frac{17}{8}\mathrm{OPT(I)} + 5$.

Berkey and Wang [9] described a variant of HFF, the finite first fit heuristic, which is a greedy one-phase algorithm. Items are sorted by decreasing heights. The first item initializes the first level in the first bin and defines the level's height. Each following item is added to the first level into which it fits, respecting the bin's width. If there is no such level, a new level is initialized in the first bin into which it fits. And, if there is no such bin, a new bin is initialized with the new level. Within a level, items are never stacked.

Set Covering Based Approaches

The approach of formulating a packing or cutting problem as a set covering problem, which we will pursue in section 5.5, originates in the work of Gilmore and Gomory [45]. They propose this technique for the one dimensional cutting stock problem. This formulation introduces a variable for each possible cutting pattern of a single bin. Since, in general, the number of these variables increases exponentially with respect to the problem size, not all variables are explicitly considered and column generation is performed.

In [47], Gilmore and Gomory applied the same basic technique to two-dimensional stock cutting. The major difference lies in the method for solving the pricing prob-

lem, i.e., in the way of determining promising patterns/variables that may improve a current solution. With respect to stage-constraints, only two-stage guillotineable patterns are considered. Unfortunately, this approach cannot directly be extended to three or more stages in an efficient way.

A faster variant of the Gilmore and Gomory approach is proposed by Oliveira and Ferreira [93], where three-stage and general multi-stage cutting stock problems are considered. For solving the pricing problem, a greedy heuristic is first applied in the hope that it will quickly find a suitable variable. Only if this heuristic fails, a slower exact algorithm is used.

Monaci and Toth [88] present a set covering based heuristic approach for bin-packing problems. In a first phase, columns (i.e. patterns) are generated using greedy and fast constructive heuristics, in a second phase the associated set-covering instance is solved by means of a Lagrangian-based heuristic algorithm.

A Compact ILP for Level Bin Packing

Recently the two-dimensional level bin packing problem (corresponding to 2-stage 2BP) was considered in Lodi et al. [76]. They introduced the first compact ILP model involving only a polynomial number of variables and constraints. This work is the basis for the ILPs we introduce in section 5.4 for 3-stage 2BP.

In level packing the bins are first partitioned in levels of a fixed height, and the items are then packed into these levels one beside the other. Some simple observations regarding the nature of optimal level packing patters are presented by Lodi et al.: For any optimal level there exists an equivalent solution in which

- the first item packed in each level is the tallest item in the level;
- the first level packed in each bin is the tallest level in the bin.

They further assume that the items are sorted according to nonincreasing heights. Four set of variables are used: the first two sets refer to the packing of items into levels, the remaining two to the packing of levels into bins. Levels and bins are respectively initialized by items and levels. The heights of the levels are determined by their initializing item. The variables are:

- $y_i \in \{0,1\}$, $i = 1, \cdots, n$, set to one if and only if (iff) item i initializes level i.
- $x_{ij} \in \{0,1\}$, $i = 1, \cdots, n-1; j > i$, set to one iff item j is packed into level i.
- $q_k \in \{0,1\}$, $k = 1, \cdots, n$, set to one iff level k initializes bin k .
- $z_{ki} \in \{0,1\}$, $k = 1, \cdots, n-1; i > k$, set to one iff level i is allocated to bin k.

Using these variables the ILP model given in [76] is:

$$\text{minimize} \quad \sum_{k=1}^{n} q_k \tag{5.1}$$

$$\text{subject to} \quad \sum_{i=1}^{j-1} x_{ij} + y_j = 1, \quad \forall j = 1, \ldots, n \tag{5.2}$$

$$\sum_{j=i+1}^{n} w_j x_{ij} \leq (W - w_i) y_i, \quad \forall i = 1, \ldots, n-1 \tag{5.3}$$

$$\sum_{k=1}^{i-1} z_{ki} + q_i = y_i, \quad \forall i = 1, \ldots, n \tag{5.4}$$

$$\sum_{i=k+1}^{n} h_i z_{ki} \leq (H - h_k) q_k, \quad \forall k = 1, \ldots, n-1 \tag{5.5}$$

$$y_i \in \{0, 1\}, \quad i = 1, \ldots, n \tag{5.6}$$

$$x_{ij} \in \{0, 1\}, \quad i = 1, \ldots, n-1; j > i \tag{5.7}$$

$$q_k \in \{0, 1\}, \quad k = 1, \ldots, n \tag{5.8}$$

$$z_{ki} \in \{0, 1\}, \quad k = 1, \ldots, n-1; i > k \tag{5.9}$$

The number of bins is minimized in the objective function (5.1). Equations (5.2) ensure that each item i has to be packed once. If item i initializes level i, y_i is set to one, and the item cannot be packed in any other level, therefore $\sum_{i=1}^{j-1} x_{ij}$ has to be zero. Analogously if $\sum_{i=1}^{j-1} x_{ij} = 1$, y_i has to be zero. Constraints (5.3) guarantee that the bins' width W is not exceeded by any level. The total width of a level i corresponds to $\sum_{j=i+1}^{n} w_j x_{ij} + w_i$ if it is initialized. If the level is not initialized at all, y_i remains zero, and no item can be packed into the level due to these inequalities. Equations (5.4) force each used level to be packed into exactly one bin. If a level i is initialized ($y_i = 1$), it has to initialize bin i ($q_i = 1$) or it has to be packed into another bin ($\sum_{k=1}^{i-1} z_{ki} = 1$). Finally, constraints (5.5) guarantee that each bin's height H is not exceeded by the total height of the packed levels. Analogously to (5.3), it is furthermore assured that levels can only be packed into initialized bins (any z_{ki} can only be greater than zero if $q_k > 0$).

Evolutionary Algorithms for 2BP

A more general overview of evolutionary algorithms for cutting and packing problems is given by Hopper [63]. Most of the EAs used to solve guillotineable 2BP, such as the

genetic algorithm described by Kröger [71], are based on a slicing-tree representation and specialized variation operators. A cutting-pattern is represented as a tree, the root node corresponds to the whole sheet, the children are the slices generated by the first cutting-stage. The following children are generated by further cutting stages, and the leaves of the tree correspond to the items cut out of the sheet.

Alternatively, an order-based encoding can be used indicating the order in which the items are placed by some FFF-like decoding heuristic. Hwang et al. [64] describe such an approach and compare it to a slicing-tree representation. They conclude that the order-based method yields better results in several cases. Bisotto et al. [11] apply the order-based encoding successfully to an industrial cutting problem with specific real-time and pattern constraints. Another effective order-based approach is presented by Monaci [87] for two-dimensional strip packing.

General 2BP

The following two algorithms solve the more general 2BP problem exactly where non-guillotineable patterns of rectangular items are also allowed. Martello and Vigo [84] describe a two-level branch-and-bound algorithm. Items are assigned to bins by an outer decision tree. Possible packing patterns for the bins are generated by trying a heuristic first; if it fails to place all necessary items, it tries to find a pattern through an inner enumeration scheme. A hybrid branch-and-price / constraint programming algorithm has been proposed by Pisinger and Sigurd [98, 115]. They use the column generation principle of Gilmore and Gomory and solve the specific pricing problem by means of constraint programming.

Lodi et al. [74] describe a general heuristic framework applicable to several variants of 2BP. Tabu search is used to assign the items to bins, and cutting patterns for individual bins are obtained through different inner heuristics.

Three-Stage 2BP

Particular real-world three-stage cutting problems with specific additional properties were treated in Vanderbeck [125] and Puchinger et al. [105]. Vanderbeck solves a three-stage two-dimensional cutting stock problem, where the main objective is to minimize waste but other issues such as aging stock pieces, urgent or optimal orders, and fixed setup costs are also considered. His solution approach uses nested decomposition of the problem and a recursive use of column generation. Puchinger et al. consider a 3-stage 2BP problem appearing in glass cutting, where specific additional

constraints with respect to the order of items are imposed. The problem is heuristically solved using an evolutionary algorithm (EA) based on an order representation, specific recombination and mutation operators, and a greedy decoding heuristic. In one variant, branch-and-bound is occasionally applied to locally optimize parts of a solution during decoding.

5.3. Three-Stage Two-Dimensional Bin Packing

A feasible solution for *3-stage 2BP* consists of a set of *bins*, where each bin is partitioned into a set of *stripes*, each stripe consists of a set of *stacks*, and each stack consists of a set of *items* having equal width. The packing patterns of such a solution can always be transformed into the so-called *normal form* by moving each item to its uppermost and leftmost position, so that void space may only appear at the bottom of stacks, to the right of the last stack in each stripe and below the last stripe; see Figure 5.1 for an example. In the sequel we consider patterns in normal form only.

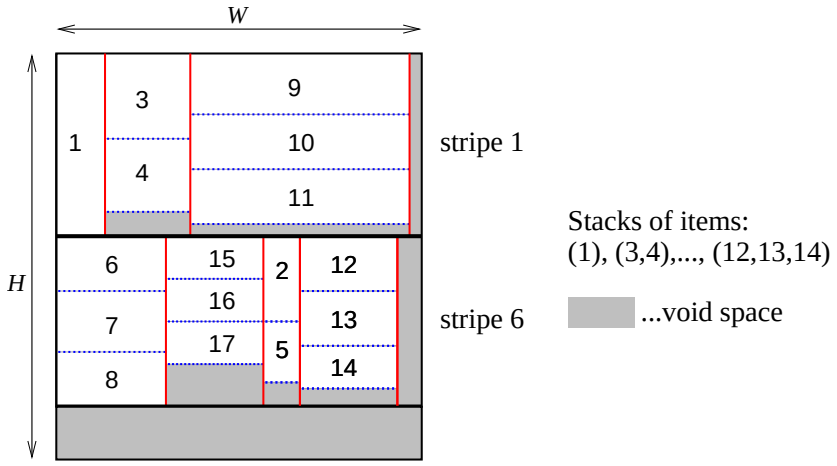

Figure 5.1.: A three-stage cutting pattern for one bin in normal form.

We assume the items to be sorted so that $h_1 \geq h_2 \geq \ldots \geq h_n$. The order of the items within each stack does not affect the feasibility and the objective value of a solution, so the items can always be sorted according to their indices. A solution may contain a maximum of n stacks. We label each stack with the index of the highest item it contains, i.e., with its smallest item index. Similarly, a solution has at most n stripes, and a stripe's label corresponds to the label of its highest stack.

Finally, a maximum of n bins is needed, and we label each of the bins with the smallest index of the stripes it contains. In the example shown in Figure 5.1, the stack labels are 1, 3, 9, 6, 15, 2, and 12, the stripes are labeled 1 and 6, and the bin's label is 1.

5.4. Integer Linear Programming Models

In this section, new ILP models for different versions of 3-stage 2BP are presented. These formulations are based on concepts for 2-stage 2BP from Lodi et al. [76].

5.4.1. Restricted Three-Stage Two-Dimensional Bin Packing

We first describe a model for *restricted 3-stage 2BP* where the highest stack of each stripe, which determines the label of the stripe, always consists of a single item. Therefore, the highest item (i.e., the one with the smallest index) of a stripe defines its height. This restriction helps to avoid some difficulties when calculating the total height of all stripes contained in a bin. An overview of other primal restriction strategies can be found in [27].

The model uses the following variables.

- $\alpha_{j,i} \in \{0,1\},\ j = 1,\dots,n,\ i = j,\dots,n$:
  set to one if and only if (iff) item i is contained in stack j

- $\beta_{k,j} \in \{0,1\},\ k = 1,\dots,n,\ j = k,\dots,n$:
  set to one iff stack j is contained in stripe k

- $\gamma_{l,k} \in \{0,1\},\ l = 1,\dots,n,\ k = l,\dots,n$:
  set to one iff stripe k is contained in bin l

The restricted 3-stage 2BP problem can now be stated as the following ILP.

$$\text{minimize} \quad \sum_{l=1}^{n} \gamma_{l,l} \tag{5.10}$$

$$\text{subject to} \quad \sum_{j=1}^{i} \alpha_{j,i} = 1, \ \forall i = 1, \ldots, n \tag{5.11}$$

$$\alpha_{j,i} = 0, \ \forall j = 1, \ldots, n-1, \forall i > j \mid w_i \neq w_j \tag{5.12}$$

$$\sum_{k=1}^{j} \beta_{k,j} = \alpha_{j,j}, \quad \forall j = 1, \ldots, n \tag{5.13}$$

$$\sum_{i=j}^{n} h_i \alpha_{j,i} \ \leq \sum_{k=1}^{j} h_k \beta_{k,j}, \quad \forall j = 1, \ldots, n-1 \tag{5.14}$$

$$\sum_{j=k}^{n} w_j \beta_{k,j} \ \leq \ W \beta_{k,k}, \quad \forall k = 1, \ldots, n-1 \tag{5.15}$$

$$\sum_{l=1}^{k} \gamma_{l,k} = \beta_{k,k}, \quad \forall k = 1, \ldots, n \tag{5.16}$$

$$\sum_{k=l}^{n} h_k \gamma_{l,k} \leq \ H \gamma_{l,l}, \quad \forall l = 1, \ldots, n-1 \tag{5.17}$$

$$\alpha_{j,i} \in \{0,1\}, \ j = 1, \ldots, n, \ i = j, \ldots, n \tag{5.18}$$

$$\beta_{k,j} \in \{0,1\}, \ k = 1, \ldots, n, \ j = k, \ldots, n \tag{5.19}$$

$$\gamma_{l,k} \in \{0,1\}, \ l = 1, \ldots, n, \ k = l, \ldots, n \tag{5.20}$$

The objective function (5.10) minimizes the number of used bins. Note that bin l is used iff $\gamma_{l,l} = 1$. Equations (5.11) ensure that each item i has to be packed once. The fact that the items packed into the same stack must have identical width and that the total height of any pair of stacked items must not exceed H is guaranteed by (5.12). In an actual implementation it is only necessary to consider the variables $\alpha_{j,i}$ for which $w_i = w_j$ and $h_i + h_j \leq H$. Here, however, we keep all variables in our model for the sake of clarity. Each *used* stack j—i.e., stack j contains item j and thus $\alpha_{j,j} = 1$—is packed exactly once into a stripe k according to equations (5.13). Constraints (5.14) ensure that the height of each stack j—i.e., the total height of all items contained in stack j—never exceeds the height of the associated stripe k, which is identical to item k's height due to our special problem restriction. The constraints (5.13) and (5.14) together further imply that no items may be packed into an unused stack. Constraints (5.15) guarantee that the bins' width W is not exceeded by any

stripe k and that no stacks are packed into unused stripes ($\beta_{k,k} = 0$). Equations (5.16) force each used stripe k to be packed into exactly one bin. Finally, constraints (5.17) guarantee that each bin's height H is not exceeded by the total height of the packed stripes and that no stripes are packed into unused bins ($\gamma_{l,l} = 0$). In total, the ILP uses $O(n^2)$ variables and $O(n)$ constraints, if taking into consideration that the fixing of variables $\alpha_{j,i}$ according to (5.12) can be done during preprocessing.

5.4.2. The Unrestricted Case

When modeling the unrestricted case we must take into account that neither the height of each stack is necessarily given by its highest item, nor that the highest stack of a stripe necessarily contains the stripe's highest item. Therefore, we additionally need variables $\beta_{k,j}$ for $j < k$, thus

- $\beta_{k,j} \in \{0,1\},\ k = 1,\dots,n,\ j = 1,\dots,n$:
  set to one iff stack j is contained in stripe k.

To extend the ILP (5.10) to (5.17), we must replace the height constraints for sheets (5.17) in particular. A straightforward way to achieve this, is to write

$$\sum_{k=l}^{n} \mathcal{H}(k)\gamma_{l,k} \leq\ H\gamma_{l,l}, \quad \forall l = 1,\dots,n-1 \tag{5.21}$$

with $\mathcal{H}(k)$ being the height of stack k

$$\mathcal{H}(k) = \sum_{i=1}^{n} h_i \alpha_{k,i}. \tag{5.22}$$

The left hand sides of inequalities (5.21) are, however, nonlinear. In order to obtain an ILP, we need to introduce additional variables

- $\delta_{l,i,j} \in \{0,1\}$, $l = 1,\dots,n-1$, $i = l+1,\dots,n$, and $j = l,\dots,i-1$:
  set to one iff item i contributes to the total height of all stripes in bin l and is contained in stack j; i.e., item i is contained in stack j, stack j is contained in stripe j (and therefore defines its height), and stripe j is contained in bin l or simply
  $$\delta_{l,i,j} = 1\ \leftrightarrow\ \alpha_{j,i} = 1 \wedge \gamma_{l,j} = 1. \tag{5.23}$$

Now, we can calculate the used height of a bin l in a linear way by

$$\sum_{i=l}^{n} h_i \gamma_{l,i} + \sum_{i=l+1}^{n} h_i \sum_{j=l}^{i-1} \delta_{l,i,j}. \tag{5.24}$$

The complete ILP for the (unrestricted) 3-stage 2BP looks as follows.

$$\text{minimize} \quad \sum_{l=1}^{n} \gamma_{l,l} \tag{5.25}$$

$$\text{subject to} \quad \sum_{j=1}^{i} \alpha_{j,i} = 1, \ \forall i = 1, \ldots, n \tag{5.26}$$

$$\sum_{i=j+1}^{n} \alpha_{j,i} \leq (n-j)\alpha_{j,j}, \quad \forall j = 1, \ldots, n-1 \tag{5.27}$$

$$\alpha_{j,i} = 0, \ \forall j = 1, \ldots, n-1 \ \ \forall i > j \mid w_i \neq w_j \vee h_i + h_j > H \tag{5.28}$$

$$\sum_{k=1}^{n} \beta_{k,j} = \alpha_{j,j}, \quad \forall j = 1, \ldots, n \tag{5.29}$$

$$\sum_{i=j}^{n} h_i \alpha_{j,i} \ < \ \sum_{i=k}^{n} h_i \alpha_{k,i} + (H+1)(1-\beta_{k,j}),$$
$$\forall k = 2, \ldots, n, \quad \forall j = 1, \ldots, k-1 \tag{5.30}$$

$$\sum_{i=j}^{n} h_i \alpha_{j,i} \ \leq \ \sum_{i=k}^{n} h_i \alpha_{k,i} + H(1-\beta_{k,j}),$$
$$\forall k = 1, \ldots, n-1, \quad \forall j = k+1, \ldots, n \tag{5.31}$$

$$\sum_{j=1}^{n} w_j \beta_{k,j} \ \leq \ W\beta_{k,k}, \quad \forall k = 1, \ldots, n \tag{5.32}$$

$$\sum_{l=1}^{k} \gamma_{l,k} = \beta_{k,k}, \quad \forall k = 1, \ldots, n \tag{5.33}$$

$$\sum_{i=l}^{n} h_i \gamma_{l,i} + \sum_{i=l+1}^{n} h_i \sum_{j=l}^{i-1} \delta_{l,i,j} \ \leq \ H\gamma_{l,l}, \quad \forall l = 1, \ldots, n-1 \tag{5.34}$$

$$\alpha_{j,i} + \gamma_{l,j} - 1 \leq \delta_{l,i,j} \leq (\alpha_{j,i} + \gamma_{l,j})/2,$$
$$\forall l = 1, \ldots, n-1, \ \forall i = l+1, \ldots, n, \ \forall j = l, \ldots, i-1 \tag{5.35}$$

$$\sum_{k=l+1}^{n} \gamma_{l,k} \leq (n-l)\gamma_{l,l}, \quad \forall l = 1, \ldots, n-1 \tag{5.36}$$

$$\alpha_{j,i} \in \{0,1\}, \ j = 1, \ldots, n, \ i = j, \ldots, n \tag{5.37}$$

$$\beta_{k,j} \in \{0,1\}, \ k = 1, \ldots, n, \ j = 1, \ldots, n \tag{5.38}$$

$$\gamma_{l,k} \in \{0,1\}, \ l = 1, \ldots, n, \ k = l, \ldots, n \tag{5.39}$$

$$\delta_{l,i,j} \in \{0,1\}, \ l = 1, \ldots, n-1, \ i = l+1, \ldots, n, \ j = l, \ldots, i-1 \tag{5.40}$$

The objective function (5.25) and constraints (5.26), (5.28), and (5.33) remain unchanged from the restricted model. Constraints (5.29) and (5.32) are also adopted, but some limits had to be modified in order to consider the additional $\beta_{k,j}$ variables. The other constraints are either new or have been substantially changed. Constraints (5.13) and (5.14) are replaced by (5.29), (5.30), and (5.31). Since it is no longer guaranteed that items are only assigned to a used stack j, inequalities (5.27) are introduced. Constraints (5.30) and (5.31) ensure that the height of each stack j never exceeds the height of the stripe k it is contained in (equal to stack k's height). We split these constraints into "strictly less" and "less than or equal to" constraints in order to avoid ambiguities when stacks have identical heights: The highest stack with the smallest index always determines the index k of the stripe. Inequalities (5.34) replace the height constraints for bins (5.17) and use expression (5.24) with the new variables $\delta_{l,i,j}$ for calculating a bin's used height. Constraints (5.35) force variables $\delta_{l,i,j}$ to be set to their intended values according to the definition in (5.23). Finally, inequalities (5.36) ensure that no stripes are packed into an unused bin. In total the ILP uses $O(n^3)$ variables and constraints, and is, according to our knowledge, the first polynomial-sized ILP for 3-stage 2BP.

5.5. A Column Generation Approach

As an alternative approach for 3-stage 2BP we propose a set covering formulation with column generation. The formulation is based on a Dantzig-Wolfe decomposition [126] of the ILP from the previous section and ideas from Gilmore and Gomory [45, 46] and Pisinger and Sigurd [98, 115]. For a recent survey about selected topics in column generation see Lübbecke and Desrosiers [80].

Dantzig-Wolfe decomposition is an effective technique for obtaining stronger models and reducing the symmetry of some LP models. It splits a suitable ILP model into a linear master problem and smaller subproblems. In our case, all the constraints restricting the assignment of items to stacks and the assignment of stripes to a bin form the detached subsystem. The constraints requiring that each item is packed once remain in the master problem, which is further reformulated into a set covering model.

5.5.1. The Set Covering Model for 3-Stage 2BP

The following set covering model can, in principle, be applied to any bin packing problem because specific geometric constraints concerning feasible layouts are encapsulated in the pricing (or column generation) sub-problem.

Let $\mathcal{P}$ be the set of all feasible packings of a single bin. The variable $x_p \in \{0,1\}$ indicates whether a packing $p \in \mathcal{P}$ appears in the solution ($x_p = 1$) or not ($x_p = 0$). For every item $i = 1, \ldots, n$ and every packing $p \in \mathcal{P}$, let the constant $A_i^p = 1$ iff packing p contains item i; otherwise $A_i^p = 0$. The set covering model for bin packing can now be stated as

$$\text{minimize} \quad \sum_{p \in \mathcal{P}} x_p \tag{5.41}$$

$$\text{subject to} \quad \sum_{p \in \mathcal{P}} x_p A_i^p \geq 1, \quad \forall i = 1, \ldots, n \tag{5.42}$$

$$x_p \in \{0,1\}, \quad \forall p \in \mathcal{P} \tag{5.43}$$

Due to inequalities (5.42), solutions with items appearing more than once are feasible. If items must appear exactly once as in the case of our 3-stage 2BP, duplicates can simply be removed from the solution without destroying its feasibility or optimality.

In general, $\mathcal{P}$ is too huge to be able to consider all variables x_p, $p \in \mathcal{P}$ explicitly. We therefore use delayed column generation to solve the linear programming (LP) relaxation of the set covering model, called the *master Problem* (MP). In this way, we do not explicitly consider the majority of the variables x_p. We start with a small set of initial patterns $\mathcal{P}' \subset \mathcal{P}$ taken from an initial feasible solution and solve the LP-relaxation of the problem restricted to $\mathcal{P}'$, the so-called *restricted master problem* (RMP). Based on the obtained solution, we search for a new pattern and its corresponding variable, whose inclusion in $\mathcal{P}'$ might allow for a better solution of the RMP. This extended LP is resolved and the whole process repeated until no further improvements are possible, and, therefore, an exact solution of the MP is obtained.

The *reduced costs* of a packing $p \in \mathcal{P}$ are

$$c_p^u = 1 - \sum_{i=1}^{n} A_i^p u_i, \tag{5.44}$$

where u_i are the dual variables from the solution of the RMP. Only variables with negative reduced costs can improve the current solution of the RMP, leading us to the challenge of finding such a variable/pattern.

Branching (described in section 5.6.2) becomes necessary if no further variables with negative reduced costs can be determined (i.e., the MP is solved to optimality) and the difference between the solution value of the MP and the value of the best integer solution so-far is greater than or equal to one, i.e., the optimality gap is greater than the granularity of the objective function.

5.5.2. The Pricing Problem

The pricing problem consists of finding a packing p with negative reduced costs c_p^u. The specific characteristics and constraints of the bin packing problem substantially determine this problem. Here, we consider the pricing problem for 3-stage 2BP, which is a *three-stage two-dimensional knapsack packing* (2DKP) problem with respect to profits u_i corresponding to the dual variable values of the current RMP solution. Based on the ILP for 3-stage 2BP presented in section 5.4.2, the pricing problem can be formulated as follows.

$$\text{maximize} \quad \sum_{i=1}^{n} u_i \sum_{j=1}^{i} \alpha_{j,i} \tag{5.45}$$

$$\text{subject to} \quad \sum_{j=1}^{i} \alpha_{j,i} \leq 1, \quad \forall i = 1, \ldots, n \tag{5.46}$$

$$\sum_{i=1}^{n} h_i \sum_{j=1}^{i} \delta_{i,j} \leq H \tag{5.47}$$

$$\alpha_{j,i} + \beta_{j,j} - 1 \leq \delta_{i,j} \leq (\alpha_{j,i} + \beta_{j,j})/2, \quad \forall i = 1, \ldots, n, \ \forall j = 1, \ldots, i \tag{5.48}$$

and the constraints
(5.27), (5.28), (5.29), (5.30), (5.31), and (5.32).

Variables $\alpha_{j,i}$ and $\beta_{k,j}$ have the same meaning as in the ILP of section 5.4.2; variables $\gamma_{l,k}$ are not needed anymore. The variables $\delta_{i,j} \in \{0,1\}$, $i = 1, \ldots, n$, $j = 1, \ldots, i$, replace $\delta_{l,i,j}$ and are set to one iff item i contributes to the total height of all used stripes, i.e., iff item i appears in stack j and stack j appears in stripe j. The correct values for $\delta_{i,j}$ are enforced by constraints (5.48). Constraints (5.46) ensure that no item is packed more than once. Constraints (5.47) limit the total height of all stripes to H.

5.5.3. Stabilizing Column Generation

In column generation a near-optimal solution is usually reached relatively quickly, but the closer to the optimum one gets, the smaller the progress per iteration becomes. The observation that dual variable values do not smoothly converge to their respective optima, but rather oscillate strongly is regarded as a major efficiency issue [80]. In order to reduce this effect, we try to stabilize the column generation process. We apply a stabilization approach using dual-optimal inequalities as suggested by

Ben Amor et al. [8] and adapt the dual *subset inequalities* (which are indeed dual optimal inequalities), introduced by Valério de Carvalho [124], to our needs.

To begin with we recall the main aspects of the subset inequalities for the cutting stock problem (CS) [124]. The CS consists of minimizing the number of stock elements needed of length L, such that all item demands b_i for each item i of length l_i are satisfied.

The master problem of CS is very similar to the one of 2BP. Here, $\mathcal{P}$ corresponds to the set of feasible cutting patterns for CS. The master problem (P_{CS}) and its dual (D_{CS}) are:

$$
\begin{aligned}
P_{CS}: \quad & \min \sum_{p \in \mathcal{P}} x_p && \\
& \sum_{p \in \mathcal{P}} x_p A_i^p \geq b_i \quad \forall i = 1, \ldots, n, \\
& x_p \geq 0 \qquad\qquad \forall p \in \mathcal{P}.
\end{aligned}
\qquad\qquad
\begin{aligned}
D_{CS}: \quad & \max \sum_{i=1}^{n} b_i u_i && \\
& \sum_{i=1}^{n} u_i A_i^p \leq 1 \qquad\qquad \forall p \in \mathcal{P} \\
& u_i \geq 0 \qquad\qquad \forall i = 1, \ldots, n.
\end{aligned}
$$

The main difference to 2BP lies in the pricing sub-problem which corresponds to a one-dimensional knapsack problem:

$$
\begin{aligned}
\max \quad & \sum_{i=1}^{n} u_i y_i \\
& \sum_{i=1}^{n} l_i y_i \leq L \\
& y_i \in \mathbb{N}, \forall i = 1, \ldots, n.
\end{aligned}
$$

Proposition 6 (Dual Subset Inequalities [8]) *For item $i \in I$ and subset $S \subset I$, any optimal solution $(u_i^* \; \forall i = 1, \ldots, n)$ for D_{CS} satisfies*

$$
l_i \geq \sum_{j \in S} l_j \rightarrow u_i^* \geq \sum_{j \in S} u_j^*.
$$

Since adding constraints (rows) to the dual problem corresponds to adding variables (columns) to the primal problem, the subset inequalities induce columns for the core master problem. For each inequality, a zero-cost variable $y_{i,S}$ with coefficients -1 in row i, +1 in rows $j \in S$, and 0 otherwise is introduced. This can be interpreted as replacing i with the items of S in any feasible pattern containing i. When a pattern contains item i and the corresponding column $y_{i,S}$, the pattern where i is replaced by S is implicitly considered and it is therefore not needed to generate this pattern.

