Universidade Estadual de Campinas
Instituto de Computação

Vinícius Loti de Lima

Integer Programming Based Methods Applied to Cutting, Packing, and Scheduling

Métodos Baseados em Programação Inteira Aplicados em Corte, Empacotamento e Escalonamento

CAMPINAS
2021

Vinícius Loti de Lima

Integer Programming Based Methods Applied to Cutting, Packing, and Scheduling

Métodos Baseados em Programação Inteira Aplicados em Corte, Empacotamento e Escalonamento

Tese apresentada ao Instituto de Computação da Universidade Estadual de Campinas como parte dos requisitos para a obtenção do título de Doutor em Ciência da Computação.

Thesis presented to the Institute of Computing of the University of Campinas in partial fulfillment of the requirements for the degree of Doctor in Computer Science.

Supervisor/Orientador: Prof. Dr. Flávio Keidi Miyazawa
Co-supervisor/Coorientador: Prof. Dr. Thiago Alves de Queiroz

Este exemplar corresponde à versão final da Tese defendida por Vinícius Loti de Lima e orientada pelo Prof. Dr. Flávio Keidi Miyazawa.

CAMPINAS
2021

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca do Instituto de Matemática, Estatística e Computação Científica
Silvania Renata de Jesus Ribeiro - CRB 8/6592

Lima, Vinícius Loti de, 1995-
L628i Integer programming based methods applied to cutting, packing, and scheduling / Vinícius Loti de Lima. – Campinas, SP : [s.n.], 2021.

Orientador: Flávio Keidi Miyazawa.
Coorientador: Thiago Alves de Queiroz.
Tese (doutorado) – Universidade Estadual de Campinas, Instituto de Computação.

1. Otimização combinatória. 2. Programação linear. 3. Programação inteira. I. Miyazawa, Flávio Keidi, 1970-. II. Queiroz, Thiago Alves de. III. Universidade Estadual de Campinas. Instituto de Computação. IV. Título.

Informações para Biblioteca Digital

Título em outro idioma: Métodos baseados em programação inteira aplicados em problemas de corte, empacotamento e escalonamento
Palavras-chave em inglês:
Combinatorial optimization
Linear programming
Integer programming
Área de concentração: Ciência da Computação
Titulação: Doutor em Ciência da Computação
Banca examinadora:
Flávio Keidi Miyazawa [Orientador]
Eduardo Uchoa Barboza
Luciana Salete Buriol
Marco Lübbecke
Stefan Irnich
Yoshiko Wakabayashi
Jean-François Côté
Pedro Augusto Munari Junior

Data de defesa: 14-12-2021
Programa de Pós-Graduação: Ciência da Computação

Identificação e informações acadêmicas do(a) aluno(a)
- ORCID do autor: https://orcid.org/0000-0002-4805-4468
- Currículo Lattes do autor: http://lattes.cnpq.br/7482208715511395

Universidade Estadual de Campinas
Instituto de Computação

Vinícius Loti de Lima

Integer Programming Based Methods Applied to Cutting, Packing, and Scheduling

Métodos Baseados em Programação Inteira Aplicados em Corte, Empacotamento e Escalonamento

Banca Examinadora:

- Prof. Dr. Flávio Keidi Miyazawa
  IC/UNICAMP
- Prof. Dr. Eduardo Uchoa Barboza
  TEP/UFF
- Profa. Dra. Luciana Salete Buriol
  INF/UFRGS
- Prof. Dr. Marco Lübbecke
  RWTH AACHEN UNIVERSITY
- Prof. Dr. Stefan Irnich
  JGU - Johannes Gutenberg-Universität Mainz
- Profa. Dra. Yoshiko Wakabayashi
  IME/USP
- Prof. Dr. Jean-François Côté
  FSA/Ulaval
- Prof. Dr. Pedro Augusto Munari Junior
  DEP/UFSCar

A ata da defesa, assinada pelos membros da Comissão Examinadora, consta no SIGA/Sistema de Fluxo de Dissertação/Tese e na Secretaria do Programa da Unidade.

Campinas, 14 de dezembro de 2021

Agradecimentos

Como um adepto do conceito de originação dependente, acredito na contribuição de todos os eventos do universo para a formação deste trabalho. Naturalmente, algumas pessoas me remetem a influências bastante diretas e gostaria de destacá-las. Em particular, agradeço às pessoas mais importantes para mim: meus pais. Agradeço ao meu orientador, Flávio, que é para mim uma grande inspiração, não só profissional, mas também pessoal. Agradeço ao meu amigo e coorientador Thiago, que me introduziu ao tema desta tese. Agradeço a todos os meus coautores, em particular, ao Manuel Iori e ao José Valério de Carvalho, que contribuíram de uma forma imensa para minha formação como pesquisador. Agradeço a todos que me deram algum suporte ou passaram pela minha vida ao longo dos anos. Finalmente, agradeço à FAPESP, pelo apoio financeiro (processo 2017/11831-1).

Resumo

Esta tese foca em otimização combinatória, um dos grandes campos em otimização. Esse campo consiste de problemas de decisão, nos quais é dado um conjunto discreto potencialmente enorme de soluções possíveis e deve-se encontrar uma única solução que otimize uma dada função objetivo. Esses problemas têm um grande número de aplicações no mundo real, principalmente em logística industrial e em cadeia de suprimentos.

Primeiro, estudamos formulações de fluxo em redes que podem ser aplicadas a problemas gerais de otimização combinatória. Apresentamos um *survey* discutindo as fundamentações teóricas e as principais aplicações bem-sucedidas dos chamados modelos de fluxo em arcos pseudo-polinomiais. Em seguida, propomos novos métodos exatos para resolver tais modelos, envolvendo geração de colunas, regras de *branching* especializadas e estratégias de fixação de variáveis. Aplicamos os métodos de solução propostos a problemas bem estudados da literatura de corte, empacotamento e escalonamento, incluindo, por exemplo, os clássicos *bin packing problem* e *cutting stock problem*. Nossos experimentos computacionais mostram a eficácia dos métodos propostos, resolvendo na otimalidade um grande número de instâncias benchmark em aberto, de vários problemas.

A segunda parte desta tese dá atenção especial à área de corte e empacotamento bidimensional, que está entre as áreas mais estudadas em otimização combinatória. Apresentamos uma revisão das referências mais relevantes que surgiram nas últimas duas décadas e propomos uma biblioteca online para organizar sistematicamente os materiais mais relevantes sobre tais problemas. Essas contribuições podem facilitar pesquisas futuras na área ativa de corte e empacotamento bidimensional.

Nossa última contribuição diz respeito a um problema do mundo real surgido de uma empresa italiana de produção de carne. O problema é composto por um grande conjunto de decisões complexas, envolvendo a alocação diária de mão de obra e o escalonamento de pedidos na empresa. Para resolver esse problema, propomos uma heurística construtiva de três fases, implementada em um *framework multi-start*. Nosso algoritmo supera em média as decisões tomadas pela estratégia anterior da empresa.

Abstract

This thesis focuses on combinatorial optimization, one of the major optimization fields. This field consists of decision problems, in which we are given a potentially huge discrete set of possible solutions, and we must find a single solution that optimizes an objective function. Such problems have an extensive number of real world applications, mainly in industrial logistics and supply chain.

First, we study network flow formulations that can be applied to general combinatorial optimization problems. We present a survey discussing theoretical foundations and main successful applications of the so-called pseudo-polynomial arc flow models. Then, we propose novel exact solution methods to solve such models, involving column generation, specialized branching rules, and variable fixing strategies. We apply the proposed solution methods to well-studied cutting, packing, and scheduling problems from the literature, including, for instance, the classical bin packing and cutting stock problems. Our computational experiments show the effectiveness of the proposed methods by solving to proven optimality an extensive number of open benchmark instances of several problems.

The second part of this thesis give special attention to the area of two-dimensional cutting and packing problems, which is among the most studied areas in combinatorial optimization. We survey the most relevant references that appeared in the last two decades and propose an online library to systematically arrange the most relevant materials regarding such problems. These contributions can facilitate future research in the active area of two-dimensional cutting and packing.

Our last contribution concerns a real world problem arising from an Italian meat-producing company. The problem consists of a large set of complex decisions, involving the daily workforce allocation and scheduling of orders in the company. To solve this problem, we propose a three-phase constructive heuristic, which is embedded in a multi-start framework. Our algorithm outperforms in average the decisions made by the former strategy of the company.

Contents

Chapter 1

Introduction

This thesis focuses on combinatorial optimization problems. These are decisions problems, in which we are given a (potentially huge) discrete set of possible solutions, and we must find a single solution that optimizes an objective function. The list of applications of such problems is extensive, and a large interest in their study is related to logistics and supply chain applications. These include, for instance, cutting and packing, scheduling, routing, lot-sizing, facility location, train timetabling, workforce allocation, and airline network design problems.

A direct way of solving a combinatorial optimization problem (i.e., finding an optimal solution) is by a complete inspection of the set of solutions. However, this is generally non-practical, since in real-world problems the set of solutions is often prohibitively huge. It is not hard to find real-world examples in which it would take centuries of computational time to explore all possible solutions. For that, the combinatorial optimization literature is mainly concerned with the development of methods to find an optimal solution while avoiding to explore the full set of solutions. The literature divides solution methods into two categories: exact methods and heuristics. Exact methods are guaranteed to return an optimal solution (if enough time is allowed for their complete execution). The success of these methods depends on techniques to eliminate large subsets of solutions that are guaranteed to be sub-optimal, without the need to fully explore them. Nonetheless, in the worst case, exact methods still have to explore all solutions, so they are often non-practical. Heuristics are methods that do not always return an optimal solution. Their main interest is in quickly finding a good-quality solution, so as to terminate the search within a practical time. In this way, this kind of method is usually preferred in a real-world scenario. In general, exact methods and heuristics are related. On the one hand, exact methods can be adapted to work as heuristics in practice by imposing a computational time limit on their search. On the other hand, heuristics are often explored as a way to accelerate the search within exact methods.

The main contributions of this thesis are related to new solution methods to solve combinatorial optimization problems in practice. We propose exact methods that can be applied to a large class of problems in the literature, and we also develop a heuristic to solve a problem arising from a real-world application. We also propose surveys that extensively review the literature on two well-studied areas, namely pseudo-polynomial arc flow models and two-dimensional cutting and packing problems.

1.1 Preliminaries

In the remainder of this thesis, we assume that the reader is familiar with the basic theory of Linear Programming (LP), Mixed-Integer Linear Programming (MILP), algorithmic complexity, and other optimization concepts that can be found in classical books (see, e.g., Garey and Johnson [144] and Nemhauser and Wolsey [250]). In the following, we briefly discuss some related concepts to contextualize and motivate the contents of this thesis. All such concepts are discussed in detail in the following chapters.

A network flow formulation aims at finding a minimum-cost flow in a network while satisfying side constraints (see, e.g., Ahuja et al. [1]). The list of applications of network flow formulations is extensive, including vehicle routing, telecommunication network planning, train timetabling, cutting, packing, scheduling, and project management. Network flow formulations modeled by MILP can derive an *arc flow model*, in which decision variables correspond to the flow on individual arcs of the network.

The LP relaxation plays a key role in the solution of a MILP model. These models are generally solved by branch-and-bound, and the bound at each node is obtained from such relaxation. Consequently, having a strong LP relaxation is important to improve pruning, and, hence, to guarantee a practical limit in the number of branch-and-bound nodes explored. In addition, strong LP relaxations usually imply that optimal fractional solutions are a good approximation of an optimal integer solution. Hence, rounding MILP heuristics usually work well in strong models.

Dantzig-Wolfe reformulation is one of the most successful ways to derive MILP models with strong relaxations (see, e.g., Vanderbeck and Savelsbergh [324]). The models resulting from the reformulation have an exponential number of variables and are usually solved by sophisticated methods, as, e.g., branch-and-cut-and-price. Nonetheless, such models can be reformulated as an arc flow model with same relaxation strength and often of pseudo-polynomial size.

The majority of the methodological contributions of this thesis is related to the solution of such pseudo-polynomial arc flow models with strong LP relaxations. One of the most popular ways to solve such models is by general MILP solvers. The potential of such solvers has increased consistently in the last decades, and they usually represent the best alternative to solve small- or medium-sized models. However, hard instances of many practical problems usually derive pseudo-polynomial arc flow models that are too large to be efficiently solved by such solvers. This gives rise to one of the main concerns of this thesis: "how to develop solution methods to efficiently solve large arc flow models with strong relaxations, while still taking advantage of the continuously increasing potential of general MILP solvers?".

1.2 Contents of this Thesis

This thesis is composed of a collection of six articles, which were written to be submitted to international indexed journals or conferences. These articles are presented in the following chapters. Each of the six following chapters presents a single article, and chapter 8 presents our concluding remarks. For a better organization of the thesis, we group the six articles

into three different parts, which are described in the following.

1.2.1 First Part

The first part reviews and proposes exact methods for pseudo-polynomial arc flow formulations.

Chapter 2 reviews over 100 references related to arc flow models, focusing on models of pseudo-polynomial size. These models have become popular in the last two decades, and, to the best of our knowledge, this is the first survey that systematically reviews their theoretical foundations and main successful applications. This chapter results from joint work with Cláudio Alves, François Clautiaux, Manuel Iori, and José M. Valério de Carvalho, and an article has been published as an invited review by the *European Journal of Operational Research* [101]. In the following, we provide its abstract.

> *Abstract.* Network flow formulations are among the most successful tools to solve optimization problems. Such formulations correspond to determining an optimal flow in a network. One particular class of network flow formulations is the arc flow, where variables represent flows on individual arcs of the network. For $\mathcal{NP}$-hard problems, polynomial-sized arc flow models typically provide weak linear relaxations and may have too much symmetry to be efficient in practice. Instead, arc flow models with a pseudo-polynomial size usually provide strong relaxations and are efficient in practice. The interest in pseudo-polynomial arc flow formulations has grown considerably in the last twenty years, in which they have been used to solve many open instances of hard problems. A remarkable advantage of pseudo-polynomial arc flow models is the possibility to solve practical-sized instances directly by a Mixed Integer Linear Programming solver, avoiding the implementation of complex methods based on column generation.
>
> In this survey, we present theoretical foundations of pseudo-polynomial arc flow formulations, by showing a relation between their network and Dynamic Programming (DP). This relation allows a better understanding of the strength of these formulations, through a link with models obtained by Dantzig-Wolfe decomposition. The relation with DP also allows a new perspective to relate state-space relaxation methods for DP with arc flow models. We also present a dual point of view to contrast the linear relaxation of arc flow models with that of models based on paths and cycles. To conclude, we review the main solution methods and applications of arc flow models based on DP in several domains such as cutting, packing, scheduling, and routing.

Chapter 3 continues the study on pseudo-polynomial arc flow models by proposing an exact solution framework, which is tailored to be efficient in solving models with strong LP relaxations. The major contributions of this chapter have a practical nature, but we also propose a number of theoretical results. The proposed framework successfully improves the state-of-the-art of a number of well-studied problems. This chapter results from joint work with Manuel Iori and Flávio K. Miyazawa. A preliminary version of this chapter

has been presented at the 22*nd Conference on Integer Programming and Combinatorial Optimization* (IPCO XXII). A full article derived from this chapter is currently under review by an international indexed journal. In the following, we provide its abstract.

> *Abstract.* We address the solution of Mixed Integer Linear Programming (MILP) models with strong relaxations that are derived from Dantzig-Wolfe decompositions and allow a pseudo-polynomial pricing algorithm. We exploit their network-flow characterization and provide a framework based on column generation, reduced-cost variable-fixing, and a highly asymmetric branching scheme that allows us to take advantage of the potential of the current MILP solvers. We apply our framework to a variety of cutting and packing problems from the literature. The efficiency of the framework is proved by extensive computational experiments, in which a significant number of open instances could be solved to proven optimality for the first time.

Chapter 4 is still concerned with the proposal of new exact methods for pseudo-polynomial arc flow models with strong relaxations. However, the main motivation of this chapter is applications on parallel machine scheduling problems with the objective of minimizing weighted completion times. For that, the chapter presents novel solution methods for general arc flow models, but also a new branching scheme tailored for this class of scheduling problems. This chapter results from joint work with Manuel Iori, Flávio K. Miyazawa, Roberto Roberti, and José M. Valério de Carvalho. The chapter presents a preliminary report from a project that is still under development. An ongoing part of the project is related to the investigation of strengthening of arc flow models with weak linear relaxations. Once the project is finished, a full article will be submitted to an international indexed journal. In the following, we provide the abstract of the report.

> *Abstract.* State-of-the-art for many scheduling problems consists of network flow formulations, in particular arc flow models. These are Mixed Integer Linear Programming models that aim at finding a minimum-cost flow in a network, while satisfying side constraints. This paper proposes general solution methods for such models, motivated by applications on identical parallel machine scheduling problems with the objective to minimize weighted completion times. The proposals include a general branching scheme and a method to deal with dual infeasibility inherent from the limited precision of general Linear Programming solvers. Extensive computational experiments show that the overall algorithm can solve all open benchmark instances to proven optimality.

1.2.2 Second Part

The second part of the thesis is devoted to two-dimensional cutting and packing problems. It contains an extensive literature review and an introduction to a new online library.

Chapter 5 reviews over 180 references related to solution methods for orthogonal two-dimensional cutting and packing problems. To the best of our knowledge, the last

extensive review on the same class of problems was presented almost two decades ago, and since then, the literature on this area has grown considerably. This chapter results from joint work with Manuel Iori, Silvano Martello, Flávio K. Miyazawa and Michele Monaci. A resulting article has been published as an invited review by the *European Journal of Operational Research* [175]. In the following, we provide its abstract.

> *Abstract.* We survey the main formulations and solution methods for two-dimensional orthogonal cutting and packing problems, where both items and bins are rectangles. We focus on exact methods and relaxations for the four main problems from the literature: finding a packing with minimum height, packing the items into the minimum number of bins, finding a packing of maximum value, and determining the existence of a feasible packing.

Chapter 6 introduces an online library to facilitate research on two-dimensional cutting and packing problems. The library is available at `http://or.dei.unibo.it/library/2dpacklib`. This chapter results from joint work with Manuel Iori, Silvano Martello, and Michele Monaci. A resulting article has been published in *Optimization Letters* [176]. In the following, we provide its abstract.

> *Abstract.* Two-dimensional cutting and packing problems model a large number of relevant industrial applications.The literature on practical algorithms for such problems is very large. We introduce the `2DPackLib`, a library on two-dimensional orthogonal cutting and packing problems. The library makes available, in a unified format, 25 benchmarks from the literature for a total of over 3000 instances, provides direct links to surveys and typologies, and includes a list of relevant links.

1.2.3 Third Part

The third and last part of the thesis consists of a single chapter and is concerned with a problem arising from a collaboration with an Italian meat-producing company.

Chapter 7 studies a real-world problem related to daily workforce and order scheduling. This problem consists of a large set of decision variables and complex constraints. For a practical solution of the problem, we developed an algorithm based on a two-phase constructive heuristic embedded within a random multi-start framework. This chapter results from joint work with Beatrice Bolsi, Thiago A. de Queiroz, and Manuel Iori and an article has been presented in the *Advances in Production Management Systems 2021* Conference [43]. In the following paragraph, we provide its abstract. We are currently working on an extended version of this work, and the resulting article will be submitted to an international indexed journal.

> *Abstract.* We address a problem from a meat company, in which orders are produced in two stages, consisting of preparing meats on benches and allocating them to conveyors to be packed in disposable trays. In an environment where machines are unrelated, the company has to take daily decisions on the

number and start time of working periods, the number of workers and their allocation to machines, and the scheduling of activities to satisfy the required orders. The objective of the problem is to minimize, in a lexicographic way, the number of unscheduled activities, the weighted tardiness, and the total production cost. To solve the problem, we propose a multi-start random constructive heuristic, which tests different combinations of number of workers in the machines and for each combination produces many different schedules of the orders. The results of our computational experiments over realistic instances show that the heuristic is effective and can support the company on its daily decisions.

Chapter 2

Arc Flow Formulations Based on Dynamic Programming: Theoretical Foundations and Applications

2.1 Introduction

Optimization problems can assume different characterizations, each allowing the reduction of the original problem to other optimization problems, leading to, possibly, different solution methods. One of such characterizations is based on a network, that is, a directed graph with costs on the arcs, and is the base of the well-known network flow problems (see, e.g., Ahuja et al. [1]). A *network flow problem* is an optimization problem that requires to determine an optimal flow on a network, by satisfying flow conservation on each node and possible additional side constraints on the flow on the arcs. The list of real-life problems that can be solved as network flow problems is extensive, including not only direct applications (i.e., the network is an input of the problem), as vehicle routing, telecommunication network planning, and train scheduling (see, e.g., Gouveia et al. [154], Minoux [242], and Cacchiani and Toth [61]), but also indirect ones (i.e., the network must be constructed), as cutting and packing, scheduling, and project management (see, e.g., Delorme and Iori [107], Kramer et al. [198], and Riedler et al. [278]).

In general, any formulation that corresponds to solving a network flow problem is called *network flow formulation*. Following Ahuja et al. [1], the two main classes of network flow formulations are the path (and cycles) flow formulations and the arc flow formulations. *Path (and cycles) flow formulations* (called path flow formulations for short in the following) have variables corresponding to flow on paths and cycles of the network, and are frequently associated with set-covering, set-packing, or set-partitioning models. In contrast, *arc flow formulations* have variables corresponding to flow on individual arcs of the network. The Flow Decomposition Theorem by Ahuja et al. [1] guarantees that the two formulations produce equivalent models when based on the same network, in the sense that any solution of one model can be mapped into a solution of the other model.

Arc flow formulations have a linear number of variables with respect to the number of arcs in the network. Often, such formulations can be used to solve medium-sized instances

directly by a commercial Mixed Integer Linear Programming (MILP) solver, avoiding the implementation of complex methods. On the other hand, path flow formulations may have a much larger number of variables, as the number of paths and cycles can be exponential with respect to the number of arcs of the network. Such exponential path flow formulations are typically solved by sophisticated methods based on column generation and branch-and-price algorithms (see, e.g., Desaulniers et al. [112] and Sadykov and Vanderbeck [290]).

Many $\mathcal{NP}$-hard problems can be formulated as compact (i.e., polynomial-sized) models. Although their size is favorable, these models are usually associated with weak linear relaxations and very symmetrical solution spaces, leading to an overly extensive enumeration on branch-and-bound algorithms. To overcome such inefficiency, one may rely on the Dantzig-Wolfe (DW) decomposition (see Dantzig and Wolfe [100]), which can lead to models with stronger linear relaxations and less symmetry. The models resulting from DW decomposition usually have exponentially more variables than the original model and are solved by column generation. When the associated pricing problem is solved by *Dynamic Programming* (DP), the model can be seen as a path flow formulation, where each variable corresponds to a path in the network inherent from the DP problem. From this observation, one can devise an equivalent arc flow model based on the DP network. Originated by a DW decomposition, the resulting arc flow model will, possibly, have a strong linear relaxation and a less symmetrical solution space, compared to the original polynomial-sized model.

To obtain strong models by a DW decomposition, one may have to pay the price of having an increased complexity on the pricing problem, as such complexity usually becomes pseudo-polynomial or exponential. When the size of the DP network from the pricing problem is pseudo-polynomial, the resulting arc flow model can still be used in practice to solve medium-sized instances by means of a MILP solver. On the other hand, when the DP network is too large (possibly exponential), one can obtain smaller (yet still strong) pseudo-polynomial arc flow models relying on a state-space relaxation of the DP. Different state-space relaxations lead to arc flow models with different sizes and different strength, allowing a flexible possibility to balance size and strength of arc flow models.

Seminal works and previous surveys

To the best of our knowledge, Ford and Fulkerson [134] were the first to propose a pseudo-polynomial arc flow model, which was based on a time-expanded network and was used to solve the maximal dynamic flow problem. Already in the sixties, Shapiro [295] studied the relation between DP and Integer Programming, by modeling the knapsack problem as a pseudo-polynomial network flow problem. More than ten years later, Wolsey [337] proposed a general methodology to build packing and covering models from the set of feasible solutions of DP problems. As an application, he introduced a pseudo-polynomial arc flow model based on a DP network of the knapsack problem for the cutting stock problem (CSP), a strongly $\mathcal{NP}$-hard problem. Due to the context, Wolsey [337] did not present computational experiments for the model, and such formulations did not get much attention for decades. It was more than twenty years later that Valério de Carvalho

[314] independently proposed this model, used it to solve to (integer) optimality open instances in the CSP literature, and showed that it is equivalent to the Gilmore-Gomory model [149, 150], a path flow model obtained from a DW decomposition.

Since the work by Valério de Carvalho [314], the popularity of pseudo-polynomial arc flow formulations based on DP has grown considerably, and several different applications to $\mathcal{NP}$-hard problems have appeared in the literature. Large instances have been solved for the first time to proven optimality by arc flow models in several cutting, packing, and scheduling problems. In addition, arc flow formulations have effectively modeled complex problems involving, for instance, multiple-stage cutting processes or the presence of setups.

Some previous surveys considered arc flow models on specific applications (see, e.g., Valério de Carvalho [315] and Delorme et al. [109] for bin packing and cutting stock problems, and Cattaruzza et al. [70] for vehicle routing problems with multiple trips).

Skutella [300] presented a survey on problems based on dynamic flow, which is also called "flows over time". One of the main formulations for this class of problems is the arc flow model based on time-expanded networks, introduced by Ford and Fulkerson [134], which is derived from DP graphs of pseudo-polynomial size. A generalization of such networks is the layered graph approach which considers, for instance, capacity-indexed networks for vehicle routing problems. Modeling techniques and efficient solution methods for pseudo-polynomial arc flow formulations based on layered graphs were surveyed by Gouveia et al. [154]. Other surveys, such as that by Sadykov and Vanderbeck [290], focused instead on how arc flow formulations can be used to devise effective branch-and-price algorithms. Conforti et al. [89] and Lancia and Serafini [203] studied techniques to transform large extended formulations into compact ones. In the case of network flow formulations, this occurs when path flow models are transformed into equivalent arc flow models with (possibly exponentially) fewer variables.

Contents

In this survey, we extend the previous studies to provide a base for some reasons behind the (primal) strength of pseudo-polynomial arc flow formulations. Based on the Flow Decomposition Theorem, we show how DW decomposition can derive arc flow models with a strong linear relaxation. We also discuss how the state-space relaxation method may allow one to obtain a balance between strength and complexity when constructing arc flow models. This relation between state-space relaxation and arc flow formulations can implicitly relate different arc flow models in the literature (see Section 2.4). We provide a dual insight that may explain arc flow models' practical computational efficiency over their equivalent path flow models, which is the richer description of the dual space. The relevance is that a better description of the dual space leads to less primal degeneracy, which is headwind for Linear Programming (LP) simplex (vertex) algorithms used in branch-and-bound techniques. We also discuss the main solution methods to solve large-scale arc flow models and present the main applications studied in the literature.

The remainder of this paper is organized as follows. Section 3.2 presents the theoretical foundations of network flow formulations and DP. The relationship between arc flow models with a strong relaxation and DW decomposition is presented in Section 2.3. In

Section 2.4, we present a discussion on how the state-space relaxation can be used to obtain smaller arc flow models. In Section 2.5, we show how arc flow formulations present a more detailed dual space than path flow formulations, which leads to a better convergence of the LP solution. The state-of-the-art methods to solve large-scale arc flow models and the main applications are shown in Sections 2.6 and 3.7, respectively. Finally, our concluding thoughts are presented in Section 2.8.

2.2 Network Flow Formulations and Dynamic Programming

A *network* $\mathcal{N}$ is composed of a directed graph $G = (\mathcal{V}, \mathcal{A})$, where $\mathcal{V}$ is the set of nodes (vertices) and $\mathcal{A} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of arcs, and of a cost function $c\colon \mathcal{A} \to \mathbb{R}$ that maps each arc (u, v) to a cost $c(u, v)$. In addition, a network has two special nodes in $\mathcal{V}$, called *source* (v_{source}) and *sink* (v_{sink}), such that no arcs in $\mathcal{A}$ enters v_{source} or leaves v_{sink}. Depending on the context, we may denote the set of nodes and the set of arcs of a network $\mathcal{N}$ as $\mathcal{V}(\mathcal{N})$ and $\mathcal{A}(\mathcal{N})$, respectively.

A *flow* in a network is a function $f\colon \mathcal{A} \to \mathbb{R}_+$ that satisfies the flow conservation constraints:

$$\sum_{(u,v)\in\mathcal{A}} f((u,v)) = \sum_{(v,w)\in\mathcal{A}} f((v,w)), \qquad \forall v \in \mathcal{V} \setminus \{v_{source}, v_{sink}\}, \tag{2.1}$$

i.e., the flow entering a node is equal to the flow leaving it, except for v_{source} and v_{sink}, where there is only outgoing and incoming flow, respectively. By the flow conservation constraints, the flow leaving v_{source} is equal to the flow entering v_{sink}.

Definition 1. *A network flow problem is an optimization problem which requires to determine a flow f that satisfies side constraints and optimizes $\sum_{(u,v)\in\mathcal{A}} c(u,v)f((u,v))$. In addition, any formulation that corresponds to solving a network flow problem is referred to as a network flow formulation.*

A *path* in a graph is a sequence of arcs which joins a sequence of nodes. A *cycle* is a path in which the first and the last nodes are the same. We denote by $\mathcal{P}$ the set of all paths from v_{source} to v_{sink}, and by $\mathcal{A}(p)$ the set of arcs of a path $p \in \mathcal{P}$. The cost of a path $p \in \mathcal{P}$ is given by $\tilde{c}_p = \sum_{(u,v)\in\mathcal{A}(p)} c(u,v)$. In practice, the solution of network flow problems can be given as a set of paths and cycles with a positive flow or as the flow on each arc of the network. Following Ahuja et al. [1], this gives rise to two different classes of network flow formulations: path flow formulations and arc flow formulations.

Definition 2. *Path flow formulations are network flow formulations where decision variables correspond to the non-negative flow on each path and each cycle of the network.*

Path flow formulations are also referred to in the literature as *path-based formulations*. In path flow formulations, the flow conservation is implicitly imposed on the variables. Since $|\mathcal{P}(\mathcal{N})|$ can be exponential with respect to $|\mathcal{A}(\mathcal{N})|$, path flow formulations usually have a huge number of variables. Nonetheless, such formulations have been successfully

solved in the literature by column generation based algorithms (see, e.g., Pessoa and Uchoa [269]).

Definition 3. *Arc flow formulations are network flow formulations where decision variables correspond to the non-negative flow on each arc of the network.*

In contrast to path flow formulations, arc flow formulations impose the flow conservation explicitly as linear constraints. For instance, let variable $\varphi_{(u,v)} \in \mathbb{R}_+$, for each arc $(u,v) \in \mathcal{A}$, represent the flow on arc (u,v), and let variable $z \in \mathbb{R}_+$ represent the total flow on the network. Flow conservation constraints can be formulated as:

$$\sum_{(u,v)\in\mathcal{A}} \varphi_{(u,v)} - \sum_{(v,w)\in\mathcal{A}} \varphi_{(v,w)} = \begin{cases} -z, & \text{if } v = v_{source}, \\ z, & \text{if } v = v_{sink}, \\ 0, & \text{otherwise,} \end{cases} \qquad \forall v \in \mathcal{V}. \tag{2.2}$$

The underlying matrix from (2.2) is totally unimodular (see, e.g., Nemhauser and Wolsey [250]), which is an interesting property, as LP models with a totally unimodular constraint matrix have the *integrality property*, i.e., every vertex of the corresponding polytope is integer, implying the existence of an integer optimal solution if the right-hand side of the constraints is integer.

Although flow conservation constraints (2.2) must be explicitly included, differently from path flow formulations, arc flow formulations have a polynomial number of variables with respect to network size. An important result that relates these two classes of formulations is the following:

Theorem 1 (Flow Decomposition Theorem (Ahuja et al. [1]))**.** *Every path and cycle flow has a unique representation as non-negative arc flows. Conversely, every non-negative arc flow can be represented as a path and cycle flow (though not necessarily uniquely) with the following two properties:*

(a) Every directed path with a positive flow connects the source to the sink;

(b) At most $|\mathcal{V}| + |\mathcal{A}|$ paths and cycles have nonzero flow; out of these, at most $|\mathcal{A}|$ cycles have nonzero flow.

Theorem 1 proves the equivalence between arc flow and path flow formulations: an arc flow solution can be transformed into positive flow on a set of paths and cycles, and a solution of path flow formulations can be decomposed into flow on individual arcs. Consequently, arc flow and path flow formulations based on the same network produce the same linear relaxation bounds for integer problems.

A fundamental network flow problem is the *longest path problem* (LPP): find a longest path, i.e., a unitary flow of maximum cost, from v_{source} to v_{sink}. When the network does not have cycles of negative cost, a longest path can be found in polynomial time with respect to the network size. In particular, when the network is acyclic, a longest path can be found in $O(|\mathcal{A}|)$ by a topological ordering of the nodes (see, e.g., Ahuja et al. [1]). The LPP often appears in solution methods for more complex network flow problems. Such problems have, for instance, side constraints characterized by generalized upper and lower

bounds on the flow on subsets of arcs (see, e.g., Clautiaux et al. [83]). In the remainder of this section, we formally define Dynamic Programming and its relation to acyclic network flow formulations and the LPP.

2.2.1 Dynamic Programming and Arc Flow Formulations

(Discrete) Dynamic Programming (DP) is a well-known method, proposed in the fifties (see, e.g., Bellman [23]), to solve combinatorial optimization problems that can be decomposed into a sequence of decisions, often represented by *stages*, each corresponding to a decision step. DP models are defined by a state space $\mathcal{S}$, where each state $s \in \mathcal{S}$ is characterized by a set of entities. Each state is defined, for instance, by the subset of clients already visited in routing problems (see, e.g., Desrochers et al. [114]) or by the level of usage of a given resource in, e.g., cutting, packing or scheduling problems (see, e.g., Christofides and Hadjiconstantinou [75]).

A key aspect in DP modeling is the fact that the description of a state $s \in \mathcal{S}$ should be sufficient to recognize the admissible decisions for s, which is coined as the *no-memory property of DP*. Then, when stages are considered, problems are divided into a sequence of sub-problems, so that the solution of a sub-problem depends only on the sub-problems from the previous stage. Nowadays, some authors changed the perspective of DP from the stage-dependent description to a table-filling method, as described by, e.g., Ahuja et al. [1].

A DP model can be represented by a recursive function, in which the value of a state is only based on the values of the precedent states and the cost (contribution) of the decisions leading to that state. Formally, for each state $s \in \mathcal{S}$, let $\Delta(s)$ be the set of precedent states, and $c(r, s) \in \mathbb{R}$ be the cost required to move from $r \in \Delta(s)$ to s. The state space $\mathcal{S}$ contains two special states, s_0 and s^*, representing the initial condition of the recursion and the optimal solution of the problem, respectively. No state precedes s_0 ($\Delta(s_0) = \emptyset$) and s^* does not precede any other state ($s^* \notin \Delta(s), \forall s \in \mathcal{S}$). The DP recursion we consider is given by:

$$f_{\mathrm{DP}}(s) = \begin{cases} \max_{\{r \in \Delta(s)\}} \{f_{\mathrm{DP}}(r) + c(r, s)\}, & \text{if } s \neq s_0, \\ 0, & \text{if } s = s_0, \end{cases} \qquad \forall s \in \mathcal{S}. \tag{2.3}$$

A priori, the DP recursion (2.3) denotes a maximization problem, but it can easily be considered as a minimization problem by inverting the sign of all decision costs. Based on recursion (2.3), a DP model can be defined on a directed acyclic graph (DAG), where vertices and arcs correspond to states and decisions, respectively. In this characterization, the DP solution is given by a longest path, that is, a path from s_0 to s^* with maximum total cost (profit). In some cases, the description of the recursion produces states that are not in any solution from the initial state to the goal state, and such states can be disregarded.

The previous characterization of DP is a classical definition, and one of the most used by the integer programming literature. Nevertheless, other definitions of DP were proposed. For instance, Martin et al. [231] defined DP over a directed acyclic hypergraph,

to obtain a better polyhedral characterization of a class of combinatorial optimization problems. This generalization was later used to solve network flow problems (see, e.g., Clautiaux et al. [86]).

Following our DP definition, the DAG from a DP model produces the *dynamic programming network* $\mathcal{N}_{\text{DP}}$. Each node is associated with a state, that is, $\mathcal{V}(\mathcal{N}_{\text{DP}}) \equiv \mathcal{S}$, where states s_0 and s^* correspond to nodes v_{source} and v_{sink}, respectively. The set of arcs is defined by $\mathcal{A}(\mathcal{N}_{\text{DP}}) = \{(r,s) \mid s \in \mathcal{S}, r \in \Delta(s)\}$, and the cost $c(r,s)$ of an arc $(r,s) \in \mathcal{A}$ is equivalent to the decision cost $c(r,s)$ for moving from state r to state s.

The fact that DP can be seen as a LPP in the DP network provides a generic recipe to transform a DP model into the arc flow model given by:

$$\max \sum_{(u,v)\in\mathcal{A}(\mathcal{N}_{\text{DP}})} c(u,v)\varphi_{(u,v)}, \tag{2.4}$$

$$\text{s.t.:} \sum_{(u,v)\in\mathcal{A}(\mathcal{N}_{\text{DP}})} \varphi_{(u,v)} - \sum_{(v,w)\in\mathcal{A}(\mathcal{N}_{\text{DP}})} \varphi_{(v,w)} = \begin{cases} -1, & \text{if } v = v_{source}, \\ 1, & \text{if } v = v_{sink}, \\ 0, & \text{otherwise,} \end{cases} \quad \forall v \in \mathcal{V}(\mathcal{N}_{\text{DP}}), \tag{2.5}$$

$$\varphi_{(u,v)} \in \{0,1\}, \quad \forall (u,v) \in \mathcal{A}(\mathcal{N}_{\text{DP}}). \tag{2.6}$$

Model (2.4)–(2.6) considers flow conservation constraints with a unitary flow ($z = 1$) and side constraints impose that the flow on each arc is binary. The objective function is to maximize the cost of the selected path, given by the sum of the contributions of the state decisions. As this model has only flow conservation constraints, it has the integrality property, because, as previously discussed, its underlying matrix is totally unimodular. This property is linked with the fact that the LPP on a DAG can be solved in linear time with respect to the number of arcs (see, e.g., Bellman [23]).

In strongly $\mathcal{NP}$-hard problems, DP often provides pseudo-polynomial algorithms to solve sub-problems that determine partial plans. Examples of such sub-problems are: finding a single cutting/packing pattern on cutting and packing problems (see, e.g., Valério de Carvalho [315]); obtaining a schedule in a single machine in scheduling problems (see, e.g., Kramer et al. [198]); determining a route of a single vehicle on vehicle routing problems (see, e.g., Pessoa and Uchoa [269]).

The DP network from the sub-problems (partial plans) may be used to derive arc flow models to solve the main problem (global plan). If the global plan is a strongly $\mathcal{NP}$-hard problem (which is often the case), these models do not have the integrality property, unless $\mathcal{P} = \mathcal{NP}$, but they usually provide strong relaxations. We provide examples of DP networks of two problems that are often associated with partial plans.

2.2.2 Example on the Knapsack Problem

Let us consider an example on the *knapsack problem* (KP): given a knapsack with capacity $W \in \mathbb{Z}_+$ and a set $I = \{1, 2, \ldots, n\}$ of items, each item $i \in I$ with a weight $w_i \in \mathbb{Z}_+$, a

profit $p_i \in \mathbb{Z}_+$, and a maximum number of copies $d_i \in \mathbb{Z}_+$, determine the number x_i (with $0 \leq x_i \leq d_i$) of copies of each item $i \in I$ in the solution, so that $\sum_{i \in I} w_i x_i \leq W$ and $\sum_{i \in I} p_i x_i$ is maximum (see, e.g., Martello and Toth [226]). The optimal solution value $f_{\text{KP}}(i, W')$ of a knapsack sub-problem that considers the items $\{1, 2, \ldots, i\}$ and a partial capacity W' can be recursively computed by:

$$f_{\text{KP}}(i, W') = \begin{cases} \max_{l \in \{0,1,\ldots,\min\{d_i, \lfloor W'/w_i \rfloor\}\}} \{f_{\text{KP}}(i-1, W' - w_i l) + p_i l\}, & \text{if } i > 0 \text{ and } W' > 0, \\ 0, & \text{otherwise.} \end{cases} \tag{2.7}$$

The optimal KP solution value is given by $OPT(I, W) = \max_{\{0 < W' \leq W\}} f_{\text{KP}}(|I|, W')$ and it is associated with a DP model where the state space $\mathcal{S}$ is a subset of $\{(i, W') \mid i \in I, W' \in \{0, 1, \ldots, W\}\} \cup \{s^*\}$, each partial set of items corresponds to a stage, the initial condition state is $s_0 = (0, 0)$, and the goal is represented by a dummy state s^*, so that the precedent states in $\Delta(s^*)$ are related to the recursion of $OPT(I, W)$. The set of precedent states of each state $(i, W') \in \mathcal{S} \setminus \{s^*\}$, where $i > 0$, is $\Delta((i, W')) = \{(i-1, W' - w_i l) \mid w_i l \leq W', l \in \{0, 1, \ldots, d_i\}\}$, and the decision cost for reaching (i, W') from a precedent state $(i-1, W' - w_i l) \in \Delta((i, W'))$ is $c((i-1, W' - w_i l), (i, W')) = p_i l$. The set of precedent states of s^* is $\Delta(s^*) = \{(|I|, W') \mid W' \in \{0, 1, \ldots, W\}\}$ and the cost to move from a precedent state $(|I|, W') \in \Delta(s^*)$ to s^* is $c((|I|, W'), s^*) = 0$.

Let $\mathcal{N}_{\text{KP}}$ be a DP network for the KP. The set of nodes $\mathcal{V}(\mathcal{N}_{KP}) \equiv \mathcal{S}$ corresponds to the set of states, where v_{source} and v_{sink} are associated with s_0 and s^*, respectively. The set of arcs is $\mathcal{A}(\mathcal{N}_{\text{KP}}) = \cup_{\{i \in I, l \in \{0,1,\ldots,\min\{d_i, \lfloor W'/w_i \rfloor\}\}} \mathcal{A}_{il} \cup \mathcal{A}_S$. For each $i \in I$, the set $\mathcal{A}_{il} = \{((i-1, W' - w_i l), (i, W')) \mid (i, W') \in \mathcal{S}, (i-1, W' - w_i l) \in \Delta((i, W'))\}$ contains the arcs corresponding to the decision of choosing l copies of i in the solution, and these arcs have profit $p_i l$. The set $\mathcal{A}_S = \{((n, W'), v_{sink}) \mid (n, W') \in \mathcal{S}\}$ contains arcs, called *loss arcs*, that link the nodes from the last stage to v_{sink}, and their profit is 0.

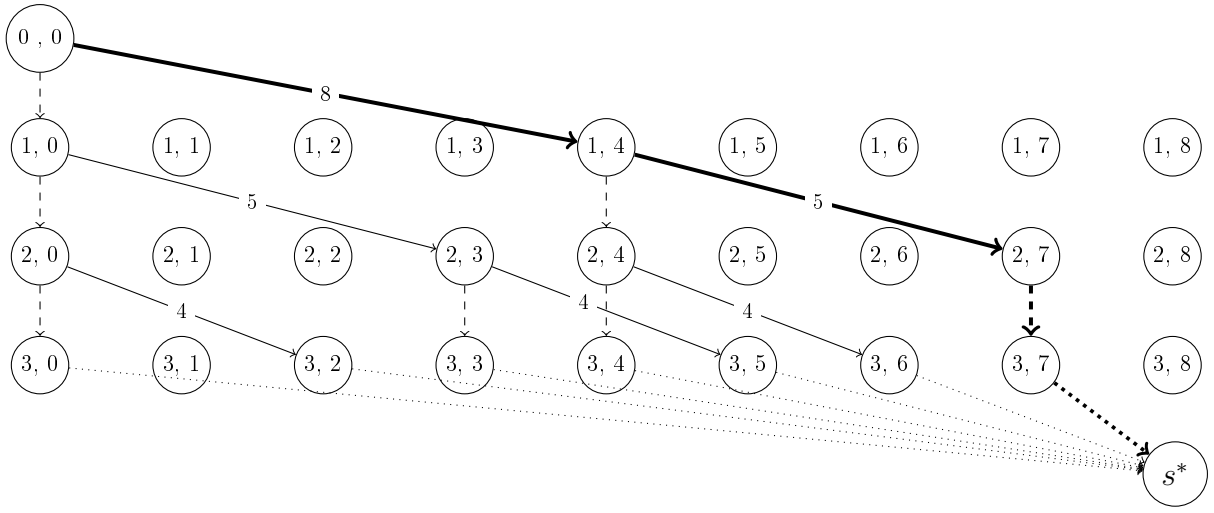

Figure 2.1: Example of network $\mathcal{N}_{\text{KP}}$ for the knapsack problem.

As an example, Figure 2.1 shows the network for a KP with capacity $W = 8$, and three items having $w_1 = 4$, $w_2 = 3$, $w_3 = 2$, $p_1 = 8$, $p_2 = 5$, $p_3 = 4$, and $d_1 = d_2 = d_3 = 1$.

There is a node s^* representing v_{sink}, a node (i, W') for each state, and $v_{source} = (0, 0)$. Arcs $\mathcal{A}_S$ are depicted as dotted links, and for every item $i \in I$, the arcs in $\mathcal{A}_{i1}$ and $\mathcal{A}_{i0}$ are depicted as full and dashed links, respectively. The arcs that do not belong to any path from $(0, 0)$ to s^* are disregarded. By definition, the only non-zero profit arcs are the ones from $\mathcal{A}_{i1}$, and their profits are shown in the figure. The bold arcs are from the longest path, i.e., the optimal solution, which contains items 1 and 2 and has total profit 13.

The network $\mathcal{N}_{\text{DP}}$ can be used in the generic recipe previously discussed, to derive an arc flow model for the KP with $O(|I|W)$ nodes and arcs. Although the resulting arc flow model has pseudo-polynomial size and has the integrality property, solving it with LP algorithms is usually not as fast as solving the problem by DP. However, the LP model can be a base for interesting results. For instance, Boyd [51] extended this model to solve the knapsack separation problem, which is a generic tool to generate cutting planes for MILP formulations.

2.2.3 Example on the Elementary Shortest Path Problem with Resource Constraints

The *elementary shortest path problem with resource constraints* (ESPPRC) is linked with determining the best route of a single vehicle in vehicle routing problems (see, e.g., Irnich and Desaulniers [182]). In the ESPPRC there is a resource capacity and a directed graph. Each vertex is associated with a resource consumption and each arc is associated with a cost. The objective is to find a path with minimum cost such that: (i) each vertex is visited at most once (elementary constraint); and (ii) the total resource consumption from the vertices in the path does not exceed the resource capacity (resource constraint).

In the context of vehicle routing, the vertices are given by $I = \{0, 1, \ldots, n\}$, where 0 represents the depot and the other n vertices represent the clients. The solution must start and finish at the depot. Generally, the ESPPRC might consider a set of resources, but in this example we are concerned only with a single resource W, which is associated with the load capacity of a vehicle. Let $e_{(i,j)} \in \mathbb{R}$ be the cost of arc $(i, j) \in I \times I$ and $w_i \in \mathbb{R}_+$ be the resource consumption of $i \in I$.

Given a set of clients $S \subseteq I \setminus \{0\}$ that satisfies the resource constraint (i.e., $\sum_{j \in S} w_j \leq W$), we denote by $f_{\text{ESPPRC}}(S, i)$ the cost of a path of minimum cost from the depot 0 to a client $i \in S$ that visits every client in S exactly once. A client $j \in S \setminus \{i\}$ that precedes i in an optimal path related to $f_{\text{ESPPRC}}(S, i)$ is one that minimizes the sum of the cost $e_{(j,i)}$ from j to i and the cost $f_{\text{ESPPRC}}(S \setminus \{i\}, j)$ of a path of minimum cost that visits every client in $S \setminus \{i\}$ exactly once and finishes in $j \in S \setminus \{i\}$. This relation can be mapped into the following recursion:

$$f_{\text{ESPPRC}}(S, i) = \begin{cases} \min_{j \in S \setminus \{i\}} \{f_{\text{ESPPRC}}(S \setminus \{i\}, j) + e_{(j,i)}\}, & \text{if } S \neq \emptyset \text{ and } i > 0, \\ 0, & \text{otherwise.} \end{cases} \tag{2.8}$$

The optimal ESPPRC solution value is given by $OPT(I, e, r) = \min\{f_{\text{ESPPRC}}(S, i) + e_{(i,0)} \mid S \subseteq I, i \in S, \sum_{j \in S} w_j \leq W\}$. The recursion can be transformed into a max-

imization problem, by inverting the sign of each $e_{(i,j)}$, resulting in a DP model. The state space $\mathcal{S}$ is a subset of $\{(S,i) \mid S \subseteq I, i \in S, \sum_{j \in S} w_j \leq W\} \cup \{s^*\}$, where the initial state is $s_0 = (\emptyset, 0)$ and the goal state s^* has precedent states related to the recursion of $OPT(I,e,r)$. For each state $(S,i) \in \mathcal{S} \setminus \{s^*\}$, the set of precedent states is $\Delta((S,i)) = \{(S \setminus \{i\}, j) \mid j \in S \setminus \{i\}\}$, and the decision cost to move from state $(S \setminus \{i\}, j) \in \Delta((S,i))$ to (S,i) is $c((S \setminus \{i\}, j), (S,i)) = e_{(j,i)}$. The set of precedent states of s^* is $\Delta(s^*) = \mathcal{S} \setminus \{s^*\}$, and the cost to move from state $(S,i) \in \Delta(s^*)$ to s^* is $c((S,i), s^*) = e_{(i,0)}$.

Let $\mathcal{N}_{\text{ESPPRC}}$ be the DP network for the ESPPRC obtained from the state space $\mathcal{S}$ presented above. The set of nodes $\mathcal{V}(\mathcal{N}_{\text{ESPPRC}}) \equiv \mathcal{S}$ corresponds to the state space, where v_{source} and v_{sink} are associated with $(\emptyset, 0)$ and $OPT(I,e,r)$, respectively. The set of arcs $\mathcal{A}(\mathcal{N}_{\text{ESPPRC}}) = \cup_{i \in I} \mathcal{A}_i$ contains arcs $\mathcal{A}_i = \{((S \setminus \{i\}, j), (S,i)) \in \mathcal{V} \times \mathcal{V}\}$ related to the decision of visiting client $i \in I \setminus \{0\}$, and arcs $\mathcal{A}_0 = \{((S,j), v_{sink}) \in \mathcal{V} \times \mathcal{V}\}$ related to the decision of returning to the depot.

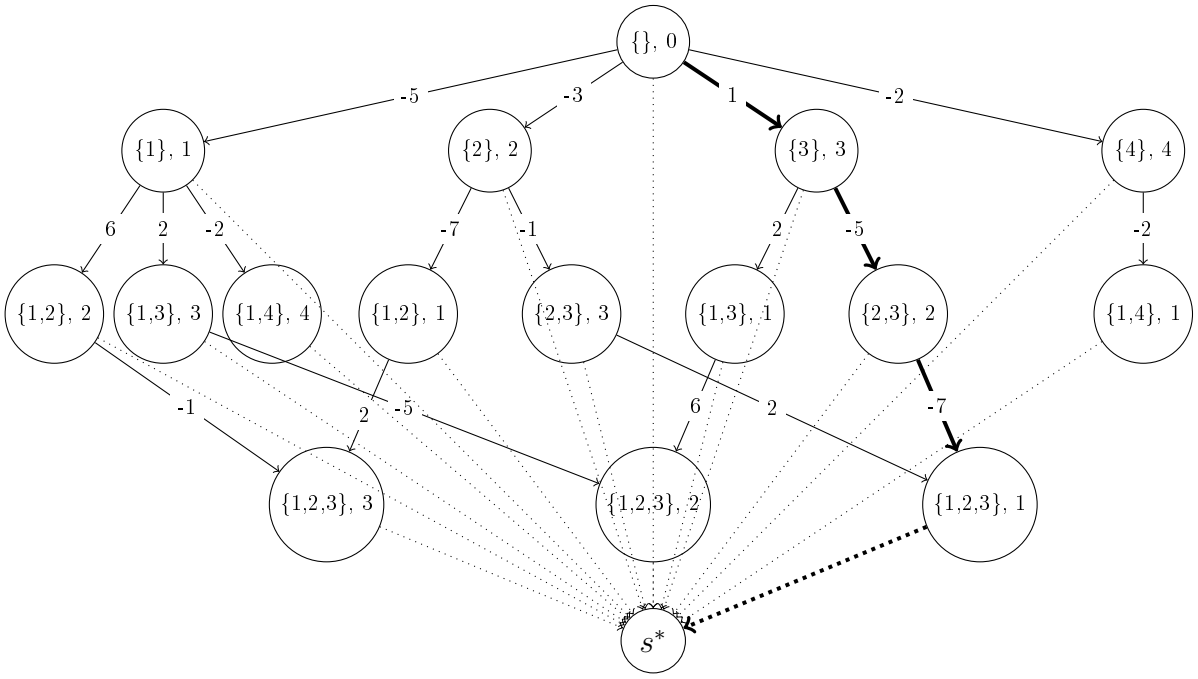

Figure 2.2: Example of network $\mathcal{N}_{\text{ESPPRC}}$ for the elementary shortest path problem with resource constraints.

