A Scheduling-based Constraint Programming Approach to Solving a Complex Two-Dimensional Two-Stage Cutting Stock Problem

by

Yiqing L. Luo

A thesis submitted in conformity with the requirements
for the degree of Master of Applied Science

Graduate Department of Mechanical and Industrial Engineering
University of Toronto

A Scheduling-based Constraint Programming Approach to Solving a Complex Two-Dimensional Two-Stage Cutting Stock Problem

Yiqing L. Luo
Master of Applied Science

Graduate Department of Mechanical and Industrial Engineering
University of Toronto
2022

Abstract

We investigate the novel Two-Dimensional Two-stage Cutting Stock Problem with Flexible Length, Flexible Demand, Order-to-Order Marriageability, and Scheduling Costs (2SCSP-FFMS): orders for rectangular items must be cut from treated rectangular stocks using guillotine cuts with the objective to minimize waste, inventory cost, and tardiness cost. Different from problems in the literature, the 2SCSP-FFMS allows the item length and total order demands to vary within customer-specified intervals. We first investigate a variation of the problem that ignores marriageability (pairwise conflicts between orders) and scheduling costs, proposing constraint programming models, mixed-integer programming models, and heuristics. Then, we study a second variation that adds the marriageability requirement before examining the full 2SCSP-FFMS problem. Accordingly, we extend the approaches that performed best in the first variation to the second one and the full 2SCSP-FFMS. For each of these problems, we perform empirical analysis on both generated and real-life industrial instances. Notably, our scheduling-based constraint programming model has orders-of-magnitude smaller memory requirements over other exact methods and can be competitive with a customized multi-phase heuristic.

Dedicated to my family.

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my supervisor, Chris Beck, for the unwavering support and thoughtful mentorship over the past two years and through the COVID-19 disruptions. You have been a beacon both academically and personally, and I genuinely appreciate the effort and faith you have placed in me.

I would like to extend my gratitude to my committee member, Andre Cire, for the valuable comments during my defense.

Special thanks to Kinaxis, our industrial collaborator, for the insights on the problem and for providing the data used in this thesis.

Thanks should also go to the other members at TIDEL. To Giovanni, *merci beaucoup* for being so caring, cheerful, and magnanimous. I am grateful for our thoughtful conversations, intense chess games, and collaborative partnership running T-SOC. To Jason, thank you for all the laughter and stories. It was truly my pleasure leading the tutorials and working on volunteer projects alongside someone as reliable, humble, and genuine as you are. To Anton, Arik, Arnoosh, Evghenii, Jasper, Kyle, Lily, Litong, Minori, Ryo, Sonia, Tan, and Victor, thanks for all the memories - I know I will always treasure them. I wish you all nothing but the best in your future endeavours.

Finally, many, many thanks goes to my family for the love, support, and sacrifices.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Constraint programming is a paradigm that solves combinatorial problems by a combination of search and logical reasoning. Drawing upon techniques from artificial intelligence, computer science, and operations research, constraint programming represents problems by declaring a set of constraints over a set of variables and is often seen as an alternative to mathematical programming [14]. A distinguishing feature of constraint programming compared to other model-based paradigms is the expressivity of its constraints: they are not limited to mathematical expressions. In particular, constraint programming offers global constraints that describe recurring problem substructures and make inferences based on the entire substructure instead of just individual mathematical expressions [14]. In addition, constraint programming allows users the flexibility to define constraints with customized inference algorithms [129].

Solving constraint programs traditionally involves building tree-search algorithms, the process of which can be complex and time-consuming [14, 33]. To improve the accessibility of constraint programming, since the 1980s, researchers have built general-purpose constraint programming solvers that search systematically [56, 57, 138]. These solvers are designed to be off-the-shelf toolboxes with built-in constraints and variable types that can be used to construct a structured model representing the problem. Then, the solvers execute a series of internal routines to search for solutions [159]. As a result, practitioners can leverage the expressivity of constraint programming without needing to develop the internal workings of the solver.

Although general-purpose solvers allow diverse problems to be modelled using the same set of tools, each problem can still be modelled using possibly different formulations of variables and constraints [229]. In constraint programming, different formulations can result in different search trees and different levels of inference at each search node, leading to varying performance [33, 34, 229]. The promise of performance gains makes examining these formulations an imperative to obtaining quality results.

Constraint programming has been applied to many different problem domains, the most successful being scheduling [157]. Scheduling problems involve the allocation of resources over time and are computationally difficult to solve, with many variations being NP-hard [87, 107, 196]. These problems have shown a natural affinity to be framed as constraint programs, as their inherent substructures, such as precedence constraints and optional tasks, naturally invite temporal reasoning, a particular type of inference integrated into most general-purpose solvers [50, 160, 199, 206]. Some solvers, such as IBM's CP Optimizer [128], also combine linear relaxation with the temporal reasoning for enhanced performance on scheduling problems [158]. A testament to its success, constraint programs are currently being relied on to schedule many major business operations, including airport flights in Hong Kong [117] and transshipment hubs in the port of Singapore [117].

Constraint programming has yet to reach state-of-the-art in a number of other problems including two-dimensional cutting stock problems. First studied in 1939 [143], cutting stock problems, also known as bin packing problems, arise from natural applications such as manufacturing [136, 163, 232] and transportation [136, 205]. In two-dimensional cutting stock problems, the goal is to cut large rectangular stocks into smaller rectangular pieces to fulfill orders, and the problem becomes even more complex if there are restrictions on the form of cuts (e.g., guillotine cuts, see Section 3.3.2). Many solution techniques have tackled this problem, with the most popular exact method being mixed-integer linear programming [137]. Around the turn of the century, academic efforts and technological breakthroughs spurred advances in constraint programming, resulting in the development of global constraints PACK for one-dimensional packing [221] and CUMULATIVE for the resource capacity [20, 226]. However, these tools have mostly been applied to other packing problems, such as the optimal rectangle packing problem [154, 155, 194, 225, 226], so the performance of constraint programming on two-dimensional cutting stock problems, especially those with complex cutting requirements, remains largely unknown.

Researchers have identified the link between packing and scheduling since the late 1990s [65, 110]. Notably, these problems share a similar problem structure: both require resource capacities to be satisfied. For example, a single machine job-shop-scheduling problem without precedence can be reduced to a one-dimensional packing problem if jobs and operations are interpreted as pieces that need to be cut and machine capacity over time as stocks. This relationship invites further investigation into constraint programming's ability to solve two-dimensional packing problems.

This thesis leverages constraint programming tools designed for scheduling problems to solve different variations of a novel packing problem from the aluminum-metal industry: the Two-Dimensional Two-Stage Cutting Stock Problem with Flexible Length, Flexible Demand, Marriageability, and Scheduling Costs (2SCSP-FFMS). We compare the results with mixed-integer programs and heuristics. At the time of writing, constraint programming has neither been used to study two-dimensional packing problems with flexible dimensions nor packing problems with guillotine cuts.

Thesis Statement This thesis develops optimization approaches to solve a novel packing problem, the 2SCSP-FFMS, that arises from the industry. The central thesis is as follows:

> Tools developed for scheduling in general-purpose constraint programming solvers can achieve state-of-the-art performance among model-based approaches and competitive performance with customized heuristics in solving large-scale industrial packing problems.

The thesis outline and the primary contributions are discussed in the following sections.

1.1 Thesis Outline

Chapter 2 defines and formalizes our problem, including the notation and terminologies used throughout the thesis. The chapter closes with a detailed description of the experiment setup and the data used.

Chapter 3 presents the background necessary in understanding the thesis. First, the chapter formalizes the main optimization paradigms, mixed-integer programming and constraint programming, and presents details on how each paradigm is typically used to model and solve problems. The chapter then reviews related literature, covering important mixed-integer linear programming and constraint programming modelling perspectives in one-dimensional (1D) and two-dimensional (2D) packing. As the 2SCSP-FFMS is a 2D problem, we provide a classification of the existing literature on 2D packing, within which our problem is then contextualized. Lastly, the chapter broadly reviews literature on batch scheduling and vehicle routing and highlights the connection between packing and these prominent scheduling problems.

The 2SCSP-FFMS can be divided into three components: packing items into stocks, treating stocks to meet order properties, and scheduling. The three subsequent chapters study different variations of the 2SCSP-FFMS, incrementally adding the components to reflect different industrial use cases.

Chapter 4 studies the packing-only scenario. Our empirical results show that the scheduling-based constraint programming model has an order of magnitude advantage in memory usage, and accordingly, is the only model-based approach to scale to larger instances. Mixed-integer models found high-quality solutions for the small instances, but struggled to scale. Lastly, we develop heuristic solutions and show that by decomposing the problem into smaller subproblems, heuristics can find solutions competitive with the scheduling-based model for the industrial instances. The work in this chapter extends our published paper in the *Proceedings of the Sixteenth International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research* [181].

Chapter 5 examines a packing scenario considering stock treatments and the resultant order properties. This chapter builds upon Chapter 4, formalizing the new requirement and augmenting the best-performing methods in Chapter 4. The scheduling-based model remains the only model to scale to industrial instances. The best heuristic is competitive with, if not slightly better than, the scheduling-based model, but requires more time to find feasible solutions.

Chapter 6 studies the full 2SCSP-FFMS, adding a scheduling-related cost factor to the packing use case with stock treatments. We again build upon the formulations in Chapter 5 and realize the performance shortcomings of the scheduling-based model: it struggled to find high-quality solutions with the more complex objective function.

Chapter 7 concludes this thesis and discusses directions for future work.

1.2 Contribution Summary

The main contributions of this thesis are as follows:

- We adapt and develop a scheduling-based single resource constraint programming model that connects scheduling to packing. We demonstrate its computational efficiency for 2SCSP-FFMS and the related use cases and its resulting affinity for larger instances.

- We recognize a connection between guillotine cutting patterns and batch scheduling and adapt constraint programming techniques proposed in batch scheduling to our packing problem.

- We formalize the novel 2SCSP-FFMS. This problem is motivated by an industrial use case in aluminum trimming and is at the intersection of many hard problems in the literature.

- We propose the first mixed-integer linear programming models, alternative constraint programming models, a first-fit-based heuristic, and a sequential heuristic framework for the different variations of 2SCSP-FFMS.

Chapter 2

Problem Overview

THE Two-Dimensional Two-Stage Cutting Stock Problem with Flexible Item, Flexible Demand, Marriageability, and Scheduling Cost, 2SCSP-FFMS, abstracts the planning process of manufacturers in the rolled-metals industry. Over a given time horizon, we want to satisfy client orders by cutting items using a guillotine machine from a limited quantity of stocks while minimizing aggregate cost. A cut item is subsequently rolled into a cylindrical coil used as feedstock for downstream processing, resulting in novel requirements that impose modelling and computational challenges.

2.1 Problem Definition

The 2SCSP-FFMS is a novel generalization of the Two-stage Two-Dimensional Cutting Stock Problem with Guillotine Constraints (2SCSP). Given a set of *orders* for rectangular *items* and a set of larger *stock rectangles*, the classic Two-Dimensional Cutting Stock Problem (2DCSP) fulfills orders by cutting items from stocks. A more constrained variant, the 2SCSP only allows stocks to be processed using *guillotine cuts*, a cut that runs from one edge of the object to another. All cuts must also be executed in two stages, each consisting of a set of parallel guillotine cuts performed on a rectangle obtained from the previous stage (Figure 2.1). Without loss of generality, we let the direction of the first stage cuts be widthwise and that of the second stage ones lengthwise. The rectangles produced in the first stage are referred to as *levels*, following the literature [86], and those produced in the second stage as *partitions*. On top of 2SCSP, the 2SCSP-FFMS has the following characteristics arising from our application.

Flexible Length: The length of an item is flexible within some integer interval dependent on the order. If a level contains items from different orders, its length must lie in the intersection of the item-length intervals. In our application, the maximum length requirement ensures a maximum coil diameter to enable mounting it on a downstream machine. The minimum length requirement comes from the desire to

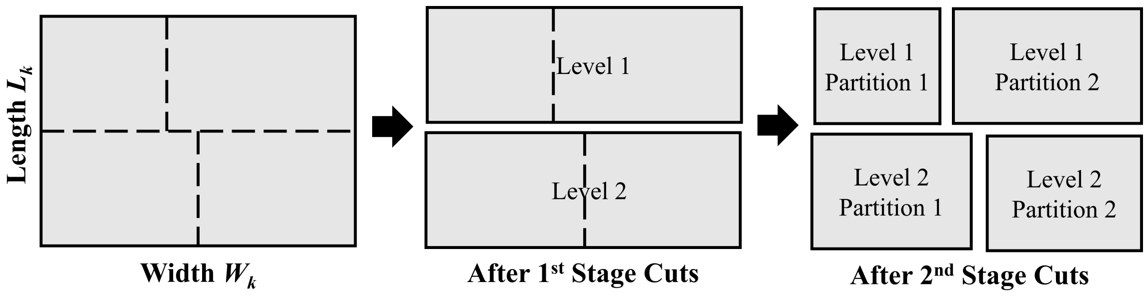


Figure 2.1: A visualization of the guillotine cutting process.

limit the number of coils.

Flexible Demand: Consistent with real-world manufacturing practices, each order can tolerate a percentage deviation from the total area demanded. For example, an order may request items totalling $10000 \pm 15\%$ units of area.

Maximum Partition Count per Level: To reflect the limitations of an industrial cutter, the maximum number of partitions on each level is fixed.

Limited Stocks with Variable Sizes: Stock rectangles of various widths and lengths are available in limited quantities.

Order-to-order Marriageability: Each stock needs to be treated so that the cut orders can have desired *properties*, such as temper, gauge ranges, and coating. Consequently, items belonging to orders with different properties cannot be cut from the same stock. Each stock can be processed to have exactly one set of properties (e.g., one coating).

Cost Minimization: The goal is to minimize the monetary cost that is comprised of two components: packing-related waste and scheduling-related expenses. The cost of waste is defined as the sum of the weighted difference between the area of the stocks used and the area of the orders fulfilled. The cost related to scheduling deals with the price of storing the cut items until their orders are due (inventory cost) and the penalty for delivering items later than their order due dates (tardiness cost). Later in the section, we introduce an approximation to simplify the scheduling-related expenses.

Figure 2.2 describes the overarching relationship between items, orders, and stocks. Formally, we are given a set of stock rectangles K, whose types are characterized by set H. Each rectangle $k \in K$ has width W_k and length L_k. Stock rectangles with identical dimensions belong to the same type, K_h, $K = \bigcup_{h \in H} K_h$. We are also given a set of orders, N, where each order $i \in N$ has a required area interval of $[q_i^{min}, q_i^{max}]$. Each item belonging to order i is required to have a fixed width w_i and a length within the interval

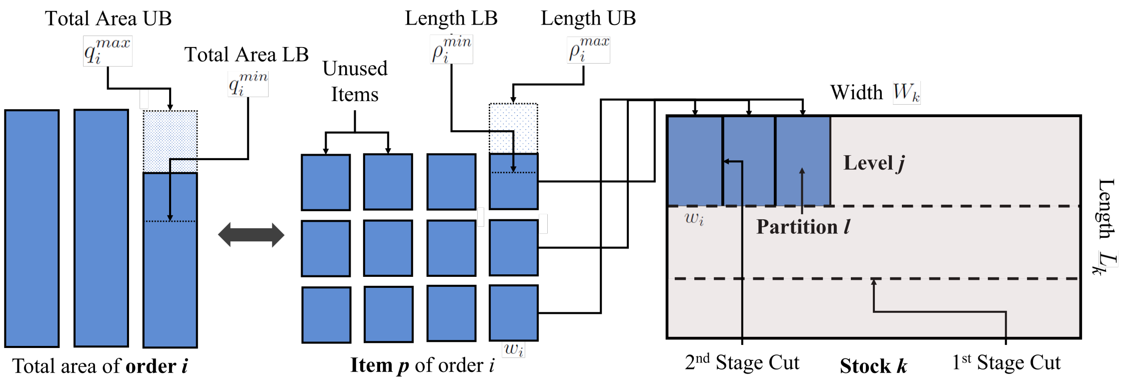


Figure 2.2: A visualization of 2SCSP-FFMS. Each order is represented by its total quantity (left) and the partitions assigned to it (middle), as illustrated by the double-arrow. Dashed lines and the dotted fills indicate flexibility in the associated parameter. Orange and blue lines represent the first and second stage cuts, respectively. LB and UB abbreviate lower and upper bound, respectively.

$[\rho_i^{min}, \rho_i^{max}]$. Due to the flexible length, the total number of items belonging to order i must be within an integer interval $[n_i^{min}, n_i^{max}] = [\lceil \frac{q_i^{min}}{\rho_i^{max}} \rceil, \lfloor \frac{q_i^{max}}{\rho_i^{min}} \rfloor]$. For order i, we denote its set of necessary items as $A_i = \{1, \ldots, n_i^{min}\}$, its set of possible, but not necessary items as $B_i = \{n_i^{min} + 1, \ldots, n_i^{max}\}$, and all possible items as $C_i = A_i \bigcup B_i$. Lastly, we let α and β be the coefficients associated with the area of stocks used and the area of orders fulfilled, respectively, and seek to minimize this weighted difference.

A stock k can be assigned to at most $\bar{j}_k = \lfloor \frac{L_k}{\min_{i \in N} \rho_i^{min}} \rfloor$ levels, and we denote the set of possible numbers of levels of stock k of type h as $J_k = J_h = \{0, \ldots, \bar{j}_k\}$. There must also be no more than η partitions on each level. We let $P = \{1, \ldots, \eta\}$ be the set of partitions on a given level, and $P_i = \{l \in P \mid l \leq n_i^{max}\}$ be the set of partitions on a given level from order i. A partition of a stock that is assigned to an order becomes an item.

In addition, we let G be the set of combinations of order properties and N_γ be the set of orders with properties $\gamma \in G$. Equivalently, we can summarize the relationship between the items of these orders in a conflict matrix: given an item from order i and another from order i', the two cannot be assigned to the same stock if $M_{ii'} = 0$. Note that $M_{ii'} = 1$ if and only if $\exists \gamma \in G$ such that $i, i' \in N_\gamma$. A stock can only be treated once, so orders of different properties cannot be cut from the same stock. If there is an infinite number of stocks, then the problem can be separated into independent subproblems for each combination of order properties. However, with a finite number of stocks, the orders for one property combination compete for stocks with orders of other property combinations, tying these orders together.