Valério de Carvalho [124] keeps the number of variables used in $O(n)$, by using only those with $|S| = 1$ and some with $|S| = 2$. Reconstructing a primal optimal solution from the solution generated with the additional columns is straightforward: For any overcovering item i (i.e., i is used more than b_i-times), the overcovering patterns must be modified, replacing item i with the items of set S for which $y_{i,S} > 0$.

Dual Subset Inequalities for General 2BP

We adapt the idea of subset inequalities for the cutting stock problem, as described by Ben Amor et al. [8], to the general 2BP. The master problem for 2BP ($P_{2\text{BP}}$) and its dual ($D_{2\text{BP}}$) are:

$$P_{2\text{BP}}\colon \min \sum_{p\in\mathcal{P}} x_p \qquad \sum_{p\in\mathcal{P}} x_p A_i^p \geq 1, \quad \forall i = 1,\dots,n, \qquad x_p \geq 0, \quad \forall p \in \mathcal{P}.$$

$$D_{2\text{BP}}\colon \max \sum_{i=1}^{n} u_i \qquad \sum_{i=1}^{n} u_i A_i^p \leq 1, \quad \forall p \in \mathcal{P} \qquad u_i \geq 0, \quad \forall i = 1,\dots,n.$$

where $\mathcal{P}$ corresponds to the set of feasible patterns for a specific type of 2BP. Using this notation, we can now state the proposition defining the dual subset inequalities for 2BP.

Proposition 7 *For any item $i \in \{1,\dots,n\}$ and subset $S \subset \{1,\dots,n\}$, any optimal solution u^* to D_{2BP} satisfies*

$$u_i^* \geq \sum_{j\in S} u_j^*$$

if the items from S can be packed into a bin of size $h_i \times w_i$ and any pattern containing item i remains feasible when item i is replaced by an appropriate sub-pattern containing all items from S.

Proof: We adapt the proof by contradiction for the cutting stock problem given in [8]: Let (x^*, u^*) be a pair of primal-dual optimal solutions and assume $u_i^* < \sum_{j\in S} u_j^*$ for item i and subset S, but the items from S can be packed into a bin of size $h_i \times w_i$. We further assume that for any feasible pattern containing items $\{i\} \cup R$, there is a feasible pattern obtained by replacing i by S. These patterns define two dual constraints:

$$u_i^* + \sum_{j\in R} u_j^* \leq 1 \quad \text{and} \quad \sum_{j\in S} u_j^* + \sum_{j\in R} u_j^* \leq 1. \tag{5.49}$$

The assumption made above implies that

$$u_i^* + \sum_{j \in R} u_j^* < \sum_{j \in S} u_j^* + \sum_{j \in R} u_j^* \leq 1 \tag{5.50}$$

and that the left constraint of (5.49) cannot be active at optimality. The complementary slackness condition

$$x_p^*(1 - \sum_{i=1}^{n} u_i^* A_i^p) = 0 \quad \forall p \in \mathcal{P} \tag{5.51}$$

becomes

$$x_p^*(1 - (u_i^* + \sum_{j \in R} u_j^*) = 0 \tag{5.52}$$

for patterns containing item i. From inequalities (5.50) it follows that $1 - (u_i^* + \sum_{j \in R} u_j^*)$ is always greater than zero. Together with equation (5.52) this means that all primal variables corresponding to patterns containing item i must be equal to 0, which is a contradiction to the feasibility and optimality of x^*. □

This general form of subset inequalities can be used for nearly arbitrary variants of 2BP, since they only have different definitions of feasible packing patterns. In order to get dual constraints for these problems, fast preprocessing heuristics can be used to generate effective feasible subset inequalities.

Dual Constraints for 3-stage 2BP

In the previous section we devised a general form of subset inequalities which we now put in concrete form for the 3-stage 2BP problem. Every item i of a specific pattern can be replaced by a stack s of items having the same width as i and a height not exceeding that of i without breaking the feasibility of the 3-stage pattern. For an example see Figure 5.2.

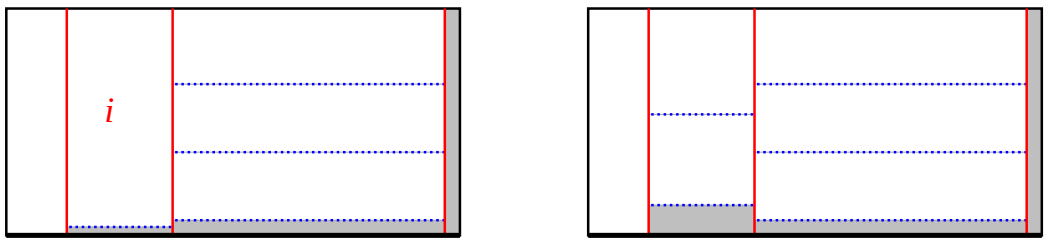

Figure 5.2.: Replacing an item i (on the left) by a stack (on the right).

We can therefore introduce the following dual constraints:

$$(\forall j \in S : w_i = w_j) \ \wedge \ (h_i \geq \sum_{j \in S} h_j) \quad \rightarrow \quad u_i^* \geq \sum_{j \in S} u_j^*, \qquad \forall i = 1, \ldots, n \quad (5.53)$$

Using (5.53), we can devise the following *type-1* constraints for $|S| = 1$:

$$w_i = w_j \quad \rightarrow \quad u_i^* \geq u_j^*, \qquad \forall i = 1, \ldots, n-1, \ \forall j = i+1, \ldots, n, \qquad (5.54)$$

based on the ordering of the items by nonincreasing heights.

Type-2 constraints for $|S| = 2$ are devised in a similar way:

$$w_i = w_j = w_k \ \wedge h_i \geq h_j + h_k \quad \rightarrow \quad u_i^* \geq u_j^* + u_k^* \qquad (5.55)$$
$$\forall i = 1, \ldots, n-2, \forall j = i+1, \ldots, n-1, \forall k = j+1, \ldots, n$$

As previously described, adding constraints (rows) to the dual problem corresponds to adding variables (columns) to the primal problem, the dual constraints induce columns for the master problem. The variables induced by the dual constraints are devised analogously to the cutting stock problem: For each constraint, a zero-cost variable $y_{i,S}$ with coefficients -1 in row i, $+1$ in rows $j \in S$, and 0 otherwise is introduced. This variable can be interpreted as an indicator for a substitution of item i with the items of S in any feasible pattern containing i. When a pattern contains item i and the corresponding column $y_{i,S}$, the pattern in which i is replaced by S is implicitly considered and therefore does not have to be created by column generation. Reconstructing a primal optimal solution from a solution and additional variables $y_{i,S}$ is straightforward [8]: for any overcovering item i (i.e., i appears more than once), the overcovering patterns must be modified by replacing item i with the items of set S for which $y_{i,S} > 0$.

5.6. The Branch-and-Price Framework

In order to obtain optimal integer solutions, we use a branch-and-price (B&P) framework. A general introduction to branch-and-price is given in Chapter 3 and a discussion of branching schemes is given in Barnhart et al. [7]. In the following, we present how to generate an initial feasible solution. Then we explain branching, which is, in general, necessary for solving the problem to integrality. Finally, we describe our hierarchical approach for solving the pricing problem.

Algorithm 8: Finite first fit (FFF) for 3-stage 2BP.

Input: Ordered list of items: *list*
while *list not empty* **do**
  b = new empty bin
  repeat
    s = new empty stripe in bin b
    forall *items* i *in list* **do**
      if $s == \emptyset \wedge i.h \leq b.uh$ **then**
        initialize s with i
        remove i from *list*
      else if $s \neq \emptyset \wedge i.w \leq s.uw \wedge i.h \leq s.h$ **then**
        add i to s
        remove i from *list*
        $uh = s.h - i.h$
        forall *items* j *with* $j > i$ **do**
          if $j.w == i.w \wedge i.h \leq uh$ **then**
            stack item j on i
            remove j from *list*
            $uh = uh - j.h$
    //no more item fitted into the bin
  until $s = \emptyset$
  discard empty stripe s
  add b to solution

Abbreviations: (used here and in the remainder of this chapter)
*$.h$: height of *
*$.w$: width of *
*$.uh$: unused height of *
*$.uw$: unused width of *

5.6.1. Generating an Initial Feasible Solution

In order to initialize the column generation algorithm, a feasible solution is needed. The packing patterns of its bins are used as initial $\mathcal{P}'$. A promising feasible solution can often be derived by considering the restricted 3-stage 2BP model from section 5.4.1 and trying to solve it using an ILP-solver with a given time limit. Another way of generating a feasible solution for 3-stage 2BP is the following order-based *finite first fit heuristic* (FFF). The heuristic gets an ordered list of all items as input. Its pseudocode is given in Algorithm 8.

As long as the ordered list of items is not empty, the algorithm fills one bin after the other. Variable b represents the current bin and variable s the current stripe. Stacks of items are added to the current stripe as long as it is possible. If no further item can be placed in the current stripe, a new empty stripe is added to the current bin. If no items fit into this stripe, it is discarded, bin b is closed, and a new bin is initialized. The described algorithm initializes the stripes with stacks containing a single item and therefore generates restricted 3-stage 2BP solutions only. The solutions obtained by FFF strongly depend on the input order of the items. Therefore, multiple runs using different orders are often meaningful.

The outer two loops run together in time $O(n)$, since either an item is placed in the inner loop, which reduces the size of the item list, or a new bin is created. The two inner loops have together a worst-case run-time of $O(n^2)$. The total worst-case effort of FFF for 3-stage 2BP is therefore $O(n^3)$.

5.6.2. Branching

If no further variables with negative reduced costs can be found by completely solving the pricing problem and the difference between the solution value of the RMP and the value of the best integer solution so-far exceeds one, branching becomes necessary. We use a branching rule similar to the one described in [98, 115]. The solution space is divided into two parts, where two different items i_1 and i_2 have to be in different bins or in the same bin respectively. We always choose the two highest items appearing in a pattern p whose corresponding variable x_p has an LP solution value closest to 0.5. Preliminary tests showed that other combinations of items were usually not better; even the use of strong branching [85] in order to choose between different branching candidates generated by diverse strategies did not improve the results obtained.

The first branch corresponds to adding the constraint

$$\sum_{p \in \mathcal{P}} x_p A^p_{i_1} A^p_{i_2} = 0, \tag{5.56}$$

the second branch corresponds to adding the two constraints

$$\sum_{p \in \mathcal{P}} x_p A^p_{i_1} (1 - A^p_{i_2}) = 0 \qquad \text{and} \qquad \sum_{p \in \mathcal{P}} x_p (1 - A^p_{i_1}) A^p_{i_2} = 0. \tag{5.57}$$

In an actual implementation it is not necessary to explicitly add constraints (5.56) and (5.57) to the RMP; instead, the variables violating the constraints are simply fixed to zero.

Furthermore, the following constraints have to be added to the subsequent pricing problems in order to guarantee that patterns violating the branching constraints are not generated anymore. In the first branch

$$\sum_{j=1}^{i_1} \alpha_{j,i_1} + \sum_{j=1}^{i_2} \alpha_{j,i_2} \leq 1, \tag{5.58}$$

and in the second branch

$$\sum_{j=1}^{i_1} \alpha_{j,i_1} = \sum_{j=1}^{i_2} \alpha_{j,i_2}. \tag{5.59}$$

In the sequel, we call i_1 and i_2 *conflicting* if constraint (5.58) is active, and say that i_1 *induces* i_2 and vice-versa if equation (5.59) is active.

5.6.3. Generating Columns using a Greedy Heuristic

Oliveira and Ferreira [93] suggested performing a fast column generation by first applying a heuristic to quickly obtain a promising pattern with negative, but not necessarily minimal, reduced costs. Only if this heuristic fails, the pricing problem is solved with an exact method. Such an approach can lead to faster overall column generation, since in general significantly fewer calls of the usually much slower exact algorithm are needed.

For the 3-stage 2BP we suggest the use of a four level hierarchy of pricing algorithms. Each of these algorithms searches for a variable with negative reduced costs, and if it fails, the next algorithm from the hierarchy is applied to the pricing problem. In each of these pricing iterations, a single variable is generated. The first level of the hierarchy is the greedy *first fit heuristic respecting branching constraints* (FFBC) shown in Algorithm 9.

FFBC considers the items in the order given by the parameter *list*. One item i after the other is packed into the first stack it fits. If the item does not fit into any existing stack, a new stack is created in the first stripe it fits. If no such stripe exists and there is enough space left in the bin, a new stack is created and packed into a new stripe. Otherwise the algorithm proceeds with the next item. If the addition of an item to a stack would increase the corresponding stripe's height, we check whether enough vertical space is left in the bin and actually add the item with a probability of 50%. The constraints resulting from branching are handled as follows. At the beginning, we recursively look for items induced by the current item i. If i and any of the induced items stay in conflict with any other induced or already packed item (checkAndFindInduced(i, *ind*) returns false), none of these

items is packed. Otherwise we immediately try to pack i and all the induced items returned by checkAndFindInduced(i, *ind*) in parameter *ind*. If this turns out to be impossible, we skip the whole chain of items.

Algorithm 9: First fit heuristic respecting branching constraints (FFBC)

```
Input: Ordered list of items: list, Bin: bin
forall i in list with u_i > 0 do
    if checkAndFindInduced(i, ind) then
        topack = {i} ∪ ind
        bin' = bin
        list = list − topack
        repeat
            choose j ∈ topack at random
            topack = topack − {j}
            if not pack(bin, j) then
                bin = bin'
                topack = ∅
        until topack = ∅
return bin
```

The outer loop of FFBC is performed $O(n)$ times. Procedure checkAndFindInduced() can be implemented in time $O(c)$, where c denotes the number of existing branching constraints. If there are induced items and not all of them could be packed together with i, it never tries to pack those items again. Therefore, the pack procedure is called only $O(n)$ times, at most once for each item in the list. The pack procedure runs in time $O(n)$, since there are at most $O(n)$ positions where an item can be placed. The worst-case total run-time of FFBC is therefore $O(n^2 + nc)$.

5.6.4. Generating Columns using an Evolutionary Algorithm

When the greedy FFBC heuristic fails to find a pattern with negative reduced costs, we apply a more sophisticated metaheuristic as a second-level pricing strategy. We decided to use an Evolutionary Algorithm (EA) operating directly on stripes, stacks, and items. Filho and Lorena [34] already successfully used an EA to generate columns for approximately solving graph-coloring problems.

Function `pack`(b, i)

forall *stripes* s *in* b **do**
  forall *stacks* a *in* s **do**
    if $w_i == a.w$ **then**
      if $h_i + a.h \leq s.h$ **then**
        pack i into a
        return *true*
      else if $a.h + h_i - s.h \leq b.uh$ **then**
        with *a probability of 50%* **do**
          pack i into a
          return *true*
forall *stripes* s *in* b **do**
  if $w_i \leq s.uw \wedge h_i \leq s.h$ **then**
    create stack a containing i
    pack a into s
    return *true*
  else if $w_i \leq s.uw \wedge h_i - s.h \leq b.uh$ **then**
    with *a probability of 50%* **do**
      create stack a containing i
      pack a into s
      return *true*
if $h_i \leq b.uh$ **then**
  create stack a containing i
  pack a into new stripe s
  pack s into b
  return *true*
return *false*

Structure of the Evolutionary Algorithm

We use a standard steady-state algorithm applying binary tournament selection with replacement and duplicate elimination, see e.g. [4, 86]. In each iteration, one new candidate solution is created by recombination of selected parents, and mutation is applied with a certain probability. The new candidate solution always replaces the worst solution in the population if it is not identical to an already existing solution.

Representation and Initialization

The chosen representation is direct: each candidate solution represents a bin as a set of stripes, each stripe as a set of stacks, and each stack as a set of item references, see Figure 5.3. Using such a hierarchy of sets makes it easy to ignore the order of items, stacks and stripes, and therefore to avoid symmetries.

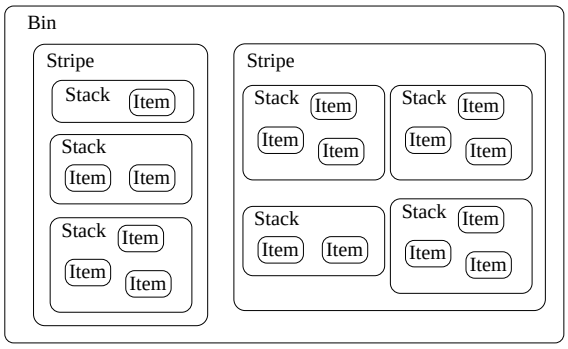

Figure 5.3.: Set-Representation of a solution (corresponds to pattern in Figure 5.1).

Initial solutions are created via the FFBC heuristic using randomly generated item orders as input. These orders are, however, created in a biased way by assigning each item i a random value $r_i \in [0, 1)$ and sorting the items according to decreasing $r_i u_i$. Only items with $u_i > 0$ are considered.

Recombination

The recombination operator, shown in Algorithm 10, first calculates the sum p_s of all contained items' values u_i and a random value $r_s \in (0, 1]$ for each stripe. Then the stripes are sorted according to decreasing $r_s p_s$. Thus, we obtain a random order

which is biased in a way so that stripes having higher total values are more likely to appear at the beginning. The stripes are then considered in this order and packed into the offspring's bin if they fit into it. Identical stripes of both parents appear twice in the ordered list but are considered only at their first appearance.

When all stripes have been processed, repairing is usually necessary in order to guarantee feasibility. First, the bin is traversed in order to delete item duplicates. Then, the branching constraints are considered: items staying in conflict with others (check(i) returns true) are removed. After this, we try to pack all induced items (returned by findInduced(i)); if this is not possible, the corresponding original items are also removed from the bin. Finally, FFBC is applied as local improvement operator to a list of items not yet present in the bin and having positive u_i. These items are sorted by assigning each item i a random value $r_i \in [0, 1)$ and sorting them according to decreasing $r_i u_i$.

The *forall*-loops are all traversed $O(n)$ times. Procedures check() and findInduced() run in $O(c)$ time, where c denotes the number of existing branching constraints. Procedure pack() is again called for each item at most once. The total run-time of stripe crossover is therefore bounded above by $O(n^2 + nc)$.

Mutation

The mutation operator removes a randomly chosen item i from the bin. If the branching constraints induce other items for i, they are also deleted. Finally, FFBC is used as local improvement operator, applied to a list of remaining items constructed in the same way as described for stripe crossover.

The mutation operator has a worst-case run-time of $O(n^2 + nc)$, since FFBC dominates the effort.

5.6.5. Pricing by Solving the Restricted 3-Stage 2DKP

Analogously to the restricted 3-stage 2BP, we can define a restricted version of our pricing problem, which we call *restricted 3-stage 2DKP*: The highest stack of each stripe may consist of a single item only. Formulating this restricted pricing problem as an ILP and trying to solve it using an ILP-solver is another heuristic approach for solving the general pricing problem. We apply this as third-level heuristic with a given time limit when the previous heuristics have failed. The ILP for restricted 3-stage 2DKP can be devised from the model (5.10) to (5.17), similar to the unrestricted case (see section 5.5.2):

Algorithm 10: StripeCrossover(A, B)

```
// crossover
forall stripes s in A and B do
    p_s = Σ_{i∈s} u_i
    r_s = random value ∈ [0, 1)
list = sort stripes by decreasing r_s p_s
b = new bin
forall stripes s in list do
    if s.h ≤ b.uh then
        copy s into b
// repairing
forall items i in b do
    if i appears in b twice then
        remove first i from b
forall items i in b do
    if check(i) then
        remove i from b
forall items i in b do
    ind =findInduced(i)
    if ind ≠ ∅ then
        b' = b
        topack = ind − b
        while topack ≠ ∅ do
            choose j ∈ topack at random
            topack = topack − {j}
            if not pack(b, j) then
                b = b'
                remove i and items in ind from b
//local improvement
ilist = {i = 1, ..., n | i ∉ b ∧ u_i > 0}
sort ilist by decreasing r_i u_i with r_i = random value ∈ [0, 1)
FFBC(ilist, b)
return b
```

$$\text{maximize} \quad \sum_{i=1}^{n} u_i \sum_{j=1}^{i} \alpha_{j,i} \tag{5.60}$$

$$\text{subject to} \quad \sum_{j=1}^{i} \alpha_{j,i} \leq 1, \quad \forall i = 1, \ldots, n \tag{5.61}$$

$$\sum_{k=1}^{n} h_k \beta_{k,k} \leq H \tag{5.62}$$

and the constraints:
(5.12), (5.13), (5.14), and (5.15).

5.6.6. Exact Pricing Algorithm

If all pricing heuristics failed to find a pattern with negative reduced costs, we use an ILP-solver in order to solve the ILP for the unrestricted 3-stage 2DKP from section 5.5.2. The optimization process is terminated—like the previous pricing heuristics—as soon as a pattern with negative reduced costs is found. Otherwise, the ILP-solver performs until it is proven that no such pattern exists. In this case, the solution of the RMP is also optimal for the MP and represents a valid lower bound for 3-stage 2BP.

5.7. Computational Experiments

The algorithms presented were implemented using GNU C++ 3.3.1 and the open-source library COIN [78], in particular the COIN/Bcp framework for branch-and-cut-and-price algorithms and COIN/Clp as LP-solver. Furthermore, CPLEX 8.1 was used with default parameters as a general purpose ILP-solver. All experiments were performed on a 2.8GHz Pentium 4 machine.

5.7.1. Settings and Parameters

A global time limit of 1 000s was given to each run for all the approaches we tested.

Initial feasible solutions are generated with the FFF heuristic described in section 5.6.1. FFF is called for $20n$ different item orders and the best obtained solution is used as the initial one. The first five item orders are determined by sorting the

items according to decreasing height, width, area, $2h_i + w_i$, and $h_i + 2w_i$, respectively; all further orders are random permutations. For some algorithm variants, CPLEX was additionally used to try to solve the restricted 3-stage 2BP model, with a time limit of 200s. The overall best solution is used as initial feasible solution.

For solving the pricing problem, FFBC (see Section 5.6.3) is applied up to 100 times using different item orders. The first five item orders are determined by sorting the items according to decreasing u_i, $\frac{u_i}{h_i \cdot w_i}$, $\frac{u_i}{h_i + w_i}$, $\frac{u_i}{h_i}$, and $\frac{u_i}{w_i}$, respectively; all further orders are random permutations.

The following EA settings were determined by preliminary experiments and turned out to be robust for many different problem instances: a population size of 100, binary tournament selection, a mutation probability of 0.75, and termination after 1 000 iterations without improvement of the so-far best solution or a total of 100 000 iterations.

If FFBC or the EA did not find a solution, CPLEX is applied to the restricted 3-stage 2DKP ILP model with a time limit of 100s. We denote this pricing method by *CPLEX(restricted 3-stage 2DKP)*. Finally, if still no variable with negative reduced costs could be devised, CPLEX is used to solve the unrestricted 3-stage 2DKP model, denoted by *CPLEX(3-stage 2DKP)*, respecting the global time limit.

In addition, the RMP is solved to integrality every M-th iteration by using CPLEX, possibly providing a new incumbent solution. $M = 100$ turned out to be a reasonable choice. The RMP is further solved to optimality before branching, because preliminary experiments showed that further branching can sometimes be avoided when a better upper bound is available.

5.7.2. Computational Results

In order to evaluate the effectiveness of the different models and algorithms described, we compare the following approaches:

2LBP:
CPLEX applied to the ILP for 2-stage 2BP from [76]

R2BP:
CPLEX applied to the ILP for restricted 3-stage 2BP

2BP:
CPLEX applied to the ILP for 3-stage 2BP
initialization: FFF and R2BP

EA:
The evolutionary algorithm from [105] applied to 3-stage 2BP; the specific variant used was EAe OX3,RX according to the notation in this article.

GuillSig:
B&P with constraint programming applied to guillotineable 2BP
Results adopted from [115] (run-time limited to 3 600s)

BPNoR:
B&P applied to 3-stage 2BP
initialization: FFF
pricing: FFBC, CPLEX(3-stage 2DKP)

BP:
B&P applied to 3-stage 2BP
initialization: FFF and R2BP
pricing: FFBC, CPLEX(restricted 3-stage 2DKP), CPLEX(3-stage 2DKP)

BPStab:
B&P applied to 3-stage 2BP
initialization: FFF and R2BP
pricing: FFBC, CPLEX(restricted 3-stage 2DKP), CPLEX(3-stage 2DKP)
stabilization: type 1 and type 2 constraints

BPStabEA:
B&P applied to 3-stage 2BP
initialization: FFF and R2BP
pricing: FFBC, EA, CPLEX(restricted 3-stage 2DKP), CPLEX(3-stage 2DKP)
stabilization: type 1 and type 2 constraints

The instances used for the experiments [2] are adopted from Lodi et al. [76] and Pisinger and Sigurd [98, 115]. We use the class numbering from [98, 115]. The first six instance classes have the following characteristics:

Class 1: h_i and w_i uniformly random in $[1, 10]$, $W = H = 10$.
Class 2: h_i and w_i uniformly random in $[1, 10]$, $W = H = 30$.
Class 3: h_i and w_i uniformly random in $[1, 35]$, $W = H = 40$.
Class 4: h_i and w_i uniformly random in $[1, 35]$, $W = H = 100$.
Class 5: h_i and w_i uniformly random in $[1, 100]$, $W = H = 100$.
Class 6: h_i and w_i uniformly random in $[1, 100]$, $W = H = 300$.

In the last four classes, $W = H = 100$ and four types of items are used:

[2] `http://www.or.deis.unibo.it/research_pages/ORinstances/ORinstances.htm`

Type 1: w_i uniformly random in $[\frac{2}{3}W, W]$, h_i uniformly random in $[1, \frac{1}{2}H]$
Type 2: w_i uniformly random in $[1, \frac{1}{2}W]$, h_i uniformly random in $[\frac{2}{3}H, H]$
Type 3: w_i uniformly random in $[\frac{1}{2}W, W]$, h_i uniformly random in $[\frac{1}{2}H, H]$
Type 4: w_i uniformly random in $[1, \frac{1}{2}W]$, h_i uniformly random in $[1, \frac{1}{2}H]$

The instances classes are:

Class 7: type 1 with prob. 0.7, types 2, 3, and 4 with prob. 0.1 each.
Class 8: type 2 with prob. 0.7, types 1, 3, and 4 with prob. 0.1 each.
Class 9: type 3 with prob. 0.7, types 1, 2, and 4 with prob. 0.1 each.
Class 10: type 4 with prob. 0.7, types 1, 2, and 3 with prob. 0.1 each.

Each class has five sub-classes having $n = 20$, 40, 60, 80, and 100 items, and 10 instances exist in each sub-class. We therefore have a set of 500 instances in total.

Table 5.1 shows average lower bounds obtained from the LP-relaxations of the ILP models. Lower bound L_0 is the continuous lower bound as defined in [76]:

$$L_0 = \left\lceil \frac{\sum_{i=1}^{n} w_i h_i}{WH} \right\rceil \tag{5.63}$$

For a few instances, some column generation approaches did not terminate within the time limit of 1 000s and, therefore, no valid lower bounds could be found. In these cases, the values from L_0 are adopted.

The 2LBP and the R2BP bounds dominate L_0. While the ILP for restricted 3-stage 2BP gives relatively good lower bounds compared to the other methods presented, the lower bound derived from the unrestricted 3-stage 2BP model is generally poor. This comes mainly from the $O(n^3)$ $\delta_{l,i,j}$ variables and the relatively weak constraints (5.27) and (5.36). The lower bounds of the column generation models dominate the others. One can observe that adding CPLEX(restricted 3-stage 2DKP) as additional pricing heuristic slightly improves the lower bounds, since column generation was able to terminate within the 1 000s time limit more often.

Results of the all computational experiments are presented in Table 5.2 and Table 5.3. For each sub-class, the average objective value of the final best feasible solution ($\overline{z}$), the number of times the algorithm could prove optimality of a solution (Opt), and the average computation time ($\overline{t}[s]$) are presented (the Opt column is omitted for the EA, since it is a heuristic method). The total averages are also given for every algorithm.

In general, differences in the objective values of final solutions obtained by the different optimization approaches are relatively small. One reason is the granularity

Table 5.1.: Lower bounds.