Figure 2.2 illustrates the network for an example with $W = 6$ and 4 clients, with $w_1 = 1$, $w_2 = 2$, $w_3 = 2$, and $w_4 = 5$. In this example, the costs are the following: $c_{(0,1)} = -5$, $c_{(0,2)} = -3$, $c_{(0,3)} = 1$, $c_{(0,4)} = -2$, $c_{(1,0)} = 0$, $c_{(1,2)} = 6$, $c_{(1,3)} = 2$, $c_{(1,4)} = -3$, $c_{(2,0)} = 0$, $c_{(2,1)} = -7$, $c_{(2,3)} = -1$, $c_{(2,4)} = 5$, $c_{(3,0)} = 0$, $c_{(3,1)} = 2$, $c_{(3,2)} = -5$, $c_{(3,4)} = 5$, $c_{(4,0)} = 0$, $c_{(4,1)} = -2$, $c_{(4,2)} = 5$, $c_{(4,3)} = 5$. On each arc, we present its associated cost, except for the arcs that enter s^*, which for the sake of conciseness we consider their cost to be 0 and present such arcs as dotted lines. The arcs of the optimal solution are highlighted in bold, and they correspond to visit the sequence of clients $3, 2$, and 1, and returning to the depot. The optimal solution has value -11.

The network $\mathcal{N}_{\text{ESPPRC}}$ can be the base of an arc flow model for the ESPPRC,

following the generic recipe previously presented. However, the number of subsets $S \subseteq I$ such that $\sum_{j \in S} w_j \leq W$ is exponential, implying that the state space presented for the ESPPRC is also exponential. Consequently, the corresponding arc flow model has an exponential number of variables and constraints, and it is not practical to solve it directly. Practical techniques that can be used to overcome this inconvenience are described in Section 2.4. In particular, Section 2.4.2 presents a network of pseudo-polynomial size for a version of the ESPPRC where the elementary constraints are partially relaxed.

2.3 Dantzig-Wolfe Decomposition and Network Flow Formulations

Several challenging problems admit compact MILP models that, although having a reasonable (polynomial) size, are usually associated with weak linear relaxations. In such cases, reformulation methods can obtain models with stronger linear relaxation. One of such methods is the well-known *Dantzig-Wolfe* (DW) *decomposition* (see Dantzig and Wolfe [100]), which has been successfully used in many applications. The resulting models may be stronger, but they usually have an exponential number of variables, and complex methods are required to solve them in practice. In the following, we show how models obtained from DW decomposition can be the base to derive equivalent arc flow models with smaller (and possibly practical) size.

First, we show how the DW decomposition can be applied to Linear Programming (LP) models. Let us express the feasible solution region of the LP model as:

$$X_{LP} = \{x \in \mathbb{R}^n_+ \mid Ax \geq b, x \in X\}, \tag{2.9}$$

where $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, and X is a bounded convex polytope of a set of constraints. According to the Minkowski's Theorem, any point in X can be represented as a convex combination of the vertices (extreme points) of X (for the general case of an unbounded polytope, it is also necessary to consider a non-negative linear combination of the extreme rays of the polytope, see, e.g., Nemhauser and Wolsey [250]). Then, by defining $\mathcal{Q}(X)$ as the set of vertices of X, it follows:

$$X = \{x \in \mathbb{R}^n_+ \mid x = \sum_{q \in \mathcal{Q}(X)} q\lambda_q, \sum_{q \in \mathcal{Q}(X)} \lambda_q = 1, \lambda_q \geq 0, \forall q \in \mathcal{Q}(X)\}. \tag{2.10}$$

By substituting x as in (2.10) in (2.9), the feasible region in the space of the variables of the reformulated model becomes:

$$X_{LPDW} = \{\lambda \in \mathbb{R}^{|\mathcal{Q}(X)|}_+ \mid \sum_{q \in \mathcal{Q}(X)} (Aq)\lambda_q \geq b, \sum_{q \in \mathcal{Q}(X)} \lambda_q = 1\}. \tag{2.11}$$

The reformulated problem, called *DW model*, has a variable associated with each vertex of X, and a new set of constraints, usually referred to as convexity constraints (see, e.g., Lübbecke and Desrosiers [218]). The DW model is just another way of expressing the same solution space. Therefore, the decomposition applied to LP models preserves the

optimal solution value.

However, in the context of MILP, to obtain linear models stronger than the original linear relaxation, the integrality constraints must be implicitly considered in the reformulated variables. A possibility is to impose the integrality constraints just in the set X, reformulating over the vertices of $Conv\{x \in X \text{ and integer}\}$, the convex hull of the integer points in X (see, e.g., Nemhauser and Wolsey [250]), to optimize over the set:

$$X_{DW} = \{x \in \mathbb{R}^n_+ | Ax \geq b, x \in Conv\{x \in X \text{ and integer}\}\}. \tag{2.12}$$

Note indeed that $X_{DW} \subseteq X_{LP}$, and the relation can be strict when the set X does not have the integrality property. When that happens, the reformulated model is stronger. The strength of a model can have a strong impact on the search for the optimal integer solution. Typically the length of the search is smaller with stronger models, leading to smaller computational times. For further details, the reader is referred to Nemhauser and Wolsey [250], which discusses at length the importance of deriving stronger models in the context of MILP.

A typical issue from the DW decomposition is that the resulting model may have an exponential number of variables. This issue is usually addressed by relying on the column generation method, a technique to solve LP models with a large number of variables (see, e.g., Lübbecke and Desrosiers [218]). The method solves a restricted version of the LP model with only a subset of variables by iteratively solving a *pricing problem*. The pricing generates non-basic variables (columns) that are candidates to improve the current restricted model. If no such variable exists, then the current basis of the restricted model is optimal for the original model, and the method halts. Column generation algorithms are complex, but it pays off to solve models resulting from a DW decomposition, because they may be stronger when the set X does not have the integrality property and integrality is enforced in the pricing problem.

When the pricing problem of a DW model can be solved by DP, each column of the model can be associated with a path in the DP network, and the model can be seen as a path flow formulation. In such cases, the coefficients of a column are given as contributions from the arcs of the corresponding path. In addition, due to the no-memory property of the DP, for each column, the contribution $\ell^k_{(u,v)} \in \mathbb{R}$ of an arc (u, v) to a row coefficient $k \in \mathcal{C}$, where $\mathcal{C}$ is the set of constraints, is independent of other arcs. Then, given a DP network $\mathcal{N}$, and $d_k \in \mathbb{R}$, for every constraint $k \in \mathcal{C}$, we obtain a general path flow formulation:

$$\min \sum_{p \in \mathcal{P}(\mathcal{N})} \tilde{c}_p \lambda_p, \tag{2.13}$$

$$\text{s.t.:} \sum_{p \in \mathcal{P}(\mathcal{N})} \sum_{(u,v) \in \mathcal{A}(p)} \ell^k_{(u,v)} \lambda_p \geq d_k, \qquad \forall k \in \mathcal{C}, \tag{2.14}$$

$$\lambda_p \in \mathbb{Z}_+, \qquad \forall p \in \mathcal{P}(\mathcal{N}). \tag{2.15}$$

The objective function (2.13) minimizes the total cost from the paths in the solution, and (2.14) is a set of general linear constraints. As discussed previously, due to the flow

decomposition theorem, the path flow formulation (2.13)–(2.15) can be reformulated as the following arc flow formulation:

$$\min \sum_{(u,v)\in\mathcal{A}(\mathcal{N})} c(u,v)\varphi_{(u,v)}, \tag{2.16}$$

$$\text{s.t.:} \sum_{(u,v)\in\mathcal{A}(\mathcal{N})} \varphi_{(u,v)} - \sum_{(v,w)\in\mathcal{A}(\mathcal{N})} \varphi_{(v,w)} = \begin{cases} -z, & \text{if } v = v_{source}, \\ z, & \text{if } v = v_{sink}, \\ 0, & \text{otherwise}, \end{cases} \quad \forall v \in \mathcal{V}(\mathcal{N}), \tag{2.17}$$

$$\sum_{(u,v)\in\mathcal{A}(\mathcal{N})} \ell^k_{(u,v)}\varphi_{(u,v)} \geq d_k, \quad \forall k \in \mathcal{C}, \tag{2.18}$$

$$z \geq 0, \tag{2.19}$$

$$\varphi_{(u,v)} \in \mathbb{Z}_+, \quad \forall (u,v) \in \mathcal{A}(\mathcal{N}). \tag{2.20}$$

The objective function (2.16) minimizes the total cost from the arcs in the solution, (2.17) are the flow conservation constraints, and (2.18) are general linear constraints, adapted from (2.14).

DW decomposition can generate path flow formulations that are usually associated with strong linear relaxations, and equivalent arc flow formulations can be derived from the DP network of the underlying pricing problem. As pointed before, to obtain stronger relaxations, one should reformulate over the convex hull of the integer points in a polytope that does not have the integrality property. Although a stronger linear relaxation is obtained, this kind of reformulation leads to pricing problems that are $\mathcal{NP}$-hard.

On the other hand, for arc flow models, the price to pay to obtain a stronger relaxation (i.e., to obtain a set equivalent to X_{DW}) is to use a set of constraints that defines a DP network with a pseudo-polynomial or an exponential number of arcs.

Usually, arc flow models of practical size can be derived when the resulting pricing problem has pseudo-polynomial complexity. However, when the pricing problem is strongly $\mathcal{NP}$-hard, the resulting arc flow model is generally too large to be solved in practice. When the pricing problem from path flow formulations is too difficult in practice, one may rely on relaxation methods. We show in Section 2.4 one of such relaxation methods, which can be used to derive smaller networks and obtain arc flow formulations that are still strong and have practical size. In the next sections, we present two examples of arc flow formulations derived from DW decomposition, with pseudo-polynomial and exponential size, respectively.

2.3.1 Example on the Cutting Stock Problem

We present an example of an arc flow model derived from a DW decomposition for the *Cutting Stock Problem* (CSP). In the CSP, a roll of width W and a set $I = \{1, 2, \ldots, n\}$ of items are given, such that each item $i \in I$ has a positive demand d_i and a width w_i. The objective is to cut the demand of all items from the minimum number of rolls.

Given an upper bound K on the minimum number of rolls, following Martello and

Toth [226], the CSP can be formulated as:

$$\min \sum_{j=1}^{K} y_j, \tag{2.21}$$

$$\text{s.t.:} \sum_{i=1}^{n} w_i x_{ij} \leq W y_j, \qquad j = 1, 2, \ldots, K, \tag{2.22}$$

$$\sum_{j=1}^{K} x_{ij} \geq d_i, \qquad \forall i \in I, \tag{2.23}$$

$$y_j \in \{0, 1\}, \qquad j = 1, 2, \ldots, K, \tag{2.24}$$

$$x_{ij} \in \mathbb{Z}_+, \qquad \forall i \in I, j = 1, 2, ..., K. \tag{2.25}$$

Each roll $j = 1, 2, \ldots, K$ is associated with a binary variable y_j that is equal to 1 if and only if roll j is used in the solution. The integer variable x_{ij} represent the number of copies of item $i \in I$ that is cut from roll j. The objective function (2.21) minimizes the number of rolls cut in the solution. Constraints (2.22) ensure that the size of each roll is satisfied and that no item is cut from an unused roll, whereas constraints (2.23) ensure that the demand of each item is satisfied.

According to Martello and Toth [226], the optimal solution value of the linear relaxation of (2.21)–(2.25) is equivalent to the continuous lower bound that is obtained from the minimum length required to cut all of the demand from a stock with unlimited width, i.e., $\sum_{i \in I} w_i d_i / W$. This bound is known to be weak in practice, and its worst-case performance ratio asymptotically tends to 1/2. To obtain a stronger bound, a DW decomposition can be applied to the model above (see, e.g., Vance [320]).

Constraints (2.22) correspond to K identical fractional knapsack polytopes. When integrality constraints (2.24) and (2.25) are taken into account, the K identical polytopes correspond to K identical sets P_{CSP} of integer knapsack solutions. A DW decomposition that includes constraints (2.22), (2.24), and (2.25) in the sub-problem results in the following set-covering model:

$$\min \sum_{p \in P_{CSP}} \lambda_p, \tag{2.26}$$

$$\text{s.t.:} \sum_{p \in P_{CSP}} a_{ip} \lambda_p \geq d_i, \qquad \forall i \in I, \tag{2.27}$$

$$\lambda_p \in \mathbb{Z}_+, \qquad \forall p \in P_{CSP}. \tag{2.28}$$

In the CSP, the integer knapsack solutions from P_{CSP} are called *cutting patterns* (or just patterns, for short, in the following), and each of them represents the cutting of a single piece of stock. The objective function (2.26) minimizes the number of cutting patterns. Constraints (2.27) ensure that the demand of each item is satisfied, where a_{ip} is the number of copies of item i in pattern p.

In cutting and packing problems, a *proper pattern* is a pattern that respects the maximum number of copies of the items, whereas a *non-proper pattern* does not. The set-covering model by Gilmore and Gomory [149, 150] for the CSP is similar to (2.26)–(2.28),

but it includes non-proper patterns that are obtained from the polytope of an *unbounded knapsack problem* (UKP), a variant of the KP where each item has an unlimited number of copies. The linear relaxation of (2.26)–(2.28), often referred to as *proper relaxation* (see, e.g., Kartak et al. [190]), is stronger than the linear relaxation of the model by Gilmore and Gomory [149, 150], but the optimal solution values of the corresponding MILP models are equal.

The linear relaxation of (2.26)–(2.28) is strong and often the number of rolls in the optimal solution is the rounded up optimal solution value from this relaxation. In fact, there is a conjecture related to the strength of this relaxation (see, e.g., Caprara et al. [64] and Kartak et al. [190]):

Conjecture 1 (Modified Integer Round-Up Property (MIRUP))**.** *The difference between the optimal solution value of the CSP and the rounded-up solution value of the linear relaxation of* (2.26)–(2.28) *is at most one.*

The pricing problem of the set-covering model is a KP (where each item has d_i copies), which, as shown in Section 2.2.2, can be solved by DP, implying on the existence of a DP network $\mathcal{N}_{\text{KP}}$. As every column from the set-covering model can be represented as a path in $\mathcal{N}_{\text{KP}}$, this network can be the base of an arc flow model for the CSP:

$$\min z, \tag{2.29}$$

$$\text{s.t.:} \sum_{(u,v)\in\mathcal{A}(\mathcal{N}_{\text{KP}})} \varphi_{(u,v)} - \sum_{(v,w)\in\mathcal{A}(\mathcal{N}_{\text{KP}})} \varphi_{(v,w)} = \begin{cases} -z, & \text{if } v = v_{source}, \\ z, & \text{if } v = v_{sink}, \\ 0, & \text{otherwise}, \end{cases} \quad \forall v \in \mathcal{V}(\mathcal{N}_{\text{KP}}), \tag{2.30}$$

$$\sum_{(u,v)\in\mathcal{A}_i(\mathcal{N}_{\text{KP}})} \ell^i_{(u,v)}\varphi_{(u,v)} \geq d_i, \quad \forall i \in I, \tag{2.31}$$

$$z \in \mathbb{Z}_+, \tag{2.32}$$

$$\varphi_{(u,v)} \in \mathbb{Z}_+, \quad \forall (u,v) \in \mathcal{A}(\mathcal{N}_{\text{KP}}). \tag{2.33}$$

The objective function (2.29) minimizes the total flow. Constraints (2.30) impose the flow conservation and constraints (2.31) are related to the demand of each item. Following the definitions in Section 2.2.2, we further define $\mathcal{A}_i(\mathcal{N}_{\text{KP}}) = \cup_{l\in\{0,1,\ldots,d_i\}}\mathcal{A}_{il}(\mathcal{N}_{\text{KP}})$, for each $i \in I$, and the contribution $\ell^i_{(u,v)}$ of arc $(u,v) \in \mathcal{A}_i(\mathcal{N}_{\text{DP}})$ is the number of copies of i associated with (u,v). The arc flow model (2.29)–(2.33), corresponds to the arc flow model proposed by Cambazard and O'Sullivan [62], under the name *DP-flow*, for the *bin packing problem* (BPP), a particular case of the CSP where $d_i = 1$ for every $i \in I$. The DP-flow is equivalent to the set-covering model (2.26)–(2.28) (see, e.g., Delorme and Iori [107]), and it follows the proper relaxation.

In this example, we started from a model that is associated with a weak linear relaxation. A DW decomposition resulted in a model with a stronger relaxation, but with the drawback of having a potentially exponential number of variables. Then, we showed

how the DP network related to the pricing problem of the exponential model derives an equivalent pseudo-polynomial network flow model. Practical successful applications of this idea are shown in Section 3.7.

2.3.2 Example on the Capacitated Vehicle Routing Problem

In the *capacitated vehicle routing problem* (CVRP), we are given a set K of identical vehicles and a set $I = \{0, 1, \ldots, n\}$ of vertices where vertex 0 corresponds to a depot, and vertices $1, \ldots, n$ correspond to n clients. Each vehicle has a load capacity W, each client $i \in I \setminus \{0\}$ has a non-negative demand w_i, and each pair of vertices $i, j \in I$ is associated with a cost $c(i, j)$. The CVRP aims at determining a routing plan with the minimum total cost, such that: (i) exactly $|K|$ routes are considered; (ii) each route starts and ends at the depot; (iii) each client is visited exactly once; and (iv) the total demand from the clients of a route does not exceed the load capacity W.

In this example, we apply a DW decomposition on a compact arc flow model for the CVRP, obtaining a set partitioning model that derives an equivalent arc flow model of exponential size.

Let $I^* = \{I \setminus \{0\}\} \cup \{0^+, 0^-\}$ be the set of clients with two additional vertices 0^+ and 0^-, which are copies of the depot, each corresponding to the source and the sink of a route, respectively. Two additional sets of arcs, $\{(0^+, j) \mid j \in \{I \setminus \{0\}\}\}$ and $\{(i, 0^-) \mid i \in \{I \setminus \{0\}\}\}$, are considered. The following compact arc flow model, also known as three-index (vehicle-flow) formulation (see, e.g., Irnich et al. [184]), solves the CVRP:

$$\min \sum_{k \in K} \sum_{i \in I^*} \sum_{j \in I^* \setminus \{i\}} c(i,j) \phi^k_{(i,j)}, \tag{2.34}$$

$$\text{s.t.:} \sum_{j \in I^* \setminus \{i\}} \phi^k_{(i,j)} - \sum_{j \in I^* \setminus \{i\}} \phi^k_{(j,i)} = \begin{cases} -1, & \text{if } i = 0^+, \\ 1, & \text{if } i = 0^-, \\ 0, & \text{otherwise,} \end{cases} \qquad \forall k \in K, i \in I^*, \tag{2.35}$$

$$\sum_{k \in K} \sum_{j \in I^* \setminus \{i\}} \phi^k_{(i,j)} = 1, \qquad \forall i \in I \setminus \{0\}, \tag{2.36}$$

$$\omega_{ik} - \omega_{jk} + W \phi^k_{(i,j)} \leq W - w_j, \qquad \forall i, j \in I^*, i \neq j, k \in K, \tag{2.37}$$

$$\omega_{ik} \leq W, \qquad \forall i \in I^*, k \in K, \tag{2.38}$$

$$\phi^k_{(i,j)} \in \{0, 1\}, \qquad \forall i, j \in I^*, k \in K, \tag{2.39}$$

$$\omega_{ik} \geq 0, \qquad \forall i \in I^*, k \in K. \tag{2.40}$$

Model (2.34)–(2.40) considers a copy of the original graph for each vehicle $k \in K$. The binary variable $\phi^k_{(i,j)}$ is equal to 1 if and only if vehicle k moves from client i to client j, and the variable ω_{ik} indicates the accumulated demand already distributed by the vehicle k when arriving at client i. The objective function (2.34) minimizes the total cost of the routes. Constraints (2.35) guarantee the conservation of a unitary flow (representing a single route) for each of the $|K|$ vehicles. Constraints (2.36) guarantee that each client is visited exactly once. Constraints (2.37) and (2.38) model the propagation

of the accumulated demand of the load capacity of each vehicle, and they are also used to ensure that the route of each vehicle is a single connected component (path).

Although model (2.34)–(2.40) has a polynomial number of variables, it has a large number of symmetries that makes it inefficient in practice (see, e.g., Irnich et al. [184]). Each possible route can be attributed to any vehicle, so each solution (routing plan) has $|K|!$ equivalent permutations. Next, we present a DW decomposition that eliminates this symmetry.

Flow conservation constraints (2.35) correspond to $|K|$ identical polytopes, each containing all paths from 0^+ to 0^-, whose vertices are always integer. By including constraints (2.37) and (2.38), each of the $|K|$ resulting polytopes contains all paths from 0^+ to 0^- satisfying the load capacity, but the integrality property is lost. Let us consider the DW decomposition over the set P_{CVRP} of vertices of the convex hull of constraints (2.35), (2.37), (2.38), and (2.40), that represent the set of integer paths that satisfy the load capacity. The resulting DW model is:

$$\min \sum_{p \in P_{CVRP}} \tilde{c}_p \lambda_p, \tag{2.41}$$

$$\text{s.t.:} \sum_{p \in P_{CVRP}} a_{pi} \lambda_p = 1, \qquad \forall i \in I \setminus \{0\}, \tag{2.42}$$

$$\sum_{p \in P_{CVRP}} \lambda_p = |K|, \tag{2.43}$$

$$\lambda_p \in \{0, 1\}, \qquad \forall p \in P_{CVRP}. \tag{2.44}$$

The set partitioning model (2.41)–(2.44) provides a strong lower bound and it is among the most successful formulations to solve the CVRP in practice (see, e.g., Pessoa and Uchoa [269] and Pecin et al. [260]). This model associates with each path $p \in P_{CVRP}$ a variable λ_p. The binary coefficient a_{ip} is equal to 1 if and only if client $i \in I \setminus \{0\}$ is visited in p. The objective function (2.41) minimizes the total cost, where the cost $\tilde{c}_p$ of a path p is the sum of the costs of the arcs in $\mathcal{A}(p)$. Constraints (2.42) guarantee that each client is visited by exactly one route. As each variable represents a unique path, the symmetry from the model (2.34)–(2.40) is eliminated.

The pricing problem associated with model (2.41)–(2.44) is equivalent to the ESPPRC. In Section 2.2.3, we presented the exponential DP network $\mathcal{N}_{\text{ESPPRC}}$ for the ESPPRC. Based on this network, and recalling that $\mathcal{A}_i(\mathcal{N}_{\text{ESPPRC}})$ is the set of arcs that visit $i \in I$,

we can reformulate the set partitioning model into the following arc flow model:

$$\min \sum_{(u,v)\in\mathcal{A}(\mathcal{N}_{\text{ESPPRC}})} c(u,v)\varphi_{(u,v)}, \tag{2.45}$$

$$\text{s.t.:} \sum_{(u,v)\in\mathcal{A}(\mathcal{N}_{\text{ESPPRC}})} \varphi_{(u,v)} - \sum_{(v,w)\in\mathcal{A}(\mathcal{N}_{\text{ESPPRC}})} \varphi_{(v,w)} = \begin{cases} -|K|, & \text{if } v = v_{source}, \\ |K|, & \text{if } v = v_{sink}, \\ 0, & \text{otherwise,} \end{cases}$$
$$\forall v \in \mathcal{V}(\mathcal{N}_{\text{ESPPRC}}), \tag{2.46}$$

$$\sum_{(u,v)\in\mathcal{A}_i(\mathcal{N}_{\text{ESPPRC}})} \varphi_{(u,v)} = 1, \qquad \forall i \in I \setminus \{0\}, \tag{2.47}$$

$$\varphi_{(u,v)} \in \{0,1\}, \qquad \forall (u,v) \in \mathcal{A}(\mathcal{N}_{\text{ESPPRC}}). \tag{2.48}$$

The objective function (2.45) minimizes the total cost of the routing plan. Constraints (2.46) impose the flow conservation, and constraints (2.47) imply that each client is visited exactly once.

As network $\mathcal{N}_{\text{ESPPRC}}$ is possibly exponential, the arc flow model (2.45)–(2.48) may be too large to be used in practice. Nonetheless, this model is presented so as to provide an example in Section 2.4.2 on how a relaxation for the pricing problem of (2.41)–(2.44) derives a pseudo-polynomial arc flow model for the CVRP.

2.4 State-Space Relaxation on Arc Flow Formulations

Dynamic programming is often an efficient tool to solve hard problems. However, when the number of states from a DP model is very large, it is not practical to solve it by enumerating all states, and more sophisticated methods are needed. From that observation, Christofides et al. [76] proposed a general relaxation procedure for DP, called *state-space relaxation*, which aggregates subsets of states in order to obtain a smaller state space which provides a bound for the original problem. Such relaxation can be embedded in exact solution methods to find the optimal solution for the original state space by (partially) disaggregating the relaxed state space during the search for feasible solutions. Since a state space may allow many different relaxations, it is desirable to determine relaxations that provide a good balance between number of states and strength of the relaxation.

Formally, a state-space relaxation consists of a mapping function $g\colon \mathcal{S} \to \mathcal{G}$ between two state spaces, where $|\mathcal{G}| < |\mathcal{S}|$. For every $s \in \mathcal{S}$ and $r \in \Delta(s)$, function g must guarantee that $g(r) \in \Delta(g(s))$, and the decision cost of going from $g(r)$ to $g(s)$ is defined as $\bar{c}_{(g(r),g(s))} = \max_{\{p,q\in\mathcal{S}\}}\{c(p,q) \mid g(p) = g(r), g(q) = g(s)\}$. The DP recursion of the

resulting state-space relaxation is given by:

$$f_{\text{SSR}}(g(s)) = \begin{cases} \max_{\{p \in \Delta(g(s))\}}(f_{\text{SSR}}(p) + \bar{c}_{(p,g(s))}), & \text{if } s \neq s_0, \\ 0, & \text{if } s = s_0, \end{cases} \qquad \forall s \in \mathcal{S}. \tag{2.49}$$

As a relaxation, recursion (2.49) guarantees that $f_{\text{SSR}}(g(s)) \geq f_{\text{DP}}(s)$, i.e., it produces an upper bound for the original recursion f_{DP}. In the context of arc flow formulations induced by DP, the size of the LP formulation is strictly related to the size of the state space. In this case, smaller arc flow formulations can be obtained from state-space relaxations. Solutions obtained with the state-space relaxation may be unfeasible for the original problem. However, there are many cases in which arc flow formulations based on networks from state-space relaxations are guaranteed to produce optimal integer solutions that are feasible for the original problem. The main drawback of state-space relaxations is that they lead to arc flow formulations with weaker linear relaxation. This weakness occurs as the relaxation can profit from paths that are not feasible in the original network, generating, for instance, non-proper patterns in cutting and packing problems or non-elementary routes in vehicle routing problems. However, in many cases, the reduction on the size of the model pays off the loss in the linear relaxation strength.

Strong pseudo-polynomial arc flow formulations can be obtained from state-space relaxations of both pseudo-polynomial and exponential state spaces. Motivated by vehicle routing problems, Gouveia et al. [154] studied modeling and solution methods of a class of pseudo-polynomial arc flow formulations obtained from state-space relaxation of exponential state-spaces, named by the authors as *layered graph formulations*.

Next, we present two examples of pseudo-polynomial arc flow formulations obtained from state-space relaxations over a pseudo-polynomial and an exponential state space.

2.4.1 Example on the State Space for the Cutting Stock Problem

In Section 2.2.2, we presented recursion (2.7) to solve the KP. This recursion produces network $\mathcal{N}_{\text{KP}}$, which has $O(|I|W)$ nodes and $O(|I|W)$ arcs. Then, in Section 2.3.1, network $\mathcal{N}_{\text{KP}}$ was the base of arc flow model (2.29)–(2.33) for the CSP, i.e., the DP-Flow model of Cambazard and O'Sullivan [62].

Let $\mathcal{N}_{\text{KP-SSR}}$ be the network of a state-space relaxation of $\mathcal{N}_{\text{KP}}$ based on the mapping function $g((i, W')) = (W')$, for every state $(i, W') \in \mathcal{V}(\mathcal{N}_{\text{KP}})$. This mapping function disregards the dimension related to the partial set of items and merges states representing the same partial capacities, into a single state. Network $\mathcal{N}_{\text{KP-SSR}}$ is smaller than $\mathcal{N}_{\text{KP}}$, with $O(W)$ nodes, but it may contain paths that violate the maximum number of copies of each item. Then, an arc flow model for the KP, based on $\mathcal{N}_{\text{KP-SSR}}$, needs additional side constraints to impose limits on the number of copies of items, while in $\mathcal{N}_{\text{KP}}$, such constraints are implicitly imposed by the configuration of the network.

In the CSP, as each demand constraint (2.31) imposes only a minimum and not a maximum number of items, there is no problem in considering a network that has paths representing cutting patterns with more copies of a single item than the required one. Hence, the arc flow model obtained by substituting $\mathcal{N}_{\text{KP}}$ by $\mathcal{N}_{\text{KP-SSR}}$ on model (2.29)–

(2.33) still solves the CSP, because unnecessary items copies, if any, can be removed from the solution without affecting its cost. The resulting model is equivalent to the arc flow model for the CSP proposed by Valério de Carvalho [314].

The model with $\mathcal{N}_{\text{KP-SSR}}$ is smaller than the original model, with $O(|I| + W)$ constraints instead of $O(|I|W)$, but its linear relaxation is weaker. Nonetheless, $\mathcal{N}_{\text{KP-SSR}}$ was obtained by Valério de Carvalho [314] after reducing the DP network of the UKP, implying the graph of $\mathcal{N}_{\text{KP-SSR}}$ is a subgraph of the underlying graph from a DP for the UKP. This implies that the linear relaxation of the model with $\mathcal{N}_{\text{KP-SSR}}$ is at least as strong as the relaxation of the model by Gilmore and Gomory [149, 150], whose subproblem is the UKP. Thus, the resulting linear relaxation still follows the MIRUP conjecture, implying that it is still strong. In practice, the arc flow model by Valério de Carvalho [314], which is associated with $\mathcal{N}_{\text{KP-SSR}}$, is preferable than the model by Cambazard and O'Sullivan [62], which is associated with $\mathcal{N}_{\text{KP}}$, as its linear relaxation is still strong and it is substantially smaller.

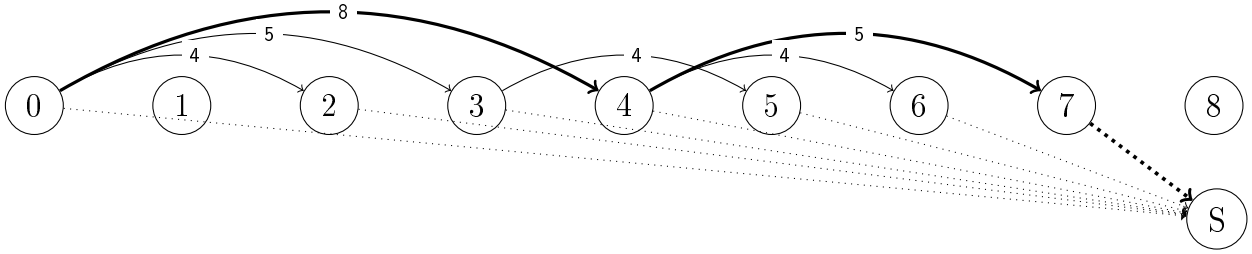

Figure 2.3: Example of network $\mathcal{N}_{\text{KP-SSR}}$ obtained from a state-space relaxation of $\mathcal{N}_{\text{KP}}$.

To exemplify, Figure 2.3 shows the network $\mathcal{N}_{\text{KP-SSR}}$ obtained from the example of Figure 2.1. We conclude that state-space relaxation can be considered to reduce the size of pseudo-polynomial arc flow formulations and obtain models with a linear relaxation that is still strong. This modeling technique to derive smaller, but yet efficient, pseudo-polynomial models from the relaxation of pseudo-polynomial state spaces has been mainly used, even without mentioning it, to model one- and two-dimensional cutting and packing problems and scheduling problems (see Section 3.7).

2.4.2 Example on the State Space for the Capacitated Vehicle Routing Problem

In Section 2.3.2, we presented a DW decomposition for the CVRP that resulted in the path flow model (2.41)–(2.44), which is known to have a strong linear relaxation. Then, we presented the corresponding arc flow model (2.45)–(2.48), which is based on the network $\mathcal{N}_{\text{ESPPRC}}$ for the ESPPRC and has exponential size. In the following, we present a state-space relaxation for $\mathcal{N}_{\text{ESPPRC}}$ that leads to a pseudo-polynomial arc flow model for the CVRP.

A pseudo-polynomial state space $\mathcal{N}_{\text{ESPPRC-SSR}}$ can be obtained from a state-space relaxation of $\mathcal{N}_{\text{ESPPRC}}$ based on the mapping function $g((S, i)) = (\sum_{j \in S} r_j, i)$, for every state $(S, i) \in \mathcal{N}_{\text{ESPPRC}}$. This mapping function (originally proposed by Christofides et

al. [76]) merges states with the same total load from the visited clients into a single state. The resulting network $\mathcal{N}_{\text{ESPPRC-SSR}}$, which has $O(|I|W)$ nodes and $O(|I|^2W)$ arcs, preserves the resource constraints of the ESPPRC, but it may consider non-elementary paths (clients may be visited more than once).

In the exponential arc flow model (2.45)–(2.48) for the CVRP, the side constraints guarantee that clients are visited exactly once. Hence, by changing network $\mathcal{N}_{\text{ESPPRC}}$ by $\mathcal{N}_{\text{ESPPRC-SSR}}$ in model (2.45)–(2.48), the resulting model still solves the CVRP and has pseudo-polynomial size. This resulting model is often referred to as capacity-indexed formulation (see, e.g., Pessoa and Uchoa [269]), and it has been studied as a layered graph formulation by Gouveia et al. [154].

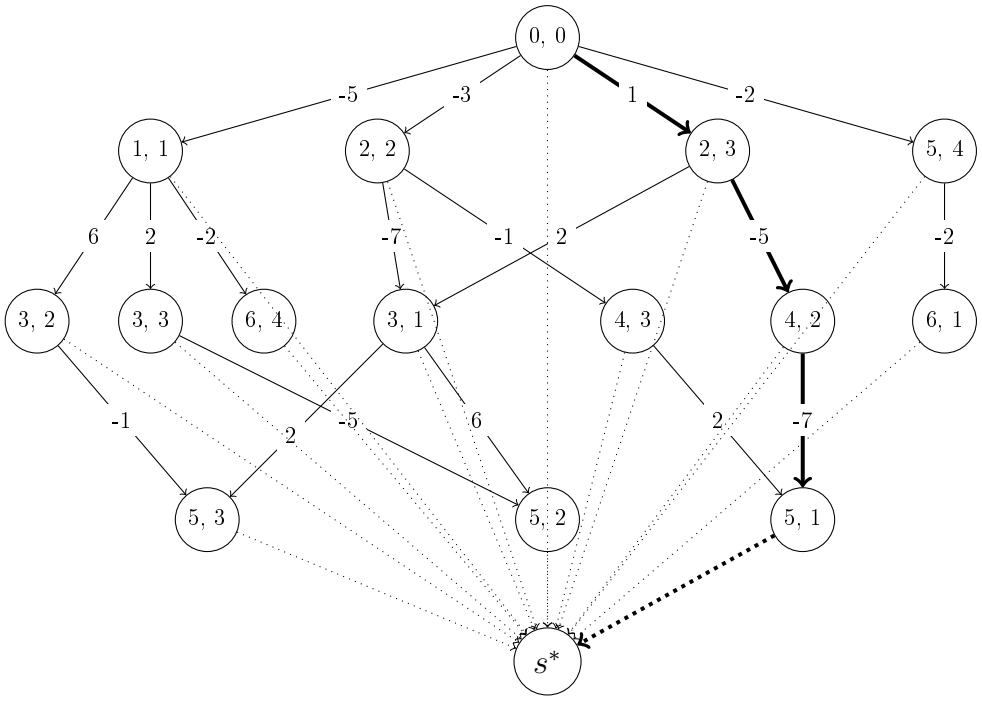

Figure 2.4: Example of network $\mathcal{N}_{\text{ESPPRC-SSR}}$ obtained from a state-space relaxation of $\mathcal{N}_{\text{ESPPRC}}$.

Figure 2.4 illustrates the network $\mathcal{N}_{\text{ESPPRC-SSR}}$ obtained from the state-space relaxation of the network from Figure 2.2. It can be noticed that nodes $(\{1,2\},1)$ and $(\{1,3\},1)$ have been aggregated into a single node $(3,1)$. As a result, $\mathcal{N}_{\text{ESPPRC-SSR}}$ has one node and one arc less than $\mathcal{N}_{\text{ESPPRC}}$. This is a minimal example of a reduction provided by $\mathcal{N}_{\text{ESPPRC-SSR}}$ that could be presented within the limits of this paper. However, due to the contrast of the exponential size of $\mathcal{N}_{\text{ESPPRC}}$ and the pseudo-polynomial size of $\mathcal{N}_{\text{ESPPRC-SSR}}$, there are many practical instances where the state-space relaxation provides a huge reduction on the size of the network.

This example shows how state-space relaxation derives pseudo-polynomial arc flow models from a network of exponential size. This kind of modeling technique is often used in vehicle routing problems, as the combinatorial structure of such problems usually leads to exponential state spaces in DW decompositions (see, e.g., Righini and Salani [284]).

2.5 Dual Insight

Research has shown that controlling the values of the dual-variables when using the primal simplex algorithm may improve computational times substantially. This happens, for instance, by modifying the model adding extra primal variables (corresponding to extra dual constraints) that must take null values at the end of the solution process. The use of this strategy may help in reducing difficulties related with instability of dual variables, primal degeneracy and long-tail effects, which are known to occur in column generation.

Other than the considerably smaller size, the use of more dual information is another advantage of the arc flow formulations over path flow formulations. Dual constraints (i.e., dual optimality conditions) of path flow formulations are non-negative combinations of dual constraints of arc-flow formulations. From a primal point of view, this competitive advantage can be interpreted as resulting from the possible recombination of basic variables to generate different paths in column generation algorithms (see, e.g., Sadykov and Vanderbeck [290]).

2.5.1 On the Dual Space of Network Flow Formulations

Many state-of-the-art algorithms to solve LP models are based on iteratively pivoting from one vertex of the LP polytope to a better neighboring vertex until an optimal solution is found. An issue that may be critical for the computational time is degeneracy. A solution is degenerate when there are basic variables with a null value; in such cases, there may be degenerate pivots that lead to the same degenerate vertex, with no change in the primal solution, nor improvement in the objective function. Hard combinatorial optimization problems are often associated with highly degenerate models, and stalling, which is a sequence of degenerate pivots, occurs in practice (see, e.g., Bazaraa et al. [17]). In fact, in a degenerate pivot, there is no change in the primal solution, but the new set of basic variables yields a change in the dual solution, leading to an alternative dual solution, often with a high oscillation of the values of the dual variables.

This instability often happens with master problems of DW reformulations, which may have a huge number of dual solutions associated to each primal solution. Several strategies showed that controlling the dual-variable values in the primal simplex algorithm may make the column generation procedure more efficient. For instance, du Merle et al. [117] introduced a stabilization procedure, combining a perturbation method and a penalty method, that amounts to penalizing dual variables when they lie outside a predefined box. Wentges [336] searches dual solutions in the neighborhood of the best dual solution found so far, thus reducing instability. Other strategies include aggregating primal constraints (aggregation may change dynamically along the solution process), which enables transferring degeneracy to a complementary problem that is able to select a more central dual solution, as in [121, 120]. Further improvements aim at identifying a set of non-basic variables that are pivoted together into the basis, avoiding degeneracy and strictly decreasing the objective function value, by solving a problem, coined as complementary problem, in [50]. For other strategies and insights on overcoming instability in combinatorial optimization algorithms, the reader is referred to, e.g., Lemaréchal [205].

Another strategy to deal with primal degeneracy and instability is to develop strong models with a more restricted dual space. When comparing different LP models for the same problem with the same primal strength, the one with a tighter description of the dual space eliminates alternative dual solutions, potentially reducing degeneracy. This concept has been used in column generation algorithms by adding dual cuts that preserve all dual optimal solutions or even at least just one dual optimal solution, leading to significant speed-ups and reductions in the number of degenerate pivots (see, e.g., Valério de Carvalho [316], Lübbecke and Desrosiers [218], and Ben Amor et al. [29]).

LP solvers rely on the dual solution to prove optimality, and each dual constraint is an optimality condition. In the following, we show that arc flow formulations provide a tighter description of the dual space than their corresponding path flow formulations, with a richer description of the optimality conditions of the LP models. For instance, consider the dual of the linear relaxation of the path flow formulation (2.13)–(2.15), given by:

$$\max \sum_{k \in \mathcal{C}} d_k \beta_k, \tag{2.50}$$

$$\text{s.t.:} \sum_{k \in \mathcal{C}} \sum_{(u,v) \in \mathcal{A}(p)} \ell^k_{(u,v)} \beta_k \leq \tilde{c}_p, \qquad \forall p \in \mathcal{P}(\mathcal{N}), \tag{2.51}$$

$$\beta_k \geq 0, \qquad \forall k \in \mathcal{C}, \tag{2.52}$$

where dual variable β_k correspond to the constraints (2.14). Now, consider the dual of the linear relaxation of the corresponding arc flow formulation (2.16)–(2.20), given by:

$$\max \sum_{k \in \mathcal{C}} d_k \beta_k, \tag{2.53}$$

$$\text{s.t.: } \alpha_v - \alpha_u + \sum_{k \in \mathcal{C}} \ell^k_{(u,v)} \beta_k \leq c(u,v), \qquad \forall (u,v) \in \mathcal{A}(\mathcal{N}), \tag{2.54}$$

$$\alpha_{source} - \alpha_{sink} \leq 0, \tag{2.55}$$

$$\alpha_v \in \mathbb{R}, \qquad \forall v \in \mathcal{V}(\mathcal{N}), \tag{2.56}$$

$$\beta_k \geq 0, \qquad \forall k \in \mathcal{C}. \tag{2.57}$$

Each dual variable α_v corresponds to the flow conservation of node $v \in \mathcal{V}(\mathcal{N})$, and each dual variable β_k corresponds to the side constraint $k \in \mathcal{C}$. For a given path $p \in \mathcal{P}(\mathcal{N})$, by performing a non-negative linear combination of the dual constraints (2.54) of every arc on $\mathcal{A}(p)$, we obtain:

$$\sum_{(u,v) \in \mathcal{A}(p)} (\alpha_v - \alpha_u) + \sum_{(u,v) \in \mathcal{A}(p)} \sum_{k \in \mathcal{C}} \ell^k_{(u,v)} \beta_k \leq \sum_{(u,v) \in \mathcal{A}(p)} c(u,v) = \tilde{c}_p. \tag{2.58}$$

Note that, each node $v \in p$, except v_{source} and v_{sink}, is the head of an arc $(u,v) \in \mathcal{A}(p)$ and the tail of an arc $(v,w) \in \mathcal{A}(p)$, producing in the first summation, respectively, the

terms α_v and $-\alpha_v$ that cancel each other. Then, (2.58) can be rewritten as:

$$(\alpha_{sink} - \alpha_{source}) + \sum_{k \in \mathcal{C}} \sum_{(u,v) \in \mathcal{A}(p)} \ell^k_{(u,v)} \beta_k \leq \tilde{c}_p, \tag{2.59}$$

which is the dual constraint from (2.51) for the path p, with an additional term $(\alpha_{sink} - \alpha_{source})$. From (2.55), we know that this term is always non-negative, implying a tighter constraint. Thus, we conclude that every dual constraint of the path flow formulation is a redundant dual constraint for the arc flow formulation, implying that the arc flow formulation provides a tighter dual space than the path flow formulation.

Generally, when the linear relaxation of either path flow or arc flow models is solved by simplex algorithms, the basis at each iteration is associated with a set of paths forming a primal-feasible solution. This set of paths is unique in the case of path flow, but not necessarily unique in the case of arc flow. If a same set of paths is considered to form the basis of an arc flow and of a path flow model, the basis of the former will be larger, due to the additional flow conservation constraints and the fact that each path is decomposed in a set of arcs in this model. But, in fact, the arc flow basis can be seen as obtained from a disaggregation of the path flow basis, which directly implies more optimality conditions. This guarantees a better description of the dual space, which, as already discussed, provides a number of practical benefits.

Concluding, in practice, each simplex iteration of an arc flow model can be more expensive when compared to a path flow model (due to the larger basis), but the number of pricing iterations needed to reach proven optimality can be substantially smaller, which in many cases is a significant advantage.

2.5.2 Example on the Cutting Stock Problem

We present a numerical example to compare the dual of the classical arc flow model from Valério de Carvalho [314] (see Section 2.4.1) to solve the CSP and the dual of its corresponding path flow model. The dual of the linear relaxation of this arc flow model is given by:

$$\max \sum_{i \in I} \beta_i, \tag{2.60}$$

$$\text{s.t.:} \quad -\alpha_v + \alpha_{v+w_i} + \beta_i \leq 0, \qquad \forall i \in I, (v, v+w_i) \in \mathcal{A}_i(\mathcal{N}_{\text{KP-SSR}}), \tag{2.61}$$

$$\alpha_{source} - \alpha_{sink} \leq 1, \tag{2.62}$$

$$\alpha_v \geq 0, \qquad \forall v \in \mathcal{V}(\mathcal{N}_{\text{KP-SSR}}), \tag{2.63}$$

$$\beta_i \geq 0, \qquad \forall i \in I. \tag{2.64}$$

Variables α_v are related to the flow conservation constraints of each $v \in \mathcal{V}(\mathcal{N}_{\text{KP-SSR}})$, and variables β_i are related to the demand constraint of each $i \in I$. Constraints (2.61) are related to the arc variables, and constraint (2.62) is related to the flow variable. The

dual of the linear relaxation corresponding path flow model is given by:

$$\max \sum_{i \in I} \beta_i, \tag{2.65}$$

$$\text{s.t.:} \sum_{i \in I} a_{ip} \beta_i \leq 1, \qquad \forall p \in \mathcal{P}(\mathcal{N}_{\text{KP-SSR}}), \tag{2.66}$$

$$\beta_i \geq 0, \qquad \forall i \in I. \tag{2.67}$$

The variables β_i are related to the demand constraints of each $i \in I$, and constraints (2.66) are related to each path (cutting pattern) p of $\mathcal{N}_{\text{KP-SSR}}$, where a_{ip} is an integer coefficient representing the number of times item i is cut from pattern p.

Consider again the example from Figure 2.3, with bin capacity 8 and three items having $w_1 = 4$, $w_2 = 3$, $w_3 = 2$, and $d_1 = d_2 = d_3 = 1$. Tables 2.1 and 2.2 present, respectively, model (2.60)–(2.64) and (2.65)–(2.67) for this example. In the dual of the linear relaxation of the arc flow model, v_{source} and v_{sink} are represented by 0 and S, respectively. In this example, the path flow model is relatively smaller than the arc flow model, which is not common for practical instances, as the former may have exponentially more primal variables. However, our goal here is only to exemplify how the dual constraints (optimality conditions) of the path flow model are redundant dual constraints for the arc flow model. This can be observed as each dual constraint of the path flow model related to a pattern can be obtained by a non-negative linear sum of the dual constraints of the arc flow model related to the arcs that form this pattern. In particular: pattern $\{1, 2\}$ can be formed by arcs $(0, 4)$, $(4, 7)$, $(7, S)$ and $(S, 0)$; pattern $\{1, 3\}$ by arcs $(0, 4)$, $(4, 6)$, $(6, S)$ and $(S, 0)$; pattern $\{2, 3\}$ by arcs $(0, 3)$, $(3, 5)$, $(5, S)$ and $(S, 0)$; pattern $\{1\}$ by arcs $(0, 4)$, $(4, S)$ and $(S, 0)$; pattern $\{2\}$ by arcs $(0, 3)$, $(3, S)$ and $(S, 0)$; pattern $\{3\}$ by arcs $(0, 2)$, $(2, S)$ and $(S, 0)$.

Table 2.1: Example of a dual arc flow formulation for the CSP.

		α_0	α_1	α_2	α_3	α_4	α_5	α_6	α_7	α_8	α_S	β_1	β_2	β_3	
arc	$(0,4)$	-1				1						1			≤ 0
	$(0,3)$	-1			1								1		≤ 0
	$(4,7)$					-1			1				1		≤ 0
	$(0,2)$	-1		1										1	≤ 0
	$(3,5)$				-1		1							1	≤ 0
	$(4,6)$					-1		1						1	≤ 0
	$(0,S)$	-1									1				≤ 0
	$(2,S)$			-1							1				≤ 0
	$(3,S)$				-1						1				≤ 0
	$(4,S)$					-1					1				≤ 0
	$(5,S)$						-1				1				≤ 0
	$(6,S)$							-1			1				≤ 0
	$(7,S)$								-1		1				≤ 0
flow	$(S,0)$	1									-1				≤ 1
max												1	1	1	

Table 2.2: Example of a dual path formulation for the CSP.

		β_1	β_2	β_3	
patterns	$\{1,2\}$	1	1		≤ 1
	$\{2,3\}$		1	1	≤ 1
	$\{1,3\}$	1		1	≤ 1
	$\{1\}$	1			≤ 1
	$\{2\}$		1		≤ 1
	$\{3\}$			1	≤ 1
max		1	1	1	

2.6 General Solution Methods

As previously discussed, an advantage of arc flow formulations over their equivalent path flow formulations is that they can be much smaller and often can be solved directly by a MILP solver (which is not practical for path flow models). However, pseudo-polynomial arc flow models can still be too large, depending on the size of the parameters of an instance. In such cases, one has to rely on more sophisticated methods to solve these models. An advantage of arc flow models derived from DW decompositions is that, since their networks are based on the underlying pricing problem, it is not always necessary to load the full network in the computer memory. Instead, one can use the structure of the pricing problem to derive methods based on column generation or iterative aggregation/disaggregation to solve the problem to integer optimality while avoiding to generate

the full network. Such methods can lead to an increase in practical efficiency and avoid memory overflow when solving instances associated with huge networks.

Column Generation

The column generation method (introduced in Section 2.3) solves the linear relaxation of models with a large number of variables, and is a popular tool to solve path flow models. The method was proposed by Ford and Fulkerson [135] to solve a path flow model for a maximal multi-commodity network flow problem and three years later generalized by Dantzig and Wolfe [100] to solve the models resulting from DW-decompositions. Gilmore and Gomory [149, 150] solved a path flow model for the CSP by a column generation algorithm and were the first to show the practical efficiency of the method. Since then, column generation has been the base of several methods to solve path flow models. For references on column generation algorithms not strictly related to arc flow formulations, we refer the interested reader to the survey by Lübbecke and Desrosiers [218] and the book by Desaulniers et al. [112].

Column generation was applied to solve an arc flow model for the first time by Valério de Carvalho [314], who proposed a column-and-row generation algorithm. In the context of arc flow models, column-and-row generation iteratively generates arcs to enter the simplex base, while the flow conservation constraints (rows) are restricted to nodes where there exists at least one incoming and one outgoing arc in the restricted problem. Sadykov and Vanderbeck [290] studied the column-and-row generation method and experimentally compared the solution of path flow models by column generation, with the solution of equivalent arc flow models by column-and-row generation, and observed a faster convergence of the latter, which follows the discussion in Section 2.5.

In column generation algorithms (and lagrangian relaxations where only flow conservation constraints are left in the master) to solve arc flow models, an LPP on the underlying network is iteratively solved, and its computational efficiency is strictly related to the network size. In this context, when the network is too large, one may rely on dynamic graph generation methods to solve the LPP, as proposed by, e.g., Fischer and Helmberg [131], to solve LPPs that arise as sub-problems from arc flow models based on large-scale time-expanded networks.

Iterative Aggregation/Disaggregation Method

Several authors have studied state-space relaxation techniques to reduce the size of arc flow formulations (see, e.g., Macedo et al. [221], Clautiaux et al. [83], Voge and Clautiaux [327], Boland et al. [40], Boland and Savelsbergh [42], and Riedler et al. [279]). These techniques are equivalent to applying a surrogate relaxation to the flow conservation constraints related to subsets of nodes. Theoretically, this only reduces the number of constraints. Practically speaking, after an aggregation, many arcs (variables) associated with the same decision become equivalent in the reduced network and can be merged. From an initial relaxation, these methods use iterative techniques in which the relaxation is refined, typically by splitting nodes that have been aggregated, until the solution produced

by the relaxation is feasible for the original problem, or its value is equal to a known primal bound.