Finally, we consider a set of scheduling requirements: stocks cannot be cut until they are available, and cutting orders before or after their due dates incurs either inventory or tardiness cost (Figure 2.3). Inventory cost is incurred if an item of an order is cut before its due date, and tardiness cost if it is cut after its due date. A stock can only be cut once.

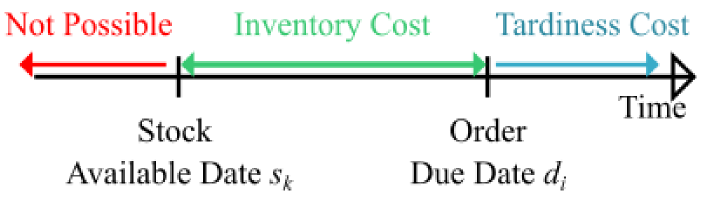


Figure 2.3: A visualization of the scheduling cost of assigning items from order i to stock k.

$\|N\|$	$\|K\|$	W_k	a_k	w_i	ρ_i^{min}	ρ_i^{max}	q_i^{min}	q_i^{max}	d_i	$\|G\|$
19	42	48	35.4	22.2	112.9	180.2	1.3e5	1.7e5	18.9	8
21	172	45.2	18.3	16.3	56.6	94.6	9.7e4	1.3e5	4.0	7
47	149	43.3	197.9	10.4	68.2	134.1	1.6e5	2.1e5	186.8	18
149	636	44.6	190.1	13.3	74.4	134.4	2.5e5	3.4e5	217.2	32

Table 2.1: Mean of the parameter combinations from the four industrial instances.

Approximating Scheduling Costs The inventory and tardiness costs are proportional to the product of an item's area and the time difference between when the item is cut and the due date of its order. Formally, we let the available time of stock k and the due date of order i be a_k and d_i. Consider an item of order i cut from stock k at time t. The inventory cost is $c_{inv}w_il_i \times \max(0, d_k - t)$ and the tardiness cost $c_{tard}w_il_i \times \max(0, t - d_k)$, where l_i is the length of the item, c_{inv} the inventory cost per unit time per unit area, and c_{tard} the tardiness cost per unit time per unit area. Accordingly, we augment the definition of stock types so that, in addition to identical dimensions, they must be available at the same time.

Typically, c_{inv} is orders of magnitude smaller than c_{tard}, so the optimal time to cut any stock is always slightly after the earliest due date of the orders assigned to a stock. Formulating a concise objective expression can be complicated, as t, l_i given a level assignment, and the matching of an item to stock k are all independent decisions. Instead, we resort to an approximation: a stock is cut as soon as it is available. Mathematically, the approximated inventory cost is $c_{inv}w_il_i \times \max(0, d_i - a_k)$ and the approximated tardiness cost $c_{tard}w_il_i \times \max(0, a_k - d_i)$.

2.2 Data Description

We conduct our analysis on a combination of 50 generated problem instances and 4 real-life instances provided by our industry partner (Table 2.1).

For the generated instances, we draw from distributions provided by our industrial collaborator (Table 2.2), generating 10 instances for each parameter combination in the set $\{(|N|, |K|)\} \in \{(4, 8), (8, 16), (16, 32), (32, 64), (64, 128)\}$. For 5 out of these 10 instances, we halve the total area tolerances to provide variability. In the rare case that, for some order i, $\rho_i^{max} \leq \rho_i^{min}$ is generated, we swap the two values. We also force the first two

Parameter	Distribution
	$\forall$ order $i \in N$
q_i	Exponential(λ=5.608e-0.5)
q_i^{max}	Constant, 0.85 q_i
q_i^{min}	Constant, 1.15 q_i
w_i	Integer Uniform(a=1, b=20)
ρ_i^{max}	Integer Uniform(a=70, b=115)
ρ_i^{min}	Integer Uniform(a=85, b=130)
d_i	Integer Uniform(a=0, b=100)
$\gamma \in G$	Integer Uniform(a=0, b=$\|N\|/2$)
	$\forall$ stock $k \in K$
W_k	Integer Uniform(a=36, b=50) with 50% chance of duplicating previous stock
L_k	Integer Uniform(a=350, b=450) with 50% chance of duplicating previous stock
a_k	Integer Uniform(a=0, b=100)

Table 2.2: Data distributions for parameters in the generated instances.

orders to be incompatible so that some item-to-item conflict is always present.

For all experiments, we set α and β, the objective weights, to 0.3 and 0.7, respectively. We also set c_{inv} and c_{tard}, the scheduling weights, to 0.0006 and 0.72 to reflect the industrial use case.

2.3 Experiment Setup

In this thesis, all experiments were implemented in Python 3.8, and computations were performed on individual nodes of the SciNet Niagara cluster [179, 211]. To solve the optimization models, we used CPLEX and CP Optimizer from the CPLEX Optimization Studio version 20.1.0 accessed via the DOcplex Modelling API with a single thread and default search and inference settings. All experiments were given 16 GB of RAM and runs that exceed this size were aborted. A one-hour time limit was used.

2.4 Summary

In this chapter, we introduced and formally defined the 2SCSP-FFMS. We also described our data generation procedure, summarized key statistics about the industrial instances, and detailed our experimental setup. In the next chapter, we review relevant background and contextualize 2SCSP-FFMS.

Chapter 3

Background

In this chapter, we present relevant background to the thesis. We first describe mixed-integer linear programming (MILP) and constraint programming (CP), two optimization techniques central to our work. Then, we review literature related to our problem, spotlighting important MILP and CP formulations in 1D and 2D packing and describing alternative approaches. Finally, we compare packing with scheduling and review two related types of scheduling problems.

3.1 Mixed-Integer Linear Programming

Mixed-integer linear programming (MILP) is an optimization paradigm that expresses problem requirements using linear mathematical expressions and continuous and integer variables [62, 143, 219]. Widely employed in various sectors, including food and agriculture [42, 228], engineering [35, 184, 186], transportation [5, 170, 220], manufacturing and energy [102, 157], MILP is one of the most popular techniques to solve optimization problems. In this section, we describe MILP models and how they are solved.

3.1.1 Modelling

A MILP is an optimization model that takes up the following form:

$$\min \ \mathbf{c^T x} + \mathbf{d^T y} \tag{3.1a}$$

$$\mathbf{Ax} + \mathbf{By} \geq \mathbf{h} \tag{3.1b}$$

$$\mathbf{x} \in \mathbb{R}^n, \mathbf{y} \in \mathbb{Z}^m \tag{3.1c}$$

where $\mathbf{x} \in \mathbb{R}^n$ is a column vector of continuous variables, $\mathbf{y} \in \mathbb{Z}^m$ is a column vector of integer variables, $\mathbf{c} \in \mathbb{R}^n$ and $\mathbf{h} \in \mathbb{R}^m$ are column vectors denoting cost, and $\mathbf{A} \in \mathbb{R}^{n \times l}$ and $\mathbf{B} \in \mathbb{R}^{m \times l}$ are constraint matrices. If $m = 0$, then the MILP is a linear program. If $n = 0$, then the MILP is referred to as a pure integer program. Constraints in MILP are usually linear (i.e. of the first order) inequalities. Contingent on the solver, MILP constraints can also express special ordered sets (SOS) [17], which restricts the number of nonzero solution values among a specified set of variables. For instance, $SOS1$, an SOS of type 1, restricts a set of variables to have at most one non-zero value.

3.1.2 Solving

Optimizing MILPs typically involves running a branch-bound-and-cut algorithm [97, 94, 161]. At each iteration, the algorithm infers a bound on the objective function via linear relaxation, partitions the model's solution space by branching on variables, and adds valid inequalities to tighten the relaxation without removing integer solutions.

The linear relaxation of a MILP ignores integrality constraints on variables, yielding a linear program that can be solved in polynomial time [61, 146] and a solution that is a bound on the MILP's objective. Having a strong linear relaxation is important, as it allows the search tree to be pruned early, thus increasing the chance of avoiding an exponential search. Typical approaches to solving a linear relaxation include the simplex method [61], the dual simplex method [242], and the interior-point method [146, 212].

Branching partitions the original problem into disjoint subproblems and involves three main decisions: variable selection, branching strategy, and node selection. Variable selection decides which variable to branch on, and common approaches include pseudo-costs [26, 88], strong branching [9, 128], and reliability branching [2]. Typically, a MILP's branching strategy is binary, creating two independent subproblems at each node [198]; using SOSes also allows solvers to employ a wide branching strategy, where multiple branches originate from the parent node [16, 17, 198]. Finally, node selection decides which subproblem is processed after branching, and typical strategies include best first search [63], depth first search [96, 233], and, more recently, cyclic best-first search [49, 144, 197]. For further details on the branch and bound algorithm, we refer readers to a survey by Morrison et al. [198].

A valid inequality (VI), also known as a cut, is a constraint added to a MILP to tighten its linear relaxation without removing integer solutions [202]. There are three main classes of VIs: general purpose, generic structure, and problem-specific. General purpose VIs are independent of any problem structures, and examples include Chvátal-Gomory cuts [52], Gomory cuts [97], mixed-integer rounding cuts [104, 201], and disjunctive cuts [10]. Consequently, these VIs are commonly implemented in MILP solving software. Generic structure VIs are derived from basic problem substructures, such as knapsack sets [11, 12, 59, 145, 244] and mixing sets [192, 241]. Finally, problem-specific VIs are specific to the problem, with successful examples spanning domains including matching [111, 237] and travelling salesman problems [1, 23, 193, 204, 207]. Adding VIs to a MILP strengthens solvers' inference at every search node, and has been a key driver to their performance improvement over the past two decades [32, 103].

3.1.3 Decomposition Techniques

Decomposition techniques divide a problem into smaller and more tractable parts that can be recombined to recover the optimal solution to the original problem. There are two main types of decompositions in MILP: column generation and Benders Decomposition.

3.1.3.1 Column Generation

Column generation decomposes a linear programming problem based on its variables [81]. It first formulates a restricted master problem, which only contains a subset of variables from the original model. Then, the subproblem tries to find a variable that has a negative reduced cost to enter the restricted master problem to improve its objective value. The algorithm repeats until no such variables can be found. Since a column generation algorithm dynamically generates variables during search, it is often most suitable when there are significantly more variables than constraints.

A major drawback of column generation is that it only solves linear programs. If optimality is not required, heuristics, such as rounding [70], can be used to recover integer solutions, although feasibility is not always guaranteed [70, 141]. To exactly solve a problem and prove optimality, Barnhart et al. [15] proposed the branch-and-price algorithm, which generates columns at every node in a branch-and-bound tree. However, designing an efficient and effective branch-and-price algorithm remains complex and intricate [76, 240].

3.1.3.2 Benders Decomposition

Benders Decomposition [25] partitions a MILP formulation into a master problem and one or more subproblems containing subsets of its variables and constraints. Each subproblem generates a set of cuts that are added to the master problem, strengthening its formulation so that eventually the MILP's optimal solution can be recovered. Since a cut added to the master problem increases its constraint matrix by one row, Benders Decomposition is also referred to as row generation [120].

Formally, consider the following MILP formulation where $\mathbf{x}$ is a vector of complicating variables belonging to a possibly *integer solution space* $\mathcal{X}$:

$$\min\ \mathbf{c^T x} + \mathbf{d^T y} \tag{3.2a}$$

$$\mathbf{Ax} + \mathbf{By} \geq \mathbf{h} \tag{3.2b}$$

$$\mathbf{x} \in \mathcal{X} \tag{3.2c}$$

$$\mathbf{x} \in \mathbb{R}_+^n, \mathbf{y} \in \mathbb{R}_+^m \tag{3.2d}$$

Benders Decomposition can be derived naturally from the Benders Reformulation [25, 32, 202], which rearranges the MILP formulation as follows:

$$\min\ \mathbf{c^T x} + \eta \tag{3.3a}$$
$$\mathbf{x} \in \mathcal{X} \tag{3.3b}$$
$$\mathbf{x} \in \mathbb{R}^n_+ \tag{3.3c}$$
$$\eta \in \mathbb{R} \tag{3.3d}$$
$$\eta \geq \boldsymbol{\pi}^T(\mathbf{h} - \mathbf{Ax}) \quad \forall \boldsymbol{\pi} \in V(\Pi) \tag{3.3e}$$
$$\boldsymbol{r}^T(\mathbf{h} - \mathbf{Ax}) \leq 0 \quad \forall \mathbf{r} \in R(\Pi) \tag{3.3f}$$

where Π is the dual feasible region of the subproblem $Q(\mathbf{x}) = \{\min \mathbf{d^T y} \mid \mathbf{Ax} + \mathbf{By} \geq \mathbf{h}, \mathbf{y} \in \mathbb{R}^m_+\}$, and $R(\Pi)$ and $V(\Pi)$ are the set of extreme rays and the set of extreme points of Π, respectively. In essence, Benders Reformulation characterizes a subproblem as a combination of extreme rays and extreme points of its dual. If the dual of the subproblem has an optimal solution, then the solution must be an extreme point in the set $V(\Pi)$, so the optimality cuts expressed in constraint (3.3e) restrict the tightest lower bound for the subproblem for any $\mathbf{x}$ feasible to the subproblem. Alternatively, if the optimal solution of the dual of the subproblem is in the direction of an extreme ray, then the subproblem itself is infeasible, so the feasibility cuts expressed in constraint (3.3f) remove any solutions infeasible to the subproblem. Unfortunately, the dual feasible region of the subproblem can have exponentially many extreme points and extreme rays, leading to intractability.

Benders Decomposition circumvents this scalability issue by dynamically generating the optimality and feasibility cuts in the subproblem using either a cutting plane algorithm or a branch-bound-and-cut algorithm. At the root node, the master problem $\{\min(3.3a) \mid (3.3b) - (3.3d)\}$ is solved. Then, the subproblem $Q(\hat{\boldsymbol{x}})$ is evaluated using the optimal solution $\hat{\boldsymbol{x}}$ from the master problem. If the subproblem is infeasible, we add a feasibility cut $\hat{\boldsymbol{r}}^T(\mathbf{h} - \mathbf{A\hat{x}}) \leq 0$, where $\hat{r}$ is the extreme ray associated with the dual of $Q(\hat{\boldsymbol{x}})$. If the subproblem is optimal with dual variables $\hat{\boldsymbol{\pi}}$, then the optimality cut $\eta \geq \hat{\boldsymbol{\pi}}^T(\mathbf{h} - \mathbf{A\hat{x}})$ is added. This process is repeated until the optimum of the master problem does not change, indicating global optimality, or the master problem becomes infeasible, indicating that the original problem is infeasible.

Geoffrion [91] developed the Generalized Benders Decomposition for mixed integer nonlinear programs. Hooker and Ottosson [121] proposed the Logic-Based Benders Decomposition, which substitutes the mathematical cuts with inferential ones. Consequently, more paradigms can be used to represent master problems and subproblems, thereby loosening the decomposition criteria.

3.1.4 Software

An assortment of commercial and open source software is available for both modelling and solving MILPs. Commercial solvers include IBM's CPLEX Optimization Suite [128], Gurobi [106], and FICO Xpress Optimizer [203]. Open source solvers include COIN-OR Branch-and-Cut Solver [82] and SCIP [28, 29].

3.2 Constraint Programming

Developed within the artificial intelligence community, constraint programming (CP) solves combinatorial problems by reasoning about problem substructures expressed as constraints [14, 18, 234]. Compared to MILPs, the variables and constraints in CP are richer, allowing more types of problems to be modelled and solved. In this section, we present background information on CP modelling and solving.

3.2.1 Modelling

There are two types of models in CP: constraint satisfaction problem (CSP) [14, 183, 235] and constraint optimization problem (COP) [14, 235].

A CSP tests the existence of a feasible solution given problem requirements. Formally, a CSP $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ consists of a set of variables $\mathcal{X} = \{x_1, \ldots, x_n\}$ and a set of constraints $\mathcal{C} = \{C_1, \ldots, C_m\}$. Each variable x_i is characterized by a domain D_i that describes its candidate values. Together, the set of variable domains is denoted as $\mathcal{D} = \{D_1, \ldots, D_n\}$. A feasible solution to a CSP is then a set of values for variables $\mathcal{X}$ from $\mathcal{D}$ that satisfies all constraints in $\mathcal{C}$.

A COP is the optimization version of the CSP in that it has an objective function. The optimal solution to a COP must be a feasible solution that produces an objective value no worse than all other feasible solutions.

Due to different developmental origins [159], the variables and constraints available to a modeller vary from solver to solver. In this thesis, we only discuss those from IBM's CP Optimizer [128], taking advantage of both non-scheduling-based and scheduling-based tools.

3.2.1.1 Variables

The types of CP variables are diverse. In addition to integer variables and binary variables, a CP modeller has access to interval variables, state functions, and cumulative functions. These variables are widely used in scheduling problems, so these CP formulations are generally referred to as CP scheduling [14, 159].

Optional Interval Variables An optional interval variable [130, 159], OptIntervalVar, is a variable whose domain is a subset of $\{\perp\} \bigcup \{[s, \epsilon) \mid s, \epsilon \in \mathbb{Z}, s \leq \epsilon\}$, where s and ϵ are the start and end times of the interval, and $\perp$ is a special value indicating absence. A duration of an interval variable refers to the value $\epsilon - s$, and can itself be a decision variable. When $\perp$ is removed from the domain, the interval variable must be present in a solution and can be declared using the signature IntervalVar. Example usage includes representing jobs and operations executed over a period of time.

CP solvers provide methods to access properties of interval variables [130, 159] (Figure 3.1). The presence of an interval variable *var* can be accessed via PresenceOf$(var) \in \mathbb{B}$. A value 1 indicates the interval variable is present and 0 not present. The starting and ending coordinates of an interval variable *var* can be accessed using StartOf$(var) \in \mathbb{Z}$ and End$(var) \in \mathbb{Z}$. The duration or length of an interval variable *var* can be accessed using Length$(var) \in \mathbb{Z}$. If *var* is absent, then these methods all evaluate to 0. An example of constraints restricting an interval variable is shown in Section 3.2.1.3.