Class	n	L_0	2LBP	R2BP	2BP	BPNoR	BP	BPStab	BPStabEA
	20	6.4	6.4	6.4	3.7	7.0	7.0	7.0	7.0
	40	12.0	12.1	12.0	6.3	13.4	13.4	13.4	13.4
1	60	18.5	18.5	18.5	9.8	20.0	20.0	20.0	20.0
	80	25.3	25.3	25.3	13.1	27.5	27.5	27.5	27.5
	100	30.5	30.5	30.5	14.7	31.5	31.5	31.5	31.5
	20	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
	40	1.9	2.0	1.9	1.0	1.9	1.9	1.9	1.9
2	60	2.5	2.5	2.5	1.0	2.5	2.5	2.5	2.5
	80	3.1	3.1	3.1	1.0	3.1	3.1	3.1	3.1
	100	3.9	3.9	3.9	1.0	3.9	3.9	3.9	3.9
	20	4.4	4.4	4.4	2.4	5.4	5.4	5.4	5.4
	40	8.2	8.4	8.2	3.7	9.7	9.7	9.7	9.7
3	60	12.5	12.7	12.5	5.4	13.9	14.0	14.0	14.0
	80	17.3	17.3	17.3	7.1	18.9	19.2	19.0	19.2
	100	20.5	20.7	20.5	7.9	21.6	21.8	22.0	21.8
	20	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
	40	1.9	1.9	1.9	1.0	1.9	1.9	1.9	1.9
4	60	2.3	2.5	2.3	1.0	2.3	2.3	2.3	2.3
	80	3.0	3.1	3.0	1.0	3.0	3.0	3.0	3.0
	100	3.7	3.7	3.7	1.0	3.7	3.7	3.7	3.7
	20	5.4	5.5	5.4	2.9	6.6	6.6	6.6	6.6
	40	10.1	10.4	10.4	5.2	12.3	12.3	12.3	12.3
5	60	15.7	15.8	15.7	8.1	18.3	18.3	18.3	18.3
	80	21.5	21.6	21.5	10.8	24.7	24.7	24.7	24.7
	100	25.9	26.1	26.0	12.7	27.7	28.5	28.5	28.5
	20	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
	40	1.5	1.8	1.7	1.0	1.5	1.5	1.5	1.5
6	60	2.1	2.1	2.1	1.0	2.1	2.1	2.1	2.1
	80	3.0	3.0	3.0	1.0	3.0	3.0	3.0	3.0
	100	3.2	3.2	3.2	1.0	3.2	3.2	3.2	3.2
	20	4.7	4.9	4.7	3.3	5.7	5.7	5.7	5.7
	40	9.7	9.9	9.7	5.9	11.5	11.5	11.5	11.5
7	60	14.0	14.1	14.0	7.8	16.1	16.1	16.1	16.1
	80	19.7	20.0	19.7	10.5	23.2	23.2	23.2	23.2
	100	23.8	23.8	23.8	11.8	27.1	27.1	26.4	27.1
	20	4.8	5.0	5.0	3.5	6.1	6.1	6.1	6.1
	40	9.6	9.8	9.8	7.0	11.4	11.4	11.4	11.4
8	60	14.1	14.3	14.3	10.3	16.4	16.4	16.4	16.4
	80	19.5	19.7	19.6	13.7	22.6	22.6	22.6	22.6
	100	24.1	24.1	24.1	16.8	28.0	28.0	28.0	28.0
	20	9.4	9.4	9.4	7.6	14.3	14.3	14.3	14.3
	40	18.0	18.2	18.1	14.7	27.8	27.8	27.8	27.8
9	60	27.6	27.9	27.9	22.9	43.7	43.7	43.7	43.7
	80	37.1	37.3	37.2	29.4	57.7	57.7	57.7	57.7
	100	45.0	45.1	45.1	35.9	69.4	69.4	69.4	69.4
	20	3.8	4.1	4.1	1.4	4.5	4.5	4.5	4.5
	40	6.9	7.1	7.1	2.3	7.6	7.6	7.6	7.6
10	60	9.4	9.9	9.7	2.9	9.8	10.2	10.1	10.2
	80	12.2	12.3	12.2	3.4	12.2	12.2	12.2	12.3
	100	15.3	15.5	15.3	3.6	15.3	15.3	15.3	15.3
	Average	11.96	12.08	12.01	6.87	14.48	14.52	14.50	14.52

of the objective function values (objective values are always integer). Another reason is that we only compared rather sophisticated approaches, which usually either find optimal solutions or solutions needing only one more bin than the optimum.

Table 5.2 shows the results of the first four algorithms. In the case of 2-stage 2BP, CPLEX was able to solve 95.2% of the instances to provable optimality, requiring 14.75 bins and 62.58s on average. The results computed for restricted 3-stage 2BP are slightly better in terms of the number of bins needed (14.68), but slightly worse considering the percentage of proven optimality (94.8%) and the average run-time (68.46s). The restricted 3-stage model yields a lower average number of bins needed for every class, except for class 9 where the same number of bins is needed for 2-stage, 3-stage, and general guillotineable 2BP (see column GuillSig). When CPLEX is directly applied on the unrestricted 3-stage 2BP model without initialization, no solutions were found for some instances. We therefore used the same initialization as for the B&P algorithms, namely FFF in combination with CPLEX applied to the restricted 3-stage 2BP model. The average number of bins needed has slightly increased (14.69); furthermore, an increase of the average run-time (282.62s) and a decrease in the number of proven optimal solutions (71.6%) can be observed. The EA described in [105] needs 14.84 bins on average, and solves the instances within an average run-time of 28.86 seconds. This is the fastest of the considered algorithms, but its solution quality is the worst. The B&P algorithm of Pisinger and Sigurd applied to general guillotineable 2BP [98, 115] yields a lower average number of bins needed (14.53), but the number of proven optimal solutions is relatively small (75.4%); furthermore the algorithm was given a global time limit of 3 600s.

Table 5.3 shows the results of the four B&P algorithms. The columns BPNoR and BP document the importance and the effectiveness of using the restricted 3-stage model inside the B&P algorithm: the average number of bins needed decreased from 14.74 to 14.67, and the percentage of optimally solved instances increased from 83.2% to 87.4%. Adding the dual subset inequalities further improves the results regarding the total number of bins needed and the run-time. Finally, using the EA and the stabilization yields the best results, the average number of bins needed is 14.65 and 87.8% of the instances have been solved to proven optimality. The run-time is, on average 160.68s, which is the fastest of our four B&P variants.

In Table 5.4, the total number of pricing problems solved in each class as well as their sums, is given for the B&P approaches. Furthermore, the bar charts shown in Figure 5.4 visualize how often— in relation to the total number of pricing problems solved—the different pricing algorithms were successful for each class. The origin of the charts was shifted to 0.5 because for almost all the variants, the greedy FFBC algorithm solved more than half of the pricing problems.

Table 5.2.: Experimental results 1.

		2LBP			R2BP			2BP			EA		GuillSig		
Class	n	$\overline{z}$	*Opt*	$\overline{t}$ [s]	$\overline{z}$	*Opt*	$\overline{t}$ [s]	$\overline{z}$	*Opt*	$\overline{t}$ [s]	$\overline{z}$	$\overline{t}$ [s]	$\overline{z}$	*Opt*	$\overline{t}$ [s]
	20	7.3	10	0.0	7.2	10	0.0	7.2	10	0.0	7.3	6.009	7.1	10	2.5
	40	13.8	10	0.2	13.7	10	0.1	13.6	7	356.6	13.8	11.074	13.4	10	7.9
1	60	20.3	10	0.3	20.1	10	0.4	20.1	3	701.2	20.4	16.598	20.0	10	422.2
	80	27.7	10	0.6	27.5	10	0.3	27.5	0	999.5	27.9	22.744	27.6	9	394.0
	100	32.4	10	0.9	31.8	10	23.1	31.8	3	693.1	32.6	40.732	31.9	8	936.5
	20	1.0	10	0.0	1.0	10	0.0	1.0	10	0.0	1	7.043	1.0	10	0.8
	40	2.0	10	0.2	2.0	10	0.2	2.0	9	100.1	2	11.999	2.0	10	291.8
2	60	2.8	9	100.9	2.6	9	191.5	2.7	8	160.6	2.8	16.862	2.6	6	1460.2
	80	3.3	10	97.6	3.3	8	212.8	3.3	8	162.3	3.4	27.44	3.3	1	3241.8
	100	4.1	9	134.8	4.0	9	173.9	4.1	8	166.5	4.1	39.825	4.0	3	2529.3
	20	5.4	10	0.0	5.4	10	0.0	5.4	10	0.1	5.4	6.066	5.1	10	4.2
	40	9.8	10	0.1	9.8	10	0.2	9.7	10	14.3	9.8	11.359	9.4	9	383.6
3	60	14.0	10	2.4	14.0	10	2.7	14.0	7	454.6	14.5	21.586	13.9	9	699.2
	80	19.7	10	3.8	19.4	10	4.6	19.4	0	995.2	19.8	38.679	19.0	9	584.9
	100	22.8	9	134.4	22.8	9	135.7	22.9	0	952.0	23.4	57.031	22.4	7	1173.0
	20	1.0	10	0.0	1.0	10	0.0	1.0	10	0.0	1	8.522	1.0	10	0.0
	40	2.0	10	0.2	2.0	10	1.3	2.0	9	99.0	2	15.95	1.9	10	55.4
4	60	2.7	8	222.2	2.5	10	36.1	2.6	7	279.9	2.9	24.74	2.5	4	2198.8
	80	3.4	9	103.7	3.4	7	304.3	3.4	6	340.2	3.4	36.744	3.3	0	3600.0
	100	4.2	6	403.0	4.2	5	508.3	4.2	5	401.1	4.2	56.44	3.8	3	2557.4
	20	6.7	10	0.0	6.7	10	0.0	6.6	10	0.0	6.7	7.73	6.5	10	1.3
	40	12.3	10	0.2	12.3	10	0.2	12.3	10	2.1	12.3	11.936	11.9	10	14.4
5	60	18.3	10	0.4	18.3	10	0.4	18.3	10	12.4	18.4	21.688	18.0	10	343.7
	80	25.0	10	3.4	24.9	10	3.2	24.8	8	309.5	25.1	41.331	24.7	9	745.3
	100	28.8	10	66.0	28.7	10	155.4	28.8	1	836.9	29.4	65.481	28.2	7	1295.5
	20	1.0	10	0.0	1.0	10	0.0	1.0	10	0.0	1	9.771	1.0	10	0.1
	40	1.9	10	0.3	1.9	10	0.4	1.9	10	17.6	1.9	18.406	1.9	8	1271.8
6	60	2.3	9	100.6	2.2	10	14.3	2.2	9	99.9	2.3	29.071	2.2	6	1813.7
	80	3.0	10	1.8	3.0	10	3.5	3.0	10	0.4	3	44.845	3.0	6	1499.0
	100	3.7	7	304.2	3.7	7	315.8	3.7	5	430.6	3.6	64.404	3.4	0	3600.0
	20	5.7	10	0.0	5.7	10	0.0	5.7	10	0.0	5.7	5.647	5.5	10	0.8
	40	11.5	10	0.2	11.5	10	0.1	11.5	9	160.0	11.5	19.644	11.1	8	1080.8
7	60	16.2	10	3.7	16.1	10	1.8	16.1	1	997.5	16.3	29.827	15.8	7	1083.0
	80	23.3	10	5.3	23.2	10	2.8	23.2	0	996.8	23.3	49.448	23.2	2	2881.3
	100	27.6	9	117.5	27.4	8	276.2	27.5	0	923.2	27.6	93.332	27.2	7	1102.3
	20	6.1	10	0.0	6.1	10	0.0	6.1	10	0.0	6.1	5.918	5.8	10	13.9
	40	11.5	10	1.1	11.4	10	1.6	11.4	10	0.0	11.8	8.24	11.3	9	721.8
8	60	16.4	10	2.0	16.4	10	7.0	16.4	10	0.1	16.5	13.927	16.1	8	1443.6
	80	22.7	9	139.8	22.6	10	40.1	22.6	10	4.5	23.1	20.227	22.4	9	369.7
	100	28.2	9	140.7	28.2	9	142.7	28.2	9	128.1	28.5	30.917	27.9	5	1819.0
	20	14.3	10	0.0	14.3	10	0.0	14.3	10	0.0	14.3	11.982	14.3	10	0.1
	40	27.8	10	0.0	27.8	10	0.0	27.8	10	0.0	27.8	16.976	27.8	10	0.6
9	60	43.7	10	0.0	43.7	10	0.0	43.7	10	0.1	43.7	22.408	43.7	10	2.2
	80	57.7	10	0.0	57.7	10	0.1	57.7	10	0.2	57.7	26.919	57.7	10	5.3
	100	69.5	10	0.1	69.5	10	0.1	69.5	10	0.3	69.5	43.266	69.5	10	16.0
	20	4.5	10	0.0	4.5	10	0.0	4.5	10	0.0	4.5	6.243	4.2	10	8.5
	40	7.7	10	1.8	7.7	10	1.8	7.7	9	123.8	7.7	16.222	7.4	9	383.0
10	60	10.5	9	117.7	10.4	9	101.3	10.4	6	512.1	10.5	27.519	10.1	7	1139.1
	80	13.5	7	327.5	13.2	8	229.3	13.2	1	832.6	13.7	41.058	13.1	2	2921.7
	100	16.4	7	589.5	16.4	6	529.3	16.5	0	866.0	16.8	61.214	16.5	0	3600.0
	Average	14.75	9.52	62.58	14.68	9.48	68.46	14.69	7.16	282.62	14.84	26.86	14.53	7.54	994.42

Table 5.3.: Experimental results 2.

		BPNoR			BP			BPStab			BPStabEA		
Class	n	$\overline{z}$	Opt	$\overline{t}$ [s]	$\overline{z}$	Opt	$\overline{t}$ [s]	$\overline{z}$	Opt	$\overline{t}$ [s]	$\overline{z}$	Opt	$\overline{t}$ [s]
	20	7.2	10	0.3	7.2	10	0.3	7.2	10	0.6	7.2	10	4.2
	40	13.6	8	202.8	13.6	8	202.8	13.6	8	201.1	13.6	8	201.5
1	60	20.1	9	201.5	20.1	9	119.7	20.1	9	113.4	20.1	9	112.6
	80	27.5	10	50.7	27.5	10	43.4	27.5	10	53.4	27.5	10	68.6
	100	32.2	5	623.6	31.7	8	244.9	31.7	8	244.2	31.7	8	236.9
	20	1.0	10	0.1	1.0	10	0.0	1.0	10	0.1	1.0	10	0.1
	40	2.0	9	100.4	2.0	9	100.4	2.0	9	100.4	2.0	9	100.5
2	60	2.8	7	301.4	2.7	8	207.3	2.7	8	207.1	2.7	8	207.1
	80	3.4	7	303.8	3.3	8	228.1	3.3	8	228.6	3.3	8	228.1
	100	4.1	8	210.1	4.1	8	239.5	4.1	8	240.0	4.1	8	239.7
	20	5.4	10	0.2	5.4	10	0.2	5.4	10	0.2	5.4	10	0.3
	40	9.7	10	34.8	9.7	10	12.8	9.7	10	11.4	9.7	10	6.1
3	60	14.0	9	237.7	14.0	10	63.7	14.0	10	51.8	14.0	10	45.2
	80	19.3	8	397.4	19.2	10	174.2	19.3	9	197.2	19.2	10	166.8
	100	23.2	3	781.7	22.8	4	669.4	22.5	5	661.1	22.5	4	651.4
	20	1.0	10	0.1	1.0	10	0.1	1.0	10	0.1	1.0	10	0.1
	40	2.0	9	100.4	2.0	9	100.4	2.0	9	100.4	2.0	9	100.5
4	60	2.7	6	401.0	2.6	7	339.4	2.6	7	338.8	2.6	7	339.2
	80	3.3	7	302.8	3.3	7	321.4	3.3	7	321.3	3.3	7	321.5
	100	4.0	7	306.7	4.0	7	353.1	4.0	7	352.6	4.0	7	352.8
	20	6.6	10	0.2	6.6	10	0.2	6.6	10	0.2	6.6	10	0.5
	40	12.3	10	3.3	12.3	10	3.3	12.3	10	2.5	12.3	10	3.0
5	60	18.3	10	17.7	18.3	10	10.6	18.3	10	9.7	18.3	10	10.2
	80	24.8	9	138.9	24.8	9	128.1	24.8	9	127.9	24.8	9	129.7
	100	28.8	5	587.4	28.7	9	364.0	28.7	9	335.8	28.7	9	326.8
	20	1.0	10	0.1	1.0	10	0.0	1.0	10	0.0	1.0	10	0.0
	40	1.9	6	400.5	1.9	6	400.5	1.9	6	400.5	1.9	6	400.9
6	60	2.3	8	201.0	2.2	9	118.3	2.2	9	118.4	2.2	9	118.2
	80	3.0	10	3.1	3.0	10	13.9	3.0	10	13.9	3.0	10	14.0
	100	3.6	6	405.2	3.6	6	431.6	3.6	6	431.5	3.6	6	431.6
	20	5.7	10	0.4	5.7	10	0.3	5.7	10	0.3	5.7	10	0.6
	40	11.5	10	4.5	11.5	10	4.1	11.5	10	3.9	11.5	10	4.8
7	60	16.1	10	28.0	16.1	10	25.5	16.1	10	20.6	16.1	10	22.9
	80	23.2	10	57.3	23.2	10	80.6	23.2	10	79.3	23.2	10	77.8
	100	27.1	10	302.7	27.1	10	349.9	27.1	8	448.0	27.1	10	305.5
	20	6.1	10	0.8	6.1	10	0.9	6.1	10	0.9	6.1	10	1.0
	40	11.4	10	9.7	11.4	10	6.9	11.4	10	6.1	11.4	10	6.7
8	60	16.4	10	17.6	16.4	10	39.3	16.4	10	30.9	16.4	10	30.0
	80	22.6	10	94.8	22.6	10	106.1	22.6	10	79.7	22.6	10	79.5
	100	28.1	9	334.1	28.2	8	332.6	28.1	9	215.2	28.1	9	215.5
	20	14.3	10	0.1	14.3	10	0.0	14.3	10	0.0	14.3	10	0.1
	40	27.8	10	0.4	27.8	10	0.2	27.8	10	0.2	27.8	10	0.3
9	60	43.7	10	1.4	43.7	10	0.8	43.7	10	0.7	43.7	10	0.9
	80	57.7	10	3.6	57.7	10	2.4	57.7	10	2.3	57.7	10	2.6
	100	69.5	10	8.0	69.5	10	6.4	69.5	10	6.5	69.5	10	7.1
	20	4.5	10	0.9	4.5	10	0.5	4.5	10	0.6	4.5	10	0.4
	40	7.7	9	152.5	7.7	9	111.0	7.7	9	111.2	7.7	9	109.6
10	60	10.8	2	846.3	10.4	8	488.4	10.4	7	462.0	10.4	8	461.6
	80	13.9	0	1 000.0	13.2	1	902.5	13.2	1	902.5	13.2	2	889.1
	100	16.9	0	1 000.0	16.6	0	1 000.0	16.4	0	1 000.0	16.4	0	1 000.0
	Average	14.72	8.32	203.56	14.67	8.74	167.00	14.66	8.70	164.71	14.65	8.78	160.68

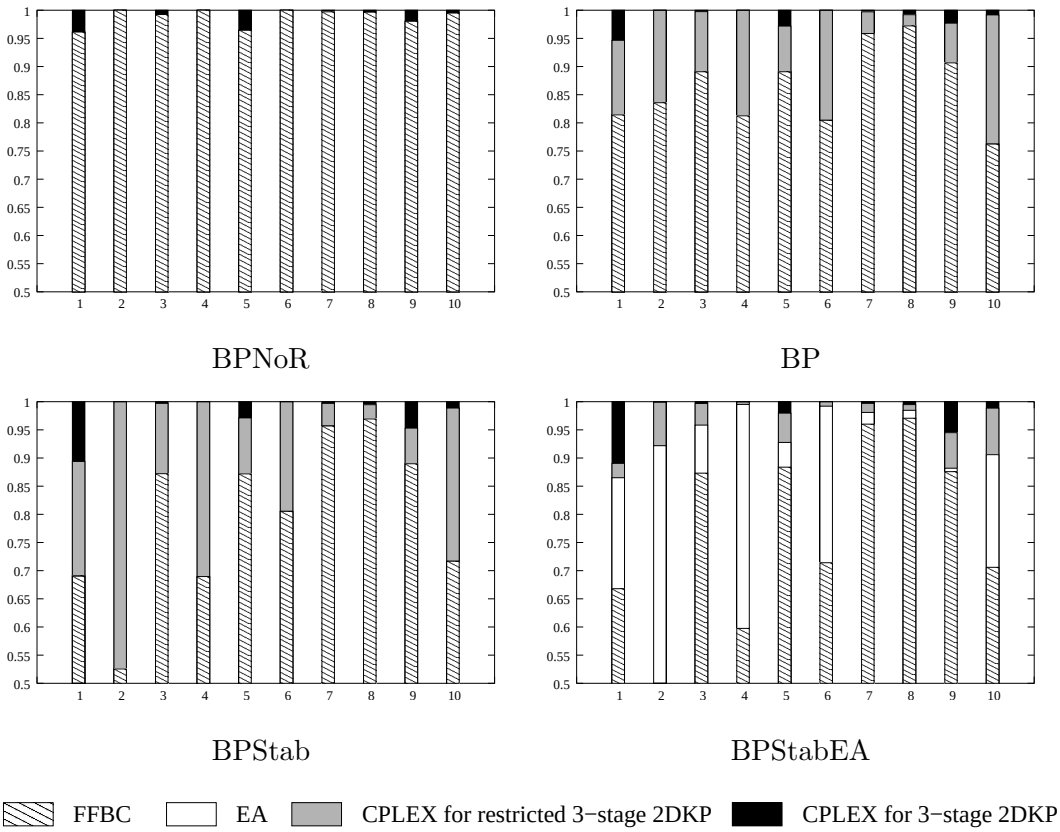

Figure 5.4.: Relative success rates of the four pricing algorithms.

Table 5.4.: Number of solved pricing problems.

	BPNoR	BP	BPStab	BPStabEA
Class 1	12 049	12 130	7 536	7 706
Class 2	7 366	9 465	2 980	3 381
Class 3	16 988	16 552	13 531	13 257
Class 4	12 785	17 451	13 821	14 342
Class 5	15 455	15 612	14 188	13 503
Class 6	15 175	15 397	14 429	14 586
Class 7	17 334	16 962	16 228	16 429
Class 8	15 081	14 815	11 067	10 742
Class 9	658	567	490	491
Class 10	26 863	35 047	31 752	32 194
Total	139 754	153 998	126 022	126 631

In Table 5.4, we can observe that when using FFBC together with CPLEX(3-stage 2DKP) only (BPNoR), the number of pricing problems solved is lower than the one of BP where CPLEX(restricted 3-stage 2DKP) is additionally used. Pricing using a more sophisticated heuristic, in this case exactly solving restricted 3-stage 2DKP, can therefore improve the overall results, see also Table 5.3. Furthermore, the stabilizing inequalities reduce the number of pricing problems to be solved and, therefore, achieve the goal of stabilizing the column generation process. Adding the EA for column generation does not significantly change the total number of pricing problems solved.

The bar charts from Figure 5.4 can be used to draw some conclusions about the characteristics of the pricing problems and the algorithms used to solve them. In general, we can observe that in every approach each implemented pricing algorithm solves a significant number of pricing problems; thus, no pricing algorithm is obviously obsolete. FFBC definitely solves the majority of pricing problems in all cases. These pricing problems can be denoted as "easy". Looking at absolute numbers shows that CPLEX(restricted 3-stage 2DKP) successfully solved 21 500 pricing problems, which approximately corresponds to the increase of solved pricing problems when BPNoR is compared to BP in Table 5.4. The bar charts showing the relative success rates of the pricing algorithms indicate that the absolute number of "easy" pricing problems roughly remained the same. When stabilization is applied (BPStab), the total number of pricing problems solved is reduced; the absolute number of "easy" pricing problems decreased, whereas the absolute number of "harder" pricing problems remained approximately the same: CPLEX(restricted 3-stage 2DKP) successfully

solved 22 751 pricing problems. Adding the EA (BPStabEA) does not significantly change the ratio of "easy" to "harder" pricing problems. A substantial number of those "harder" problems are solved by the EA (21 501), whereas CPLEX(restricted 3-stage 2DKP) is successful 4 915 times. Note that the total number of problems solved using CPLEX(3-stage 2DKP) is approximately the same for BP, BPStab, and BPStabEA: 1 602, 1 733, and 1 665 respectively.

5.8. Conclusions

We developed two polynomial-sized ILP models for 3-stage 2BP; a restricted model and an unrestricted one. The restricted model is particularly useful for obtaining near-optimal solutions to 3-stage 2BP quickly. Solving the unrestricted model is computationally more expensive.

Further, a branch-and-price algorithm based on a set covering formulation for 2BP was proposed. This B&P algorithm was enhanced by dual subset inequalities stabilizing the column generation process. Column generation is performed by applying a hierarchy of up to four pricing methods having specific advantages and disadvantages: FFBC is the fastest, but can only solve "easy" pricing problems; the EA is slower, but is able to solve "harder" pricing problems in an efficient way. CPLEX(restricted 3-stage 2DKP) solves a restricted form of the pricing problem to proven optimality; finally, CPLEX(3-stage 2DKP) solves the unrestricted pricing problem to optimality, but is most time-consuming.

The ILPs for restricted and unrestricted 3-stage 2DKP were derived from the corresponding 3-stage 2BP models and proved to be efficient and very useful in the context of column generation.

We performed extensive computational experiments on standard benchmark instances in order to analyze the performance of the models and algorithms developed here. The lower bounds obtained by column generation are strong. The best average results were achieved by B&P with all the proposed enhancements, in particular the four-level pricing strategy. These are, to our knowledge, the best known results for the 3-stage two-dimensional bin packing problem.

More generally, column generation performed by using a hierarchy of smart heuristics, also including metaheuristics such as evolutionary algorithms and exact algorithms, can significantly improve the optimization speed and the capabilities of branch-and-price in finding optimal or near-optimal solutions.

Chapter 6

The Multidimensional Knapsack Problem

In this chapter, we will study the multidimensional knapsack problem, present some theoretical and empirical results about its structure, and evaluate different ILP-based, metaheuristic, and collaborative approaches for it.

We will first give a short introduction to the multidimensional knapsack problem, followed by an empirical analysis of widely used benchmark instances. Firstly the distances between optimal solutions to the LP-relaxation and the original problem are studied. Secondly we introduce the new core concept for the MKP, which we study extensively. The empirical analysis is then used to develop new concepts for solving the MKP using ILP-based and memetic algorithms. We then describe the newly developed Relaxation Guided Variable Neighborhood Search in general, and its implementation for the MKP in particular. Different collaborative combinations of the presented algorithms are discussed and evaluated. Further computational experiments with longer run-times are also performed in order to compare the solutions of our approaches to the best known solutions for the MKP. Finally, we conclude with a summary of the developed approaches and an outlook for future work.

Parts of this chapter have been published in [103, 104, 106].

6.1. Introduction

The Multidimensional Knapsack Problem (MKP) is a well-studied, strongly NP-hard combinatorial optimization problem occurring in many different applications. It can be defined by the following ILP:

$$\text{(MKP)} \qquad \text{maximize} \quad z = \sum_{j=1}^{n} p_j x_j \tag{6.1}$$

$$\text{subject to} \quad \sum_{j=1}^{n} w_{ij} x_j \leq c_i, \quad i = 1, \ldots, m \tag{6.2}$$

$$x_j \in \{0, 1\}, \quad j = 1, \ldots, n. \tag{6.3}$$

n items with profits $p_j > 0$ and m resources with capacities $c_i > 0$ are given. Each item j consumes an amount $w_{ij} \geq 0$ from each resource i. The goal is to select a subset of items with maximum total profit, see (6.1); chosen items must, however, not exceed resource capacities, see (6.2). The 0–1 decision variables x_j indicate which items are selected.

The MKP first appeared in the context of capital budgeting [77, 81]. A comprehensive overview of practical and theoretical results for the MKP can be found in the monograph on knapsack problems by Kellerer et al. [66]. A recent review of the MKP was given by Fréville [38]. Besides exact techniques for solving small to moderately sized instances, based on dynamic programming [44, 130] and branch-and-bound [114, 43] many kinds of metaheuristics have already been applied to the MKP.

To our knowledge, the method currently yielding the best results, at least for commonly used benchmark instances, was described by Vasquez and Hao [127] and has recently been refined by Vasquez and Vimont [128]. It is a hybrid approach based on tabu search. The search space is reduced and partitioned via additional constraints, thereby fixing the total number of items to be packed. Bounds for these constraints are calculated by solving a modified LP-relaxation. For each remaining part of the search space, tabu-search is independently applied, starting with a solution derived from the LP-relaxation of the partial problem. The improvement described in [128] lies mainly in an additional variable fixing heuristic.