Macedo et al. [221] were the first to use these techniques in pseudo-polynomial arc flow models. The authors used two aggregations: one produces a relaxation, the other a heuristic solution. These results were later generalized by Voge and Clautiaux [327] and Clautiaux et al. [83], who studied the difficulty of the different sub-problems and the performance ratio obtained by an aggregated model. More efficient refining strategies are studied in Riedler et al. [279]. The authors show that their path-based techniques are more effective and underline the importance of heuristic methods in the algorithm. In early works, the elements to be aggregated were decided beforehand. In Boland and Savelsbergh [42], the authors introduced the paradigm of *dynamic discretization discovery*, in which the discretization is constructed on the fly, producing a better relaxation, by using information from the network construction process.

Such aggregation techniques have been generalized to flows in hypergraphs by Benkirane et al. [32] to solve a joint rolling-stock and train selection problem for the French railway company. The authors show that even for hypergraphs, one can safely use reduced-cost filtering on aggregated variables. Another type of relaxation is used in [249] to deal with problems where several constraints can be reformulated as network-flow constraints. In their model, a network is created for each constraint, and the flow conservation constraints of all networks but one are relaxed in a Lagrangian way.

Graph Reduction Methods

An important element of efficient solution methods for arc flow models is to determine arcs that can be removed from the network without losing optimality. Removing redundant arcs is important, as a smaller network may lead to a reduction in symmetry, a tighter relaxation, and a smaller branch-and-bound tree. Many techniques to reduce the number of arcs are problem-dependent, usually based on dominance criteria of the underlying DP, and we discuss some of them in Section 3.7, under specific applications. Nonetheless, general reduction techniques have been used to enhance arc flow models, as, for instance, the reduced-cost variable-fixing method (see, e.g., Pessoa et al. [265] and Kramer et al. [199]).

Given a MILP minimization model, the reduced-cost variable-fixing method performs a domain propagation based on a dual-feasible solution of the corresponding linear relaxation and an upper bound value z_{ub} corresponding to an available feasible solution. The idea is that, given the objective value z_{lb} of the dual-feasible solution, any integer variable with a reduced cost greater than or equal to $z_{ub} - z_{lb}$ can be removed from the model. Irnich et al. [183] proposed an efficient reduced-cost variable fixing algorithm for eliminating arcs in network flow models that computes a bound on the reduced cost of the arcs in arc flow models based on a dual solution of the linear relaxation of its equivalent path flow model. This method was later extended by Desaulniers et al. [113] to determine and efficiently handle pairs of sequential arcs that cannot be in the same path in an optimal solution. The impact of different dual solutions in reduced-cost variable-fixing for network flow models has been recently discussed by de Lima et al. [104].

2.7 Successful Applications of Pseudo-Polynomial Arc Flow Models

In this section, we discuss the main applications of arc flow models, and discuss problem-dependent solution methods and reduction criteria.

2.7.1 Cutting and Packing Problems

Arc flow models have been used in a variety of cutting and packing problems, both in one and multiple dimensions.

One-dimensional Problems

The classical arc flow model for the CSP in [314] has a node for each partial stock size, the arcs relate the items with cut positions (item arcs) or represent loss stock (loss arcs), and the combination of arcs into paths represents cutting patterns. To solve this model to integer optimality, Valério de Carvalho [314] proposed a branch-and-price algorithm based on column-and-row generation. To accelerate the column generation's convergence, the pricing problem generates paths (instead of single arcs). Valério de Carvalho [314] proved that the arc flow model is equivalent to the path (set-covering) model by Gilmore and Gomory [149, 150], whereas Martinovic et al. [235] and Delorme and Iori [107] proved that the arc flow model is equivalent to the one-cut model by Rao [276] and Dyckhoff [118]. The one-cut is a pseudo-polynomial model where variables represent cutting operations on the roll.

To reduce the number of arcs, Valério de Carvalho [314] constructs the graph considering that items of a single roll can always be ordered by non-increasing width. The resulting graph independently follows the state-space relaxation discussed in Section 2.4.1. To obtain even smaller networks, Côté and Iori [93] proposed the meet-in-the-middle technique: each path representing a cutting pattern can be transformed into an equivalent one by left aligning the items whose left border is at the left of a given threshold parameter t, and right aligning the remaining items. Recently, de Lima et al. [104] proposed a new way to reduce the graph in [314] by considering a maximum waste for each roll. They developed a branch-and-price framework in which the branching produces a series of small arc flow models which are solved one at a time by a general purpose MILP solver.

Cambazard and O'Sullivan [62] proposed the DP-flow, an arc flow model for the CSP (already introduced in Section 2.3.1) based on the DP network of the KP. Differently from the classical arc flow model for the CSP, which considers a node for each partial stock size, the DP-flow considers a node for each pair of items and partial stock size. The network has a level for each item, and each feasible path visits the level of each item exactly once. Although this modeling technique can substantially increase the number of nodes, it allows one to consider only proper patterns. On the other hand, the classical arc flow model is smaller, but it cannot distinguish between proper and non-proper patterns, so its relaxation can be weaker than the one of the DP-flow.

A generalization of the classical arc flow model for the CSP was proposed by Brandão

and Pedroso [54]. This generalization can model related problems, such as the vector BPP, the BPP with conflicts, the cardinality constrained BPP and CSP, and the graph coloring problem. The resulting network may be significantly large, but several techniques (referred to as graph compression) are proposed to reduce the network size. To break symmetry, the authors proposed a modeling technique similar to the graph construction of the DP-flow, considering an extra dimension, related to the items, on the nodes.

Delorme and Iori [107] proposed the *reflect formulation*, a pseudo-polynomial model for the CSP that considers nodes and arcs only from half of the stock size. In practice, this formulation is significantly smaller than the classical arc flow model. Besides, the reflect formulation has been proven to be as strong as the classical arc flow model without reduction criteria. Other than the CSP, Delorme and Iori [107] extended the reflect formulation to solve the variable-sized BPP and the BPP with item fragmentation. Dell'Amico et al. [105] adapted the reflect formulation to solve a feasibility problem in a Benders' decomposition algorithm for the multiple knapsack problem.

Alves and Valério de Carvalho [9] presented a branch-and-price-and-cut algorithm for the multiple length CSP that solves the Gilmore and Gomory [150] machine balance model, which is a path flow formulation, using the original arc flow model to generate attractive columns in a single subproblem and the variables of the arc flow model to implement a branching scheme, by expressing the branching constraints in terms of the Gilmore and Gomory model variables. The equivalence of the path and arc flow models ensures a correct transferral of dual information. Valid dual inequalities are used to stabilize and accelerate the search in the entire branch-and-bound tree.

Recently, arc flow models were proposed for the *skiving stock problem* (SSP), which is strongly related to the dual BPP. In the SSP, we are given a set of items, each having a length and a maximum number of copies, to be combined into the maximum number of larger items of minimum length W. Martinovic and Scheithauer [233] proposed an arc flow model for the SPP where nodes represent the length sum of combinations of items, arcs represent the positioning of the items in a combination, and paths represent a combination of items. This model, which is similar to the classical arc flow model for the CSP, cannot distinguish between proper and non-proper patterns. Martinovic and Scheithauer [234] proposed an arc flow model that considers only proper patterns, which, similarly to the DP-flow model, considers a node for each pair of items and possible length sum, and the network has a level for each item. Martinovic et al. [232] proposed two arc flow models for the SSP: one considers reversed loss arcs, which lead to a reduction of the worst-case number of nodes from $2W$ to W, and the other is based on the reflect model for the CSP.

Multi-dimensional Problems

Macedo et al. [220] extended the classical arc flow model to minimize the number of bins in the two-stage two-dimensional guillotine CSP. Their model has a one-dimensional arc flow graph for the first stage cuts (which cut the bins to obtain strips), and a one-dimensional arc flow graph for each possible strip size to determine the second stage cuts (which produce the items from the strips, possibly admitting a final trim loss cut). To

reduce symmetry, the flow in the second stage graph of a given strip size is equal to the sum of the flows for that strip size in the first stage graph. Note that these strips may belong to the same bin or different bins. Procedures to reduce the size of the graph and a new family of cutting planes based on the height of the items were proposed. The arc flow model in Macedo et al. [220] was later adapted by Mrad [246] to solve the two-stage two-dimensional guillotine strip packing problem.

Nesello et al. [251] proposed an arc flow model, similar to the one by Macedo et al. [220], to solve a three-stage two-dimensional strip packing problem where a limit is imposed on the number of shelves and setup times between items must be taken into account. Delorme et al. [110] adapted the classical arc flow model for the CSP to solve the one-dimensional contiguous bin packing problem, which often appears as a sub-problem in two-dimensional cutting and packing solution methods. The authors used it to solve the sub-problem of a Benders' decomposition algorithm for the two-dimensional strip packing with item rotations and for the pallet loading problem.

Clautiaux et al. [86] solved the four-stage two-dimensional guillotine bounded knapsack problem with a network flow model based on a directed acyclic hypergraph. They compared the efficiency of several algorithms based on this representation, including a MILP model and an iterative state-space relaxation based on the corresponding DP.

2.7.2 Scheduling Problems

In the scheduling field, both time-indexed (see, e.g., Sousa and Wolsey [302]) and path flow (see, e.g., van den Akker et al. [318]) formulations have been widely used to solve a variety of optimization problems. In a closely related context, polynomial arc flow models have been proposed more than three decades ago by Eppen and Martin [123] to solve a lot-sizing problem. However, pseudo-polynomial arc flow formulations have only been adopted for scheduling problems during the last decade. It is worth mentioning that time-indexed formulations can be used to derive arc flow models of the same strength, which are associated to a sparser constraint matrix. The equivalence between time-indexed and arc flow formulations has been shown in different contexts starting from Valério de Carvalho [315], who proved such equivalence by a unimodular transformation, in the context of the CSP.

Pessoa et al. [265] developed a branch-and-cut-and-price algorithm for the problem of minimizing weighted tardiness on identical parallel machines (denoted as $P||\sum w_j T_j$ in the three-field classification of Graham et al. [156]). Their algorithm is based on an arc-time-indexed formulation and is improved with a number of combinatorial techniques, including variable fixing by reduced costs, extended capacity cuts, dual stabilization, and the direct solution of the formulation by a MILP solver if the fixing procedure had consistently reduced the number of variables. The method in Pessoa et al. [265] was later extended by Bulhões et al. [55], who proposed a branch-and-cut-and-price algorithm to solve a path flow formulation for parallel machine scheduling in which the branching is based on the variables of the arc flow model.

Lancia et al. [202] developed a branch-and-price algorithm for the job shop problem with a general min-sum objective function. Their algorithm is based on the solution of

an arc flow model in which a path has to be chosen for each job (which is composed of multiple operations). The model was strengthened by clique inequalities.

Ratli et al. [277] presented a comprehensive list of mathematical models for scheduling jobs on a single machine by minimizing weighted earliness and tardiness. The problem, denoted as $1||\sum \alpha_j E_j + \sum \beta_j T_j$, is relevant in the context of just-in-time production. In computational tests on random instances, the arc flow model achieved the lowest optimality gaps.

Mrad and Souayah [248] presented a direct extension of the arc flow formulation by Valério de Carvalho [314] to the problem of scheduling jobs on identical parallel machines with the objective of minimizing the makespan ($P||C_{\max}$).

Kramer et al. [197] considered again the problem of scheduling jobs on identical parallel machines, but focused on the minimization of the weighted sum of the completion times ($P||\sum w_j C_j$). They presented an arc flow model, and then enhanced it by grouping jobs having the same weight and processing time and creating time windows for each group by considering job priorities. A computational comparison showed that the enhanced arc flow performed very well compared to other time-indexed, convex integer quadratic programming, and path flow models.

Kramer et al. [198] extended the work in [197] to deal with the case of family setup times ($P|s_i|\sum w_j C_j$). The authors proposed three different arc flow models. In the most efficient one, the network is divided into a set of layers, one layer per family. Arcs within the same layer considered only the processing time of a job, whereas arcs connecting two layers considered both setup and processing times. Computational results showed that setup times worsen the linear relaxation value of the arc flow models, which outperformed a path flow model only on a handful of instances.

A further generalization of [197] was provided in [196], who considered the case of release dates ($P|r_j|\sum w_j C_j$) and obtained stricter time windows for the jobs. The resulting arc flow model obtained better results than a branch-and-price based on the path flow formulation on instances having jobs of small and moderate processing time.

We also mention that interesting integrations between scheduling and other combinatorial problems have been tackled in the literature. Cappanera and Scutellà [63] considered a joint assignment, scheduling, and routing problem arising in home care optimization. An arc flow model was used for the generation of patterns, which represent feasible combinations of the three decision levels of the problem. Cire et al. [79] proposed an arc flow formulation for a rotation assignment and scheduling problem arising in the context of clinical rotations. They demonstrate that the network model was computationally superior to a classical MILP model on a real-world set of instances.

Braga et al. [53] compared two formulations for a combined cutting stock and scheduling problem: a compact formulation strengthened with knapsack inequalities and an arc flow formulation. Using a revised version of the latter formulation based on aggregated time periods, they derived a heuristic solution procedure that proved to be effective for the solution of medium size instances. Rietz et al. [280] proposed and analyzed an arc flow formulation for a combined cutting stock and scheduling problem on parallel machines. Different strategies to simplify the formulation by reducing the number of arcs were presented.

Trindade et al. [308] solved scheduling problems with batch processing machines. In such problems, the jobs are grouped into batches to be scheduled in machines, where the batches have a capacity to be respected by the size of its grouped items. The author proposed arc flow models where nodes represent a discretization of the capacity of a batch and arcs represent either a scheduled job in a batch or an unused capacity.

2.7.3 Routing Problems

In routing problems, the input is usually based on a network, where clients and depots are given as nodes and arcs are related to transportation between the nodes (see, e.g., Toth and Vigo [306]). For such problems, arc flow models based on the input network lead to compact models that are small but may have weak relaxations and too much symmetry to be solved in practice. Examples of such compact models are the classical MTZ model by Miller et al. [241] for the traveling salesman problem (TSP) and the three-index formulation (see Section 2.3.2) for the CVRP. In many routing problems, the ability to avoid non-elementary routes is an important aspect to determine the strength of a relaxation. For this reason, the best solution methods for many variants, especially the ones with multiple vehicles, are usually based on path flow formulations (see, e.g., Pessoa and Uchoa [269] and Pessoa et al. [263]), as they can handle non-elementary routes more easily than arc flow formulations. However, pseudo-polynomial arc flow formulations have still been used to solve open problems of a number of variants, and to enhance state-of-the-art methods based on path flow formulations.

Pseudo-polynomial arc flow formulations for vehicle routing problems were first introduced by Godinho et al. [152] and Pessoa et al. [261], which, inspired by the early work by Picard and Queyranne [266] for single machine scheduling with minimum tardiness, proposed the capacity-indexed formulation for the CVRP. The capacity-indexed formulation, which follows the network from the state-space relaxation in Section 2.4.2, has a node (i, q) for each pair of client (or depot) i and partial capacity q, where the partial capacities group the nodes into levels (layers). Then, each path arriving at a node (i, q) represents a route that finishes in client i and has total load q.

The capacity-indexed formulation is based on a network where paths may be associated with non-elementary routes (which weakens its linear relaxation), and may be too large to be solved in practice (see, e.g., Pessoa et al. [261]). The main interest in this formulation is that its variables can be used to define cutting planes (e.g., the extended capacity cuts) to strengthen path flow formulations for vehicle routing problems (see, e.g., Pessoa and Uchoa [269]). According to Uchoa [310], cutting planes based on the capacity-indexed formulation have been successfully used in other vehicle routing problems, and even in parallel machine scheduling problems.

Macedo et al. [221] proposed an iterative aggregation/disaggregation algorithm to solve an arc flow formulation for the vehicle routing problem with time windows and multiple routes. In the underlying network, each node corresponds to a time instant, and each arc corresponds to a possible subtour. Aggregation based on rounding procedures was used to make the routes with non-integer travel times fit the model's graph. Whenever an infeasible solution was found, the nodes involved in the infeasibility were disaggregated,

and the resulting model was solved again.

Braga et al. [52] proposed an arc flow formulation for the multi-trip inventory routing problem where vehicles can perform more than a single route per time period. Following Macedo et al. [221], nodes and arcs of the underlying network correspond to, respectively, time instants and routes that may be assigned to a vehicle.

Algorithms based on iterative aggregation/disaggregation were later proposed to solve time-expanded arc flow formulations for the TSP with time windows (TSPTW). The network of time-expanded formulations has a node (i, t) for each client i and time instant t. Each path from the source to a node (i, t) represents a path arriving at client i at time t. Boland et al. [41] and Riedler et al. [279] proposed iterative aggregation/disaggregation algorithms to solve a time-expanded model for the TSPTW, starting with a reduced network that is sufficient to produce a bound for the problem and is iteratively refined to find an optimal solution. The method in Boland et al. [41] was later extended by Vu et al. [328] to solve the generalization where travel times are time-dependent. This kind of approach was also used by Boland et al. [40] to solve a continuous-time service network design problem without the need to approximate the solution by a discretization of the time-horizon.

2.7.4 Miscellaneous

Earth Observation Satellite scheduling requires to determine the pictures to be taken by a set of satellites in a given time period, so as to satisfy side constraints and optimize an objective function. Pseudo-polynomial arc flow models have been proposed to solve such problems by Gabrel and Murat [142] and Wang et al. [331]. In these models, the nodes are related to a discretization of the time horizon, and the arcs are related to the decisions on the pictures to be taken.

Kramer et al. [199] solved the dynamic berth allocation problem, which aims at allocating vessels into quays that are divided into berths, while optimizing an objective function based on the service time of each vessel. The authors proposed an arc flow model where nodes are associated with time instants and arcs are related to vessels serving. Problem-dependent reduction criteria and a reduced-cost variable-fixing algorithm were proposed to improve the solution time.

In the capacitated p-center problem, a set of customers must be attributed to capacitated facility locations, minimizing the maximum distance between each client and its facility. To solve this problem, Kramer et al. [200] proposed an arc flow model where the underlying graph has a component for each location, in which nodes correspond to partial filling of the facility capacity, and arcs correspond to customers and unused capacity.

Ramos et al. [275] described an arc flow formulation for the multi-trip production, inventory, distribution, and routing problem with time windows. Nodes and arcs represent time instants and vehicle routes, respectively.

Train timetabling problems require to determine a periodic timetable for a set of trains that satisfies operational constraints and optimizes an objective function (see, e.g., Cacchiani and Toth [61]). The input for such problems is based on a graph where nodes represent stations and arcs represent tracks. Train timetabling problems have been suc-

cessfully solved by arc flow models based on time-expanded networks by Caprara et al. [65] and Fischer and Helmberg [131].

Recently, van Hoeve [319] proposed an arc flow model to solve the graph coloring problem, which asks to partition a graph into the minimum number of independent sets. The proposed arc flow model is based on decision diagrams, which are closely related to DP (see, e.g., Hooker [170]) and result in acyclic graphs. The resulting network in van Hoeve [319] is equivalent to a DP network to solve the maximum independent set problem, which is usually the pricing problem of path (set-partitioning) formulations for the graph coloring problem (see, e.g., Gualandi and Malucelli [159]). An iterative refinement method is proposed by the authors to deal with the exponential size of the model.

2.8 Conclusion and Future Research Directions

In this survey, we reviewed over one hundred papers related to arc flow formulations. Many of these papers present arc flow models having pseudo-polynomial size and a strong linear relaxation. The number of applications of pseudo-polynomial arc flow formulations has grown considerably since the work by Valério de Carvalho [314], making them a valid alternative to path flow formulations. For many combinatorial optimization problems, path flow formulations are still the best alternative, because they can embed difficult constraints in the subproblem solved to build the paths. However, arc flow formulations have many positive aspects, among which we highlight that: they are powerful modeling tools that allow one to model complex issues from real systems; pseudo-polynomial arc flow models are often related to DW decompositions, providing strong linear relaxations; differently from path flow formulations, they provide models that usually have a number of variables which allows practical solutions directly by general MILP solvers, avoiding complex implementations; compared to equivalent path flow formulations, they have a richer description of the dual space, leading to a faster convergence of simplex-based methods; pseudo-polynomial arc flow models can be derived from state-space relaxations from the underlying DP network from path flow models.

This survey is a contribution to a systematic study of arc flow formulations, but we would like to point out that there are many open questions and research lines to be pursued. Some of them are the following.

One of the advantages of arc flow formulations is that they provide a tighter description of the dual feasible space keeping the primal strength. Besides the methodologies presented in this paper, are there any general hints on how to do both primal and dual strengthening in arc flow models? For example, in extended formulations, using new sets of variables may strengthen the primal model and, as new (primal) variables are dual cuts, there is also a richer description of the dual feasible space.

Models presented in this survey explore solution spaces that are convex combinations of flows, each corresponding to an $s-t$ path, which is a sequence of arcs and vertices. Are there other types of structures (e.g., involving sequences of operations) that also lead to models with strong bounds? In fact, there are pseudo-polynomial models that, instead of using a sequence of arcs to form a path (which is an extremal solution), use a sequence of

operations to form a solution that is extremal. Examples are Dyckhoff [118] for the CSP, and Silva et al. [297] and Furini et al. [141] for the two-dimensional CSP with guillotine cuts. In these cases, one-cut operations are/have to be combined to form a cutting pattern (the extremal solution).

There are pseudo-polynomial models that do not provide LP lower bounds as strong as those of column generation (e.g., position indexed models for two-dimensional non-guillotine CSP). Is there any structural (extremal) property, in these cases, that can be explored and may lead to models with stronger bounds?

Several solution methods for large-scale arc flow models, like the ones presented in Section 2.6, are general and can be applied to any arc flow model based on DP. A software/library containing such general tools to solve general large-scale arc flow models would be an interesting contribution to the optimization and operations research community.

Chapter 3

Exact Solution of Network Flow Models with Strong Relaxations

3.1 Introduction

Mixed Integer Linear Programming (MILP) is one of the most popular mathematical programming tools to solve optimization problems. *Dantzig-Wolfe* (DW) decomposition [100] has been extensively used to enhance the strength of MILP models in many applications, by reformulating an original *compact model* into a so-called *Dantzig-Wolfe Master* (DWM) problem where variables represent integer solutions of polyhedra induced by a set of constraints from the compact model. A significant importance of models with stronger linear relaxation is to avoid an excessive enumeration in branch-and-bound trees. Moreover, as the polyhedron of their linear relaxation is closer to the convex hull of the integer solutions, their optimal fractional solutions may provide good hints on how to derive good-quality integer solutions.

Models derived from DW decomposition may be stronger than the original compact models, but they usually require an exponential number of variables. Consequently, their linear relaxation is typically solved by column generation (CG), a method where a restricted set of variables is initially considered and new variables are iteratively generated by solving a pricing problem (see, e.g., [218]). The exact solution of a DWM usually relies on sophisticated branch(-and-cut)-and-price (B&P) algorithms, in which CG (and primal cuts) is embedded within branch-and-bound. These algorithms are state-of-the-art for many problems, but they are complex to implement, and their efficiency usually relies on problem-specific techniques.

The relation between integer solutions of bounded polyhedra with paths in acyclic networks allows us to associate a discrete pricing problem with an acyclic decision network in which each path is associated with a DWM variable (see, e.g., [321]). Hence, the DWM can be seen as a *path flow model* (where variables correspond to the flow on each *path* of the network). The interest in such representation is that it allows us to derive an *arc flow model* (where variables correspond to the flow on each *arc*) having the same linear relaxation value but exponentially fewer variables. In particular, any DWM of exponential size that allows a pseudo-polynomial pricing algorithm can be transformed

into an equivalent arc flow model of pseudo-polynomial size.

To the best of our knowledge, Valério de Carvalho [314] was the first to solve a pseudo-polynomial arc flow model in practice, over two decades ago, by developing a B&P for the cutting stock problem (CSP). Still, the popularity of such models increased only in the last decade, following the consistent improvements in general MILP solvers. The fact that pseudo-polynomial arc flow models have the same strength, and a much smaller size, than their equivalent path flow models has allowed their solution by general MILP solvers to be an efficient alternative to sophisticated B&P algorithms in many applications (see, e.g., [101]). However, large-scale instances often produce huge networks, which become a strong limitation in the efficiency of the MILP solver. In addition, in cases where the linear relaxation is not very strong, specialized B&P algorithms are still a better alternative than the solution of an arc flow model by a MILP solver. Further research in this area is thus highly envisaged.

In this paper, we propose a general framework, which we call *network flow framework* (NF-F), to address the issue raised by huge networks in the solution of arc flow models, while exploiting the potential of CG and general MILP solvers. We focus on arc flow models of pseudo-polynomial size with very strong linear relaxation, where: either (i) finding an optimal solution is hard, but proving optimality is easy; or (ii) finding an optimal solution is easy, but proving optimality is hard. Both cases happen, for instance, in the classical pattern-based formulation for the CSP [149, 150], which is a path flow model derived by a DW decomposition, in which the optimal integer solution value is conjectured to be either $\lceil z_{lb} \rceil$ (case (i)) or $\lceil z_{lb} \rceil + 1$ (case (ii)), where z_{lb} is the optimal dual bound (see, e.g., [64]).

NF-F is effective in solving case (i) instances thanks to an innovative and highly asymmetric branching scheme in which a series of small-sized arc flow models are solved by a general MILP solver, in the spirit of the classical local branching [132]. These models are expected to be sufficiently small to be quickly solved, but large enough to likely contain an optimal solution. In instances of case (ii), once an optimal solution is at hand, optimality is typically proven by completely exploring a branch-and-bound tree. This process is usually accelerated by the use of primal cuts to strengthen the relaxation. However, in many cases, the most effective cuts are problem-dependent and usually produce a heavy impact on the pricing problem, while the huge number of arcs may still be an unaddressed major issue. NF-F does not exploit the use of general primal cuts but rather focuses on the use of reduced-cost variable-fixing (RCVF) procedures, which are usually very effective for instances of case (ii). We particularly exploit the impact of sub-optimal dual solutions in variable-fixing with a two-fold interest: providing an increased reduction in the network size and possibly strengthening the linear relaxation.

We also provide specific contributions regarding the linear relaxation solution of arc flow and path flow models, as: a formalization of a dual correspondence between these models; a non-trivial CG algorithm that generates multiple paths and can be used to solve the relaxation of both models; and a generalization of a method to deal with dual-infeasibility in CG due to the limited precision of most Linear Programming (LP) solvers. We provide applications to cutting and packing (C&P) problems that allow pseudo-polynomial models with very strong relaxation, namely, the CSP, the two-stage

guillotine CSP, the skiving stock problem, and the ordered open-end bin packing problem. For all such problems, we performed extensive computational experiments that prove the outstanding performance of NF-F.

The remainder of this paper is organized as follows. Section 3.2 presents a brief review on network flow models derived from DW decompositions. Section 3.3 provides an overview of NF-F. Sections 4.3, 3.5, and 3.6 present the details of the techniques that we use to solve the linear relaxation, apply RCVF, and perform branching, respectively. Section 3.7 discusses the applications to a number of C&P problems. The outcome of the computational experiments is discussed in Section 7.4, where we also contrast our results with those obtained by state-of-the-art algorithms. Finally, Section 7.5 gives conclusions and future research directions. A preliminary version of this work appeared in [104].

3.2 Network Flow and Dantzig-Wolfe Decompositions: Preliminaries

Due to a well-known relation between integer solutions of polyhedra with paths in acyclic networks (see, e.g., [321]), a DWM can be directly represented as a path flow model. Then, based on the flow decomposition theorem by Ahuja et al. [1], we can derive an arc flow model of the same strength (i.e., same linear relaxation value). This section briefly reviews path flow and arc flow models derived from DW decompositions.

A network $\mathcal{N}$ is a directed graph with a set of nodes $\mathcal{V}$ and a set of arcs $\mathcal{A} \subseteq \mathcal{V} \times \mathcal{V}$. Two special nodes in $\mathcal{V}$ are the source v^+ and the sink v^-, for which no arcs in $\mathcal{A}$ enters v^+ or leaves v^-. The set of all paths from v^+ to v^- is given by $\mathcal{P}$, and the set of all arcs of a path $p \in \mathcal{P}$ is given by $\mathcal{A}_p$. The set of all paths in $\mathcal{P}$ that contain an arc $(u,v) \in \mathcal{A}$ is given by $\mathcal{P}_{(u,v)}$.

We assume that $\mathcal{N}$ is acyclic, as we are only concerned with decision networks underlying pricing problems from DW decompositions. In this context, $\mathcal{V}$ and $\mathcal{A}$ represent, respectively, the states and decisions of a pricing algorithm. Each arc/decision $(u,v) \in \mathcal{A}$ is associated with a cost $c_{(u,v)} \in \mathbb{Z}$ and a contribution $a_{(u,v)} \in \mathbb{Z}^m$ to the constraints induced by the problem. Each path $p \in \mathcal{P}$ in the pricing network represents a DWM variable with cost $c_p = \sum_{(u,v)\in\mathcal{A}_p} c_{(u,v)}$ and constraint coefficients given by column $a_p = \sum_{(u,v)\in\mathcal{A}_p} a_{(u,v)}$. Then, the DWM can be formulated as a path flow model:

$$\min \sum_{p\in\mathcal{P}} c_p\lambda_p, \tag{3.1}$$

$$\text{s.t.:} \sum_{p\in\mathcal{P}} a_p\lambda_p \geq b, \tag{3.2}$$

$$\lambda_p \in \mathbb{Z}_+, \qquad \forall p \in \mathcal{P}, \tag{3.3}$$

where $b \in \mathbb{Z}^m$ and each variable $\lambda_p \in \mathbb{Z}^+$ represents the flow on path $p \in \mathcal{P}$. The following general arc flow model is equivalent to (4.6)–(4.8):

$$\min \sum_{(u,v)\in\mathcal{A}} c_{(u,v)}\varphi_{(u,v)}, \tag{3.4}$$

$$\text{s.t.: } F_{\mathcal{N},\varphi}(v) = \begin{cases} -z, & \text{if } v = v^+, \\ z, & \text{if } v = v^-, \\ 0, & \text{otherwise,} \end{cases} \qquad \forall v \in \mathcal{V}, \tag{3.5}$$

$$\sum_{(u,v)\in\mathcal{A}} a_{(u,v)}\varphi_{(u,v)} \geq b, \tag{3.6}$$

$$\varphi_{(u,v)} \in \mathbb{Z}_+, \qquad \forall (u,v) \in \mathcal{A}, \tag{3.7}$$

$$z \in \mathbb{Z}, \tag{3.8}$$

where $F_{\mathcal{N},\varphi}(v) = \sum_{(u,v)\in\mathcal{A}} \varphi_{(u,v)} - \sum_{(v,w)\in\mathcal{A}} \varphi_{(v,w)}$. Each variable $\varphi_{(u,v)}$ corresponds to the flow on arc $(u,v) \in \mathcal{A}$ and variable z gives the total flow.

The flow decomposition theorem by Ahuja et al. [1] guarantees a one-to-many correspondence that preserves the objective value, between the primal space of the linear relaxation of path flow and arc flow models that are based on the same network. In this way, formulations (4.6)–(4.8) and (3.4)–(4.4) have the same primal strength. This allows us to handle primal solutions of both path flow and arc flow models equivalently. In Section 4.3, we also formalize a correspondence between the dual space of such models.

Examples of equivalent arc flow and path flow models derived from DW decompositions include the arc flow model in [314] and the pattern-based formulation in [149, 150] for the CSP. In the context of the capacitated vehicle routing problem, equivalent models are the q-route formulation by Christofides et al. [76] and the capacity-indexed formulation (see, e.g., [138, 261]). Although path flow and arc flow models based on the same network solve the same problem and have equally strong relaxations, each formulation may be preferable under certain specific aspects, which we contrast in the following.

Overall Solution. The solution of path flow models typically relies on B&P due to the exponential number of variables. State-of-the-art B&P algorithms are usually specialized and contain problem-specific features (with the recent exception of [264], which solves a variety of problems). Arc flow models too can be solved by B&P, but, due to their much smaller size, they may be solved in practice by MILP solvers, which is the most popular approach for these models. Few authors explore the correspondence between path flow and arc flow models by combining them into a unique solution method. For instance, Pessoa et al. [265] proposed a B&P algorithm based on a path flow model that solves the problem at the root node as an arc flow model by a MILP solver if the network is sufficiently small.

Dealing with Exponential Networks. In many strongly $\mathcal{NP}$-hard problems, the pricing only admits a decision network of exponential size (unless $\mathcal{P} = \mathcal{NP}$). Path flow models usually address the exponential nature of the network implicitly in specialized pricing algorithms that consider dominance and reduction criteria. On the other hand, practical solutions for arc flow models are limited to networks of at most pseudo-polynomial size. However, pseudo-polynomial arc flow models can be derived from exponential networks

by relying on a state-space relaxation of the pricing (see, e.g., [101]).

Linear Relaxation Solution. The LP relaxation of path flow models is usually solved by CG. For arc flow, the LP relaxation of small- and medium-sized models may be efficiently solved by simplex/barrier algorithms. Still, larger models must rely on column-and-row generation (as in [314]). The basis of a path flow model is smaller as it does not consider flow conservation constraints. Thus, at each iteration, the restricted master problem is solved faster. In contrast, an arc flow basis is larger but less degenerate, so CG usually converges within fewer pricing iterations (see, e.g., [101, 290]).

Branching and Cutting Planes. It is well known that additional constraints based on arc flow variables are *robust*, i.e., they do not impact on the pricing structure. In contrast, additional constraints based on path flow variables are often non-robust and may heavily impact the pricing. In the context of network flow, this impact may cause an expansion of the network, increasing its complexity. Some authors solve path flow models by using robust branching and cutting planes based on arc flow variables (see, e.g., [9, 310]). However, many state-of-the-art B&P algorithms for path flow models still use non-robust branching constraints and cuts, since the improvement in terms of strength may pay off the increased pricing complexity. Then, a clear disadvantage of solving arc flow models with weak relaxations directly by a MILP solver is that they must rely solely on branching and cutting planes for general MILP, which may not be competitive when compared to the state-of-the-art for the equivalent path flow models.

To summarize, path flow better addresses networks of exponential size and is also better suited for methods that rely on non-robust branching and/or cutting planes. On the other hand, arc flow is usually a good alternative to address problems with pseudo-polynomial networks and when the use of non-robust branching or cutting planes is not crucial, and their reduced size often allows the solution by MILP solvers.

3.3 An Overview of the Solution Framework

In this section, we provide an overview of NF-F. A depictive example is given in Figure 4.1. The input of NF-F is a network $\mathcal{N}$, an upper bound z_{ub} on the optimal integer solution value, the coefficients $c_{(u,v)}$ and $a_{(u,v)}$ for each arc $(u,v) \in \mathcal{A}$, and b. The input is first used to build a path flow model of type (4.6)–(4.8). The LP relaxation of the model is solved by the CG algorithm of Section 4.3. To remove arcs that do not improve the current incumbent, we use three RCVF strategies, each based on a different search for dual solutions (described in Section 3.5). These strategies are effective in those cases where the optimal solution is at hand and the challenge is to prove optimality. The first strategy is a traditional one based on the dual solution obtained at the end of the CG for the original LP relaxation. The second and third strategies are more expensive and are based on dual solutions obtained by solving specialized path flow and arc flow LP models. At the root node, we apply the first and second strategies after solving the LP relaxation.

NF-F uses the branching scheme of Section 3.6, which is particularly effective when the incumbent is still not optimal. The tree is limited to K levels. At each level, the left branch is obtained by a large elimination of arcs and is directly solved as an arc flow

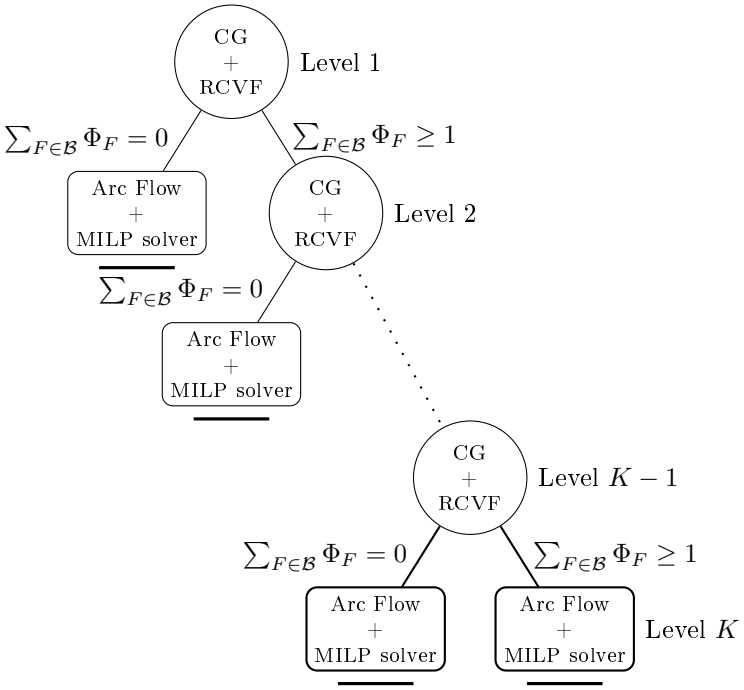

Figure 3.1: Example of a branching tree with K levels.

model by a MILP solver. Although the aim is to provide relatively easier problems in the left branch, in some cases these problems can be small but still hard enough and consume most of the overall time limit. However, no additional stopping criterion is used to deal with such cases. In the right branch, the linear relaxation with an additional branching constraint is solved, followed by the application of the first RCVF strategy. The right branch is branched again in the first $K-1$ levels or solved as an arc flow model by a MILP solver in the last level. In this last node, we apply the third (and most expensive) RCVF strategy before invoking the MILP solver. The tree is explored by breadth-first search, by prioritizing left branches, and no parallelism is implemented.

3.4 On the Solution of the Linear Relaxation

In hard instances of strongly $\mathcal{NP}$-hard problems, the network underlying a DWM is usually huge, and efficient methods for the associated network flow models typically relies on CG algorithms. In such algorithms, the LP relaxation with a restricted set of columns, called *restricted master problem* (RMP), is iteratively solved. At each iteration, an oracle solves the pricing problem to generate non-basic columns with negative reduced cost. The algorithm halts if no such column can be found. For a deeper discussion, see, e.g., [112, 218].

This section describes the solution of the LP relaxation of path flow and arc flow models in NF-F. We use column(-and-row) generation, and the dual plays a key role. For

that, consider the dual LP relaxation of (4.6)–(4.8):

$$\max\ b^\intercal \beta, \tag{3.9}$$
$$\text{s.t.:}\ a_p{}^\intercal \beta \leq c_p, \qquad \forall p \in \mathcal{P}, \tag{3.10}$$
$$\beta \in \mathbb{R}^m_+, \tag{3.11}$$

where β are the dual variables associated with constraints (4.7). The following is the dual LP relaxation of (3.4)–(4.4):

$$\max\ b^\intercal \beta, \tag{3.12}$$
$$\text{s.t.:}\ -\alpha_u + \alpha_v + a_{(u,v)}{}^\intercal \beta \leq c_{(u,v)}, \qquad \forall (u,v) \in \mathcal{A}, \tag{3.13}$$
$$\alpha_{v^+} - \alpha_{v^-} \leq 0, \tag{3.14}$$
$$\alpha_u \in \mathbb{R}, \qquad \forall u \in \mathcal{V}, \tag{3.15}$$
$$\beta \in \mathbb{R}^m_+, \tag{3.16}$$

where α and β are the dual variables associated with flow conservation constraints (3.5) and side constraints (4.3), respectively.

3.4.1 A Dual Correspondence

Recently, de Lima et al. [101] showed how arc flow models provide a richer description of the dual space when compared to path flow models, which results in less degeneracy in the LP solution. The proof is based on the fact that dual constraints of a path flow model can be obtained by aggregating dual constraints of the equivalent arc flow model. Here, we further exploit this fact to present a correspondence between the dual space of these models.

Lemma 1. *Given a feasible solution* $\langle \overline{\alpha}, \overline{\beta} \rangle$ *of* (4.9)–(4.13)*, it holds that* $\overline{\beta}$ *is also feasible for* (4.16)–(4.18)*.*

Proof. The proof follows by showing that $\overline{\beta}$ satisfies constraints (4.17), i.e., $a_p^\intercal \overline{\beta} \leq c_p$, for all $p \in \mathcal{P}$. Since $\langle \overline{\alpha}, \overline{\beta} \rangle$ is dual-feasible for the arc flow model, then by constraints (4.10) it holds that $\overline{\alpha}_v - \overline{\alpha}_u + a_{(u,v)}{}^\intercal \overline{\beta} \leq c_{(u,v)}$, for every arc $(u,v) \in \mathcal{A}$, and by constraints (4.11) it holds that $\overline{\alpha}_{v^+} - \overline{\alpha}_{v^-} \leq 0$. Then, it follows that, for each $p \in \mathcal{P}$, $c_p = \sum_{(u,v)\in\mathcal{A}_p} c_{(u,v)} \geq \sum_{(u,v)\in\mathcal{A}_p} (\overline{\alpha}_v - \overline{\alpha}_u + a_{(u,v)}{}^\intercal \overline{\beta}) = \overline{\alpha}_{v^-} - \overline{\alpha}_{v^+} + \sum_{(u,v)\in\mathcal{A}_p} a_{(u,v)}{}^\intercal \overline{\beta} \geq \sum_{(u,v)\in\mathcal{A}_p} a_{(u,v)}{}^\intercal \overline{\beta} = a_p{}^\intercal \overline{\beta}$. □

Lemma 2. *Given a feasible solution* $\overline{\beta}$ *of* (4.16)–(4.18)*, there exist* $\overline{\alpha} \in \mathbb{R}^{|\mathcal{V}|}$ *such that* $\langle \overline{\alpha}, \overline{\beta} \rangle$ *is feasible for* (4.9)–(4.13)*.*

Proof. Let $\overline{\alpha}$ be defined by the following recursion:

$$\overline{\alpha}_v = \begin{cases} \min\{\overline{\alpha}_u + (c_{(u,v)} - a_{(u,v)}{}^\intercal \overline{\beta}) : (u,v) \in \mathcal{A}\}, & \text{if } v \neq v^+, \\ 0, & \text{if } v = v^+. \end{cases} \tag{3.17}$$

In this definition, $\overline{\alpha}_v$ represents the cost of a shortest path from v^+ to v, where the cost of each arc $(u,v) \in \mathcal{A}$ is given by $(c_{(u,v)} - a_{(u,v)}{}^\intercal \overline{\beta})$. The proof follows by showing that $\overline{\alpha}$ and $\overline{\beta}$ satisfy the dual constraints (4.10) and (4.11). Constraints (4.10) are satisfied, for every arc $(u,v) \in \mathcal{A}$, because, by the definition of $\overline{\alpha}_v$, it holds that $\overline{\alpha}_v \leq \overline{\alpha}_u + (c_{(u,v)} - a_{(u,v)}{}^\intercal \overline{\beta})$, which directly implies that $\overline{\alpha}_v - \overline{\alpha}_u + a_{(u,v)}{}^\intercal \overline{\beta} \leq c_{(u,v)}$. To show that constraint (4.11) is satisfied, we first highlight that, since $\overline{\alpha}_{v^-}$ corresponds to the cost of a shortest path from v^+ to v^-, then $\overline{\alpha}_{v^-} = \min\{c_p - a_p{}^\intercal \overline{\beta} : p \in \mathcal{P}\}$. Since $\overline{\beta}$ is dual-feasible for the path flow model, then it follows by constraints (4.17) that $c_p - a_p{}^\intercal \overline{\beta} \geq 0$, for every $p \in \mathcal{P}$, which directly implies that $\overline{\alpha}_{v^-} \geq 0$. Then, since $\overline{\alpha}_{v^+} = 0$ and $\overline{\alpha}_{v^-} \geq 0$, it follows that $\overline{\alpha}_{v^+} - \overline{\alpha}_{v^-} \leq 0$. □

By combining Lemmas 1 and 2, the following result is derived:

Theorem 2 (Dual Correspondence)**.** *There is a one-to-many correspondence between the solution space of* (4.16)–(4.18) *and* (4.9)–(4.13)*, respectively, in which variables* β *preserve the same solution on both sides of the mapping.*

Theorem 2 guarantees that dual solutions $\overline{\beta}$ of either an arc flow or a path flow model may be used without distinction in the methods presented next.

3.4.2 Computing the Minimum Reduced Cost of Arcs

Let us define the minimum reduced cost of an arc as follows:

Definition 4. *Given a dual solution* $\overline{\beta} \in \mathbb{R}^m_+$*, the* minimum reduced cost of an arc $(u,v) \in \mathcal{A}$ *is the minimum reduced cost of a path among all paths that contain* (u,v)*, and is given by* $\overline{c}_{(u,v)} = \min\{c_p - a_p{}^\intercal \overline{\beta} : p \in \mathcal{P}_{(u,v)}\}$.

The minimum reduced costs of the arcs are used in the oracle presented in Section 3.4.4 and in the RCVF strategies presented in Section 3.5. Irnich et al. [183] showed how to efficiently compute $\overline{c}_{(u,v)}$, for all $(u,v) \in \mathcal{A}$, by bidirectional Dynamic Programming (DP). First, one has to compute the values $\overline{c}^+_v$ and $\overline{c}^-_v$, for each node $v \in \mathcal{V}$, which represent the minimum reduced cost associated with all paths from v^+ to v and from v to v^-, respectively. The fact that $\mathcal{N}$ is acyclic allows one to compute $\overline{c}^+_v$ and $\overline{c}^-_v$, for each $v \in \mathcal{V}$, by means of the following recursions:

$$\overline{c}^+_v = \begin{cases} \min\{\overline{c}^+_u + (c_{(u,v)} - a_{(u,v)}{}^\intercal \overline{\beta}) : (u,v) \in \mathcal{A}\}, & \text{if } v \neq v^+, \\ 0, & \text{if } v = v^+, \end{cases}$$

$$\overline{c}^-_v = \begin{cases} \min\{\overline{c}^-_w + (c_{(v,w)} - a_{(v,w)}{}^\intercal \overline{\beta}) : (v,w) \in \mathcal{A}\}, & \text{if } v \neq v^-, \\ 0, & \text{if } v = v^-. \end{cases}$$

Efficient implementations of a labelling DP algorithm based on the two recursions allows us to simultaneously compute $\overline{c}^+_v$ and $\overline{c}^-_v$, for all $v \in \mathcal{V}$, in $O(\mathcal{A})$ time complexity. Then, the minimum reduced cost of an arc (u,v) is simply given by $\overline{c}_{(u,v)} = \overline{c}^+_u + (c_{(u,v)} - a_{(u,v)}{}^\intercal \overline{\beta}) + \overline{c}^-_v$. The recursive structure also allows us to efficiently retrieve a path associated with $\overline{c}_{(u,v)}$ after the execution of the DP algorithm.

3.4.3 Path-Based Pricing in Arc Flow

In CG algorithms for (4.6)–(4.8), given a dual solution $\overline{\beta}$, the pricing problem

$$\min\{c_p - a_p{}^\intercal \overline{\beta} : p \in \mathcal{P}\} \tag{3.18}$$

can be solved as a shortest path problem on $\mathcal{N}$, with arc costs $c_{(u,v)} - a_{(u,v)}{}^\intercal \overline{\beta}$, for $(u,v) \in \mathcal{A}$. As networks from a DWM are acyclic, a shortest path can be found in $O(|\mathcal{A}|)$ by a topological ordering of the nodes (see, e.g., [1]). In arc flow models, to search for a negative reduced cost variable, one can solve

$$\min\{c_{(u,v)} - (-\overline{\alpha}_u + \overline{\alpha}_v + a_{(u,v)}{}^\intercal \overline{\beta}) : (u,v) \in \mathcal{A}\}, \tag{3.19}$$

which generates a single arc. In practice, however, generating complete paths (each to be decomposed in a set of arcs) may significantly improve convergence time (see [314]). In what follows, we formalize the correctness of this approach.

Proposition 1. *Let $\overline{\alpha} \in \mathbb{R}^{|\mathcal{V}|}$ and $\overline{\beta} \in \mathbb{R}^m_+$ be a solution of (4.9)–(4.13), possibly infeasible but satisfying (4.11). Then, any path $p \in \mathcal{P}$ with negative reduced cost contains at least one arc $(u,v) \in \mathcal{A}_p$ with negative reduced cost.*

Proof. Let $p \in \mathcal{P}$ be a path with reduced cost $c_p - a_p{}^\intercal \overline{\beta} < 0$. We prove that at least one arc in $\mathcal{A}_p$ has negative reduced cost by showing that the sum of the reduced costs of arcs in $\mathcal{A}_p$, given by $\sum_{(u,v)\in\mathcal{A}_p}(c_{(u,v)} - (-\overline{\alpha}_u + \overline{\alpha}_v + a_{(u,v)}{}^\intercal \overline{\beta}))$, is negative. By properly canceling $\overline{\alpha}$ terms from intermediate nodes in p, the sum of reduced costs is equal to $\sum_{(u,v)\in\mathcal{A}_p}(c_{(u,v)} - a_{(u,v)}{}^\intercal \overline{\beta}) + (\overline{\alpha}_{v^+} - \overline{\alpha}_{v^-}) = (c_p - a_p^\intercal \overline{\beta}) + (\overline{\alpha}_{v^+} - \overline{\alpha}_{v^-})$. Since we suppose that $c_p - a_p{}^\intercal \overline{\beta} < 0$ and, by (4.11), $(\overline{\alpha}_{v^+} - \overline{\alpha}_{v^-}) < 0$, it follows that the sum of reduced costs of arcs in $\mathcal{A}_p$ is negative, and, therefore, at least one arc in $\mathcal{A}_p$ has negative reduced cost. □

Proposition 1 ensures that in the solution of an arc flow model relaxation by CG, if the pricing problem is solved as (3.18) and all arcs in the resulting path are introduced in the RMP, then the CG algorithm only halts when all paths in $\mathcal{P}$ have non-negative reduced cost. Hence, the $\overline{\beta}$ solution at the end of the CG algorithm is feasible for (4.16)–(4.18), and, by relying on Theorem 2, there exists an $\overline{\alpha}$, computed by (3.17), such that $\langle \overline{\alpha}, \overline{\beta} \rangle$ is feasible for (4.9)–(4.13).

3.4.4 Column(-and-Row) Generation Algorithm

NF-F adopts a CG algorithm for path flow models and a column-and-row generation algorithm for arc flow models. For path flow, all rows are included in the RMP from the beginning. For arc flow, instead, we generate flow conservation constraints on demand, as proposed in [314], by adding a constraint (3.5) only when the corresponding vertex v first appears as head or tail of an arc associated with an RMP variable.

Both algorithms include in the initial RMP base artificial variables with sufficiently high objective cost, so as to avoid infeasibility. In addition, they reuse the optimal basis of previous LP relaxations, when available.

In many problems, generating a single column per pricing iteration may lead to a slow convergence of the algorithm. In network flow models, it may be preferable to generate multiple paths at each iteration. Therefore, we implemented an oracle that generates multiple paths with negative reduced cost (if any exists), and, by relying on the discussion in Section 3.4.3, can be used to solve the pricing of both path flow and arc flow models.

The oracle that we developed begins by computing $\overline{c}_{(u,v)}$, for each $(u,v) \in \mathcal{A}$, by using the method described in Section 3.4.2. Then, for each row $k = 1, \ldots, m$ in (4.7) or in (4.3), the algorithm selects an arc (u,v) with minimum $\overline{c}_{(u,v)}$ among the arcs that cover k (i.e., that have non-zero coefficient on row k), and, by using the DP structure, it generates a path in $\mathcal{P}_{(u,v)}$ of minimum reduced cost. At the end, the oracle has generated, for each row k, a path of minimum reduced cost that covers k. For instance, the algorithm generates the columns of minimum reduced cost, such that each item (in packing problems) or each client (in routing problems) is covered by at least one of such columns. Notice that different rows may lead to the same column, so repeated columns are discarded. The overall algorithm has $O(m\zeta + |\mathcal{A}|)$ time complexity, where ζ is the maximum length of a path, and it is preferable to use it when the matrix of (4.7) is sparse, so that more different columns are generated.

3.4.5 Dealing with Dual Infeasibility

To apply RCVF (Section 3.5) and to prune nodes by a dual bound in the search of an integer optimal solution, a dual-feasible solution is needed. Most state-of-the-art LP solvers work within the limited precision of floating-point arithmetic and produce solutions that are feasible within a small margin of error. In particular, the LP solver that we use (Gurobi 9.1.1) assumes a dual solution of a minimization problem to be feasible if its minimum reduced cost is greater than $-\epsilon$, where ϵ is at minimum 10^{-9}. Then, when the minimum reduced cost lies between -10^{-9} and 0, the dual solution is still infeasible but cannot be improved by CG because the solver may not include new columns in the RMP base. This issue could be solved with the use of an LP solver with exact precision, but at the cost of a significant efficiency loss. Instead, we implemented a method that attempts transforming a dual-infeasible solution (with a minimum reduced cost between -10^{-9} and 0) into a feasible one.