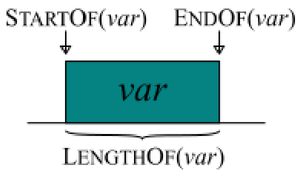


Figure 3.1: Illustration of an interval variable *var* and some of its methods.

State Functions A state function [135, 159] is a variable whose domain consists of sets of non-overlapping intervals, with each interval characterized by a non-negative integer state. In other words, the value of a state function is a sequence of state intervals and state values formalized as $\{[s_i, \epsilon_i) : v_i \mid \forall i \in [1, n]\}$, such that $\forall i \in [1, n], s_i, \epsilon_i, v_i \in \mathbb{Z} \wedge s_i \leq \epsilon_i$ and $\forall i \in [1, n-1], \epsilon_i \leq s_{i+1}$, where s_i, ϵ_i, and v_i are the start time, end time, and state value of state interval i, respectively, and $n \in \mathbb{Z}^+$ is the number of states in the state function.

An example application of the state function is the description of an exercise regimen with three possible intensity levels indexed by 0, 1, and 2. For the solution depicted in Figure 3.2, the first state is between time 0 to 20, and its state value, 0, indicates normal intensity. The second state describes a low intensity workout (state value 1) from time 20 to 29, and the third state returns to normal intensity (state value 0) from time 29 to 33. Finally, the last stage of the workout is of high intensity (state value 2) and lasts from time 33 to 40. Constraints that can be applied to a state function are formalized in Section 3.2.1.2.2. An example using these constraints to restrict the state function is shown in Section 3.2.1.3.

We will make extensive use of the state functions to model guillotine cuts in Chapter 4 and stock treatments in Chapter 5.

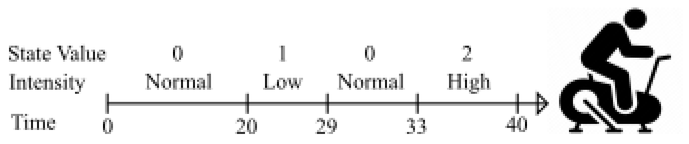


Figure 3.2: Example of a state function representing a workout regimen with intensities indexed by 0, 1, and 2.

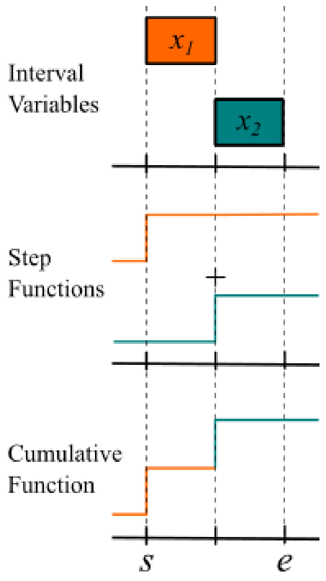


Figure 3.3: Step functions starting at the start time of the interval variables.

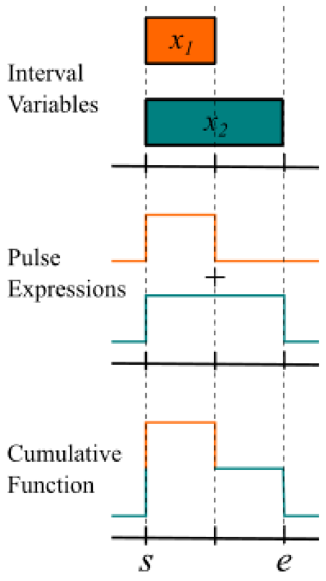


Figure 3.4: Pulses expressions over the interval variables.

Cumulative Function Expressions Cumulative functions [126, 159] are expressions that represent the sum of contributions of intervals over time. Formally, a cumulative function f is defined as the summation of elementary cumulative functions f_i such that $f = \sum_i f_i$ [34]. The most basic elementary function is a $\textsc{Step}(time, height)$, which equals 0 before *time* and *height* after *time*. Another elementary cumulative function is a $\textsc{Pulse}(interval, height)$, which equals *height* over the *interval* and 0 outside of the *interval*. Examples of cumulative functions using step functions and pulse expressions are shown in Figure 3.3 and Figure 3.4, respectively. A typical usage of these cumulative function expressions is to model resource consumption over time. An example of constraints restricting a cumulative function is shown in Section 3.2.1.3.

Both state functions and cumulative functions describe changes in behaviour over time. The difference is that the former describe absolute changes, while the latter describe relative ones [135].

3.2.1.2 Constraints

Just like the variables, the types of CP constraints are expansive [159, 160]. In general, we can distinguish two types of constraints: generic constraints and global constraints.

3.2.1.2.1 Generic Constraints Generic constraints are simple constraints widely adopted in different CP solvers.

Arithmetic Constraints Arithmetic constraints [123] are constraints that use arithmetic operators between expressions.

Logical Constraints/Expressions Logical constraints and expressions [132] are those that use the following logical operators: conjunction ($\wedge$), disjunction ($\vee$), negation ($\neg$), and implication ($\Rightarrow$). Logical expressions can be combined with arithmetic constraints so that "True" is evaluated as 1 and "False" as 0. For example, the constraint $(True \wedge True) + \neg(False) == 2$ is always "True", because both terms on the left hand side evaluate to "True" and $1 + 1 = 2$.

3.2.1.2.2 Global Constraints Global constraints are declarative constraints used to represent frequently recurring substructures found in different problems. Modelling substructures using global constraints often allows an enhanced level of inference compared to modelling using generic ones. In this section, we only describe the global constraints used in the thesis; a more exhaustive list can be found at the Global Constraint Catalogue [21].

Element Constraint The Element constraint [113, 127], $\textsc{Element}(array, index)$, acts as a subscripting operator to access the $index^{th}$ element in the *array*. In this thesis, we let the index of the first element in *array* always be 0. For example, consider an integer variable x representing the array indices and an integer array $a = [1, 2, 3, 4, 5]$. If we prescribe the constraint $\textsc{Element}(a, x) == 3$, then the only feasible solution is $x = 2$. The $\textsc{Element}(array, index)$ constraint is often abbreviated as $array_{index}$.

Count Constraint The $\textsc{Count}(array, val)$ constraint [125] counts the number of occurrences of *val* in the *array*. If *val* is not in *array*, then the constraint returns 0.

Lexicographic Constraint The lexicographic constraint [131], $\textsc{Lexicographic}(a, b)$, ensures that array a is always lexicographically not greater than array b. Formally, $\textsc{Lexicographic}(a, b)$ is satisfied if and only if either $a = b$ or $\exists i < size(a)$ such that $\forall j < i$, $a_j = b_j$ and $a_i < b_i$. For example, given array $a = [2, 1, 3]$, $b = [2, 2, 1]$, and $c = [1, 1, 1]$, a is lexicographically less than b, because there exists $i = 1$ such that, for $j = 0$, $a_0 = b_0 = 2$ and $a_1 = 1 < b_1 = 2$. However, a is not lexicographically less than c.

Pack Constraint The Pack constraint [133, 221], $\textsc{Pack}(load, where, size)$, describes a 1D packing substructure where the items are packed into bins and the bin capacity cannot be exceeded. Syntactically, *load* is a vector of variables describing the sum of item sizes in each bin; *where* is a vector of variables representing the bin index each item is assigned to;

and *size* is a vector of values dictating item size. Each element in *where* must correspond to the element in the same index in *size*.

No Overlap (Disjunctive) Constraint The NoOverlap(*sequence*) constraint [159, 160] ensures that interval variables in the set *sequence* do not overlap. This constraint is also referred to as the Disjunctive constraint in the Global Constraint Catalogue [21].

Forbid Extent Constraint The ForbidExtent(*var*, *sf*) constraint [159, 160] explicitly prohibits the interval variable *var* to overlap with non-zero regions of a step function *sf*. If *var* is absent, then the constraint is always satisfied.

AlwaysIn Constraint The AlwaysIn(*function*, *interval*, *min*, *max*) constraint [159, 160] restricts the value of a cumulative function or a state function, *function*, to a particular range [*min*, *max*] during an interval *interval*.

AlwaysEqual Constraint The AlwaysEqual(*stateFunction*, *interval*, *value*, *startAlign*, *endAlign*) constraint [159, 160] ensures that the value of a state function *stateFunction* during an *interval* is always *value*. CP Optimizer provides optional boolean arguments *startAlign* and *endAlign* that align the starting and ending coordinates of *interval* with the interval of a state in *stateFunction* if True. These are False by default.

AlwaysConstant Constraint The AlwaysConstant(*stateFunction*, *interval*, *startAlign*, *endAlign*) constraint [159, 160] is identical to AlwaysEqual, except that it is satisfied as long as the state value during *interval* is constant.

3.2.1.3 Examples

The following satisfaction problems exemplify the usage of variables and select constraints used in this thesis.

Example 1 (Constrained Interval Variables). *Given an interval variable x, we can constrain the start time, end time, presence, and length using constraints (3.4a) - (3.4d). The resulting variable domain is illustrated in Figure 3.5.*

$$\textsc{StartOf}(x) \geq s \tag{3.4a}$$

$$\textsc{EndOf}(x) \leq \epsilon \tag{3.4b}$$

$$\textsc{PresenceOf}(x) = 1 \tag{3.4c}$$

$$l \leq \textsc{LengthOf}(x) \leq u \tag{3.4d}$$

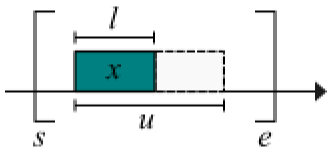


Figure 3.5: Illustration of a constrained interval variable.

Example 2 (Intervals in a State Function)**.** *Given interval variables x_i $\forall i = 1 \dots 6$ and a state function g, we can constrain the intervals to match the states in the state function with a possible solution displayed in Figure 3.6. This solution will remain feasible with constraints (3.5a) - (3.5d). Constraints (3.5a) - (3.5c) restrict the value of intervals to always take on a particular state. Constraint (3.5d) ensures that, within the interval $[b, d)$, the value of the states does not change.*

$$\textsc{AlwaysEqual}(g,\ x_1, 0, True, True) \tag{3.5a}$$

$$\textsc{AlwaysEqual}(g,\ x_i, 1, False, False) \quad \forall i = 2 \dots 5 \tag{3.5b}$$

$$\textsc{AlwaysEqual}(g,\ x_6, 2, False, False) \tag{3.5c}$$

$$\textsc{AlwaysConstant}(g, [b, d], False, False) \tag{3.5d}$$

$$\tag{3.5e}$$

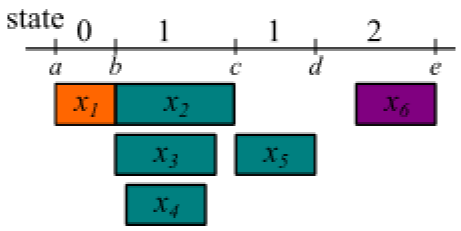


Figure 3.6: Illustration of interval variables being restricted by a state function.

The values of the boolean arguments startAlign and endAlign are important. These arguments in constraint (3.5a) align both the start and end coordinates of state 0 with those of x_1. The arguments in constraint (3.5b) do not enforce the alignment of coordinates, which is reflected in the positions of x_2, x_3, and x_4. However, the starting coordinate of x_4 in the current solution would violate the hypothetical constraint $\textsc{AlwaysEqual}(g, x_i, 1, True, False)$ *for $i = 2 \dots 5$. Similarly, the ending coordinates of both x_3 and x_4 in this solution would violate the hypothetical constraint* $\textsc{AlwaysEqual}(g, x_i, 1, False, True)$ *for $i = 2, \dots, 5$.*

Example 3 (Construction of a Cumulative Function)**.** *The solution illustrated in Figure 3.4 has interval variables x_1 and x_2, each contributing a pulse of magnitude h_1 and h_2, respectively, towards the cumulative function f. The solution will remain feasible if we apply the constraint (3.6a), as the maximum of the cumulative function is not greater than*

$h_1 + h_2$ *and the minimum not less than* 0.

$$\textsc{AlwaysIn}(f,\ x,\ 0,\ h_1 + h_2) \tag{3.6a}$$

3.2.2 Solving

Like MILP, CP employs a branch-and-bound scheme that branches on decision variables, bounds the objective, and backtracks to previous branches; the biggest difference is the use of inference as the main mechanism to reduce search. MILP takes advantage of the geometric properties of the solution space to develop a linear relaxation, make cuts, and prune branches. CP, in contrast, relies on constraint propagation, a logic-based algorithmic approach to communicate domain reductions of variables [122, 124, 216].

The extent to which variable domains are reduced can be described by *consistency* [122, 142]. The literature on the types of consistency is broad, so we only discuss domain consistency, the most fundamental type [34, 122, 142]. Intuitively, for a constraint, domain consistency, also known as generalized arc consistency, refers to the state where all the values in the domains of all variables that appear in the constraint participate in at least one solution to that constraint [122, 142]. Formally, given a CSP $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$, a constraint $C_k \in \mathcal{C}$ acting on a subset of variables $\mathcal{X}_k \subseteq \mathcal{X}$ is domain consistent only if $\forall X_i \in \mathcal{X}_k$, $\forall \delta_i \in D_i$, $\exists$ a tuple of values d with elements $d_j \in \{D_j \in \mathcal{D} \mid i \neq j\}$ such that $C_k(X_i = \delta_i, X_j = d_j \forall X_j \in \mathcal{X}_k \mid i \neq j)$ is satisfied. The subset of variables $\mathcal{X}_k$ is also known as the scope of this constraint [142]. Accordingly, a CSP is domain consistent if and only if all constraints are domain consistent [122, 142]. At each iteration, solvers execute filtering algorithms to prune the variable domains. Ideally, the pruning enforces some form of consistency, but this can be computationally expensive. So, instead, filtering algorithms implemented in solvers sometimes sacrifice a guarantee on consistency for a polynomial runtime [122, 142, 216]. Other theoretical forms of consistencies include bounds consistency and k-consistency [122].

3.2.3 Software

CP solvers are also widely available. Popular options include IBM's ILOG CP Optimizer [128], Choco [199], Gecode [50], and Google's CP-SAT Solver [206].

3.3 Packing Problems

The first documented study of packing dates back to the 1930s, when Kantorovich solved the first mathematical formulation to increase the industrial efficiency for Soviet Union [143]. Since, packing-related problems have expanded beyond borders, seeping into many facets of the modern society including manufacturing [136, 163, 232], technology [222, 236], and supply chain [136, 205]. In this section, we review related literature on two of the most prominent types of packing problems: one-dimensional bin packing and two-dimensional bin packing.

3.3.1 One-dimensional Packing Problems

In this section, we review literature on two main classes of one-dimensional packing: the 1D Bin Packing Problem (1DBPP) and the 1D Cutting Stock Problem (1DCSP) [93, 239].

An NP-hard optimization problem, 1DBPP packs a set of items, I, of size $w_i, i \in I$ into a set of bins K, B_k of which have size W_k, while minimizing the empty space within the bins. This leftover empty space is termed waste. As the sizes of items are constants, 1DBPP can be equivalently expressed as a problem that minimizes the total size of bins used.

The 1DCSP considers packing a set of orders $i \in N$ for b_i items with size w_i into an a set of stocks K of size W_k while minimizing waste. The two problems are very similar: 1DCSP can be reduced to a 1DBPP if the order demand $b_i = 1$ for $i \in N$ [239]. For the remainder of this section, we present models to describe the more general 1DCSP and interchangeably use the terminologies "bin" and "stock".

Next, we highlight common MILP and CP formulations and summarize solution approaches.

3.3.1.1 Mixed-Integer Programming

Assignment-based Models Assignment-based models use integer variables x_{ik} to explicitly decide if item i is assigned to stock k. Should there be duplicates of orders, the demands can be aggregated to reduce symmetry between these items. Below, we reproduce the MILP model proposed by Kantorovich [143], who additionally used binary variables y_k to track if stock k is used. Constraints (3.7b) and (3.7c) capture the demand and the size-wise capacity, respectively. The rest define the variables.

$$\begin{aligned}
\min \quad & \sum_{k \in K} W_k y_k && && (3.7a)\\
\text{s.t.} \quad & \sum_{k \in K} x_{ik} \geq b_i && \forall i \in N && (3.7b)\\
& \sum_{i \in N} w_i x_{ik} \leq W_k y_k && \forall k \in K && (3.7c)\\
& x_{ik} \in \mathbb{Z}^+ && \forall i \in N, k \in K && (3.7d)\\
& y_k \in \mathbb{B} && \forall k \in K && (3.7e)
\end{aligned}$$

Compared to the models presented later, this formulation typically yields a weak linear relaxation and hence a weak lower bound. For a 1DBPP with identical bins ($W_k = W$ for $k \in K$), the relaxed lower bound is always $\lceil \sum_{i \in I} w_i / W \rceil$ [187].

Position-indexed Models Position-indexed models use variables indexed by numerical coordinates on a stock to indicate if an item starts at a coordinate in a solution. To achieve a more compact representation, the positional relationship of items and stocks is often characterized by an arc-flow graph G with vertices $V = \{0, \ldots, W_{max}\}$ being the positions and arcs $A = \{(s, e) : (0 \leq s < e \leq W_{max}) \wedge (\exists i \in N | e - s = w_i)\}$ being the item placements. Intuitively, an item belonging to order i can only be packed into any stock with the leftmost coordinate being s if the arc $(s, s + w_i)$ is selected. If the stocks are identical, then the arc flow graph G is sufficient, as none of the arcs will finish beyond the stock size. However, with differently sized stocks, two sets of arcs need to be added to A to discriminate stock types. First, all vertices in V are connected to a sink node with coordinate W_k for each stock k. Each unit of flow that reaches the sink at W_k signals the usage of another stock of size W_k. Then, as the number of total stocks consumed always equals the units of flow across all sinks, we can add an additional arc that connects the sinks to coordinate 0, so as to complete the arc flow cycle.

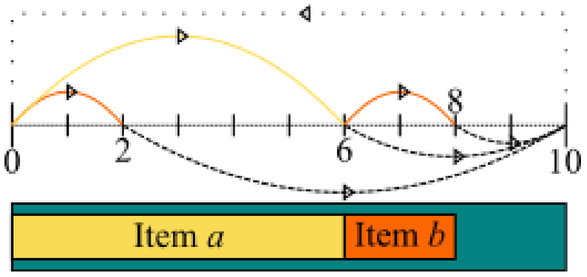


Figure 3.7: Example of the one-dimensional arc flow model for packing two items into a bin of size 10. Item a is of size 6 and item b is of size 2. The upper half of the diagram is the complete arc flow graph and the lower half is one feasible packing scheme. Orange and yellow arcs indicate different items packed into the bin. Dashed arcs are added to ensure a common sink. A dotted arc is added to complete the arc flow cycle.