Various other metaheuristics have been described for the MKP [48, 14], including several variants of hybrid evolutionary algorithms (EAs); see [109] for a recent survey and comparison of EAs for the MKP.

Characterization of the LP-relaxation

There is one very important property characterizing the structure of the optimal solution x^{LP} to the linear programming (LP) relaxation of the MKP [66].

Proposition 8 *There exists an optimal solution* x^{LP} *with at most* $\min\{m, n\}$ *fractional values.*

Efficiency Measures

The one-dimensional 0/1-knapsack problem (KP) consists of items $j = 1, \ldots, n$ associated profits p_j and weights w_j. A subset of these items has to be selected and packed into a knapsack having a capacity c. The total profit of the items in the knapsack has to be maximized, while the total weight is not allowed to exceed c. Obviously, KP is a special case of MKP with $m = 1$. The classical greedy heuristic for KP is based on sorting the items according to decreasing efficiencies:

$$e_j = \frac{p_j}{w_j} \tag{6.4}$$

The items are then put into the knapsack in a greedy way, i.e. the next item on the list is put into the knapsack if the knapsack constraint is not violated.

An analogous strategy could be used for the MKP, but in contrast to KP there is no obvious definition of efficiency anymore. Consider the most obvious form of efficiency for the MKP, which is a direct generalization of the one-dimensional case [28]:

$$e_j(\text{simple}) = \frac{p_j}{\sum_{i=1}^{m} w_{ij}}. \tag{6.5}$$

Different orders of magnitude of the constraints are not considered and a single constraint may dominate the others. This drawback can easily be avoided by scaling:

$$e_j(\text{scaled}) = \frac{p_j}{\sum_{i=1}^{m} \frac{w_{ij}}{c_i}}. \tag{6.6}$$

Taking into account the relative contribution of the constraints, Senju and Toyoda [113] get:

$$e_j(\text{st}) = \frac{p_j}{\sum_{i=1}^{m} w_{ij}(\sum_{j=1}^{n} w_{ij} - c_i)}. \tag{6.7}$$

For more details on efficiency values we refer to Kellerer et al. [66] where a general form of efficiency is defined by introducing relevance values r_i for every constraint:

$$e_j(\text{general}) = \frac{p_j}{\sum_{i=1}^{m} r_i w_{ij}}. \tag{6.8}$$

The relevance values r_i can also be seen as kind of surrogate multipliers. Pirkul calculates good multipliers heuristically [94]. Fréville and Plateau [39] suggested setting

$$r_i = \frac{\sum_{j=1}^{n} w_{ij} - c_i}{\sum_{j=1}^{n} w_{ij}}, \tag{6.9}$$

giving the efficiency value $e_j(\text{fp})$. Setting the relevance values r_i to the values of an optimal solution to the dual problem of the MKP's LP-relaxation was a successful choice [14], yielding the efficiency value $e_j(\text{duals})$.

Benchmark Instances

Chu and Beasley's [14] benchmark library[2] is a widely used benchmark in the literature. They generated the instances as suggested by Fréville and Plateau [39]. The instance classes consist of ten instances each with $n \in \{100, 250, 500\}$ items, $m \in \{5, 10, 30\}$ constraints, and tightness ratios

$$\alpha = c_i / \sum_{j=1}^{n} w_{ij} \in \{0.25, 0.5, 0.75\}.$$

The w_{ij} are integers randomly chosen from $(0, 1000)$. The profits are correlated to the weights and generated as:

$$p_j = \sum_{i=1}^{m} w_{ij}/m + \lfloor 500 r_j \rfloor,$$

where r_j is a randomly chosen real number from $(0, 1)$.

[2] `http://people.brunel.ac.uk/~mastjjb/jeb/info.html`

6.2. The MKP and its LP-relaxation

For the classical 0/1-knapsack problem Goldberg and Marchetti-Spaccamela [53] showed that the number of items which have to be changed when moving from the solution of the KP's LP-relaxation to the optimal solution of KP grows logarithmically in expectation with increasing problem size. For the MKP, Dyer and Frieze [32] showed that the distance of the LP-relaxation and the optimal solution grows at least logarithmically in expectation with increasing problem size.

6.2.1. Empirical Analysis

Since there is only this negative result by Dyer and Frieze [32] on the distance of the relaxation and the optimum of the MKP, we performed an empirical in-depth examination on smaller instances of Chu and Beasley's benchmark library for which we were able to compute optimal solutions x^* (with $n = 100$ items, $m \in \{5, 10\}$ constraints, and $n = 250$ items, $m = 5$ constraints).

In Table 6.1 we display the average distances between optimal solutions of the MKP x^* and the solutions to the LP-relaxation x^{LP}

$$\Delta LP = \sum_{j=1}^{n} |x_j^* - x_j^{\mathrm{LP}}|, \tag{6.10}$$

the integral part of x^{LP}

$$\Delta LP_{int} = \sum_{j \in J_{int}} |x_j^* - x_j^{\mathrm{LP}}|, \quad \text{with } J_{int} = \{j = 1, \ldots, n : x_j^{\mathrm{LP}} \text{ is integral}\}, \tag{6.11}$$

and the fractional part of x^{LP}

$$\Delta LP_{frac} = \sum_{j \in J_{frac}} |x_j^* - x_j^{\mathrm{LP}}|, \quad \text{with } J_{frac} = \{j = 1, \ldots, n : x_j^{\mathrm{LP}} \text{ is fractional}\}. \tag{6.12}$$

We further display the Hamming distance between x^* and the (possibly infeasible) arithmetically rounded LP solution x^{RLP}

$$\Delta LP_{rounded} = \sum_{j=1}^{n} |x_j^* - x_j^{\mathrm{RLP}}| \quad \text{with} \quad x_j^{\mathrm{RLP}} = \lceil x_j^{\mathrm{LP}} - 0.5 \rceil, \quad j = 1, \ldots, n, \tag{6.13}$$

and the Hamming distance between x^* and a feasible solution x' created by sorting the items according to decreasing LP-relaxation solution values applying a greedy-fill procedure

$$\Delta LP_{feasible} = \sum_{j=1}^{n} |x_j^* - x'_j|. \tag{6.14}$$

The distances are displayed as percentages of the number of items ($\%n$), except ΔLP_{frac} which is displayed as a percentage of the number of constraints ($\%m$).

Table 6.1.: Distances between LP and optimal solutions (average over 10 instances and average over all problem classes)

n	m	α	ΔLP $\%n$	ΔLP_{int} $\%n$	ΔLP_{frac} $\%m$	$\Delta LP_{rounded}$ $\%n$	$\Delta LP_{feasible}$ $\%n$
100	5	0.25	5.88	3.60	45.68	5.60	7.70
		0.5	6.72	4.40	46.32	6.60	9.30
		0.75	6.56	4.30	45.17	6.50	11.60
250	5	0.25	3.12	2.20	46.25	3.12	3.80
		0.5	3.42	2.56	42.81	3.36	5.52
		0.75	3.15	2.28	43.25	3.20	7.04
100	10	0.25	9.01	4.50	45.12	8.40	11.50
		0.5	6.88	3.40	34.75	5.70	14.60
		0.75	6.75	2.60	41.51	6.50	17.40
Average			5.72	3.32	43.43	5.44	9.83

We observed that ($\Delta LP_{rounded}$) is almost always smaller than 10% of the total number of variables (see also Tables A.1 to A.3 in Appendix A.1, containing the absolute values) and was 5.44% on average. It therefore makes sense, for these instances, to reduce the search space for good or optimal solutions to the neighborhood of the solution to the LP-relaxation. Or to explore this more promising part of the search space first, if the available computation time is restricted. The most successful algorithms for the MKP exploit this fact [109, 127, 128].

One can further observe that the distance between the integer part of the solution to the LP-relaxation and an optimal solution to the MKP seems to depend on the number of variables, and that the distance between the fractional part and an optimal MKP solution seems to depend on the number of constraints (about 45% of the number of constraints). This can partly be explained with the result from Proposition 8. If we assume that our LP solution is the one with, at most, $\min\{m, n\}$ fractional values, the distance to the optimum of the fractional values cannot be greater than $\min\{m, n\}$.

6.2.2. Local Branching Based Approaches

In this section we present two approaches based on ideas developed by Fischetti and Lodi [35], known as local branching. In our empirical study in Section 6.2.1 we observed that the optimal solutions to the MKP are always close to the arithmetically rounded (potentially infeasible) LP-solutions. Focusing the optimization to such a neighborhood seems therefore to be highly promising. We further remarked that the integral distance (ΔLP_{int}) and the distance between the fractional variables (ΔLP_{frac}) evolve differently. The distance in the fractional variables seems to correlate with the number of constraints, whereas the distance in integers does not. In order to explore this fact, we propose to combine the local branching idea with an LP-relaxation based variable fixing approach.

Simple Locally Constrained Approach

In the simple Locally Constrained approach (LC), we first focus the optimization to the neighborhood of the arithmetically rounded LP solutions. This can be done by adding a single constraint to the MKP similar to the local branching constraints presented by Fischetti and Lodi [35]. The following inequality restricts the search space to a neighborhood of Hamming distance k around the rounded LP solution x^{RLP}:

$$\Delta(x, x^{\mathrm{RLP}}) = \sum_{j \in S^{\mathrm{RLP}}} (1 - x_j) + \sum_{j \notin S^{\mathrm{RLP}}} x_j \leq k, \tag{6.15}$$

where $S^{\mathrm{RLP}} = \{j = 1, \ldots, n \mid x_j^{\mathrm{RLP}} = 1\}$ is the binary support of x^{RLP}.

In our implementation we use CPLEX as B&C system and initially partition the search space by constraint (6.15) into the more promising part and by the inverse constraint $\Delta(x, x^{\mathrm{RLP}}) \geq k + 1$ into a second, remaining part. CPLEX is forced to first completely solve the neighborhood of x^{RLP} before considering the remaining search space.

Fractional Variables Free Locally Constrained Approach

In the Fractional Variables Free Locally Constrained approach (FFLC) we explore the previously noted fact that the integral distance (ΔLP_{int}) and the distance between the fractional variables (ΔLP_{frac}) evolve differently.

We leave the variables having fractional values in the LP-relaxation free, and "weakly fix" the other variables to the integer values given by the solution to the LP-relaxation. This fixing is performed by a local branching constraint, imposing that at most k-variables diverge from the fractional solution.

Let S^{LP} be the binary support of the variables with integer values in the solution of the LP-relaxation, i.e. $S^{\mathrm{LP}} = \{j = 1, \ldots, n \mid x_j^{\mathrm{LP}} = 1\}$, we further define $\overline{S}^{\mathrm{LP}} = \{j = 1, \ldots, n \mid x_j^{\mathrm{LP}} = 0\}$, the index-set of the variables set to zero in the solution of the LP-relaxation. The variables with index not in $S^{\mathrm{LP}} \cup \overline{S}^{\mathrm{LP}}$ have fractional values.

Weak variable fixing is then performed by introducing the following constraint to the MKP's ILP formulation (6.1)–(6.3):

$$\sum_{j \in S^{\mathrm{LP}}} (1 - x_j) + \sum_{j \in \overline{S}^{\mathrm{LP}}} x_j \leq k \tag{6.16}$$

Computational Experiments

In Table 6.2 we present the results of LC and CPLEX without additional constraints, with a fixed run-time of 500 seconds using different values for k. Listed are the average percentage gaps to the optimal objective value of the LP-relaxation ($\overline{\%_{\mathrm{LP}}} = 100 \cdot (z^{\mathrm{LP}} - z)/z^{\mathrm{LP}}$), the number of times this neighborhood size yields the best solution of this experiment (#), and the average number of explored nodes of the branch and bound tree.

We used the hardest instances of Chu and Beasley's benchmark library (with $n = 500$ items and $m \in \{5, 10, 30\}$ constraints). CPLEX 9.0 was used and we performed the experiments on a 2.4 GHz Intel Pentium 4 computer.

The results obtained by forcing CPLEX to first explore a more promising part of the search space can be better than if CPLEX is applied to the MKP without additional constraints. Especially for $k = 25$, which corresponds to 5% of the total number of variables, we obtain better results than with the standard approach. For $k = 10$ results were worse than those of CPLEX without additional constraints, for $k = 50$ results are not improved on average, whereas the mean number of best solutions reached is higher.

Table 6.3 shows the average results of FFLC for different values of k and CPLEX without additional constraints. Listed are the percentage gaps to the optimal objective value of the LP-relaxation ($\overline{\%_{\mathrm{LP}}}$), the number of times this neighborhood size

Table 6.2.: Results of different locally constrained MKP problems (average over 10 instances and average over all problem classes, $n = 500$).

m	α	no constraint			$k = 10$			$k = 25$			$k = 50$		
		$\overline{\%_{\mathrm{LP}}}$	#	$\overline{\mathrm{Nnodes}}$	$\overline{\%_{\mathrm{LP}}}$	#	$\overline{\mathrm{Nnodes}}$	$\overline{\%_{\mathrm{LP}}}$	#	$\overline{\mathrm{Nnodes}}$	$\overline{\%_{\mathrm{LP}}}$	#	$\overline{\mathrm{Nnodes}}$
5	0.25	0.080	8	5.50E5	**0.079**	9	5.58E5	0.080	8	5.38E5	**0.079**	8	5.38E5
	0.5	0.040	7	5.06E5	0.040	7	5.09E5	**0.039**	10	4.88E5	**0.039**	10	4.92E5
	0.75	**0.025**	8	5.36E5	**0.025**	9	5.49E5	**0.025**	7	5.24E5	**0.025**	7	5.28E5
10	0.25	**0.206**	9	3.15E5	0.221	4	3.00E5	**0.206**	9	3.03E5	**0.206**	9	3.06E5
	0.5	**0.094**	8	3.01E5	0.102	5	2.87E5	0.095	7	2.91E5	**0.094**	8	2.93E5
	0.75	**0.066**	8	3.05E5	0.068	5	2.98E5	**0.066**	8	2.95E5	**0.066**	9	2.98E5
30	0.25	0.598	5	1.11E5	0.601	1	1.02E5	**0.555**	9	1.08E5	0.605	4	1.09E5
	0.5	0.258	2	1.15E5	0.258	5	1.07E5	**0.257**	4	1.12E5	**0.257**	4	1.12E5
	0.75	0.158	5	1.12E5	0.162	4	1.07E5	**0.155**	8	1.07E5	0.159	4	1.07E5
Average		0.169	6.7	3.17E5	0.173	5.4	3.13E5	**0.164**	7.8	3.07E5	0.170	7.0	3.09E5

yielded the best solution of this experiment (#), and the number of explored nodes of the branch and bound tree.

Table 6.3.: Results of different FFLC MKP problems (average over 10 instances and average over all problem classes, $n = 500$).

m	α	no constraint			$k = 10$			$k = 20$			$k = 30$		
		$\overline{\%_{\mathrm{LP}}}$	#	$\overline{\mathrm{Nnodes}}$	$\overline{\%_{\mathrm{LP}}}$	#	$\overline{\mathrm{Nnodes}}$	$\overline{\%_{\mathrm{LP}}}$	#	$\overline{\mathrm{Nnodes}}$	$\overline{\%_{\mathrm{LP}}}$	#	$\overline{\mathrm{Nnodes}}$
5	0.25	0.080	8	5.50E5	**0.079**	8	5.46E5	0.081	8	5.21E5	0.080	8	5.42E5
	0.5	0.040	7	5.06E5	0.040	9	5.08E5	**0.039**	10	4.75E5	**0.039**	10	4.99E5
	0.75	**0.025**	9	5.36E5	**0.025**	9	5.31E5	0.026	6	5.04E5	0.026	7	5.37E5
10	0.25	0.206	7	3.15E5	**0.199**	10	3.10E5	0.206	7	2.96E5	0.206	7	3.07E5
	0.5	**0.094**	7	3.01E5	0.095	8	2.84E5	0.095	6	2.89E5	**0.094**	7	2.96E5
	0.75	**0.066**	7	3.05E5	**0.066**	7	2.88E5	**0.066**	7	2.95E5	**0.066**	6	3.00E5
30	0.25	**0.598**	6	1.11E5	0.603	7	1.03E5	0.607	4	1.08E5	0.605	5	1.09E5
	0.5	0.258	5	1.15E5	**0.253**	4	1.08E5	0.256	6	1.12E5	0.257	5	1.12E5
	0.75	**0.158**	8	1.12E5	0.159	7	1.03E5	0.159	7	1.09E5	0.159	7	1.08E5
Average		**0.169**	7.1	3.17E5	**0.169**	7.7	3.09E5	0.171	6.8	3.01E5	0.170	6.9	3.12E5

From looking at the results shown here, there is no obvious advantage in using one of the displayed methods, in total the FFLC approach is, on average, not better than the version without constraints. This is probably due to the fact that the search space is not as strongly reduced as it is the case with LC.

6.3. The Core Concept

The core concept was first presented for the classical 0/1-knapsack problem [6], which led to the very successful KP algorithms [83, 95, 96]. The main idea is to reduce the original problem to a core of items for which it is hard to decide if they will occur in an optimal solution or not, whereas all items outside the core can be immediately fixed to their optimal value.

6.3.1. The Core Concept for KP

If the items of one-dimensional 0/1-knapsack problem are sorted according to decreasing efficiencies (6.4), it is well known that the solution of the LP-relaxation consists of three parts: The first part contains the variables set to 1, the second part consists only of the so called *split item* s, which corresponds to the single variable set to a fractional value, and finally the remaining variables, which are set to zero, build the third part.

For most instances of KP (except those with a very special structure of profits and weights) the integer optimal solution coincides with this partitioning, in that it contains most of the items of the first part with high efficiency, some of the items with medium efficiencies near the split item, and almost no items with low efficiencies from the third part. Items of medium efficiency constitute the so called core.

Core Definition

Balas and Zemel [6] gave the following precise definition of the core of a one-dimensional 0/1-knapsack problem, based on the knowledge of an optimal integer solution x^*. Assume that the items are sorted according to decreasing efficiencies and define

$$a := \min\{j | x_j^* = 0\}, \qquad b := \max\{j | x_j^* = 1\}. \tag{6.17}$$

The core is given by the items in the interval $C = \{a, \ldots, b\}$. It is obvious that the split item is always part of the core.

The KP Core (KPC) problem is defined as

$$\text{(KPC)} \qquad \text{maximize} \quad z = \sum_{j \in C} p_j x_j + \tilde{p} \tag{6.18}$$

$$\text{subject to} \quad \sum_{j \in C} w_j x_j \leq c - \tilde{w}, \tag{6.19}$$

$$x_j \in \{0, 1\}, \quad j \in C, \tag{6.20}$$

with $\tilde{p} = \sum_{j=1}^{a-1} p_j$ and $\tilde{w} = \sum_{j=1}^{a-1} w_j$. Obviously, the solution of KPC would suffice to compute the optimal solution of KP, which, however, has to be already partially known to determine C.

Pisinger [96] reported experimental investigations of the exact core size. He also studied the hardness of core problems, also giving a model for their expected hardness in [97].

Fixed Core Algorithms

The first class of core algorithms is based on solving a core problem with an approximate core of fixed size $c = \{s - \delta, \ldots, s + \delta\}$ with various choices of δ, e.g. $\delta = 100$ or $\delta = \sqrt{n}$. An example is the MT2 algorithm by Martello and Toth [83]. First the core is solved, then an upper bound is derived in order to possibly prove optimality. If this is not possible, a variable reduction is performed, which tries to fix as many variables as possible to their optimal values. Finally the remaining problem is solved to optimality.

Expanding Core Algorithms

Since it is impossible to estimate the core size in advance, Pisinger proposed two expanding core algorithms. Expknap [95] uses branch and bound for enumeration, whereas Minknap [96] (which enumerates at most the smallest symmetrical core) uses dynamic programming. For more details we refer to Kellerer et al. [66].

6.3.2. The Core Concept for MKP

This concept can be expanded to MKP without major difficulties. The main problem, however, lies in the fact that there is no obvious efficiency measure. The core and the core problem have to be defined depending on a specific efficiency measure

e. Let x^* be an optimal solution and assume that the items are sorted according to decreasing efficiency e, then define

$$a_e := \min\{j|x_j^* = 0\}, \qquad b_e := \max\{j|x_j^* = 1\}. \tag{6.21}$$

The core is given by the items in the interval $C_e := \{a_e, \ldots, b_e\}$, and the core problem is defined as

$$\begin{aligned}
(\text{MKPC}_\text{e}) \qquad \text{maximize} \quad & z = \sum_{j \in C} p_j x_j + \tilde{p} && (6.22)\\
\text{subject to} \quad & \sum_{j \in C} w_{ij} x_j \leq c_i - \tilde{w}_i, \quad i = 1, \ldots, m && (6.23)\\
& x_j \in \{0, 1\}, \quad j \in C, && (6.24)
\end{aligned}$$

with $\tilde{p} = \sum_{j=1}^{a-1} p_j$ and $\tilde{w}_i = \sum_{j=1}^{a-1} w_{ij},\ i = 1, \ldots, m$.

In contrast to KP, the solution of the LP-relaxation of MKP does not consist of a single fractional split item, but its at the most m fractional values give rise to a whole *split interval* $S_e := \{s_e, \ldots, t_e\}$, where s_e and t_e are the first and the last index of variables with fractional values after sorting by efficiency e. Note that depending on the choice of the efficiency measure, the split interval can also contain variables with integer values. Moreover, the sets S_e and C_e can have almost any relation to each other, from inclusion to disjointedness. For a "reasonable" choice of e they can be expected to overlap to a large extent.

If the dual solution values of the LP-relaxation are taken as relevance values, the split interval S_e resulting from the corresponding efficiency values e_j(duals) can be precisely characterized. Let x^{LP} be the optimal solution of the LP-relaxation of MKP.

Theorem 5

$$x_j^{LP} = \begin{cases} 1 & \textit{if} \ \ e_j > 1\,, \\ \in [0,1] & \textit{if} \ \ e_j = 1\,, \\ 0 & \textit{if} \ \ e_j < 1\,. \end{cases} \tag{6.25}$$

Proof: The dual LP associated with the LP-relaxation of MKP is given by

$$\begin{aligned}
(\text{D(MKP)}) \qquad \text{minimize} \quad & \sum_{i=1}^{m} c_i u_i + \sum_{j=1}^{n} v_j && (6.26)\\
\text{subject to} \quad & \sum_{i=1}^{m} w_{ij} u_i + v_j \geq p_j, \quad j = 1, \ldots, n && (6.27)\\
& u_i, v_j \geq 0, \quad i = 1, \ldots, m,\ j = 1, \ldots, n, && (6.28)
\end{aligned}$$

where u_i are the dual variables corresponding to the capacity constraints (6.2) and v_j correspond to the equations $x_j \leq 1$. For the optimal primal and dual solutions the following complementary slackness conditions hold (see any textbook on linear programming, e.g. [10]):

$$x_j \left(\sum_{i=1}^{m} w_{ij} u_i + v_j - p_j \right) = 0 \tag{6.29}$$

$$v_j (x_j - 1) = 0 \tag{6.30}$$

Recall that $e_j(\text{duals}) = \frac{p_j}{\sum_{i=1}^{m} u_i w_{ij}}$. Hence, $e_j > 1$ implies $p_j > \sum_{i=1}^{m} w_{ij} u_i$, which means that (6.27) can only be fulfilled by $v_j > 0$. Now (6.30) immediately yields $x_j = 1$, which proves the first part of the theorem.

If $e_j < 1$, there is $p_j < \sum_{i=1}^{m} w_{ij} u_i$ which together with $v_j \geq 0$ makes the second factor of (6.29) strictly positive and requires $x_j = 0$. This proves the theorem since nothing has to be shown for $e_j = 1$. □

It follows from Theorem 5 that $S_e \subseteq \{j \mid e_j = 1,\ j = 1, \ldots, n\}$. Together with Proposition 8, this means that there exists an optimal solution x^{LP} yielding a split interval with size at most $\min\{m, n\}$.

It should be noted that the theorem gives only a structural result which does not yield any direct algorithmic advantage in computing the primal solution x^{LP}, since it requires knowing the dual optimal solution.

6.3.3. Experimental Study of MKP Cores and Core Sizes

MKP Cores and Efficiency Measures

In order to analyze the core sizes in dependence of different efficiency values, we performed an empirical in-depth examination on smaller instances of Chu and Beasley's benchmark library for which we were able to compute optimal solutions x^* (with $n = 100$ items, $m \in \{5, 10\}$ constraints, and $n = 250$ items, $m = 5$ constraints).

In Tables 6.4 and 6.5 we examine cores generated by using the scaled efficiency $e(\text{scaled})$ as defined in equation (6.6), the efficiency $e(\text{st})$ as defined in equation (6.7), the efficiency $e(\text{fp})$ as defined in equations (6.8) and (6.9), and finally the efficiency $e(\text{duals})$ setting the relevance values r_i of equation (6.8) to the optimal dual variable values of the MKP's LP-relaxation. Listed are average values of the sizes (as a percentage of the number of items) of the split interval ($|S_e|$) and of the exact core ($|C_e|$), the percentage of how much the split interval covers the core (ScC) and how much the core covers the split interval (CcS), and the distance (as

a percentage of the number of items) between the center of the split interval and the center of the core (C_{dist}). The absolute values of all the results obtained are displayed in Tables A.4 to A.6 in Appendix A.2.

Table 6.4.: Split intervals, core sizes and their mutual coverages and distances for e(scaled) and e(st) (average percentage values taken from 10 instances and average over all problem classes).

			e(scaled)					e(st)				
n	m	α	$\lvert S_e\rvert$	$\lvert C_e\rvert$	ScC	CcS	C_{dist}	$\lvert S_e\rvert$	$\lvert C_e\rvert$	ScC	CcS	C_{dist}
100	5	0.25	23.40	30.50	72.69	94.71	4.05	27.20	30.20	78.85	88.11	4.80
		0.5	29.50	37.60	71.93	88.45	5.95	27.00	35.60	69.88	89.01	5.90
		0.75	24.30	27.00	72.61	83.13	5.05	22.80	25.20	77.72	84.08	4.30
250	5	0.25	17.44	22.40	77.20	97.38	1.88	17.12	22.20	76.91	94.62	2.46
		0.5	22.88	29.44	71.71	94.25	3.44	23.76	30.88	74.95	94.69	4.04
		0.75	11.44	17.84	56.14	88.45	4.60	11.96	16.64	63.82	85.86	3.62
100	10	0.25	42.60	38.30	92.62	84.39	4.35	43.30	38.20	88.78	79.36	5.55
		0.5	39.40	45.20	80.80	91.20	5.30	44.40	46.50	85.43	88.49	5.65
		0.75	37.50	34.80	94.29	86.42	2.55	38.60	36.20	93.04	87.16	2.10
Average			27.61	31.45	76.67	89.82	4.13	28.46	31.29	78.82	87.93	4.27

Table 6.5.: Split intervals, core sizes and their mutual coverages and distances for e(fp) e(duals) (average percentage values taken from 10 instances and average over all problem classes).

			e(fp)					e(duals)				
n	m	α	$\lvert S_e\rvert$	$\lvert C_e\rvert$	ScC	CcS	C_{dist}	$\lvert S_e\rvert$	$\lvert C_e\rvert$	ScC	CcS	C_{dist}
100	5	0.25	24.70	30.10	75.50	91.94	4.20	5.00	20.20	28.12	100.00	3.30
		0.5	27.10	35.80	70.36	89.74	6.35	5.00	22.10	27.49	100.00	3.45
		0.75	23.20	26.10	74.47	84.22	4.55	5.00	19.60	26.95	100.00	3.20
250	5	0.25	16.92	21.72	76.87	95.63	2.24	2.00	12.68	18.16	100.00	2.46
		0.5	22.96	29.68	74.79	95.02	3.56	2.00	12.20	18.45	100.00	1.38
		0.75	11.40	17.12	59.00	87.27	4.06	2.00	10.40	20.18	100.00	1.56
100	10	0.25	42.10	38.20	90.41	83.74	4.75	10.00	23.20	46.57	100.00	2.90
		0.5	41.90	45.60	84.52	90.85	5.15	9.80	25.70	48.17	95.00	3.15
		0.75	37.90	35.30	94.55	86.96	2.40	9.70	18.80	55.74	99.00	2.75
Average			27.58	31.07	77.83	89.49	4.14	5.61	18.32	32.20	99.33	2.68

As expected from Theorem 5, the smallest split intervals, consisting of the fractional variables only are derived with e(duals). They further yield the smallest cores. Using any of the other efficiency measures results in significantly larger split intervals and cores. Furthermore, the smallest distances between the centers of the split intervals

and the cores are also produced by e(duals) for almost all the subclasses. The most promising information for devising approximate cores is therefore available from the split intervals generated with e(duals), on which we will concentrate our further investigations.