A constraint in (4.7) is defined as a *covering* constraint if all of its left-hand-side coefficients are non-negative and as a *packing* constraint if all of its left-hand-side coefficients are non-positive. The following result shows how to efficiently derive dual-feasible solutions for a large class of models:

Proposition 2. *Consider a path flow model* (4.6)-(4.8) *where the first* $1, \ldots, m'$ *constraints are covering constraints and the last* $m'+1, \ldots, m$ *constraints are packing constraints. Let* $\epsilon > 0$, *and* $\overline{\beta} \in \mathbb{R}^m_+$ *be an (infeasible) dual solution with minimum reduced cost of a path between* $-\epsilon$ *and* 0. *If* c_p *is a multiple of* ϵ, *for each* $p \in \mathcal{P}$, *then,* $\overline{\beta}' = (\epsilon\lfloor\epsilon^{-1}\overline{\beta}_1\rfloor, \ldots, \epsilon\lfloor\epsilon^{-1}\overline{\beta}_{m'}\rfloor, \epsilon\lceil\epsilon^{-1}\overline{\beta}_{m'+1}\rceil, \ldots, \epsilon\lceil\epsilon^{-1}\overline{\beta}_m\rceil)$ *is a dual-feasible solution.*

Proof. Let $p \in \mathcal{P}$ be an arbitrary path and $\overline{c}_p(\overline{\beta})$ and $\overline{c}_p(\overline{\beta}')$ be the reduced cost of p associated with the dual solutions $\overline{\beta}$ and $\overline{\beta}'$, respectively. We prove that $\overline{\beta}'$ is dual-feasible by showing that $\overline{c}_p(\overline{\beta}') \geq 0$. By considering that $a_{pk} \geq 0$ and $\overline{\beta}'_k \leq \overline{\beta}_k$, for each

$k = 1, \ldots, m'$, and that $a_{pk} \leq 0$ and $\overline{\beta}'_k \geq \overline{\beta}_k$, for each $k = m'+1, \ldots, m$, it is easy to check that $\overline{c}_p(\overline{\beta}') \geq \overline{c}_p(\overline{\beta})$, which directly implies that $\overline{c}_p(\overline{\beta}') > -\epsilon$, since $\overline{c}_p(\overline{\beta}) > -\epsilon$. Now, considering that c_p is a multiple of ϵ and that a_{pk} is integer, for all $k = 1, \ldots, m$, we have that $\overline{c}_p(\overline{\beta}')$ is also a multiple of ϵ. Finally, the fact that $\overline{c}_p(\overline{\beta}')$ is a multiple of ϵ and is strictly greater than $-\epsilon$ implies that $\overline{c}_p(\overline{\beta}') \geq 0$. □

Proposition 2 is a generalization of the technique by Held et al. [164] to obtain safe dual-bounds for the vertex coloring problem. We consider $\epsilon = 10^{-9}$, which is always a divisor of the integer coefficients c_p. Hence, in our algorithm, the proposition can always be applied to models where constraints are only of covering and/or packing type. By relying on Theorem 2, Proposition 2 can also be applied in the case of arc flow models.

When (4.7) has additional constraints which are not covering nor packing, Proposition 2 cannot be applied, and we use a heuristic to try to obtain a dual-feasible solution by treating these additional constraints as if they were of covering type, and then applying the proposition to obtain β'. In theory, the heuristic procedure may fail to obtain a dual-feasible solution, and in such a case NF-F would continue the optimization without the ability to perform RCVF or prune the current node by dual bound. In practice, in our computational experiments the only model which did not follow the hypothesis of Proposition 2 is the model for the two-stage guillotine cutting stock problem (Section 3.7.2), but for this problem the heuristic procedure was always sufficient to provide a dual-feasible solution at the end of the CG.

3.5 Variable-Fixing Based on Reduced Costs

Reduced-cost variable-fixing is a well-known technique used to filter the domain of integer variables in general MILP models (see, e.g., [160] and page 389 of [250]). Irnich et al. [183] showed how the dual solution of a path flow model can be used in an RCVF algorithm to remove arcs. The overall idea is:

Variable-Fixing Algorithm. *Given a primal bound z_{ub} and a dual-feasible solution $\overline{\beta}$ of objective value $z_{lb} = b^\intercal \overline{\beta}$, first compute the minimum reduced cost $\overline{c}_{(u,v)}$ associated with each arc $(u,v) \in \mathcal{A}$ (e.g., as described in Section 3.4.2), and then remove from $\mathcal{N}$ every arc $(u,v) \in \mathcal{A}$ such that $\overline{c}_{(u,v)} > z_{ub} - z_{lb} - 1$.*

An equivalent approach has been implemented by Bergman et al. [36] in the general context of Lagrangian bounds for multivalued decision diagrams.

Different dual solutions may correspond to different reduced costs, and this directly affects the effectiveness of the RCVF. For instance, let $\theta_{(u,v)} = c_{(u,v)} - (-\alpha_u + \alpha_v + a_{(u,v)}{}^\intercal \beta)$ denote the reduced cost of $(u,v) \in \mathcal{A}$ in model (4.9)–(4.13). A dual solution which allows the removal of (u,v) by RCVF is one that satisfies $b^\intercal \beta + \theta_{(u,v)} \geq z_{ub}$. If such a solution exists, it can be obtained by solving the dual arc flow model with a modified objective:

$$\max\{b^\intercal \beta + \theta_{(u,v)} : (4.9), (4.10), (4.11), (4.12), (4.13)\}. \tag{3.20}$$

However, optimizing a model tailored for each arc can be very time consuming in practice. In the general context of non-convex mixed-integer nonlinear programming, a similar

concern has been addressed in optimization-based bound tightening (see, e.g., [151]). In the particular context of MILP, Bajgiran et al. [13] proposed a method to maximize the number of variables fixed by RCVF, by solving a single MILP model derived from an extension of the dual linear relaxation of the original model. This MILP model has an additional binary variable for each original variable, indicating whether the resulting dual solution is able to eliminate the associated original variable by RCVF. In our case, this method is generally impractical, because we are concerned with models with a huge number of variables that must rely on CG.

Here, we propose two strategies to search for dual solutions that may improve the RCVF effectiveness, motivated by:

Remark 1. *The removal of arcs with a positive primal value in the current optimal basis leads to a basis-violation, which strengthens the linear relaxation.*

In practice, this strengthening usually increases the effectiveness of further variable-fixing. However, as shown in the following section, such arc-removal by RCVF can only be obtained by relying on sub-optimal dual solutions.

3.5.1 Sub-Optimal Dual Solutions in Variable-Fixing

Sub-optimal dual solutions often produce a greater effectiveness in RCVF. This counter-intuitive fact was already observed almost two decades ago, for instance, by Sellmann [293] in the context of constraint programming-based Lagrangian relaxations. Here we extend this observation by the following:

Proposition 3. *Let $(u,v) \in \mathcal{A}$ be any arc with $\overline{\varphi}_{(u,v)} > 0$ in a given primal-optimal solution $\overline{\varphi}$. Then, there does not exist any dual-optimal solution $\overline{\beta}$ such that (u,v) can be removed by the variable-fixing algorithm based on $\overline{\beta}$ and a primal bound $z_{ub} \geq \lceil b^{\intercal}\overline{\beta} \rceil + 1$.*

Proof. We assume that $z_{ub} \geq \lceil b^{\intercal}\overline{\beta} \rceil + 1$, since in the trivial case in which $z_{ub} = \lceil b^{\intercal}\overline{\beta} \rceil$, optimality is already proven. By the classical complementary slackness theorem, it follows that if $\overline{\varphi}$ and $\overline{\beta}$ are, respectively, primal-optimal and dual-optimal solutions, then, either $\overline{\varphi}_{(u,v)} = 0$, or $\overline{c}_{(u,v)} = 0$. Then, by supposing that $\overline{\varphi}_{(u,v)} > 0$, we have that $\overline{c}_{(u,v)} = 0$. Consequently, $b^{\intercal}\overline{\beta} + \overline{c}_{(u,v)} = b^{\intercal}\overline{\beta} \leq \lceil b^{\intercal}\overline{\beta} \rceil \leq z_{ub} - 1$, which implies that (u,v) cannot be removed by the variable-fixing algorithm based on $\overline{\beta}$ and z_{ub}. □

By disregarding the trivial case in which $z_{ub} = \lceil b^{\intercal}\overline{\beta} \rceil$, Proposition 3 ensures that, to remove arcs with a positive primal value in the current optimal basis, we need a dual solution that is sub-optimal. A generalization of Proposition 3 to the case of general MILP models (not restricted to network flow) is straightforward.

3.5.2 Variable-Fixing Strategies

In this section, we present the three RCVF strategies adopted in NF-F. In a preliminary version of this paper [104], we used a different strategy to obtain alternative dual solutions. This strategy is not presented here, because in our experiments it was always outperformed by the second and third strategies presented below. The three adopted strategies differ

mainly in the way in which a dual solution is computed. In all strategies, each dual solution obtained is used as input to the variable-fixing algorithm.

First strategy. The traditional strategy based on a dual solution obtained at the end of CG for the original linear relaxation.

Second strategy. We obtain dual solutions by solving a dual path flow model in which, motivated by Proposition 3, the solution is allowed to be sub-optimal and the reduced-cost of paths with positive value on a primal-optimal solution are included in the objective. For that, we first consider the primal-optimal solution $\overline{\lambda}$ in terms of path flow variables. Then, the dual path flow model to be solved is given by:

$$\max\ b^\intercal\beta + \sum_{p\in\mathcal{P}:\overline{\lambda}_p>0} \theta_p, \tag{3.21}$$

$$\text{s.t.: } {a_p}^\intercal\beta + \theta_p = c_p, \qquad \forall p \in \mathcal{P}, \overline{\lambda}_p > 0, \tag{3.22}$$

$${a_p}^\intercal\beta \leq c_p, \qquad \forall p \in \mathcal{P}, \overline{\lambda}_p = 0, \tag{3.23}$$

$$b^\intercal\beta \geq z_{lb} - \epsilon, \tag{3.24}$$

$$\beta \in \mathbb{R}^m_+, \tag{3.25}$$

$$\theta_p \in \mathbb{R}_+, \qquad \forall p \in \mathcal{P}, \overline{\lambda}_p > 0, \tag{3.26}$$

in which each θ_p explicitly models the reduced cost of variable λ_p, with $\overline{\lambda}_p > 0$. Constraints (3.24) allow the resulting dual solution to be sub-optimal for the original linear relaxation, but limited by a given $\epsilon > 0$. Model (3.21)–(3.26) is solved by row generation (i.e., CG of its primal), by considering the primal variables associated with (3.22) and (3.24) already in the initial RMP. Notice that, when the variable-fixing successfully removes a basic arc, model (3.21)–(3.26) possibly provides a different dual solution if solved again. Then, we iteratively solve (3.21)–(3.24) and use the resulting dual solution in the variable-fixing algorithm, until no more arcs are removed.

Third strategy. This is a refinement (motivated by Remark 1) of the expensive method that solves a model (3.20) tailored for each arc. It considers the primal-optimal solution $\overline{\varphi}$ in terms of arc flow variables and solves a restricted sequence of models (3.20), each tailored for an arc $(u, v) \in \mathcal{A}$, in which $0 < \overline{\varphi}_{(u,v)} < 1$. Since we expect an arc (u, v) with $\overline{\varphi}_{(u,v)}$ closer to 0 to be more likely to be removed by variable-fixing, we follow a non-decreasing order of $\overline{\varphi}_{(u,v)}$ to build the sequence of models to be solved. Each model is solved by column-and-row generation, producing a dual-feasible solution, which in turn is used as input to the variable-fixing algorithm. Although we solve only a restricted set of dual models, this strategy can still be expensive, and stopping criteria should be considered. In NF-F, the strategy halts once the remaining number of arcs is smaller than $50m$ (where m is the number of side constraints). Despite the high computational cost, this strategy can be very effective for cases in which the incumbent solution is optimal and the challenge is to prove optimality. Hence, we only use it on the most expensive node in our branch-and-price tree, i.e., the right branch of the last level.

3.6 Branching Scheme

A major issue in B&P is that efficient branching schemes are not always robust. A number of works propose general branching schemes that minimize the impact on the pricing problem and at the same time help convergence to optimality (see, e.g., [323]). However, not many B&P schemes exploit the network flow representation of a DWM (as done, e.g., in [9]). In general, any constraint $\sum_{(u,v)\in\mathcal{A}} a'_{(u,v)}\varphi_{(u,v)} \geq b'$ based on a linear combination of arc flow variables impacts on a pricing solved as a shortest path problem by simply incrementing ${a'_{(u,v)}}^{\intercal}\beta'$ to the cost of each arc $(u,v) \in \mathcal{A}$, where β' is the dual solution related to the constraint. A consequent result is that branching rules based solely on arc flow variables are robust.

Based on the primal correspondence given by the flow decomposition theorem, any arc flow variable $\varphi_{(u,v)}$ can be represented as a sum $\sum_{p\in\mathcal{P}_{(u,v)}} \lambda_p$ of variables from the equivalent path flow model. Consequently, any arc flow constraint $\sum_{(u,v)\in\mathcal{A}} a'_{(u,v)}\varphi_{(u,v)} \geq b'$ can be directly represented as a path flow constraint $\sum_{(u,v)\in\mathcal{A}} a'_{(u,v)} \sum_{p\in\mathcal{P}_{(u,v)}} \lambda_p \geq b'$. On the other hand, it is not always possible to rewrite a linear constraint based on path flow variables in terms of arc flow variables. This further motivates branching rules based on arc flow variables, as the resulting branching constraints can be easily handled by both path flow and arc flow models.

3.6.1 Proposed Branching Scheme

The branching scheme we propose is based on sets of arcs and exploits the potential of a general MILP solver in finding optimal solutions of small/medium-sized models. Initially, arcs that share mutual characteristics are grouped into subsets. It then considers arcs in all subsets having null flow in the optimal linear solution of a model, and either eliminates all of them or forces at least one of them to be in the solution.

We define an *arc family* $\mathcal{F} \subset 2^{\mathcal{A}}$ as an arbitrary set of mutually disjoint subsets of $\mathcal{A}$. For each $F \in \mathcal{F}$, let variable $\Phi_F = \sum_{(u,v)\in F} \varphi_{(u,v)}$ represent the aggregated sum of arc flow variables associated with arcs in F. Given a primal solution $\overline{\varphi}$ in terms of arc flow variables, we represent as $\overline{\Phi}_F = \sum_{(u,v)\in F} \overline{\varphi}_{(u,v)}$ the cumulated sum of the solution values of arcs in F.

The variable selection considers all variables related to the set of arcs $\mathcal{B} = \{(u,v) \in F \in \mathcal{F} : \overline{\Phi}_F = 0\}$, which are used to create two branches. In the *left branch*, we implicitly consider the branching constraint

$$\sum_{(u,v)\in\mathcal{B}} \varphi_{(u,v)} = 0 \tag{3.27}$$

by removing all arcs in $\mathcal{B}$ from $\mathcal{N}$. In the *right branch* we add the constraint

$$\sum_{(u,v)\in\mathcal{B}} \varphi_{(u,v)} \geq 1 \tag{3.28}$$

to the model, implying that at least one arc in $\mathcal{B}$ must be in the solution. Depending on the definition of $\mathcal{F}$, the left branch is expected to lead to a great reduction in the size

of the network, while keeping variables that should provide good feasible solutions. The reduced problem may be solved by an alternative method. In particular, we conclude this branch by solving the residual arc flow model by a MILP solver. The domain reduction in the right branch may be weaker, but the branching constraint may behave as a cutting plane, hopefully improving the optimal dual bound. In fact, although no arcs are explicitly removed from the network in the right branch, the strengthening in the relaxation may improve the effectiveness of RCVF.

In Section 3.7, we present the arc families used in our applications to C&P problems. Further examples for other applications can be found in [104].

3.6.2 Lifting the Right-Branch Constraint

Constraint (3.28) imposes that at least one arc in $\mathcal{B}$ must be in the solution. The network structure can be exploited to determine redundant arcs in $\mathcal{B}$ that can be in a solution only if other arcs in $\mathcal{B}$ are also in the solution. The use of any of such redundant arcs implies that the left-hand side of constraint (3.28) is at least 2. Thus, even if a redundant arc is removed from $\mathcal{B}$ the set of integer solutions that are feasible w.r.t. (3.28) does not change. On the other hand, such removal may violate fractional solutions, and this strengthens the relaxation of the resulting model.

Once a redundant arc is removed from $\mathcal{B}$, the list of remaining redundant arcs must be updated, since some redundant arcs may become non-redundant. In this way, a greedy removal of redundant arcs is not necessarily optimal w.r.t the number of arcs removed from $\mathcal{B}$. For that, we implemented a heuristic DP approach with $O(|\mathcal{A}|)$ time complexity to maximize the number of redundant arcs removed. The algorithm iterates over a topological ordering of $\mathcal{V}$. At the iteration of node u^*, it computes whether there exists any path from v^+ to u^* that does not have an arc in $\mathcal{B}$. If true, it proceeds to the next iteration, otherwise, it sets $\mathcal{B} \leftarrow \mathcal{B} \setminus \{(u^*, v) \in \mathcal{B}\}$. To further remove redundant arcs from $\mathcal{B}$, we also apply the algorithm in the reversed network, obtained by inverting the direction of the arcs.

3.7 Applications to Cutting and Packing Problems

We apply NF-F to four C&P problems that allow pseudo-polynomial arc flow models with strong relaxations. In the first three applications, we considered network flow models from the literature, which for conciseness are not explicitly reported here. However, since the arc flow model for the fourth application is new, we give its coefficients to model (4.6)–(4.8) and (3.4)–(4.5).

3.7.1 Cutting Stock Problem

In the CSP, we are given an unlimited number of stock rolls of length $W \in \mathbb{Z}_+$ and a set I of items, each $i \in I$ associated with a width $w_i \in \mathbb{Z}_+$ and a demand $d_i \in \mathbb{Z}_+$. The objective is to cut the minimum number of stock rolls to obtain all demands. An

equivalent problem is the bin packing problem (BPP), where $d_i = 1$, for all $i \in I$. We refer to [109] for a recent survey.

The classical pattern-based model in [149, 150] is a path flow model derived from a DW decomposition of the textbook CSP model (see, e.g., [226]). The underlying network of the model in [149, 150] is a DP network of an unbounded knapsack problem. The equivalent arc flow model was first solved in [314]. The network of this model discretizes the stock roll into W unitary positions, and each node in $\mathcal{V} \subseteq \{v : v = 0, \ldots, W\}$ is associated with a position, where $v^+ = 0$ and $v^- = W$. Each item $i \in I$ has a set $\mathcal{A}_i \subseteq \{(v, v + w_i) : v \in \mathcal{V}, v \leq W - w_i\}$ of arcs, where each arc $(v, v + w_i) \in \mathcal{A}_i$ represents the cut of item i starting from v in a stock roll. An additional set of arcs $\mathcal{A}^- = \{(v, v^-) : v \in \mathcal{V}\}$ represents waste portions of a stock roll. The overall set of arcs is $\mathcal{A} = \cup_{i \in I} \mathcal{A}_i \cup \mathcal{A}^-$.

A subset relation is used in the definition of $\mathcal{V}$ and $\mathcal{A}_i$ ($i \in I$) since not all positions are necessary to solve the model exactly. Indeed, [314] proposed reduction criteria to remove redundant arcs by considering that items can always be cut following a non-increasing width ordering. Later, Côté and Iori [93] proposed the *meet-in-the-middle* (MIM) patterns, which allowed to produce significantly smaller networks. Based on the following remark, we developed a technique that further reduces the network from [314] and may lead to smaller networks than the ones resulting from the MIM patterns.

Remark 2. *Given a CSP instance, let $\overline{W} \in \mathbb{Z}_+$ be a value ensuring that there exists an optimal solution where the maximum waste of a single stock roll is at most $\overline{W}$. Then, all arcs contained only in paths associated with cutting patterns with a waste larger than $\overline{W}$ can be removed from the network.*

A straightforward way to compute $\overline{W}$ considers that in any CSP solution with K stock rolls the maximum waste on each roll is at most $KW - \sum_{i \in I} w_i d_i$. All arcs that only lead to cutting patterns with a waste larger than $\overline{W}$ are computed by a back propagation in the network, similarly to the method by Trick [307] to propagate knapsack networks in constraint programming. The computation of the *waste-limited network* is given in Algorithm 1. By considering items ordered by non-increasing w_i, lines 1 to 7 create the standard network in [314], and lines 8 to 15 impose the reduction from Remark 2.

We consider two general classes of arc families. The first arc family

$$\mathcal{F}_k^a = \{F_n = \{(u, v) \in \mathcal{A} : u \in \{nk, \ldots, nk + (n-1)\}\} : n = 0, \ldots, \lceil W/k \rceil - 1\}$$

considers the sets of nodes sequentially partitioned in $\lceil W/k \rceil - 1$ parts of up to k nodes each. Then, each set F_n represents the n-th part, and it contains all arcs whose tail lies in such part. In general, larger values of k provide a conservative reduction in the left branch. The second arc family

$$\mathcal{F}_k^b = \{F_{in} = \{(u, v) \in \mathcal{A}_i : u \in \{nW/k, \ldots, (n+1)W/k - 1\}\} : n = 0, \ldots, k-1\}$$

considers the set of arcs of each item partitioned into k parts, following an increasing order of the nodes. Each set F_{in} contains each arc in $\mathcal{A}_i$ whose tail lies in the n-th part. For instance, when $k = 2$, $\mathcal{F}_2^b$ represents a partition of the arcs of each item into two

Algorithm 1: WASTELIMITEDNETWORKCSP

Input: W, I, and $\overline{W}$

$\mathcal{V}^+ \leftarrow \{0\}$
for $i = 1, \ldots, |I|$ **do**
  for $copy = 1, \ldots, d_i$ **do**
    for $v \in \mathcal{V}^+$ *in decreasing order* **do**
      if $v + w_i \leq W$ **then**
        $\mathcal{A}_i \leftarrow \mathcal{A}_i \cup \{(v, v + w_i)\}$
        $\mathcal{V}^+ \leftarrow \mathcal{V}^+ \cup \{v + w_i\}$
$\mathcal{V}^- \leftarrow \{W - \overline{W}, \ldots, W\}$
for $i = |I|, \ldots, 1$ **do**
  $S \leftarrow \{\}$
  for $copy = 1, \ldots, d_i$ **do**
    for $(v, v + w_i) \in \mathcal{A}_i$ *in increasing order of* v **do**
      if $v + w_i \in \mathcal{V}^-$ **then**
        $\mathcal{V}^- \leftarrow \mathcal{V}^- \cup \{v\}$
        $S \leftarrow S \cup \{(v, v + w_i)\}$
  $\mathcal{A}_i \leftarrow S$
$\mathcal{V} \leftarrow \{v : (v, v + w_i) \in \mathcal{A}_i \text{ or } (v - w_i, v) \in \mathcal{A}_i, i \in I\} \cup \{0, W\}$
$\mathcal{A}^- \leftarrow \{(v, W) : v \geq W - \overline{W}, v \in \mathcal{V}\}$
return $\mathcal{V}$, $\mathcal{A}_i$ *(for all* $i \in I$*) and* $\mathcal{A}^-$

parts, each representing either the first half or the second half of a stock roll. If $k = W$, then each part contains up to a single arc. For this family, larger values of k generate in the left branch problems that are smaller, but also less likely to contain good integer solutions.

3.7.2 Two-Stage Guillotine Cutting Stock Problem

In the two-stage guillotine cutting stock problem (2GCSP), we are given an unlimited number of two-dimensional stock sheet with width W and height H, and a set I of two-dimensional items. Each item $i \in I$ has width w_i, height h_i, and demand d_i. The aim is to cut all items from the minimum number of stock sheets, by using two-stage guillotine cuts. The first and second cut stages consist of, respectively, horizontal and vertical cuts parallel to the edges of the stock sheet. To separate items from waste, a third trimming stage is allowed.

The state-of-the-art exact method for the 2GCSP is the arc flow model in [93], which corresponds to the model proposed by Macedo et al. [220] enhanced by the use of the MIM patterns. Let $H^* = \{h_i : i \in I\}$ be the set of all different item heights. The arc flow model has a graph $(\mathcal{V}^1, \mathcal{A}^1)$ representing first-stage cut decisions, in which $\mathcal{V}^1 \subseteq \{v : v = 0, \ldots, H\}$ and $\mathcal{A}^1 \subseteq \{(v, v + h) : v \in \mathcal{V}^1, h \in H^*\}$. There is also a graph $(\mathcal{V}^2_h, \mathcal{A}^2_h)$ representing second-stage cut decisions in each strip of height $h \in H^*$ cut in the first stage, in which $\mathcal{V}^2_h \subseteq \{v : v = 0, \ldots, W\}$ and $\mathcal{A}^2_h \subseteq \{(v, v + w_i) : v = 0, \ldots, W, i \in I, h_i \leq h\}$. There are also source arcs connecting v^+ to 0 in $\mathcal{V}^1$ and $\mathcal{V}^2_h$ (for each $h \in H^*$) and sink arcs connecting each node in $\mathcal{V}^1$ and $\mathcal{V}^2_h$ (for each $h \in H^*$) to v^-. A full description of the

model is reported in [93].

In our experiments for the 2GCSP, we solve the arc flow model in [93] with NF-F by setting $K = 10$ and using the arc family:

$$\begin{aligned}\mathcal{F} =&\{F_v = \{(v, v+h) \in \mathcal{A}_h^1 : h \in H^*\} : v \in \mathcal{V}^1\} \\ &\cup \{F_v = \{(v, v+w_i) \in \mathcal{A}_h^2 : h \in H^*, i \in I, h_i \leq h\} : v \in \cup_{h \in H^*} \mathcal{V}_h^2\},\end{aligned}$$

where the first and second parts correspond to all arcs related to specific first-stage and second-stage cut positions, respectively.

Since the arc flow model for the 2GCSP does not follow the hypothesis of Proposition 2, we must rely on the heuristic method in Section 3.4.5 to convert dual-infeasible solutions into feasible ones, which may fail. We attested that this heuristic conversion always succeeded in the solution of the LP relaxation in all nodes of the branching tree. However, it did not always succeed for the (many) dual-infeasible solutions obtained in the second and third variable-fixing strategies, which were consequently deactivated.

3.7.3 Skiving Stock Problem

In the skiving stock problem (SSP), we are given the minimum width W of a large object and a set I of items, where each item $i \in I$ has a width w_i and a maximum number of copies b_i. The objective is to recompose the items into the maximum number of large objects.

The state-of-the-art exact method for the SSP is the reflect formulation in [232]. In our experiments, we solve the pseudo-polynomial arc flow model in [233]. The network of this model uses a set $\mathcal{V} \subseteq \{v : v = 0, \ldots, W'\}$ of nodes to represent integer linear combinations of the given item widths, where $W' \geq W$ is a sufficiently large value, $v^+ = 0$, and $v^- = W'$. A set $\mathcal{A}_i \subseteq \{(v, v+w_i) : v \in \mathcal{V}, v \leq W' - w_i\}$ of arcs is associated with each $i \in I$, where each arc $(v, v+w_i) \in \mathcal{A}_i$ represents the inclusion of an item i from positions v to $v+w_i$ of a large object. There is also a set of sink arcs $\mathcal{A}^- \subseteq \{(v, v^-) : v \in \mathcal{V}, v \geq W\}$, where each (v, v^-) represents the final composition of a large object with total width v in the solution. The overall set of arcs is $\mathcal{A} = \cup_{i \in I} \mathcal{A}_i \cup \mathcal{A}^-$.

In our experiments, the arc flow model is solved by NF-F based on an arc family equivalent to $\mathcal{F}_{20}^b$ (Section 3.7.1) and with $K = 10$.

3.7.4 Ordered Open-End Bin Packing Problem

In the ordered open-end bin packing problem (OOEBPP), we are given an unlimited number of copies of a one-dimensional bin with capacity W and a sequence $I = (1, \ldots, m)$ of one-dimensional items, where each item $i \in I$ has weight w_i. The aim is to pack all items into the minimum number of bins, by allowing the last item (and only the last item) of the sequence in each bin to exceed the bin capacity.

The state-of-the-art exact method for the OOEBPP is a B&P algorithm for a set-covering formulation (hence a path flow model) by Ceselli and Righini [71]. We use the model in [71] to derive an equivalent arc flow model where the network is based on a

pseudo-polynomial pricing algorithm. This network is similar to the one of the DP-flow model for the CSP (see, e.g., [109]), which is based on the classical DP recursion for the knapsack problem. The set of nodes is given by $\mathcal{V} \subseteq \{(i, W') : i = 1, \ldots, m, W' = 0, \ldots, W-1\} \cup \{v^-\}$, where $v^+ = (1, 0)$. Each item $i = 1, \ldots, m-1$ has a set $\mathcal{A}_i \subseteq \{((i, W'), (i+1, W'+w_i)) : W' = 0, \ldots, W-w_i-1\}$ of arcs representing its selection as a non-last item in a bin, and a set $\mathcal{A}_i^d \subseteq \{((i, W'), (i+1, W')) : W' = 0, \ldots, W-1\}$ of dummy arcs representing the decision of not taking i item in a path. Each item $i = 1, \ldots, m$ is associated with a set $\mathcal{A}_i^- \subseteq \{((i, W'), v^-) : W' = 0, \ldots, W-1\}$, of sink arcs representing the decision of taking i as the last item in a bin. For the last item ($i = m$) in the input, we define $\mathcal{A}_m = \mathcal{A}_m^d = \emptyset$. The full set of arcs is given by $\mathcal{A} = \cup_{i \in I}(\mathcal{A}_i \cup \mathcal{A}_i^d \cup \mathcal{A}_i^-)$. An advantage of this network is that it explicitly models the ordering of the items in all feasible paths.

In the resulting arc flow model, the objective function minimizes the number of arcs reaching the sink node, i.e., the number of paths (bins) used. In this way, $c_{(u,v)}$ is 1, if $(u, v) \in \cup_{i \in I}\mathcal{A}_i^-$, and 0, otherwise. The side constraints guarantee that each item is included in at least one path (bin). Thus, for all $i \in I$, $b_i = 1$ and $a_{i(u,v)} = 1$, for all $(u, v) \in \mathcal{A}_i \cup \mathcal{A}_i^-$, and 0, otherwise.

In our experiments, we solve this model by NF-F, using $K = 10$ levels and the simple arc family $\mathcal{F} = \{F_{(u,v)} = \{(u, v)\} : (u, v) \in \mathcal{A}\}$, which partitions $\mathcal{A}$ into individual arcs.

3.8 Computational Experiments

This section discusses the results of our computational experiments. The algorithms were coded in C++ and the LP and MILP models were solved by Gurobi 9.1.1. The experiments were run on a computer with an Intel Xeon E3-1245 v5 at 3.50GHz and 32GB RAM, with a single-thread limit. First, based on the CSP, we evaluate the performance of the NF-F components. Then, we also compare our results with those obtained by the state-of-the-art algorithms on each problem.

3.8.1 Experiments on the Cutting Stock Problem

For the CSP, we consider the benchmark used to test the most recent exact methods, available at the BPPLIB [111]. They include classes Falkenauer, Hard, Scholl, Schwerin, and Waescher, which are all well-solved by state-of-the-art methods, and classes AI and ANI from [109], which contain several open instances. Classes AI and ANI are both composed of 250 instances, divided into 5 groups of 50 instances having the same number of items. Instances in AI have optimal solution value equal to the optimal dual bound z_{lb} of the pattern-based model, whereas instances in ANI have optimal solution value equal to $z_{lb}+1$.

Comparison of variable-fixing strategies. We propose four algorithms to evaluate the RCVF effectiveness. They consist in applying (or not) the RCVF strategies from Section 3.5 to some extent, and then solving the residual problem as an arc flow model by the MILP solver: in *No RCVF*, no variable-fixing is applied; *RCVF 1* uses the first strategy; *RCVF 1+2* uses the first and second strategies in sequence; and *RCVF 1+2+3*

Table 3.1: Comparison of RCVF strategies for the CSP (time limit 600s)

		No RCVF		RCVF 1			RCVF 1+2			RCVF 1+2+3		
Class	# ins	time	opt	rd. %	time	opt	rd. %	time	opt	rd. %	time	opt
AI200	50	29.9	**50**	35.2	16.3	**50**	61.8	14.4	**50**	66.6	22.7	**50**
AI400	50	410.6	27	41.5	242.8	36	78.5	171.1	38	84.7	141.8	**41**
AI600	50	600.0	0	30.8	543.7	7	68.5	350.4	25	78.3	254.6	**33**
AI800	50	600.0	0	29.2	593.5	2	74.0	371.3	24	83.6	301.4	**35**
AI1000	50	600.0	0	31.4	589.7	1	66.0	480.5	14	73.8	433.3	**24**
ANI200	50	53.9	**50**	61.1	13.0	**50**	96.9	2.0	**50**	97.9	1.4	**50**
ANI400	50	546.0	11	47.0	203.5	39	94.9	47.7	47	99.7	7.5	**50**
ANI600	50	600.0	0	29.4	583.8	4	84.7	204.4	34	99.8	36.9	**50**
ANI800	50	600.0	0	31.6	600.0	0	80.0	328.0	26	98.0	154.0	**47**
ANI1000	50	600.0	0	35.9	600.0	0	79.7	428.7	20	88.1	357.9	**31**
Falkenauer T	80	0.3	**80**	4.4	0.2	**80**	4.5	0.3	**80**	8.7	0.4	**80**
Falkenauer U	80	0.2	**80**	3.2	0.2	**80**	3.4	0.3	**80**	3.4	0.3	**80**
Hard	28	60.6	**28**	65.2	30.0	**28**	69.5	27.7	**28**	69.8	28.9	**28**
Random	3840	1.9	**3838**	15.9	2.2	**3838**	16.9	2.3	**3838**	17.0	15.0	3823
Scholl	1210	12.7	**1195**	12.9	20.7	1191	13.8	21.1	1190	13.8	45.5	1160
Schwerin	200	3.5	**200**	3.5	3.3	**200**	3.5	3.3	**200**	3.6	5.8	**200**
Waescher	17	298.0	**14**	17.4	305.1	**14**	17.7	306.1	**14**	17.7	362.2	**14**
Overall	5955	295.1	5573	29.1	255.8	5620	53.8	162.3	5758	59.1	127.6	**5796**

uses all strategies in sequence. A time limit of 600 seconds per instance is imposed. Table 3.1 gives average running times (time) in seconds, numbers of instances optimally solved (opt), and average percentages of arcs removed (rd. %).

Classes Falkenauer, Hard, and Schwerin are well-solved by all algorithms. The number of instances solved in class Waescher remains unchanged for all algorithms. Random and Scholl are the only classes in which applying the more expensive strategies worsens the number of instances solved. However, most of the instances in these classes fall in the case in which $z_{opt} = \lceil z_{lb} \rceil$, in which RCVF is not very helpful. However, by analyzing the subset of instances in these two classes in which $z_{opt} = \lceil z_{lb} \rceil + 1$, we noticed that RCVF led to a good average improvement.

For the hardest classes (AI and ANI), it is always worth to apply more expensive RCVF strategies, because the number of solved instances increases consistently. Already with the first strategy, a substantial reduction in the network size is observed. A larger improvement is obtained when the second and third strategies are used, especially for the ANI instances. These instances particularly benefit from RCVF, because at the beginning of all algorithms we already have the optimal solution (easily found by simple heuristics) and the challenge is just to prove optimality. Then, the use of sub-optimal dual solutions consistently strengthens the LP relaxation, and optimality of 134 out of 250 ANI instances is proven even before invoking the MILP solver. Clearly, this did not occurr in any AI instance. The average ratio of basic arcs removed per iteration for AI and ANI is, respectively, 10.1 and 56.5 in the second strategy, and 5.7 and 8.4 in the

third strategy. Overall, the results prove that applying the three strategies leads to better results on average.

Analysis of the heuristic behavior of the arc families. To analyze whether the branching scheme of Section 3.6 is a good approach to quickly find optimal solutions, we solve a single left branch for six different arc families: $\mathcal{F}_1^a$, $\mathcal{F}_5^a$, $\mathcal{F}_{10}^a$, $\mathcal{F}_5^b$, $\mathcal{F}_{10}^b$, and $\mathcal{F}_{50}^b$. For each family, we solve the linear relaxation (no RCVF is applied) and a single left branch, to obtain a feasible solution. Then, optimality is determined by comparing the solution value with the dual bound. For the sake of conciseness we do not report extended results.

The best balance between number of instances solved to proven optimality and computational time is given by $\mathcal{F}_1^a$. With this family, the number of AI instances with 200, 400, 600, 800, and 1000 items solved to proven optimality by this procedure is respectively 48, 34, 32, 28, and 16, with average computational time being, respectively, 0.6, 7.9, 68.9, 121.5, and 237.9 seconds. By comparing these results with column 'No RCVF' in Table 3.1, it is clear that the branching improves the MILP solver in quickly finding an optimal solution. All next experiments on the CSP are based on arc family $\mathcal{F}_1^a$.

Comparison of different networks. To evaluate the effectiveness of the new waste-limited network, we provide a comparison with network based on the MIM patterns [93]. For that, we solved the models derived from both networks by NF-F, with a time limit of 600 seconds per instance. Table 3.2 presents the results, giving average number of arcs (# arcs), average time (time), and number of proven optimal solutions (opt).

In the AI and ANI classes, the waste-limited network performs better than the MIM, mainly because goal solutions (of value $\lceil z_{lb} \rceil$) of instances in these classes do not allow any waste, which greatly benefits the reduction based on Remark 2. The remaining classes are all well-solved by using both networks, except for Waescher, in which three instances are always unsolved, and Scholl, in which the MIM performs better. Since there is no overall dominance among the networks, in the next experiments we compute both networks a priori and use smallest one.

Comparison with the state-of-the-art. We compare the results obtained by NF-F with: the enhanced solution of the reflect formulation (a pseudo-polynomial CSP arc flow model) by Delorme and Iori [108], solved with an Intel Xeon at 3.10 GHz and 8 GB RAM; the branch-and-cut-and-price algorithm by Wei et al. [333], solved with an Intel Xeon E5-1603 at 2.80-GHz and 8 GB RAM; and the branch-and-cut-and-price algorithm for general network flow problems with resource constraints by Pessoa et al. [264], solved with an Intel Xeon E5-2680 at 2.50 GHz with 128 GB RAM shared by 8 copies of the algorithm running in parallel. All experiments considered a time limit of 3600 seconds per instance. To compare the performance of each CPU, we provide their single-thread passmark indicators (STPI) (available at `www.passmark.com`), where higher values are associated with better performance. The computer used in the experiments of [108], [264], [333], and the present work have STPI 2132, 1763, 1635, and 1739, respectively.

The results are presented in Table 3.3. Under columns 'Arc Flow' and 'NF-F', we report the results of the arc flow model solved, respectively, directly by Gurobi and by

Table 3.2: Comparison of different networks (time limit 600s)

		MIM			Waste-limited network		
Class	# ins	# arcs	time	opt	# arcs	time	opt
AI200	50	85507.8	10.4	**50**	59745.4	2.0	**50**
AI400	50	671348.2	169.8	48	507569.6	25.2	**50**
AI600	50	2128423.7	327.0	37	1670828.2	118.1	**48**
AI800	50	6004603.6	475.4	23	4789367.9	271.2	**41**
AI1000	50	15134832.1	596.8	2	11949055.6	513.7	**18**
ANI200	50	83947.6	32.2	49	59289.7	3.0	**50**
ANI400	50	668211.3	278.1	44	503879.3	24.9	**50**
ANI600	50	2121692.8	487.5	17	1663330.5	105.1	**49**
ANI800	50	5990194.7	590.1	4	4772556.4	291.8	**43**
ANI1000	50	15104071.5	600.0	0	11915143.1	507.9	**21**
Falkenauer T	80	4840.1	0.3	**80**	1285.8	0.3	**80**
Falkenauer U	80	1513.8	0.1	**80**	2906.6	0.1	**80**
Hard	28	22219.4	23.7	**27**	27066.5	24.8	**27**
Random	3840	4238.4	0.6	**3840**	9833.7	0.9	**3840**
Scholl	1210	11384.0	1.3	**1210**	26426.9	12.9	1193
Schwerin	200	5560.4	0.2	**200**	11928.4	1.0	**200**
Waescher	17	101141.2	211.9	**14**	121536.2	334.9	**14**
Overall	5955	2831984.1	223.8	5725	2240691.2	131.6	**5854**

Table 3.3: Comparison with the state-of-the-art for the CSP (time limit 3600s)

		DI [108]		WLBL [333]		PSUV [264]		Arc Flow		NF-F	
Class	# ins	time	opt	time	opt	time	opt	time	opt	time	opt
AI200	50	8.5	**50**	4.2	**50**	52.3	**50**	21.5	**50**	2.0	**50**
AI400	50	1205.0	40	398.1	46	491.4	47	904.2	44	25.2	**50**
AI600	50	-	-	1759.6	27	1454.1	35	3326.9	9	192.4	**49**
AI800	50	-	-	2766.3	15	2804.7	28	3600.0	0	566.5	**46**
AI1000	50	-	-	3546.1	2	-	-	3600.0	0	1577.1	**36**
ANI200	50	49.3	**50**	13.9	**50**	16.7	**50**	16.3	**50**	3.0	**50**
ANI400	50	2703.9	17	436.2	47	96.0	**50**	1252.5	42	24.9	**50**
ANI600	50	-	-	3602.7	0	3512.5	3	3473.5	6	140.7	**50**
ANI800	50	-	-	3605.9	0	3600.0	0	3600.0	0	393.2	**49**
ANI1000	50	-	-	3637.7	0	-	-	3600.0	0	1302.5	**43**
Falkenauer T	80	1.0	**80**	1.9	**80**	16.0	**80**	0.8	**80**	0.3	**80**
Falkenauer U	80	0.1	**80**	3.8	**80**	-	-	0.1	**80**	0.1	**80**
Hard	28	4.2	**28**	41.5	**28**	17.0	**28**	39.9	**28**	23.6	**28**
Random	3840	-	-	6.2	**3840**	-	-	1.4	**3840**	0.9	**3840**
Scholl	1210	6.6	**1210**	5.0	**1210**	-	-	8.2	**1210**	1.4	**1210**
Schwerin	200	0.2	**200**	0.3	**200**	-	-	1.3	**200**	0.2	**200**
Waescher	17	41.3	**17**	8.7	**17**	-	-	1510.0	**17**	161.2	**17**

Table 3.4: Comparison with the state-of-the-art for the 2GCSP (time limit 7200s)

		MMH [247]		MAV [220]		SAV [297]		CI [93]*		NF-F	
Class	# ins	time	opt	time	opt	time	opt	time	opt	time	opt
A	43	277.2	42	377.9	**43**	735	41	1.9	**43**	0.3	**43**
A-r	43	1014.2	37	-	-	-	-	606.9	40	59.7	**43**
APT	20	-	-	-	-	3642.6	12	729.6	**18**	472.4	**18**
APT-r	20	-	-	-	-	-	-	1599.4	**16**	1557.4	**16**

∗: results of our reimplementation of the arc flow model in [93]

NF-F. For the AI and ANI classes, the best previous results were obtained by Pessoa et al. [264], who optimally solved 263 out of 500 instances. NF-F raised the number of proven optima to 473, and it could solve for the first time many ANI instances with 800 and 1000 items, mainly due to the new RCVF strategies. The LP relaxation strengthening obtained by these strategies allowed some of these instances to be solved already at the root node. The new branching scheme helped finding an optimal solution for several hard AI instances within a reasonably short computational time. For the remaining classes, the method by Wei et al. [333] already performed well. Our method could improve the average solution time of all classes with the exception of Waescher, because of a few instances that required a large computational time in the solution of specific left branches.

3.8.2 Experiments on the Two-Staged Cutting Stock Problem

For the 2GCSP, we compare NF-F with the most recent exact methods for the problem, namely: the B&P by Mrad et al. [247]; and the pseudo-polynomial MILP models by Macedo et al. [220], Silva et al. [297], and Côté and Iori [93]. The best published results are the ones by the arc flow model in [93], which is also the model that we solve with NF-F. The experiments were based on benchmark classes A and APT, and their respective variants A-r and APT-r in which items and stock sheets are rotated by 90 degrees. These instances can be downloaded from the 2DPackLib (`http://or.dei.unibo.it/library/2dpacklib`).

Table 3.4 reports, for each algorithm and each class, the average time and the number of instances solved to proven optimality. The algorithm in [247] was solved on a Pentium IV at 2.2 GHz with 4 GB RAM (STPI 562), and the models in [220, 297] were solved on an Intel Core Duo at 1.87 GHz with 2 GB RAM (STPI 675). We also compare NF-F with the arc flow model solved directly by Gurobi, which is the approach in [93]. Thus, column CI does not report the results in [93], but rather the updated (and improved) results obtained by solving their model by the current version of Gurobi in our computer. All results consider a time limit of 7200 seconds per instance.

Although not all previous publications addressed all classes, we can see that the arc flow model in [93] dominates on average the previous works. NF-F provides the best results overall: It uses a smaller average time, mainly due to the effectiveness of the branching scheme, and solves all instances in class A-r for the first time. The six remaining open instances in classes APT and APT-r have an absolute gap of just one stock sheet.

3.8.3 Experiments on the Skiving Stock Problem

For the SSP, we compare NF-F with the reflect formulation in [232] and with a reimplementation of the arc flow model in [233] executed with the latest version of Gurobi on our computer. The results in [232] were obtained by an AMD A10-5800K with 16 GB RAM (STPI 1493). Our experiments are based on the three benchmark classes A1, composed of 1260 easy instances, A2, composed of 1050 hard instances, and B, composed of 160 hard instances. The arc flow model at the basis of NF-F is the one in [233].

Table 3.5 gives average times and numbers of instances optimally solved. For class A1, all methods could solve all instances very quickly. For class A2, all methods could solve all instances with up to 100 items, but NF-F was quicker. For instances with 250 and 500 items, NF-F could not improve the solution of the arc flow model as a (general) MILP. The reason is that for some instances there are left branches whose resulting problems, although consistently smaller, are very hard to solve. We believe that the source of such difficulties is often an excessive restriction on the number of optimal solutions remaining in these branches. For class B, NF-F improved upon the other methods and optimally solved all instances for the first time. It consistently decreased the average time w.r.t. the direct solution of the arc flow as a MILP, showing that the branching scheme is very effective for all instances in this class.

3.8.4 Experiments on the Ordered Open-End Bin Packing Problem

For the OOEBPP, we solve the arc flow model derived from the set covering formulation in [71], either directly by Gurobi or by NF-F. We compare our results also with the ones obtained by the B&P in [71], which were obtained by a Pentium IV 1.6 GHz with 512 MB of RAM (STPI 562).

The first test is based on the benchmark used in [71], which are available at the 2DPackLib. The results are reported in Table 3.6. Ceselli and Righini [71] did not solve instances GCUT5-13. Their B&P already performed very well and left just one open instance. The arc flow model solved either directly by Gurobi or by NF-F could close all instances (the latter approach used a slightly smaller time). Since the available benchmarks for the OOEBPP appear to be very easy for current solvers, we propose a second set of experiments based on new randomly generated instances that we derived from the CSP class Random. We consider all instances with 50, 100, and 200 items.

Table 3.5: Comparison with the state-of-the-art for the SSP (time limit 3600s)

		MDISS [232]		MS [233]*		NF-F	
Class	# ins	time	opt	time	opt	time	opt
A1	1260	0.1	**1260**	0.1	**1260**	0.1	**1260**
A2	1050	250.9	1011	183.4	**1030**	148.7	1024
B	160	970.7	126	1354.5	136	61.4	**160**

∗: results of our reimplementation of the arc flow model in [233]

Table 3.6: Comparison with the state-of-the-art for the OOEBPP (time limit 3600s)

		CR [71]		Arc Flow		NF-F	
Class	# ins	time	opt	time	opt	time	opt
GCUT1-4	4	0.6	**4**	0.1	**4**	0.0	**4**
GCUT5-13	9	-	-	1.3	**9**	0.1	**9**
NGCUT	12	0.1	**12**	0.0	**12**	0.0	**12**
CGCUT	3	0.1	**3**	0.3	**3**	0.4	**3**
BENG	10	0.1	**10**	0.3	**10**	0.1	**10**
HT	9	0.1	**9**	0.0	**9**	0.0	**9**
CLASS	500	8.7	499	11.3	**500**	3.7	**500**
Overall	547	1.6	537	1.9	**547**	0.6	**547**
Random50	480	-	-	1.7	**480**	1.3	**480**
Random100	480	-	-	73.4	**479**	38.7	**479**
Random200	480	-	-	863.3	426	168.9	**466**
Overall	1440	-	-	312.8	1385	69.6	**1425**

The ordering of the items required by the OOEBPP is obtained by sorting the original items by weights randomly generated from a uniform distribution. The results for these instances are also presented in Table 3.6. Overall, NF-F performs better than arc flow both in terms of average time and number of instances solved. The arc flow solution as a MILP left 55 open instances, but with NF-F this number decreased to 15, while using a consistently smaller computing time.

3.9 Conclusions

We proposed a framework for the exact solution of network flow models. It is tailored to efficiently solve pseudo-polynomial arc flow models that have very strong relaxation but also a huge number of arcs. Its main practical components include a general column(-and-row) generation algorithm for network flow models, RCVF strategies that explore alternative and possibly sub-optimal dual solutions, and a highly asymmetric branching scheme that exploits the potential of MILP solvers. The correctness of the LP-based methods and of the RCVF strategies are supported by a number of theoretical results.

We performed extensive computational experiments on well-studied C&P problems, in which we solved a large number of instances to proven optimality for the first time. The new RCVF strategies closed many hard instances already at the root node. The new branching scheme helped finding optimal solutions for many hard instances within short computational time, largely improving the results by state-of-the-art algorithms on all problems addressed.

Although we focus on network flow, an extension to more general DWM characterizations is also envisaged. The core of the proposed RCVF strategies and branching scheme lies in the ability to efficiently forbid pricing decisions (which in our case are given as arcs in a network). By having this ability in a general pricing algorithm, one can extend

such techniques to solve a DWM without relying on a network flow characterization. In this case, the restricted problem given in the left branch would be solved by alternative methods (as, e.g., specialized combinatorial algorithms) instead than as an arc flow model.

Other than the C&P problems presented in this paper, we also tested the solution of a number of other problems by our framework in preliminary experiments. This revealed some interesting drawbacks that we would like to address in future research:

(i) Iterative aggregation/disaggregation is a state-of-the-art technique to deal with the huge number of arcs in pseudo-polynomial arc flow models for a couple of problems, as, e.g., the time-dependent traveling salesman problem with time windows (see [328]). In some of these problems, hard instances often produce networks so huge that cannot even fit in the computer memory. This is a critical issue for our framework, because the full network is required in input. We believe that embedding aggregation/disaggregation techniques within the framework is an interesting research direction that may improve the solution of such problems;

(ii) Our branching scheme is tailored to be effective on models with very strong relaxations, but we tested models with weaker relaxations derived, for instance, from vehicle routing and scheduling problems. As expected, the results were not competitive with the state-of-the-art. The left branch generates small arc flow models, but their weak linear relaxations usually does not allow an efficient solution by a general MILP solver. In addition, the number of right-branch constraints required to significantly raise the bound is often unpractical. Then, to better address such models, we aim at investigating effective primal cuts tailored for general arc flow models.

Chapter 4

A Branching Scheme for a Class of Parallel Machine Scheduling

4.1 Introduction

This paper addresses a parallel machine scheduling problem with the objective of minimizing weighted completion times. In this problem, we are given a set $\mathcal{J}$ of jobs to be scheduled on a set $\mathcal{M}$ of identical parallel machines. Each job $j \in \mathcal{J}$ has a processing time p_j and a penalty weight w_j, and it must be assigned to and processed by a unique machine without preemption. Each machine processes at most one job at a time. By denoting C_j as the completion time of a job j in a solution, the objective is to find a feasible schedule that minimizes $\sum_{j \in \mathcal{J}} w_j C_j$. Following the three-field classification of Graham et al. [155], this problem is denoted as $P||\sum w_j C_j$.

To the best of our knowledge, the state-of-the-art for the $P||\sum w_j C_j$ is the arc flow model by Kramer et al. [197]. The authors propose an enhanced arc flow model, based on problem-specific properties, which can solve many more hard instances to proven optimality when compared with the previous state-of-the-art methods. The enhancement is based on finding and removing redundant arcs from the network, to reduce the number of variables, and grouping equivalent jobs into demands, to reduces the number of constraints. This model has a very strong linear relaxation, and the optimal integer solution value of benchmark instances is usually equal to the round-up optimal solution value of the linear relaxation. Hence, the main difficulty is usually to find an optimal integer solution, but proving its optimality is generally easy. The method by Kramer et al. [197] finds an initial incumbent by means of a non-trivial iterated local search based metaheuristic, and their overall method solves 502 out of 560 benchmark instances to proven optimality.

In this paper, we propose an improved solution method of the enhanced model of Kramer et al. [197], that can solve all benchmark instances to proven optimality. The solution is based on a three-phase branching scheme. The first phase consists of reduced-cost based branching (RCBB). This branching scheme selects a large set of variables, whose reduced costs make them unlikely to be part of any optimal solutions. In the left branch, these variables are removed from the model, leading to a small restricted model that likely holds an optimal solution; in the right branch, a linear constraint imposes that

at least one of such variables is in the solution.