Figure 3.7 illustrates an example of the position-indexed model packing two items into a bin of size 10. Item a has size 6 and item b has size 2. In total, there are three possible packing combinations: packing only item a, packing only item b, and packing both item a and b. The first combination is illustrated by the orange arc (0, 2), which indicates the bin usage from coordinate 0 to coordinate 2, and the dashed arc (2, 10), which connects to the sink node. Similarly, the second combination is illustrated by the yellow arc (0, 6) and the dashed arc (6, 10). The last combination, cutting both item a and b, has arcs (0, 6) and (6, 8), as well as the dashed arc (8, 10). In all three cases, one unit of flow exits the origin node and finishes at the sink node at coordinate 10.

We reproduce Carvalho's 1D position-indexed model [47] below. Integer variables x_{se} are introduced to represent the number of items obtained from arcs (s, e). Integer variables z_k are added as a feedback arc from vertex W_k to vertex 0 to represent the number of stocks of type k used. Constraint (3.8b) enforces that the amount of flow in the incoming arcs must equal to the amount in the outgoing arcs. Semantically, in the case where vertex e equals 0, Constraint (3.8b) ensures that the number of items with their leftmost coordinates being 0 equals the number of stocks used. In the case where $e = W_k$ for some stock type k, the number of items with their rightmost coordinates being e must equal to the summation of the number of stocks of type k used and the number of items with their leftmost coordinates being e. The final case is similar to the case where $e = W_k$, except no stocks are exhausted yet. Constraint (3.8c) ensures the number of arcs with size w_i is enough to satisfy the demand of order i. Constraint (3.8d) ensures only available stocks are used. The rest describe the variables. Since the model size is dependent on the graph size, recent work [40] has developed graph compression techniques to address scaling challenges.

$$\min \sum_{k=1}^{K} W_k z_k \tag{3.8a}$$

$$\text{s.t.} \; -\sum_{(d,e)\in A} x_{de} + \sum_{(e,f)\in A} x_{ef} = \begin{cases} \sum_{k=1}^{K} z_k & \text{if } e = 0 \\ -z_k & \text{for } e = W_k, \quad \forall k = 1,\dots,K \\ 0 & \text{otherwise} \end{cases} \tag{3.8b}$$

$$\sum_{(d,d+w_i)\in A} x_{d,d+w_i} \geq b_i \qquad \forall i = 1,\dots,m \tag{3.8c}$$

$$z_k \leq B_k \qquad \forall k = 1,\dots,K \tag{3.8d}$$

$$x_{de} \in \mathbb{Z}^+ \qquad \forall (d,e) \in A \tag{3.8e}$$

$$z_k \in \mathbb{Z}^+ \qquad \forall k = 1,\dots,K \tag{3.8f}$$

One-Cut Models Instead of assignments or positions, one-cut models focus on modelling cuts and their byproducts. The key observation here is that, after each cut, we always obtain an item and a leftover rectangle, the latter of which is referred to as a residual plate. As any large enough residual plate can be cut to produce an item and another residual plate, we can obtain all combinations of item assignments inductively.

Figure 3.8 illustrate the running example packing two item of size 2 and 6 into a bin of size 10. The first cut is at coordinate 2, producing a targeted plate of size 2, which fulfills the first item, and a residual plate of size 8. Another cut is executed on the residual plate to produce a targeted plate of size 6, which fulfills the second item.

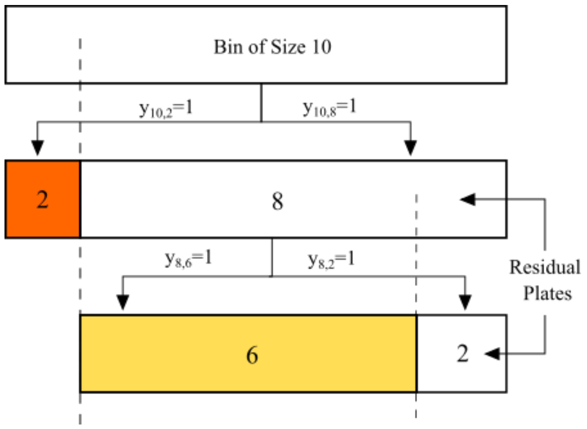


Figure 3.8: Example of a one-dimensional one-cut model for packing two items (size 2 and 6) into a bin (size 10). The white rectangles are either the bin or residual plates.

To illustrate, we reproduce a 1D one-cut model by Dyckhoff [71] in Model 3.9. Given a set of stock size $H = \{W_k, k \in K\}$ and a set of orders with unique size $D = \{w_i, i \in N\}$, let $y_{p,q}$ denote the number of plates with size p that are divided into a targeted plate of size q and a residual plate of size $p-q$. We also let R be the set of these residual plates excluding those that are smaller than the smallest order size, and N_q be the demand of rectangular items of size q for $q \in \{1, \ldots, \max_{k\in K} W_k\}$. Note that if type $q \notin D$, then $N_q = 0$. If a residual plate of size $q \in H$ is cut (possibly to produce an item-sized plate), then there must be at least one resource that is cut to yield the residual plate. This resource can either be a stock of size q or another residual plate derived from a larger stock, so the number of these resources cannot be less than their derivatives, as is formalized by constraint (3.9b). For plates with non-stock sizes, constraint (3.9c) ensures that they are cut as either residual plates or targeted plates and are either used to fulfill corresponding orders, if any, or cut into smaller plates.

$$\min \sum_{q\in H} W_q z_q \tag{3.9a}$$

$$\text{s.t.}\quad z_q + \sum_{p\in D: p+q\in H\cup R} y_{p+q,p} \geq \sum_{p\in D:p<q} y_{q,p} \qquad \forall q \in H \tag{3.9b}$$

$$\sum_{p\in H\cup R:p>q} y_{p,q} + \sum_{p\in D:p+q\in H\cup R} y_{p+q,p} \geq \sum_{p\in D:p<q} y_{q,p} + N_q \forall q \in (D\cup R)\backslash H \tag{3.9c}$$

$$z_q \in \mathbb{Z}^+ \qquad \forall q \in H \tag{3.9d}$$

$$y_{p,q} \in \mathbb{B} \qquad \forall p \in H\cup R, q \in D, q < p \tag{3.9e}$$

Set Cover Models Set cover models explicitly enumerate all size-wise combinations of items and try to find an optimal subset [94, 240]. This formulation typically yields a strong linear relaxation, but, as the number of these combinations is exponential, solving it can often be intractable. To avoid generating all combinations, these models use the column generation technique embedded in a branch-and-price algorithm to generate good combinations during runtime, albeit at the expense of being less flexible and more difficult to implement [76, 240].

$$\min \sum_{p \in P} x_p \tag{3.10a}$$

$$\text{s.t.} \quad \sum_{p \in P} a_p^i x_p \geq b_i \quad \forall i \in N \tag{3.10b}$$

$$x_p \in \mathbb{Z}^+ \quad \forall p \in P \tag{3.10c}$$

Here, we denote the set of all size-wise patterns as P. For each pattern $p \in P$, the parameter a_p^i describes the number of items belonging to order i in pattern p. Since the combinatorics of packing is encapsulated in the pattern set, the model only needs to constrain the demand.

3.3.1.2 Constraint Programming

The constructs used in CP models are highly dependent on the underlying solvers. Modern solvers, such as CP Optimizer [128] and Gecode [50], offer the PACK constraint introduced in Section 3.2.1.2. Below, we describe a formulation using PACK. Other formulations have been investigated, albeit using features from dated solvers, such as the DIFFN and CUMULATIVE constraints from the CHIP solver [20, 231].

Bin-Indexed Model Bin-indexed models introduce, for each item i, an integer variable x_i that takes on the index of the assigned bin. In particular, this type of model shows particular affinity with the global constraint PACK, which uses a filtering algorithm to prune a CP search if a subset of items cannot be packed within some bound on the net usage of a bin. Shaw [221] demonstrated that this global constraint can cut the search by orders of magnitude. An 1DBPP modelled using PACK is as follows.

$$
\begin{aligned}
\min \quad & \sum_{k \in K} W_k (b_k > 0) && && \text{(3.11a)}\\
\text{s.t.} \quad & \textsc{Pack}(b, x, w) && && \text{(3.11b)}\\
& b_k \leq W_k && \forall k \in K && \text{(3.11c)}\\
& x_i \in K && \forall i \in N && \text{(3.11d)}\\
& b_k \in \mathbb{Z}^+ && \forall k \in K && \text{(3.11e)}
\end{aligned}
$$

As required by the Pack in constraint (3.11b), this model introduces variables b_k to track the net usage of each bin k. The capacity of each bin is restricted in constraint (3.11c).

3.3.1.3 Other Works

Other exact approaches are as follows. Vance [240] proposed a branch-and-price algorithm to solve a Dantzig-Wolfe decomposition of the Kantorovich model and showed that, with a constant bin size, the decomposed model is equivalent to the set cover model. Valério de Carvalho [47] proposed a branch-and-price scheme to solve the position-indexed model. Exact search algorithms have also been proposed. Martello and Toth [187] proposed the Martello-Toth Procedure (MTP), a branch-and-bound algorithm that branches on item assignments, derives upper bounds using a First-fit Decreasing heuristic (FFD), and prunes branches via dominance relationships. Scholl et al. [218] developed a fast hybrid procedure called BISON, which uses a tabu search to improve solutions found by a fast non-exact heuristic and subsequently applies branch-and-bound. Their experiments show that the solutions found are superior to those of MTP while requiring less time on average. While MTP branches on items, Korf [152, 153] developed a bin completion branch-and-bound algorithm that branches on item subsets comprising a bin. They showed that their branching scheme enables a stronger dominance relationship between nodes and hence a smaller search tree. Later, Fukunaga and Korf [85] extended the bin completion algorithm to other variations of the 1DBPP.

The literature on non-exact approaches on 1DCSP and 1DBPP is substantial. Johnson [140, 139] proposed the well-known FFD and Best-Fit Decreasing (BFD) greedy algorithms and derived their asymptotic worst-case behaviours. Gupta and Ho [105] developed the Minimum-Bin-Slack (MBS) heuristic that, at each iteration, tries to pack a set of items as close to a bin's capacity as possible. Fleszar and Hindi [80] proposed three variations of MBS and proposed a variable-neighbourhood-search algorithm. They showed that coupling the search algorithm with the modified MBS is advantageous. Alvim and Ribeiro [7] proposed a heuristic that used load redistribution and a tabu search. Fleszar and Charalambous [79] developed heuristics that pack a sufficient average weight for each remaining bin. Loh et al. [178] developed a weight annealing heuristic that changes the item size based on search history to escape local extrema. Falkenauer [75] solved the 1DBPP using genetic algorithms. He proposed three different encodings and demonstrated that a group-based encoding can find solutions as well as MTP but with less time. Singh and Gupta [227], Poli et al. [209], and Rohlfshagen and Bullinaria [217] investigated other evolutionary approaches to 1DBPP.

3.3.2 Two-dimensional Rectangular Packing Problems

Two-dimensional packing problems involve packing rectangles of fixed sizes into rectangular bins a.k.a stocks. In this section, we first provide a classification scheme for these problems. As the literature is diverse, we limit our review to literature related to our problem.

3.3.2.1 Classification Scheme

We can categorize two-dimensional packing problems based on problem types and characteristics. An overview of our classification scheme is shown in Figure 3.9.

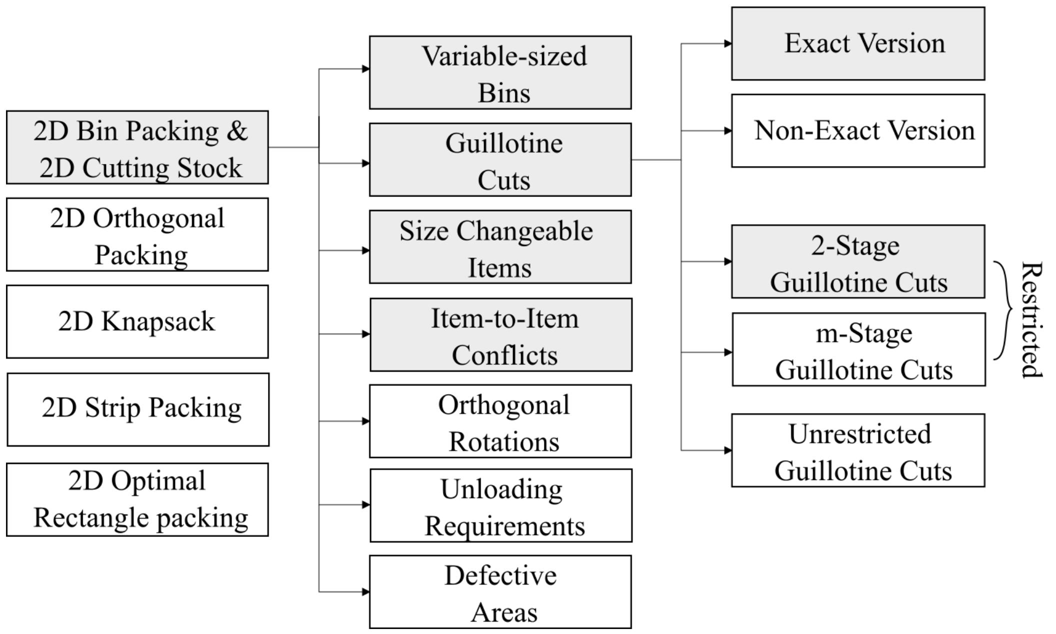


Figure 3.9: Classification of 2D packing problems. Categories associated with our problem are highlighted in grey.

There are five main types of two-dimensional packing problems (Table 3.1): the 2D Orthogonal Packing [54, 55, 58, 98, 99, 137, 167, 191], 2D Single and Multiple Knapsack Problem [38, 46, 72, 85, 115, 116, 149, 165, 177, 238, 187], 2D Strip Packing Problem [24, 30, 31, 37, 39, 41, 53, 64, 101, 147, 164, 230], 2D Optimal Rectangle Packing Problem [154, 155, 194, 226, 225], and 2D Bin Packing and Cutting Stock Problem. The 2D Orthogonal Packing is a feasibility problem, checking if a set of rectangular items can be packed into a single stock. The 2D Single Knapsack Problem packs the optimal subset of rectangles into

	Goal	Single Bin	Multiple Bins	Infinite Bin Length
2D Orthogonal Packing	Feasibility	x		
2D Single Knapsack	Max Packed Value	x		
2D Multiple Knapsack	Max Packed Value		x	
2D Strip Packing	Min Length	x		x
2D Optimal Rectangle Packing	Min Enclosing Rectanglular Area	x		
2D Bin Packing/Cutting Stock	Min Waste		x	

Table 3.1: Common Types of 2D Packing Problems.

a single stock while maximizing the rectangles' values. The 2D Multiple Knapsack Problem generalizes the 2D Single Knapsack Problem to include multiple stocks of different sizes. The 2D Strip Packing Problem packs items into a strip of infinite length while minimizing the used length. The 2D Optimal Rectangle Packing Problem attempts to find a rectangle with the smallest area that encloses all packed items. The 2D Cutting Stock Problem (2DCSP) and 2D Bin Packing Problem (2DBPP) pack items into stocks while minimizing waste.

The 2SCSP-FFMS is a generalization of 2DBPP and 2DCSP, and industrial applications have fostered many variations of these problems, including those with Variable-sized Bins [19, 86, 177, 208, 210, 223], Guillotine cuts [19, 86, 149, 148, 177, 190, 182, 223], Size Changeable Items [163], Item-to-Item Conflicts [73, 150, 168], Orthogonal Rotations [147, 171, 245], Loading and Unloading requirements [58, 166, 224], and Defective Areas [3, 4, 100, 180, 189, 200]. Their characteristics are described below.

- Variable-sized Bins: Different bins have different fixed dimensions.
- Guillotine Cuts: All cuts must be guillotine, a type of cut that runs from one side of the object to the other. The left and middle plot in Figure 3.10 show a guillotine cutting pattern and a non-guillotine one, respectively.
- Size Changeable Items: One or more dimensions of items are flexible.
- Item-to-Item Conflicts: Conflicting items cannot be placed in the same stock.
- Orthogonal Rotations: Items can rotate 90 degrees.
- Unloading Requirements: Packed items must be unloaded according to some sequence.
- Defective Areas: Items cannot overlap with parts of the stock that are defective.

Finally, the literature segments different guillotine cutting schemes according to cut restrictions and exactness. Cut restrictions limit the number of cuts that can be executed,

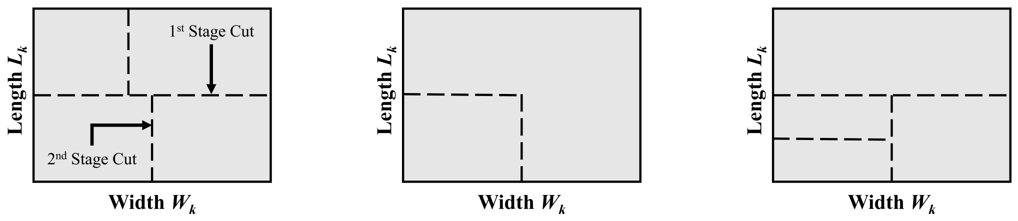


Figure 3.10: (Left) A 2-stage guillotine pattern. (Middle) An invalid guillotine pattern. (Right) A valid guillotine pattern that requires three stages of cuts.

often due to the limitations of industrial machines. *Two-Stage* problems allow only two stages of the cuts to be executed, each stage orthogonal to the other (Figure 3.10, left). Similarly, a solution from an *m-Stage* problem must be cuttable from m stages (e.g. Figure 3.10 shows a 3-stage pattern, right). If there are no restrictions on the number of stages, the problem is described as *unrestricted*.

Exactness describes if an item cut requires subsequent trimming. If items can be trimmed after cutting, it is a *non-exact version* of the problem. If no trimming is allowed, then the problem is *exact*. Indeed, the two perspectives can be unified. For instance, a 2-stage non-exact version is identical to a 3-stage exact version.