A Fixed Core Approach

In order to evaluate the influence of core sizes on solution quality and run-times, we propose a fixed core size algorithm, where we solve approximate cores using the general purpose ILP-solver CPLEX 9.0. We performed the experiments on a 2.4 GHz Intel Pentium 4 computer.

In analogy to KP, the approximate core is generated by adding δ items on each side of the center of the split interval. We created the cores by setting δ to $0.1n$, $0.15n$, $0.2n$, $2m + 0.1n$, and $2m + 0.2n$. The e(duals) efficiency was used. The different values of δ where chosen in accordance to the results of the previous section, where an average core size of about $0.2n$ was observed. Since outliers and the distances between the centers of the core and the split intervall have to be taken into consideration we also used bigger approximate core sizes. We also used linear combinations of m and n, since the core sizes do not depend on the number of items only, but also on the number of constraints. Tables 6.6 and 6.7 list average objective values and run-times for the original problem, percentage gaps ($\overline{\%_{\text{opt}}} = 100 \cdot (z^* - z)/z^*$) to the optimal solution, the number of times the optimum was reached (#), as well as the average run-times (as a percentage of the run-time required for solving the original problem $\overline{\%t}$ for cores of different sizes. The absolute values of all the results obtained are displayed in Tables A.7 to A.9 in Appendix A.2.

Observing the results of CPLEX applied to cores of different sizes, we see that smaller cores can be solved substantially faster and the solution obtained values are only slightly worse than the optimal ones given by the *no core* column. The best results with respect to average run-times were achieved with $\delta = 0.1n$, with which the run-time could be reduced by factors ranging from 3 to 1000, whereas, most importantly, the obtained objective values are very close to the respective optima (0.1% on average). Solving the bigger cores requires more run-time, but almost all of the optimal results could be reached with still significant time savings.

6.3.4. A Memetic Algorithm

The MA which we consider here is based on Chu and Beasley's principles and includes some improvements suggested in [107, 56, 109]. The framework is steady-state

Table 6.6.: Solving cores of different sizes exactly (average over 10 instances and average over all problem classes).

n	m	α	no core		$\delta = 0.1n$			$\delta = 0.15n$		
			$\overline{z}$	$\overline{t}[s]$	$\overline{\%_{\text{opt}}}$	$\#$	$\overline{\%t}$	$\overline{\%_{\text{opt}}}$	$\#$	$\overline{\%t}$
100	5	0.25	24197	21	0.097	5	1	0.034	7	9
		0.5	43253	27	0.053	4	1	0.018	6	6
		0.75	60471	6	0.038	5	4	0.021	7	17
250	5	0.25	60414	1474	0.008	7	36	0.003	9	81
		0.5	109293	1767	0.002	8	21	0.000	10	63
		0.75	151560	817	0.000	10	17	0.000	10	47
100	10	0.25	22602	189	0.473	1	0	0.152	4	1
		0.5	42661	97	0.234	3	0	0.084	5	1
		0.75	59556	29	0.036	6	0	0.015	8	3
Average			63778	492	0.105	5.4	9	0.036	7.3	25

Table 6.7.: Solving cores of different sizes exactly (average over 10 instances and average over all problem classes).

n	m	α	$\delta = 0.2n$			$\delta = 2m + 0.1n$			$\delta = 2m + 0.2n$		
			$\overline{\%_{\text{opt}}}$	$\#$	$\overline{\%t}$	$\overline{\%_{\text{opt}}}$	$\#$	$\overline{\%t}$	$\overline{\%_{\text{opt}}}$	$\#$	$\overline{\%t}$
100	5	0.25	0.015	9	32	0.015	9	32	0.000	10	62
		0.5	0.002	9	24	0.002	9	24	0.002	9	64
		0.75	0.001	9	39	0.001	9	39	0.000	10	61
250	5	0.25	0.000	10	82	0.003	9	69	0.000	10	91
		0.5	0.000	10	67	0.000	10	59	0.000	10	73
		0.75	0.000	10	72	0.000	10	40	0.000	10	61
100	10	0.25	0.002	9	10	0.000	10	46	0.000	10	66
		0.5	0.030	8	13	0.022	8	60	0.000	10	75
		0.75	0.011	9	22	0.000	10	54	0.000	10	70
Average			0.007	9.2	40	0.005	9.3	47	0.000	9.9	69

and the creation of initial solutions is guided by the LP-relaxation of the MKP, as described in [56]. Each new candidate solution is derived by selecting two parents via binary tournaments, performing uniform crossover on their characteristic vectors x, flipping each bit with probability $1/n$, performing repair if a capacity constraint is violated, and always performing local improvement. If such a new candidate solution is different from all solutions in the current population, it replaces the worst of them.

Both repair and local improvement are based on greedy first-fit strategies and guarantee that any resulting candidate solution lies at the boundary of the feasible region,

in which optimal solutions are always located. The repair procedure considers all items in a specific order Π and removes selected items ($x_j = 1 \rightarrow x_j = 0$) as long as any capacity constraint is violated. Local improvement works vice-versa: It considers all items in the reverse order $\overline{\Pi}$ and selects items not yet appearing in the solution as long as no capacity limit is exceeded.

Crucial for these strategies to work well is the choice of the ordering Π. Items that are likely to be selected in an optimal solution must appear near the end of Π. Following the results of Section 6.3.3 we can determine Π by ordering the items according to e(duals), as done in [14].

6.3.5. Weak Approximate Cores

In order to overcome the problem of the fixed core of a given size, we combine local branching with the core concept. Analogously to the previous section the variables in the fixed core part remain free, whereas the variables outside the core are allowed to change by a Hamming-distance constraint, limiting the number of changes to k but therefore allowing cores of potentially any size.

We consider the items to be sorted by non-increasing efficiencies e. Let $C_e^\delta(x^{\mathrm{LP}})$ be the artificial core created by adding δ items on each side of the center of the split interval $S_e(x^{\mathrm{LP}})$. Then let $a(C_e^\delta(x^{\mathrm{LP}}))$ and $b(C_e^\delta(x^{\mathrm{LP}}))$ respectively be the first and the last item of the artificial core. We can then achieve our "weak approximate core" (in analogy to (6.16)) by adding the following constraint to the standard formulation of MKP (6.1) – (6.3):

$$\sum_{j=1}^{a(C_e^\delta(x^{\mathrm{LP}}))-1} (1 - x_j) + \sum_{j=b(C_e^\delta(x^{\mathrm{LP}}))+1}^{n} x_j \leq k \tag{6.31}$$

6.3.6. Computational Experiments

We present several computational experiments where we evaluated the influence of differently sized cores on the performance of CPLEX and the presented MA. The algorithms were given 500 seconds per run. Since the MA converges much earlier, it was restarted every 1 000 000 generations, always keeping the so-far best solution in the population. We used the hardest instances of Chu and Beasley's benchmark library (with $n = 500$ items and $m \in \{5, 10, 30\}$ constraints). As before, CPLEX 9.0 was used and we performed the experiments on a 2.4 GHz Intel Pentium 4 computer.

In Table 6.8 we display the results of CPLEX applied to cores of different sizes. For comparison CPLEX was also applied to the original problem with the same time limit. We list averages over ten instances of the percentage gaps to the optimal objective value of the LP-relaxation ($\overline{\%_{\mathrm{LP}}}$), the number of times this core size yielded the best solution of this algorithm (#), and the number of explored nodes of the branch and bound tree.

First of all, it can be observed that CPLEX applied to approximate cores of different sizes always yields, on average, better results than CPLEX applied to the original problem. Secondly, the number of explored nodes increases with decreasing problem/core size. The best average results are obtained with higher core sizes ($\delta = 0.2n$).

Table 6.8.: Solving cores of different sizes with CPLEX (average over 10 instances and average over all problem classes, $n = 500$).

m	α	no core			$\delta = 0.1n$			$\delta = 0.15n$			$\delta = 0.2n$		
		$\overline{\%_{\mathrm{LP}}}$	#	$\overline{\mathrm{Nnodes}}$	$\overline{\%_{\mathrm{LP}}}$	#	$\overline{\mathrm{Nnodes}}$	$\overline{\%_{\mathrm{LP}}}$	#	$\overline{\mathrm{Nnodes}}$	$\overline{\%_{\mathrm{LP}}}$	#	$\overline{\mathrm{Nnodes}}$
5	0.25	0.080	5	5.50E5	**0.075**	9	1.00E6	0.076	9	9.85E5	0.076	8	8.34E5
	0.5	0.040	6	5.06E5	**0.039**	7	1.05E6	**0.039**	9	1.00E6	**0.039**	9	8.38E5
	0.75	0.025	6	5.36E5	**0.024**	10	1.05E6	0.025	8	1.02E6	0.025	8	9.04E5
10	0.25	0.206	1	3.15E5	0.198	5	1.10E6	**0.195**	6	6.99E5	0.198	4	5.68E5
	0.5	0.094	4	3.01E5	**0.088**	8	1.11E6	0.090	6	6.95E5	0.092	5	5.73E5
	0.75	0.066	4	3.05E5	0.065	5	1.07E6	**0.064**	7	6.83E5	0.065	7	5.59E5
30	0.25	0.598	2	1.11E5	0.621	0	4.22E5	0.566	4	3.06E5	**0.537**	6	2.28E5
	0.5	0.258	2	1.15E5	0.246	3	4.50E5	**0.243**	4	3.28E5	0.250	2	2.38E5
	0.75	0.158	2	1.12E5	**0.151**	6	4.48E5	0.160	1	3.14E5	**0.151**	5	2.36E5
Average		0.169	3.6	3.17E5	0.167	5.9	8.55E5	0.162	6.0	6.70E5	**0.159**	6.0	5.53E5

In Table 6.9 the results of the MA applied to approximate cores of different sizes are shown. In order to evaluate the benefits of using a core-based approach, we also applied the MA to the original problem. The table lists ($\overline{\%_{\mathrm{LP}}}$), the number of times this core size yielded the best solution of this algorithm (#), and the average numbers of MA iterations.

As observed with CPLEX, the use of approximate cores consistently increases the achieved solution quality. The core size has a significant influence on the number of iterations performed by the MA, which can be explained by the smaller size of the problem to be solved. This also seems to be a reason for the better results, since more candidate solutions can be examined in the given run-time. Furthermore, the search space of the MA is restricted to a highly promising part of the original search space. The best average results were obtained with $\delta = 0.15n$. The smaller approximate cores yield better results on average.

Table 6.9.: Solving cores of different sizes with the MA (average over 10 instances and average over all problem classes, $n = 500$).

m	α	no core			$\delta = 0.1n$			$\delta = 0.15n$			$\delta = 0.2n$		
		$\overline{\%_{\mathrm{LP}}}$	#	$\overline{\mathrm{Niter}}$	$\overline{\%_{\mathrm{LP}}}$	#	$\overline{\mathrm{Niter}}$	$\overline{\%_{\mathrm{LP}}}$	#	$\overline{\mathrm{Niter}}$	$\overline{\%_{\mathrm{LP}}}$	#	$\overline{\mathrm{Niter}}$
5	0.25	0.078	6	1.40E7	**0.073**	10	5.08E7	0.074	9	4.07E7	0.074	9	3.33E7
	0.5	0.040	6	1.35E7	**0.039**	9	5.07E7	**0.039**	9	4.07E7	0.040	7	3.33E7
	0.75	0.025	7	1.46E7	**0.024**	9	5.07E7	**0.024**	10	4.08E7	**0.024**	9	3.34E7
10	0.25	0.208	5	1.26E7	**0.202**	5	4.54E7	**0.202**	6	3.62E7	0.208	4	2.90E7
	0.5	0.099	2	1.21E7	0.093	6	4.51E7	**0.091**	8	3.59E7	0.093	5	2.89E7
	0.75	0.066	6	1.31E7	**0.065**	8	4.53E7	0.067	4	3.59E7	0.068	4	2.87E7
30	0.25	0.604	1	9.10E6	0.573	5	3.08E7	0.575	5	2.39E7	**0.569**	6	1.92E7
	0.5	0.254	3	8.10E6	0.257	1	3.08E7	**0.246**	7	2.37E7	0.253	3	1.90E7
	0.75	0.159	4	8.12E6	**0.156**	5	3.14E7	0.157	3	2.35E7	0.157	5	1.96E7
Average		0.170	4.4	1.17E7	0.165	6.4	4.23E7	**0.164**	6.8	3.35E7	0.165	5.8	2.72E7

In Table 6.10, we list the results of CPLEX applied to weak approximate cores of different sizes. In order to evaluate this concept, we compare it to the the results of CPLEX applied to the original problem. Since the weakening of the approximate cores can dramatically increase the size of the search space, we used smaller δ values than in the two previous experiments. The table lists ($\overline{\%_{\mathrm{LP}}}$), the number of times this core size yielded the best solution of this algorithm (#), and the number of explored branch and bound nodes.

Applying CPLEX to weak approximate cores slightly improves the results obtained, for a small approximate core size and small k. For the other parameter settings, there is no improvement. This can be due to fact, that the weak approximate core is not much easier to solve than the original problem and CPLEX therefore finds very similar solutions for the different variants within the limited run-time.

Comparing our results to the best known solutions [128], we are able to reach the best solutions for $m = 5$, and stay only 0.5% below these solutions for $m \in \{10, 30\}$. This was achieved in 500 seconds whereas in [128] up to 33 hours were required.

In summary, we applied CPLEX and a MA to approximate cores of hard to solve benchmark instances and observed that using approximate cores of fixed size instead of the original problem clearly and consistently improves the solution quality when using a fixed run-time.

Table 6.10.: Solving weak approximate cores of different sizes with CPLEX (average over 10 instances and average over all problem classes, $n = 500$).

m	α	no core			$\delta = 0.05n$, $k = 5$			$\delta = 0.1n$, $k = 5$			$\delta = 0.05n$, $k = 10$		
		$\overline{\%_{\text{LP}}}$	#	$\overline{\text{Nnodes}}$	$\overline{\%_{\text{LP}}}$	#	$\overline{\text{Nnodes}}$	$\overline{\%_{\text{LP}}}$	#	$\overline{\text{Nnodes}}$	$\overline{\%_{\text{LP}}}$	#	$\overline{\text{Nnodes}}$
5	0.25	0.080	7	5.50E5	0.078	9	5.54E5	0.078	9	5.55E5	**0.077**	9	5.63E5
	0.5	0.040	7	5.06E5	**0.039**	9	5.03E5	**0.039**	9	5.05E5	**0.039**	9	5.06E5
	0.75	**0.025**	10	5.36E5	0.026	8	5.39E5	0.026	8	5.39E5	0.026	8	5.42E5
10	0.25	**0.206**	10	3.15E5	**0.206**	10	3.13E5	0.207	9	3.15E5	**0.206**	10	3.14E5
	0.5	**0.094**	9	3.01E5	**0.094**	9	3.02E5	**0.094**	9	3.00E5	**0.094**	8	3.00E5
	0.75	**0.066**	7	3.05E5	**0.066**	8	3.05E5	**0.066**	8	3.03E5	**0.066**	7	3.05E5
30	0.25	0.598	5	1.11E5	**0.574**	7	1.10E5	0.606	4	1.11E5	0.605	4	1.11E5
	0.5	0.258	3	1.15E5	**0.254**	5	1.13E5	0.256	4	1.15E5	0.255	5	1.15E5
	0.75	**0.158**	7	1.12E5	**0.158**	5	1.11E5	0.159	5	1.11E5	0.159	6	1.12E5
Average		0.169	7.2	3.17E5	**0.166**	7.8	3.17E5	0.170	7.2	3.17E5	0.170	7.3	3.19E5

6.4. Relaxation Guided VNS for the MKP

Using LP/IP-based concepts in metaheuristics is a promising approach (see Chapter 4). Therefore we want to investigate *Relaxation Guided Variable Neighborhood Search* here. It is a new variant of Variable Neighborhood Search (VNS) [58, 59], which is based on a standard VNS scheme and a new Variable Neighborhood Descent (VND) algorithm. How to order the given neighborhoods is often a difficult yet performance-significant decision. We guide VND by always sorting the neighborhoods according to estimations of the improvement-potentials depending on the current solution. For each neighborhood this potential is determined by quickly solving a relaxation. Searching the neighborhoods in this order is expected to increase solution quality and/or to speed up VNS.

In the next section we present the general scheme of Relaxation Guided VNS. We then describe the neighborhood structures used for the MKP, together with computational experiments comparing standard VNS and relaxation guided VNS. Some extensions of this approach using more neighborhood structures and further experiments are also presented.

6.4.1. Relaxation Guided VNS

Relaxation Guided VNS (RGVNS) follows the the general VNS scheme [58, 59] and incorporates an improved VND, which we call Relaxation Guided Variable Neighborhood Descent (RGVND). Let us assume that the neighborhood structures

$N_1, \ldots, N_{k_{\max}}$ are to be used within VND. A significant question, which is often of crucial importance for the algorithm's performance, is the order in which the neighborhoods are to be considered. Often, rules of thumb such as searching smaller neighborhoods or neighborhoods which are considered to be more promising in some sense first, are used. However, in many situations finding the ideal ordering is not straight-forward. Furthermore, the ordering that is best suited in a particular situation might in general depend on the current solution. We are not aware of any previous work where the ordering of the neighborhoods is determined in an automatic way and, especially, is adapted during the search.

The main point of our extended variant of VND is that we control the order in which the neighborhood structures are processed by estimating improvement-potentials. These potentials are devised by quickly solving a relaxation of each neighborhood structure. We expect that this scheme allows more promising neighborhoods to be explored earlier, yielding better and faster overall results.

In the following we will consider maximization problems; minimization problems can be treated analogously.

Assume that we are given a combinatorial optimization problem (COP) defined as

$$z^{\mathrm{COP}} = \max\{f(x) \mid x \in S\},$$

with S being a finite set of solutions and $f(x) : S \to \mathbb{R}$ an objective function. In analogy to the relaxation of an ILP (see Section 3.2), we introduce the following formal definition of a relaxation of a COP.

Definition 17 *A* relaxation *R of COP is a maximization problem defined as*

$$z^{\mathrm{R}} = \max\{f^{\mathrm{R}}(x) \mid x \in S^{\mathrm{R}}\}$$

with the following properties:

(i) $S \subseteq S^{\mathrm{R}}$

(ii) $f(x) \leq f^{\mathrm{R}}(x)$, $\forall x \in S$.

The following evident result [132] yields a bound for COP.

Proposition 9 *If R is a relaxation of COP, $z^{\mathrm{R}} \geq z$.*

Algorithm 11: Relaxation Guided VND (RGVND)

Input: A feasible solution x
$l \leftarrow 1$
π = DetermineOrderOfNeighborhoods(x)
repeat
  Find the best neighbor $x^* \in N_{\pi(k)}(x) \mid f(x^*) \geq f(x') \; \forall x' \in N_{\pi(k)}(x)$
  if $f(x^*) > f(x)$ **then**
    $x \leftarrow x^*$
    $k \leftarrow 1$
    π = DetermineOrderOfNeighborhoods(x)
  else
    $k \leftarrow k + 1$
until $k = k_{\max}$
return x

Often, it is substantially faster to calculate the optimal solution of a relaxation than of the original problem. An example is the widely used linear programming (LP) relaxation of an integer linear programming (ILP) formulation of a COP, which can be solved in polynomial time. It is a prerequisite for the RGVND scheme that the used relaxations can be solved to optimality much faster than their corresponding original neighborhood structures.

A second precondition on the used neighborhoods is that they are not fully contained in each other since this would lead to trivial orderings and render our approach meaningless. Therefore $N_k \not\subseteq N_{k'}$ and $N_{k'} \not\subseteq N_k$ must hold for any $N_k, N_{k'}$ with $k \neq k'$.

Algorithm 12: DetermineOrderOfNeighborhoods(x)

for $k = 1, \ldots, k_{\max}$ **do**
  Solve $N_k^{\mathrm{R}}(x)$ yielding solution value z_k^{R}
Sort $\pi = (1, \ldots, k_{\max})$ according to decreasing z_k^{R}
return π

In Algorithm 11 the pseudocode of RGVND is given. The significant differences to the standard VND scheme, as described in [58, 59], are the calls of function DetermineOrderOfNeighborhoods(x) in lines 10 and 10. This function determines the order of the neighborhood structures by first solving their relaxations yielding objective values z_k^{R}, and then sorting the neighborhoods according to decreasing z_k^{R}. Ties are broken arbitrarily or according to some static heuristic rules.

6.4.2. Relaxation Guided VNS for the MKP

We now focus on the problem-specific details of our RGVNS implementation for the MKP, introducing the used neighborhoods and their relaxations, and present results for indicating the effectiveness of the new approach in comparison to standard VNS.

Representation and Initialization

Solutions are directly represented by binary strings, and all our neighborhoods are defined on the space of feasible solutions only. We denote by $I_1(x^f) = \{j \mid x_j^f = 1\}$ the index-set of the items contained in the knapsack of a current solution x^f and by $I_0(x^f) = \{j \mid x_j^f = 0\}$ its complement.

The initial solution for our VNS is generated using a greedy first-fit heuristic, considering the items in a certain order, which is determined by sorting the items according to decreasing values of the solutions to the MKP's LP-relaxation; see [109].

ILP Based Neighborhoods

We want to force a certain number of items of the current feasible solution x^f to be removed from or added to the knapsack. This is realized by adding neighborhood-defining constraints depending on x^f to the ILP formulation of the MKP.

In the first neighborhood, *ILP-Remove-and-Fill* $IRF(x^f, \kappa)$, we force precisely κ items from I_1 to be removed from the knapsack and any combination of items from I_0 is allowed to be added to the knapsack as long as the solution remains feasible. This is accomplished by adding the following equation to (6.1)–(6.3):

$$\sum_{j \in I_1(x^f)} x_j = \sum_{j \in I_1(x^f)} x_j^f - \kappa. \tag{6.32}$$

In the second neighborhood, *ILP-Add-and-Remove* $IAR(x^f, \kappa)$, we force precisely κ items not yet packed, i.e. from I_0, to be included in the knapsack. To achieve feasibility any combination of items from I_1 may be removed. This is achieved by adding the following equation to (6.1)–(6.3):

$$\sum_{j \in I_0(x^f)} x_j = \kappa. \tag{6.33}$$

As relaxations $IRF^R(x^f, \kappa)$ and $IAR^R(x^f, \kappa)$ we use the corresponding LP-relaxations in which the integrality constraints (6.3) are replaced by $0 \leq x_j \leq 1$, $j = 1, \ldots, n$. Note that depending on the specific instance's characteristics, both neighborhoods may become quite large even for $\kappa = 1$. Nevertheless, the LP-relaxations can be solved to optimality very quickly by means of standard LP algorithms. For searching the (integer) neighborhoods we use a general purpose ILP-solver (CPLEX) with a certain time limit.

Relaxation Guided VNS

The Relaxation Guided Variable Neighborhood Search (RGVNS) is based on the previously defined neighborhoods $IRF(x^f, \kappa)$ and $IAR(x^f, \kappa)$. We first solve the LP-relaxations of $IRF(x^f, \kappa)$ and $IAR(x^f, \kappa)$ for $k = 1, \ldots, \kappa_{\max}$, where $\kappa_{\max}$ is a prespecified upper limit on the number of items we want to remove or add. The neighborhoods are sorted according to decreasing LP-relaxation solution values. Ties are broken by considering smaller κs earlier.

Shaking

In the VNS framework, after RGVND has explored all neighborhoods, shaking is performed. Shaking flips l different, randomly selected variables of the current best solution and applies greedy repair and local improvement according to [14], as described in 6.3.4 using efficiencies e(dual).

As usual in general VNS, l runs from 1 to some $l_{\max}$ and is reset to 1 if an improved solution is found.

Relaxation Guided VNS versus Standard VNS

We compare RGVNS to standard VNS using the ILP based neighborhoods $IRF(x^f, \kappa)$ and $IAR(x^f, \kappa)$, for $\kappa = 1, \ldots, 10$. In RGVNS the neighborhoods are ordered according to their LP-relaxations, whereas in standard VNS the neighborhoods are statically ordered according to increasing κ and always switching between $IRF(x^f, \kappa)$ and $IAR(x^f, \kappa)$. We further compare RGVNS to RandVNS where the neighborhoods are always ordered randomly. For shaking $l_{\max}$ was set to n.

The algorithms tested were implemented in C++ using CPLEX 9.0. The ILP-based neighborhoods were not always fully explored but CPLEX was terminated after at most 2 seconds. The total run-time given to the algorithms was limited to 500 seconds. The experiments were performed on a 2.4GHz Intel Pentium 4 machine.

As before, we used the hardest instances of Chu and Beasley's benchmark library (with $n = 500$ items and $m \in \{5, 10, 30\}$ constraints). In order to evaluate the results of our experiments statistically, we performed 30 independent runs on each instance, therefore the experiments were performed on 9 instances only: The first instance of each instance category.

Table 6.11 lists the mean and median percentage gaps of the final solutions' objective values with respect to the LP-relaxation. Corresponding standard deviations are shown in parentheses. The $p_{\text{VNS,RGVNS}}$ columns list the error probabilities in t-tests and Wilcoxon rank sum tests of the hypotheses that differences exist. These statistical tests were computed using the statistics software R[3].

Table 6.11.: Comparison of VNS and RGVNS; listed are average and median percentage gaps, standard deviations in parentheses, and error probabilities $p_{\text{VNS,RGVNS}}$ obtained by t-tests and Wilcoxon rank sum tests.

		VNS		RandVNS		RGVNS		$p_{\text{VNS,RGVNS}}$		$p_{\text{RandVNS,RGVNS}}$	
m	α	mean	median	mean	median	mean	median	t-test	W-test	t-test	W-test
5	0.25	0.091	0.096	0.088	0.088	**0.082**	**0.076**	< 0.01	0.04	< 0.01	< 0.01
		(0.011)		(0.006)		(0.010)					
	0.5	0.042	0.041	0.037	0.036	**0.034**	**0.034**	< 0.01	< 0.01	< 0.01	< 0.01
		(0.005)		(0.004)		(0.000)					
	0.75	**0.023**	**0.023**	**0.023**	**0.023**	**0.023**	**0.023**	n.a.	n.a.	n.a.	n.a.
		(0.000)		(0.000)		(0.000)					
10	0.25	0.251	0.251	0.229	0.236	**0.212**	**0.204**	< 0.01	< 0.01	< 0.01	< 0.01
		(0.018)		(0.025)		(0.016)					
	0.5	0.115	**0.108**	**0.105**	**0.108**	0.108	**0.108**	< 0.01	< 0.01	0.20	0.42
		(0.009)		(0.009)		(0.007)					
	0.75	0.073	0.075	**0.071**	**0.070**	0.075	0.079	0.19	0.19	< 0.01	< 0.01
		(0.003)		(0.003)		(0.005)					
30	0.25	0.685	0.686	0.639	0.642	**0.635**	**0.614**	< 0.01	< 0.01	0.583	0.383
		(0.047)		(0.025)		(0.034)					
	0.5	0.291	0.304	**0.256**	**0.244**	0.272	0.277	< 0.01	< 0.01	< 0.01	< 0.01
		(0.032)		(0.019)		(0.022)					
	0.75	0.152	0.154	0.139	0.0136	**0.131**	**0.131**	< 0.01	< 0.01	< 0.01	< 0.01
		(0.016)		0.009		(0.000)					

For seven out of the nine instances RGVNS yields significantly better results than the VNS approach with fixed neighborhood ordering. For one of the test cases ($m = 5,\ \alpha = 0.75$) all obtained results were equal, whereas for the ($m = 10,\ \alpha = 0.75$) case the standard approach yielded slightly better results than RGVNS, but without statistical significance. When comparing RGVNS to RandVNS, one can

[3] `http://www.r-project.org/`

observe that in four cases RGVNS was significantly better, in one case results were equal, in two cases RandVNS was better, and in two cases no conclusions can be drawn. As expected the random ordering yields better results than the fixed order, but the relaxation guided approach outperforms both of the naive orderings.

6.4.3. Extending RGVNS for the MKP

The pure RGVNS approach of the previous section did not provide satisfactory results when comparing it to state-of-the-art metaheuristics. We therefore extend the RGVNS described in the previous section by some fast to solve standard neighborhood structures. Those simpler neighborhood structures are not ordered according to relaxations, but explored in a fixed order, before relying on the relaxation based ordering for calling $IRF(x^f, \kappa)$ and $IAR(x^f, \kappa)$ with $\kappa = 1, \ldots, \kappa_{\max}$.

Furthmore, we introduce an additional parameter: $\beta_{\max}$, used to limit the total number of explored ILP-based neighborhoods of RGVND explored before shaking. If, for example, we choose $\beta_{\max} = 10$, and $\kappa_{\max} = 10$, a total of 20 ILP-based neighborhoods are sorted according to their LP-relaxations, but only the first 10 are considered in VND.