The second phase of branching is specialized to parallel machine scheduling problems. It first identifies, for each job, the minimum time interval that contains all occurrences of the job in the current fractional solution. Then, the left branch imposes that each job must be within its respective interval; this is done by removing a (hopefully large) set of arcs from the model. The right branch adds a linear constraint to the model imposing that at least one of the jobs must be outside of its respective time interval. The restricted problem resulting from the left branch is usually small and holds an optimal solution for the original problem. In this way, the third phase of the branching scheme solves this restricted problem directly by a general MILP solver. As shown by our extensive computational experiments, the overall scheme usually allows us to quickly find an optimal integer solution, even without relying on an initial good quality solution.

The remaining of this paper is organized as follows. Section 4.2 briefly reviews general arc flow formulations and the application to the $P||\sum w_j C_j$. Section 4.3 provides details on the solution of the linear relaxation; discusses a variable fixing algorithm; and presents a method to transform dual infeasible solutions into feasible ones. The overall algorithm is discussed in Section 4.4, and Section 4.5 discusses the results of our computational experiments. Finally, Section 7.5 contains our conclusions and future research directions.

4.2 Preliminaries

In this section, we formally define a general arc flow formulation and present an application to the $P||\sum w_j C_j$.

4.2.1 General Arc Flow Formulations

A network $\mathcal{N}$ is composed of a set $\mathcal{V}$ of nodes, and a set $\mathcal{A} \subseteq \mathcal{V} \times \mathcal{V}$ of arcs. The node set contains a source node $v^+ \in \mathcal{V}$ and a sink node $v^- \in \mathcal{V}$, such that no arcs enter v^+ or leave v^-. Following de Lima et al. [104], we assume that $\mathcal{N}$ is acyclic, a property that allows us to derive efficient LP-based algorithms later discussed in Section 4.3.1. This assumption encompasses a large class of resource constrained network flow models, usually derived from Dantzig-Wolfe decompositions.

In an arc flow model, we are given a vector $b \in \mathbb{R}^m$ of right-hand-side coefficients and, for each arc $(u, v) \in \mathcal{A}$, we are given an objective coefficient $c_{(u,v)} \in \mathbb{R}$ and a side-constraint

matrix column $a_{(u,v)} \in \mathbb{R}^m$. The following is a general arc flow model:

$$\min \sum_{(u,v)\in\mathcal{A}} c_{(u,v)}\varphi_{(u,v)}, \tag{4.1}$$

$$\text{s.t.:} \sum_{(u,v)\in\mathcal{A}} \varphi_{(u,v)} - \sum_{(v,w)\in\mathcal{A}} \varphi_{(v,w)} = \begin{cases} z, & \text{if } v = v^+, \\ -z, & \text{if } v = v^-, \\ 0, & \text{otherwise,} \end{cases} \qquad \forall v \in V, \tag{4.2}$$

$$\sum_{(u,v)\in\mathcal{A}} a_{(u,v)}\varphi_{(u,v)} \geq b, \tag{4.3}$$

$$\varphi_{(u,v)} \in \mathbb{Z}_+, \qquad \forall (u,v) \in \mathcal{A} \tag{4.4}$$

$$z \in \mathbb{Z}_+. \tag{4.5}$$

For each arc $(u,v) \in \mathcal{A}$, the integer variable $\varphi_{(u,v)}$ represents the flow on (u,v). Constraints (4.2) are flow conservation constraints, and (4.3) are general side constraints on the network flow.

Let $\mathcal{P}$ represent the set of all paths in $\mathcal{N}$ from the source to the sink and $\mathcal{A}_p \subseteq \mathcal{A}$ be the set of all arcs in $p \in \mathcal{P}$. For each path $p \in \mathcal{P}$, let $c_p = \sum_{(u,v)\in\mathcal{A}_p} c_{(u,v)}$ and $a_p = \sum_{(u,v)\in\mathcal{A}_p} a_{(u,v)}$. By the flow conservation theorem by Ahuja et al. [1], the LP relaxation of (4.1)–(4.5) is equivalent to the LP relaxation of the following path flow model (see, e.g., de Lima et al. [101]):

$$\min \sum_{p\in\mathcal{P}} c_p\lambda_p, \tag{4.6}$$

$$\text{s.t.:} \sum_{p\in\mathcal{P}} a_p\lambda_p \geq b, \tag{4.7}$$

$$\lambda_p \in \mathbb{Z}_+, \qquad \forall p \in \mathcal{P}, \tag{4.8}$$

where λ_p represents the flow on each path $p \in \mathcal{P}$. The flow conservation theorem proves the existence of a one-to-one primal correspondence between the LP relaxation of both models that preserves objective value.

4.2.2 Application to the $P||\sum w_jC_j$

Next, we provide a brief review of the arc flow model for the $P||\sum w_jC_j$ proposed by Kramer et al. [197]. The arc flow model is equivalent to the time indexed formulation by Sousa and Wolsey [302], and each node is associated with a time instant, i.e., $\mathcal{V} \subseteq \{0, ..., T\}$, where T represents an upper bound on the makespan of any optimal solution. Moreover, $v^+ = 0$ and $v^- = T$. There is a set $\mathcal{A}_j \subseteq \{(t, t+p_j) : t \in \mathcal{V}\}$ of arcs associated with each job j. Each arc $(t, t+p_j) \in \mathcal{A}_j$ is associated with the decision of processing job j starting at time instant t and finishing at $t+p_j$. Moreover, there is a set $\mathcal{A}^- \subseteq \{(t, v^-) : t \in \mathcal{V}\}$ of sink arcs, which represent the decisions of stopping a machine to operate. The overall set of arcs is given by $\mathcal{A} = \cup_{j\in\mathcal{J}}\mathcal{A}_j \cup \mathcal{A}^-$.

The model has a side constraint (4.3) associated with each job, enforcing that at least

one arc associated with the respective job is in the solution. Then, we define $b = (1, 1, ..., 1)$ and $m = |\mathcal{J}|$. For each arc $(t, t + p_j) \in \mathcal{A}_j$, its objective function coefficient is $w_j(t + p_j)$, and its side-constraint column has coefficient 1 at the j-th position and 0 in the remaining positions. Each arc $(t, v^-) \in \mathcal{A}^-$ has null objective coefficient and column.

In the $P||\sum w_j C_j$, the schedule of each machine in an optimal solution follows the *weighted shortest processing time* rule - that is, the sequence of jobs in a single machine is sorted according to non-increasing w_j/p_j. This allows a great reduction in the initial network size, by following the classical algorithm by Valério de Carvalho [314] to create a reduced network for the cutting stock problem.

The above definitions are enough to build the standard model for the $P||\sum w_j C_j$. Now we briefly discuss the enhancements by Kramer et al. [197] that allow to reduce the number of variables and constraints of the standard model without loss of optimality. The first enhancement provides a computation of T (based on the procedure by van den Akker et al. [317]) and also an earlier instant at which each machine could finish operating. The second enhancement computes a time window for each job j, based on two sets of jobs which can be demonstrated to always be processed, respectively, before or after j in an optimal solution. The last enhancement consists of grouping equivalent jobs (i.e., jobs with the same processing time and weight) into demands, by combining their side constraints.

4.3 Solution of the Linear Relaxation

Since the arc flow model for the $P||\sum w_j C_j$ has pseudo-polynomial size, large instances usually derive models with a huge number of variables. Their linear relaxation could be solved by column-and-row generation, as discussed by de Lima et al. [103]. However, we found that the linear relaxation of this formulation is highly degenerate, and that the barrier algorithm implemented in state-of-the-art MILP solvers provides a better average performance than all column generation algorithms we implemented in preliminary experiments. Due to the high degeneracy, column generation for the path flow model provides the worst performance. Arc flow models are less degenerate and converge within a smaller number of iterations, but since they have a larger basis, each column generation iteration is more expensive (see, e.g., de Lima et al. [101]). Regarding column generation for the arc flow model, we implemented algorithms that generate either single or multiple paths, each to be decomposed into a set of arcs to be inserted in the restricted master problem. We also implemented in-out separation strategies (see, e.g., Ben-Ameur and Neto [28]), which did not provide a substantial increase in computational performance. Hence, our algorithm simply uses the barrier algorithm of a general MILP solver to solve the LP relaxation of each arc flow model.

In the remainder of this section, we are mainly concerned with the dual solution of

the LP relaxation of an arc flow model. For that, consider the dual LP model:

$$\max\ b^\intercal \beta, \tag{4.9}$$

$$\text{s.t.:}\ -\alpha_u + \alpha_v + a_{(u,v)}{}^\intercal \beta \leq c_{(u,v)}, \qquad \forall (u,v) \in \mathcal{A}, \tag{4.10}$$

$$\alpha_{v^+} - \alpha_{v^-} \leq 0, \tag{4.11}$$

$$\alpha_u \in \mathbb{R}, \qquad \forall u \in \mathcal{V}, \tag{4.12}$$

$$\beta \in \mathbb{R}^m_+, \tag{4.13}$$

where α and β are the dual variables associated with (4.2) and (4.3), respectively. The dual constraints (4.10) and (4.11) are associated with primal variables φ and z, respectively.

4.3.1 Reduced-Cost Variable-Fixing

This section briefly reviews the reduced-cost variable-fixing (RCVF) method proposed by Irnich et al. [183] and recently applied to general arc flow models by de Lima et al. [103]. For that, we define $\mathcal{P}_{(u,v)} \subseteq \mathcal{P}$ as the set of all paths that contain $(u,v) \in \mathcal{A}$. To improve the RCVF effectiveness, we use a technique to compute the minimum reduced cost of all arcs (u,v), given a fixed $\overline{\beta}$ solution, but allowing α to be variable. For that, we exploit the reduced cost of paths $p \in \mathcal{P}_{(u,v)}$, which can be defined solely in terms of β variables. Let us define the minimum reduced cost of an arc as follows:

Definition 5. *Given a partial dual solution $\overline{\beta} \in \mathbb{R}^m_+$, the* minimum reduced cost of an arc *$(u,v) \in \mathcal{A}$ is the minimum reduced cost of any path that contains (u,v), and is given by $\overline{c}_{(u,v)} = \min\{c_p - a_p^\intercal \overline{\beta} : p \in \mathcal{P}_{(u,v)}\}$.*

The minimum reduced cost of all arcs can be computed by bidirectional dynamic programming by solving the following recursions:

$$\overline{c}_v^+ = \begin{cases} \min\{\overline{c}_u^+ + (c_{(u,v)} - a_{(u,v)}{}^\intercal \overline{\beta}) : (u,v) \in \mathcal{A}\}, & \text{if } v \neq v^+, \\ 0, & \text{if } v = v^+, \end{cases} \tag{4.14}$$

$$\overline{c}_v^- = \begin{cases} \min\{\overline{c}_w^- + (c_{(v,w)} - a_{(v,w)}{}^\intercal \overline{\beta}) : (v,w) \in \mathcal{A}\}, & \text{if } v \neq v^-, \\ 0, & \text{if } v = v^-. \end{cases} \tag{4.15}$$

Then, the minimum reduced cost of an arc (u,v) is given by $\overline{c}_{(u,v)} = \overline{c}_u^+ + (c_{(u,v)} - a_{(u,v)}{}^\intercal \overline{\beta}) + \overline{c}_v^-$. For details, see Irnich et al. [183] and de Lima et al. [103].

After computing the minimum reduced costs of all arcs, we can proceed with reduced-cost variable-fixing to remove arcs from the model without loss of optimality. The overall idea is the following. Suppose we have a incumbent integer solution of objective value z_{ub}. Since the objective function is integer, to improve the incumbent we are only interested in finding solutions with objective value at most $z_{ub} - 1$. Then, given a (partial) dual-feasible solution $\overline{\beta}$ of objective value $z_{lb} = b^\intercal \overline{\beta}$, every arc $(u,v) \in \mathcal{A}$ with $\overline{c}_{(u,v)} > (z_{ub} - 1) - z_{lb}$ can be removed from the model, since they do not improve the incumbent solution.

4.3.2 Dealing with Dual Infeasibility

To safely perform RCVF, we need a feasible dual solution. However, since the most efficient LP solvers work under the limited precision of floating-point arithmetic, the dual solution returned by the solver may be feasible only within such precision. For instance, the solver we use (Gurobi 9.1.1) has minimum dual feasibility tolerance of 10^{-9}. Recently, de Lima et al. [103] applied a generalization of the method by Held et al. [164] to transform a dual infeasible solution into a feasible one. This method is valid for formulations where the row of each side constraint has only coefficients of the same sign. By considering a tolerance of 10^{-9}, the maximum expected loss in the dual objective of the resulting dual solution is $10^{-9} \sum_{i=1}^{m} b_i$, but this resulting solution is not guaranteed to be in a face of the dual polytope.

In preliminary experiments, we noticed that using the default (10^{-6}) instead of the minimum (10^{-9}) feasibility tolerance of Gurobi makes a significant difference in computational efficiency when solving the LP arc flow model for the $P||\sum w_j C_j$. For that, we decided to use the default tolerance. Motivated to find a dual solution that minimizes the impact on the dual objective, we derived a method that finds a solution that is guaranteed to be in the face of the dual polytope. This method can be used in any formulation (not restricted to any special structure).

Let us consider the dual LP relaxation of the path flow model (4.6)–(4.8):

$$\max\ b^{\intercal}\beta, \tag{4.16}$$

$$\text{s.t.:}\ a_p^{\intercal}\beta \leq c_p, \qquad \forall p \in \mathcal{P}, \tag{4.17}$$

$$\beta \in \mathbb{R}_+^m, \tag{4.18}$$

where β are the dual variables associated with primal constraints (4.7). The primal LP path flow model has an exponential number of variables and is usually solved by column generation, which translates to row generation of the dual model (4.16)–(4.18). For that, the dual model is initialized with a restricted subset of constraints, and a separation problem is solved iteratively to identify non-basic constraints that are violated by the current dual solution $\overline{\beta}$. In a path flow formulation, the separation can be to determine a shortest path on $\mathcal{N}$, and can be formulated as in (4.14).

De Lima et al. [103] prove a one-to-many correspondence between the solution of space of (4.10)–(4.13) and (4.17)–(4.18) that preserves the solution values of β. In particular, any dual-feasible solution $\overline{\beta}$ of the dual path flow model can be transformed into a dual-feasible solution for the arc flow model by maintaining the same $\overline{\beta}$ and computing $\overline{\alpha}$ as in (4.14), by setting $\overline{\alpha}_v := \overline{c}_v^+$. Then, given a dual-infeasible solution $\langle \overline{\alpha}, \overline{\beta} \rangle$ of the arc-flow model, we exploit formulation (4.16)–(4.18) to transform $\overline{\beta}$ into a dual-feasible solution and then use (4.14) to recompute a feasible solution for α.

Let β_{in} and β_{out} represent, respectively, an interior and exterior point of the dual polytope. Naturally, β_{in} is any dual-feasible solution, and in problems where $c_p \geq 0$, for all $p \in \mathcal{P}$, as in the case of the $P||\sum w_j C_j$, we have that $(0, 0, \ldots, 0)$ is one such solution. We consider β_{out} as the dual-infeasible solution at hand. Notice that for any $\mu \in [0, 1]$, the solution $\beta_\mu = \mu\beta_{in} + (1 - \mu)\beta_{out}$ lies in a line segment connecting β_{in} and

β_{out}. In particular, since β_{in} is an interior point and β_{out} is an exterior point, there exists μ such that β_μ represents a point in face of the polytope, which is also the dual-feasible solution in the segment that maximizes the dual objective. Our method aims at finding such μ. This discussion is highly related with the in-out separation method for stabilized column generation (see, e.g, Ben-Ameur and Neto [28] and Pessoa et al. [262]) and the ray projection by Porumbel [271].

We start with $\mu := \mu^0 = 1$, which gives $\beta_{\mu^0} = \beta_{out}$. Then, we iteratively compute μ^t as follows. At iteration t, we use the separation algorithm to compute the dual constraint that is most violated by β_{μ^t}. Suppose this constraint is associated with path $p^t \in \mathcal{P}$, i.e.:

$$a_{p^t}^\intercal \beta \leq c_{p^t}.$$

We want to compute μ^{t+1} such that this violated constraint is tightly satisfied by $\beta_{\mu^{t+1}}$. For this, we substitute β in the violated constraint by $\beta_{\mu^{t+1}}$, deriving:

$$a_{p^t}^\intercal \beta_{in} + \mu^{t+1} a_{p^t}^\intercal (\beta_{out} - \beta_{in}) \leq c_{p^t}.$$

Since the first term $a_{p^t}^\intercal \beta_{in}$ is already less than or equal to c_{p^t}, we can suppose that $a_{p^t}^\intercal (\beta_{out} - \beta_{in})$ is non-negative, otherwise the constraint is satisfied for any μ, a contradiction. In this way, the inequality can be rewritten as:

$$\mu^{t+1} \leq \frac{c_{p^t} - a_{p^t} \beta_{in}}{a_{p^t}(\beta_{out} - \beta_{in})}$$

For that, we define $\mu^{t+1} := \frac{c_{p^t} - a_{p^t} \beta_{in}}{a_{p^t}(\beta_{out} - \beta_{in})}$. At any iteration t, if no violated constraint can be found, then we know that β_{μ^t} is a dual-feasible solution that lies in the polytope face associated with path p^t. In addition, it is easy to see that $\mu^t \geq \mu^{t+1}$, which guarantees the convergence of the algorithm.

At the end of the algorithm, we have derived a dual solution β_μ that is feasible and is a point in a face of the polytope of (4.17)–(4.18). As previously mentioned, this solution can be easily converted to a dual-feasible solution of the LP arc flow model with the same objective value.

4.4 Overall Algorithm

This section presents the proposed algorithm to solve the arc flow model for the $P||\sum w_j C_j$. It consists of the three-phase branch-and-bound that is now described in detail. In the first two phases, we construct a large set $\mathcal{B} \subset \mathcal{A}$ of arcs in a way that we expect that by removing $\mathcal{B}$ from the current model the resulting model could be quickly solved by a MILP solver while still returning good quality solutions. The construction of $\mathcal{B}$ is explained in the next sections. Then, in the first two phases, the left branch removes $\mathcal{B}$ from the model, whereas in the right branch a linear constraint $\sum_{(u,v)\in\mathcal{B}} \varphi_{(u,v)} \geq 1$ is added to impose that at least one arc in $\mathcal{B}$ is in the solution. The third phase consists of simply solving the current arc flow model directly by a MILP solver. Figure 4.1 provides a graphical example of the branching tree.

The linear relaxation on the first- and second-phase nodes are solved by a general LP solver. Whenever the solver returns an infeasible dual solution, we use the method in Section 4.3.2 to obtain a dual-feasible solution. The dual-feasible solution is then used to perform RCVF in its respective node.

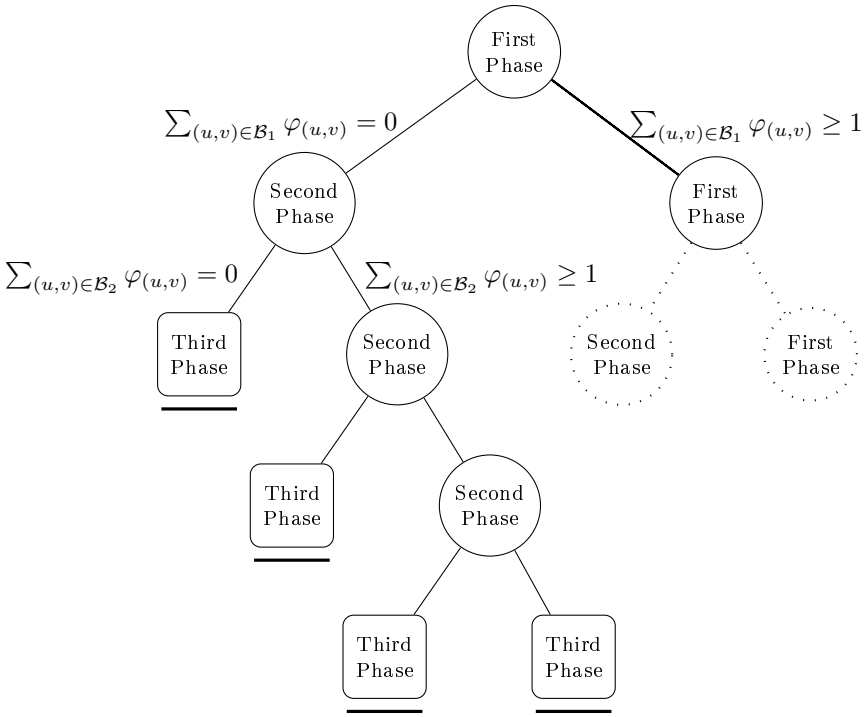

Figure 4.1: Example of a branching tree with a limit of $\Pi_b = 3$ consecutive second-phase branches.

4.4.1 First-Phase Branching

The first phase consists of RCBB, which can be applied to general MILP models. It may be useful for models with a strong relaxation, where the absolute integrality gap is likely to be of at most a known value $\Pi \in \mathbb{Z}_+$. For instance, in the $P||\sum w_j C_j$, the optimal integer solution value is usually equal to the rounded-up optimal solution value of the linear relaxation. In this case, we know that the absolute integrality gap is usually at most $\Pi = 1$.

This branching rule receives in input a dual solution and the minimum reduced cost $\bar{c}_{(u,v)}$ of each arc $(u,v) \in \mathcal{A}$. Then, we define $\mathcal{B} = \{(u,v) \in \mathcal{A} : \bar{c}_{(u,v)} \geq \Pi\}$ and proceed by creating a left branch where all arcs in $\mathcal{B}$ are removed from the model, and a right branch where the linear constraint $\sum_{(u,v)\in\mathcal{B}} \varphi_{(u,v)} \geq 1$ is included. The left branch generates a second-phase node and the right branch generates another first-phase node.

Note that this branching scheme does not require a primal solution but rather only a dual solution. Differently from branching based on a primal solution, it is not clear whether a vertex dual solution would lead to a better branching effectiveness, when compared to a non-vertex solution. In this way, if the LP is solved by a barrier algorithm, the crossover step can be disabled to reduce computational time without clear loss of

branching effectiveness.

4.4.2 Second-Phase Branching

The second phase aims at further reducing the size of the model resulting from the first-phase branching. First, let us provide some formal definitions. Given a fractional primal solution $\bar{\varphi}$, we define $\mathcal{L}_j = \{(t, t+p_j) \in \mathcal{A}_j : \bar{\varphi}_{(t,t+p_j)} > 0\}$ as the set of arcs of job j with positive value. Then, for each job j, we define a time interval $[t_j^-, t_j^+]$, where $t_j^- = \arg\min_t\{(t, t+p_j) \in \mathcal{L}_j\}$ and $t_j^+ = \arg\max_t\{(t, t+p_j) \in \mathcal{L}_j\}$. Since the $P||\sum w_jC_j$ has a very strong linear relaxation, these intervals are tight, and for the benchmark instances, there is often an optimal integer solution where all jobs start to be processed within their corresponding interval. We construct $\mathcal{B} = \{(t, t+p_j) \in \mathcal{A}_j : t \notin [t_j^-, t_j^+], j \in \mathcal{J}\}$ and proceed by creating a left branch where all arcs in $\mathcal{B}$ are removed from the model, and a right branch that includes the linear constraint $\sum_{(u,v)\in\mathcal{B}} \varphi_{(u,v)} \geq 1$. The left branch generates a third-phase node, and the right branch generates another second-phase node. To avoid an excessive number of second-phase right branches, after $\Pi_b = 3$ consecutive such branches, the resulting node is solved as a third-phase node.

This branching rule can also be applied to general time-indexed formulations for non-preemptive scheduling problems.

4.5 Computational Results

We performed computational experiments to compare the efficiency of our algorithm with the state-of-the-art. The latter consists of solving the arc flow model directly by a general MILP solver. The proposed algorithms were coded in C++, and the LP and MILP models were solved by Gurobi 9.1.1. The experiments were run by using a single thread on a computer with an Intel Xeon E3-1245 v5 at 3.50GHz and 32GB RAM.

We considered the most difficult benchmark set solved by Kramer et al. [197]. In this set, $|\mathcal{J}| \in \{30, 100, 400, 700, 1000\}$ and $|\mathcal{M}| \in \{2, 4, 6, 8, 16, 30\}$. The job processing times were obtained from a uniform distribution $U[1, p_{max}]$, where $p_{max} \in \{20, 100\}$, and the penalty weights were drawn according to a uniform distribution $U[1, 20]$. For each combination of $|\mathcal{J}|$, $|\mathcal{M}|$, and p_{max}, except when $|\mathcal{J}| = 30$ and $|\mathcal{M}| \in \{16, 30\}$, 10 instances were created. There is a total of 560 instances.

For a comparison, we solved the arc flow model for each instance in two ways: directly by Gurobi (reported as 'Arc Flow') and by the branch-and-bound algorithm proposed here (reported as 'Our Solver'). In both cases, an initial incumbent solution is computed by a single iteration of a simple greedy heuristic. For each instance, we considered a time limit of 3600 seconds. Table 4.1 presents the overall results for both solution methods. For each class (defined by $|\mathcal{J}|$ and $|\mathcal{M}|$), we report the average time ('time'), the number of instances solved to proven optimality ('opt') and for our solver, the number of times we invoked the MILP solver, i.e., the number of third-phase nodes visited ('# MILP').

Table 4.1 shows that all instances with $p_{max} = 20$ could be solved to proven optimality by both methods. However, our solver is always faster and could reduce the computational time by more than half for most classes. Our solver could solve all instances with

Table 4.1: Overall results (time limit 3600s)

		$p_{max} = 20$					$p_{max} = 100$				
		Arc Flow		Our Solver			Arc Flow		Our Solver		
$\|\mathcal{J}\|$	$\|\mathcal{M}\|$	time	opt	# MILP	time	# opt	time	opt	# MILP	time	# opt
30	2	0.0	**10**	0.6	0.0	**10**	0.1	**10**	1.8	0.1	**10**
	4	0.0	**10**	1.4	0.0	**10**	0.1	**10**	1.4	0.1	**10**
	6	0.0	**10**	1	0.0	**10**	0.1	**10**	2.2	0.1	**10**
	8	0.0	**10**	1.1	0.0	**10**	0.1	**10**	1.9	0.0	**10**
100	2	0.2	**10**	2.2	0.2	**10**	1.6	**10**	1.2	0.9	**10**
	4	0.2	**10**	1.6	0.1	**10**	8.5	**10**	4.8	1.3	**10**
	6	0.1	**10**	1.6	0.1	**10**	4.0	**10**	2.9	0.8	**10**
	8	0.1	**10**	1.5	0.1	**10**	1.8	**10**	3.6	0.5	**10**
	16	0.1	**10**	1.2	0.0	**10**	0.5	**10**	1.8	0.2	**10**
	30	0.0	**10**	1	0.0	**10**	0.2	**10**	1.2	0.1	**10**
400	2	8.3	**10**	3.1	3.0	**10**	199.7	**10**	4.1	42.3	**10**
	4	6.5	**10**	2.3	1.3	**10**	1374.0	7	3.5	31.3	**10**
	6	4.8	**10**	2.2	1.2	**10**	725.0	9	3.2	15.8	**10**
	8	2.9	**10**	1.6	0.7	**10**	527.4	**10**	2.7	26.3	**10**
	16	0.8	**10**	1.2	0.3	**10**	243.7	**10**	3.2	18.4	**10**
	30	0.4	**10**	1.3	0.2	**10**	33.9	**10**	1.6	2.2	**10**
700	2	18.0	**10**	1.9	6.6	**10**	1523.9	8	5	135.9	**10**
	4	18.1	**10**	2.9	5.7	**10**	2707.4	5	2.9	132.5	**10**
	6	12.7	**10**	1.3	2.2	**10**	2553.6	6	2	82.5	**10**
	8	11.8	**10**	1.3	1.7	**10**	2891.4	5	1.3	93.8	**10**
	16	3.9	**10**	1.2	0.8	**10**	1873.8	7	1.4	30.6	**10**
	30	1.0	**10**	1	0.4	**10**	1510.0	**10**	1.4	12.9	**10**
1000	2	17.7	**10**	0.2	8.4	**10**	2640.9	7	5	641.0	**10**
	4	24.8	**10**	1.9	5.7	**10**	3610.0	0	2.7	564.3	**10**
	6	20.4	**10**	1.3	5.1	**10**	3607.2	0	2.2	414.2	**10**
	8	29.0	**10**	1.1	2.7	**10**	3605.3	0	1	115.0	**10**
	16	11.6	**10**	1.4	1.4	**10**	3313.6	6	1.1	84.2	**10**
	30	3.7	**10**	1.1	0.7	**10**	2680.1	7	1.1	49.7	**10**
Overall:		7.0	**280**	1.5	1.7	**280**	1272.8	217	2.4	89.2	**280**

$p_{max} = 100$ to proven optimality, whereas the direct solution by Gurobi left many instances unsolved. For many classes, the computational time decreased drastically by the use of our solver. In fact, the average number of times we have to solve a MILP sub-problem is relatively small for all classes, which explains the fast computational performance.

Finally, Table 4.2 compares our results with the results obtained by Kramer et al. [197]. Their experiments were performed by using a single thread on a computer with an Intel Xeon E5530 at 2.40 gigahertz and 20GB RAM. Recall that the method by Kramer et al. [197] also solves the same arc flow model directly by Gurobi, but they use in input an incumbent solution obtained by a sophisticated iterated local search based metaheuristic, which can make a significant difference when compared with our basic greedy heuristic.

Table 4.2: Results summarized by time limit.

time limit	Kramer et al. [197]	Arc Flow	Our Solver
300	440	421	**541**
600	449	434	**552**
900	458	441	**553**
1200	460	452	**554**
1500	472	456	**559**
1800	480	465	**559**
2100	486	473	**559**
2400	490	482	**560**
2700	493	486	**560**
3000	496	490	**560**
3300	499	495	**560**
3600	502	497	**560**

This table presents the total number of instances solved to proven optimality by each method, under different time limits. As expected, the results by Kramer et al. [197] are always better than 'Arc Flow', since it makes use of a more sophisticated initial heuristic. However, already under 300 seconds, our solver could solve more instances to proven optimality than the basic solution by Gurobi under 3600 seconds by both [197] and Arc Flow. Under 1500 seconds, our solver left only one instance unsolved, which was then solved under 2400 seconds.

4.6 Conclusions

In this paper, we proposed a general branching scheme for arc flow models for parallel machine scheduling problems. The branching scheme, which is tailored to be efficient with models with strong relaxations, was applied to an arc flow model for the classical $P||\sum w_j C_j$. The proposed algorithm could solve all hard benchmark instances to proven optimality, closing 58 open instances.

In future research, we aim at applying the proposed scheme to other parallel machine scheduling problems that are associated with arc flow or time-indexed models with strong relaxations. Also, we believe that investigating new general primal cuts for arc flow models can be an interesting research direction to successfully use the proposed scheme to address models with weak LP relaxation.

Chapter 5

Exact Solution Techniques for Two-dimensional Cutting and Packing

5.1 Introduction

The number of publications on cutting and packing problems has been increasing considerably in recent years. In cutting problems, we are given a set of standardized stock units to be cut into smaller *items* so as to fulfill a given demand, while in packing problems a set of items has to be packed into one or more containers. These two classes of problems are strongly correlated, as packing an item into a container may be equivalent to cutting an item from a stock unit, and hence the same solution methods are often adopted. In the following, we will denote as *bins* both the stock units and the containers. In certain applications the unique container is a *strip* of (theoretically) infinite height.

Cutting and packing problems have been widely studied in the literature both for their theoretical interest and their many practical applications, in which they appear in a number of different variants. Some problems consider one-dimensional items and bins, whereas other problems refer to higher dimensions, like, e.g., the (three-dimensional) container loading problem (see Bortfeldt and Wäscher [45]). Items and bins may have rectangular or general (convex or non-convex) shape. Several practical constraints may also be part of the problems, as cargo stability in container loading problems or the need of producing guillotine patterns in cutting problems. In certain applications, cutting and packing problems are combined with other problems, for example in the integrated lot-sizing and cutting stock problems studied by Melega, Araujo, and Jans [237] or in the routing problems with loading constraints considered by Iori and Martello [177]. Packing problems also appear in many other fields, such as telecommunications (Lodi et al. [211], Martello [223]), newspapers paging (Strecker and Hennig [303]), production (Nesello et al. [252]), scheduling (Kwon and Lee [201]), and maritime logistics (Xu and Lee [338]). Most cutting and packing problems are $\mathcal{NP}$-hard and very challenging in practice. For this reason, sophisticated solution methods are needed for their solution, motivating the frequent literature updating of this area of research.

Books and surveys

Several surveys have been dedicated to solution methods for cutting and packing problems. Specifically, referring to surveys published in the last two decades:

- *One-dimensional cutting and packing.* Valério de Carvalho [312] reviews linear programming models for one-dimensional bin packing and cutting stock problems. Approximation algorithms for packing problems generally belong to two main categories: (i) *on-line* algorithms sequentially pack the items in the order encountered on input, without knowledge of items not yet packed; (ii) *off-line* algorithms have complete information about the item set, and may perform preprocessing, reordering, grouping, etc. before packing. Approximation results (for both on-line and off-line algorithms) have been reviewed by Coffman et al. [88] (Sections 3 and 4, respectively). A recent survey on mathematical models and exact algorithms for one-dimensional packing problems was presented by Delorme, Iori, and Martello [109], who also set up a library, the BPPLIB [111], of computer codes and benchmark instances (see `http://or.dei.unibo.it/library/bpplib`). The books by Martello and Toth [226] and Kellerer, Pferschy, and Pisinger [191] present a comprehensive treatment on the knapsack problem and its variants but do not consider the corresponding two-dimensional versions;

- *Two-dimensional rectangular shape cutting and packing.* After the classical 2002 surveys by Lodi, Martello, and Vigo [214] and Lodi, Martello, and Monaci [210], a partial updating was presented by the same authors in 2014 [212]. The following results appeared in the last decade. A survey on guillotine packing was produced by Ntene and van Vuuren [253]. Silva, Oliveira, and Wäscher [299] reviewed exact and heuristic algorithms for a particular problem (pallet loading) in which all items are identical, including a thorough analysis of benchmark instances and methodologies adopted in the literature for numerical experiments. Oliveira et al. [254] presented a review of heuristic algorithms for the strip packing problem. Christensen et al. [74] proposed a survey on approximation and on-line algorithms, also including a list of open problems in this area. Recently, Russo et al. [287] presented a survey on relaxations for two-dimensional cutting problems with guillotine constraints and a categorization of the resulting bounds, while Bezerra et al. [39] reviewed models for the two-dimensional level strip packing problem;

- *Two-dimensional irregular shape cutting and packing.* To the best of our knowledge, the first survey dedicated to the packing of irregular shapes into rectangular containers was presented in the Nineties by Dowsland and Dowsland [116]. More recently, two tutorials by Bennell and Oliveira [33, 34] reviewed the main geometric methodologies and algorithmic approaches for the heuristic solution of these problems. The latest survey on this area, presented by Leao et al. [204] in 2020, provides an extensive review of mathematical models for packing irregular shaped objects both in rectangular and irregular containers;

- *Multi-dimensional cutting and packing.* In 1990, Dyckhoff [119] proposed a typology of cutting and packing problems (in one, two and three dimensions). About twenty

years later, a successful improved typology was proposed by Wäscher, Hauβner, and Schumann [332]. More recently, Bortfeldt and Wäscher [45] considered the (three-dimensional) container loading problem and reviewed modeling approaches, as well as exact and heuristic algorithms. A recent survey on multi-dimensional packing problems was presented by Crainic, Perboli, and Tadei [94];

- *Integrated variants.* Reviews of algorithms for integrated routing and packing problems (with two- and three-dimensional packing constraints) have been presented by Iori and Martello [177, 178] and Pollaris et al. [270]. Melega, de Araujo, and Jans [237] recently reviewed integrated lot-sizing and cutting stock problems.

The constant interest in cutting and packing problems is shown by:

- the trend of number of publications in the last 20 years according to the major databases. A search for titles of papers including specific keywords (like two-dimensional cutting/packing/ knapsack, guillotine, strip packing, two-stage cutting) gave a total 1410, 774, and 575 publications (according to Scholar, Scopus, and WoS, respectively). The historical trend is shown in Figure 5.1;

- a number of special issues devoted to this area by international journals like, e.g., *INFOR* (see Martello [222]), *European Journal of Operational Research* (see Oliveira and Wäscher [255]), *International Journal of Production Economics* (see Bennell, Oliveira, and Wäscher [35]);

- the recent comprehensive book by Scheithauer [292] dedicated to cutting and packing optimization;

- the visual application for two-dimensional packing problems made available by Costa et al. [90];

- the working group on cutting and packing (ESICUP) of the *Association of European Operational Research Societies* (EURO), see `https://www.euro-online.org/web/ewg/25/`.

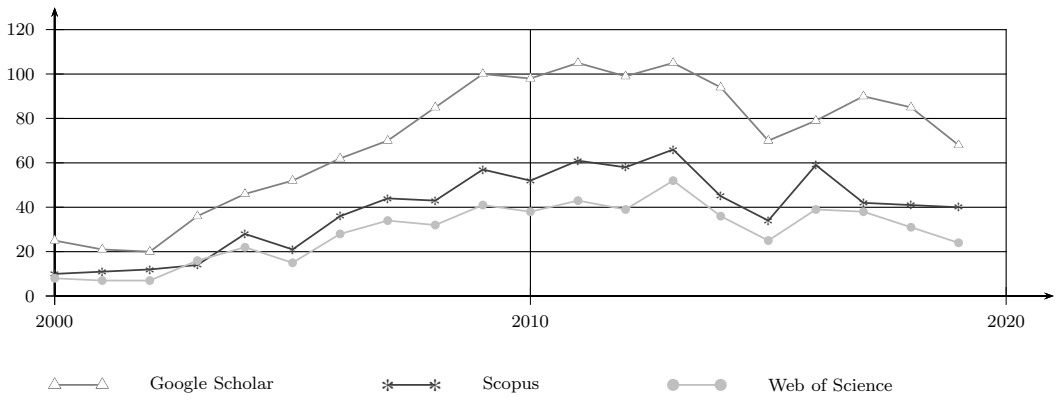

Figure 5.1: Trend of number of publications on two-dimensional cutting and packing.

Contents

In this paper, we propose an extensive review of cutting and packing problems of two-dimensional rectangular items. We concentrate in particular on *orthogonal* cutting and packing, i.e., on the case in which the items must be cut/packed with their edges parallel to those of the bin. Some authors include the term "geometric" in the problem names. The main problems that we address are:

- the two-dimensional strip packing problem: find a packing of minimum height into a single bin with fixed width;
- the two-dimensional bin packing problem: determine the minimum number of bins needed to pack the items;
- the two-dimensional knapsack problem: find a packing of maximum value into a single bin;
- the two-dimensional orthogonal packing problem: find a feasible item packing (if any) into a single bin.

We also consider some generalizations of these problems, such as the cutting stock problem, as well as relevant variants, e.g., the case in which the items can be rotated by 90 degrees or guillotine cuts are required. We review the main relaxation methods, along with the main heuristics and exact methods.

The literature often considers formulations for two-dimensional cutting and packing problems which are based on a set of points where an item may be packed or cut. We will discuss the main methods for generating such sets of points. Another contribution of our work is the review of preprocessing methods for cutting and packing problems, i.e., methods that are considerably fast, if compared to the overall solution time, and may reduce the size of the instances, thus simplifying their resolution.

The remainder of this survey is organized as follows. Section 5.2 formally introduces the basic problems we address and the main variants considered in the literature. Section 5.3 reviews the principal techniques that can be used, in a preprocessing phase, to simplify a problem instance. Section 5.4 discusses relaxation methods that provide valid bounds and, in some cases, are the starting point to derive a heuristic solution. Section 5.5 briefly mentions the main heuristic techniques that are frequently embedded into exact approaches (although the scope of this survey is not to extensively review the huge literature on this topic). Section 5.6 examines mathematical models that explicitly require a MILP solver, by classifying them according to their size (polynomial, pseudo-polynomial, or exponential). Section 5.7 reviews enumeration schemes that do not explicitly make use of a solver (branch-and-bound, graph-based approaches, and constraint programming). Section 5.8 provides pointers to relevant unsolved or challenging instances. Finally, Section 7.5 discusses the state-of-the-art for each problem and presents some conclusions and future research directions.

5.2 Problems and Definitions

In this section, we present a formal definition of the main problems and variants we consider. Given a bin and a set of items, a *packing* consists of a placement of the items such that the items lie completely inside the bin and there is no overlapping between any pair of items.

We are mainly interested in two-dimensional packing problems, in which the bins and the items are rectangles. Only orthogonal packings will be considered, i.e., we require that all edges of all items are parallel to the edges of the bin. Unless otherwise specified, we assume that the items have *fixed orientation*: as most of the literature considers this case, we will only address as a variant the case in which the items can be rotated by 90 degrees.

A rectangular *two-dimensional bin* $\mathcal{B}$ is defined by its *width* $W \in \mathbb{Z}_+$ and *height* $H \in \mathbb{Z}_+$. We denote by $\mathcal{I}$ a set of rectangular two-dimensional *items*. Each item $i \in \mathcal{I}$ has *width* $w_i \in \mathbb{Z}_+$ and *height* $h_i \in \mathbb{Z}_+$, such that $0 < w_i \leq W$ and $0 < h_i \leq H$. Unless otherwise specified, copies of identical items are treated as distinct items. A *packing* of $\mathcal{I}$ into $\mathcal{B}$ can be represented by a function $\mathcal{F} : \mathcal{I} \to \mathbb{Z}_+^2$ that maps each item $i \in \mathcal{I}$ to a pair $\mathcal{F}(i) = (x_i, y_i)$ representing the relative coordinates of the bottom-left corner of the item with respect to the bottom-left corner of the bin. $\mathcal{F}$ is a *feasible packing* if

$$x_i \in \{0, ..., W - w_i\} \text{ and } y_i \in \{0, ..., H - h_i\} \qquad (i \in \mathcal{I}) \quad (5.1)$$

$$[x_i, x_i + w_i) \cap [x_j, x_j + w_j) = \emptyset \text{ or } [y_i, y_i + h_i) \cap [y_j, y_j + h_j) = \emptyset \quad (i, j \in \mathcal{I}, i \neq j). \quad (5.2)$$

In other words, the bin is seen on a Cartesian plane with its edges parallel to the x and y axes and its bottom-left corner on the origin. A packing defines, for every item i, the coordinates (x_i, y_i) where its bottom-left corner is placed. Constraints (6.1) impose that each item is entirely inside the bin, while constraints (6.2) forbid overlapping between any pair of items.

5.2.1 Problems

In this section, we define the main two-dimensional orthogonal packing problems we consider. As mentioned in Section 6.1, some typologies have been proposed in the literature to classify the great variety of possible packing problems. As we will mainly deal with the four problems defined below, we preferred to use the simple names we provide. For the sake of completeness, their definition is followed, in Section 5.2.2, by the corresponding notation in the main typologies. A simple visual example of a set $\mathcal{I}$ containing 10 items is given in Figure 5.2(a), and then used to clarify the main problem variants below.

Two-Dimensional Strip Packing Problem

We are given a single bin $\mathcal{B}$ having fixed width W and infinite height, usually called a *strip*. The *Two-Dimensional Strip Packing Problem* (2D-SPP) asks for a feasible packing of $\mathcal{I}$ into $\mathcal{B}$ such that the height at which the strip is used (i.e., the height of the topmost

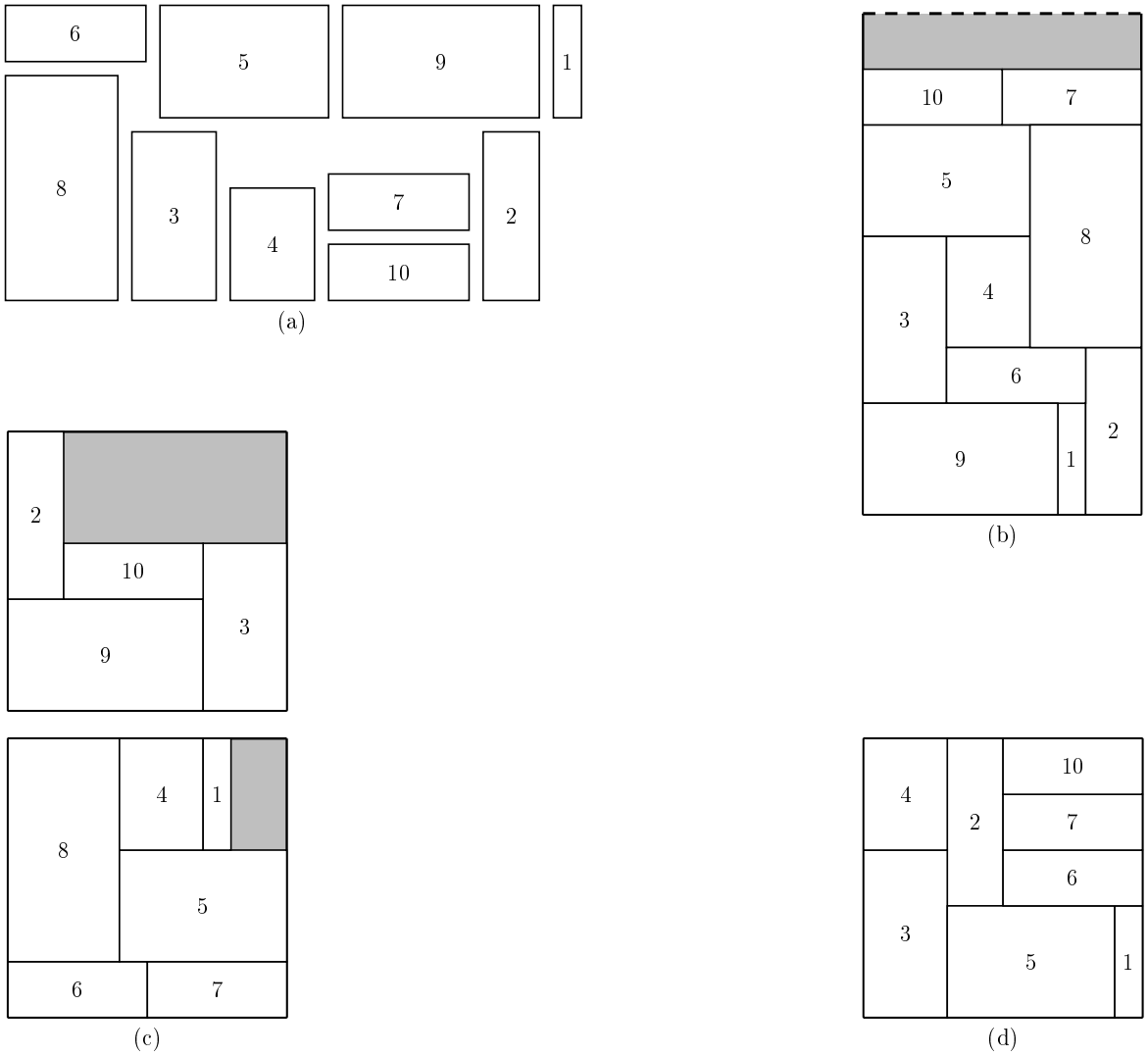

Figure 5.2: (a) set of items; (b) an optimal 2D-SPP solution; (c) an optimal 2D-BPP solution; (d) an optimal 2D-KP solution (in case items profits correspond to their areas).

edge of an item) is minimized. Figure 5.2(b) shows a minimum height arrangement of the items of Figure 5.2(a) into a strip.

Two-Dimensional Bin Packing Problem

In this case we have an unlimited number of identical finite bins $\mathcal{B}$ having width W and height H. The *Two-Dimensional Bin Packing Problem* (2D-BPP) requires to determine a partition of $\mathcal{I}$ into the minimum number of subsets, such that each subset can be feasibly packed into a bin. A generalization of the problem is the *Two-Dimensional Cutting Stock Problem* (2D-CSP), in which one is asked to pack a given number d_i (*demand*) of each

item $i \in \mathcal{I}$. Both problems generalize their one-dimensional version, the well-known (one-dimensional) *Bin Packing Problem* (1D-BPP) and *Cutting Stock Problem* (1D-CSP), in which the items are segments of size w_i ($i \in \mathcal{I}$) and the bins are segments of size W (*capacity*). An extensive literature exists on these problems, see [109] for a recent survey. Figure 5.2(c) shows an optimal 2D-BPP solution, in which the items of Figure 5.2(a) are packed into two separate bins.

Two-Dimensional Knapsack Problem

The problems listed so far ask for a feasible packing of *all* items of $\mathcal{I}$. Assume now that every item $i \in \mathcal{I}$ has an associated value (*profit*) $v_i \in \mathbb{Z}_+$. The *Two-Dimensional Knapsack Problem* (2D-KP) requires to determine a subset of items $\mathcal{I}' \subseteq \mathcal{I}$ such that: (i) there exists a feasible packing of $\mathcal{I}'$ into a single bin $\mathcal{B}$; and (ii) the corresponding total profit, $\sum_{i \in \mathcal{I}'} v_i$, is maximized. A special case of the 2D-KP, the *Two-Dimensional Rectangular Packing Problem*, arises when the profit of each item is equal to its area (i.e., $v_i = w_i h_i \; \forall \; i \in \mathcal{I}$). An example of an optimal two-dimensional rectangular packing problem solution for the items of Figure 5.2(a) is given in Figure 5.2(d).

Two-Dimensional Orthogonal Packing Problem

While the above problems are in *optimization version*, the *Two-Dimensional Orthogonal Packing Problem* (2D-OPP) is to decide if there exists a feasible packing of a given set $\mathcal{I}$ of items into a single bin $\mathcal{B}$. Although this problem is in *decision version*, i.e., it just asks for a 'yes/no' answer, in practical applications one is generally requested to also produce the specific packing, if any. The 2D-OPP appears as a subproblem in the optimization version of a number of packing problems.

5.2.2 Typologies

As previously mentioned, a number of typologies have been introduced in the literature. As the reader could encounter the same problems considered in the present survey but identified in a different way, we next provide the notations adopted by the most common typologies from the literature.

According to the typology proposed by Dyckhoff [119] in 1990, the 2D-BPP, the 2D-CSP and the 2D-KP are denoted as 2/V/I/M, 2/V/I/R, and 2/B/O/, respectively. The 2D-SPP and the 2D-OPP have not been formally defined in this typology.

In 1999 Lodi, Martello, and Vigo [213] used a three-field typology (later extended by Martello, Monaci, and Vigo [225]). The 2D-SPP, the 2D-BPP, and the 2D-KP are denoted as 2SP|O|F, 2BP|O|F, and 2KP|O|F, respectively, while the 2D-OPP is not classified. The second field of this notation also covers the variant (see below) in which orthogonal rotation of the items is allowed ('R' instead of 'O'), while the third field can handle the variant in which guillotine cuts are required ('G' instead of 'F').

More recently (2007), Wäscher, Hauβner, and Schumann [332] proposed a successful typology, partially based on Dyckhoff's original ideas. According to it, the 2D-SPP is denoted as the "(Two-dimensional) Open Dimension Problem" (ODP), the 2D-BPP as

the "Single Bin Size Bin Packing Problem" (SBSBPP), and the 2D-CSP as the "Single Stock Size Cutting Stock Problem" (SSSCSP). The 2D-KP is denoted either as the "Single Knapsack Problem" (SKP), or as the "Single Large Object Placement Problem" (SLOPP) in the generalization in which each item $i \in \mathcal{I}$ is available in d_i copies. The 2D-OPP is not explicitly classified in [332], but it can be interpreted as the recognition version of the 2D-BPP.

5.2.3 Complexity

The *one-dimensional* version of the 2D-KP is the well-known *knapsack problem* (1D-KP) in which each item has a *weight* w_i and the bin (*knapsack*) has capacity W. The 1D-KP can be solved in pseudo-polynomial time through dynamic programming (see, e.g., Martello and Toth [226] or Kellerer, Pferschy, and Pisinger [191]). Instead, none of the above two-dimensional problems admits a pseudo-polynomial time algorithm, unless $\mathcal{P} = \mathcal{NP}$. Recall indeed that the 1D-BPP is strongly $\mathcal{NP}$-hard (see Garey and Johnson [144]). Given an instance of the 1D-BPP, define two-dimensional items having width w_i and height $h_i = 1$ ($i \in \mathcal{I}$). Then:

- the solution of a 2D-SPP instance with strip width W solves the 1D-BPP instance;
- the solution of a 2D-BPP instance with bins of width W and height 1 solves the 1D-BPP instance;
- associate a profit $v_i = 1$ to each item $i \in \mathcal{I}$. The minimum value $\bar{H}$ for which the optimal solution of a 2D-KP instance with a bin of width W and height $\bar{H}$ has value n gives the optimal 1D-BPP solution value;
- the solution of a 2D-OPP instance with a bin of width W and height k answers the decision version of the 1D-BPP: can the items of $\mathcal{I}$ be packed into k bins?

It follows that the existence of a polynomial-time algorithm for any of the four problems above would imply a polynomial-time algorithm for the 1D-BPP, i.e., the 2D-SPP, the 2D-BPP, and the 2D-KP are strongly $\mathcal{NP}$-hard, and the 2D-OPP is strongly $\mathcal{NP}$-complete.

5.2.4 Variants

Motivated by practical applications, a number of variants of two-dimensional packing problems have been considered in the literature (see, e.g., Lodi, Martello, and Vigo [213], Pisinger and Sigurd [267], and Wäscher, Hauβner, and Schumann [332]).