The problem that we focus on in this thesis, 2SCSP-FFMS (Section 2.1), is situated at the intersection of many difficult problems: 2DCSP with Variable-sized Bins, 2-stage Guillotine Cuts, Size Changeable Items, and Item-to-Item conflicts (Figure 3.9, highlighted). Next, we review 2DCSP literature with these characteristics.

3.3.2.2 Two-dimensional Packing with Guillotine Cuts and Variable-sized Bins

We first review 2DCSP and 2DBPP with Guillotine Cuts and Variable-sized Bins with a focus on modelling perspectives. The notations used to reproduce different formulations are as follows. We are given a set of stocks K with width W_k and length L_k and let H be the set of unique stock dimensions. We wish to fulfill a set of orders $i \in N$ of b_i items of dimensions $w_i \times l_i$. We refer to the problem as 2DBPP only if $b_i = 1$ and 2DCSP otherwise.

3.3.2.2.1 Mixed-Integer Programming In this section, we review the main MILP models for two-dimensional packing problems with 2-stage guillotine constraints and variable-sized bins, two variations most associated to our problem.

Assignment-based Model An assignment-based formulation for two-dimensional two-stage cutting stock problems (2SCSP) uses decision variables to explicitly represent the matching between items, levels, and stocks [86, 177]. Extending the assignment-based formulation from 1D to 2DCSP requires the representation of levels, whose quantity is unknown a priori. A frequently used upper bound on the number of levels is the total number of items, $\tilde{n} = \sum_{o=1}^{n} d_o$ where d_o is the demand of items from order $o \in N$ and $N = \{1, \ldots, n\}$

[86]. However, matching these levels to stocks invites significant symmetry, so studies have proposed assignment restriction schemes at the expense of enumerating all possible stock combinations, making this formulation ideal only if the total number of items is not large.

Here, we detail the assignment restriction scheme proposed by Furini and Malaguti [86]. First, they assume orders are sorted according to non-increasing width. Then, any items from order i must be cut from levels with indices in $\{1, \dots, \alpha_i\}$, where $\alpha_i = \sum_{o=1}^{i} d_o$ and $\alpha_0 = 0$. Conversely, level k can only take on items from orders $\{\beta_k, \dots, n\}$, where $\beta_k = \min\{r : r \in N, \alpha_r \leq k\}$. Similarly, a level k can only be assigned to stocks with indices $\{k \dots \tilde{n}\}$.

They used four sets of variables, all of which are initialized according to the restriction scheme. Binary variables y_j^h and q_k^h equal 1 only if level j and stock k of type h are initialized, respectively. A stock type characterizes the set of stocks with identical dimensions and let H denote the set of these types. Integer variable x_{ij}^h counts the number of items of order i assigned to level j of type h. Binary variable z_{jk}^h equals 1 only if level j is assigned to stock k of type h. Constraint (3.12b) ensures demand is fulfilled. Constraint (3.12c) and (3.12d) restrict the width of levels and the length of stocks, respectively. Constraint (3.12e) links the stock usage with level assignments.

$$\min \sum_{h \in H} W_h L_h \sum_{k=1}^{\bar{n}} q_k^h \tag{3.12a}$$

$$\sum_{h \in H} \Big(\sum_{j=1}^{\alpha_i} x_{ij}^h + \sum_{j=\alpha_{i-1}+1}^{\alpha_i} y_j^h \Big) \geq b_i \qquad i = 1 \dots n \tag{3.12b}$$

$$\sum_{i=\beta_j}^{n} w_i x_{ij}^h \leq (W_h - w_{\beta_j}) y_j^h \qquad j = 1 \dots \tilde{n} - 1, h \in H \tag{3.12c}$$

$$\sum_{j=k+1}^{\hat{n}} l_{\beta_j} z_{jk} \leq (L_h - l_{\beta_k}) q_k^h \qquad k = 1 \dots \tilde{n} - 1, h \in H \tag{3.12d}$$

$$\sum_{k=1}^{j-1} z_{jk}^h + q_j^h = y_j^h \qquad j = 1 \dots \tilde{n}, h \in H \tag{3.12e}$$

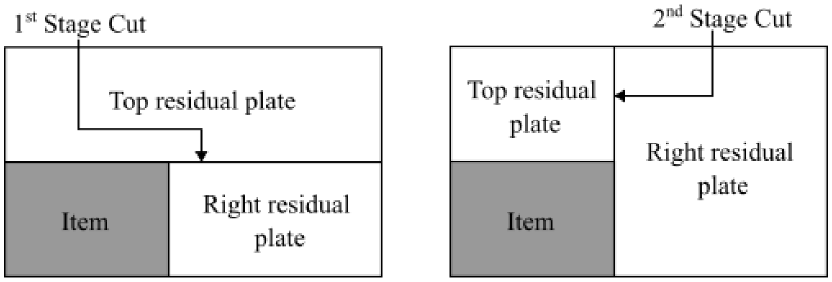
$$y_j^h \in \{0, 1\} \qquad j = 1 \dots n - 1, h \in H \tag{3.12f}$$

$$x_{ij}^h \in \mathbb{Z}_+ \qquad j = 1, ..\tilde{n} - 1, i = \beta_j, ..n, h \in H \tag{3.12g}$$

$$q_k^h \in \{0, 1\} \qquad k = 1, \dots, \tilde{n} : h \in H \tag{3.12h}$$

$$z_{jk}^h \in \{0, 1\} \qquad k = 1 \dots \bar{n} - 1, j = k \dots \bar{n}, h \in H \tag{3.12i}$$

3.3.2.2.1.1 Cut-and-Plate Model The Cut-and-Plate formulation extended the 1D One-Cut formulation to 2SCSP by expanding the definition of residual plates to include byproducts from first-stage cuts and second-stage cuts [19, 86, 223]. After cutting an item, the stock is divided into two residual plates, one above and one to the right (Figure 3.11).



(a) Residual plates after a first-stage cut

(b) Residual plates after a second-stage cut

Figure 3.11: Residual Plates after different types of cuts.

Cuts					**Plates**					
Plate Type			**Item Type**		**Top**			**Right**		
Width	**Height**	**Stage**	**Width**	**Height**	**Width**	**Height**	**Stage**	**Width**	**Height**	**Stage**
W_h	L_h	1	w_i	l_i	W_h	$L_h - l_i$	1	$W_h - w_i$	l_i	2
W_h	L_h	2	w_i	l_i	w_i	$L_h - l_i$	Waste	$W_h - w_i$	L_h	2

Table 3.2: Plate types resulting from cuts in different stages for a two-stage problem.

Their dimensions are dependent on the stage of the cut and are summarized in Table 3.2. The set of all possible residual plates is captured in the parameter a_{ijk}, which equates to 1 if plate type k results from cutting item type i from plate type j and 0 otherwise. Then, letting integer variables x_{ij} represent the number of items from order i cut from a plate j and m the total number of enumerated plates, the Cut-and-Plate model is as follows:

$$\min \sum_{h \in H} W_h L_h \sum_{i=1}^{n} x_{ih} \tag{3.13a}$$

$$\sum_{j=1}^{m} x_{ij} \geq b_i \qquad i = 1 \dots n \tag{3.13b}$$

$$\sum_{j=1}^{m} \sum_{i=1}^{n} a_{ijk} x_{ij} \geq \sum_{i=1}^{n} x_{ik} \qquad k = 1 \dots m \tag{3.13c}$$

$$x_{ij} \in \mathbb{Z}^+ \qquad i = 1 \dots n, j = 1 \dots m \tag{3.13d}$$

Objective (3.13a) describes the waste minimization objective. Constraint (3.13b) satisfies the demand. Constraint (3.13c) ensures that a plate can only be cut if it is a residual plate. The rest defines the variables.

3.3.2.2.1.2 Position-Indexed Model Extending the 1D position-indexed formulation, the 2D position-indexed formulation is an arc-flow model that associates each cutting stage with a graph whose arcs represent cut positions. In essence, the model combines two 1D arc-flow formulations, one to cut levels and another to cut items.

One example is the model of Macedo et al. [182], who tackled a problem with identical stock dimensions ($W = W_h$ and $L = L_h$ for $h \in H$). For the first stage, they considered the arc-flow graph $G_0 = \{V_0, A_0\}$, where the vertex set $V_0 = \{0, \dots, L\}$ is the set of possible integer positions and the arc set $A_0 = \{(a, b) : 0 \leq a < b \leq L \wedge b - a \in L^*\}$ represents cut items. Here, L^* is the set of possible length values. In the second stage, for a given level s with length $l_s^* \in L^*$, the arc-flow graph G_s has vertices $V_s = \{0, \dots, W\}$ and arcs $A_s = \{(d, e) : 0 \leq d < e \leq W \wedge \exists i \in N | (e - d = w_i \wedge l_i \leq h_s)\}$.

This model requires four sets of variables. Integer variable z^0 represents the number of stocks used, and integer variables z^s represent the number of cut levels with length $s \in L^*$. Integer variables x^0_{ab} describe the flow in G^0 and integer variables x^s_{del} the flow in G^s. In particular, x^s_{del} also represents the number of items with length l and width $e - d$ cut from a level with length s. If all items share different width values, then the last index l can be removed from x^s_{del}. Constraint (3.14b) enforces flow conservation in G^0. Constraint (3.14c) ensures that the number of levels with length s matches those cut from the first stage. Constraint (3.14d) conserves the flow in G^s. Constraint (3.14e) ensures the demand is fulfilled.

$$\min \quad z^0 \tag{3.14a}$$

$$\text{s.t.} \quad \sum_{(a,b)\in A^0} x^0_{ab} - \sum_{(b,c)\in A^0} x^0_{bc} = \begin{cases} -z^0 & \text{if } b = 0 \\ 0 & \text{if } b = 1, 2, \dots, H-1 \\ z^0 & \text{if } b = H \end{cases} \tag{3.14b}$$

$$\sum_{(c,c+s)\in A^0} x^0_{c,c+s} - z^s = 0 \qquad \forall s \in L^* \tag{3.14c}$$

$$\sum_{\substack{(d,e)\in A^s \\ l\in L^*}} x^s_{del} - \sum_{\substack{(e,f)\in A^s \\ l\in L^*}} x^s_{efl} = \begin{cases} -z^s & \text{if } e = 0, \\ 0 & \text{if } e = 1, 2, \dots, W-1 \\ z^s & \text{if } e = W, \end{cases} \qquad \forall s \in L^* \tag{3.14d}$$

$$\sum_{s\in L^*} \sum_{(f,f+w_i)\in A^s} x^s_{f,f+w_i,l_i} \geq b_i \qquad \forall i \in N \tag{3.14e}$$

$$x^0_{ab} \in \mathbb{Z}^+ \qquad \forall (a,b) \in A^0, s \in L^* \tag{3.14f}$$

$$x^s_{del} \in \mathbb{Z}^+ \qquad \forall (d,e) \in A^s, l, s \in L^* \tag{3.14g}$$

3.3.2.2.1.3 Set Cover Model The 2D set cover models employ the same idea as their 1D siblings, thus retaining their advantages and disadvantages. The difference lies in the pattern set: instead of widthwise ones, we now consider those that are feasible cutting schemes for the entire rectangular stock. In other words, each pattern must be executable using two stages of guillotines. Exemplifying the exponential-sized formulation with variable stock sizes is Model (3.15) by Furini and Malaguti [86], which is reproduced below.

$$\begin{aligned}
\min \quad & \sum_{h \in H} W_h L_h \sum_{p \in P^h} x_p && && (3.15a)\\
\text{s.t.} \quad & \sum_{h \in H, p \in P^h} a_p^i x_p \geq b_i && \forall i \in N && (3.15b)\\
& x_p \in \mathbb{Z}^+ && \forall p \in P^h, h \in H && (3.15c)
\end{aligned}$$

Differing from the 1D set cover model, which contains only 1D stocks of the same type, this model distinguishes patterns cuttable from rectangular stocks of types $h \in H$ by adding index h to the pattern set. The constraint and variable definitions are augmented accordingly.

3.3.2.2.2 Constraint Programming Dincbas and Simonis proposed the first CP-based approach [68] to the 2SCSP, generating stock patterns using a combination of back-track search and a finite domain model. Later, Beldiceanu and Contejean introduced DIFFN [22, 20], a global constraint with an option to enforce guillotine cuts; however, no experimental results related to guillotine cuts were provided. Since then, CP has largely been investigated in other 2D packing contexts. For the two-dimensional optimal rectangle packing problem, Korf [154, 155] considered solving a constraint satisfaction problem using the absolute positions of items. Moffitt and Pollack [194] studied the same satisfaction problem from a relative placement perspective, focusing on the pairwise relationships between items. For the same problem, Clautiaux et al. [55] considered a scheduling approach, representing the width and length of items as two interval variables. This model was improved by Mesyagutov et al. [191], who integrated linear-programming-based pruning rules to propagate the constraints. Simonis and O'Sullivan investigated CP search strategies to pack squares into rectangles using the CUMULATIVE global constraint [226, 225].

3.3.2.2.3 Other Approaches Puchinger and Raidl [214] proposed a branch-and-price algorithm for the set cover formulation. They added dual-optimal cuts, a column generation stabilization technique [8, 48] to accelerate subproblem convergence, and branched on if two items are in the same stock. Furini and Malaguti [86] also proposed a branch-and-price algorithm for the set cover model, but with a different branching rule and subproblem formulation. Monaci and Toth [195] and Cui et al. [60] developed two-phase algorithms, first generating two-stage patterns for a single stock and then assigning patterns to stocks. They differ in that the former used a heuristic and the latter a MILP model to minimize the number of stocks used in the second phase. Recently, Martin et al. [188] proposed a

Benders Decomposition on a position-based MIP formulation.

Many inexact algorithms have been proposed in the literature. Incidentally, the solutions of early heuristics designed for 2D packing problems without guillotines are naturally feasible for problems with guillotine constraints [173]. Chung et al. [51] presented the Hybrid First-fit Decreasing Heuristic (HFFD) that packs items into levels without taking into account the level length and then packs these levels into stocks, both according to a first-fit decreasing scheme. Frenk and Galambos [84] studied the Hybrid NFD heuristic (HNFD), a Next-fit version of the HFFD. Berkey and Wong [27] developed the Finite First-Fit (FFF), Finite Next-Fit (FNF), Finite Best-Strip (FBS), and Finite Bottom Left (FBL) heuristics. FFF and FNF both pack items into levels directly into stocks. If the item cannot be packed into a level, a new level is opened, and if the item cannot fit into the new level, then a new bin is opened. FBS is essentially an HFFD using best-fit strategies. Lastly, FBL always packs into the closest bottom-left corner of the stock if possible. In the literature, HFFD, HNFD, FBS are classified as *2-phase heuristics*, and the rest as *1-phase heuristics* [173]. Caprara et al. [45] proposed an algorithm for 2SCSP that combined grouping techniques with the simple heuristics. They show that this algorithm always executes in polynomial time if the instance size approaches infinity, making it an asymptotically polynomial-time algorithm.

Metaheuristics have also been studied. Lodi et al. [114, 172, 175, 176] proposed a generalized tabu search procedure for two-dimensional packing problems. Given a solution, they first try to find a stock with large waste and a high number of items. Then, to find a solution using fewer stocks, they execute a move: one item from that stock and items from k other stocks are repacked by a heuristic. They also consider diversifying the search by incrementing k and destroying poorly packed bins. For both problems with and without guillotine requirements, they show that their solutions are superior to other heuristics. Gharsellaoui and Hasni [92] developed a genetic algorithm that replaces the mutation step with a tabu search. Alvelosa et al. [6] explored neighbourhoods for a local search, swapping both adjacent item types and levels.

3.3.2.3 Two-dimensional Packing with Size Changeable Items

While the two-dimensional packing problems have been widely studied, we could find only one work addressing item flexibility in the 2D setting. Lee et al. [163] considered a variant of the 2SCSP with flexible width and length and proposed a multi-stage heuristic to iteratively pack items and adjust level dimensions. Notably, their heuristic uses three main components. Given a set of unpacked items, the algorithm first uses a knapsack-based algorithm to greedily pack items into a level on a candidate available stock. If no new levels can be opened on the stock, the current packing scheme is compared with a hypothetical one that forcibly divides the last level into two, and the solution with less waste is accepted. Finally, the heuristic checks if this packing scheme yields less waste if it is executed on a differently sized stock. These steps are repeated until all items are packed. Their empirical experiments on a set of industrial data show superior performance over historical decisions. They also proposed a non-linear model but did not investigate its performance.

3.3.2.4 Two-dimensional Packing with Conflicts

Literature on two-dimensional packing problems with conflicts has exclusively focused on heuristics. Epstein et al. [73] studied the problem of packing squares into a set of identical square stocks with bipartite and perfect conflict graphs and proposed an ensemble of heuristics for each graph type. Their algorithms for bipartite graphs and perfect graphs have an approximation ratio of at most $2+\epsilon$ for $\epsilon > 0$ and 3.2744, respectively. Khanafer et al. [150] developed a multi-step algorithm to tackle two-dimensional bin packing problems with any conflict graph. They first compute the compatibility graph and apply a tree-based decomposition to identify clusters of compatible items. Items appearing in multiple clusters are assigned heuristically to one of the clusters. Then, orders in each cluster are packed using simple heuristics, and the solutions of clusters are merged. They show practical effectiveness and note that exact algorithms can replace the heuristics, but did not investigate this scheme's performance. Li et al. [168] adapted a maximal space algorithm proposed by El Hayek et al. [112] for vanilla 2D bin packing problems and proposed a local search approach. They showed that, in their industrial application, combining the adapted maximal space algorithm with the local search yields strong solutions.

3.4 Resource Constrained Scheduling

Scheduling refers to the problem of optimizing resource usage over time. Despite being completely different application areas, the relationship between packing and scheduling has been linked since as early as the late 1990s for both 1D [65] and 2D settings [110]: items that need to be packed can be viewed as activities to be scheduled. The literature on scheduling is diverse, so we focus on vehicle routing problems and batch scheduling, two areas related to the substructures of our problem. Here, we provide a brief intuition of their relations to our problem, and leave the details to Section 4.2.1.