Swap Neighborhood

The first neighborhood we use is a simple swap $SWP(x^f)$, where a pair of items $(x_i^f, x_j^f) \mid i \in I_1$ and $j \in I_0$ are exchanged, i.e. $x_i^f := 0$ and $x_j^f := 1$. Infeasible solutions are discarded. Note that this neighborhood is contained in both, $IRF(x^f, 1)$ and $IAR(x^f, 1)$. Its main advantage is that it can be explored much faster.

Greedy Neighborhoods

Based on the ideas of Chu and Beasley [14] and as another simplification of IRF and IAR but an extension of SWP, we define two additional neighborhoods based on greedy concepts.

In the first case, the *Remove-and-Greedy-Fill* neighborhood $RGF(x^f, \kappa)$, κ items are removed from x^f, i.e. a κ-tuple of variables from $I_1(x^f)$ are flipped. The resulting solution is then locally optimized using the greedy first-fit heuristic from Section 6.4.2.

In the second case, the *Add-and-Greedy-Repair* neighborhood $AGR(x^f, \kappa)$, κ items are added to x^f, i.e. κ variables from $I_0(x^f)$ are flipped. The resulting solution,

Table 6.12.: Results of the different approaches, using the whole Chu and Beasley 500-variables instance set.

		VNS–$\mathcal{N}_{1-3}$		VNS–$\mathcal{N}_{1-5}$		VNS		RGVNS		RGVNS+$\mathcal{N}_{1-3}$		RGVNS+$\mathcal{N}_{1-5}$	
m	α	$\overline{\%_{\text{LP}}}$	#	$\overline{\%_{\text{LP}}}$	#	$\overline{\%_{\text{LP}}}$	#	$\overline{\%_{\text{LP}}}$	#	$\overline{\%_{\text{LP}}}$	#	$\overline{\%_{\text{LP}}}$	#
5	0.25	0.124	0	0.109	0	0.113	2	0.090	2	0.088	4	**0.087**	6
	0.5	0.065	0	0.053	0	0.049	0	**0.042**	7	0.043	5	**0.042**	4
	0.75	0.041	1	0.029	2	0.032	1	**0.026**	7	0.027	6	**0.026**	8
10	0.25	0.357	0	0.293	0	0.271	1	0.234	5	**0.230**	6	0.232	4
	0.5	0.180	0	0.137	0	0.131	0	0.108	4	0.108	3	**0.103**	8
	0.75	0.103	0	0.083	0	0.084	1	**0.069**	7	**0.069**	6	0.072	5
30	0.25	0.890	0	0.825	0	0.716	0	0.609	5	**0.595**	6	0.615	4
	0.5	0.414	0	0.332	0	0.310	0	0.265	4	**0.263**	6	0.268	3
	0.75	0.232	0	0.216	0	0.189	1	**0.167**	3	0.168	3	**0.167**	3
Average		0.267	0.1	0.231	0.2	0.211	0.7	0.179	4.9	**0.177**	5.0	0.179	5.0

which is usually infeasible, is then repaired and locally improved using the greedy algorithms from Section 6.4.2.

Computational Experiments

In these experiments, we combined the simpler neighborhoods, which were explored using a best improvement strategy, with the ILP-based neighborhoods. We used the whole set of the hardest instances of Chu and Beasley's benchmark library (with $n = 500$ items and $m \in \{5, 10, 30\}$ constraints). As berfore, the experiments were performed on a 2.4GHz Intel Pentium 4 machine and each run was terminated after 500s of CPU-time. The neighborhoods are ordered as follows: $\mathcal{N}_1$:=$SWP(x^f)$, $\mathcal{N}_2$:=$RGF(x^f, 1)$, $\mathcal{N}_3$:=$AGR(x^f, 1)$, $\mathcal{N}_4$:=$RGF(x^f, 2)$, $\mathcal{N}_5$:=$AGR(x^f, 2)$.

In Table 6.12 we show results of the following algorithm variants: VNS with neighborhoods $\mathcal{N}_1$ to $\mathcal{N}_3$ only (VNS–$\mathcal{N}_{1-3}$), VNS with neighborhoods $\mathcal{N}_1$ to $\mathcal{N}_5$ only (VNS–$\mathcal{N}_{1-5}$), VNS with the ILP-based neighborhoods (VNS), RGVNS with the ILP-based neighborhoods (RGVNS), RGVNS with additionally $\mathcal{N}_1$ to $\mathcal{N}_3$, and RGVNS with additionally $\mathcal{N}_1$ to $\mathcal{N}_5$.

We can observe a clear performance difference between $\mathcal{N}_{1-3}$, $\mathcal{N}_{1-5}$, and the ILP based neighborhoods. RGVNS+$\mathcal{N}_{1-3}$ yields the best average percentage gap on average. Furthermore the RGVNS+ methods yield the highest number of best solutions found.

The fact that the RGVNS variants yielded the best average percentage gaps for all classes together with the results from the previous section, clearly documents

the benefits of sorting the neighborhoods according to a dynamically determined potential for improvement.

Experiments on Cores

Since using the core-concept for reducing the search space for the MA was a very successful idea, we want to investigate the effects of applying RGVNS to approximate MKP cores of different sizes. Here CPLEX was given a maximum of 5 seconds for exploring the ILP-based neighborhoods, $k_{\max}$ and $\beta_{\max}$ were set to 10, and $\kappa_{\max} = n$.

In Table 6.13, the results of RGVNS when applied to approximate cores of different sizes are shown together with the results of RGVNS on the original problem. The table lists average percentage gaps to the optimal objective value of the LP-relaxation ($\overline{\%_{\mathrm{LP}}}$) and the number of times this core size yields the best solution of this experiment (#). Furthermore the average total number of iterations performed by RGVND inside RGVNS is displayed.

The results obtained by applying RGVNS to the smaller approximate cores clearly dominate the results obtained without core and with $\delta = 0.2n$. This can be explained by the fact that CPLEX is used in RGVNS, and that it is able to find better solutions when dealing with smaller problem sizes. Interestingly, the number of iterations stays about the same for the different settings. The reason is that CPLEX is given the same constant time limit for searching the neighborhoods within RGVND.

Table 6.13.: Solving cores of different sizes with RGVNS (average over 10 instances and average over all problem classes, $n = 500$).

m	α	no core			$\delta = 0.1n$			$\delta = 0.15n$			$\delta = 0.2n$		
		$\overline{\%_{\mathrm{LP}}}$	#	$\overline{\mathrm{Niter}}$	$\overline{\%_{\mathrm{LP}}}$	#	$\overline{\mathrm{Niter}}$	$\overline{\%_{\mathrm{LP}}}$	#	$\overline{\mathrm{Niter}}$	$\overline{\%_{\mathrm{LP}}}$	#	$\overline{\mathrm{Niter}}$
5	0.25	0.088	4	230	**0.080**	5	208	**0.080**	6	223	0.082	4	230
	0.5	0.043	5	236	**0.040**	7	215	**0.040**	8	226	**0.040**	7	239
	0.75	0.027	5	246	**0.026**	8	230	**0.026**	8	252	**0.026**	7	240
10	0.25	0.230	0	225	**0.198**	7	200	0.211	2	193	0.210	3	205
	0.5	0.108	1	209	**0.096**	5	201	**0.096**	3	199	0.100	1	205
	0.75	0.069	2	208	**0.066**	7	207	**0.066**	7	211	**0.066**	4	214
30	0.25	0.595	5	202	0.599	3	196	**0.593**	4	191	0.609	5	195
	0.5	0.263	3	197	0.260	0	198	**0.254**	6	189	0.261	3	197
	0.75	0.168	2	191	**0.158**	5	191	0.164	3	187	0.164	2	191
Average		0.177	3.0	216	**0.169**	5.2	205	0.170	5.2	208	0.173	4.0	213

As noted before, using approximate MKP cores of fixed size instead of the original problem consistently improves the solution quality when using a fixed run-time.

6.5. Collaborative Approaches for the MKP

Until now, we have developed individual exact and metaheuristic approaches, here we want to let those approaches collaborate in their quest for finding better solutions. We study a hybrid system in which metaheuristics and the ILP-based approaches are executed in parallel and continuously exchange information in a bidirectional, asynchronous way.

The intention is to run the metaheuristics and the ILP-based approach in parallel on two individual machines. In our tests, however, we executed the algorithms in a pseudo-parallel way as individual processes on a single machine. The two processes were started at the same time and their pseudo-parallel execution was handled by the operating system.

We will see that this combination of a metaheuristic and an exact optimization method is able to benefit from synergy: Experimental results document that within the same limited total CPU-time, the cooperative system can yield better heuristic solutions than each algorithm alone.

6.5.1. Collaborative MA and B&C

In this section we describe several variants of combining the MA, as described in Section 6.3.4, with the different ILP-based approaches (see Sections 6.2 and 6.3).

If a new so-far best solution is encountered by one of the algorithms, it is immediately sent to the partner. If the MA receives such a solution, it is included into the population by replacing the worst solution, as in the case of any other newly created solution candidate. In the ILP-based approaches, a received solution is set as new incumbent solution, providing a new global lower bound.

When the ILP-based approach finds a new incumbent solution, it also sends the current dual variable values associated to the MKP-constraints, which are devised from the LP-relaxation of the node in the B&C tree currently being processed. When the MA receives these dual variable values, it recalculates the efficiencies and the item ordering Π for repair and local improvement as described in Section 6.3.4.

6.5.2. Collaborative RGVNS and B&C

We further tested collaborative variants of RGVNS (see Section 6.4.1) and different ILP-based approaches (see Sections 6.2 and 6.3).

The collaborative strategy is analogous to the one described in the previous section: If a new so-far best solution is encountered by one of the algorithms, it is immediately sent to the partner. If RGVNS receives such a solution, it is considered as the newly encountered best solution, k is reset to one, and a new RGVND is started from this new solution. When RGVND receives these dual variable values, it also recalculates the efficiencies and the item ordering Π for repair and local improvement.

6.5.3. Computational Experiments

The computational experiments were performed, as before, on a LINUX operated Intel Pentium 4 computer with 2.4 GHz. The algorithms were given a total CPU-time of 500 seconds.

The metaheuristics and the ILP-based methods were started at the same time and were each given 250 seconds (equal), or running time was assigned by a 2:1 ratio, terminating the metaheuristics after 167 seconds and the ILP-based approach was performed with a time-limit of 333 seconds (skewed). We studied these two variants, since preliminary tests with the cooperative approaches suggested that the metaheuristics were sometimes the main contributor in finding improved solutions during the early stages of the optimization process.

In Table 6.14 we show the results of the collaboration between the MA and CPLEX without any additional constraints, exchanging so far best solutions only (B&C_MA), and additionally exchanging dual variable values (B&C_MA_D). Both versions were tested with the equal and skewed cooperation strategies. For comparison purposes we also list the results of CPLEX (B&C) and the MA running alone.

The results presented in Table 6.14 show a slight advantage for the cooperative strategies. Use of the skewed collaboration scheme and the exchange of dual variable values improved the solution quality obtained. Interestingly, the MA executed alone obtained the highest number of best solutions obtained, whereas it yielded on average the worst solution quality. The best average solution quality and the second highest number of obtained best solutions is achieved with B&C_MA_D using the skewed collaboration strategy.

In Table 6.15 we show the results of the collaboration between the MA and the locally constrained ILP-based approach (LC), where constraint 6.15 is added to the

Table 6.14.: Collaborative strategies (average over 10 instances and average over all problem classes, $n = 500$).

		No Cooperation				Equal Cooperation				Skewed Cooperation			
		B&C		MA		B&C_MA		B&C_MA_D		B&C_MA		B&C_MA_D	
m	α	$\overline{\%_{\text{LP}}}$	#	$\overline{\%_{\text{LP}}}$	#	$\overline{\%_{\text{LP}}}$	#	$\overline{\%_{\text{LP}}}$	#	$\overline{\%_{\text{LP}}}$	#	$\overline{\%_{\text{LP}}}$	#
5	0.25	0.080	6	0.078	7	0.079	6	**0.077**	7	0.078	6	0.078	8
	0.5	0.040	7	0.040	9	0.041	5	0.041	5	**0.039**	8	**0.039**	10
	0.75	**0.025**	7	**0.025**	8	**0.025**	8	**0.025**	7	**0.025**	7	**0.025**	7
10	0.25	0.206	4	0.208	5	0.207	2	0.203	5	0.205	3	**0.199**	5
	0.5	0.094	5	0.099	3	**0.093**	5	0.098	2	0.095	3	0.096	4
	0.75	**0.066**	4	**0.066**	4	0.067	3	0.067	4	**0.066**	5	**0.066**	2
30	0.25	0.598	3	0.604	3	0.594	3	0.596	3	0.592	3	**0.574**	5
	0.5	0.258	2	**0.254**	5	0.257	4	0.255	4	0.254	3	0.257	4
	0.75	0.158	3	0.159	6	0.158	3	**0.156**	7	0.158	2	**0.156**	4
Average		0.169	4.6	0.170	5.6	0.169	4.3	0.169	4.9	0.168	4.4	**0.166**	5.4

ILP formulation (6.1)–(6.3) of the MKP, first restricting the search space to the more promising part in the neighborhood of the solution to the LP-relaxation. The neighborhood size parameter k was set to 25, which yielded the best results in Section 6.2.2. We list different variants exchanging so far best solutions only (LC_MA), and additionally exchanging dual variable values (LC_MA_D). Both versions were tested with the equal and skewed cooperation strategies. For comparison purposes we also list the results of LC and the MA running alone.

The results shown in Table 6.15 do not allow clear conclusions. On the one hand, the best average solution quality is obtained using LC alone. On the other hand, this result is due to the very good solutions obtained for $m = 30$, $\alpha = 0.25$. For the remaining results, LC_MA_D provides better or equal results, which can be seen by the highest average number of times it obtained the best solutions. Again the skewed collaboration strategy provides slightly better results than the equal strategy.

In Table 6.16 we show the results of the collaboration between the MA and CPLEX (B&C) applied to cores of different sizes ($\delta = 0.15n$ and $\delta = 0.2n$). We list the results for the variant where so far best solutions and dual variable values are exchanged with the skewed cooperation strategy (B&C_MA_D) . For comparison purposes we also list the results of B&C and the MA running alone.

When solving cores of different sizes, the cooperative approach cannot always improve the results of the individual algorithms. Considering the core size $\delta = 0.1n$, the collaborative approach dominates the individual algorithms, especially for the instances with $m = 30$. In case of $\delta = 0.2n$ the results are not as clear anymore,

Table 6.15.: Collaborative strategies (average over 10 instances and average over all problem classes, $n = 500$).

		No Cooperation				Equal Cooperation				Skewed Cooperation			
		LC		MA		LC_MA		LC_MA_D		LC_MA		LC_MA_D	
m	α	$\overline{\%_{\text{LP}}}$	#	$\overline{\%_{\text{LP}}}$	#	$\overline{\%_{\text{LP}}}$	#	$\overline{\%_{\text{LP}}}$	#	$\overline{\%_{\text{LP}}}$	#	$\overline{\%_{\text{LP}}}$	#
5	0.25	0.080	6	0.078	7	0.077	7	0.080	7	**0.075**	10	0.078	7
	0.5	**0.039**	9	0.040	8	0.040	7	**0.039**	7	**0.039**	8	**0.039**	8
	0.75	**0.025**	7	**0.025**	7	**0.025**	9	**0.025**	6	**0.025**	7	**0.025**	9
10	0.25	0.206	3	0.208	5	0.206	2	**0.200**	5	0.202	3	0.202	4
	0.5	0.095	3	0.099	2	0.095	4	**0.092**	6	**0.092**	5	0.094	4
	0.75	0.066	4	0.066	5	0.067	5	0.066	5	0.066	5	**0.065**	6
30	0.25	**0.555**	7	0.604	3	0.594	4	0.607	2	0.591	2	0.571	5
	0.5	0.257	2	0.254	3	**0.251**	6	0.257	3	**0.251**	5	0.260	2
	0.75	0.155	6	0.159	4	0.156	5	**0.154**	6	0.156	5	0.155	8
Average		**0.164**	5.2	0.170	4.9	0.168	5.4	0.169	5.2	0.166	5.6	0.165	5.9

since restricting the search space to cores enables the individual algorithms to find very high quality solutions.

Table 6.16.: Collaborative strategy with different core sizes (average over 10 instances and average over all problem classes, $n = 500$).

		B&C				MA				B&C_MA_D			
		$\delta = 0.15n$		$\delta = 0.2n$		$\delta = 0.15n$		$\delta = 0.2n$		$\delta = 0.15n$		$\delta = 0.2n$	
m	α	$\overline{\%_{\text{LP}}}$	#	$\overline{\%_{\text{LP}}}$	#	$\overline{\%_{\text{LP}}}$	#	$\overline{\%_{\text{LP}}}$	#	$\overline{\%_{\text{LP}}}$	#	$\overline{\%_{\text{LP}}}$	#
5	0.25	0.076	6	0.076	6	**0.074**	8	**0.074**	8	0.076	6	0.075	7
	0.5	**0.039**	8	**0.039**	7	**0.039**	8	0.040	7	**0.039**	8	**0.039**	7
	0.75	0.025	8	0.025	8	**0.024**	9	**0.024**	8	**0.024**	8	0.025	8
10	0.25	**0.195**	6	0.198	5	0.202	3	0.208	1	0.197	5	0.202	3
	0.5	0.090	5	0.092	4	0.091	4	0.093	3	**0.088**	7	0.089	6
	0.75	**0.064**	7	0.065	8	0.067	2	0.068	1	0.065	5	0.065	6
30	0.25	0.566	3	**0.537**	5	0.575	2	0.569	3	0.549	5	0.548	3
	0.5	0.243	4	0.250	2	0.246	2	0.253	1	**0.241**	3	0.246	3
	0.75	0.160	0	**0.151**	6	0.157	2	0.157	1	0.154	2	0.154	4
Average		0.162	5.2	**0.159**	5.7	0.164	4.4	0.165	3.7	**0.159**	5.4	0.160	5.2

In Table 6.17 we show the results of the collaboration variants between RGVNS and CPLEX (B&C) applied to the original problem and to cores generated with $\delta = 0.1n$. In RGVNS CPLEX was given a maximum of 5 seconds for exploring the ILP-based neighborhoods, $k_{\max}$ and $\beta_{\max}$ were set to 10, and $\kappa_{\max} = n$. We tested the collaboration variant where dual variable values are exchanged (B&C_RGVNS)

with the skewed cooperation strategy. For comparison purposes we also list the results of (B&C) and the RGVNS running alone.

As observed with the previous collaborative variants, the cooperation can slightly improve the results obtained when solving the original problems, whereas this is not obvious when solving core problems, where the individual algorithms can achieve very high quality solutions.

Table 6.17.: Collaborative strategy with and without core (average over 10 instances and average over all problem classes, $n = 500$).

		No Core						Core $\delta = 0.1n$					
		B&C		RGVNS		B&C_RGVNS		B&C		RGVNS		B&C_RGVNS	
m	α	$\overline{\%_{\text{LP}}}$	#	$\overline{\%_{\text{LP}}}$	#	$\overline{\%_{\text{LP}}}$	#	$\overline{\%_{\text{LP}}}$	#	$\overline{\%_{\text{LP}}}$	#	$\overline{\%_{\text{LP}}}$	#
5	0.25	0.080	5	0.088	1	0.083	3	**0.075**	9	0.080	5	**0.075**	9
	0.5	0.040	5	0.043	1	**0.039**	7	**0.039**	6	0.040	4	0.040	6
	0.75	0.025	5	0.027	2	0.026	4	**0.024**	9	0.026	4	**0.024**	7
10	0.25	0.206	3	0.230	0	0.206	3	**0.198**	6	**0.198**	6	0.199	6
	0.5	0.094	4	0.108	0	0.096	3	**0.088**	8	0.096	3	0.089	7
	0.75	0.066	4	0.069	1	0.067	3	**0.065**	4	0.066	5	0.066	4
30	0.25	0.598	4	0.595	4	**0.580**	5	0.621	0	0.599	3	0.606	1
	0.5	0.258	3	0.263	2	0.253	4	**0.246**	4	0.260	0	0.247	3
	0.75	0.158	2	0.168	0	0.162	2	**0.151**	8	0.158	2	0.153	6
Average		0.169	3.9	0.177	1.2	0.168	3.8	**0.167**	6.0	0.169	3.6	**0.167**	5.4

6.6. Further Computational Experiments

In order to compare the approaches we developed, to the Tabu Search based approach from Vasquez and Vimont [128], which yielded the best known results for the benchmark instances used, we tested some of our methods on a dual AMD Opteron 250 machine with 2.4 GHz, with total CPU times of 1800 seconds.

In Table 6.18 we list the results of [128], CPLEX without additional constraints (B&C), CPLEX applied to cores generated with e(duals) and $\delta = 0.25$ (B&C C), B&C_MA_D applied to the same cores with the equal cooperation strategy (B&C_MA_D C e) and with the skewed cooperation strategy(B&C_MA_D C s). Since we used a dual-processor machine, the parallel approaches were really executed in parallel, and wall-clock times of about 900 and 1200 seconds respectively were needed. Shown are the average percentage gaps to the optimal objective value

of the LP-relaxation ($\overline{\%_{\mathrm{LP}}}$), the number of times this algorithm yielded the best solution for this experiment (#), and for [128] we further display the average running times in seconds on a Intel Pentium 4 computer with 2 GHz.

Table 6.18.: Solving the MKP with different variants and total CPU times of 1800 seconds per instance, compared to best known approach (average over 10 instances and average over all problem classes, $n = 500$).

		[128]			B&C		B&C C		B&C_MA_D C e		B&C_MA_D C s	
m	α	$\overline{\%_{\mathrm{LP}}}$	#	$\overline{t}[s]$	$\overline{\%_{\mathrm{LP}}}$	#	$\overline{\%_{\mathrm{LP}}}$	#	$\overline{\%_{\mathrm{LP}}}$	#	$\overline{\%_{\mathrm{LP}}}$	#
5	0.25	0.074	8	47469	**0.073**	9	**0.073**	9	**0.073**	9	**0.073**	9
	0.5	**0.038**	7	20486	**0.038**	10	**0.038**	10	**0.038**	10	**0.038**	9
	0.75	**0.024**	10	24883	**0.024**	8	**0.024**	9	**0.024**	9	**0.024**	9
10	0.25	**0.174**	9	34964	0.190	2	0.189	2	0.192	2	0.185	3
	0.5	**0.082**	8	26333	0.087	4	0.083	6	0.084	5	**0.082**	6
	0.75	**0.057**	10	21156	0.063	2	0.061	3	0.062	2	0.061	3
30	0.25	**0.482**	10	97234	0.546	1	0.544	1	0.540	2	0.534	3
	0.5	**0.210**	10	113418	0.237	0	0.234	0	0.236	0	0.235	0
	0.75	**0.135**	10	148378	0.150	0	0.147	2	0.149	1	0.148	1
Average		**0.142**	9.1	59369	0.156	4.0	0.155	4.7	0.155	4.4	0.153	4.8

The results shown provide the same overall picture as in the previous sections, the parallel approach with the skewed cooperation strategy can slightly improve the results of the individual algorithm.

We outperform the results obtained in [128] for the $m = 5$ class, since we achieve slightly better results in substantially shorter running times. For the classes with $m \in \{10, 30\}$ the results provided in [128] are usually better than those for our approach in terms of solution quality, but not in terms of running times.

Most of the best known solutions for the instances tested are achieved by the approach proposed in [128]. However, the main drawbacks of this approach are its huge running times of more than 80 hours for the largest OR-Library instances. Running our pseudo-parallel (B&C_MA_D) approach for up to 20 hours on one instance of each type indicated that the results of [128] can be reached in 6 out of 9 cases.

6.7. Conclusions

We first studied the distance between LP-relaxed and optimal solutions of the MKP. For the benchmark instances used we empirically observed that theses distances were small, below 10% of the problem size, and depended on the number of variables

as well as on the number of constraints. This fact was explored for solving hard to solve benchmark instances, where we restricted our search to explore this more promising neighborhood of the LP-relaxations first, which improved the performance of CPLEX applied to those instances.

We then presented the core concept for MKP, and proved an interesting result about the structure of the solution to its LP-relaxation. We then empirically studied the size of MKP cores of instances we were able to solve to proven optimality, and further investigated the usefulness of solving approximate cores of fixed size. We then applied CPLEX, as well as a memetic algorithm to the core problems which provided clearly and consistently better results than solving the original problems within the given fixed run-time.

As the next step we presented a new VNS variant: Relaxation Guided Variable Neighborhood Search (RGVNS). The order in which the neighborhoods are investigated is determined dynamically by estimating their improvement-potential using quickly determined solutions to relaxations. This idea seems to be particularly useful if the order of the neighborhoods is not obvious and their relaxations can be quickly solved and yield relatively tight bounds. We tested this approach on standard benchmark instances of the multidimensional knapsack problem. The results obtained in our computational experiments show a clear advantage of RGVNS compared to VNS without guidance.

Finally we studied several collaborative combinations of the presented MA, RGVNS and the ILP-based approaches, where a metaheuristic and an exact method are executed in parallel. The collaborative approaches were given the same total CPU-times as the individual algorithms, and were able to improve the results obtained in some of the tested variants. We were able to achieve competitive results compared to best-known solutions needing significantly lower running times.

The structural analysis of LP-relaxed and optimal solutions of combinatorial optimization problems can lead to interesting results, such as the core concept, which in turn can be used in different ways for improving the solution quality of already available algorithms. In the future we want to investigate whether the core concept can be usefully expanded to other combinatorial optimization problems. The incorporation of ILP-based techniques in metaheuristics such as VNS yielded the successful RGVNS algorithm, a promising method we intend to evaluate further on other optimization problems. Finally, the cooperation of metaheuristics and ILP-based techniques is very promising, since the collaborative approaches managed to achieve better or equally good results as the individual algorithms within the same total CPU-time. Using these approaches in a parallel computing environment (e.g. multiprocessor machines or clusters) could lead to strongly improved solution quality using the same wall clock times as the individual algorithms.

Chapter 7

Conclusions and Future Research Directions

The topic of this thesis is the combination of metaheuristics and integer programming based algorithms for solving two different $\mathcal{NP}$-hard cutting and packing problems: A problem originating from the glass cutting industry, the three-stage two-dimensional bin packing problem (2BP), and a problem first mentioned in the context of capital budgeting, the multidimensional knapsack problem (MKP).

First we introduced metaheuristics and integer programming and then discussed different state-of-the-art approaches of combining them to solve combinatorial optimization problems. The two main categories into which we divided these techniques were collaborative and integrative combinations. Some of these combinations are dedicated to very specific combinatorial optimization problems, whereas others are designed to be more generally useful. Altogether, the work surveyed documents that both exact optimization techniques and metaheuristics have specific advantages which complement each other. Suitable combinations of exact algorithms and metaheuristics can benefit greatly from synergy and often exhibit significantly higher performance with respect to solution quality and time. Some of the techniques presented are mature, whereas others are still in their infancy and need substantial further research in order to develop them fully. Future work on such hybrid systems is highly promising.

Two-dimensional Bin Packing

We developed two polynomial-sized ILP models for 3-stage 2BP; a restricted model and an unrestricted one. The restricted model is particularly useful for obtaining near-optimal solutions to 3-stage 2BP quickly. Solving the unrestricted model is computationally more expensive.

Further to this, a branch-and-price algorithm based on a set covering formulation for 2BP was proposed. This B&P algorithm was enhanced by dual subset inequalities stabilizing the column generation process. Column generation was performed by applying a hierarchy of up to four pricing methods: (a) a fast greedy heuristic, (b) an evolutionary algorithm, (c) solving a restricted form of the pricing problem using CPLEX, and finally (d) solving the complete pricing problem using CPLEX.

We performed extensive computational experiments on standard benchmark instances in order to analyze the performance of the models and algorithms developed. The lower bounds obtained by column generation were strong. The best average results were achieved by B&P with all the proposed enhancements, in particular the four-level pricing strategy. These are, to our knowledge, the best known results for the 3-stage two-dimensional bin packing problem.

More generally, column generation performed by using a hierarchy of smart heuristics, which also includes metaheuristics such as evolutionary algorithms and exact algorithms, can significantly improve the optimization speed and the capabilities of branch-and-price in finding optimal or near-optimal solutions.

The Multidimensional Knapsack Problem

We first studied the distance between LP-relaxed and optimal solutions of the MKP. We empirically observed, for the smaller benchmark instances that theses distances were small, usually below 10% of the problem size, and depended on the number of variables as well as on the number of constraints. This fact was exploited when solving hard benchmark instances by restricting the search to exploring this more promising neighborhood of the LP-relaxations first. This improved the performance of CPLEX applied to those instances significantly.

We then presented the core concept for MKP, proved an interesting result about the structure of the solution to its LP-relaxation and empirically studied the size of MKP cores of smaller benchmark instances. The usefulness of solving approximate cores of fixed size was investigated. We then applied CPLEX as well as a memetic algorithm to the core problems, the results of which were clearly and consistently better than solving the original problems within the given fixed run-time.