Orthogonal Rotation

While in the basic problems the items have fixed orientation, relevant variants allow item *rotation* by 90 degrees. For example, while in the cutting of corrugated or decorated stock units rotation is forbidden, when the surfaces are uniform it may be feasible to rotate the items in order to produce better (more dense) packings. Note however that this also leads to more complex problems as it increases the number of decisions to be

taken, and hence the corresponding models usually involve a higher number of variables and constraints. Indeed, in most approaches, item rotation is typically handled either by adding, for each item, a "rotate-or-not" binary decision (like, e.g., in Jakobs [185]), or by creating a companion (rotated) copy of each item and forbidding that both are selected (like, e.g., in Lodi and Monaci [217]).

Guillotine Cuts

Normally, automatic cutting machines can only produce the items through a sequence of *guillotine cuts*, i.e., edge-to-edge cuts parallel to the edges of the bin. Imposing such constraint may lead to worse solutions as not all item sets allow guillotine patterns in a bin (see Figure 5.3(a)). Frequently, the machines are restricted to only alternate horizontal and vertical cuts, possibly with a hard limit k on the number of cuts per bin (k-*staged* problems). In most applications k is two or three, possibly allowing an extra cut (called *trimming*) to separate an item from waste (see Figure 5.3(b) and (c)). When $k = 2$, the problem is frequently called *level packing*, as it can be seen as the problem of packing the items side-by-side on horizontal shelves having width equal to that of the bin/strip and height coinciding with the tallest packed item (see again Figure 5.3(b)). When $k = 3$, Puchinger and Raidl [272] distinguish between the case where the latter condition holds (*restricted case*) and the one where the height of a shelf is not necessarily given by its highest item (*unrestricted case*). The presence of guillotine constraints consistently affects the combinatorial structure of the problem, and hence the solution techniques. Indeed, for such cases, it is common to adopt techniques based on column generation and/or dynamic programming, that would not be suitable for standard non-guillotine problems.

Variable-Sized Bins

The bin packing and cutting stock problems consider an unlimited number of identical bins. The *variable-sized* generalization of these problems deals instead with different types of bins, each having a specific size (width and height), cost, and availability. The problem is then to pack all the items at minimum cost. The addition of variable sizes is usually handled in the solution techniques by considering, e.g., the different sizes of the bins when creating new nodes in enumerative approaches, or when pricing variables in branch-

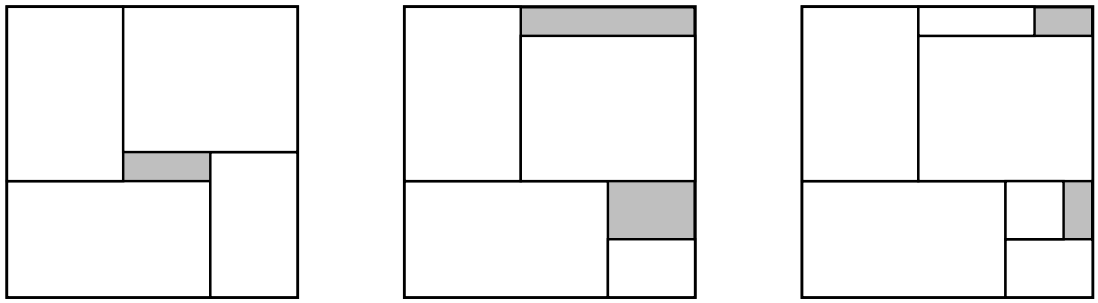

Figure 5.3: (a) non-guillotine pattern; (b) 2-staged guillotine pattern with trimming; (c) 3-staged guillotine pattern with trimming.

and-price methods (as, e.g., in Pisinger and Sigurd [267]). This comes at the cost of an additional computational effort, and indeed, from a practical point of view, the solution of variable-sized problems is typically more challenging.

Loading and Unloading Constraints

Loading and/or unloading constraints often arise in applications where goods pertaining to different customers have to be loaded into the same bin, which represents, e.g., the loading area of a truck (see, e.g., Gendreau et al. [145]). In these cases, the vehicles have a loading/unloading orientation, and the sequence of visit to the customers must be such that each item may be moved in/out without moving any other item. These constraints can be directly included in mathematical models where the variables indicate the position of the items (see Section 5.6.2) as the packing of an item restricts the range of positions that can be taken by other items (as, e.g., in Côté, Gendreau and Potvin [92]). They can also be imposed within branch-and-bound algorithms, where additional fathoming criteria can be devised to reduce the number of decision nodes (as, e.g., in Iori, Salazar González and Vigo [180]).

The first two variants were already discussed in the Sixties by Gilmore and Gomory [148]. In the Eighties, Friesen and Langston [137] introduced (for the one-dimensional case) the variable-size variant. The studies on loading and unloading constraints started in the Noughties, see Gendreau et al. [145].

5.3 Sets of Points and Preprocessing Techniques

We start this section by reviewing techniques based on sets of points, that can be used in both *Integer Linear Programming* (ILP) models and solution methods reviewed in the following sections. We then examine some preprocessing techniques, used to decrease the size of a given instance.

5.3.1 Sets of Points

Several authors in the literature use *sets of points* to represent the possible positions where an item can be packed. According to the definitions given in Section 5.2, packing an item at a point $p = (x, y)$ means to allocate the item into the bin with its bottom-left corner in the position identified by p.

As we assume that items and bins have integer sizes, only considering integer point coordinates does not affect optimality. Trivial sets of points for an item i are thus given by coordinate sets $X_i = \{0, \ldots, W - w_i\}$ for the x-axis and $Y_i = \{0, \ldots, H - h_i\}$ for the y-axis.

It is clear that the smaller the number of points the smaller the search domain. We review in the following the main techniques that have been proposed in the literature for generating reduced sets of points by preserving optimality.

Already in the Seventies, Herz [166] and Christofides and Whitlock [77] independently observed that any feasible packing pattern can be transformed into an equivalent one (*normal pattern*) in which the items are shifted to the bottom and left as much as possible. In a normal pattern, the left and bottom edges of each item touch another item or the bin. This leads to the definition of the set of "normal" x-coordinates

$$\mathcal{N}^{\mathrm{x}} = \{x \,|\, x < W,\, x = \sum_{i \in \mathcal{I}} w_i \zeta_i,\, \zeta_i \in \{0,1\} (i \in \mathcal{I})\}, \tag{5.3}$$

defining the only x-coordinates at which an item can be packed. A similar definition holds for the set $\mathcal{N}^{\mathrm{y}}$ of normal y-coordinates.

Boschetti, Hadjiconstantinou, and Mingozzi [46] proposed to separately compute a set of normal patterns for each item $k \in \mathcal{I}$ by excluding those patterns that include k itself. The resulting set $\mathcal{N}^{\mathrm{x}}_k$ ($k \in \mathcal{I}$) is obtained from (5.3) by replacing $\mathcal{I}$ with $\mathcal{I} \setminus \{k\}$. (Similarly for $\mathcal{N}^{\mathrm{y}}_k$.)

Terno, Lindemann, and Scheithauer [304] proposed the use of *reduced raster points*, obtained by removing redundant positions from the set of normal patterns. The idea is that, given two coordinates $p, q \in \mathcal{N}^{\mathrm{x}}$ ($p < q$), if every possible combination of items that can be packed to the right of p can also be packed to the right of q, then p can be removed from $\mathcal{N}^{\mathrm{x}}$.

Recently, Côté and Iori [93] proposed a new set of patterns called *meet-in-the-middle*, generated by first defining a threshold $t \in \{1, 2, \ldots, W\}$ and then left-aligning patterns that are to the left of t and right-aligning those that are to its right. The resulting set of patterns is proved to be never larger than $\mathcal{N}^{\mathrm{x}}$, and in practice is usually much smaller.

The fact that normal patterns preserve optimality was proved in the seminal paper by Herz [166]. Proofs were later provided for other types of patterns like, e.g., in Côté and Iori [93]. All such proofs show that any solution in which the items are packed without restrictions in the continuous two-dimensional space can be transformed, through simple translations, into a better or equivalent solution in which the items are packed according to the patterns. The same arguments apply to problem variants like, e.g., those involving guillotine cuts or several item copies or item rotation.

5.3.2 Preprocessing Techniques

Two main approaches for preprocessing the instances by preserving optimality can be found in the literature: methods that fix some decisions and methods that modify the input parameters. These techniques are useful for improving bounds based on the area of the items or of the bins. We describe in the following one method of the former type and two of the latter. Although developed for specific problems, the methods of the latter type can be directly used for all problems. The extension of methods of the former type is instead not straightforward.

All methods will be presented referring to widths. Those developed for problems on bins can be identically applied using heights instead of widths, while the same does not hold for problems on strips. The sequence of presentation follows the order in which it is advisable to execute these methods in order to effectively reduce the given instance.

Packing a Subset of Items

This reduction was originally proposed by Martello, Monaci, and Vigo [225] for the 2D-SPP. Suppose that the items are sorted by non-increasing width. The method can be applied if $w_1 > W/2$. Let $B = \{ i \in \mathcal{I} \mid w_i = w_1 \}$. Observe that these items cannot be packed side-by-side, so we can pile them aligned to the left edge of the strip. Now define the set of those items that can be packed side-by-side with an item of B, namely $S = \{ j \mid w_j \leq W - w_1 \}$. If there exists a feasible packing of all the items of S into the empty right part of the strip, with overall height not greater than $\sum_{i \in B} h_i$, then this packing can be optimally fixed on the bottom of the strip, and the process can be iterated on the reduced instance. When the required packing is not found, the method finds the first item $\ell \notin B$. If $w_\ell > W/2$, the current set B is updated by adding the items of width w_ℓ, the current set S is updated accordingly, and a new attempt is performed.

This reduction was later used within other solution methods for the 2D-SPP (see, e.g., Alvarez-Valdes, Parreño, and Tamarit [7], Boschetti and Montaletti [49] and Côté, Dell'Amico, and Iori [91]) and extended to other packing problems, among which the 2D-BPP (see, e.g., Carlier, Clautiaux, and Moukrim [82]).

Shrinking the Size of the Bin

Alvarez-Valdes, Parreño, and Tamarit [7] proposed to solve a (one-dimensional) *subset-sum problem* (find a subset of a set of given integers whose sum is closest to, without exceeding, a prefixed threshold) to determine the maximum value $\overline{W} \leq W$ such that there exists a sum of item widths equal to $\overline{W}$. If $\overline{W} < W$ then the width of the bin/strip can be reduced to $\overline{W}$.

Lifting the Size of the Items

A similar approach had been used by Boschetti, Hadjiconstantinou, and Mingozzi [46] to increase item widths. They proposed to solve, for each item $i \in \mathcal{I}$, a subset-sum problem to determine the maximum value $\overline{W}_i \leq W - w_i$ such that there exists a set of items in $\mathcal{I} \setminus \{i\}$ with total width equal to $\overline{W}_i$. If $w_i + \overline{W}_i < W$ then the width of i can be increased by $W - \overline{W}_i$. Carlier, Clautiaux, and Moukrim [82] defined similar methods for updating the size of the items by removing small items and increasing the width of large items.

5.4 Relaxations

Several relaxation methods for two-dimensional packing problems have been proposed in the literature. They are used within exact algorithms and, in some cases, as a base to construct a heuristic solution. Obviously, relaxations provide lower bounds for the 2D-SPP and the 2D-BPP and upper bounds for the 2D-KP. As the 2D-OPP is a decision problem, relaxations can be used to prove that the required packing does not exist.

5.4.1 Continuous Relaxation

Splitting each item i into $w_i \times h_i$ unit squares produces the most immediate relaxation for all considered problems. For the 2D-OPP, the 2D-SPP and the 2D-BPP, the *continuous lower bound* is then

$$\left\lceil \sum_{i \in \mathcal{I}} \frac{w_i h_i}{WH} \right\rceil \tag{5.4}$$

(computed with $H = 1$ for the 2D-SPP). For the 2D-KP, the *continuous upper bound* is obtained, following Dantzig [99] by: (i) sorting the items by non-increasing ratios $v_i/(w_i h_i)$; (ii) finding the first item s such that $\sum_{i=1}^{s} w_i h_i > WH$ and defining $c = WH - \sum_{i=1}^{s-1} w_i h_i$; (iii) computing the upper bound as

$$\left\lfloor \sum_{i=1}^{s-1} v_i + \frac{c}{w_s h_s} v_s \right\rfloor . \tag{5.5}$$

Efficient implementations allow all these bounds to be computed in linear time. Because of its simplicity, the continuous relaxation has been used in almost all works on two-dimensional packing problems.

5.4.2 Combinatorial Bounds

Martello and Vigo [228] extended to the 2D-BPP some lower bounds proposed by Martello and Toth [227] for the 1D-BPP and by Dell'Amico and Martello [106] for the $P||C_{\max}$, a parallel machine scheduling problem that is strictly related to the 1D-BPP. The idea is to identify, for a given parameter p: (i) two item sets, say $J_1(p)$ and $J_2(p)$, such that no two items of $J_1(p) \cup J_2(p)$ may be packed into the same bin; (ii) a third set, $J_3(p)$, of items that cannot be packed into a bin used for an item of $J_1(p)$. A valid lower bound is then $L(p) = |J_1(p) \cup J_2(p)| + L_{23}(p)$, where $L_{23}(p)$ denotes a lower bound on the number of additional bins needed for the items of $J_3(p)$. It is shown that the overall lower bound, $\max_p\{L(p)\}$ can be computed in $O(n^2)$ time.

Boschetti and Mingozzi [47, 48] improved these bounds and extended them to the variant in which orthogonal rotation of the items is allowed. Similarly, relaxations for the 2D-KP can be obtained from induced 1D-KP instances where each item has a weight equal to its area and the knapsack has capacity WH.

Although combinatorial bounds have a low computational cost, they can be quite effective, especially in branch-and-bound approaches (see, e.g., Martello, Monaci, and Vigo [225] and Alvarez-Valdes, Parreño and Tamarit [7]).

5.4.3 Linear Relaxation and Column Generation

Already in 1965, the seminal paper by Gilmore and Gomory [148] introduced a column generation algorithm for two-dimensional packing problems. The algorithm is based on the linear relaxation of a mathematical formulation (the so-called *set covering model*) that they had developed for the 1D-CSP (see Section 5.2.1). The model has a variable for each possible combination of items that fits into a single bin (*pattern*). Since the resulting

Linear Program (LP) has a huge (exponential) number of *columns* (variables), the *Column Generation* method starts with a restricted model that only includes a subset of columns and iteratively adds columns that may improve the current solution. This relaxation has been used in branch-and-price algorithms for the 2D-BPP and related variants (see, e.g., Pisinger and Sigurd [267, 268]), and in problems involving guillotine cuts (see, e.g., Belov and Scheithauer [27], Bettinelli, Ceselli, and Righini [38], and Cintra et al. [78]).

5.4.4 Dual Feasible Functions

Dual feasible functions were originally introduced by Johnson [187] for a normalized version of the 1D-BPP in which the bin capacity is 1 and the item sizes w_i are values in $[0, 1]$.

A function $f : [0, 1] \to [0, 1]$ is a *Dual Feasible Function* (DFF), if for any finite set S of non-negative real numbers, it holds that:

$$\sum_{x \in S} x \leq 1 \implies \sum_{x \in S} f(x) \leq 1. \tag{5.6}$$

Three classes of DFFs were proposed by Fekete and Schepers [125]. The survey by Clautiaux, Alves, and Valério de Carvalho [80] describes relevant (superadditive and maximal) DFFs that can be used in packing contexts. Rietz, Alves, and Valério de Carvalho [281, 282, 283], analyzed a number of properties of DFFs (maximality, extremality, worst-case performance) and studied the best parameter tuning for a specific function.

Dual feasible functions can be extended to consider discrete domains and used to derive relaxations for higher-dimensional cutting and packing problems (see Fekete and Schepers [127]). For the 2-dimensional case, let f and g be two DFFs. Applying f to the item widths and g to the item heights of an instance of any two-dimensional packing problem, we obtain a new instance whose relaxations are valid for the original instance.

Carlier, Clautiaux, and Moukrim [82] proposed DFF-based lower bounds for the 2D-BPP and introduced DFFs depending on the sizes of items and bins (*data-dependent* DFFs). Caprara and Monaci [67] solved a bilinear programming problem to determine DFFs that provide the best bound for a given two-dimensional instance.

Fekete and Schepers [127] extended the concept of DFF to that of conservative scale. A *conservative scale* is a set of modified item sizes $\widetilde{w}$ (resp. $\widetilde{h}$) such that any subset $\mathcal{I}'$ of items satisfying $\sum_{i \in \mathcal{I}'} w_i \leq W$ (resp. $\sum_{i \in \mathcal{I}'} h_i \leq H$) also satisfies $\sum_{i \in \mathcal{I}'} \widetilde{w}_i \leq W$ (resp. $\sum_{i \in \mathcal{I}'} \widetilde{h}_i \leq H$). Note that a conservative scale is instance dependent while dual feasible functions have general validity. The relationship between (data-dependent) dual feasible functions and conservative scales has been deeply investigated by Belov et al. [25].

Recently, Serairi and Haouari [294] presented a theoretical and experimental study of the most important polynomial-time lower bounding procedures for the 2D-BPP. Among the algorithms they tested, the one by Carlier, Clautiaux, and Moukrim [82] turned out to provide the tightest values.

For a comprehensive study on the use of dual feasible functions for integer programming and combinatorial optimization problems the reader is referred to the recent book by Alves et al. [8]. Due to their computational efficiency, dual-feasible functions are often

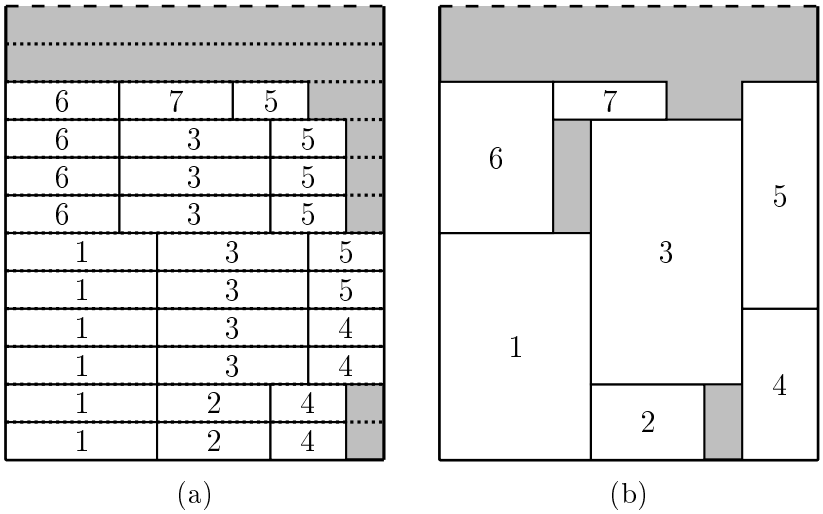

Figure 5.4: (a) contiguous relaxation of items 1-7; (b) two-dimensional packing obtained from (a).

implemented in branch-and-bound algorithms (see, e.g., Clautiaux, Carlier, and Moukrin [82], and Alvarez-Valdes, Parreño and Tamarit [7]).

5.4.5 Contiguous One-Dimensional Relaxations

In order to obtain a relaxation of the 2D-SPP, Martello, Monaci, and Vigo [225] introduced the *One-dimensional Contiguous Bin Packing Problem* (1D-CBPP). The idea is to horizontally "cut" each item $i \in \mathcal{I}$ into h_i unit-height *slices* of width w_i and to solve a 1D-BPP problem with capacity W: the resulting number of one-dimensional bins is then a lower bound on the height of any 2D-SPP solution. To tighten the bound, the 1D-CBPP additionally imposes *contiguity*: all slices obtained from an item i must be packed into h_i *consecutive* one-dimensional bins (see Figure 5.4(a)). Although the 1D-CBPP is a relaxation of the 2D-SPP, it remains strongly $\mathcal{NP}$-hard and hence it can only be optimally solved through enumeration. A branch-and-bound algorithm was presented in [225] for its exact solution and the resulting lower bound was used in an enumerative algorithm for the 2D-SPP. Later, Alvarez-Valdes, Parreño, and Tamarit [7] proposed an ILP model for the 1D-CBPP, and used a commercial software to solve it within a branch-and-bound algorithm for the 2D-SPP.

Belov et al. [24] used the 1D-CBPP as a relaxation of the 2D-OPP. They proposed an alternative ILP model for the 1D-CBPP and a weaker relaxation in which, instead of requiring contiguity, it is imposed that each one-dimensional bin can contain at most one slice from each item (*bar relaxation*). Friedow and Scheithauer [136] solved the 1D-CBPP with a cutting plane approach based on column generation for the bar relaxation. A branch-and-bound algorithm for the 1D-CBPP was later proposed by Mesyagutov et al. [240].

Côté, Dell'Amico, and Iori [91] proposed an algorithm for the 2D-SPP based on Benders' decomposition (see Section 5.6.3 below) of an ILP model. The resulting master problem, obtained by "transposing" the instance and vertically cutting the items into

unit-width bars, is a parallel processor scheduling problem with contiguity constraints. The slave problem, which is still strongly $\mathcal{NP}$-hard, determines whether the solution provided by the master can be transformed into a feasible packing (see Figure 5.4(b)). Delorme, Iori, and Martello [110] applied a similar decomposition to solve the 2D-SPP with orthogonal rotations. They proposed an ILP model for the 1D-CBPP based on the classical arc-flow formulation of the 1D-CSP (see Valério de Carvalho [314]).

5.4.6 State Space Relaxations

The state space relaxation, originally developed by Christofides, Mingozzi, and Toth [76] for routing problems, modifies the state space of a dynamic programming recursion so that its optimal solution is a valid bound for the original problem. This relaxation has been used for the (strongly NP-hard) guillotine 2D-KP, for which a state space relaxation can disregard the constraint on the maximum number of copies of each item. The relaxed problem (known in the literature as the *unconstrained two-dimensional guillotine knapsack problem*) can be solved in pseudo-polynomial time through dynamic programming (see, e.g., Gilmore and Gomory [148], Beasley [18], Cintra et al. [78] and Russo et al. [289]).

Christofides and Hadjiconstantinou [75] strengthened the state space relaxation by associating a non-negative integer multiplier with each item and adopted a subgradient-like procedure to determine good multipliers. Improvements were proposed by Morabito and Pureza [245] and by Velasco and Uchoa [325].

5.5 Heuristics

There is a huge literature on approximation algorithms and heuristics for two-dimensional packing problems, and a thorough review of these methods is outside the scope of this survey. To get an idea of the vastity of this area, it is enough to consider that Ortmann and van Vuuren [258] computationally compared a total of 252 heuristics and variants for two-dimensional strip packing problems. As heuristics are frequently used within exact algorithms (e.g., to initialize the incumbent solution), we provide in the following some pointers to a number of relevant results.

5.5.1 Approximation and On-line Algorithms

Concerning approximation and on-line algorithms, we refer the reader to the recent (2017) comprehensive survey by Christensen et al. [74], that gives theoretical insight on fast solution methods characterized by relevant theoretical properties for multi-dimensional packing problems. We only mention here recent relevant works that appeared after the publication of such survey.

Gálvez et al. [143] proposed a polynomial-time 1.89-approximation algorithm for the 2D-KP with orthogonal rotations, breaking the previous 2-approximation barrier.

Yu, Mao, and Xiao [341, 343] studied the *on-line* 2D-SPP, presenting new lower and upper bounds with guaranteed worst-case performance. In [342] they presented an upper bound for the special case of the 2D-SPP with square items. Further improvements were

presented by Han et al. [163] for the 2D-SPP and by Balogh et al. [16] for the special case of the 2D-BPP in which items are squares.

For the 2D-SPP, Henning et al. [165] studied pseudo-polynomial approximation algorithms with respect to the width of the strip, i.e., algorithms whose time complexity is a polynomial function of the strip width. They proved that there cannot exist a pseudo-polynomial algorithm with a ratio better than 5/4 unless $\mathcal{P} = \mathcal{NP}$. Jansen and Rau [186] closed the gap between inapproximability result and best known algorithm by presenting an algorithm with approximation ratio $5/4 + \varepsilon$.

5.5.2 Constructive Heuristics, Local Search, and Metaheuristics

A huge number of constructive heuristics for two-dimensional cutting and packing problems can be found in the literature. The reader is referred to Lodi et al. [212] for classical results in this area. We only mention here two strategies that have been the basis of many solution approaches. The classical *bottom-left algorithm*, proposed in 1980 by Baker, Coffman, and Rivest [14], packs one item at a time in the lowest possible position, left justified. Another classical result (the *best-fit* approach), proposed by Burke, Kendall, and Whitwell [60], selects and packs an item that better fits the lowest available area. (See Chazelle [72] and Imahori and Yagiura [174] for efficient implementations of these methods.)

Local search approaches explore the neighborhood of a given solution. Several methods for two-dimensional packing problems are based on fixing part of the solution, unpacking the other items, and completing the solution according to some strategy. For example, Alvarez-Valdes, Parreño, and Tamarit [5] create "holes" in the current packing and try to use them when completing the solution. Other approaches pack items according to an input sequence, and perform changes in such sequence to obtain alternative solutions (see, e.g., Burke, Hyde, and Kendall [59] and Wei et al. [335]).

Constructive heuristics and local search are the base of many efficient metaheuristic algorithms. We briefly review in the following some results for the two main metaheuristic approaches.

Single-solution Metaheuristics

Alvarez-Valdés, Parajón, and Tamarit [3] proposed a GRASP and a Tabu search for the 2D-KP with guillotine constraints. Alvarez-Valdes, Parreño, and Tamarit [5, 6] presented a Tabu search algorithm for the 2D-KP and a GRASP for the 2D-SPP. Wei et al. [334] proposed an effective Tabu search for the two-dimensional rectangular packing problem (see Section 5.2.1), both with and without orthogonal rotation. They also embedded it in a binary search approach to solve the corresponding versions of the 2D-SPP.

Some of the metaheuristics proposed in the literature also work for the *three-dimensional* generalization of the 2D-BPP, in which items and bins are three-dimensional boxes. For both dimensions, Parreño et al. [259] presented a combination of GRASP and variable neighborhood search, while Lodi, Martello, and Vigo [216] implemented a unified C computer code based on Tabu search (available at `http://or.dei.unibo.it/research_pages/ORcodes/TSpack.html`).

Tabu search heuristics for routing problems with two- and three-dimensional loading/unloading constraints have been proposed by Gendreau et al. [145, 146]

Population-Based Metaheuristics

Concerning the 2D-SPP, Iori, Martello, and Monaci [179] developed a hybrid Tabu search-genetic algorithm. Different genetic approaches were presented by Burke et al. [58], Matayoshi [236], and Borgulya [44].

Genetic algorithms for the 2D-KP were proposed by Hadjiconstantinou and Iori [162] and by Kierkosz and Luczak [193]. Gonçalves and Resende [153] presented a biased random key genetic algorithm for the 2D-BPP and its three-dimensional generalization.

5.5.3 Set covering based heuristics

Monaci and Toth [243] developed algorithms for the 2D-BPP based on the set-covering formulation (see Section 5.4.3) induced by a heuristic generation of the patterns.

Cintra et al. [78] proposed dynamic programming algorithms for the guillotine 2D-KP, both in the k-staged and in the non-staged versions. They then used these algorithms in a column generation heuristic for the corresponding versions of the 2D-CSP and the 2D-SPP. Other heuristics based on column generation for the 2-staged 2D-CSP were presented by Furini et al. [140], by Cui and Zhao [95] and by Cui, Zhou and Cui [96].

For the 3-staged 2D-CSP, a set covering based heuristic was presented by Vanderbeck [322].

5.6 Exact Methods based on Integer Linear Programming Models

In this section we review solution approaches that are based on ILP and *Mixed Integer Linear Programming* (MILP) formulations and hence require the use of a solver. The models may be classified on the basis of their size: polynomial, pseudo-polynomial, or exponential.

5.6.1 Polynomial Models

A mathematical mixed-integer model for a general three-dimensional bin packing problem, involving a polynomial number of variables and constraints, was presented by Chen, Lee, and Shen [73]. (This obviously implies a polynomial model for the two-dimensional classical problems.) The model in [73], that is based on the enumeration of all possible relative placements of each pair of items, can be seen as an extension to the three-dimensional case of the modeling technique proposed by Onodera, Taniguchi, and Tamaru [257] for a two-dimensional block placement problem.

Polynomial formulations for 2-staged guillotine packing were proposed by Lodi and Monaci [217] for the 2D-KP and by Lodi, Martello, and Vigo [215] for the 2D-BPP and the 2D-SPP. These models were later extended to the 3-staged case of the 2D-BPP by

Puchinger and Raidl [272] and to the 2-staged 2D-CSP with variable-sized bins by Furini and Malaguti [139].

5.6.2 Pseudo-polynomial Models

While polynomial models associate variables to the items, pseudo-polynomial models include variables associated with the positions into the bins where the items can be packed.

Beasley [20] proposed an ILP model for the 2D-KP, based on a discretization of the packing area through a pseudo-polynomial number of two-dimensional coordinates (see Section 5.3.1) where the bottom-left corner of an item can be packed. A pseudo-polynomial number of constraints imposes that no unit square is covered by more than one item. This formulation was later used for the solution of other two-dimensional packing problems. Alvarez-Valdes, Parreño, and Tamarit [4] adapted the formulation to the special case of the two-dimensional rectangular packing problem (see Section 5.2.1) with orthogonal rotations in which all items are identical (*Pallet Loading Problem*). de Queiroz and Miyazawa [274] adapted the model to a variant of the 2D-SPP in which the items have to be arranged to form a physically stable packing satisfying a predefined item unloading order. Martello and Monaci [224] used a similar model to solve the problems of orthogonally packing a set of rectangles (with or without allowing rotations) into the smallest square.

Valério de Carvalho [314] presented an *arc-flow* model for the solution of the 1D-CSP. The model is based on a digraph with a pseudo-polynomial number of vertices, in which paths correspond to feasible packings of a bin and a flow provides an overall problem solution. This formulation was extended to the 2-staged guillotine 2D-CSP by Macedo, Alves, and Valério de Carvalho [219] and to the 2-staged guillotine 2D-SPP by Mrad [246].

Another pseudo-polynomial formulation of the 1D-CSP that was further extended to two-dimensional problems is the *one-cut* model, independently obtained by Rao [276] and Dyckhoff [118]. The model is based on variables that describe the feasible cuts that split a bin (or a residual of a bin) into an item and a residual (or another item). Silva, Alvelos, and Valério de Carvalho [296] extended this model to the 2-staged and the 3-staged guillotine 2D-CSP, while Furini, Malaguti, and Thomopulos [141] used it for the non-staged guillotine 2D-SPP, 2D-CSP, and 2D-KP.

Another family of pseudo-polynomial models derives from time representations of scheduling problems where disjunctive constraints determine the relative positions between each pair of items. Following this methodology, Castro and Oliveira [69] developed ILP models for the 2D-SPP, and showed how to extend them to the 2D-BPP and to the 2D-KP. Castro and Grossmann [68] proposed further improvements for the 2D-SPP.

5.6.3 Exponential Models

Most exponential models for two-dimensional packing problems are based either on formulations that associate variables with feasible patterns (see Section 5.4.3), which have an exponential number of variables (columns), or on Benders' decomposition (see below), which produces an exponential number of constraints (rows). The models of the former

type can be either set covering/set partitioning models (for bin packing or cutting stock problems) or set packing models (for knapsack problems).

Branch(-and-cut)-and-price

Models based on the set covering formulation are usually implemented through column generation (see Section 5.4.3) for solving the associated linear program relaxation. Embedding the linear program into an enumerative scheme allows one to obtain an optimal integer solution through branch-and-price or branch-and-cut-and-price (when cuts are added at the decision nodes).

To the best of our knowledge, Belov and Scheithauer [27] were the first to propose a branch-and-cut-and-price algorithm (based on a set-packing formulation) for a two-dimensional problem, namely the 2-staged guillotine 2D-KP.

Puchinger and Raidl [272] solved the 3-staged guillotine 2D-BPP with a branch-and-price algorithm with special techniques (*column generation stabilization*) to accelerate the convergence of the method, as previously suggested by Valério de Carvalho [313] and Ben Amor, Desrosiers, and Valério de Carvalho [10].

Pisinger and Sigurd [267, 268] proposed branch-and-price algorithms for 2D-BPPs with variable-sized and fixed-sized bins. In these algorithms (which are, from a computational point of view, the current state-of-the-art), the columns are generated through decomposition into a one-dimensional knapsack problem and a 2D-OPP, which are then solved by constraint programming. Bettinelli, Ceselli, and Righini [38] developed a branch-and-price approach to the level (2-staged) 2D-SPP in which the pricing problem consists of a penalized one-dimensional knapsack problem. Furini and Malaguti [139] and Mrad, Meftahi and Haouari [247] solved the 2-staged guillotine 2D-CSP, respectively with and without variable-sized bins, through branch-and-price algorithms.

Benders' Decomposition

Already in 1962, Benders [30] proposed a method to decompose an MILP model having a special *block structure* into an MILP *master problem* and an LP *slave problem*. At each iteration, the slave receives the current master solution and either proves its optimality or generates a cut (*Benders' cut*) that is added to the master. The approach is also referred to as *row generation*.

Geoffrion [147] generalized the Benders' decomposition to the case in which the subproblem too is an MILP. The latter decomposition was used by Caprara and Monaci [66] and by Baldacci and Boschetti [15] to solve the 2D-KP: in both approaches the master is an ILP based on the one-dimensional knapsack problem while the slave solves an associated 2D-OPP.

Hooker and Ottosson [169] defined the logic-based Benders' decomposition for general MILP problems: in this approach both the master and the slave are MILPs, but the latter is solved through logical deduction methods, such as constraint programming. This decomposition was successfully applied by Pisinger and Sigurd [267, 268] to the 2D-KP and by de Queiroz et al. [273] to a special 2D-KP that includes pairs of items that cannot both be in the solution (2D-KP with conflict graphs).

Although Benders' decomposition has been successfully applied to cutting and packing problems, the resulting cuts in the master problem may be weak. Codato and Fischetti [87] defined stronger *combinatorial Benders' cuts* for general integer problems. Côté, Dell'Amico, and Iori [91] and Delorme, Iori, and Martello [110] adopted such cuts for the solution of the 2D-SPP without and with item rotation, respectively. In their logic-based Benders' decomposition, the master problem is a parallel processor scheduling problem with contiguity constraints (see Section 5.4.5) that produces the x-coordinate of each item, while the slave checks whether feasible y-coordinates exist. In the negative case, heuristic methods look for a minimal infeasible set of items that produces a combinatorial Benders' cut. A similar decomposition, with standard Benders' cuts, was used by Côté, Gendreau, and Potvin [92] for the 2D-OPP with unloading constraints.

Recently, Martin et al. [229] proposed a Benders' decomposition algorithm to solve the guillotine 2D-KP. In their decomposition, the master problem is modeled using the pseudo-polynomial model by Beasley [13], whereas the slave checks the guillotine restriction.

5.7 Exact Methods based on Implicit Enumeration

The algorithms reviewed in this section consist of enumeration schemes that do not explicitly make use of MILP models and hence do not require the use of a solver.

5.7.1 Branch-and-Bound

Several enumeration algorithms were derived from the bottom-left strategy proposed by Baker, Coffman, and Rivest [14] (see Section 5.5.2), which produces an approximate solution by placing one item at a time, in the lowest possible feasible position, left justified. Hadjiconstantinou and Christofides [161] developed a tree-search exact algorithm for the 2D-KP that packs the next item in every possible position such that the item's left and bottom edges touch either the bin or edges of other items (*left-most downward* strategy). Martello and Vigo [228] adopted this strategy in a branch-and-bound algorithm for the 2D-OPP, and embedded it in a two-level enumeration algorithm for the 2D-BPP.

This strategy later evolved into two classes of implicit enumeration schemes, namely the *staircase placement* and the *niche placement*. Both strategies enumerate the possible packing of the items with their bottom-left corner in particular positions (*corner points*), induced by the current (partial) packing, whose definition depends on the specific strategy.

Staircase Placement

The *staircase placement* makes use of the *envelope* (see Figure 5.5 (a)), defined as the monotone (right-down) staircase-like boundary, that: (i) separates the area where previously packed items are placed from the area available for the remaining items, and (ii) is composed by segments touching at least one edge of a packed item or the bin. The *corner points* are those in which the slope of the envelope changes from vertical to horizontal. These concepts were originally introduced by Scheithauer [291].

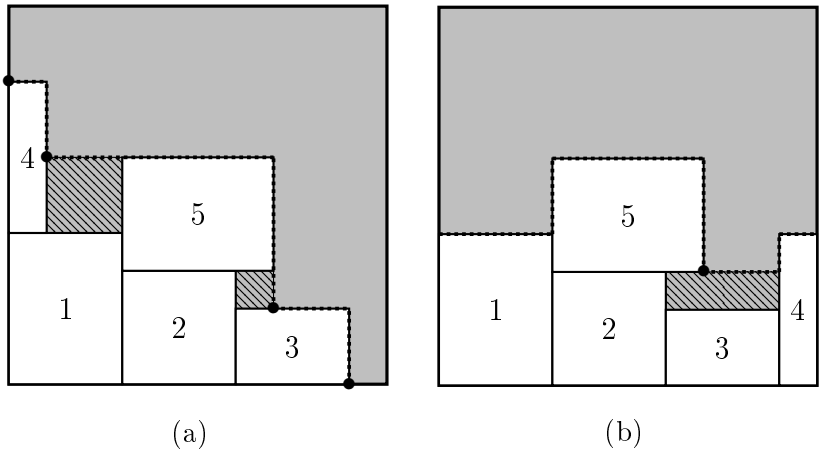

Figure 5.5: corner points of (a) staircase placement; (b) niche placement.

Martello, Monaci, and Vigo [225] proposed a branch-and-bound algorithm for the 2D-SPP which is based on the staircase placement and on a strategy that avoids the enumeration of symmetric solutions. The method was later modified by Iori, Salazar González, and Vigo [181] to solve the 2D-OPP with loading/unloading constraints. Other branch-and-bound algorithms based on the staircase placement were proposed by Alvarez-Valdes, Parreño, and Tamarit [7] for the 2D-SPP, by Clautiaux, Carlier, and Moukrim [81] for the 2D-OPP, and by Bekrar et al. [22] for the 2D-SPP with guillotine constraint. The latter algorithm makes use of a method by Messaoud, Chu, and Espinouse [238] to detect non-guillotine patterns in polynomial time.

Recently, Xu and Lee [338] studied the *continuous berth allocation problem*, in which incoming vessels need to be assigned a time and a berth location on a quay. Each vessel can be interpreted as a rectangle with given width (the space it occupies in the berth) and height (the time it consumes): the problem is to "pack" the vessels into a strip of given width (the total space available at the berth) so as to minimize the total weighted completion times of the activities on the vessels. They solved the problem with a branch-and-bound algorithm that uses a staircase branching rule and obtains at each node a valid lower bound by invoking a model reformulation.

Niche Placement

The *niche placement* uses the *skyline structure* (see Figure 5.5 (b)), a boundary composed by orthogonal line segments separating the area of previously packed items from the area that is available for the remaining items. At each iteration, the *niche* is the lowest and (in case of ties) leftmost horizontal segment: the left extreme of the niche is the only *corner point* considered for packing the next item.

A complete branching is obtained by creating at most $n+1$ nodes as follows. The first n (at most) nodes are created by packing an item in the niche (obviously disregarding those that would lead to overlapping). Then, a last node in which no item is packed in the niche is also created. In such node, the niche is closed, so that at the next iteration a new niche will be considered.

The niche placement was used by Boschetti and Montaletti [49] in a branch-and-bound

algorithm for the 2D-SPP and by Lesh et al. [206] for the 2D-SPP with *perfect packing* (a packing in which no wasted space is allowed). Kenmochi et al. [192] proposed two branch-and-bound algorithms, respectively adopting niche and staircase placement, for the 2D-SPP with perfect packing, with or without orthogonal rotation. They additionally derived algorithms for the general 2D-SPP. Improvements of the latter algorithms were presented by Arahori, Imamichi, and Nagamochi [12].

Other Enumeration Schemes

Certain solution methods for the 2D-KP are based on an enumeration of candidate subsets of items combined with an inner solution method that checks whether the current subset can be feasibly packed. This method has been used by Caprara and Monaci [66] for the 2D-KP and by Fekete, Schepers, and van der Veen [128] for knapsack problems with an arbitrary number of dimensions.

Christofides and Whitlock [77] and Christofides and Hadjiconstantinou [75] introduced enumerative approaches for the 2D-KP with guillotine constraints. Their algorithms are based on a tree-search approach where branchings correspond to cuts on a rectangle and bounds are obtained by solving relaxations through dynamic programming. Dolatabadi, Lodi, and Monaci [115] improved this enumeration scheme, obtaining a branch-and-bound and a branch-and-cut algorithm. Another search strategy that has been extensively used for problems with guillotine constraints is the bottom-up approach (see, e.g., Viswanathan and Bagchi [326], Hifi [167], Cung, Hifi, and Le Cun [97], and Fleszar [133]). It is based on enumerating all feasible patterns obtained through *builds*: a build is a rectangle produced by the (horizontal or vertical) combination of a pair of items or other builds. The bottom-up approach also inspired the development of MILP formulations for guillotine problems (see, e.g., Martin, Morabito, and Munari [230]).

A special case of 2-staged guillotine cutting patterns are the *checkerboard patterns*. While the guillotine constraint allows each cut to separate a bin into two pieces, which can then be treated as new (smaller) bins, the checkerboard constraint imposes that the solution be obtained through a set of horizontal cuts and a set of vertical cuts, all performed on the original bin. Yanasse and Katsurayama [339, 340] presented enumerative algorithms for the 2D-KP with checkerboard patterns.

Clautiaux et al. [86] proposed label setting algorithms for the four-staged guillotine 2D-KP. They presented reduction procedures, filtering rules based on a Lagrangian relaxation, and a state space relaxation.

5.7.2 Graph-Based Approaches

Morabito and Arenales [244] proposed an exact algorithm for the guillotine 2D-KP based on AND/OR graphs. In an AND/OR-graph representation, the nodes correspond to rectangles, the arcs correspond to cuts, and cutting patterns are represented as *complete paths* in the graph.

Fekete and Schepers [126] proposed a graph-theoretical characterization of the packing of a set of items into a bin. Their representation makes use of two interval graphs, associated with the horizontal and vertical dimensions, respectively: each item corresponds

to a vertex, and two vertices are connected by an edge if and only if the projections of the corresponding items on the horizontal/vertical axis overlap. Theoretical properties allow one to detect the feasibility of a packing. This *interval graph model* easily extends to packings in higher dimensions.

Based on the interval graph representation, Fekete, Schepers, and van der Veen [128] proposed a branch-and-bound algorithm for higher-dimensional orthogonal packing problems. The algorithm was later improved by Belov and Rohling [26] through LP bounds based on the bar relaxation (see Section 5.4.5). In order to avoid the enumeration of symmetrical interval graphs and unnecessarily enumerated packing classes (as observed by Ferreira and Oliveira [130]), Joncour and Pêcher [188] proposed to adopt the so-called *consecutive ones matrices* to enumerate relevant interval graphs. Further improvements for the 2-dimensional case were proposed by Joncour, Pêcher, and Valicov [189].

Clautiaux, Jougler, and Moukrim [85] presented a graph-theoretical model for the 2D-OPP with guillotine constraints. The model is based on an arc-colored directed graph, which can be replaced by an uncolored undirected multigraph, and can be used for solving the problem through constraint programming.

5.7.3 Constraint Programming

As previously mentioned (see Section 5.6.3), Pisinger and Sigurd [267, 268] used a constraint programming formulation to solve 2D-OPP instances arising as subproblems in their decomposition approaches for the 2D-BPP (with fixed-sized or variable-sized bins). The main constraints are related to the relative positioning of each pair of items. This strategy was later adopted for other two-dimensional packing problems, e.g., by Korf, Moffitt, and Pollack [195] for the problem of orthogonally packing a set of two-dimensional items into a rectangle having minimum area, and by de Queiroz et al. [273] for a variant of the 2D-KP (see Section 5.6.3).

Clautiaux et al. [84] proposed a constraint-based scheduling model for the 2D-OPP, and solved it through constraint programming and effective propagation techniques. The approach was improved by Mesyagutov, Scheithauer, and Belov [239] in their algorithms for the 2D-SPP and the 2D-OPP, by embedding LP-based pruning rules into the constraint propagation process.

Soh et al. [301] solved the 2D-OPP by iteratively reducing it to *satisfiability testing* (SAT) problems, which look for a feasible assignment of a set of boolean variables. The approach extends to the solution of the 2D-SPP. Grandcolas and Pinto [157] proposed a SAT encoding of the interval graph model by Fekete and Schepers [126] for higher dimensional problems, and compared its efficiency with that of other SAT encodings.

Delorme, Iori, and Martello [110] proposed, for the 2D-CSP, a constraint programming formulation based on non-overlapping intervals to determine whether a solution of the contiguous one-dimensional relaxation (see Section 5.4.5) produces a feasible two-dimensional packing.

5.8 Open Problems

Our study shows that considerable improvements in the exact solution of two-dimensional cutting and packing problems emerged in the last decades, typically allowing to determine an optimal solution even for large-size instances. However, the inherent hardness of this class of problems is witnessed by the existence of unsolved instances with relatively few items. In the following, we provide a list of the main benchmarks that are still unsolved to proven optimality, not only for the main problems addressed in this survey, but also for other relevant problem variants:

- for the 2D-SPP, specific instances with 20 items cannot be solved to proven optimality even by recent and sophisticated algorithms (see, e.g., Côté, Dell'Amico, and Iori [91]). Among the 500 instances of the so called *10 classes* benchmark (proposed a long time ago by Berkey and Wang [37] and Martello and Vigo [228] for the 2D-BPP), only 322 are solved to proven optimality. A number of instances with 40 items are still open for the 2D-SPP with guillotine cuts (see, e.g., Mrad [142]);

- the 2D-SPP with item rotation is even more difficult from a computational perspective: among the above mentioned 500 instances, just 176 could be solved to optimality, and in particular 56 instances with only 20 items are still open (see, e.g., Delorme, Iori, and Martello [110]);

- the work by Delorme, Iori, and Martello [110] also presents extensive computational results for the *pallet loading problem* (a 2D-OPP variant in which all items have the same dimensions and rotation of 90 degrees is allowed) and the *rectangle packing problem* (a 2D-SPP variant in which items must be packed, without rotation, in a square of minimum area, see Martello and Monaci [224]). For the former problem, a few dozen instances remain unsolved (see `http://lagrange.ime.usp.br/~lobato/packing/cover3.php` for an up-to-date list). For the latter problem, more than 200 instances are unsolved, 50 of these (set RND_R15 in [224]) involving just 15 items;

- after more than 20 years, many 2D-BPP instances of the 10 classes benchmark (see [37], [228]) with $n \geq 60$ are still unsolved, and the same holds even for some instances with $n = 40$ (see Pisinger and Sigurd [268]). For all such instances the difference between the best upper and lower bounds is one bin;

- Pisinger and Sigurd [267] created 500 instances of the 2D-BPP with variable bin sizes and costs, by modifying the 10 classes above. They could only solve 164 of them, leaving as open problems 29 instances with 20 items, and 62 with 40 items. All instances are available at `https://www.computational-logistics.org/orlib/topic/2D%20Variable-sized%20Bin%20Packing/index.html#intro~` ;

- a very difficult 2D-KP instance with only 32 items, proposed 35 years ago by Beasley [20], is still open (see Caprara and Monaci [66] and Baldacci and Boschetti [15]). For the 2D-KP with guillotine constraints, several instances with either 25 or 50 items, recently proposed by Velasco and Uchoa [325], are open;

- for the 2D-OPP, two difficult sets of instances have been created by Mesyagutov, Scheithauer, and Belov [239], and can be downloaded at `http://www.math.tu-dresden.de/~capad/TESTS/OPP/12_cp2_data.zip`. The zipfile includes, among others, 1080 instances (`Gleb_lpcs` and `Gleb_opp_gen`) with 20 items and bin width equal to either 100 or 1000, 18 of them being still unsolved. Other sets, known as C, N, and T (available at `https://www.euro-online.org/websites/esicup/data-sets/#1535972088188-55fb7640-4228`), contain 91 instances with about 70 items and bin width at most 200: only 39 of them have been solved to optimality (see Côté and Iori [93]);
- for the 2D-OPP with unloading constraints, Côté, Gendreau, and Potvin [92] generated 6 classes containing in total 3282 instances, with 26 items per instance on average: only 2179 of them have been solved to proven optimality. The instances can be downloaded from `https://w1.cirrelt.ca/~cotejean/` (see 2OPP-UL).

Most of the above instances can also be downloaded from Internet repositories like, e.g., the well-known *OR Library* `http://people.brunel.ac.uk/~mastjjb/jeb/info.html` by John Beasley [19] or the library of the OR group of the University of Bologna `http://or.dei.unibo.it/library`. In addition, a generator of instances for two-dimensional rectangular cutting and packing problems, proposed by Silva, Oliveira, and Wäscher [298], can be downloaded from `https://sites.google.com/gcloud.fe.up.pt/cutting-and-packing-tools`.

5.9 Conclusions and Future Research Directions

We reviewed over 180 papers related to two-dimensional orthogonal cutting and packing problems. The literature in this field has grown considerably in recent years, as also shown by the fact that about half of our references appeared in the last ten years, and just 20% were published before 2000. We described the main preprocessing and relaxation methods and briefly examined heuristic and approximation algorithms. We discussed mathematical models and reviewed the most effective implicit enumeration approaches. Finally, we provided an extensive list of instances for which the optimal solution is still unknown.

To facilitate future research, we provide in Table 5.1 a summary of the publications from the last 20 years on exact methods for 2D problems. The table presents the main problems, variants and techniques studied by each paper. The problems are divided into the four main categories discussed in Section 5.2 (column BPP gives both references to 2D-BPP and 2D-CSP). The variants are given in the columns OR (orthogonal rotation), GL (guillotine cuts), VS (variable-sized bins), LU (loading/unloading constraints) and OTH (other variants). The main techniques are shown in columns MILP, B&B (branch-and-bound), B&P (branch(-and-cut)-and-price), BD (Benders' decomposition), GB (graph-based), CP (constraint programming), and REL (relaxations).

Some statistics on the recent exact methods can be drawn from the references in the table: 20 papers studied the 2D-SPP, 17 studied the 2D-BPP and/or the 2D-CSP, 14

studied the 2D-KP, and 19 studied the 2D-OPP. The most studied variant is the guillotine case. The most popular solution methods are MILP and branch-and-bound (both with 14 references). Note, however, that the number of papers on Benders' decomposition is relatively high although the use of this technique for cutting and packing problems is quite recent.

We also provide the reader some hints of what we consider promising future research directions:

- Constraint Programming, either as a stand-alone algorithm or as a sub-problem in decomposition methods, recently led to consistent improvements in computational results on 2D problems. Further research on this type of methodology is envisaged, as we see the potential to obtain further improvements;

- the use of Benders' decompositions (with or without Constraint Programming) for solving 2D problems is quite recent. For some of the classical problems, these decompositions represent the state-of-the-art, as they could solve to proven optimality several open instances. Such techniques are easily adaptable to many other problem variants, as one can consider embedding the additional features either in the master or in the slave. We thus envisage further research on this decomposition to solve new problem variants;

- current research trends appear to be intensive for 2D problems with additional constraints, like loading/unloading, stability, fragility, etc.;

- most of the techniques we presented can be generalized to the case of three or more dimensions, although this is not always straightforward. The amount of research on three-dimensional cutting and packing problems has been so far quite limited, in spite of their large number of applications. Interesting research directions can be explored for these problems, starting from the 2D ones.

We hope that this review will encourage researchers to pursue investigations in these fascinating topics where there is still room for improving the algorithmic approaches.

Table 5.1: Summary of papers with exact methods and relaxations from the last 20 years.