3.4.1 Vehicle Routing Problem

Vehicle Routing Problems (VRP) optimize the routes of vehicles while some criteria associated with each route are satisfied. Packing problems and VRPs are similar: each vehicle can be interpreted as a bin with capacity corresponding to trip length or time. The distance or travel time between visits, then, are items to be packed. In this thesis, we explore this idea, applying techniques proposed for VRP to our packing problem.

MILP models of VRPs can be categorized into three main types: the three-indexed model, two-indexed model, and set cover model [34]. The three-indexed model treats the route of each vehicle as independent of other vehicles. This is particularly beneficial if the

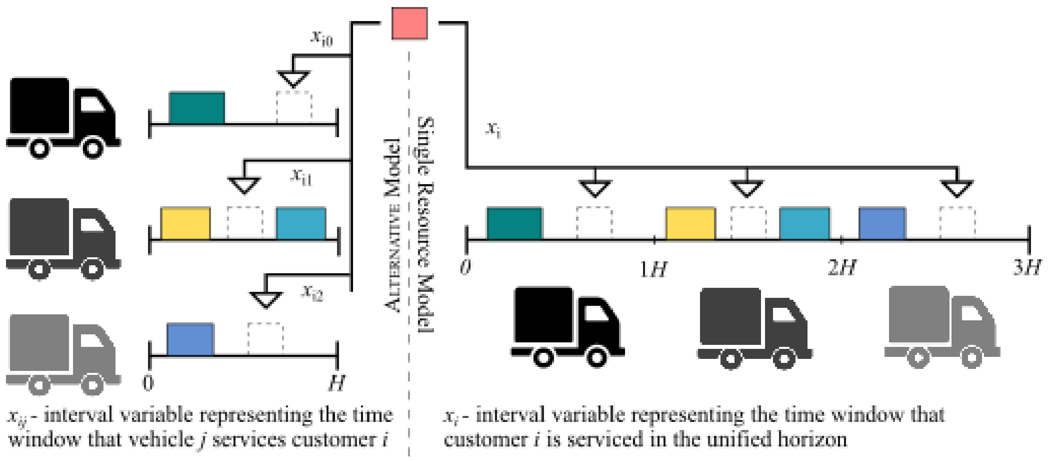

x_{i0}
x_{i1}
x_{i2}
x_i
0
H
1H
2H
3H
Alternative Model
Single Resource Model
x_{ij} - interval variable representing the time window that vehicle j services customer i
x_i - interval variable representing the time window that customer i is serviced in the unified horizon

Figure 3.12: Comparison between the representation of trip decisions in the Alternative CP model (left) and the single resource CP model (right). The coloured blocks portray the customers that a vehicle needs to service. The duration of service is represented by the block's horizontal size. The arrows represent the allocation decision of the pink block (customer), and the horizon below each arrowhead corresponds to the domain of the variable. The time horizon of each vehicle is from 0 to H.

vehicle types are heterogeneous [78, 95, 162]. Problems with homogeneous vehicles can be modelled using a two-indexed formulation, which merges the routes of different vehicles in the decision process [66]. Finally, the set cover models explicitly enumerate all possible routes and are generally solved using a branch-and-price algorithm [13, 67, 76].

A number of CP models for VRPs [44, 89, 151] represent the problem from a scheduling perspective, modelling the trip-to-vehicle assignments using interval variables and the Alternative constraint. Recently, for a capacity- and time-constrained routing problem, Booth and Beck [36] introduced the single resource model, where multiple resources are unified into a single resource on an expanded time horizon. Consequently, variables and constraints are declared over the single unified horizon instead of alternatively on each resource, which allows for a stronger inference. Their experiments demonstrate that this formulation is computationally advantageous over alternative CP constructs, especially for larger instances.

Instead of reproducing their proposed Alternative and single resource model, both of which contain other problem-specific requirements, we summarize the essentials in a simplified example: we would like to service a customer with one of three vehicles. The Alternative CP formulation would need three decision variables, each assigning the customer to one vehicle. In contrast, the single resource formulation would need only one variable representing the assignment of the customer onto the entire horizon (Figure 3.12). Similarly, to satisfy a limited vehicle capacity, the Alternative model would need three sets of constraints, each of which, for example, restricts a cumulative function describing the load of a vehicle; the single resource model would only need one set, as the constraints are applied over all vehicle routes simultaneously.

For more details on VRPs, we refer readers to surveys by Drexl [69] and Erdelić [74].

3.4.2 Batch Scheduling

Batch scheduling arises when a set of jobs with common characteristics need to be processed together. We observe that making guillotine cuts is very similar to scheduling batches: in both cases, some events are executed concurrently. Ham [107] proposed a 4-index MILP formulation that uses binary decision variables to match steps in jobs with batch positions on machines. Ham and Cakici [109] proposed a 3-index formulation that uses two sets of binary decision variables to match steps in jobs with batches and to distribute batches onto machines. They demonstrated that this formulation requires significantly fewer variables and constraints than the 4-index one. Liao and Liao [169] introduced a MILP model similar to the 3-index formulation for a two-machine flowshop problem, where the machines are batching machines. Cakici et al. [43] developed a time-indexed MILP model that decides on the time index at which a job starts and restricts the number of batches at each time index. Monma and Potts [196] constructed a disjunctive MILP model where variables represent job-to-job precedence.

Constraint programming formulations typically use interval variables and state functions to represent jobs and regulate their start times, respectively [108, 109, 232]. Notably, Ham and Cakici [109] found that this CP formulation drastically outperformed the 3-index and 4-index MILP formulations, finding the same quality solutions while using significantly less memory. Malapert et al. [185] took an alternative perspective to solve a parallel-batch machine problem and developed the global constraint, SequenceEDD, to describe the maximal lateness for a single batching machine. The performance of the CP model using this constraint, however, was inferior to a MILP model proposed for the same problem by Kosch and Beck [156].

For non-model-based approaches, we refer readers to review surveys by Potts and Kovalyov [213] and by Fowler and Mönch [83].

3.5 Summary

In this chapter, we presented foundations of mixed-integer linear programming (MILP) and constraint programming (CP) and reviewed relevant literature. We showed that a substantial number of exact and approximation algorithms have been proposed for 1D packing. For 2D packing, we provided a classification system with a focus on bin packing and cutting stock problems and contextualized our problem. We described the main MILP models to represent guillotine cuts and traced their origins to the 1D formulations. We noted a lack of CP-based studies on 2DBPP and 2DCSP with guillotine constraints; instead, we reviewed main CP modelling perspectives on the optimal rectangle packing problem. We also identified a shortfall of literature on two-dimensional packing problems with changeable item size and conflicts. Lastly, we examined relevant studies on vehicle routing and batch scheduling. Concluding this chapter, we remind the reader that the 2SCSP-FFMS contains several difficult packing problems, the intersection of which has not been studied in the literature.

Chapter 4

2SCSP with Flexible Item Length and Flexible Demand

In this chapter, we study the Two-Dimensional Two-stage Cutting Stock Problem with Flexible Length and Flexible Demand (2SCSP-FF), a reduced version of 2SCSP-FFMS that ignores marriageability constraints and removes scheduling costs. We first formalize our assumptions, before introducing both exact and heuristic algorithms. Our experiments show that the single resource CP formulation has an order-of-magnitude advantage over other exact approaches, and our multi-phase sequential heuristic yields the best solutions overall.

4.1 Problem Assumptions

Compared to 2SCSP-FFMS, the two-dimensional two-stage cutting stock problem with flexible length and demand, 2SCSP-FF, makes the following assumptions.

Trivial Order Marriageability All orders share identical order properties, so any combination of orders can be cut from the same stock. In other words, $M_{ii'} = 1$ for $i, i' \in N$.

Negligible Scheduling Cost The due dates of orders and the available dates of stocks are identical ($d_i = a_k$ for $i \in N, k \in K$), so the scheduling costs can be ignored.

Even after these assumptions, 2SCSP-FF is a difficult problem to solve, being comprised of NP-hard problems, such as the 2SCSP [86] and the generalized assignment problem with flexible jobs [215]. Thus, 2SCSP-FF is also NP-hard.

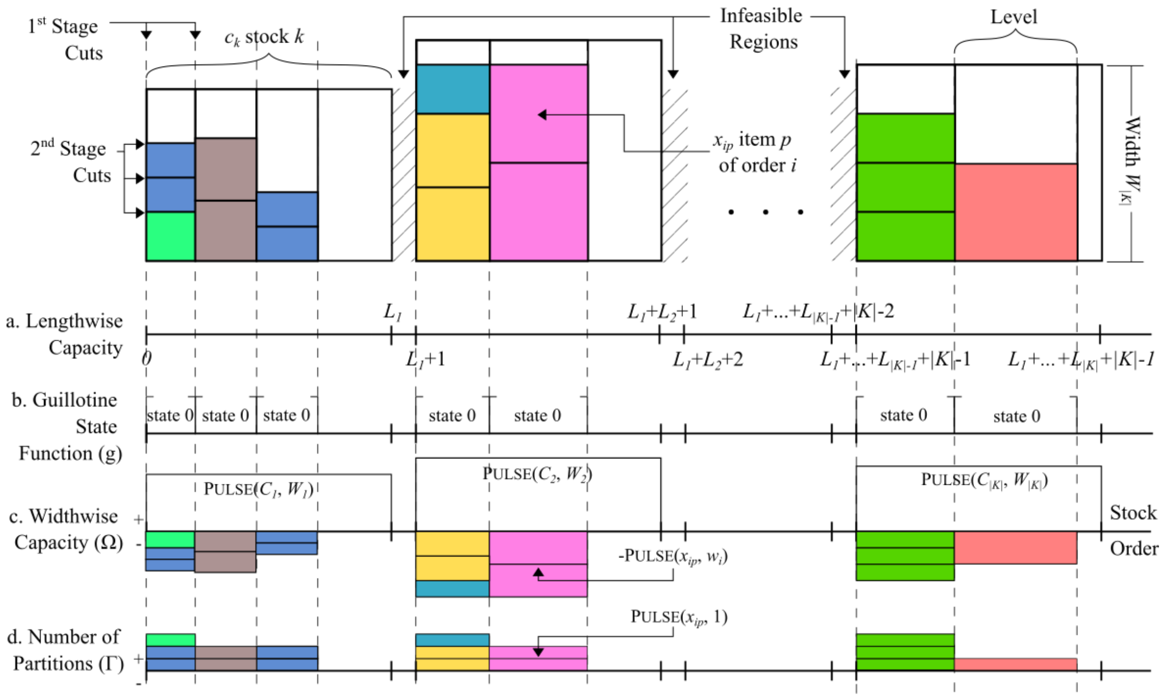


Figure 4.1: Illustration of the CP_{SR} model. The lengths of the stock rectangles are concatenated along the horizontal axis. Here, a level is a vertical strip. The smaller rectangles and the dashed lines represent items and guillotine cuts, respectively.

4.2 Exact Approaches

4.2.1 The Single Resource CP Formulation

First, we present a novel approach that exhibits significant representational efficiency: the Single Resource CP model, CP_{SR}. Our model poses the 2SCSP-FF as a scheduling problem composed of three main components: a unified domain of stock length, a state function for guillotine cuts, and cumulative functions tracking widthwise resources.

Unified Lengthwise Domain Our model adapts the single resource transformation [36], a CP modelling technique that unifies alternative resources into a single horizon, to the 2SCSP-FF. CP_{SR} concatenates the stock rectangles so that the total length of the stocks is analogous to a temporal horizon on which items belonging to all orders need to be allocated (Figure 4.1a). For each possible item $p \in C_i$ belonging to order i, we introduce an *optional interval variable* x_{ip}. The start time of x_{ip} represents an item's leftmost lengthwise coordinate, and the duration of x_{ip} its length, which we further restrict to be within $[\rho_i^{min}, \rho_i^{max}]$. For necessary items (i.e., in set A_i), we remove the absent value, $\{\perp\}$, from the domain of x_{ip} for all $p \in A_i$ and simply declare them as *interval variables*.

To avoid an item spanning multiple stocks in the unified horizon, we insert a dummy unit of forbidden space between adjacent stocks to create an infeasible region (Figure 4.1a, hatched). The horizon is thus augmented from $\sum_{k\in K} L_k$ to $\sum_{k\in K}(L_k+1)$, and items can only be placed in the feasible region $\bar{F} = \bigcup_{k\in K}\mathcal{F}_k$, where $\mathcal{F}_k = [\sum_{k'\in K|k'<k} L_{k'} + k - 1, \sum_{k'\in K|k'\leq k} L_{k'} + k - 1]$. We denote the augmented horizon as $\mathcal{H} = [0, \sum_{k\in K}(L_k+1)]$ and use ForbidExtent to restrict the domain of the item variables x_{ip} accordingly.

Guillotine State Function We draw inspiration from batch scheduling to model guillotine cuts: we treat each level in a stock rectangle as a batch so that the level's lengthwise endpoints coincide with the corresponding endpoints of the items. We first introduce a *state function*, g (Figure 4.1b). Then, we associate the items with a level using an AlwaysConstant constraint, which coerces their interval variables to align with an interval in g. Thus, items can only be on the same level if they belong to the same interval in the state function.

Cumulative Resource Function Expressions The width of stocks and a level's partition count limit are interpreted as widthwise resources. Typical of CP scheduling, we use *cumulative functions* and *pulses*. We let Ω, a cumulative function, be the net widthwise capacity over the horizon. In Ω, we generate a pulse with magnitude W_k for each stock rectangle k and a pulse with magnitude $-w_i$ for every item belonging to order i (Figure 4.1c). As long as Ω is non-negative, the widthwise capacity is satisfied. A similar construct is used to express the limit on the number of partitions on each level (Figure 4.1d), where a positive pulse with unit magnitude is generated for each item. We constrain the total cumulative function of these unit pulses, Γ, to be within η.

Overall, our decision variables are as follows:

$x_{ip} :=$ (interval) lengthwise interval of item p belonging to order i.
$c_k :=$ (interval) lengthwise interval representing stock k.
$g :=$ (state function) guillotine state function.

CP_{SR} is defined in Model 4.1. Objective (4.1a) describes our cost, the weighted difference between the areas of stocks used and orders fulfilled. Expressions PresenceOf and SizeOf are used to access the presence and the duration of an interval variable. Constraints (4.1b) and (4.1c) define the widthwise usage of each stock. The last two parameters in the AlwaysIn constraint respectively dictate the minimum and maximum values that the cumulative function Ω can take on over the horizon $\mathcal{H}$. Constraints (4.1d) and (4.1e) define the restriction on the number of partitions on each level. We remark that Constraint (4.1d) can be substituted by $\Gamma = \sum_{k\in K}\text{Pulse}(\mathcal{H},\eta) - \sum_{i\in N, p\in C_i}\text{Pulse}(x_{ip},1)$, which yields identical semantics overall, but uses more terms on the right-hand side. Constraint (4.1f) defines the guillotine cut restrictions. The last two parameters in AlwaysConstant ensure that the start and end times of the variables x_{ip} are aligned with those of the intervals within the state function g. Constraints (4.1g) and (4.1h) ensure that the total quantity of the order fulfilled is within the demand tolerance. Constraint (4.1i) ensures that no item is assigned across two stock rectangles. The remaining constraints declare the decision variables.

$$
\begin{aligned}
\min\ & \alpha \sum_{k \in K} L_k W_k \textsc{PresenceOf}(c_k) && (CP_{SR}) && (4.1a)\\
& -\beta \sum_{i \in N} \sum_{p \in C_i} w_i \textsc{PresenceOf}(x_{ip}) \textsc{SizeOf}(x_{ip}) && && \\
\text{s.t.}\ & \Omega = \sum_{k \in K} \textsc{Pulse}(c_k, W_k) - \sum_{i \in N} \sum_{p \in C_i} \textsc{Pulse}(x_{ip}, w_i) && && (4.1b)\\
& \textsc{AlwaysIn}(\Omega, \mathcal{H}, 0, \max_{k \in K} W_k) && && (4.1c)\\
& \Gamma = \sum_{i \in N} \sum_{p \in C_i} \textsc{Pulse}(x_{ip}, 1) && && (4.1d)\\
& \textsc{AlwaysIn}(\Gamma, \mathcal{H}, 0, \eta) && && (4.1e)\\
& \textsc{AlwaysConstant}(g, x_{ip}, True, True) && \forall i \in N, p \in C_i && (4.1f)\\
& \sum_{p \in C_i} \textsc{SizeOf}(x_{ip}) \geq q_i^{min} / w_i && \forall i \in N && (4.1g)\\
& \sum_{p \in C_i} \textsc{SizeOf}(x_{ip}) \leq q_i^{max} / w_i && \forall i \in N && (4.1h)\\
& \textsc{ForbidExtent}(x_{ip}, \bar{F}) && \forall i \in N, p \in C_i && (4.1i)\\
& x_{ip} : \textsc{IntervalVar}(\mathcal{H}, [\rho_i^{min}, \rho_i^{max}]) && \forall i \in N, p \in A_i && (4.1j)\\
& x_{ip} : \textsc{OptIntervalVar}(\mathcal{H}, [\rho_i^{min}, \rho_i^{max}]) && \forall i \in N, p \in B_i && (4.1k)\\
& c_k : \textsc{IntervalVar}(\mathcal{F}_k, L_k) && \forall k \in K && (4.1l)\\
& g : \textsc{StateFunction}() && && (4.1m)
\end{aligned}
$$

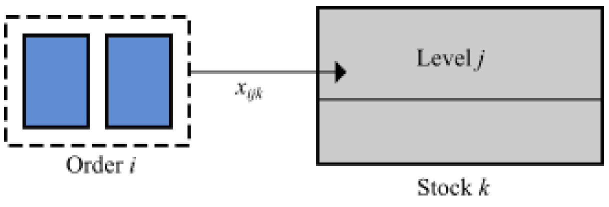
4.2.2 Integer-based CP Formulations

Next, we present three integer-based CP formulations motivated by different modelling perspectives summarized in Figure 4.2.