As next step a new variable neighborhood search variant was presented: Relaxation Guided Variable Neighborhood Search (RGVNS). In this, the order in which the neighborhoods are investigated within variable neighborhood descent is dynamically determined by estimating their improvement-potential using quickly determined solutions to relaxations. This idea seems to be particularly useful if the order of the neighborhoods is not obvious and their relaxations can be solved quickly and yield relatively tight bounds. We tested this approach on standard benchmark instances of the MKP. The results obtained in our computational experiments show a clear advantage of RGVNS when compared to VNS without guidance.

We finally studied several collaborative combinations of the presented MA, RGVNS and the ILP-based approaches, where a metaheuristic and an exact method are executed in parallel. The collaborative approaches were given the same total CPU-times as the individual algorithms, and they were able to improve the obtained results in some of the tested variants. We were able to achieve competitive results compared to those of the currently leading algorithms for MKP while needing significantly lower running times.

Future Research Directions

The structural analysis of LP-relaxed and optimal solutions of combinatorial optimization problems can lead to interesting results, such as the core concept, which can be used in different ways for improving the solution quality of already available algorithms. In the future we want to investigate whether the core concept can be beneficially applied to other combinatorial optimization problems.

The incorporation of ILP-based techniques in metaheuristics such as VNS yielded the successful RGVNS algorithm, a promising method we intend to evaluate further on other optimization problems. Incorporating metaheuristics in ILP-based techniques was shown to be successful for the two-dimensional bin-packing problem. Both of these approaches show the effectiveness of integrative combinations.

Finally the cooperation of metaheuristics and ILP-based techniques is very promising, since the collaborative approaches managed to achieve better or equally good results as the individual algorithms within the same total CPU-time. Using these approaches in a parallel computing environment such as multiprocessor machines or clusters could lead to strongly improved solution quality using the same wall clock times as the individual algorithms.

In the future, we want to investigate further cooperative strategies for different collaborative combinations, as well as integrative approaches for solving combinatorial optimization problems.

Appendix A

Additional Tables for the MKP

A.1. Distances between LP-relaxed and optimal solutions

In Tables A.1 to A.3 we display the absolute values of the distances between optimal solutions of the MKP x^* and the solutions to the LP-relaxation x^{LP}

$$\Delta LP = \sum_{j=1}^{n} |x_j^* - x_j^{\mathrm{LP}}|,$$

the integral part of x^{LP}

$$\Delta LP_{int} = \sum_{j \in J_{int}} |x_j^* - x_j^{\mathrm{LP}}|, \quad \text{with } J_{int} = \{j = 1, \ldots, n : x_j^{\mathrm{LP}} \text{ is integral}\},$$

and the fractional part of x^{LP}

$$\Delta LP_{frac} = \sum_{j \in J_{frac}} |x_j^* - x_j^{\mathrm{LP}}|, \quad \text{with } J_{frac} = \{j = 1, \ldots, n : x_j^{\mathrm{LP}} \text{ is fractional}\}.$$

We further display the Hamming distance between x^* and the (possibly infeasible) arithmetically rounded LP solution x^{RLP}

$$\Delta LP_{rounded} = \sum_{j=1}^{n} |x_j^* - x_j^{\mathrm{RLP}}| \quad \text{with} \quad x_j^{\mathrm{RLP}} = \lceil x_j^{\mathrm{LP}} - 0.5 \rceil, \quad j = 1, \ldots, n,$$

and the Hamming distance between x^* and a feasible solution x' created by sorting the items according to decreasing LP-relaxation solution values applying a greedy-fill procedure

$$\Delta LP_{feasible} = \sum_{j=1}^{n} |x_j^* - x_j'|.$$

Table A.1.: Distances between LP and optimal solutions

n	m	α	ΔLP	ΔLP_{int}	ΔLP_{frac}	$\Delta LP_{rounded}$	$\Delta LP_{feasible}$
100	5	0.25	5.04	2	3.04	4	7
			6.04	3	3.04	7	11
			5.96	4	1.96	6	6
			6.55	5	1.55	7	7
			6.44	4	2.44	6	6
			5.76	4	1.76	5	8
			6.17	4	2.17	5	8
			4.93	2	2.93	5	9
			6.28	4	2.28	6	9
			5.67	4	1.67	5	6
		0.5	6.83	4	2.83	7	9
			6.62	5	1.62	7	10
			8.99	7	1.99	9	12
			6.14	4	2.14	6	6
			4.59	2	2.59	4	6
			6.35	5	1.35	7	8
			4.84	3	1.84	4	12
			5.55	3	2.55	5	7
			8.29	5	3.29	8	12
			8.96	6	2.96	9	11
		0.75	6.56	4	2.56	7	11
			3.75	3	0.75	3	9
			7.88	6	1.88	7	9
			7.98	6	1.98	8	9
			6.16	3	3.16	6	10
			7.30	4	3.30	8	14
			7.54	6	1.54	7	17
			6.91	4	2.91	7	10
			5.22	3	2.22	5	20
			6.28	4	2.28	7	7

Table A.2.: Distances between LP and optimal solutions

n	m	α	ΔLP	ΔLP_{int}	ΔLP_{frac}	$\Delta LP_{rounded}$	$\Delta LP_{feasible}$
250	5	0.25	6.90	6	0.90	6	7
			9.91	7	2.91	10	10
			5.47	3	2.47	5	14
			11.63	8	3.63	12	13
			6.53	5	1.53	6	9
			6.18	4	2.18	7	7
			7.75	5	2.75	9	9
			6.88	5	1.88	7	7
			7.74	6	1.74	7	9
			9.13	6	3.13	9	10
		0.5	5.69	4	1.69	6	12
			7.03	6	1.03	6	15
			9.87	7	2.87	10	20
			5.72	4	1.72	6	9
			10.60	8	2.60	10	12
			7.19	5	2.19	7	6
			10.37	7	3.37	11	11
			10.53	9	1.53	9	25
			7.61	6	1.61	7	8
			10.78	8	2.78	12	20
		0.75	9.01	7	2.01	9	18
			9.12	7	2.12	9	18
			7.24	5	2.24	7	34
			8.89	7	1.89	8	18
			7.03	6	1.03	6	8
			6.45	4	2.45	7	7
			8.36	6	2.36	9	30
			7.19	5	2.19	8	7
			6.20	4	2.20	6	7
			9.13	6	3.13	11	29

Table A.3.: Distances between LP and optimal solutions

n	m	α	ΔLP	ΔLP_{int}	ΔLP_{frac}	$\Delta LP_{rounded}$	$\Delta LP_{feasible}$
100	10	0.25	9.36	5	4.36	8	10
			9.13	5	4.13	8	12
			7.6	2	5.6	7	14
			10.11	4	6.11	10	17
			8.97	5	3.97	8	8
			8.75	6	2.75	9	9
			10	5	5	10	13
			11.44	6	5.44	11	13
			8.81	6	2.81	7	11
			5.95	1	4.95	6	8
		0.5	8.76	5	3.76	8	10
			4.58	2	2.58	4	10
			6.49	3	3.49	5	17
			9.26	6	3.26	8	10
			6.84	4	2.84	7	16
			3.36	0	3.36	0	7
			8.25	5	3.25	6	23
			6.49	3	3.49	5	14
			5.8	2	3.8	5	19
			8.92	4	4.92	9	20
		0.75	3.74	1	2.74	2	19
			4.97	1	3.97	5	22
			8.1	4	4.1	8	12
			5.97	2	3.97	6	11
			5.53	2	3.53	5	11
			9.43	5	4.43	10	16
			9.06	3	6.06	10	24
			8.13	4	4.13	7	22
			6.25	2	4.25	7	19
			6.33	2	4.33	5	18

A.2. MKP Core Structure

In Tables A.4 to A.6 we examine cores generated by using the scaled efficiency e(scaled), the efficiency e(st), the efficiency e(fp), and finally the efficiency e(duals) (see also Section 6.3.3). Listed are the indices of the first zero (f) and the last one (l), as well as the absolute sizes of the cores (s).

Table A.4.: Split intervals and cores for different efficiency measures, listed are their first (f) and last (l) indices as well as their sizes ($n = 100$, $m = 5$).

	e(scaled)						e(st)						e(fp)						e(duals)					
α	S_e			C_e			S_e			C_e			S_e			C_e			S_e			C_e		
	f	l	s	f	l	s	f	l	s	f	l	s	f	l	s	f	l	s	f	l	s	f	l	s
0.25	17	34	18	17	54	38	16	39	24	16	50	35	17	37	21	17	50	34	28	32	5	19	32	14
	22	33	12	18	33	16	14	40	27	19	40	22	17	38	22	20	38	19	27	31	5	23	32	10
	24	40	17	17	41	25	22	40	19	15	41	27	23	41	19	17	40	24	27	31	5	18	45	28
	7	46	40	8	44	37	7	53	47	7	43	37	8	49	42	7	44	38	26	30	5	16	37	22
	17	43	27	12	54	43	18	43	26	14	57	44	19	42	24	14	55	42	27	31	5	11	41	31
	25	36	12	20	37	18	24	39	16	16	37	22	25	36	12	19	38	20	27	31	5	20	36	17
	14	50	37	16	50	35	14	50	37	20	50	31	14	50	37	17	50	34	28	32	5	25	40	16
	22	50	29	22	56	35	21	49	29	21	46	26	22	48	27	22	52	31	26	30	5	26	57	32
	12	33	22	12	52	41	17	33	17	17	51	35	14	32	19	14	51	38	26	30	5	22	37	16
	13	32	20	21	37	17	9	38	30	21	43	23	11	34	24	21	41	21	25	29	5	18	33	16
0.5	44	71	28	37	73	37	46	69	24	37	74	38	45	70	26	38	73	36	50	54	5	41	57	17
	45	68	24	29	77	49	44	66	23	35	70	36	45	67	23	29	73	45	48	52	5	42	56	15
	36	65	30	34	74	41	43	65	23	30	78	49	37	64	28	33	76	44	52	56	5	45	68	24
	37	87	51	26	74	49	32	86	55	25	73	49	36	87	52	27	73	47	50	54	5	45	68	24
	39	58	20	37	62	26	46	58	13	36	59	24	44	58	15	37	59	23	51	55	5	38	58	21
	34	68	35	43	78	36	43	64	22	47	74	28	40	65	26	45	73	29	52	56	5	48	59	12
	49	68	20	40	62	23	45	67	23	42	62	21	48	67	20	41	62	22	51	55	5	38	63	26
	41	60	20	7	65	59	43	64	22	2	64	63	45	62	18	4	63	60	51	55	5	11	65	55
	37	75	39	37	70	34	36	77	42	36	63	28	37	75	39	37	66	30	50	54	5	47	57	11
	45	72	28	45	66	22	42	64	23	42	61	20	43	66	24	43	64	22	52	56	5	46	61	16
0.75	63	78	16	72	91	20	60	82	23	71	90	20	62	80	19	71	91	21	74	78	5	73	88	16
	46	98	53	58	99	42	53	93	41	65	96	32	50	95	46	59	99	41	75	79	5	60	81	22
	68	82	15	65	92	28	66	83	18	61	94	34	67	83	17	64	93	30	75	79	5	66	86	21
	64	95	32	55	86	32	64	90	27	70	90	21	64	92	29	63	86	24	74	78	5	68	86	19
	66	90	25	62	94	33	67	89	23	60	94	35	66	89	24	60	94	35	75	79	5	71	93	23
	74	90	17	65	87	23	75	93	19	62	91	30	76	91	16	65	87	23	73	77	5	66	85	20
	63	87	25	69	89	21	67	84	18	71	84	14	64	87	24	71	87	17	74	78	5	67	84	18
	69	86	18	57	86	30	69	84	16	57	84	28	69	85	17	57	85	29	74	78	5	69	81	13
	67	91	25	67	86	20	67	91	25	67	87	21	67	91	25	67	87	21	73	77	5	72	85	14
	64	80	17	60	80	21	65	82	18	64	80	17	67	81	15	62	81	20	73	77	5	52	81	30

Table A.5.: Split intervals and cores for different efficiency measures, listed are their first (f) and last (l) indices as well as their sizes ($n = 250$, $m = 5$).

α	e(scaled)						e(st)						e(fp)						e(duals)					
	S_e			C_e			S_e			C_e			S_e			C_e			S_e			C_e		
	f	l	s	f	l	s	f	l	s	f	l	s	f	l	s	f	l	s	f	l	s	f	l	s
0.25	55	82	28	55	101	47	53	88	36	53	96	44	56	82	27	56	97	42	71	75	5	63	91	29
	69	81	13	50	99	50	61	87	27	38	102	65	65	85	21	42	100	59	71	75	5	50	83	34
	66	96	31	65	96	32	65	94	30	65	92	28	65	95	31	65	95	31	75	79	5	60	81	22
	48	108	61	38	106	69	43	103	61	41	98	58	46	105	60	43	101	59	68	72	5	54	102	49
	46	98	53	50	98	49	46	98	53	57	98	42	45	98	54	56	98	43	71	75	5	62	79	18
	51	101	51	51	101	51	55	106	52	54	106	53	54	103	50	54	103	50	72	76	5	58	94	37
	65	110	46	52	145	94	67	110	44	52	151	100	66	110	45	52	145	94	73	77	5	69	119	51
	49	91	43	49	102	54	45	86	42	45	103	59	48	88	41	48	100	53	71	75	5	66	104	39
	54	118	65	52	108	57	57	111	55	42	101	60	56	115	60	44	105	62	74	78	5	68	85	18
	66	110	45	54	110	57	73	100	28	55	100	46	70	103	34	54	103	50	70	74	5	63	82	20
0.5	89	178	90	101	162	62	98	174	77	109	175	67	94	176	83	105	166	62	130	134	5	109	145	37
	104	171	68	108	171	64	99	176	78	106	176	71	106	174	69	107	174	68	130	134	5	125	146	22
	125	137	13	117	145	29	115	148	34	118	148	31	122	143	22	118	143	26	128	132	5	119	140	22
	91	184	94	74	184	111	100	169	70	78	169	92	94	179	86	76	179	104	131	135	5	106	152	47
	103	155	53	83	155	73	107	152	46	89	153	65	105	153	49	87	153	67	127	131	5	106	141	36
	98	192	95	94	192	99	105	200	96	85	200	116	102	195	94	88	195	108	129	133	5	123	141	19
	128	144	17	96	160	65	127	141	15	99	157	59	128	143	16	98	159	62	132	136	5	122	144	23
	123	171	49	107	171	65	123	182	60	92	182	91	124	177	54	101	177	77	131	135	5	119	141	23
	95	133	39	103	209	107	95	132	38	103	202	100	95	133	39	104	206	103	128	132	5	113	165	53
	96	149	54	96	156	61	91	170	80	91	170	80	94	155	62	94	158	65	128	132	5	117	139	23
0.75	195	213	19	155	213	59	178	211	34	153	211	59	189	213	25	154	213	60	189	193	5	175	200	26
	174	200	27	159	200	42	177	200	24	175	200	26	175	201	27	166	201	36	189	193	5	171	195	25
	175	198	24	174	210	37	170	200	31	167	205	39	175	198	24	171	205	35	190	194	5	168	198	31
	157	194	38	182	227	46	167	195	29	184	225	42	162	195	34	182	226	45	189	193	5	184	210	27
	162	196	35	167	211	45	167	197	31	176	204	29	167	197	31	173	207	35	187	191	5	183	201	19
	172	197	26	167	220	54	171	194	24	161	214	54	171	195	25	164	217	54	188	192	5	180	203	24
	180	212	33	166	212	47	182	209	28	174	209	36	181	211	31	171	211	41	189	193	5	184	199	16
	180	217	38	161	208	48	182	217	36	159	210	52	181	216	36	159	209	51	190	194	5	174	206	33
	178	206	29	178	215	38	176	202	27	176	216	41	178	202	25	178	216	39	188	192	5	173	205	33
	191	207	17	176	205	30	189	223	35	174	211	38	188	214	27	175	206	32	188	192	5	173	198	26

Table A.6.: Split intervals and cores for different efficiency measures, listed are their first (f) and last (l) indices as well as their sizes ($n = 100$, $m = 10$).

	e(scaled)						e(st)						e(fp)						e(duals)					
α	S_e			C_e			S_e			C_e			S_e			C_e			S_e			C_e		
	f	l	s	f	l	s	f	l	s	f	l	s	f	l	s	f	l	s	f	l	s	f	l	s
0.25	6	58	53	11	48	38	4	62	59	12	49	38	6	59	54	12	47	36	22	31	10	18	48	31
	14	54	41	15	53	39	9	60	52	10	54	45	11	57	47	13	54	42	23	32	10	22	42	21
	8	39	32	8	53	46	8	36	29	8	47	40	8	37	30	8	52	45	21	30	10	15	31	17
	4	54	51	6	56	51	4	51	48	3	54	52	4	52	49	6	54	49	23	32	10	16	35	20
	10	50	41	2	35	34	15	48	34	3	35	33	13	49	37	3	36	34	22	31	10	17	41	25
	5	58	54	3	45	43	7	58	52	2	46	45	6	57	52	2	46	45	24	33	10	14	42	29
	19	60	42	19	52	34	19	62	44	14	49	36	19	60	42	18	52	35	23	32	10	19	47	29
	6	54	49	7	48	42	7	59	53	6	48	43	7	56	50	6	48	43	21	30	10	17	37	21
	6	45	40	5	35	31	3	47	45	3	37	35	4	46	43	4	36	33	22	31	10	11	37	27
	12	34	23	12	36	25	17	33	17	21	35	15	17	33	17	17	36	20	23	32	10	23	34	12
0.5	15	65	51	28	92	65	15	70	56	28	93	66	16	68	53	28	93	66	46	54	9	35	63	29
	30	83	54	30	81	52	29	85	57	29	81	53	29	84	56	29	81	53	50	58	9	48	68	21
	28	58	31	28	75	48	24	62	39	24	80	57	27	58	32	27	77	51	46	55	10	40	81	42
	18	64	47	20	67	48	15	66	52	24	70	47	17	65	49	22	67	46	50	59	10	41	66	26
	26	61	36	21	73	53	19	63	45	22	71	50	23	62	40	22	71	50	46	55	10	33	64	32
	24	69	46	37	62	26	20	70	51	36	62	27	22	69	48	36	61	26	48	57	10	49	53	5
	37	72	36	22	69	48	30	74	45	21	72	52	33	73	41	22	71	50	48	57	10	40	70	31
	36	72	37	36	71	36	36	72	37	36	73	38	36	73	38	36	72	37	49	58	10	47	59	13
	38	69	32	38	69	32	41	72	32	44	72	29	41	72	32	41	72	32	48	57	10	46	57	12
	44	67	24	39	82	44	39	68	30	35	80	46	40	69	30	37	81	45	48	57	10	38	83	46
0.75	62	96	35	61	88	28	67	97	31	62	88	27	63	97	35	62	89	28	72	81	10	68	81	14
	62	95	34	61	95	35	59	97	39	58	97	40	61	95	35	59	95	37	70	79	10	65	78	14
	54	98	45	53	96	44	49	99	51	54	94	41	52	98	47	54	96	43	72	81	10	63	82	20
	42	99	58	42	99	58	46	98	53	46	98	53	45	98	54	45	98	54	70	78	9	66	79	14
	66	97	32	69	85	17	66	99	34	68	84	17	66	97	32	69	84	16	71	79	9	67	91	25
	57	93	37	61	94	34	57	93	37	58	94	37	57	93	37	60	94	35	72	80	9	52	82	31
	62	94	33	54	94	41	58	95	38	53	95	43	60	95	36	54	95	42	71	80	10	63	87	25
	58	94	37	58	94	37	55	99	45	53	99	47	58	97	40	55	97	43	70	79	10	66	82	17
	60	98	39	64	90	27	61	97	37	65	91	27	61	98	38	64	90	27	71	80	10	70	84	15
	67	91	25	60	86	27	67	87	21	61	90	30	67	91	25	61	88	28	72	81	10	69	81	13

A.3. Fixed MKP Core Results

Tables A.7 to A.9 list the results obtained by exactly solving cores of fixed size using CPLEX (see Section 6.3.3). We created the cores by setting δ to $0.1n$, $0.15n$, $0.2n$, $2m + 0.1n$, and $2m + 0.2n$. The e(duals) efficiency was used. Given are objective values and run-times.

Table A.7.: Solving cores of different sizes exactly ($n = 100$, $m = 5$).

α	no core		$\delta = 0.1n$		$\delta = 0.15n$		$\delta = 0.2n$		$\delta = 2m + 0.1n$		$\delta = 2m + 0.2n$	
	z	$t[s]$	z	$t[s]$	z	$t[s]$	z	$t[s]$	z	$t[s]$	z	$t[s]$
0.25	24381	32.16	24329	0.51	24381	2.75	24381	10.46	24381	10.46	24381	26.43
	24274	12.51	24274	0.09	24274	0.80	24274	2.91	24274	2.91	24274	5.82
	23551	22.80	23523	0.40	23538	2.40	23551	5.86	23551	5.86	23551	12.45
	23534	63.82	23477	0.84	23534	4.68	23534	20.05	23534	20.05	23534	37.66
	23991	30.97	23939	0.18	23959	2.42	23991	9.25	23991	9.25	23991	16.10
	24613	11.37	24613	0.16	24613	0.91	24613	3.68	24613	3.68	24613	6.58
	25591	2.90	25591	0.18	25591	0.63	25591	1.60	25591	1.60	25591	2.64
	23410	8.97	23367	0.19	23374	1.25	23374	5.76	23374	5.76	23410	5.00
	24216	14.57	24216	0.32	24216	0.99	24216	3.99	24216	3.99	24216	9.94
	24411	14.44	24411	0.11	24411	1.55	24411	4.70	24411	4.70	24411	9.50
0.5	42757	3.75	42705	0.21	42757	0.43	42757	1.22	42757	1.22	42757	1.92
	42545	5.46	42545	0.24	42545	0.93	42545	2.38	42545	2.38	42545	4.67
	41968	180.83	41959	0.35	41968	3.93	41968	36.83	41968	36.83	41968	108.66
	45090	17.28	45033	0.21	45070	2.70	45090	4.65	45090	4.65	45090	11.37
	42218	10.02	42148	0.43	42198	1.65	42218	2.67	42218	2.67	42218	6.80
	42927	2.42	42927	0.12	42927	0.54	42927	1.08	42927	1.08	42927	1.69
	42009	1.22	41980	0.18	41980	0.53	42009	0.51	42009	0.51	42009	0.83
	45020	41.25	45010	0.32	45010	4.00	45010	12.99	45010	12.99	45010	30.95
	43441	2.31	43441	0.16	43441	0.60	43441	1.15	43441	1.15	43441	1.57
	44554	8.95	44554	0.33	44554	0.96	44554	2.10	44554	2.10	44554	6.21
0.75	59822	2.28	59799	0.13	59822	0.42	59822	0.97	59822	0.97	59822	1.84
	62081	1.71	61983	0.19	62019	0.71	62081	0.44	62081	0.44	62081	0.53
	59802	7.11	59760	0.51	59802	1.67	59802	2.98	59802	2.98	59802	4.24
	60479	13.44	60479	0.22	60479	2.04	60479	5.39	60479	5.39	60479	7.70
	61091	5.00	61029	0.24	61029	1.53	61091	1.69	61091	1.69	61091	2.90
	58959	8.44	58959	0.09	58959	0.84	58959	4.38	58959	4.38	58959	5.88
	61538	2.85	61538	0.20	61538	0.59	61538	1.23	61538	1.23	61538	1.85
	61520	2.87	61520	0.29	61520	0.51	61520	1.39	61520	1.39	61520	1.97
	59453	1.43	59453	0.11	59453	0.39	59453	0.76	59453	0.76	59453	1.01
	59965	14.60	59960	0.13	59960	1.38	59960	4.28	59960	4.28	59965	8.53

Table A.8.: Solving cores of different sizes exactly ($n = 250$, $m = 5$).

α	no core		$\delta = 0.1n$		$\delta = 0.15n$		$\delta = 0.2n$		$\delta = 2m + 0.1n$		$\delta = 2m + 0.2n$	
	z	$t[s]$	z	$t[s]$	z	$t[s]$	z	$t[s]$	z	$t[s]$	z	$t[s]$
0.25	59312	112.63	59312	27.04	59312	60.59	59312	101.18	59312	36.66	59312	77.26
	61472	1755.26	61472	327.09	61472	1628.63	61472	1722.28	61472	1106.27	61472	1830.74
	62130	50.36	62130	14.28	62130	19.25	62130	29.38	62130	19.37	62130	32.16
	59463	3563.95	59453	1820.16	59463	2289.01	59463	2851.70	59463	2106.51	59463	3772.26
	58951	3223.04	58951	383.31	58951	2402.69	58951	2543.56	58951	2181.86	58951	2479.33
	60077	2729.24	60077	487.50	60077	2341.98	60077	2231.30	60077	1743.16	60077	2352.71
	60414	853.05	60396	819.66	60396	1746.54	60414	664.16	60396	1929.33	60414	675.84
	61472	1919.33	61449	1248.75	61472	1245.94	61472	1552.72	61472	817.38	61472	1852.49
	61885	473.90	61885	98.08	61885	249.26	61885	330.36	61885	206.50	61885	349.91
	58959	61.60	58959	24.14	58959	30.93	58959	38.35	58959	32.75	58959	40.14
0.5	109109	1027.54	109109	114.38	109109	496.63	109109	621.98	109109	418.25	109109	669.66
	109841	186.65	109841	37.92	109841	63.77	109841	102.99	109841	57.77	109841	117.61
	108508	1518.97	108508	294.34	108508	853.16	108508	1037.08	108508	864.18	108508	1207.03
	109383	1023.42	109378	187.48	109383	473.01	109383	593.83	109383	435.13	109383	744.05
	110720	3339.30	110720	914.55	110720	2288.79	110720	2465.61	110720	2284.07	110720	2815.36
	110256	1850.41	110256	201.17	110256	1801.53	110256	1847.45	110256	1281.71	110256	1633.33
	109040	2071.48	109040	304.08	109040	940.12	109040	1018.66	109040	985.95	109040	1211.04
	109042	3232.11	109042	1075.12	109042	2073.19	109042	1923.72	109042	2066.98	109042	2134.04
	109971	1364.62	109957	200.49	109971	308.85	109971	544.78	109971	344.41	109971	594.77
	107058	2058.78	107058	383.94	107058	1784.36	107058	1761.84	107058	1660.88	107058	1796.97
0.75	149665	1836.91	149665	297.77	149665	647.57	149665	1125.22	149665	487.92	149665	1011.71
	155944	431.14	155944	79.72	155944	228.58	155944	286.43	155944	303.58	155944	339.55
	149334	1693.64	149334	247.96	149334	852.67	149334	1019.46	149334	891.75	149334	1012.22
	152130	816.43	152130	167.97	152130	460.42	152130	540.05	152130	485.89	152130	535.75
	150353	897.36	150353	316.83	150353	438.71	150353	1372.04	150353	415.68	150353	572.61
	150045	31.81	150045	9.07	150045	13.65	150045	15.92	150045	12.58	150045	17.38
	148607	13.73	148607	5.58	148607	5.99	148607	6.88	148607	5.26	148607	8.31
	149782	1461.65	149782	180.35	149782	867.56	149782	1082.42	149782	380.46	149782	1052.60
	155075	140.55	155075	24.15	155075	67.03	155075	71.34	155075	55.45	155075	79.08
	154668	848.93	154668	77.07	154668	299.36	154668	378.62	154668	224.39	154668	395.30

Table A.9.: Solving cores of different sizes exactly ($n = 100$, $m = 10$).