Reference	2D Problems				Variants					Techniques						
	SPP	BPP	KP	OPP	OR	GL	VS	LU	OTH	MILP	B&B	B&P	BD	GB	CP	REL
Alvarez-Valdes, Parreño, and Tamarit [7]											✓					✓
Arahori, Imamichi, and Nagamochi[12]	✓										✓					✓
Baldacci and Boschetti [15]			✓										✓			
Bekrar et al. [22]	✓	✓				✓					✓					✓
Belov et al. [24]				✓												✓
Belov et al. [25]				✓												✓
Belov and Rohling [26]				✓										✓		✓
Belov and Scheithauer [27]			✓			✓						✓				
Bettinelli, Ceselli, and Righini [38]	✓					✓						✓				
Boschetti, Hadjiconstantinou, and Mingozzi [46]			✓							✓						✓
Boschetti and Mingozzi [47]		✓														✓
Boschetti and Mingozzi [48]		✓			✓											✓
Boschetti and Montaletti [49]	✓										✓					✓
Caprara and Monaci [66]			✓										✓			
Caprara and Monaci [67]		✓														✓
Carlier, Clautiaux, and Moukrim [82]		✓														✓
Castro and Grossmann [68]	✓									✓						
Castro and Oliveira [69]	✓	✓	✓							✓						
Cintra et al. [78]	✓	✓	✓		✓	✓										✓
Clautiaux, Carlier, and Moukrim [82]				✓							✓					✓
Clautiaux et al.[84]				✓											✓	
Clautiaux, Jouglet, and Moukrim [85]				✓		✓								✓		
Clautiaux et al. [86]			✓			✓					✓					✓
Côté, Dell'Amico, and Iori [91]	✓			✓									✓			
Côté, Gendreau, and Potvin [92]				✓				✓					✓			
Delorme, Iori, and Martello [110]	✓				✓								✓			
Dolatabadi, Lodi, and Monaci [115]			✓			✓					✓		✓			
Fekete and Schepers [126]				✓										✓		
Fekete and Schepers [127]				✓												✓
Fekete, Schepers, and Veen [128]				✓							✓			✓		
Fleszar [133]				✓	✓	✓					✓					
Friedow and Scheithauer [136]	✓															✓
Furini and Malaguti [139]		✓				✓	✓			✓		✓				
Furini, Malaguti, and Thomopulos [141]	✓	✓	✓			✓				✓						
Grandcolas and Pinto [157]				✓											✓	
Iori, Salazar-Gonzalez, and Vigo [180]				✓				✓			✓					
Joncour and Pêcher [188]				✓										✓		
Joncour, Pêcher, and Valicov [189]				✓										✓		
Kenmochi et al. [192]	✓				✓						✓					
Lesh et al. [206]	✓										✓					
Lodi, Martello, and Vigo [215]	✓	✓				✓				✓						✓
Lodi and Monaci [217]			✓			✓				✓						
Macedo, Alves, and Valério de Carvalho [219]		✓				✓				✓						
Martello, Monaci, and Vigo [225]	✓										✓					✓
Martin et al. [229]			✓			✓							✓			
Martin, Morabito, and Munari [230]			✓			✓				✓						
Matayoshi [236]																
Melega, de Araujo, and Jans [237]																
Messaoud et al. [238]	✓	✓		✓		✓				✓						
Mesyagutov, Scheithauer, and Belov [239]	✓			✓											✓	✓
Mesyagutov et al. [240]																✓
Mrad [246]	✓					✓				✓						
Pisinger and Sigurd [267]		✓					✓					✓				
Pisinger and Sigurd [268]		✓										✓				
Puchinger and Raidl [272]		✓				✓				✓		✓				
Queiroz et al. [273]			✓						✓				✓			
Queiroz and Miyazawa [274]	✓								✓	✓			✓			
Serairi and Haouari [294]		✓														✓
Silva, Alvelos, and Valério de Carvalho [296]		✓				✓				✓						
Soh et al. [301]	✓														✓	
Velasco and Uchoa [325]			✓			✓										✓
Yanasse and Katsurayama [339]				✓					✓		✓					

Chapter 6

2DPackLib: A Two-dimensional Cutting and Packing Library

6.1 Introduction

Cutting and packing is a major field in optimization. Two-dimensional (rectangular) cutting and packing problems model indeed a large number of relevant industrial applications. In the former class of problems a set of rectangular *items* must be cut from a set of rectangular *stock units*, whereas in the latter a set of rectangular items must be packed into a set of rectangular *containers*. It is thus clear that the two problems are strongly related, in the sense that cutting an item from a stock unit is practically equivalent to packing an item into a container. In the following, we use the terms cutting and packing interchangeably and we refer to both stock units and containers as *bins*. Typical cutting applications derive from industrial productions involving steel, wood or glass, where the aim is to reduce waste material. Typical packing applications include optimal use of spaces in warehouses or in packaging processes. According to specific applications, two-dimensional cutting and packing problems may require to optimize different objective functions, and to satisfy specialized side constraints.

In the following, we deal with the most studied variants of these problems, where the edges of the items must be parallel to those of the bin in which they are packed (*orthogonal rectangular* cutting and packing). Let $\mathcal{B} = (W, H)$ be a rectangular bin of width $W \in \mathbb{Z}_+$ and height $H \in \mathbb{Z}_+$, and $\mathcal{I}$ be a set of rectangular items, where each item $i \in \mathcal{I}$ has width $w_i \in \mathbb{Z}_+$ and height $h_i \in \mathbb{Z}_+$ (with $w_i \leq W$ and $h_i \leq H$). A *packing* of $\mathcal{I}' \subseteq \mathcal{I}$ into $\mathcal{B}$ can be described as a function $\mathcal{F} : \mathcal{I}' \to \mathbb{Z}_+^2$ that maps each item $i \in \mathcal{I}'$ to a pair of coordinates $\mathcal{F}(i) = (x_i, y_i)$, such that

$$x_i \in \{0, ..., W - w_i\} \text{ and } y_i \in \{0, ..., H - h_i\} \qquad (i \in \mathcal{I}'), \tag{6.1}$$

$$[x_i, x_i{+}w_i) \cap [x_j, x_j{+}w_j) = \emptyset \text{ or } [y_i, y_i{+}h_i) \cap [y_j, y_j + h_j) = \emptyset \ (i, j{\in}\mathcal{I}', i{\neq}j). \tag{6.2}$$

In this representation, the bin is located on a Cartesian plane with its edges parallel to the axes and its bottom-left corner on the origin. The coordinates $\mathcal{F}(i) = (x_i, y_i)$ represent the position where the bottom-left corner of each $i \in \mathcal{I}'$ is placed. Constraints (6.1) impose

that each item is entirely inside the bin, while constraints (6.2) forbid overlapping between pairs of items. Constraints (6.1) and (6.2) impose that items have fixed orientation (i.e., cannot be rotated), but they can be adapted to consider the popular variant (discussed in Section 6.2) where items can be rotated by 90 degrees.

Iori et al. [175] recently presented a survey on the main exact solution methods for the most popular two-dimensional orthogonal cutting and packing problems. The huge number of algorithms proposed for these problems has involved, along the years, the adoption of many benchmarks for evaluating their performance. The survey provides a precise classification of the different problem variants. Concerning benchmarks and computational experiments, some issues must be considered:

- the number of benchmarks in the literature is huge, and even instances proposed in the Seventies are still used to evaluate new methods;
- there is no clear correspondence between tested variants and adopted benchmarks as, due to similarity among some of the variants, the same benchmarks have been adopted for different variants;
- many benchmarks are available in different specific libraries, where they are provided in different formats, not always clearly detailed;
- some benchmarks were originally published in personal web pages which no longer exist.

Besides the classification of variants and benchmarks, the authors implemented a standardization of the benchmarks and a unified format for the instances, as well as an analysis of the matching variants-benchmarks. We now collected these results and other useful instruments (visual tools, links to surveys, bibliographies) in a publicly available web page, the `2DPackLib`.

Web-based libraries for combinatorial optimization problems have become a popular instrument for researchers wanting to study specific fields and to test their findings on reliable benchmarks. The seminal instruments of this kind, realized 30 years ago, are still very active: the `OR-Library`, developed in 1990 by Beasley [19] (see `people.brunel.ac.uk/~mastjjb/jeb/info.html`), mostly devoted to a number of combinatorial optimization problems, and the `QAPLIB`, implemented in 1991 by Burkard et al. [56, 57] for the *Quadratic Assignment Problem* (see `https://coral.ise.lehigh.edu/data-sets/qaplib/`). In the following decade, Applegate et al. [11] presented a famous, rich web page (see `http://www.math.uwaterloo.ca/tsp/`) for the *Traveling Salesman Problem*. In the last ten years, new libraries have been implemented for *Mixed Integer Programming* (the `MIPLIB` by Koch et al. [194], see `http://miplib.zib.de/`), for *Vehicle Routing* (the `VRPH` by Groër et al. [158], see `http://sites.google.com/site/vrphlibrary/`, and the `CVRPLIB` by Uchoa et al. [311], see `http://vrp.atd-lab.inf.puc-rio.br/`), and for *Bin Packing* (the `BPPLIB` by Delorme et al. [111], see `http://or.dei.unibo.it/library/bpplib`).

The purpose of this paper is to present the contents of the `2DPackLib` and to provide systematized relationships and links that should hopefully assist researchers in the development and testing of new solution approaches.

The library is available at `http://or.dei.unibo.it/library/2dpacklib`. In the next sections, we describe its contents, namely:

- synthetic classification of problems and variants;
- pointers to surveys and typologies;
- standardized benchmarks and their matching with problems and variants;
- bibliographies and additional tools.

6.2 Classification

Using the basic definitions provided in Section 6.1, the `2DPackLib` classifies the main two-dimensional orthogonal cutting and packing problems as follows:

- *Two-Dimensional Strip Packing Problem* (2D-SPP): given a set $\mathcal{I}$ of items and a width W, determine the minimum H value such that there exists a packing of $\mathcal{I}$ into a bin $\mathcal{B} = (W, H)$. In this problem, there is a unique container, called strip, having a fixed width and an infinite height;
- *Two-Dimensional Cutting Stock Problem* (2D-CSP): given an unlimited number of identical bins and a set $\mathcal{I}$ of items, where each item is associated with a *demand* $d_i \in \mathbb{Z}_+$ (the minimum number of copies to be packed), determine the minimum number of bins needed to pack all demands. The *Two-Dimensional Bin Packing Problem* (2D-BPP) is the special case in which $d_i = 1$, for all $i \in \mathcal{I}$;
- *Two-Dimensional Knapsack Problem* (2D-KP): given a bin $\mathcal{B}$ and a set $\mathcal{I}$ of items, where each item $i \in \mathcal{I}$ is associated with a profit p_i, determine a subset $\mathcal{I}' \subseteq \mathcal{I}$ that can be packed into $\mathcal{B}$ and such that $\sum_{i \in I'} p_i$ is a maximum. In the 2D-KP, each item $i \in \mathcal{I}$ may be associated with a maximum number of copies b_i. This version of the problem is known as the *constrained 2D-KP*, whereas in the *unconstrained* version of the problem an unlimited number of copies of each item is available;
- *Two-Dimensional Orthogonal Packing Problem* (2D-OPP): given a bin $\mathcal{B}$ and a set $\mathcal{I}$ of items, determine whether there exists a packing of $\mathcal{I}$ into $\mathcal{B}$.

Many variants of these problems have been studied in the literature, the most popular ones being:

- *guillotine cuts*: in each bin, cuts are restricted to be edge-to-edge and parallel to the edges of the bin. A cut splits a bin into two rectangles, which in turn can be waste, items, or bins to be recursively cut again;
- *orthogonal rotations*: items are allowed to be rotated by 90 degrees, as opposed to having *fixed orientation*;

- *variable-sized bins*: this variant arises when, instead of an unlimited number of identical bins, one is given a set of bin types, each having a specific size (width and height), cost, and availability. Typically, this variant is studied for problems 2D-CSP and 2D-BPP;
- *loading and unloading constraints*: a packing is associated with a loading/unloading item sequence and must be such that each item can be loaded/unloaded with no need to physically repositioning other items.

These problems are strongly $\mathcal{NP}$-hard (the 2D-OPP, strongly $\mathcal{NP}$-complete) and very difficult to solve in practice. The combinations problem-variant produce a high number of specific issues, and many of them have been investigated in the literature. (The survey [175] examines over 180 related references.)

6.3 Surveys and typologies

In 1990, Dyckhoff [119] proposed the first typology for classifying cutting and packing problems. Other typologies were later developed by Lodi et al. [213] (further extended by Martello et al. [225]) and Wäscher et al. [332]. Table 6.1 provides the classification of the optimization problems considered in the `2DPackLib` according to the different typologies.

Table 6.1: Problem classifications according to classical typologies.

	Dyckhoff [119]	Lodi et al. [213]	Wäscher et al. [332]
2D-SPP	-	2SP|O|F	ODP
2D-BPP	2/V/I/M	2BP|O|F	SBSBPP
2D-CSP	2/V/I/R	2BP|O|F	SSSCSP
2D-KP	2/B/O	2KP|O|F	SLOPP

In the Noughties, surveys on two-dimensional packing problems were presented by Lodi et al. [210, 214], while Ntene and van Vuuren [253] produced a specialized review and comparison of guillotine heuristics for the 2D-SPP.

In the following decade, a survey on general two-dimensional packing problems was proposed by Alvarez-Valdes et al. [2]. Surveys on specific problems, variants, and methodologies were produced by Lodi et al. [212] (2D-BPP and 2D-SPP), Silva et al. [299] (*pallet loading problem*, a variant of the 2D-BPP in which one has to pack, with orthogonal rotations, the maximum number of identical items into a single bin), Oliveira et al. [254] (2D-SPP), Christensen et al. [74] (approximation and online algorithms), and Russo et al. [288](upper bounds for problems with guillotine cuts).

Finally, in the last two years, Bezerra et al. [39] proposed a survey on a special case of the 2D-SPP, while Iori et al. [175] reviewed exact algorithms for two-dimensional cutting and packing problems.

The `2DPackLib` provides direct pointers to the PDF files of all surveys mentioned above (journal subscriptions may be requested).

6.4 Benchmarks

This is the most relevant section of the 2DPackLib. Many benchmarks have been proposed to evaluate the computational performances of algorithms for different variants of two-dimensional cutting and packing problems. Unfortunately the literature has a number of drawbacks: unclear correspondence between variants and benchmarks, non-uniform data formats, sometimes hard to understand, unavailability of benchmarks included in dead personal web pages.

In order to facilitate future computational studies, we converted the main benchmark sets to a standard format that comprises the four main 2D cutting and packing problems with identical bins. For that, let W and H be, respectively, the width and height of the bin, and w_i, h_i, d_i, b_i, and p_i the width, height, demand, maximum number of copies, and profit of each item $i \in \mathcal{I} = \{1, ..., m\}$. The instances are in the following format:

- m
- W H
- for each item i $(i = 1, ..., m)$: i w_i h_i d_i b_i p_i

The width and height of the items appear in all problems. The demands appear in the 2D-CSP and, sometimes, in variants of the 2D-SPP and the 2D-OPP. The maximum number of copies of items is usually considered in the constrained 2D-KP and variants. The profit is considered in the 2D-KP only.

The format can obviously be used for the variants that include guillotine cuts and orthogonal rotations as well, but not for the cases of loading/unloading constraints and variable-sized bins. However, very few benchmarks have been proposed for these two cases, and a more general format for handling them would be considerably more complex for the most commonly used benchmarks. The library provides however pointers to benchmarks for a number of variants and related problems that do not fall in our format.

The benchmark sets can be downloaded, in the unified format, from the 2DPackLib as Zip files. Since most benchmarks have been adopted for different problem variants, the library also provides, for each benchmark, an up-to-date list of recently published results on different problems. The next sections provide a synthetic description of the benchmark sets.

6.4.1 Benchmarks originally proposed for the 2D-SPP

- **N and T**: two benchmarks, composed by 35 instances each, proposed by Hopper and Turton [171]. The number of items ranges between 17 and 199 and the strip width is always equal to 200;
- **C**: 21 instances proposed by Hopper and Turton [172], in which the number of items ranges between 16 and 197, and the width of the strip ranges between 20 and 160. The instances are generated so that the optimal solution is known in advance. A subset of 9 instancs from this benchmark has been referred to as "HT" by some authors;

- **BKW**: 13 instances proposed by Burke et al. [60], in which the number of items ranges between 10 and 3152, and the width of the strip ranges between 40 and 640. These instances, that are also referred to as "n" by some authors, were generated so that an optimal solution consists of a perfect packing that can be produced by guillotine cuts;

- **ZDF**: 16 instances produced by Leung and Zhang [207] by combining other instances from the literature. The number of items ranges between 580 and 75 032, and the width of the strips ranges between 100 and 9000.

The files provided for 2D-SPP instances also report a value for H, which is either the optimal height or -1. The first case only occurs for benchmarks where the optimal solution was already produced by the generation process.

6.4.2 Benchmarks proposed for the 2D-BPP and 2D-CSP

- **BENG**: ten instances introduced by Bengtsson [31] by generating both the size of the items and the dimensions of the bins according to a uniform distribution. The number of items ranges between 20 and 200, whereas the largest bin is $(40, 25)$.

- **CLASS**: this benchmark includes 500 instances, divided into 10 classes, and was introduced by Berkey and Wang [37] (classes 1 to 6) and by Martello and Vigo [228] (classes 7 to 10). Each class includes instances of different sizes, namely it has 10 instances for each value of the number of items in {20, 40, 60, 80, 100}. The bins are identical for all instances in each class, and they vary from $(10, 10)$ in class 1 to $(300, 300)$ in class 6;

- **A**: 43 instances introduced by Macedo et al. [220] for the 2D-CSP with guillotine cuts. In these instances, that are derived from the furniture industry, the number of items ranges between 13 and 809 and the bins are either $(2750, 1220)$, $(2550, 2100)$, or $(2470, 2080)$.

6.4.3 Benchmarks proposed for the 2D-KP

- **CGCUT**: three small instances generated by Christofides and Whitlock [77]. The dimensions and profits of the items were generated following a uniform distribution. Some authors refer to this set as "ChW";

- **WANG**: three small instances obtained by Wang [329] by modifying the bin and the maximum number of copies of one instance from CGCUT, and by considering the profit of each item equal to its area;

- **NGCUT**: 12 small instances generated by Beasley [20]. The dimensions and profits of the items were generated following a uniform distribution;

- **GCUT**: 13 instances proposed by Beasley [18]. The first 12 instances are associated with each combination of numbers of items in the set $\{10, 20, 30, 50\}$ and bins in

the set $\{(250, 250), (500, 500), (1000, 1000)\}$. The dimensions of the items were generated following a uniform distribution. The last instance corresponds to a real-world problem with 32 items and bin $(3000, 3000)$. In all instances, profits coincide with item areas;

- **OF**: two small instances generated by Oliveira and Ferreira [256], where item dimensions follow a beta distribution and profits coincide with item areas;
- **OKP**: five instances generated by Fekete and Schepers [129] in the same manner as the NGCUT set. The number of items ranges between 15 and 33 and the width and height of the bins is 100;
- **CU and CW**: 22 instances generated by Fayard et al. [124]. The number of items ranges between 25 and 60 and the largest bin is $(992, 970)$. CU has 11 unweighted instances (i.e., with item profits given by their area), whereas CW has 11 weighted instances, where profits are randomly generated;
- **LU, LW, LX**: three groups of 5 large-scale instances each, proposed by Hifi [168] and Russo et al. [289]. The number of items ranges between 100 and 550 and the largest bin is $(45\,237, 35\,983)$;
- **APT**: 40 instances proposed by Alvarez-Valdés et al. [3]. The range of number of items and dimensions of the bin follows the policies adopted for CU and CW, but the instances also contain some smaller items to increase their difficulty;
- **NGCUTFS**: 630 instances generated by Beasley [21] in the same manner as the OKP set, but with the number of items ranging between 40 and 1000;
- **MP**: 450 instances proposed by Morabito and Pureza [245]. The number of items is in $\{10, 20, 30, 40, 50\}$ and the width and height of the bins is 100. This set is divided into three classes of 150 instances and each class is characterized by different ranges used to create the item dimensions;
- **VU**: 80 instances proposed by Velasco and Uchoa [325]. The number of items is 50, and the bins have dimensions ranging between 100 and 400. The items are relatively small compared to the bin, a feature often associated with hard instances.

6.4.4 Benchmarks proposed for the 2D-OPP

- **CJCM**: 42 instances proposed by Clautiaux et al. [84]. The number of items ranges between 10 and 23 and the width and height of the bins is 20;
- **MSB**: 1080 instances proposed by Mesyagutov et al. [239], divided into two classes. The first class has 630 instances, each with 20 items and width and height of the bins equal to 1000. The second has 450 instances, each with number of items in $\{10, 15, 20, 25, 30\}$ and width and height of the bins equal to 100.

6.5 Bibliographies and additional tools

The `2DPackLib` includes

- `TwoBinPack`, an open source software to interactively solve rectangular cutting and packing problems, developed by Costa et al. [90], an application useful to practitioners and developers thanks to its visual tools;
- the BibTeX file containing almost 200 references, mostly appeared in the last ten years, from the survey by Iori et al. [175];
- a list of links related to two-dimensional cutting and packing, such as working groups, library pages, and the instance generator developed by Silva et al. [298].

6.6 Conclusions

We presented the `2DPackLib`, a library dedicated to two-dimensional orthogonal cutting and packing problems that provides pointers to surveys and typologies, benchmarks, an interactive visual tool, a BibTeX file of almost 200 references, and a list of relevant links. The benchmarks (comprising over 3000 instances) are standardized to provide a unified format for all the instances. This addresses previous issues on benchmarks with different formats, not always clearly detailed.

We are confident that the `2DPackLib` will facilitate future research in the active area of two-dimensional cutting and packing problems.

Chapter 7

Integrated Workforce Scheduling and Flexible Flow Shop Problem in the Meat Industry

7.1 Introduction

Production scheduling is one of the most common classes of problems faced by companies seeking to optimize their manufacturing system. In those problems, a set of jobs has to be scheduled in a set of machines while satisfying a set of practical constraints and optimizing a given objective. In this paper, we study a scheduling problem faced daily by a meat producing company. The company receives daily a set of orders to be produced in a single day. Each order is associated with a due date and is produced by following up to two stages: in the first stage, the meat is processed (cut) on a given bench, and in the second stage it is sent to a conveyor to be packed into disposable trays. Benches and conveyors are seen as heterogeneous parallel machines, and their productivity depends on the number of workers operating each. Hence, the company needs to derive a new production plan every day, comprising the number of workers to operate each machine and the scheduling of operations composing the final orders. The scheduling problem faced by the company (which has a number of operational constraints that are formally discussed in Section 7.2) is a generalization of the well-known two-stage flexible (or hybrid) flow shop problem (see, e.g., [122]). The overall problem considers, in lexicographic way, the minimization of the number of unscheduled orders, the weighted tardiness, and the production costs.

The literature on flexible flow shop problems is broad. For surveys on general flexible flow shop variants, we refer the reader to [209], [286], and [305]. Concerning two-stage variants, in [309], the authors investigate the influence of repetitive scheduling in an environment with a single machine in the first stage and multiple lines in the second stage. In [285], we find an application related to a sterilization plant, aiming at reducing the number of tardy jobs and the makespan, while respecting sequence-independent setup times and jobs processed in parallel batches. The work of [208] handles a system with unrelated parallel machines on each stage and task tail group constraints, aiming at minimizing the total tardiness. The authors propose a new scheduling rule able to outperform twelve

dispatching rules from the literature. In [173], the authors study environments composed of a single machine either in the first or in the second stage and propose several solution procedures for such problems. The work of [330] deals with a problem in the glass-ceramic industry, whose objective is to minimize the makespan and energy consumption. Besides an integer linear programming model, they propose constructive, tabu search, and ant colony optimization algorithms.

In this paper, we propose a multi-start random heuristic to solve the integrated workforce scheduling and two-stage flexible flow shop problem faced by the company. The heuristic iteratively tests different combinations for the number of workers, and, for each combination, it generates the production schedule by following a constructive heuristic. At each iteration, it assigns orders to benches and conveyors by following a list of priorities. By means of extensive computational experiments based on realistic instances, we can show that the heuristic is effective in finding good-quality solutions and can provide a quick support to the company on its daily decisions.

The remaining of the paper is organized as follows. Section 7.2 provides a formal description of the problem. Section 7.3 presents the proposed solution method. Section 7.4 provides the results of the computational experiments. Finally, Section 7.5 presents concluding remarks and some directions for future research.

7.2 Problem Definition

In this section, we present a detailed description of the problem. The company receives a set $\mathcal{O}$ of orders to be scheduled in a workday, by following several operational constraints. Each order is expected to be produced before its due date, which corresponds to the time in which it has to be shipped to its final destination. Then, the problem is to determine the workforce and production schedule of a workday.

A workday.

A workday is defined over a time horizon of 24 hours, and may have up to two disjoint working periods. The two periods are subject to time-related constraints, such as minimum and maximum start time, end time, and duration. Moreover, there is a minimum and a maximum number of workers that can be hired for each period. Thus, a first set of decisions involves the number of working periods, as well as their start time, end time, and hired workers.

A period in a workday.

After deciding the duration and number of workers for each period, production-related decisions must be made. The production in each period is composed by two stages. Each stage has a set of heterogeneous parallel machines whose speed is variable. We assume workers have the same efficiency and the speed of each machine is proportional to the number of workers operating it. In addition, workers cannot be reallocated to different machines during a period. Thus, another decision is to determine the number of workers operating each machine during the period.

In our case study, first-stage machines are benches on which the meat is prepared, whereas second-stage machines are conveyors where the products are distributed to be packed into disposable trays that finally receive a stamp with the product information.

Details on the production in a period.

The last set of decisions considers the scheduling of the orders in each period. Each order in $\mathcal{O}$ is associated with: a final product identification code; a due date; a type of disposable tray (referred to as the order family); a type of stamp; a net weight; a raw product type, which has a productivity measure on each machine of the first and second stage where it can be processed. In addition, some orders have to be processed in a single stage, whereas others have to be processed (sequentially) on both stages. For orders that have to be processed in both stages, the product quality is preserved by imposing a maximum waiting time (e.g., 60 minutes or so) for the buffer between the two stages. We assume the buffer has unlimited size. All machines have setup times: first-stage and second-stage machines must undergo setups whenever the raw product is changed and whenever different tray or stamp types are required, respectively. Another machine-related constraint imposes a fixed transportation time between the first and second stage.

Summarizing, the decisions of the overall problem involve the distribution of the working periods, the number of workers on each period, the number of workers operating each machine in each period, and the production scheduling. The quality of a solution is measured by three objectives to be minimized according to a lexicographic order: (1) the total number of unscheduled orders; (2) the total weighted tardiness; and, (3) the total production cost. Although machine setups are not formally included in the objective, they are highly disliked by the company and it is always preferable to keep their number as small as possible.

7.3 Proposed Heuristic

To solve the problem under consideration, we propose a multi-start random heuristic. First, let us describe function `schedule`$(\mathcal{O}, L)$, which schedules orders in $\mathcal{O}$ to a period that is already set (i.e., its time window is already fixed and workers are already allocated to machines). The list L has a priority coefficient l_o for each order $o \in \mathcal{O}$. The orders are iteratively scheduled by choosing the best order to be scheduled at each iteration based on the priority list in use, which considers, for instance, the creation of setups, the due dates, the coefficients, the size of the gaps created between productions. Whenever necessary, the sequence in $\mathcal{O}$ is used to break ties. Orders with early due dates that are processed in a single stage only are always chosen to be scheduled first. Whenever we are about to schedule orders that are processed in both stages, we first choose the second-stage machine with earliest available time, then we choose the best order to be scheduled in that machine following one of the priority lists, and, finally, we choose the best first-stage machine to schedule the chosen order, also based on a priority list.

The function `schedule` is used within a constructive heuristic for the integrated problem. The overall structure of this heuristic is presented in Algorithm 2. The algorithm

creates several combinations of first- and second-period schedules by testing different combinations of number of workers and by iteratively updating the list of priority coefficients. The algorithm begins by testing different combinations of number of workers for the first period, and for each combination, the workers are allocated to machines by means of a function that consider a balance of the workload. After creating the schedule for the first working period, the heuristic follows by testing different possible schedules of the remaining orders in the second period. Again, for the second period all possible combinations of number of workers are tested and the workload based function is used again to allocate the workers to the machines. Then, different schedules for the second working period is created for each combination of workers. During the process of creating a schedule for a period, after fixing the number of workers for any of the two periods, we produce up to Π_{list} different schedules, which are iteratively obtained by updating the list L and calling the function `schedule`. Recall that L is used in the internal of `schedule` as an order-priority quantifier. In this way, L is updated in order to avoid tardiness, by increasing the priority coefficients of tardy orders in the current schedule.

Finally, our multi-start random heuristic is obtained by running Algorithm 2 for Π_{shuffle} iterations. In each of these iterations, we perform a reshuffle of the order sequence in $\mathcal{O}$. Recall that the sequence in $\mathcal{O}$ is used in a tie-breaking process in the internal of the `schedule` function, and, thus, it is expected to produce an impact in the final solution.

Algorithm 2: Heuristic for the integrated problem

$L \leftarrow (0, 0, ..., 0) \in \mathbb{Z}^{|\mathcal{O}|}$. // list of priority coefficients associated with $\mathcal{O}$
for all combinations of number of workers of stage one and two of the first working period **do**
  Assign workers to machines of the first working period by considering a workload balance
  for $i \leftarrow 1, \ldots, \Pi_{list}$ **do**
    $t_1 \leftarrow$ **schedule**$(\mathcal{O}, L)$
    $\mathcal{O}' \leftarrow \mathcal{O}$ minus the orders scheduled in the working period t_1
    $L' \leftarrow L$ updated for the orders in $\mathcal{O}'$
    for all combinations of number of workers of stage one and two of the second working period **do**
      Assign workers to machines of the second working period by considering a workload balance
      for $j \leftarrow 1, \ldots, \Pi_{list}$ **do**
        $t_2 \leftarrow$ **schedule**$(\mathcal{O}', L')$
        Update the best solution if its (lexicographic) objective function is worse than that of the current solution (t_1, t_2)
        if there is tardiness in the working period t_2 **then**
          Increase the coefficient of priority in L' of the orders with tardiness
        else
          Break the loop
        end if
      end for
      if there is tardiness in the working period t_1 **then**
        Increase the coefficient of priority in L of the orders with tardiness
      else
        Break the loop
      end if
    end for
  end for
end for

7.4 Computational Experiments

We implemented the multi-start random heuristic algorithm presented in Section 7.3 in C++. The algorithm was tested on 12 realistic instances, each representing one day of production at the company. Some randomization was applied to all instances in order to meet the company's privacy requirements. The tests were executed on a MacBook Air with an Intel Core i7 1.2 GHz quad-core processor. The goal of the experiments is to analyze the impact of different parameters in the performance of the heuristic. The parameters under consideration are:

(i) the range of workers available for each working period;

(ii) the number Π_{shuffle} of random shuffles of the sequence $\mathcal{O}$;

(iii) the number Π_{list} of iterations related to the update of the priority coefficients of the orders to try to prevent tardiness.

For each parameter configuration, we evaluate the three objectives of the problem plus the number of setups in the second stage.

We present in Table 7.1 the results related to parameter (i), which is the number of workers hired in each working period. We consider three configurations: L, meaning the Lowest fixed number of workers we may have; H, meaning the Highest fixed number of workers; and, R, meaning a Range/variable number of workers from L to H. These results were obtained with $\Pi_{\text{shuffle}} = 25$ and $\Pi_{\text{list}} = 5$, that are values defined after preliminary trial and error tests. By observing the results, we can note that the higher is the number of workers, the better is the solution. However, the best results are obtained when the heuristic has a range of workers (situation R) to decide on. For all instances, configurations R and H have provided the same number of unscheduled orders and weighted tardiness (except for $I11$, where R is better), but with R the production cost is always equal or smaller. The number of setups in stage two is small and comparable among all configurations. The average computational time in seconds for L, H and R is 9.2, 8.7 and 1666.3, respectively, so R requires a consistently larger amount of time to achieve the improved solutions.

As better solutions are obtained with configuration R, the following results consider only this configuration. In Table 7.2, we present the results we obtained when evaluating parameter (ii), the number of Π_{shuffle} iterations. We attempted three values, namely 1, 25, and 100. When Π_{shuffle} is set to 1, the heuristic looses its multi start characteristic. Hence, we expect to have better solutions as Π_{shuffle} increases, because we may change the sequence in which orders are scheduled. In these experiments, we have set $\Pi_{\text{list}} = 5$.

Observing Table 7.2, the heuristic returns better solutions when more iterations are available. This means that restarting with possible different orders has a positive impact on the final solution even if the order sequence is the last criterion in the list of priorities. With 25 and 100 iterations, the heuristic is able to obtain the same number of unscheduled orders for all instances, although better weighted tardiness and production costs are achieved with 100 iterations for 3 instances (see $I06$, $I09$, and $I11$). The number of setups is relatively small and within the target value indicated the company (which is 3). The

Table 7.1: Results for different (i) values of works in each working period.

Inst.	Unsch. Jobs			Weight. Tardiness			Prod. Cost			N. Setups		
	L	H	R	L	H	R	L	H	R	L	H	R
I01	2	0	0	68795	0	0	192	223	206	2	2	2
I02	2	0	0	1182	0	0	280	314	301	2	2	2
I03	0	0	0	7811	0	0	317	326	319	2	2	2
I04	0	0	0	209663	31738	31653	210	244	240	2	1	2
I05	3	0	0	25212	2414	2414	229	270	266	2	2	2
I06	13	1	1	156181	37174	37174	241	272	272	1	1	1
I07	3	0	0	26965	0	0	259	285	285	1	2	2
I08	0	0	0	0	0	0	91	95	92	1	2	2
I09	0	0	0	124397	46	46	291	344	344	2	2	2
I10	3	0	0	48977	0	0	318	332	328	1	2	2
I11	12	2	2	252213	255366	76081	329	380	373	1	1	1
I12	15	2	2	119295	197832	197832	340	395	395	1	1	1

average computational time in seconds for Π_{shuffle}=1, 25, and 100 is 0.4, 7.6 and 30.8, respectively, so all three configurations are fast.

Table 7.2: Results for different (ii) numbers of Π_{shuffle} iterations.

Inst.	Unsch. Jobs			Weight. Tardiness			Prod. Cost			N. Setups		
	1	25	100	1	25	100	1	25	100	1	25	100
I01	0	0	0	525	0	0	213	206	206	1	2	2
I02	1	0	0	1118	0	0	304	301	301	1	2	2
I03	0	0	0	91	0	0	334	319	319	1	2	2
I04	0	0	0	51765	31653	31653	238	240	240	2	2	2
I05	0	0	0	18895	2414	2414	268	266	266	2	2	2
I06	4	1	1	117886	37174	16019	283	272	277	1	1	2
I07	0	0	0	244	0	0	295	285	285	2	2	2
I08	0	0	0	0	0	0	92	92	92	2	2	2
I09	0	0	0	1348	46	36	332	344	316	2	2	2
I10	0	0	0	0	0	0	347	328	328	1	2	2
I11	2	2	2	76081	76081	40502	373	373	371	1	1	1
I12	3	2	2	99445	197832	197832	399	395	395	1	1	1

Finally, we present in Table 7.3 the results for parameter (iii), the Π_{list} number of iterations to change the order coefficient of priority. For these results, we selected configuration R and $\Pi_{\text{shuffle}} = 100$, and tested Π_{list} equal to 1, 5, and 20. Overall, we have conclusions similar to the previous ones, meaning that higher values of Π_{list} can provide better solutions, especially if comparing the weighted tardiness and the production cost. The average computational time in seconds for Π_{list}=1, 5, 20 is 4.7, 30.8 and 177.1, respectively, which is again fast and compatible with a real-world use of the algorithm by a decision maker.

Table 7.3: Results for different (iii) numbers of Π_{list} iterations.

Inst.	Unsch. Jobs			Weight. Tardiness			Prod. Cost			N. Setups		
	1	5	20	1	5	20	1	5	20	1	5	20
I01	0	0	0	22069	0	0	211	206	206	1	2	2
I02	0	0	0	308	0	0	313	301	301	1	2	2
I03	0	0	0	100883	0	0	327	319	318	1	2	2
I04	0	0	0	117752	31653	31650	222	240	241	2	2	2
I05	0	0	0	23168	2414	2411	265	266	264	1	2	1
I06	2	1	1	83453	16019	9137	285	277	275	1	2	2
I07	0	0	0	162	0	0	287	286	286	2	2	2
I08	0	0	0	0	0	0	92	92	92	2	2	2
I09	0	0	0	99604	36	0	302	316	292	2	2	1
I10	0	0	0	13829	0	0	321	328	328	0	2	2
I11	2	2	2	90988	40502	11029	365	371	366	1	1	1
I12	3	2	2	99445	197832	197832	399	395	395	1	1	1

7.5 Concluding Remarks

We have proposed a multi-start random heuristic for a complex integrated workforce scheduling and two-stage flexible flow shop problem from a meat production system. The heuristic schedules orders by following a list of priorities and considering different workers on machines in order to reach solutions with all orders scheduled, no tardiness, and the smallest possible production cost. The computational experiments have indicated that better solutions can be achieved when allowing more shuffles in the vector of orders (Π_{shuffle}), more iterations to handle the orders with tardiness (Π_{list}), and allowing the heuristic to decide on the number of works to set in each working period.

In future research, we are interested in studying how dynamic changes in assignments of workers to machines may affect productivity. Another direction is related to the extension of the proposed heuristic to include a local search step and possibly a tabu list in order to escape from local optima solutions. Finally, the proposed algorithm has been recently deployed to the company and, as attested by the company, has been producing satisfactory results. We are currently working with the company to provide, as future research, an extensive comparison regarding the impact of the algorithm on their production system.

Chapter 8

Conclusions

This thesis studied solution methods for a number of combinatorial optimization problems. It proposes a collection of six articles: three of them are concerned with novel solution methods for problems from the literature and for a real-world problem; two of them present an extensive literature review on well-studied areas; and the last one introduces an online library.

The first part of this thesis studied pseudo-polynomial arc flow models. Chapter 2 reviewed over 100 references and presented theoretical foundations, modeling techniques, insights behind the strength of such models, general solution methods for large models, and main successful applications. Chapter 3 proposed a number of innovative solution methods to solve large pseudo-polynomial arc flow models with strong relaxations. Our column generation algorithm, which generates multiple paths in each pricing iteration, has provided satisfactory practical performance. Our reduced-cost variable-fixing strategies could drastically reduce the model size in several difficult instances. In addition, the use of sub-optimal dual solutions allowed to strengthen the model through variable fixing, solving many difficult instances already at the root node. Finally, our innovative branching scheme, which exploits the potential of general MILP solvers, has shown to be a good approach to quickly find optimal or near-optimal solutions. The correctness of the LP-based methods and a motivation of the variable fixing strategies are supported by several theoretical results. The proposed methods compose an exact solution framework, which solves a large number of open instances of the cutting stock problem, the two-staged cutting stock problem, the ordered open-end bin packing problem, and the skiving stock problem. In all these problems, finding a near-optimal solution is already easy by means of simple greedy heuristics. Thus, reduced-cost variable-fixing is very effective already at the root node. In Chapter 4, we studied a parallel machine scheduling problem with the objective of minimizing weighted completion times. This problem is associated with a strong LP relaxation, but finding a near-optimal solution is generally not trivial. For that, we implemented a reduced-cost-based branching rule, which simulates the effectiveness of reduced-cost variable-fixing based on a near-optimal solution, without the need to have such a solution already at hand. This chapter also proposed other solution methods, which were combined into a three-phase branching scheme to solve the scheduling problem. The overall algorithm could solve all open instances to proven optimality. We believe that the new proposals are promising to solve arc flow models with strong relaxations for

many other difficult problems. We are currently investigating cutting planes for general arc flow models. We believe that the proposal of strong cutting planes for such models can be a promising direction to strengthen arc flow models with weak LP relaxations and successfully solve such models in practice employing the methods proposed here. Another future research direction is the combination of the proposed methods with iterative aggregation/disaggregation techniques, to address problems that derive huge networks that cannot fit in the computer memory.

The second part of this thesis gives special attention to two-dimensional cutting and packing problems. Chapter 5 surveyed over 180 references related to such problems, focusing mainly on exact methods and relaxations. We noticed that the literature on such problems has grown substantially in the last two decades. Thus, our survey is a significant update of the last extensive review concerning the same problems, proposed almost two decades ago. We noticed that in the last years there was an increase in popularity in Constraint Programming and Benders' Decomposition in the solution of two-dimensional cutting and packing. We believe that, although the literature on such problems has been growing considerably, there is still room for improving the state-of-the-art methods and extending the solution techniques to address three-dimensional problems. In Chapter 6, we introduce the `2DPackLib`, an online library dedicated to two-dimensional cutting and packing problems. The library systematically arranges relevant research material related to such problems. In particular, the `2DPackLib` provides a unified format for the main benchmark sets (comprising over 3000 instances). This addresses previous issues related to benchmarks that were originally published in personal web pages which no longer exist or benchmarks that are available in formats that are not clearly detailed. We are confident that all such contributions will facilitate research on the active area of two-dimensional cutting and packing.

In the third and last part of this thesis, Chapter 7 studied a real-world problem consisting of workforce scheduling and flexible flow shop arising from a meat producing company. This problem is very complex from a practical point of view, consisting of a large number of decision variables and constraints. To solve the problem, we proposed a two-phase constructive heuristic, which in the first phase determines the workforce allocation, and in the second phase schedule the orders to the machines. To improve the optimization, the constructive heuristic was embedded into a random multi-start framework. As attested by the company, the proposed algorithm has been producing satisfactory results. We are currently working with the company to provide, as future research, an extensive comparison regarding the impact of the algorithm on their production system. We are also investigating an extension of the algorithm by embedding the constructive heuristic within a metaheuristic framework (e.g., variable neighborhood search).

Related projects done in parallel with this thesis

We worked on a few projects in parallel to the development of this thesis. We have been working on the solution of an integrated lot sizing and two-dimensional cutting stock problem under demand uncertainty. We propose a two-stage stochastic model and a robust optimization model. The linear relaxation of the models are solved by column

generation, and integer solutions are obtained by a rolling horizon heuristic combined with a restricted master heuristic. This project is joint work with Eduardo Curcio, Flávio K. Miyazawa, Elsa Silva, and Pedro Amorim. A full article derived from this project is currently under review by an international indexed journal [98]. We believe that the non-deterministic nature of the studied problem diverges from the main focus of this thesis. For this reason, we did not include the resulting article as a chapter here.

We also worked on a new set of patterns for the bin packing problem. These patterns can be used to derive reduced networks in arc flow models. The proposed set of patterns derives, in average, networks that are 40% smaller, when compared to the state-of-the-art (i.e., the *meet-in-the-middle patterns* [93]). This project is joint work with Thiago A. de Queiroz, Manuel Iori, and Flávio K. Miyazawa. A preliminary report has been presented at the *LI Simpósio Brasileiro de Pesquisa Operacional* [102]. The report presents only the results of preliminary experiments, in which we compare the size of the networks resulting from different patterns, but do not compare the solution of the corresponding arc flow models. An interesting research direction is related to the use of the set of patterns in [102] to improve the waste-limited network for the bin packing problem, presented in Chapter 3.

Acknowledgements

We would like to thank the São Paulo Research Foundation (FAPESP) for the financial support (process number 2017/11831-1).

Bibliography

[1] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network flows: theory, algorithms, and applications*. Prentice-Hall, 1993.

[2] R. Alvarez-Valdes, M.A. Carravilla, and J.F. Oliveira. Cutting and packing. In R. Martí, P.M. Pardalos, and M.G.C. Resende, editors, *Handbook of Heuristics*, pages 931–977. Springer, 2018.

[3] R. Alvarez-Valdés, A. Parajón, and J. M. Tamarit. A tabu search algorithm for large-scale guillotine (un)constrained two-dimensional cutting problems. *Computers & Operations Research*, 29(7):925–947, 2002.

[4] R. Alvarez-Valdes, F. Parreño, and J. M. Tamarit. A branch-and-cut algorithm for the pallet loading problem. *Computers & Operations Research*, 32(11):3007–3029, 2005.

[5] R. Alvarez-Valdes, F. Parreño, and J. M. Tamarit. A tabu search algorithm for a two-dimensional non-guillotine cutting problem. *European Journal of Operational Research*, 183(3):1167–1182, 2007.

[6] R. Alvarez-Valdes, F. Parreño, and J. M. Tamarit. Reactive GRASP for the strip-packing problem. *Computers & Operations Research*, 35(4):1065–1083, 2008.

[7] R. Alvarez-Valdes, F. Parreño, and J. M. Tamarit. A branch and bound algorithm for the strip packing problem. *OR Spectrum*, 31(2):431–459, 2009.

[8] C. Alves, F. Clautiaux, J. M. Valério de Carvalho, and J. Rietz. *Dual-Feasible Functions for Integer Programming and Combinatorial Optimization*. Springer International Publishing, Cham, 2016.

[9] C. Alves and J.M. Valério de Carvalho. A stabilized branch-and-price-and-cut algorithm for the multiple length cutting stock problem. *Computers & Operations Research*, 35(4):1315–1328, 2008.

[10] H. Ben Amor, J. Desrosiers, and J. M. Valério de Carvalho. Dual-optimal inequalities for stabilized column generation. *Operations Research*, 54(3):454–463, 2006.

[11] D.L. Applegate, R.E. Bixby, V. Chvatal, and W.J. Cook. *The Traveling Salesman Problem - A Computational Study*. Princeton University Press, Princeton, NJ, 2006.

[12] Y. Arahori, T. Imamichi, and H. Nagamochi. An exact strip packing algorithm based on canonical forms. *Computers & Operations Research*, 39(12):2991–3011, 2012.

[13] O.S. Bajgiran, A.A. Cire, and L.-M. Rousseau. A first look at picking dual variables for maximizing reduced cost fixing. In D. Salvagnin and M. Lombardi, editors, *Integration of AI and OR Techniques in Constraint Programming*, pages 221–228. Springer International Publishing, 2017.

[14] B. S. Baker, E. G. Coffman, Jr., and R. L. Rivest. Orthogonal packing in two dimensions. *SIAM Journal on Computing*, 9(4):846–855, 1980.

[15] R. Baldacci and M. A. Boschetti. A cutting-plane approach for the two-dimensional orthogonal non-guillotine cutting problem. *European Journal of Operational Research*, 183(3):1136–1149, 2007.

[16] J. Balogh, J. Békési, G. Dósa, L. Epstein, and A. Levin. Lower bounds for several online variants of bin packing. In *Approximation and Online Algorithms - 15th International Workshop, WAOA 2017, Revised Selected Papers*, pages 102–117. Springer Verlag, 2018.

[17] M.S. Bazaraa, J.J. Jarvis, and H.D. Sherali. *Linear programming and network flows*. John Wiley & Sons, 2011.

[18] J. E. Beasley. Algorithms for unconstrained two-dimensional guillotine cutting. *Journal of the Operational Research Society*, 36(4):297–306, 1985.

[19] J. E. Beasley. Or-library: distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41(11):1069–1072, 1990.

[20] J.E. Beasley. An exact two-dimensional non-guillotine cutting tree search procedure. *Operations Research*, 33(1):49–64, 1985.

[21] J.E. Beasley. A population heuristic for constrained two-dimensional non-guillotine cutting. *European Journal of Operational Research*, 156(3):601–627, 2004.

[22] A. Bekrar, I. Kacem, C. Chu, and C. Sadfi. An improved heuristic and an exact algorithm for the 2D strip and bin packing problem. *International Journal of Product Development*, 10(1-3):217–240, 2010.

[23] R. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[24] G. Belov, V. M. Kartak, H. Rohling, and G. Scheithauer. One-dimensional relaxations and LP bounds for orthogonal packing. *ITOR*, 16(6):745–766, 2009.

[25] G. Belov, V. M. Kartak, H. Rohling, and G. Scheithauer. Conservative scales in packing problems. *OR Spectrum*, 35(2):505–542, 2013.

[26] G. Belov and H. Rohling. LP bounds in an interval-graph algorithm for orthogonal-packing feasibility. *Operations Research*, 61(2):483–497, 2013.

[27] G. Belov and G. Scheithauer. A branch-and-cut-and-price algorithm for one-dimensional stock cutting and two-dimensional two-stage cutting. *European Journal of Operational Research*, 171(1):85–106, 2006.

[28] J. Ben-Ameur, W.and Neto. Acceleration of cutting-plane and column generation algorithms: Applications to network design. *Networks*, 49(1):3–17, 2007.

[29] H. Ben Amor, J. Desrosiers, and J.M. Valério de Carvalho. Dual-optimal inequalities for stabilized column generation. *Operations Research*, 54(3):454–463, 2006.

[30] J. F. Benders. Partitioning procedures for solving mixed variables programming problems. *Numerische Mathematik*, 4(1):238–252, 1962.

[31] B. E. Bengtsson. Packing rectangular pieces – a heuristic approach. *The Computer Journal*, 25:353–357, 1982.

[32] M. Benkirane, F. Clautiaux, J. Damay, and B. Detienne. A Hypergraph Model for the Rolling Stock Rotation Planning and Train Selection. working paper or preprint, December 2019.

[33] J. A. Bennell and J. F. Oliveira. The geometry of nesting problems: A tutorial. *European Journal of Operational Research*, 184(2):397–415, 2008.

[34] J. A. Bennell and J. F. Oliveira. A tutorial in irregular shape packing problems. *Journal of the Operational Research Society*, 60(supp 1):S93–S105, 2009.

[35] J. A. Bennell, J. F. Oliveira, and G. Wäscher. Cutting and packing. *International Journal of Production Economics*, 145(2):449–450, 2013.

[36] D. Bergman, A.A. Cire, and W.J. van Hoeve. Lagrangian bounds from decision diagrams. *Constraints*, 20:346–361, 2015.

[37] J. O. Berkey and P. Y. Wang. Two dimensional finite bin packing algorithms. *Journal of the Operational Research Society*, 38:423–429, 1987.

[38] A. Bettinelli, A. Ceselli, and G. Righini. A branch-and-price algorithm for the two-dimensional level strip packing problem. *4OR*, 6(4):361–374, 2008.

[39] V. M. R. Bezerra, A. A. S. Leao, J. F. Oliveira, and M. O. Santos. Models for the two-dimensional level strip packing problem - a review and a computational evaluation. *Journal of the Operational Research Society*, 71(4):606–627, 2020.

[40] N. Boland, M. Hewitt, L. Marshall, and M. Savelsbergh. The continuous-time service network design problem. *Operations Research*, 65(5):1303–1321, 2017.

[41] N. Boland, M. Hewitt, D.M. Vu, and M. Savelsbergh. Solving the traveling salesman problem with time windows through dynamically generated time-expanded networks. In D. Salvagnin and M. Lombardi, editors, *CPAIOR: Integration of AI and OR Techniques in Constraint Programming*, pages 254–262. Springer International Publishing, 2017.

[42] N.L. Boland and M.W.P. Savelsbergh. Perspectives on integer programming for time-dependent models. *TOP*, 27:147–173, 2019.

[43] B. Bolsi, V.L. de Lima, T.A. de Queiroz, and M. Iori. Integrated workforce scheduling and flexible flow shop problem in the meat industry. In A. Dolgui, A. Bernard, D. Lemoine, G. von Cieminski, and D. Romero, editors, *Advances in Production Management Systems. Artificial Intelligence for Sustainable and Resilient Production Systems*, pages 594–602. Springer International Publishing, 2021.

[44] I. Borgulya. A parallel hyper-heuristic approach for the two-dimensional rectangular strip-packing problem. *Journal of computing and information technology*, 22(4):251–265, 2014.

[45] A. Bortfeldt and G. Wäscher. Constraints in container loading – A state-of-the-art review. *European Journal of Operational Research*, 229(1):1–20, 2013.

[46] M. A. Boschetti, E. Hadjiconstantinou, and A. Mingozzi. New upper bounds for the two-dimensional othogonal non guillotine cutting stock problem. *IMA Journal of Management Mathematics*, 13(2):95–119, 2002.

[47] M. A. Boschetti and A. Mingozzi. The two-dimensional finite bin packing problem. Part I: New lower bounds for the oriented case. *4OR*, 1(1):27–42, 2003.

[48] M. A. Boschetti and A. Mingozzi. The two-dimensional finite bin packing problem. Part II: New lower and upper bounds. *4OR*, 1(2):135–147, 2003.

[49] M. A. Boschetti and L. Montaletti. An exact algorithm for the two-dimensional strip-packing problem. *Operations Research*, 58(6):1774–1791, 2010.

[50] H. Bouarab, I. El Hallaoui, A. Metrane, and F. Soumis. Dynamic constraint and variable aggregation in column generation. *European Journal of Operational Research*, 262(3):835–850, 2017.

[51] E.A. Boyd. A pseudopolynomial network flow formulation for exact knapsack separation. *Networks*, 22(5):503–514, 1992.

[52] N. Braga, C. Alves, and R. Macedo. Exact solution of the multi-trip inventory routing problem using a pseudo-polynomial model. In *6th International Conference on Operations Research and Enterprise Systems*, volume 2, pages 250–257, 2017.

[53] N. Braga, C. Alves, R. Macedo, and J. Valério de Carvalho. Combined cutting stock and scheduling: a matheuristic approach. *International Journal of Innovative Computing and Applications*, 7(3):135–146, 2016.

[54] F. Brandão and J.P. Pedroso. Bin packing and related problems: General arc-flow formulation with graph compression. *Computers & Operations Research*, 69:56–67, 2015.

[55] T. Bulhões, R. Sadykov, A. Subramanian, and E. Uchoa. On the exact solution of a large class of parallel machine scheduling problems. *Journal of Scheduling*, 23:411–429, 2020.

[56] R.E. Burkard, S.E Karisch, and F. Rendl. QAPLIB– a quadratic assignment problem library. *European Journal of Operational Research*, 55(1):115–119, 1991.

[57] R.E. Burkard, S.E. Karisch, and F. Rendl. QAPLIB - a quadratic assignment problem library. *Journal of Global Optimization*, 10:391–403, 1997.