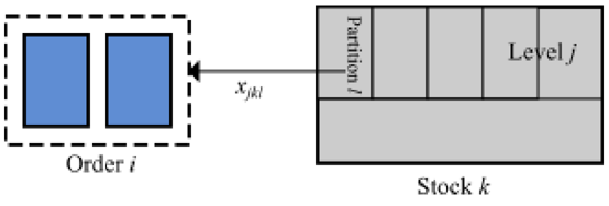
4.2.2.1 Counting-based CP Model

Due to the two-stage cuts, partitions assigned to the same order on a level must be identical; hence, we can count them. For a given level j from stock k, we use an integer variable x_{ijk} in our counting-based model, CP_{CO}, to denote the number of partitions assigned to order i (Figure 4.2a). We use an integer variable y_{jk} to represent the length of level j on stock k. As the position of the lengthwise cut is not restricted to be integral, we magnify the domain using a precision parameter $\mathcal{P}$ equal to some power of 10, so that y_{jk} represents the first $\log_{10} \mathcal{P}$ decimal places of the actual length. More formally, our decision variables are as follows:

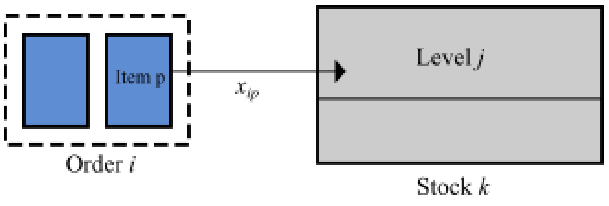
$x_{ijk} :=$ (integer) # of partitions on level j of stock k assigned to order i



(a) Counting-based CP model



(b) Stock-based CP model



(c) Item-based CP model

Figure 4.2: A visualization of the integer-based CP models.

$y_{jk} :=$ (integer) length of level j of stock k magnified by $\mathcal{P}$

$$\min \;\alpha \sum_{k\in K} L_k W_k c_k - \beta \sum_{i\in N}\sum_{j\in J_k}\sum_{k\in K} w_i x_{ijk} y_{jk}/\mathcal{P} \qquad (CP_{CO}) \tag{4.2a}$$

$$\text{s.t.} \;\sum_{i\in N} x_{ijk} w_i \leq W_k s_{jk} \qquad \forall j\in J_k, k\in K \tag{4.2b}$$

$$y_{jk}/\mathcal{P} \leq \rho_i^{max} + (x_{ijk} == 0)\max_{n\in N}(\rho_n^{max}) \qquad \forall i\in N, j\in J_k, k\in K \tag{4.2c}$$

$$y_{jk}/\mathcal{P} \geq \rho_i^{min}(x_{ijk} \geq 1) \qquad \forall i\in N, j\in J_k, k\in K \tag{4.2d}$$

$$\sum_{j\in J_k}\sum_{k\in K} y_{jk} x_{ijk}/\mathcal{P} \geq q_i^{min}/w_i \qquad \forall i\in N \tag{4.2e}$$

$$\sum_{j\in J_k}\sum_{k\in K} y_{jk} x_{ijk}/\mathcal{P} \leq q_i^{max}/w_i \qquad \forall i\in N \tag{4.2f}$$

$$\sum_{j\in J_k} y_{jk}/\mathcal{P} \leq L_k c_k \qquad \forall k\in K \tag{4.2g}$$

$$s_{jk} = \textsc{Any}([x_{ijk} > 0, \forall i\in N]) \qquad \forall j\in J_k, k\in K \tag{4.2h}$$

$$c_k = \textsc{Any}([s_{jk} = 1, \forall j\in J_k]) \qquad \forall k\in K \tag{4.2i}$$

$$x_{ijk} \in \{0, ..., \eta\} \qquad \forall i\in N, j\in J_k, k\in K \tag{4.2j}$$

$$y_{jk} \in \{0, \min_{i\in N}\rho_i^{min}\mathcal{P}, \ldots, \max_{i\in N}\rho_i^{max}\mathcal{P}\} \qquad \forall j\in J_k, k\in K \tag{4.2k}$$

Model 4.15 formalizes CP_{CO}. Objective (4.2a) describes the cost. Since y_{jk} is magnified, we divide it by $\mathcal{P}$ to recover its actual length. Constraint (4.2b) restricts the width of the stocks. Constraints (4.2c) and (4.2d) restrict the length of a level by the tightest interval determined by the allotted orders. Constraints (4.2e) and (4.2f) ensure that partitions of each order fulfilled satisfy the total quantity range demanded. Constraint (4.2g) restricts the length of the stocks in use. Constraints (4.2h) and (4.2i) describe if a level and a stock is used, respectively.

4.2.2.2 Stock-based CP Model

The stock-based CP model, CP_{ST}, takes advantage of the limited number of possible partitions on a level, matching each partition to some order (Figure 4.2b). Specifically, we define integer variables x_{jkl} representing the index of the order to which the l^{th} partition of the j^{th} level on the k^{th} stock is assigned. As not all partitions are always needed, we define a dummy order that serves as a placeholder. Formally, the dummy order, indexed by $D = |N| + 1$, has width $w_D = 0$ and length interval $[\rho_D^{min}, \rho_D^{max}] = [0, \max_{i \in N}(\rho_i^{max})]$. We use $\overline{N} = N \bigcup \{D\}$ to denote the set of original orders plus the dummy order; $\overline{w}$ to denote the set of widths of original orders union the dummy width w_D; $\overline{\rho^{min}}$ and $\overline{\rho^{max}}$ to denote the lengthwise bounds of orders union the dummy bounds ρ_D^{min} and ρ_D^{max}. Similar to CP_{CO}, we let y_{jk} be the length of level j on stock k and magnify its domain using $\mathcal{P}$.

$$
\begin{aligned}
\text{minimize} \quad & \sum_{k \in K} L_k W_k c_k - \sum_{j \in J_k} \sum_{k \in K} \sum_{l \in P} \overline{w}_{x_{jkl}} y_{jk}/\mathcal{P} && (CP_{ST}) && (4.3a)\\
\text{s.t.} \quad & \sum_{l \in P} \overline{w}_{x_{jkl}} \le W_k s_{jk} && \forall j \in J_k, k \in K && (4.3b)\\
& y_{jk}/\mathcal{P} \le \overline{\rho^{max}}_{x_{jkl}} && \forall j \in J_k, k \in K, l \in P && (4.3c)\\
& y_{jk}/\mathcal{P} \ge \overline{\rho^{min}}_{x_{jkl}} && \forall j \in J_k, k \in K, l \in P && (4.3d)\\
& \sum_{j \in J_k} \sum_{k \in K} \sum_{l \in P} (x_{jkl} == i) y_{jk}/\mathcal{P} \ge q_i^{min}/w_i && \forall i \in N && (4.3e)\\
& \sum_{j \in J_k} \sum_{k \in K} \sum_{l \in P} (x_{jkl} == i) y_{jk}/\mathcal{P} \le q_i^{max}/w_i && \forall i \in N && (4.3f)\\
& s_{jk} = \textsc{Any}([x_{jkl} \ne D, \forall l \in P]) && \forall j \in J_k, k \in K && (4.3g)\\
& x_{jkl} \in \overline{N} && \forall j \in J_k, k \in K, l \in P && (4.3h)\\
& (4.2g), (4.2i), (4.2k)
\end{aligned}
$$

Model 4.3 formalizes CP_{ST}. Objective (4.3a) minimizes the cost. Constraint (4.3b) ensures that the widthwise capacity is satisfied on each stock. In particular, $\overline{w}$ is indexed by x_{jkl} using the ELEMENT constraint. Constraints (4.3c) and (4.3d) constrain the length of a level by the items assigned on it. If x_{jkl} takes on the dummy value, these constraints are trivially satisfied. Constraints (4.3e) and (4.3f) satisfy the total area of each order. Constraint (4.3g) instantiates intermediate parameters indicating level usage.

4.2.2.3 Item-based CP Model

Our final integer-based CP model, CP_{IT} , takes advantage of PACK, a global constraint for the 1D packing substructure. In 2SCSP-FF, we observe two such substructures: the widthwise packing of items into levels, and the lengthwise packing of levels into stock rectangles. While the former can be represented by PACK, the latter cannot due to the flexibility in length.

To use PACK for the first substructure, we propose a new set of indices for levels. First, we let $\overline{J} = \{1, \ldots, \sum_{k \in K} |J_k|\}$ be the flattened set of the maximally possible levels over all available stocks. Then, for every possible item p belonging to order i, we introduce a set of integer variables x_{ip} having domain $\overline{J}$ (Figure 4.2c). However, in a feasible assignment, not all possible items need to be allotted to a level in the available stocks, so we expand the domain of x_{ip} to include a dummy level with infinite width indexed $D = |\overline{J}| + 1$ to absorb any unneeded items. Accordingly, we introduce integer variables Ω_{jk} to represent the width usage of level j on stock k and an additional integer variable Ω_D for the dummy level. The length of level j on stock k is again represented by an integer variable y_{jk}.

$$
\begin{aligned}
\min \quad & \alpha \sum_{k \in K} c_k W_k L - \beta \sum_{i \in N} \sum_{p \in C_i} w_i \widetilde{y}^{min}_{x_{ip}} / \mathcal{P} && (CP_{IT}) && (4.4a)\\
\text{s.t.} \quad & \textsc{Pack}(\widetilde{\Omega}, x, w) && && (4.4b)\\
& \widetilde{y}^{max}_{x_{ip}} / \mathcal{P} \le \rho^{max}_i && \forall i \in N, p \in C_i && (4.4c)\\
& \widetilde{y}^{min}_{x_{ip}} / \mathcal{P} \ge \rho^{min}_i && \forall i \in N, p \in C_i && (4.4d)\\
& \sum_{p \in C_i} \widetilde{y}^{min}_{x_{ip}} / \mathcal{P} \le q^{max}_i / w_i && \forall i \in N && (4.4e)\\
& \sum_{p \in C_i} \widetilde{y}^{max}_{x_{ip}} / \mathcal{P} \ge q^{min}_i / w_i && \forall i \in N && (4.4f)\\
& \textsc{Count}(x, j) \le \eta && \forall j \in \tilde{J} && (4.4g)\\
& s_{jk} = \textsc{Any}([x_{ip} == j + k|J|, \forall i \in N, p \in C_i]) && \forall j \in J_k, k \in K && (4.4h)\\
& x_{ip} \in \overline{J} \cup \{D\} && \forall i \in N, p \in C_i && (4.4i)\\
& \Omega_{jk} \in \{0, ..., W_k\} && \forall j \in J_k, k \in K && (4.4j)\\
& \Omega_D \in \{0, ..., \sum_{i \in N} w_i |C_i|\} && \forall j \in J_k, k \in K && (4.4k)\\
& (4.2g), (4.2i), (4.2k)
\end{aligned}
$$

Model 4.4 formalizes CP_{IT}. Constraint (4.4b) makes item-to-level assignments. Here, variables x_{ip} and parameters w_i are reshaped into two one-dimensional vectors x and w of size $\sum_{i\in N}|C_i|$ so that the i'^{th} element in x has width of $w_{i'}$. We also use $\widetilde{\Omega_D} = \{\Omega_{jk} \mid \forall j \in J_k, k \in K\} \bigcup \{\Omega_D\}$ to denote the flattened set of stock width variables. Constraints (4.4c) and (4.4d) restrict the level's length by the orders allotted on it. In associating the item assignment x_{ip} with y_{jk}, we define notations $\widetilde{y}^{max} = \overline{y} \bigcup \{0\}$ and $\widetilde{y}^{min} = \overline{y} \bigcup \{\max_i \rho_i^{max}\}$, where $\overline{y} = \{y_{jk},\ \forall j \in J_k, k \in K\}$. The last entry in $\widetilde{y}^{min}$ and in $\widetilde{y}^{max}$ corresponds to the dummy index and is important. If x_{ip} takes on the dummy value D, constraints (4.4c) and (4.4d) evaluate to expressions $0 \le \rho_i^{max}$ and $\max_i \rho_i^{max} \ge \rho_i^{min}$, respectively, both of which are always true. If x_{ip} does not take on the dummy value, then the corresponding level length is restricted. Also, for conciseness, we abbreviate the global constraint Element(A, e) as A_e. Constraints (4.4e) and (4.4f) ensure that the quantity fulfilled satisfies the quantity demanded. Here, we always use $\widetilde{y}^{max}$ so that no contribution is made if x_{ip} takes on the dummy value. Constraint (4.4g) enforces a maximum of η items on any level. Constraint (4.4h) defines if a level is used. The rest define the variables.

4.2.2.4 Symmetry-breaking

The problem has a number of inherent symmetries due to the homogeneous items, levels, and stock rectangles. Hence, we augment CP_{CO}, CP_{ST}, and CP_{IT} with the following symmetry-breaking constraints:

$$y_{jk} \ge y_{(j+1)k} \quad \forall j \in J'_k, k \in K \tag{4.5a}$$

$$c_k \ge c_{k+1} \quad \forall k \in K'_h, h \in H \tag{4.5b}$$

These constraints break the symmetry between the lengths of consecutive levels on the same stock and the presence of homogeneous stocks, respectively. We use a prime to indicate an ordered set without its last element: $J' = J \setminus \{|J|\}$. For CP_{ST}, we also specify a lexicographic ordering of the order indices on consecutive levels of the same stock via Lexicographic($[x_{jkl}, \forall l \in P], [x_{(j+1)kl}, \forall l \in P]$). For CP_{IT}, we add an additional constraint to break the symmetry for items belonging to the same order: $x_{ip} \le x_{i(p+1)}, \forall i \in N, p \in C'_i$.

4.2.3 MILP Formulations

We also introduce two mixed-integer programs that use binary variables to assign partitions on each level to orders. Formulating a strong MILP model is challenging, as determining the area of each order requires information related to two independent decisions: the order-to-level assignment and the level length given order assignments. We linearize this relationship at the expense of introducing new variables, each one packing an item of an order into a level of a stock. While it is tempting to decompose them into independent orders-to-levels and levels-to-stocks decisions similar to the compact formulation in Furini and Malaguti [86], representing both the area of each order and the variable length of each level using linear constraints is nontrivial. Here, we do not investigate this further.

4.2.3.1 Assignment-based MILP Model

The assignment-based MILP model, MIP_{AS}, explicitly enumerates all possible pieces on each level of the stock that can be assigned to each order. Binary variable x_{ijkl} decides if the l^{th} piece of the j^{th} level of the k^{th} stock is assigned to i^{th} order. Continuous decision variables y_{jk} describe the varying length for each j^{th} level of the k^{th} stock. The resulting quantities of each l^{th} piece on the j^{th} level of the k^{th} stock assigned to each i^{th} order are denoted by continuous variables a_{ijkl}, which linearize the product representing the weight of each piece of stock between continuous variable y_{jk} and binary variable x_{ijkl}. Binary variables c_k indicate whether the k^{th} stock is used or not.

MIP_{AS} is defined in Model 4.6. Objective (4.6a) describes the cost. Constraint (4.6b) restricts the stocks' width. Constraint (4.6c) limits the number of lengthwise cuts. Constraint (4.6d) ensures that the lengthwise capacity of each stock is satisfied. Constraints (4.6e) and (4.6f) assert that the level's length must respect the minimum and maximum length of items assigned to it. Constraints (4.6g) and (4.6h) ensure that the quantity of each order assigned across all stocks is satisfactory. Constraints (4.6i), (4.6j), and (4.6k) define the area of each partition on a level.

$$
\begin{aligned}
\min \quad & \alpha \sum_{k \in K} L_k W_k c_k - \beta \sum_{i \in N, j \in J_k, k \in K, l \in P_i} a_{ijkl} && (MIP_{AS}) && \text{(4.6a)}\\
\text{s.t.} \quad & \sum_{i \in N, l \in P_i} w_i x_{ijkl} \le W_k c_k && \forall j \in J_k, k \in K && \text{(4.6b)}\\
& \sum_{i \in N, l \in P_i} x_{ijkl} \le \eta c_k && \forall j \in J_k, k \in K && \text{(4.6c)}\\
& \sum_{j \in J_k} y_{jk} \le L_k c_k && \forall k \in K && \text{(4.6d)}\\
& y_{jk} \ge \rho_i^{min} x_{ijkl} && \forall i \in N, j \in J_k, k \in K, l \in P_i && \text{(4.6e)}\\
& y_{jk} \le \rho_i^{max} x_{ijkl} + \max_{i' \in N}(\rho_{i'}^{max})(1 - x_{ijkl}) && \forall i \in N, j \in J_k, k \in K, l \in P_i && \text{(4.6f)}\\
& \sum_{l \in P_i, j \in J_k, k \in K} a_{ijkl} \ge q_i^{min} && \forall i \in N && \text{(4.6g)}\\
& \sum_{l \in P_i, j \in J_k, k \in K} a_{ijkl} \le q_i^{max} && \forall i \in N && \text{(4.6h)}\\
& a_{ijkl} \le w_i y_{jk} && \forall i \in N, j \in J_k, k \in K, l \in P_i && \text{(4.6i)}\\
& a_{ijkl} \ge w_i y_{jk} - \rho_i^{max} w_i (1 - x_{ijkl}) && \forall i \in N, j \in J_k, k \in K, l \in P_i && \text{(4.6j)}\\
& a_{ijkl} \le \rho_i^{max} w_i x_{ijkl} && \forall i \in N, j \in J_k, k \in K, l \in P_i && \text{(4.6k)}\\
& x_{ijkl} \in \{0,1\} && \forall i \in N, j \in J_k, k \in K, l \in P_i && \text{(4.6l)}\\
& y_{jk} \in \mathbb{R}^+ && \forall j \in J_k, k \in K && \text{(4.6m)}\\
& a_{ijkl} \in \mathbb{R}^+ && \forall i \in N, j \in J_k, k \in K, l \in P_i && \text{(4.6n)}\\
& c_k \in \{0,1\} && \forall k \in K && \text{(4.6o)}
\end{aligned}
$$

Symmetry-breaking: We can again add symmetry-breaking constraints (4.5a) and (4.5b) to *MIP* similar to CP_{CO} and CP_{ST}. Furthermore, we add constraints (4.7a) and (4.7b) to break the symmetry between partitions on the same level belonging to the same order and the length of the first level of identical stocks, respectively.

$$
\begin{aligned}
& x_{ijkl} \ge x_{ijk(l+1)} && \forall i \in N, j \in J, k \in K, l \in P_i' && \text{(4.7a)}\\
& y_{0k} \ge y_{0(k+1)} && \forall k \in K_h', h \in H && \text{(4.7b)}
\end{aligned}
$$

4.2.3.2 Counting-based MILP Model

In order to retain linearity, MIP_{AS} treats the assignment of partitions on the same level to an order as individual decisions. Alternatively, we can count them, leading to a counting-based MILP model denoted as MIP_{CO}. We introduce binary variable x_{ijkl} that takes the value 1 if and only if there are l partitions (with identical dimensions) on level j of stock k assigned to order i. Differing from Section 4.2.3.1 which uses a_{ijkl} to describe the area of partition l, we let the continuous variable a_{ijkl} represent the total area of all l counts of partitions on level j of stock k. Again, we use a continuous variable y_{jk} to describe the length of the j^{th} level on the k^{th} stock.