α	no core		$\delta = 0.1n$		$\delta = 0.15n$		$\delta = 0.2n$		$\delta = 2m + 0.1n$		$\delta = 2m + 0.2n$	
	z	$t[s]$	z	$t[s]$	z	$t[s]$	z	$t[s]$	z	$t[s]$	z	$t[s]$
0.25	23064	348.29	22954	0.13	23050	1.10	23059	19.08	23064	141.60	23064	233.12
	22801	299.35	22750	0.13	22750	4.44	22801	45.69	22801	124.26	22801	175.43
	22131	55.65	22081	0.20	22131	1.27	22131	8.30	22131	23.01	22131	33.39
	22772	703.33	22694	0.09	22772	1.13	22772	50.78	22772	311.79	22772	438.98
	22751	26.50	22528	0.14	22654	1.34	22751	4.72	22751	14.72	22751	19.47
	22777	316.51	22613	0.16	22716	1.74	22777	31.06	22777	193.82	22777	245.42
	21875	33.90	21671	0.39	21841	0.82	21875	7.91	21875	17.66	21875	26.08
	22635	25.21	22542	0.13	22635	0.72	22635	7.15	22635	12.02	22635	16.30
	22511	30.70	22418	0.14	22423	1.77	22511	6.94	22511	16.48	22511	19.20
	22702	54.94	22702	0.04	22702	0.17	22702	6.49	22702	24.57	22702	45.31
0.5	41395	107.79	41331	0.09	41331	0.60	41395	6.00	41395	44.27	41395	86.53
	42344	54.89	42157	0.18	42344	0.93	42344	9.13	42344	31.60	42344	45.01
	42401	96.34	42306	0.32	42350	2.61	42350	20.20	42350	118.13	42401	64.96
	45624	120.55	45408	0.25	45624	1.21	45624	13.43	45624	61.86	45624	102.12
	41884	43.01	41737	0.07	41801	0.56	41884	5.22	41884	22.86	41884	27.81
	42995	129.76	42995	0.08	42995	0.91	42995	13.02	42995	63.05	42995	99.15
	43574	169.38	43435	0.15	43552	0.97	43574	8.56	43574	46.33	43574	114.08
	42970	64.81	42970	0.06	42970	0.81	42970	4.45	42970	35.81	42970	45.81
	42212	99.74	42212	0.13	42212	0.54	42212	9.59	42212	32.04	42212	72.61
	41207	85.19	41050	0.16	41078	1.61	41134	35.86	41165	131.58	41207	67.38
0.75	57375	4.79	57375	0.03	57375	0.37	57375	1.56	57375	2.88	57375	3.43
	58978	54.72	58978	0.07	58978	1.19	58978	12.90	58978	41.01	58978	40.66
	58391	51.69	58310	0.11	58391	0.57	58391	8.18	58391	29.65	58391	37.82
	61966	8.64	61966	0.05	61966	0.22	61966	2.19	61966	4.01	61966	6.23
	60803	9.96	60797	0.08	60797	0.24	60803	3.01	60803	5.66	60803	6.63
	61437	24.14	61336	0.23	61348	1.90	61368	14.57	61437	9.87	61437	13.18
	56377	89.21	56351	0.05	56377	1.22	56377	11.39	56377	38.32	56377	62.72
	59391	11.13	59391	0.15	59391	0.74	59391	3.15	59391	6.25	59391	7.78
	60205	14.38	60205	0.07	60205	0.48	60205	1.70	60205	7.47	60205	9.84
	60633	17.12	60633	0.11	60633	1.26	60633	4.34	60633	10.60	60633	11.42

Bibliography

[1] E. Aarts and J. K. Lenstra. *Local Search in Combinatorial Optimization*. John Wiley and Sons, 1997.

[2] R. K. Ahuja, Ö. Ergun, J. B. Orlin, and A. P. Punnen. A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics*, 123(1-3):75–102, 2002.

[3] D. Applegate, R. Bixby, V. Chvátal, and W. Cook. On the solution of the traveling salesman problem. *Documenta Mathematica*, Extra Volume ICM III:645–656, 1998.

[4] T. Bäck, D. B. Fogel, and Z. Michalewicz. *Handbook of Evolutionary Computation*. Oxford University Press, New York, 1997.

[5] J. E. Baker. Reducing bias and inefficiency in the selection algorithm. In *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application*, pages 14–21, Hillsdale, NJ, USA, 1987. Lawrence Erlbaum Associates, Inc.

[6] E. Balas and E. Zemel. An algorithm for large zero-one knapsack problems. *Operations Research*, 28:1130–1154, 1980.

[7] C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh, and P. H. Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, 46(3):316–329, 1998.

[8] H. Ben Amor, J. Desrosiers, and J. Valério de Carvalho. Dual-optimal inequalities for stabilized column generation. Technical Report G-2003-20, Les Cahiers du GERAD, HEC Montréal and GERAD, Canada, 2003. Under revision for *Operations Research*.

[9] J. O. Berkey and P. Y. Wang. Two-dimensional finite bin packing algorithms. *Journal of the Operational Research Society*, 38:423–429, 1987.

[10] D. Bertsimas and J. N. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, 1997.

[11] S. Bisotto, F. Corno, P. Prinetto, M. Rebaudengo, and M. S. Reorda. Optimizing Area Loss in Flat Glass Cutting. In *GALESIA97, IEE/IEEE International Conference on Genetic ALgorithms in Engineering Systems: Innovations and Applications*, Glasgow, UK, 1997.

[12] E. K. Burke, P. I. Cowling, and R. Keuthen. Effective local and guided variable neighborhood search methods for the asymmetric travelling salesman problem. In E. Boers et al., editors, *Applications of Evolutionary Computing: EvoWorkshops 2001*, volume 2037 of *LNCS*, pages 203–212. Springer, 2001.

[13] S. Chen, S. Talukdar, and N. Sadeh. Job-shop-scheduling by a team of asynchronous agents. In *IJCAI-93 Workshop on Knowledge-Based Production, Scheduling and Control*, Chambery, France, 1993.

[14] P. C. Chu and J. Beasley. A genetic algorithm for the multiconstrained knapsack problem. *Journal of Heuristics*, 4:63–86, 1998.

[15] F. Chung, M. Garey, and D. Johnson. On packing two-dimensional bins. *SIAM Journal of Algebraic and Discrete Methods*, 3:66–76, 1982.

[16] D. Clements, J. Crawford, D. Joslin, G. Nemhauser, M. Puttlitz, and M. Savelsbergh. Heuristic optimization: A hybrid AI/OR approach. In *Proceedings of the Workshop on Industrial Constraint-Directed Scheduling*, 1997. In conjunction with the Third Intenational Conference on Principles and Practice of Constraint Programming (CP97).

[17] R. K. Congram. *Polynomially Searchable Exponential Neighbourhoods for Sequencing Problems in Combinatorial Optimisation*. PhD thesis, University of Southampton, Faculty of Mathematical Studies, UK, 2000.

[18] R. K. Congram, C. N. Potts, and S. L. van de Velde. An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem. *INFORMS Journal on Computing*, 14(1):52–67, 2002.

[19] C. Cotta and J. M. Troya. Embedding branch and bound within evolutionary algorithms. *Applied Intelligence*, 18:137–153, 2003.

[20] E. Danna, E. Rothberg, and C. Le Pape. Exploring relaxation induced neighbourhoods to improve mip solutions. Technical report, ILOG, 2003.

[21] G. B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, 1963.

[22] C. Darwin. *The Origin of Species*. John Murray, 1859.

[23] L. Davis. Applying adaptive algorithms to epistatic domains. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 162–164, 1985.

[24] R. Dawkins. *The selfish gene*. Oxford University Press, Oxford, 1976.

[25] J. Denzinger and T. Offermann. On cooperation between evolutionary algorithms and other search paradigms. In *Proc. Congress on Evolutionary Computation (CEC) 1999*. IEEE Press, 1999.

[26] G. Desaulniers, J. Desrosiers, and M. Solomon, editors. *Column Generation*. Kluwer, 2005.

[27] G. Desaulniers, J. Desrosiers, and M. M. Solomon. Accelerating strategies in column generation methods for vehicle routing and crew scheduling problems. In C. C. Ribeiro and P. Hansen, editors, *Essays and Surveys in Metaheuristics*, pages 309–324. Kluwer, Boston, 2001.

[28] G. Dobson. Worst-case analysis of greedy heuristics for integer programming with nonnegative data. *Mathematics of Operations Research*, 7:515–531, 1982.

[29] M. Dorigo and T. Stützle. *Ant Colony Optimization*. MIT Press, Cambridge, MA, 2004.

[30] I. Dumitrescu and T. Stuetzle. Combinations of local search and exact algorithms. In G. R. Raidl, J.-A. Meyer, M. Middendorf, S. Cagnoni, J. J. R. Cardalda, D. W. Corne, J. Gottlieb, A. Guillot, E. Hart, C. G. Johnson, and E. Marchiori, editors, *Applications of Evolutionary Computation*, volume 2611 of *LNCS*, pages 211–223. Springer, 2003.

[31] H. Dyckhoff, G. Scheithauer, and J. Terno. Cutting and packing: An annotated bibliography. In M. Dell'Amico, F. Maffioli, and S. Martello, editors, *Annotated Bibliographies in Combinatorial Optimization*, pages 393–412. Wiley, 1997.

[32] M. Dyer and A. Frieze. Probabilistic analysis of the multidimensional knapsack problem. *Mathematics of Operations Research*, 14:162–176, 1989.

[33] A. Eiben and J. Smith. *Introduction to Evolutionary Computing*. Springer, Berlin Heidelberg, 2003.

[34] G. R. Filho and L. A. N. Lorena. Constructive genetic algorithm and column generation: an application to graph coloring. In *Proceedings of APORS 2000 - The Fifth Conference of the Association of Asian-Pacific Operations Research Societies within IFORS*, 2000.

[35] M. Fischetti and A. Lodi. Local Branching. *Mathematical Programming Series B*, 98:23–47, 2003.

[36] L. J. Fogel, A. J. Owens, and M. Walsh. *Artificial Intelligence through Simulated Evolution*. Wiley, Chichester, UK, 1966.

[37] A. P. French, A. C. Robinson, and J. M.Wilson. Using a hybrid genetic-algorithm/branch and bound approach to solve feasibility an optimization integer programming problems. *Journal of Heuristics*, 7:551–564, 2001.

[38] A. Fréville. The multidimensional 0-1 knapsack problem: An overview. *European Journal of Operational Research*, 155:1–21, 2004.

[39] A. Fréville and G. Plateau. An efficient preprocessing procedure for the multidimensional 0–1 knapsack problem. *Discrete Applied Mathematics*, 49:189–212, 1994.

[40] A. Fritsch. Verschnittoptimierung durch iteriertes Matching. Master's thesis, University of Osnabrück, Germany, 1994.

[41] J. E. Gallardo, C. Cotta, and A. J. Fernández. Solving the multidimensional knapsack problem using an evolutionary algorithm hybridized with branch and bound. In *Proceedings of the First International Work-Conference on the Interplay Between Natural and Artificial Computation*, volume 3562 of *LNCS*, pages 21–30. Springer, 2005.

[42] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York, 1979.

[43] B. Gavish and H. Pirkul. Efficient algorithms for solving the multiconstraint zero-one knapsack problem to optimality. *Mathematical Programming*, 31:78–105, 1985.

[44] P. Gilmore and R. Gomory. The theory and computation of knapsack functions. *Operations Research*, 14:1045–1075, 1966.

[45] P. C. Gilmore and R. E. Gomory. A linear programming approach to the cutting-stock problem (part I). *Operations Research*, 9:849–859, 1961.

[46] P. C. Gilmore and R. E. Gomory. A linear programming approach to the cutting-stock problem (part II). *Operations Research*, 11:363–888, 1963.

[47] P. C. Gilmore and R. E. Gomory. Multistage cutting-stock problems of two and more dimensions. *Operations Research*, 13:90–120, 1965.

[48] F. Glover and G. Kochenberger. Critical event tabu search for multidimensional knapsack problems. In I. Osman and J. Kelly, editors, *Metaheuristics: Theory and Applications*, pages 407–427. Kluwer Academic Publishers, 1996.

[49] F. Glover and G. Kochenberger, editors. *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*. Kluwer Academic Publishers, Norwell, MA, 2003.

[50] F. Glover and G. Kochenberger, editors. *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*. Kluwer Academic Publishers, 2003.

[51] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997.

[52] F. Glover, M. Laguna, and R. Martí. Fundamentals of scatter search and path relinking. *Control and Cybernetics*, 39(3):653–684, 2000.

[53] A. V. Goldberg and A. Marchetti-Spaccamela. On finding the exact solution of a zero-one knapsack problem. In *STOC '84: Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 359–368, New York, NY, USA, 1984. ACM Press.

[54] D. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.

[55] D. Goldberg and R. Lingle. Alleles, loci, and the travelling salesman problem. In J. J. Grefenstette, editor, *Proceedings of the First Int. Conf. on Genetic Algorithms*, pages 154–159. Lawrence Erlbaum, 1985.

[56] J. Gottlieb. On the effectivity of evolutionary algorithms for multidimensional knapsack problems. In C. Fonlupt et al., editors, *Proceedings of Artificial Evolution: Fourth European Conference*, volume 1829 of *LNCS*, pages 22–37. Springer, 1999.

[57] P. Hansen and N. Mladenović. An introduction to variable neighborhood search. In S. Voß, S. Martello, I. Osman, and C. Roucairol, editors, *Metaheuristics: advances and trends in local search paradigms for optimization*, pages 433–438. Kluwer Academic Publishers, 1999.

[58] P. Hansen and N. Mladenović. An introduction to variable neighborhood search. In S. Voss, S. Martello, I. Osman, and C. Roucairol, editors, *Meta-heuristics, Advances and Trends in Local Search Paradigms for Optimization*, pages 433–458. Kluwer, 1999.

[59] P. Hansen and N. Mladenović. A tutorial on variable neighborhood search. Technical Report G-2003-46, Les Cahiers du GERAD, HEC Montréal and GERAD, Canada, 2003.

[60] R. Hinterding. Mapping order-independent genes and the knapsack problem. In *Proc. of the First IEEE Int. Conf. on Evolutionary Computation*, pages 13–17. IEEE Press, 1994.

[61] J. Holland. *Adaptation In Natural and Artificial Systems*. University of Michigan Press, 1975.

[62] H. Hoos and T. Stützle. *Stochastic Local Search – Foundations and Applications*. Morgan Kaufmann, San Francisco, CA, 2004.

[63] E. Hopper. *Two-Dimensional Packing Utilising Evolutionary Algorithms and Other Meta-Heuristic Methods*. PhD thesis, University of Wales, Cardiff, U.K., 2000.

[64] S.-M. Hwang, C.-Y. Kao, and J.-T. Horng. On solving rectangle bin packing problems using GAs. In *Proceedings of the 1994 IEEE International Conference on Systems, Man, and Cybernetics*, pages 1583–1590. IEEE Press, 1997.

[65] N. Karmakar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984.

[66] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer, 2004.

[67] L. Khachiyan. A polynomial algorithm in linear programming (english translation). *Soviet Mathematics Doklady*, 20:191–194, 1979.

[68] S. Kirkpatrick, C. Gellat, and M. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[69] G. Klau, I. Ljubić, A. Moser, P. Mutzel, P. Neuner, U. Pferschy, G. Raidl, and R. Weiskircher. Combining a memetic algorithm with integer programming to solve the prize-collecting Steiner tree problem. In K. Deb et al., editors, *Genetic and Evolutionary Computation – GECCO 2004*, volume 3102 of *LNCS*, pages 1304–1315. Springer, 2004.

[70] K. Kostikas and C. Fragakis. Genetic programming applied to mixed integer programming. In M. Keijzer, U.-M. O' Reilly, S. M. Lucas, E. Costa, and T. Soule, editors, *Genetic Programming - EuroGP 2004*, volume 3003 of *LNCS*, pages 113–124. Springer, 2004.

[71] B. Kröger. Guillotineable bin packing: A genetic approach. *European Journal of Operational Research*, 84:545–661, 1995.

[72] A. Z.-Z. Lin, J. Bean, and I. C. C. White. A hybrid genetic/optimization algorithm for finite horizon partially observed markov decision processes. *Journal on Computing*, 16(1):27–38, 2004.

[73] A. Lodi, S. Martello, and M. Monaci. Two-dimensional packing problems: A survey. *European Journal of Operational Research*, 141:241–252, 2002.

[74] A. Lodi, S. Martello, and D. Vigo. Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems. *INFORMS Journal on Computing*, 11:345–357, 1999.

[75] A. Lodi, S. Martello, and D. Vigo. Recent advances on two-dimensional bin packing problems. *Discrete Applied Mathematics*, 123:373–390, 2002.

[76] A. Lodi, S. Martello, and D. Vigo. Models and bounds for two-dimensional level packing problems. *Journal of Combinatorial Optimization*, 8:363–379, 2004.

[77] J. Lorie and L. Savage. Three problems in capital rationing. *The Journal of Business*, 28:229–239, 1955.

[78] R. Lougee-Heimer. The Common Optimization INterface for Operations Research: Promoting open-source software in the operations research community. *IBM Journal of Research and Development*, 47(1):57–66, 2003.

[79] H. R. Lourenço, O. Martin, and T. Stützle. Iterated local search. In Glover and Kochenberger [50], pages 321–353.

[80] M. Lübbecke and J. Desrosiers. Selected topics in column generation. Les Cahiers du GERAD G-2002-64, HEC Montréal, Canada, 2002. *Operations Research* Scheduled for Publication in 53(6).

[81] A. Manne and H. Markowitz. On the solution of discrete programming problems. *Econometrica*, 25:84–110, 1957.

[82] A. Marino, A. Prügel-Bennett, and C. A. Glass. Improving graph colouring with linear programming and genetic algorithms. In *Proceedings of Eurogen99*, pages 113–118, 1999.

[83] S. Martello and P. Toth. A new algorithm for the 0-1 knapsack problem. *Management Science*, 34:633–644, 1988.

[84] S. Martello and D. Vigo. Exact solutions of the two-dimensional finite bin packing problem. *Management Science*, 44:388–399, 1998.

[85] A. Martin. General Mixed Integer Programming: Computational Issues for Branch-and-Cut Algorithms. In M. Jünger and D. Naddef, editors, *Computational Combinatorial Optimization: Optimal or Provably Near-Optimal Solutions*, volume 2241 of *LNCS*, pages 1–25. Springer, 2001.

[86] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, Berlin, 1996.

[87] M. Monaci. *Algorithms for Packing and Scheduling Problems*. PhD thesis, University of Bologna, Italy, 2002.

[88] M. Monaci and P. Toth. A set covering based heuristic approach for bin-packing problems. Technical Report OR-03-1, Dipartimento di Elettronica, Informatica e Sistemistica, University of Bologna, 2003. Under revision for INFORMS Journal on Computing.

[89] P. Moscato. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Technical Report C3P 826, Pasadena, CA, 1989.

[90] P. Moscato. Memetic algorithms: A short introduction. In D. Corne et al., editors, *New Ideas in Optimization*, pages 219–234. McGraw Hill, 1999.

[91] A. Nagar, S. S. Heragu, and J. Haddock. A meta-heuristic algorithm for a bi-criteria scheduling problem. *Annals of Operations Research*, 63:397–414, 1995.

[92] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons, 1988.

[93] J. F. Oliveira and J. S. Ferreira. A faster variant of the Gilmore and Gomory technique for cutting stock problems. *JORBEL – Belgium Journal of Operations Research, Statistics and Computer Science*, 34(1):23 –38, 1994.

[94] H. Pirkul. A heuristic solution procedure for the multiconstraint zero-one knapsack problem. *Naval Research Logistics*, 34:161–172, 1987.

[95] D. Pisinger. An expanding-core algorithm for the exact 0–1 knapsack problem. *European Journal of Operational Research*, 87:175–187, 1995.

[96] D. Pisinger. A minimal algorithm for the 0–1 knapsack problem. *Operations Research*, 45:758–767, 1997.

[97] D. Pisinger. Core problems in knapsack algorithms. *Operations Research*, 47:570–575, 1999.

[98] D. Pisinger and M. Sigurd. Using decomposition techniques and constraint programming for solving the two-dimensional bin packing problem. Technical Report 03/01, University of Copenhagen, Denmark, 2003. Submitted for publication.

[99] A. Plateau, D. Tachat, and P. Tolla. A hybrid search combining interior point methods and metaheuristics for 0-1 programming. *International Transactions in Operational Research*, 9:731–746, 2002.

[100] J. Puchinger and G. R. Raidl. An evolutionary algorithm for column generation in integer programming: an effective approach for 2D bin packing. In X. Yao et al., editor, *Parallel Problem Solving from Nature - PPSN VIII*, volume 3242 of *LNCS*, pages 642–651. Springer, 2004.

[101] J. Puchinger and G. R. Raidl. Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification. In *Proceedings of the First International Work-Conference on the Interplay Between Natural and Artificial Computation*, volume 3562 of *LNCS*, pages 41–53. Springer, 2005.

[102] J. Puchinger and G. R. Raidl. Models and algorithms for three-stage two-dimensional bin packing. *European Journal of Operational Research (EJOR), Feature Issue on Cutting and Packing*, 2005. Accepted for publication.

[103] J. Puchinger and G. R. Raidl. Relaxation guided variable neighborhood search. In *Proceedings of the XVIII Mini EURO Conference on VNS*, Tenerife, Spain, 2005.

[104] J. Puchinger, G. R. Raidl, and M. Gruber. Cooperating memetic and branch-and-cut algorithms for solving the multidimensional knapsack problem. In *Proceedings of MIC2005, the 6th Metaheuristics International Conference*, pages 775–780, Vienna, Austria, 2005.

[105] J. Puchinger, G. R. Raidl, and G. Koller. Solving a real-world glass cutting problem. In J. Gottlieb and G. R. Raidl, editors, *Evolutionary Computation in Combinatorial Optimization – EvoCOP 2004*, volume 3004 of *LNCS*, pages 162–173. Springer, 2004.

[106] J. Puchinger, G. R.Raidl, and U. Pferschy. The core concept for the multidimensional knapsack problem. In *Evolutionary Computation in Combinatorial Optimization - EvoCOP 2006*, LNCS. Springer, 2006.

[107] G. R. Raidl. An improved genetic algorithm for the multiconstrained 0–1 knapsack problem. In D. Fogel et al., editors, *Proceedings of the 5th IEEE International Conference on Evolutionary Computation*, pages 207–211. IEEE Press, 1998.

[108] G. R. Raidl and H. Feltl. An improved hybrid genetic algorithm for the generalized assignment problem. In H. M. Haddadd et al., editors, *Proceedings of the 2003 ACM Symposium on Applied Computing*, pages 990–995. ACM Press, 2004.

[109] G. R. Raidl and J. Gottlieb. Empirical analysis of locality, heritability and heuristic bias in evolutionary algorithms: A case study for the multidimensional knapsack problem. *Evolutionary Computation Journal*, 13(4), to appear 2005.

[110] I. Rechenberg. *Evolutionsstrategie, Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution*. Frommann-Holzboog Verlag, 1973.

[111] A. Schrijver. *Combinatorial optimization : polyhedra and efficiency*. Springer, 2003.

[112] H.-P. Schwefel. *Evolution and Optimum Seeking*. Wiley, New York, 1995.

[113] S. Senju and Y. Toyoda. An approach to linear programming with 0–1 variables. *Management Science*, 15:196–207, 1968.

[114] W. Shih. A branch and bound method for the multiconstraint zero-one knapsack problem. *Journal of the Operational Research Society*, 30:369–378, 1979.

[115] M. M. Sigurd. *Column Generation Methods and Application*. PhD thesis, University of Copenhagen, Denmark, 2004.

[116] A. T. Staggemeier, A. R.Clark, U. Aickelin, and J. Smith. A hybrid genetic algorithm to solve a lot-sizing and scheduling problem. In *16th triannual Conference of the International Federation of Operational Research Societies*, Edinburgh, 2002.

[117] G. Syswerda. Schedule optimization using genetic algorithms. pages 332–349. Int. Thomson Computer Press, 1991.

[118] S. Talukdar, L. Baeretzen, A. Gove, and P. de Souza. Asynchronous teams: Cooperation schemes for autonomous agents. *Journal of Heuristics*, 4:295–321, 1998.

[119] S. Talukdar, S. Murty, and R. Akkiraju. Asynchronous teams. In Glover and Kochenberger [50], pages 537–556.

[120] H. Tamura, A. Hirahara, I. Hatono, and M. Umano. An approximate solution method for combinatorial optimisation. *Transactions of the Society of Instrument and Control Engineers*, 130:329–336, 1994.

[121] P. Thompson and J. Orlin. The theory of cycle transfers. Technical Report OR-200-89, MIT Operations Research Center, Boston, MA, 1989.

[122] P. Thompson and H. Psaraftis. Cycle transfer algorithm for multivehicle routing and scheduling problems. *Operations Research*, 41:935–946, 1993.

[123] A. M. Turing. Intelligent machinery. In D. Ince, editor, *Collected Works of A.M. Turing : Mechanical Intelligence*. North Holland, 1992.

[124] J. Valério de Carvalho. Using extra dual cuts to accelerate convergence in column generation. *INFORMS Journal on Computing*, 2003. To appear.

[125] F. Vanderbeck. A nested decomposition approach to a 3-stage 2-dimensional cutting stock problem. *Management Science*, 47(2):864–879, 1998.

[126] F. Vanderbeck. On Dantzig-Wolfe decomposition in integer programming and ways to perform branching in a branch-and-price algorithm. *Operations Research*, 48(1):111–128, 2000.

[127] M. Vasquez and J.-K. Hao. A hybrid approach for the 0–1 multidimensional knapsack problem. In *Proceedings of the Int. Joint Conference on Artificial Intelligence 2001*, pages 328–333, 2001.

[128] M. Vasquez and Y. Vimont. Improved results on the 0-1 multidimensional knapsack problem. *European Journal of Operational Research*, 165:70–81, 2005.

[129] V. V. Vazirani. *Approximation algorithms*. Springer, 2001.

[130] H. M. Weingartner and D. N. Ness. Methods for the solution of the multidimensional 0/1 knapsack problem. *Operations Research*, 15:83–103, 1967.

[131] D. Whitley. The GENITOR algorithm and selection pressure: why rank-based allocation of reproductive trial is best. In J. D. Schaffer, editor, *Proceedings of the Third Int. Conf. on Genetic Algorithms*, pages 116–121. Morgan Kaufmann, 1989.

[132] L. A. Wolsey. *Integer Programming*. Wiley-Interscience, 1998.

[133] D. L. Woodruff. A chunking based selection strategy for integrating metaheuristics with branch and bound. In S. Voss et.al., editor, *Metaheuristics: Advances and Trends in Local Search Paradigms for Optimization*, pages 499–511. Kluwer Academic Publishers, 1999.

Curriculum Vitae

Personal Information

- Name: Jakob Puchinger
- Date and place of birth: May 24, 1978 in Vienna, Austria
- Nationality: Austrian

Education

- since 07/2003: PhD student in computer science, Vienna University of Technology. Supervisor: Prof. Günther Raidl.
- 10/1998 - 06/2003: Computer science studies with graduation to "Diplom-Ingenieur" (MSc), Vienna University of Technology.
- 10/1996 - 03/1998: Physics studies, Swiss Federal Institute of Technology Zurich, Switzerland.
- 09/1983 - 06/1996: Preschool, elementary and secondary school, Lycée Français de Vienne, Vienna, Austria. Baccalauréat/Matura (school leaving examination) passed with distinction.

Work Experience

- since 3/2005: Research and teaching assistant, Algorithms and Data Structures Group, Institute of Computer Graphics and Algorithms, Vienna University of Technology.
- since 6/2003: Employed in the FWF project *Combining Memetic Algorithms with Branch and Cut and Price for Some Network Design Problem* under grant P16263-N04, Algorithms and Data Structures Group, Institute of Computer Graphics and Algorithms, Vienna University of Technology.

Publications

Refereed Journal Articles

- Jakob Puchinger and Günther R. Raidl. Models and algorithms for three-stage two-dimensional bin packing. *European Journal of Operational Research*, feature issue on Cutting and Packing. Accepted for publication 2005.

Refereed Conference Papers

- Jakob Puchinger, Günther R. Raidl, and Ulrich Pferschy. The Core Concept for the Multidimensional Knapsack Problem. To appear in *Evolutionary Computation in Combinatorial Optimization - EvoCOP 2006*, Budapest, Hungary. LNCS, Springer 2006.
- Jakob Puchinger and Günther R. Raidl. Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification. In *Proceedings of the First International Work-Conference on the Interplay Between Natural and Artificial Computation*, volume 3562 of LNCS, pages 41-53. Springer, 2005.
- Jakob Puchinger, Günther R. Raidl, and Martin Gruber. Cooperating memetic and branch-and-cut algorithms for solving the multidimensional knapsack problem In *Proceedings of the 6th Metaheuristics International Conference (MIC)*, pages 775-780, Vienna, Austria, 2005.
- Jakob Puchinger and Günther R. Raidl. Relaxation Guided Variable Neighborhood Search. In *Proceedings of the XVIII Mini EURO Conference on VNS*. Tenerife, Spain, 2005.

- Jakob Puchinger, Günther R. Raidl, and Gabriele Koller. Solving a real-world glass cutting problem. In Jens Gottlieb and Günther R. Raidl, editors, *Evolutionary Computation in Combinatorial Optimization - EvoCOP 2004*, volume 3004 of LNCS, pages 162-173. Springer, 2004.
- Jakob Puchinger and Günther R. Raidl. An evolutionary algorithm for column generation in integer programming: an effective approach for 2D bin packing. In X. Yao et. al, editor, *Parallel Problem Solving from Nature - PPSN VIII*, volume 3242 of LNCS, pages 642-651. Springer, 2004.

Thesis

- Jakob Puchinger. Methods for solving a glass-cutting problem (Verfahren zur Lösung eines Glasverschnittproblems). Master's thesis, Vienna University of Technology, Institute of Computer Graphics and Algorithms, May 2003. supervised by G. R. Raidl and G. Koller.