[58] E. K. Burke, M. Hyde, G. Kendall, and J. Woodward. A genetic programming hyper-heuristic approach for evolving 2-D strip packing heuristics. *IEEE Transactions on Evolutionary Computation*, 14(6):942–958, 2010.

[59] E. K. Burke, M. R. Hyde, and G. Kendall. A squeaky wheel optimisation methodology for two-dimensional strip packing. *Computers & Operations Research*, 38(7):1035–1044, 2011.

[60] E. K. Burke, G. Kendall, and G. Whitwell. A new placement heuristic for the orthogonal stock-cutting problem. *Operations Research*, 52(4):655–671, 2004.

[61] V. Cacchiani and P. Toth. Nominal and robust train timetabling problems. *European Journal of Operational Research*, 219(3):727–737, 2012.

[62] H. Cambazard and B. O'Sullivan. Propagating the bin packing constraint using linear programming. In D. Cohen, editor, *Principles and Practice of Constraint Programming*, pages 129–136. Springer Berlin Heidelberg, 2010.

[63] P. Cappanera and M.G. Scutellà. Joint assignment, scheduling, and routing models to home care optimization: a pattern-based approach. *Transportation Science*, 49(4):830–852, 2015.

[64] A. Caprara, M. Dell'Amico, J.C. Díaz-Díaz, M. Iori, and R. Rizzi. Friendly bin packing instances without integer round-up property. *Mathematical Programming*, 150:5–17, 2015.

[65] A. Caprara, M. Fischetti, and P. Toth. Modeling and solving the train timetabling problem. *Operations Research*, 50(5):851–861, 2002.

[66] A. Caprara and M. Monaci. On the two-dimensional knapsack problem. *Operations Research Letters*, 32(1):5–14, 2004.

[67] A. Caprara and M. Monaci. Bidimensional packing by bilinear programming. *Mathematical Programming*, 118(1):75–108, 2009.

[68] P. M. Castro and I. E. Grossmann. From time representation in scheduling to the solution of strip packing problems. *Computers & Chemical Engineering*, 44:45–57, 2012.

[69] P. M. Castro and J. F. Oliveira. Scheduling inspired models for two-dimensional packing problems. *European Journal of Operational Research*, 215(1):45–56, 2011.

[70] D. Cattaruzza, N. Absi, and D. Feillet. Vehicle routing problems with multiple trips. *4OR*, 14(3):223–259, 2016.

[71] A. Ceselli and G. Righini. An optimization algorithm for the ordered open-end bin-packing problem. *Operations Research*, 56(2):425–436, 2008.

[72] B. Chazelle. The bottomn-left bin-packing heuristic: An efficient implementation. *IEEE Transactions on Computers*, C-32(8):697–707, 1983.

[73] C. S. Chen, S. M. Lee, and Q. S. Shen. An analytical model for the container loading problem. *European Journal of Operational Research*, 80(1):68–76, 1995.

[74] H. I. Christensen, A. Khan, S. Pokutta, and P. Tetali. Approximation and online algorithms for multidimensional bin packing: A survey. *Computer Science Review*, 24:63–79, 2017.

[75] N. Christofides and E. Hadjiconstantinou. An exact algorithm for orthogonal 2-D cutting problems using guillotine cuts. *European Journal of Operational Research*, 83(1):21–38, 1995.

[76] N. Christofides, A. Mingozzi, and P. Toth. State-space relaxation procedures for the computation of bounds to routing problems. *Networks*, 11(2):145–164, 1981.

[77] N. Christofides and C. Whitlock. An algorithm for two-dimensional cutting problems. *Operations Research*, 25(1):30–44, 1977.

[78] G. F. Cintra, F. K. Miyazawa, Y. Wakabayashi, and E. C. Xavier. Algorithms for two-dimensional cutting stock and strip packing problems using dynamic programming and column generation. *European Journal of Operational Research*, 191(1):61–85, 2008.

[79] A.A. Cire, A. Diamant, T. Yunes, and A. Carrasco. A network-based formulation for scheduling clinical rotations. *Production and Operations Management*, 28(5):1186–1205, 2019.

[80] F. Clautiaux, C. Alves, and J. M. Valério de Carvalho. A survey of dual-feasible and superadditive functions. *Annals of Operations Research*, 179(1):317–342, 2010.

[81] F. Clautiaux, J. Carlier, and A. Moukrim. A new exact method for the two-dimensional orthogonal packing problem. *European Journal of Operational Research*, 183(3):1196–1211, 2007.

[82] F. Clautiaux, J. Carlier, and A. Moukrim. New reduction procedures and lower bounds for the two-dimensional bin packing problem with fixed orientation. *Computers & Operations Research*, 34(8):2223–2250, 2007.

[83] F. Clautiaux, S. Hanafi, R. Macedo, M.-E. Voge, and C. Alves. Iterative aggregation and disaggregation algorithm for pseudo-polynomial network flow models with side constraints. *European Journal of Operational Research*, 258(2):467–477, 2017.

[84] F. Clautiaux, A. Jouglet, J. Carlier, and A. Moukrim. A new constraint programming approach for the orthogonal packing problem. *Computers & Operations Research*, 35(3):944–959, 2008.

[85] F. Clautiaux, A. Jouglet, and A. Moukrim. A new graph-theoretical model for k-dimensional guillotine-cutting problems. In Catherine C. McGeoch, editor, *Experimental Algorithms*, pages 43–54, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[86] F. Clautiaux, R. Sadykov, F. Vanderbeck, and Q. Viaud. Combining dynamic programming with filtering to solve a four-stage two-dimensional guillotine-cut bounded knapsack problem. *Discrete Optimization*, 29:18–44, 2018.

[87] G. Codato and M. Fischetti. Combinatorial Benders' cuts for mixed-integer linear programming. *Operations Research*, 54(4):756–766, 2006.

[88] E. G. Coffman Jr., J. Csirik, G. Galambos, S. Martello, and D. Vigo. *Bin Packing Approximation Algorithms: Survey and Classification*, pages 455–531. Springer New York, New York, NY, 2013.

[89] M. Conforti, G. Cornuéjols, and G. Zambelli. Extended formulations in combinatorial optimization. *4OR*, 8:1–48, 2010.

[90] G. Costa, M. Delorme, M. Iori, E. Malaguti, and S. Martello. Training software for orthogonal packing problems. *Computers & Industrial Engineering*, 111:139–147, 2017.

[91] J.-F. Côté, M. Dell'Amico, and M. Iori. Combinatorial Benders' cuts for the strip packing problem. *Operations Research*, 62(3):643–661, 2014.

[92] J.-F. Côté, M. Gendreau, and J.-Y. Potvin. An exact algorithm for the two-dimensional orthogonal packing problem with unloading constraints. *Operations Research*, 62(5):1126–1141, 2014.

[93] J.-F. Côté and M. Iori. The meet-in-the-middle principle for cutting and packing problems. *INFORMS Journal on Computing*, 30(4):646–661, 2018.

[94] T. G. Crainic, G. Perboli, and R. Tadei. Recent advances in multi-dimensional packing problems. In Constantin Volosencu, editor, *New Technologies - Trends, Innovations and Research*. IntechOpen, Rijeka, 2012.

[95] Y. Cui and Z. Zhao. Heuristic for the rectangular two-dimensional single stock size cutting stock problem with two-staged patterns. *European Journal of Operational Research*, 231(2):288–298, 2013.

[96] Y.-P. Cui, Y. Zhou, and Y. Cui. Triple-solution approach for the strip packing problem with two-staged patterns. *Journal of Combinatorial Optimization*, 34(2):588–604, 2017.

[97] V.-D. Cung, M. Hifi, and B. Le Cun. Constrained two-dimensional cutting stock problems a best-first branch-and-bound algorithm. *International Transactions in Operational Research*, 7(3):185–210, 2000.

[98] E. Curcio, V.L. de Lima, F.K. Miyazawa, E. Silva, and P. Amorim. The integrated lot-sizing and cutting stock problem under demand uncertainty. *Submited*, pages 1–35, 2021.

[99] G. B. Dantzig. Discrete variable extremum problems. *Operations Research*, 5(2):266–277, 1957.

[100] G.B. Dantzig and P. Wolfe. The decomposition algorithm for linear programs. *Econometrica*, 29(4):767–778, 1961.

[101] V.L. de Lima, C. Alves, F. Clautiaux, M. Iori, and J.M. Valério de Carvalho. Arc flow formulations based on dynamic programming: Theoretical foundations and applications. *European Journal of Operational Research*, 296(1):3–21, 2022.

[102] V.L. de Lima, T.A. de Queiroz, M. Iori, and F.K. Miyazawa. Improved sets of points for the bin packing problem. In *LI Simpósio Brasileiro de Pesquisa Operacional*. Galoá, 2019.

[103] V.L. de Lima, M. Iori, and F.K. Miyazawa. Exact solution of network flow models with strong relaxations. https://arxiv.org/abs/2105.14961, 2021.

[104] V.L. de Lima, M. Iori, and F.K. Miyazawa. New exact techniques applied to a class of network flow formulations. In M. Singh and D.P. Williamson, editors, *Integer Programming and Combinatorial Optimization*, pages 178–192, Cham, 2021. Springer International Publishing.

[105] M. Dell'Amico, M. Delorme, M. Iori, and S. Martello. Mathematical models and decomposition methods for the multiple knapsack problem. *European Journal of Operational Research*, 274(3):886–899, 2019.

[106] M. Dell'Amico and S. Martello. Optimal scheduling of tasks on identical parallel processors. *ORSA Journal on Computing*, 7(2):191–200, 1995.

[107] M. Delorme and M. Iori. Enhanced pseudo-polynomial formulations for bin packing and cutting stock problems. *INFORMS Journal on Computing*, 32(1):101–119, 2020.

[108] M. Delorme and M. Iori. Enhanced pseudo-polynomial formulations for bin packing and cutting stock problems. *INFORMS Journal on Computing*, 32(1):101–119, 2020.

[109] M. Delorme, M. Iori, and S. Martello. Bin packing and cutting stock problems: Mathematical models and exact algorithms. *European Journal of Operational Research*, 255(1):1–20, 2016.

[110] M. Delorme, M. Iori, and S. Martello. Logic based Benders' decomposition for orthogonal stock cutting problems. *Computers & Operations Research*, 78:290–298, 2017.

[111] M. Delorme, M. Iori, and S. Martello. BPPLIB: a library for bin packing and cutting stock problems. *Optimization Letters*, 12(2):235–250, 2018.

[112] G. Desaulniers, J. Desrosiers, and M.M. Solomon. *Column Generation*. Springer Science & Business Media, 2006.

[113] G. Desaulniers, T. Gschwind, and S. Irnich. Variable fixing for two-arc sequences in branch-price-and-cut algorithms on path-based models. *Transportation Science*, 54(5):1153–1438, 2020.

[114] M. Desrochers, J. Desrosiers, and M. Solomon. A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research*, 40(2):342–354, 1992.

[115] M. Dolatabadi, A. Lodi, and M. Monaci. Exact algorithms for the two-dimensional guillotine knapsack. *Computers & Operations Research*, 39(1):48–53, 2012.

[116] K. A. Dowsland and W. B. Dowsland. Solution approaches to irregular nesting problems. *European Journal of Operational Research*, 84:506–521, 1995.

[117] O. du Merle, D. Villeneuve, J. Desrosiers, and P. Hansen. Stabilized column generation. *Discrete Mathematics*, 194(1):229–237, 1999.

[118] H. Dyckhoff. A new linear programming approach to the cutting stock problem. *Operations Research*, 29(6):1092–1104, 1981.

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

[120] I. Elhallaoui, A. Metrane, G. Desaulniers, and F. Soumis. An improved primal simplex algorithm for degenerate linear programs. *INFORMS Journal on Computing*, 23(4):569–577, 2011.

[121] I. Elhallaoui, D. Villeneuve, F. Soumis, and G. Desaulniers. Dynamic aggregation of set-partitioning constraints in column generation. *Operations Research*, 53(4):632–645, 2005.

[122] Hamilton Emmons and George Vairaktarakis. *The Hybrid Flow Shop*, pages 161–187. Springer US, Boston, MA, 2013.

[123] G.D. Eppen and R.K. Martin. Solving multi-item capacitated lot-sizing problems using variable redefinition. *Operations Research*, 35(6):832–848, 1987.

[124] D. Fayard, M. Hifi, and V. Zissimopoulos. An efficient approach for large-scale two-dimensional guillotine cutting stock problems. *Journal of the Operational Research Society*, 49(12):1270–1277, 1998.

[125] S. P. Fekete and J. Schepers. New classes of fast lower bounds for bin packing problems. *Mathematical Programming*, 91(1):11–31, 2001.

[126] S. P. Fekete and J. Schepers. A combinatorial characterization of higher-dimensional orthogonal packing. *Mathematics of Operations Research*, 29(2):353–368, 2004.

[127] S. P. Fekete and J. Schepers. A general framework for bounds for higher-dimensional orthogonal packing problems. *Mathematical Methods of Operations Research*, 60(2):311–329, 2004.

[128] S. P. Fekete, J. Schepers, and J. C. van der Veen. An exact algorithm for higher-dimensional orthogonal packing. *Operations Research*, 55(3):569–587, 2007.

[129] S.P. Fekete and J. Schepers. A new exact algorithm for general orthogonal d-dimensional knapsack problems. In R. Burkard and G. Woeginger, editors, *Algorithms – ESA*, volume 1284, pages 144–156. Springer, 1997.

[130] E. P. Ferreira and J. F. Oliveira. Fekete and Schepers' graph-based algorithm for the two-dimensional orthogonal packing problem revisited. In Bortfeldt A., Homberger J., Kopfer H., Pankratz G., and Strangmeier R., editors, *Intelligent Decision Support*, pages 15–31. Gabler, Wiesbaden, 2008.

[131] F. Fischer and C. Helmberg. Dynamic graph generation for the shortest path problem in time expanded networks. *Mathematical Programming*, 143:257–297, 2014.

[132] M. Fischetti and A. Lodi. Local branching. *Mathematical Programming*, 98:23–47, 2003.

[133] K. Fleszar. An exact algorithm for the two-dimensional stage-unrestricted guillotine cutting/packing decision problem. *INFORMS Journal on Computing*, 28(4):703–720, 2016.

[134] L.R., Jr. Ford and D.R. Fulkerson. Constructing maximal dynamic flows from static flows. *Operations Research*, 6(3):419–433, 1958.

[135] L.R, Jr. Ford and D.R. Fulkerson. A suggested computation for maximal multi-commodity network flows. *Management Science*, 5(1):97–101, 1958.

[136] I. Friedow and G. Scheithauer. Using contiguous 2D-feasible 1D cutting patterns for the 2D strip packing problem. In *Operations Research Proceedings 2015*, pages 71–77. Springer International Publishing, 2017.

[137] D. K. Friesen and M. A. Langston. Variable sized bin packing. *SIAM journal on computing*, 15(1):222–230, 1986.

[138] R. Fukasawa, H. Longo, J. Lysgaard, M.P. De Aragão, M. Reis, E. Uchoa, and R.F. Werneck. Robust branch-and-cut-and-price for the capacitated vehicle routing problem. *Mathematical programming*, 106(3):491–511, 2006.

[139] F. Furini and E. Malaguti. Models for the two-dimensional two-stage cutting stock problem with multiple stock size. *Computers & Operations Research*, 40(8):1953–1962, 2013.

[140] F. Furini, E. Malaguti, R. M. Durán, A. Persiani, and P. Toth. A column generation heuristic for the two-dimensional two-staged guillotine cutting stock problem with multiple stock size. *European Journal of Operational Research*, 218(1):251–260, 2012.

[141] F. Furini, E. Malaguti, and D. Thomopoulos. Modeling two-dimensional guillotine cutting problems via integer programming. *INFORMS Journal on Computing*, 28(4):736–751, 2016.

[142] V. Gabrel and C. Murat. Mathematical programming for earth observation satellite mission planning. In T.A. Ciriani, G. Fasano, S. Gliozzi, and R. Tadei, editors, *Operations Research in Space and Air*, pages 103–122. Springer US, 2003.

[143] W. Gálvez, F. Grandoni, S. Heydrich, S. Ingala, A. Khan, and A. Wiese. Approximating geometric knapsack via l-packings. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 260–271, 2017.

[144] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, San Francisco, 1979.

[145] M. Gendreau, M. Iori, G. Laporte, and S. Martello. A tabu search algorithm for a routing and container loading problem. *Transportation Science*, 40(3):342–350, 2006.

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

[147] A. M. Geoffrion. Generalized Benders decomposition. *Journal of Optimization Theory and Applications*, 10(4):237–260, 1972.

[148] P. C. Gilmore and R. E. Gomory. Multistage cutting stock problems of two and more dimensions. *Operations Research*, 13(1):94–120, 1965.

[149] P.C. Gilmore and R.E. Gomory. A linear programming approach to the cutting-stock problem. *Operations Research*, 9(6):849–859, 1961.

[150] P.C. Gilmore and R.E. Gomory. A linear programming approach to the cutting stock problem - part II. *Operations Research*, 11(6):863–888, 1963.

[151] A.M. Gleixner, T. Berthold, B. Müller, and S. Weltge. Three enhancements for optimization-based bound tightening. *Journal of Global Optimization*, 67(4):731–757, 2017.

[152] M.T. Godinho, L. Gouveia, T. Magnanti, P. Pesneau, and J. Pires. On time-dependent models for unit demand vehicle routing problems. In *International Network Optimization Conference (INOC)*, 2007.

[153] J. F. Gonçalves and M. G. C. Resende. A biased random key genetic algorithm for 2D and 3D bin packing problems. *International Journal of Production Economics*, 145(2):500–510, 2013.

[154] L. Gouveia, M. Leitner, and M. Ruthmair. Layered graph approaches for combinatorial optimization problems. *Computers & Operations Research*, 102:22–38, 2019.

[155] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G.R. Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[156] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. In P.L. Hammer, E.L. Johnson, and B.H. Korte, editors, *Discrete Optimization II*, volume 5 of *Annals of Discrete Mathematics*, pages 287–326. Elsevier, 1979.

[157] S. Grandcolas and C. Pinto. A sat encoding for multi-dimensional packing problems. In Andrea Lodi, Michela Milano, and Paolo Toth, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 141–146, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[158] C. Groër, B. Golden, and E. Wasil. A library of local search heuristics for the vehicle routing problem. *Mathematical Programming Computation*, 2:79–101, 2010.

[159] S. Gualandi and F. Malucelli. Exact solution of graph coloring problems via constraint programming and column generation. *INFORMS Journal on Computing*, 24(1):81–100, 2012.

[160] A. Hadjar, O. Marcotte, and F. Soumis. A branch-and-cut algorithm for the multiple depot vehicle scheduling problem. *Operations Research*, 54(1):130–149, 2006.

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

[162] E. Hadjiconstantinou and M. Iori. A hybrid genetic algorithm for the two-dimensional single large object placement problem. *European Journal of Operational Research*, 183(3):1150–1166, 2007.

[163] X. Han, K. Iwama, D. Ye, and G. Zhang. Approximate strip packing: Revisited. *Information and Computation*, 249:110–120, 2016.

[164] S. Held, W. Cook, and E.C. Sewell. Maximum-weight stable sets and safe lower bounds for graph coloring. *Mathematical Programming Computation*, 4:363–381, 2012.

[165] S. Henning, K. Jansen, M. Rau, and L. Schmarje. Complexity and inapproximability results for parallel task scheduling and strip packing. *Theory of Computing Systems*, 64:120–140, 2020.

[166] J.C. Herz. Recursive computational procedure for two-dimensional stock cutting. *IBM Journal of Research and Development*, 16:462–469, 1972.

[167] M. Hifi. An improvement of Viswanathan and Bagchi's exact algorithm for constrained two-dimensional cutting stock. *Computers & Operations Research*, 24(8):727–736, 1997.

[168] M. Hifi. Exact algorithms for large-scale unconstrained two and three staged cutting problems. *Combinatorial Optimization and Applications*, 18:63–88, 2001.

[169] J. N. Hooker and G. Ottosson. Logic-based Benders decomposition. *Mathematical Programming*, 96(1):33–60, 2003.

[170] J.N. Hooker. Decision diagrams and dynamic programming. In *International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 94–110. Springer, 2013.

[171] E. Hopper and B. Turton. Problem generators for rectangular packing problems. *Stud. Inform. Univ.*, 2:123–136, 2002.

[172] E. Hopper and B.C.H. Turton. An empirical investigation of meta-heuristic and heuristic algorithms for a 2D packing problem. *European Journal of Operational Research*, 128(1):34–57, 2001.

[173] F.J. Hwang and B.M.T. Lin. Survey and extensions of manufacturing models in two-stage flexible flow shops with dedicated machines. *Computers & Operations Research*, 98:103–112, 2018.

[174] S. Imahori and M. Yagiura. The best-fit heuristic for the rectangular strip packing problem: An efficient implementation and the worst-case approximation ratio. *Computers & Operations Research*, 37(2):325–333, 2010.

[175] M. Iori, V.L. de Lima, S. Martello, F.K. Miyazawa, and M. Monaci. Exact solution techniques for two-dimensional cutting and packing. *European Journal of Operational Research*, 289(2):399–415, 2021.

[176] M. Iori, V.L. de Lima, S. Martello, and M. Monaci. 2DPackLib: A two-dimensional cutting and packing library. *Optimization Letters*, 2021 (forthcoming).

[177] M. Iori and S. Martello. Routing problems with loading constraints. *TOP*, 18(1):4–27, 2010.

[178] M. Iori and S. Martello. An annotated bibliography of combined routing and loading problems. *Yugoslav Journal of Operations Research*, 23(3):311–326, 2013.

[179] M. Iori, S. Martello, and M. Monaci. *Metaheuristic Algorithms for the Strip Packing Problem*, pages 159–179. Springer US, 2003.

[180] M. Iori, J. J. Salazar González, and D. Vigo. An exact approach for the vehicle routing problem with two-dimensional loading constraints. *Transportation Science*, 41:253–264, 2007.

[181] M. Iori, J.-J. Salazar-González, and D. Vigo. An exact approach for the vehicle routing problem with two-dimensional loading constraints. *Transportation Science*, 41(2):253–264, 2007.

[182] S. Irnich and G. Desaulniers. Shortest path problems with resource constraints. In G. Desaulniers, J. Desrosiers, and M.M. Solomon, editors, *Column Generation*, pages 33–65. Springer US, Boston, MA, 2005.

[183] S. Irnich, G. Desaulniers, J. Desrosiers, and A. Hadjar. Path-reduced costs for eliminating arcs in routing and scheduling. *INFORMS Journal on Computing*, 22(2):297–313, 2010.

[184] S. Irnich, P. Toth, and D. Vigo. The family of vehicle routing problems. In P. Toth and D. Vigo, editors, *Vehicle Routing: Problems, Methods, and Applications*, chapter 1, pages 1–33. SIAM, 2nd edition, 2014.

[185] S. Jakobs. On genetic algorithms for the packing of polygons. *European Journal of Operational Research*, 88(1):165–181, 1996.

[186] K. Jansen and M. Rau. Closing the gap for pseudo-polynomial strip packing. 2019. (presented at ESA 2019, Munich). `http://arxiv.org/abs/1712.04922`.

[187] D. S. Johnson. *Near-optimal bin packing algorithms*. PhD thesis, MIT, Cambridge, MA, 1973.

[188] C. Joncour and A. Pêcher. Consecutive ones matrices for multi-dimensional orthogonal packing problems. *Electronic Notes in Discrete Mathematics*, 36:327–334, 2010.

[189] C. Joncour, A. Pêcher, and P. Valicov. MPQ-trees for the orthogonal packing problem. *Electronic Notes in Discrete Mathematics*, 11:423–429, 2012.

[190] V.M. Kartak, A.V. Ripatti, G. Scheithauer, and S. Kurz. Minimal proper non-IRUP instances of the one-dimensional cutting stock problem. *Discrete Applied Mathematics*, 187:120–129, 2015.

[191] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer, Berlin, Germany, 2004.

[192] M. Kenmochi, T. Imamichi, K. Nonobe, M. Yagiura, and H. Nagamochi. Exact algorithms for the two-dimensional strip packing problem with and without rotations. *European Journal of Operational Research*, 198(1):73–83, 2009.

[193] I. Kierkosz and M. Luczak. A hybrid evolutionary algorithm for the two-dimensional packing problem. *Central European Journal of Operations Research*, 22(4):729–753, 2014.

[194] T. Koch, T. Achterberg, E. Andersen, O. Bastert, T. Berthold, R.E. Bixby, E. Danna, G. Gamrath, A.M. Gleixner, S. Heinz, A. Lodi, H. Mittelmann, T. Ralphs, D. Salvagnin, D.E. Steffy, and K. Wolter. MIPLIB 2010. *Mathematical Programming Computation*, 3:103, 2011.

[195] R. E. Korf, M. D. Moffitt, and M. E. Pollack. Optimal rectangle packing. *Annals of Operations Research*, 179(1):261–295, 2010.

[196] A. Kramer, M. Dell'Amico, D. Feillet, and M. Iori. Scheduling jobs with release dates on identical parallel machines by minimizing the total weighted completion time. *Computers & Operations Research*, 123 article 105018, 2020.

[197] A. Kramer, M. Dell'Amico, and M. Iori. Enhanced arc-flow formulations to minimize weighted completion time on identical parallel machines. *European Journal of Operational Research*, 275(1):67–79, 2019.

[198] A. Kramer, M. Iori, and P. Lacomme. Mathematical formulations for scheduling jobs on identical parallel machines with family setup times and total weighted completion time minimization. *European Journal of Operational Research*, 289(3):825–840, 2021.

[199] A. Kramer, E. Lalla-Ruiz, M. Iori, and S. Voß. Novel formulations and modeling enhancements for the dynamic berth allocation problem. *European Journal of Operational Research*, 278(1):170–185, 2019.

[200] R. Kramer, M. Iori, and T. Vidal. Mathematical models and search algorithms for the capacitated-center problem. *INFORMS Journal on Computing*, 32(2):444–460, 2020.

[201] B. Kwon and G. M. Lee. Spatial scheduling for large assembly blocks in shipbuilding. *Computers & Industrial Engineering*, 89:203–212, 2015.

[202] G. Lancia, F. Rinaldi, and P. Serafini. A time-indexed LP-based approach for min-sum job-shop problems. *Annals of Operations Research*, 186:175–198, 2011.

[203] G. Lancia and P. Serafini. Deriving compact extended formulations via LP-based separation techniques. *4OR*, 12(3):201–234, 2014.

[204] A. A. S. Leao, F. M. B. Toledo, J. F. Oliveira, M. A. Carravilla, and R. Alvarez-Valdés. Irregular packing problems: A review of mathematical models. *European Journal of Operational Research*, 282(3):803–822, 2020.

[205] C. Lemaréchal. Lagrangian relaxation. In M. Jünger and D. Naddef, editors, *Computational Combinatorial Optimization: Optimal or Provably Near-Optimal Solutions*, pages 112–156. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[206] N. Lesh, J. Marks, A. McMahon, and M. Mitzenmacher. Exhaustive approaches to 2D rectangular perfect packings. *Information Processing Letters*, 90(1):7–14, 2004.

[207] S.C.H. Leung and D.Zhang. A fast layer-based heuristic for non-guillotine strip packing. *Expert Systems with Applications*, 38(10):13032–13042, 2011.

[208] Z.-T. Li, Q.-X. Chen, N. Mao, X. Wang, and J. Liu. Scheduling rules for two-stage flexible flow shop scheduling problem subject to tail group constraint. *International Journal of Production Economics*, 146(2):667–678, 2013.

[209] R. Linn and W. Zhang. Hybrid flow shop scheduling: A survey. *Computers & Industrial Engineering*, 37(1):57–61, 1999.

[210] A. Lodi, S. Martello, and M. Monaci. Two-dimensional packing problems: A survey. *European Journal of Operational Research*, 141(2):241–252, 2002.

[211] A. Lodi, S. Martello, M. Monaci, C. Cicconetti, L. Lenzini, E. Mingozzi, C. Eklund, and J. Moilanen. Efficient two-dimensional packing algorithms for mobile WiMAX. *Management Science*, 57(12):2130–2144, 2011.

[212] A. Lodi, S. Martello, M. Monaci, and D. Vigo. Two-dimensional bin packing problems. In V. Th. Paschos, editor, *Paradigms of Combinatorial Optimization: Problems and New Approaches*, pages 107–129. John Wiley & Sons, Ltd, Hoboken, NJ, USA, 2014.

[213] 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(4):345–357, 1999.

[214] A. Lodi, S. Martello, and D. Vigo. Recent advances on two-dimensional bin packing problems. *Discrete Applied Mathematics*, 123(1):379–396, 2002.

[215] A. Lodi, S. Martello, and D. Vigo. Models and bounds for two-dimensional level packing problems. *Journal of Combinatorial Optimization*, 8(3):363–379, 2004.

[216] A. Lodi, S. Martello, and D. Vigo. TSpack: a unified tabu search code for multi-dimensional bin packing problems. *Annals of Operations Research*, 131(1-4):203–213, 2004.

[217] A. Lodi and M. Monaci. Integer linear programming models for 2-staged two-dimensional knapsack problems. *Mathematical Programming*, 94(2):257–278, 2003.

[218] M.E. Lübbecke and J. Desrosiers. Selected topics in column generation. *Operations Research*, 53(6):1007–1023, 2005.

[219] R. Macedo, C. Alves, and J. M. Valério de Carvalho. Arc-flow model for the two-dimensional guillotine cutting stock problem. *Computers & Operations Research*, 37(6):991–1001, 2010.

[220] R. Macedo, C. Alves, and J.M. Valério de Carvalho. Arc-flow model for the two-dimensional guillotine cutting stock problem. *Computers & Operations Research*, 37(6):991–1001, 2010.

[221] R. Macedo, C. Alves, J.M. Valério de Carvalho, F. Clautiaux, and S. Hanafi. Solving exactly the vehicle routing problem with time windows and multiple routes using a pseudo-polynomial model. *European Journal of Operational Research*, 214(3):536–545, 2011.

[222] S. Martello. Knapsack, packing and cutting. *INFOR*, 32(4):217–218, 1994.

[223] S. Martello. Two-dimensional packing problems in telecommunications. *Pesquisa Operacional*, 34(1):31–38, 2014.

[224] S. Martello and M. Monaci. Models and algorithms for packing rectangles into the smallest square. *Computers & Operations Research*, 63:161–171, 2015.

[225] S. Martello, M. Monaci, and D. Vigo. An exact approach to the strip-packing problem. *INFORMS Journal on Computing*, 15(3):310–319, 2003.

[226] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, Inc., 1990.

[227] S. Martello and P. Toth. Lower bounds and reduction procedures for the bin packing problem. *Discrete Applied Mathematics*, 28(1):59–70, 1990.

[228] S. Martello and D. Vigo. Exact solution of the two-dimensional finite bin packing problem. *Management Science*, 44(3):388–399, 1998.

[229] M. Martin, E. G. Birgin, R. D. Lobato, R. Morabito, and P. Munari. Models for the two-dimensional rectangular single large placement problem with guillotine cuts and constrained pattern. *International Transactions in Operational Research*, 27(2):767–793, 2020.

[230] M. Martin, R. Morabito, and P. Munari. A bottom-up packing approach for modeling the constrained two-dimensional guillotine placement problem. *Computers & Operations Research*, 115:104851, 2020.

[231] R.K. Martin, R.L. Rardin, and B.A. Campbell. Polyhedral characterization of discrete dynamic programming. *Operations Research*, 38(1):127–138, 1990.

[232] J. Martinovic, M. Delorme, M. Iori, G. Scheithauer, and N. Strasdat. Improved flow-based formulations for the skiving stock problem. *Computers & Operations Research*, 113 article 104770, 2020.

[233] J. Martinovic and G. Scheithauer. Integer linear programming models for the skiving stock problem. *European Journal of Operational Research*, 251(2):356–368, 2016.

[234] J. Martinovic and G. Scheithauer. The proper relaxation and the proper gap of the skiving stock problem. *Mathematical Methods of Operations Research*, 84(3):527–548, 2016.

[235] J. Martinovic, G. Scheithauer, and J.M. Valério de Carvalho. A comparative study of the arcflow model and the one-cut model for one-dimensional cutting stock problems. *European Journal of Operational Research*, 266(2):458–471, 2018.

[236] M. Matayoshi. The 2D strip packing problem: A new approach with verification by EA. In *2010 IEEE International Conference on Systems, Man and Cybernetics*, pages 2492–2499, 2010.

[237] G. M. Melega, S. A. Araujo, and R. Jans. Classification and literature review of integrated lot-sizing and cutting stock problems. *European Journal of Operational Research*, 271(1):1–19, 2018.

[238] S. B. Messaoud, C. Chu, and M.-L. Espinouse. Characterization and modelling of guillotine constraints. *European Journal of Operational Research*, 191(1):112–126, 2008.

[239] M. Mesyagutov, G. Scheithauer, and G. Belov. LP bounds in various constraint programming approaches for orthogonal packing. *Computers & Operations Research*, 39(10):2425–2438, 2012.

[240] M. A. Mesyagutov, E. A. Mukhacheva, G. N. Belov, and G. Scheithauer. Packing of one-dimensional bins with contiguous selection of identical items: An exact method of optimal solution. *Automation and Remote Control*, 72(1):141–159, 2011.

[241] C.E. Miller, A.W. Tucker, and R.A. Zemlin. Integer programming formulation of traveling salesman problems. *Journal of the ACM*, 7(4):326–329, 1960.

[242] M. Minoux. Multicommodity network flow models and algorithms in telecommunications. In M.G.C. Resende and P.M. Pardalos, editors, *Handbook of Optimization in Telecommunications*, pages 163–184. Springer US, 2006.

[243] M. Monaci and P. Toth. A set-covering-based heuristic approach for bin-packing problems. *INFORMS Journal on Computing*, 18(1):71–85, 2006.

[244] R. Morabito and M. N. Arenales. Staged and constrained two-dimensional guillotine cutting problems: An AND/OR-graph approach. *European Journal of Operational Research*, 94(3):548–560, 1996.

[245] R. Morabito and V. A. Pureza. Heuristic approach based on dynamic programming and and/or-graph search for the constrained two-dimensional guillotine cutting problem. *Annals of Operations Research*, 179:297–315, 2010.

[246] M. Mrad. An arc flow-based optimization approach for the two-stage guillotine strip cutting problem. *Journal of the Operational Research Society*, 66(11):1850–1859, 2015.

[247] M. Mrad, I. Meftahi, and M. Haouari. A branch-and-price algorithm for the two-stage guillotine cutting stock problem. *Journal of the Operational Research Society*, 64(5):629–637, 2013.

[248] M. Mrad and N. Souayah. An arc-flow model for the makespan minimization problem on identical parallel machines. *IEEE Access*, 6:5300–5307, 2018.

[249] S. Nadarajah and A.A. Cire. Network-based approximate linear programming for discrete optimization. *SSRN Electronic Journal*, 2017.

[250] G. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization*. Wiley & Sons, 1988.

[251] V. Nesello, M. Delorme, M. Iori, and A. Subramanian. Mathematical models and decomposition algorithms for makespan minimization in plastic rolls production. *Journal of the Operational Research Society*, 69(3):326–339, 2018.

[252] V. Nesello, M. Delorme, M. Iori, and A. Subramanian. Mathematical models and decomposition algorithms for makespan minimization in plastic rolls production. *Journal of the Operational Research Society*, 69(3):326–339, 2018.

[253] N. Ntene and J. H. van Vuuren. A survey and comparison of guillotine heuristics for the 2D oriented offline strip packing problem. *Discrete Optimization*, 6(2):174–188, 2009.

[254] J. F. Oliveira, A. Neuenfeldt Júnior, E. Silva, and M. A. Carravilla. A survey on heuristics for the two-dimensional rectangular strip packing problem. *Pesquisa Operacional*, 36(2):197–226, 2016.

[255] J. F. Oliveira and G. Wäscher. Cutting and packing. *European Journal of Operational Research*, 183(3):1106–1108, 2007.

[256] J.F. Oliveira and J.S. Ferreira. An improved version of Wang's algorithm for two-dimensional cutting problems. *European Journal of Operational Research*, 44(2):256–266, 1990.

[257] H. Onodera, Y. Taniguchi, and K. Tamaru. Branch-and-bound placement for building block layout. In *28th ACM/IEEE Design Automation Conference*, pages 433–439, 1991.

[258] F. G. Ortmann and J. H. van Vuuren. Modified strip packing heuristics for the rectangular variable-sized bin packing problem. *ORiON*, 26(1):21–44, 2010.

[259] F. Parreño, R. Alvarez-Valdes, J. F. Oliveira, and J. M. Tamarit. A hybrid GRASP/VND algorithm for two- and three-dimensional bin packing. *Annals of Operations Research*, 179(1):203–220, 2010.

[260] D. Pecin, A. Pessoa, M. Poggi, and E. Uchoa. Improved branch-cut-and-price for capacitated vehicle routing. *Mathematical Programming Computation*, 9:61–100, 2017.

[261] A. Pessoa, M.P. de Aragão, and E. Uchoa. Robust branch-cut-and-price algorithms for vehicle routing problems. In B. Golden, S. Raghavan, and E. Wasil, editors, *The Vehicle Routing Problem: Latest Advances and New Challenges*, pages 297–325. Springer US, 2008.

[262] A. Pessoa, R. Sadykov, E. Uchoa, and F. Vanderbeck. Automation and combination of linear-programming based stabilization techniques in column generation. *INFORMS Journal on Computing*, 30(2):339–360, 2018.

[263] A. Pessoa, R. Sadykov, E. Uchoa, and F. Vanderbeck. A generic exact solver for vehicle routing and related problems. In A. Lodi and V. Nagarajan, editors, *Integer Programming and Combinatorial Optimization*, pages 354–369. Springer International Publishing, 2019.

[264] A. Pessoa, R. Sadykov, E. Uchoa, and F. Vanderbeck. A generic exact solver for vehicle routing and related problems. *Mathematical Programming*, 183:483–523, 2020.

[265] A. Pessoa, E. Uchoa, M.P. de Aragão, and R. Rodrigues. Exact algorithm over an arc-time-indexed formulation for parallel machine scheduling problems. *Mathematical Programming Computation*, 2:259–290, 2010.

[266] J.-C. Picard and M. Queyranne. The time-dependent traveling salesman problem and its application to the tardiness problem in one-machine scheduling. *Operations Research*, 26(1):86–110, 1978.

[267] D. Pisinger and M. Sigurd. The two-dimensional bin packing problem with variable bin sizes and costs. *Discrete Optimization*, 2(2):154–167, 2005.

[268] D. Pisinger and M. Sigurd. Using decomposition techniques and constraint programming for solving the two-dimensional bin-packing problem. *INFORMS Journal on Computing*, 19(1):36–51, 2007.

[269] M. Poggi and E. Uchoa. New exact algorithms for the capacitated vehicle routing problem. In P. Toth and D. Vigo, editors, *Vehicle Routing: Problems, Methods, and Applications*, chapter 3, pages 59–86. SIAM, 2nd edition, 2014.

[270] H. Pollaris, K. Braekers, A. Caris, G. K. Janssens, and S. Limbourg. Vehicle routing problems with loading constraints: State-of-the-art and future directions. *OR Spectrum*, 37(2):297–330, 2015.

[271] D. Porumbel. Ray projection for optimizing polytopes with prohibitively many constraints in set-covering column generation. *Mathematical Programming*, 155(1-2):147–197, 2016.

[272] J. Puchinger and G. R. Raidl. Models and algorithms for three-stage two-dimensional bin packing. *European Journal of Operational Research*, 183(3):1304–1327, 2007.

[273] T. A. de Queiroz, P. H. D. B. Hokama, R. C. S. Schouery, and F. K. Miyazawa. Two-dimensional disjunctively constrained knapsack problem: Heuristic and exact approaches. *Computers & Industrial Engineering*, 105:313–328, 2017.

[274] T. A. de Queiroz and F. K. Miyazawa. Order and static stability into the strip packing problem. *Annals of Operations Research*, 223(1):137–154, 2014.

[275] B. Ramos, C. Alves, and J. Valério de Carvalho. An arc flow formulation to the multitrip production, inventory, distribution, and routing problem with time windows. *International Transactions in Operational Research*, 2020, forthcoming.

[276] M.R. Rao. On the cutting stock problem. *Journal of the Computer Society of India*, 7:35–39, 1976.

[277] M. Ratli, R. Benmansour, R. Macedo, S. Hanafi, and C. Wilbaut. Mathematical programming and heuristics for scheduling problems with early and tardy penalties. In B. Jarboui, P. Siarry, and J. Teghem, editors, *Metaheuristics for Production Scheduling*, chapter 8, pages 183–223. John Wiley & Sons, Ltd, 2013.

[278] M. Riedler, T. Jatschka, J. Maschler, and G.R. Raidl. An iterative time-bucket refinement algorithm for a high-resolution resource-constrained project scheduling problem. *International Transactions in Operational Research*, 27(1):573–613, 2020.

[279] M. Riedler, M. Ruthmair, and G.R. Raidl. Strategies for iteratively refining layered graph models. In M.J. Blesa Aguilera, C. Blum, H. Gambini Santos, P. Pinacho-Davidson, and J. Godoy del Campo, editors, *Hybrid Metaheuristics*, pages 46–62. Springer International Publishing, 2019.

[280] J. Rietz, C. Alves, N. Braga, and J.M. Valério de Carvalho. An exact approach based on a new pseudo-polynomial network flow model for integrated planning and scheduling. *Computers & Operations Research*, 76:183–194, 2016.

[281] J. Rietz, C. Alves, and J. M. Valério de Carvalho. Theoretical investigations on maximal dual feasible functions. *Operations Research Letters*, 38(3):174–178, 2010.

[282] J. Rietz, C. Alves, and J. M. Valério de Carvalho. Worst-case analysis of maximal dual feasible functions. *Optimization Letters*, 6(8):1–19, December 2011.

[283] J. Rietz, C. Alves, and J. M. Valério de Carvalho. On the extremality of maximal dual feasible functions. *Operations Research Letters*, 40(1):25–30, 2012.

[284] G. Righini and M. Salani. New dynamic programming algorithms for the resource constrained elementary shortest path problem. *Networks*, 51(3):155–170, 2008.

[285] A. Rossi, A. Puppato, and M. Lanzetta. Heuristics for scheduling a two-stage hybrid flow shop with parallel batching machines: application at a hospital sterilisation plant. *International Journal of Production Research*, 51(8):2363–2376, 2013.

[286] R. Ruiz and J.A. Vázquez-Rodríguez. The hybrid flow shop scheduling problem. *European Journal of Operational Research*, 205(1):1–18, 2010.

[287] M. Russo, M. Boccia, A. Sforza, and C. Sterle. Constrained two-dimensional guillotine cutting problem: upper-bound review and categorization. *International Transactions in Operational Research*, 27(2):794–834, 2020.

[288] M. Russo, M. Boccia, A. Sforza, and C. Sterle. Constrained two-dimensional guillotine cutting problem: upper-bound review and categorization. *International Transactions in Operational Research*, 27(2):794–834, 2020.

[289] M. Russo, A. Sforza, and C. Sterle. An exact dynamic programming algorithm for large-scale unconstrained two-dimensional guillotine cutting problems. *Computers & Operations Research*, 50:97–114, 2014.

[290] R. Sadykov and F. Vanderbeck. Column generation for extended formulations. *EURO Journal on Computational Optimization*, 1:81–115, 2013.

[291] G. Scheithauer. Equivalence and dominance for problems of optimal packing of rectangles. *Ricerca Operativa*, 83:3–34, 1997.

[292] G. Scheithauer. *Introduction to Cutting and Packing Optimization*. Springer International Publishing, 2018.

[293] M. Sellmann. Theoretical foundations of cp-based lagrangian relaxation. In M. Wallace, editor, *Principles and Practice of Constraint Programming*, pages 634–647, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[294] M. Serairi and M. Haouari. A theoretical and experimental study of fast lower bounds for the two-dimensional bin packing problem. *RAIRO-Operations Research*, 52(2):391–414, 2018.

[295] J.F. Shapiro. Dynamic programming algorithms for the integer programming problem-I: The integer programming problem viewed as a knapsack type problem. *Operations Research*, 16(1):103–121, 1968.

[296] E. Silva, F. Alvelos, and J. M. Valério de Carvalho. An integer programming model for two- and three-stage two-dimensional cutting stock problems. *European Journal of Operational Research*, 205(3):699–708, 2010.

[297] E. Silva, F. Alvelos, and J.M. Valério de Carvalho. An integer programming model for two- and three-stage two-dimensional cutting stock problems. *European Journal of Operational Research*, 205(3):699–708, 2010.

[298] E. Silva, J. F. Oliveira, and G. Wäscher. 2DCPackGen: A problem generator for two-dimensional rectangular cutting and packing problems. *European Journal of Operational Research*, 237(3):846–856, 2014.

[299] E. Silva, J. F. Oliveira, and G. Wäscher. The pallet loading problem: a review of solution methods and computational experiments. *International Transactions in Operational Research*, 23(1-2):147–172, 2016.

[300] M. Skutella. An introduction to network flows over time. In W. Cook, L. Lovász, and J. Vygen, editors, *Research Trends in Combinatorial Optimization: Bonn 2008*, pages 451–482. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[301] T. Soh, K. Inoue, N. Tamura, M. Banbara, and H. Nabeshima. A sat-based method for solving the two-dimensional strip packing problem. *Fundamenta Informaticae*, 102(3-4):467–487, 2010.

[302] J.P. Sousa and L.A. Wolsey. A time indexed formulation of non-preemptive single machine scheduling problems. *Mathematical Programming*, 54(1):353–367, 1992.

[303] T. Strecker and L. Hennig. Automatic layouting of personalized newspaper pages. In B. Fleischmann, K.-H. Borgwardt, R. Klein, and A. Tuma, editors, *Operations Research Proceedings 2008*, pages 469–474, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[304] J. Terno, R. Lindemann, and G. Scheithauer. Zuschnittprobleme and ihre praktische lösung. Technical report, Verlag Harry Deutsch, Thun und FrankfurtMain, 1987.

[305] Ying-Tai Loong Tian-Soon Lee. A review of scheduling problem and resolution methods in flexible flow shop. *International Journal of Industrial Engineering Computations*, 10(1):67–88, 2019.

[306] P. Toth and D. Vigo. *Vehicle Routing: Problems, Methods, and Applications*. SIAM, 2nd edition, 2014.

[307] M.A. Trick. A dynamic programming approach for consistency and propagation for knapsack constraints. *Annals of Operations Research*, 118(1-4):73–84, 2003.

[308] R.S. Trindade, O.C.B. de Araújo, and M. Fampa. Arc-flow approach for parallel batch processing machine scheduling with non-identical job sizes. In M. Baïou, B. Gendron, O. Günlük, and A.R. Mahjoub, editors, *Combinatorial Optimization*, pages 179–190. Springer International Publishing, 2020.

[309] H. Tsubone, M. Suzuki, T. Uetake, and M. Ohba. A comparison between basic cyclic scheduling and variable cyclic scheduling in a two-stage hybrid flow shop. *Decision Sciences*, 31(1):197–222, 2000.

[310] E. Uchoa. Cuts over extended formulations by flow discretization. In A.R. Mahjoub, editor, *Progress in Combinatorial Optimization*, chapter 8, pages 255–282. Wiley, 2012.

[311] E. Uchoa, D. Pecin, A. Pessoa, M. Poggi, T. Vidal, and A. Subramanian. New benchmark instances for the capacitated vehicle routing problem. *European Journal of Operational Research*, 257(3):845–858, 2017.

[312] J. M. Valério de Carvalho. LP models for bin packing and cutting stock problems. *European Journal of Operational Research*, 141(2):253–273, 2002.

[313] J. M. Valério de Carvalho. Using extra dual cuts to accelerate column generation. *INFORMS Journal on Computing*, 17(2):175–182, 2005.

[314] J.M. Valério de Carvalho. Exact solution of bin-packing problems using column generation and branch-and-bound. *Annals of Operations Research*, 86:629–659, 1999.

[315] J.M. Valério de Carvalho. LP models for bin packing and cutting stock problems. *European Journal of Operational Research*, 141(2):253–273, 2002.

[316] J.M. Valério de Carvalho. Using extra dual cuts to accelerate column generation. *INFORMS Journal on Computing*, 17(2):175–182, 2005.

[317] J.M. van den Akker, J.A. Hoogeveen, and S.L. van de Velde. Parallel machine scheduling by column generation. *Operations Research*, 47(6):862–872, 1999.

[318] J.M. van den Akker, C.A.J. Hurkens, and M.W.P. Savelsbergh. Time-indexed formulations for machine scheduling problems: Column generation. *INFORMS Journal on Computing*, 12(2):111–124, 2000.

[319] W.-J. van Hoeve. Graph coloring lower bounds from decision diagrams. In D. Bienstock and G. Zambelli, editors, *Integer Programming and Combinatorial Optimization*, pages 405–418. Springer International Publishing, 2020.

[320] P.H. Vance. Branch-and-price algorithms for the one-dimensional cutting stock problem. *Computational Optimization and Applications*, 9(3):211–228, 1998.

[321] 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.

[322] F. Vanderbeck. A nested decomposition approach to a three-stage two-dimensional cutting-stock problem. *Management Science*, 47(6):864–879, 2001.

[323] F. Vanderbeck. Branching in branch-and-price: a generic scheme. *Mathematical Programming*, 130:249–294, 2011.

[324] F. Vanderbeck and M.W.P. Savelsbergh. A generic view of dantzig–wolfe decomposition in mixed integer programming. *Operations Research Letters*, 34(3):296–306, 2006.

[325] A. S. Velasco and E. Uchoa. Improved state space relaxation for constrained two-dimensional guillotine cutting problems. *European Journal of Operational Research*, 272(1):106–120, 2019.

[326] K. V. Viswanathan and A. Bagchi. Best-first search methods for constrained two-dimensional cutting stock problems. *Operations Research*, 41(4):768–776, 1993.

[327] M.-E. Voge and F. Clautiaux. Theoretical investigation of aggregation in pseudo-polynomial network-flow models. In A.R. Mahjoub, V. Markakis, I. Milis, and V.T. Paschos, editors, *Combinatorial Optimization*, pages 213–224. Springer Berlin Heidelberg, 2012.

[328] D.M. Vu, M. Hewitt, N. Boland, and M. Savelsbergh. Dynamic discretization discovery for solving the time-dependent traveling salesman problem with time windows. *Transportation Science*, 54(3):703–720, 2020.

[329] P.Y. Wang. Two algorithms for constrained two-dimensional cutting stock problems. *Operations Research*, 31(3):573–586, 1983.

[330] Shijin Wang, Xiaodong Wang, Feng Chu, and Jianbo Yu. An energy-efficient two-stage hybrid flow shop scheduling problem in a glass production. *International Journal of Production Research*, 58(8):2283–2314, 2020.

[331] X. Wang, G. Wu, L. Xing, and W. Pedrycz. Agile earth observation satellite scheduling over 20 years: Formulations, methods, and future directions. *IEEE Systems Journal*, 2020, forthcoming.

[332] G. Wäscher, H. Hauβner, and H. Schumann. An improved typology of cutting and packing problems. *European Journal of Operational Research*, 183(3):1109–1130, 2007.

[333] L. Wei, Z. Luo, R. Baldacci, and A. Lim. A new branch-and-price-and-cut algorithm for one-dimensional bin-packing problems. *INFORMS Journal on Computing*, 32(2):428–443, 2020.

[334] L. Wei, W.-C. Oon, W. Zhu, and A. Lim. A skyline heuristic for the 2D rectangular packing and strip packing problems. *European Journal of Operational Research*, 215(2):337–346, 2011.

[335] L. Wei, H. Qin, B. Cheang, and X. Xu. An efficient intelligent search algorithm for the two-dimensional rectangular strip packing problem. *International Transactions in Operational Research*, 23(1-2):65–92, 2016.

[336] P. Wentges. Weighted dantzig–wolfe decomposition for linear mixed-integer programming. *International Transactions in Operational Research*, 4(2):151–162, 1997.

[337] L.A. Wolsey. Valid inequalities, covering problems and discrete dynamic programs. *Annals of Discrete Mathematics*, 1:527–538, 1977.

[338] Z. Xu and C.-Y. Lee. New lower bound and exact method for the continuous berth allocation problem. *Operations Research*, 66(3):778–798, 2018.

[339] H. H. Yanasse and D. M. Katsurayama. Checkerboard pattern: proposals for its generation. *International Transactions in Operational Research*, 12(1):21–45, 2005.

[340] H. H. Yanasse and D. M. Katsurayama. An enumeration scheme to generate constrained exact checkerboard patterns. *Computers & Operations Research*, 35(6):2114–2128, 2008.

[341] G. Yu, Y. Mao, and J. Xiao. A new lower bound for online strip packing. *European Journal of Operational Research*, 250(3):754–759, 2016.

[342] G. Yu, Y. Mao, and J. Xiao. A new upper bound for the online square packing problem in a strip. *Journal of Combinatorial Optimization*, 33(4):1411–1420, 2017.

[343] G. Yu, Y. Mao, and J. Xiao. New upper bounds for online strip packing. *Discrete Optimization*, 23:20–32, 2017.