Constraint (4.8b) ensures that each order can only take on one range of cumulative quantity on each level. In addition, we take advantage of the special orders sets inherent in MILP solvers. Constraint (4.8c) enforces the total width of all items assigned to each stock to be less than the stock width if that stock is used, or zero if it is unused. Constraint (4.8d) ensures the number of lengthwise cuts is within limits. Constraint (4.8e), (4.8f), and (4.8g) defines the total quantity of partitions of each order.

$$
\begin{array}{llll}
\text{minimize} & (4.6a) & (\mathbb{MIP}_{order}) & (4.8a) \\
\text{s.t.} & SOS1(x_{ijkl}, \forall l \in P_i) & \forall i \in N, j \in J_k, k \in K & (4.8b) \\
& \displaystyle\sum_{i \in N} \sum_{l \in P_i} w_i (l x_{ijkl}) \leq W_k c_k & \forall j \in J_k, k \in K & (4.8c) \\
& \displaystyle\sum_{i \in N} \sum_{l \in P_i} l x_{ijkl} \leq \eta & \forall j \in J_k, k \in K & (4.8d) \\
& a_{ijkl} \leq l w_i y_{jk} & \forall i \in N & (4.8e) \\
& a_{ijkl} \geq l w_i y_{jk} - l \rho_i^{max} w_i (1 - x_{ijkl}) & \forall i \in N & (4.8f) \\
& a_{ijkl} \leq l \max\limits_{i' \in N} (\rho_{i'}^{max}) w_i x_{ijkl} & \forall i \in N, j \in J_k, k \in K, l \in P_i & (4.8g) \\
& (4.6d) - (4.6h), (4.6l) - (4.6o) & &
\end{array}
$$

4.2.4 Model Comparison

Amongst the models proposed are two common abstract substructures that manifest themselves differently due to modelling perspectives. The first substructure is a 1D packing problem that assigns items of the orders to levels without exceeding the levels' width, which is inherited from its respective stock. The second substructure again is a 1D packing problem, fitting levels into stocks lengthwise; however, differing from a classic packing problem, the level length is flexible. These substructures are linked by two types of constraints: the orders that are assigned to the level restricts the length of each level, and the total quantity fulfilled for each order is derived from the multiplicative relationship between the presence of the item and the level length.

All CP models, except CP_{SR}, share almost identical constraints and variable definitions for the second substructure, in particular, using a decision variable y_{jk} for the j^{th} level on the k^{th} stock; where these models differ is in the modelling of the first substructure.

CP_{ST} takes advantage of the maximum number of lengthwise cuts by using them as the indices for decision variables. So, its model size is largely dependent on the number of partitions on the stocks, and its search space is unaffected by any changes in the number of items. In contrast, with an increase in the number of items, the search space for CP_{CO} increases, but its size does not change, as neither new variables nor new constraints need to be introduced. The CP_{IT} model leverages the global constraint Pack at the expense of explicitly instantiating all possible partitions for all orders, making the model size highly sensitive to the number of items and the number of orders.

The central difference between CP_{SR} and the other exact models is that the former uses interval variables to describe the item assignments, and the latter uses integer variables. For the first substructure, CP_{SR} models the width of each stock as a cumulative resource for the items to deplete, whereas all other models employ arithmetic and logical constraints. Also, CP_{SR} models the second substructure by framing stock length as time in a scheduling problem. Typically, a continuous time horizon implies that the stocks are joined into one resource, but packing problems require stocks to be discrete. The variables from integer-based CP models, as a result, use an index dedicated to each stock. For instance, y_{jk} is described by the index k. Instead, CP_{SR} adds forbidden regions between stocks representing the discretization without adding indices to variables. Furthermore, by taking advantage of the state functions, CP_{SR} is the only model that does not explicitly enumerate all possible levels in stocks to represent guillotine cuts. This implicit representation, as well as the use of different CP substructures, significantly reduces the number of variables and constraints declared.

The MILP models differ mostly in their interpretation of the variable index l. In MIP_{AS}, decisions related to partition l is independent of any other partitions on the same level. MIP_{CO}, however, groups the decisions of the l counts of partitions together. This subtlety results in differing definitions for area variables and related constraints.

4.3 A First-fit based Heuristic

In addition to the exact approaches, we develop a two-phase first-fit-based heuristic, *FFMH*. An extension of the Finite First-fit Heuristic [27], the first phase sorts the orders' items in a lexicographically decreasing order based on their width and length interval size and packs each one into a level. The intuition is that orders with less lengthwise flexibility and larger width should be packed into a level first, as they can be more difficult to pack into a partial solution. A new level or stock is opened if an item cannot fit into the previous level or stock. Packing an item into a stock's level only narrows its length interval: another decision is required to obtain its exact length and thereafter each order's total area. For simplicity, we pack items of an order until the sum of the average possible area of each item is not less than the middle of the required area interval for that order. In the second phase, given the complete item-to-level assignment, we solve a linear program (Model 4.9) to determine each level's length, while minimizing cost.

$$\max \sum_{i\in N, j\in J_k, k\in K} w_i \Phi_{ijk} y_{jk} \tag{4.9a}$$

$$\text{s.t.} \quad \sum_{j\in J_k} y_{jk} \le L_k \qquad \forall k \in K \tag{4.9b}$$

$$y_{jk} \ge \Phi^{ind}_{ijk} \rho^{min}_i \qquad \forall i \in N, j \in J_k, k \in K \tag{4.9c}$$

$$y_{jk} \le \Phi^{ind}_{ijk} \rho^{max}_i + (1 - \Phi^{ind}_{ijk}) \max_{i'\in N}(\rho^{max}_{i'}) \forall i \in N, j \in J_k, k \in K \tag{4.9d}$$

$$\sum_{j\in J_k, k\in K} w_i \Phi_{ijk} y_{jk} \ge q^{min}_i \qquad \forall i \in N \tag{4.9e}$$

$$\sum_{j\in J_k, k\in K} w_i \Phi_{ijk} y_{jk} \le q^{max}_i \qquad \forall i \in N \tag{4.9f}$$

$$y_{jk} \in \mathbb{R}^+ \qquad \forall j \in J_k, k \in K \tag{4.9g}$$

The only variables in Model 4.9 are the continuous variables y_{jk} describing the length of the j^{th} level on the k^{th} stock. The parameter Φ_{ijk} is the number of partitions on the j^{th} level of the k^{th} stock that belongs to order i, and the parameter Φ^{ind}_{ijk} is a 0-1 indicator for $\Phi_{ijk} > 0$. We simplify the cost minimization objective to maximize total fulfillment (4.9a) because the number of stocks used is fixed given the item-to-level assignment. Constraint (4.9b) constrains the stock length. Constraints (4.9c) and (4.9d) satisfy the length specifications of the partitions. Constraints (4.9e) and (4.9f) ensure the total fulfillment of each order to be within tolerance limits.

4.4 A Sequential Heuristic Framework

In *FFMH*, items of orders are naively assigned to stocks; here, we attempt to develop a more sophisticated heuristic that solves components of the problem sequentially (Figure 4.3). The sequential heuristic first identifies subsets of orders with overlapping length intervals. Then, for each of these subsets, it generates a set of widthwise patterns. Finally, these patterns are packed into stocks, and their length values determined. These steps are repeated until termination, and, for each of these steps, we attempt several approaches, summarized in Table 4.1. We call a combination of these approaches a *configuration* of the heuristic and denote a configuration comprised of approaches $S1$, $S2$, and $S3$ as $S : S1 + S2 + S3$.

Before detailing the heuristic, we first define two terms used throughout this section.

Definition 1 (Congruent Orders). *Two orders are congruent if and only if the length intervals of these orders overlap.*

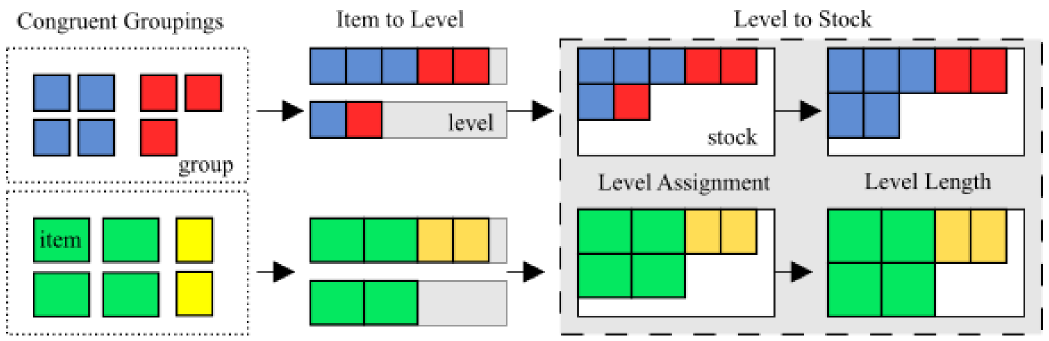


Figure 4.3: A three step outline of the sequential heuristic.

Step	Task	Type	Approach	
S1	Finding Congruent Groups	Constrained Clustering	CC^{S1}	
		CP	CP^{S1}	
S2	Item-to-level Assignments	CP	CP^{S2}	
		MILP	MIP^{S2},	
S3	Level-to-Stock Assignment	CP	CP^{S3}	
		MILP	Simple Formulation	MIP^{S3}_{simp}
			Compact Formulation	MIP^{S3}_{comp}
		BD of MIP^{S3}_{simp}	From Length LB	$BD^{S3,lb}_{simp}$
			From Length UB	$BD^{S3,ub}_{simp}$
		BD of MIP^{S3}_{comp}	From Length LB	$BD^{S3,lb}_{comp}$
			From Length UB	$BD^{S3,ub}_{comp}$

Table 4.1: Approaches used in the sequential heuristics at each step. BD, LB, and UB abbreviates Benders Decomposition, lower bound, and upper bound, respectively.

Definition 2 (Congruent Group)**.** *A set of orders that are all congruent with each other form a congruent group.*

Finding congruent groups is critical in reaching a feasible solution, as all orders that are on the same level form a congruent group.

Algorithm 1 outlines the a generic configuration of the sequential heuristic.

Algorithm 1: Sequential Heuristics Framework

```
Input  : Set of orders $N$, Cannot link constraints $\mathcal{C}$
Output: Set of level patterns $bestPattern$
$iter \leftarrow 0$ ;
$k \leftarrow$ GetHoffmanLB($N, C$);
while within time limit and $k \leq |N|$ do
    $groups \leftarrow$ FindCongruentGroups ($N$, $\mathcal{C}$, $k$) ;
    $levels \leftarrow$ DivideGroupsIntoLevels ($groups$) ;
    $assignments \leftarrow$ AssignLevelsToStocks ($levels$) ;
    if $assignments$ has lower objective than $bestPattern$ then
        $bestPattern \leftarrow assignments$
    end
    $iter \leftarrow iter + 1$ ;
    if $iter$ mod $|N| == |N| - 1$ then
        $k \leftarrow k + 1$ ;
    end
end
```

We allocate the runtime of each step in the heuristic as follows. `FindCongruentGroups` is allowed one-tenth of the overall runtime. With h seconds remaining, for each congruent group $g \in G$, the second stage algorithm takes up the maximum of $h/(|G|+1)$ seconds or the time it takes to find a feasible solution if less, leaving the rest for the third stage. One issue is that, especially for larger instances, instantiating the model can take a noticeable amount of time, so the actual clock-time may exceed the initially allotted runtime.

4.4.1 Finding Congruent Groups

The first step finds subsets of orders whose items can be placed on the same level. Restricting same-level assignments is a set of cannot link constraints $\mathcal{C}$ derived from non-overlapping length intervals. Formally, the relation (i, i') of orders i and i' is in the set $\mathcal{C}$ if and only if $[\rho_i^{min}, \rho_i^{max}]$ does not overlap with $[\rho_{i'}^{min}, \rho_{i'}^{max}]$. Here, we attempt two approaches: a constrained K-means algorithm and a constraint program.

4.4.1.1 Finding A Lower Bound on the Number of Clusters

Determining the initial number of clusters is essential: because of the cannot link constraints, starting off with too few clusters will lead to infeasibility. We initialize the number of clusters with a lower bound. First, we note that finding a feasible solution to the cannot-link constraints is equivalent to solving a vertex colouring problem, where each vertex is an order, each edge a cannot-link relation, and each colour a cluster. In other words, the minimum number of clusters is simply the chromatic number. One simple lower bound on the chromatic number is the Hoffman's bound [118], which states that:

$$\chi(G) \geq 1 - \frac{\lambda_1(G)}{\lambda_n(G)} \tag{4.10}$$

where $\chi(G)$ denotes the chromatic number, and $\lambda_1(G)$ and $\lambda_n(G)$ is the largest and smallest eigenvalues of the adjacency matrix of a graph G. Here, G is the conflict graph defined by $\mathcal{C}$, where (i, i') is an edge in G if and only if $(i, i') \in \mathcal{C}$. We use the function signature `GetHoffmanLB` to return $\chi(G)$. Finally, we remark that a solution with the number of groups being the chromatic number may not always lead to minimum waste, as the combinatorics of packing is not captured in this step.

4.4.1.2 Constrained K-Means

We consider using a popular constrained clustering algorithm to find congruent orders: the Constrained K-means [243], denoted as CC^{S1}. CC^{S1} augments the standard K-means clustering algorithm by considering cannot-link constraints. First, CC^{S1} initializes a set of cluster centroids. Then, each data point is assigned to the closest centroid that does not contain any data points in conflict. A data point of an order is the one-hot encoding of a set of non-overlapping sub-intervals discretized from the set of length intervals for all orders. After all data points are assigned, the location of each centroid is recalculated using the data points assigned to it. This process is repeated until the centroid positions do not change. The pseudocode of the CC^{S1} algorithm is described in Algorithm 2.

Algorithm 2: CC^{S1} Algorithm

```
Input  : dataset $D$, cannot-link constraints $\mathcal{C}$, cardinality $k$
Output: labels for dataset $D$
Initialize cluster centres $\mathcal{G} = \{G_1, \dots, G_k\}$;
while not converge do
    for data $d_j \in D$ do
        assign the closest cluster to $d_j$ such that none of the data previously
          assigned to this cluster conflicts with $d_j$;
    end
    if cannot find any cluster for a data point then
        restart algorithm with random centroids;
    end
    for centroid $G_i \in \mathcal{G}$ do
        update $G_i$ by averaging all of the points $d_j$ that has been assigned to it;
    end
end
```

Initializing Cluster Centers We randomly generate the cluster centers. The hope is that randomness allows good order groupings to be explored early.

4.4.1.3 Constraint Programming Formulation

Given the upper bound on the number of clusters k, finding congruent groups is inherently a satisfaction problem, so we can find clusters using CP solvers. We introduce an integer variable x_i for each order i with the domain being the set of indices of the k clusters. We hypothesize that having fewer congruent groups is beneficial to finding better solution to the overall problem, as this invites more diversity in the widthwise patterns. So, our satisfaction model is augmented with an objective function (4.11a) that minimizes the number of clusters. Constraint (4.11b) enforces the cannot-link relations.

$$\begin{aligned}
\min \quad & \sum_{i \in N} \max_{i \in N}(x_i) && && (4.11a)\\
\text{s.t.} \quad & x_i \neq x_{i'} && \forall (i, i') \in \mathcal{C} && (4.11b)\\
& x_i \in \{1, \dots, k\} && \forall i \in N && (4.11c)
\end{aligned}$$

Solution Cut For a given k, there may be multiple feasible combinations of order clusters. So, after the solver finds a solution, we add a cut to this CP model to avoid generating the same combination in the subsequent iterations of Algorithm 1. The cut is comprised of the disjunction of boolean clauses, each describing the non-homogeneity of the values of variables that previously belonged to the same cluster. This is formally expressed in Expression (4.12):

$$\bigvee_{r \in R} (\neg \bigwedge_{i,i' \in N_r \times N_r | i < i'} x_i == x_{i'}) \qquad (4.12)$$

where the set $N_r = \{i \in N | x_i^* = r\}$ denotes indices of orders belonging to cluster r, and x_i^* represents the solution of the variable x_i in the previous iteration.

4.4.2 Dividing Groups into Levels

The second stage distributes items of orders in each congruent group to levels of varying width to minimize any widthwise loss. Dividing orders within a congruent group into levels is inherently a 1D packing problem: we want to pack items belonging to these orders into potential levels widthwise. On top of 1DBPP, we need to consider the total area fulfilled for each order; however, until the levels are assigned to stocks, we cannot determine the exact value of the level length and thus the area fulfilled. Fortunately, we can approximate the total area using lower and upper bounds on the level length. Here, we solve for the level patterns iteratively, developing a mixed-integer linear program and a constraint program for each congruent group. In both formulations, we recall that $\sum_{k \in K} |J_k|$ is an upper bound on the number of levels. We recall $\overline{J} = \{1, \dots, \sum_{k \in K} |J_k|\}$ from Section 4.2.2.3 and let $J_h = \{j \in \overline{J} \mid W_j = W_h\}$ be the set of levels for a stock type. We denote the set of congruent groups as $G = \{N_1, \dots, N_{|G|}\}$, where N_g is the g^{th} congruent group. Given a congruent group, we approximate the lower and upper bounds to be $\rho_g^{min} = \max_{i \in N_g} \rho_i^{min}$ and $\rho_g^{max} = \min_{i \in N_g} \rho_i^{max}$.

4.4.2.1 Mixed-integer Formulation

We present a mixed-integer program, MIP^{S2}, to divide items of orders in the g^{th} congruent group into levels. We initialize a set of artificial levels with different widths and attempt to assign items of orders within each congruent group. For stocks of type $h \in H$, the number of these artificial levels with width W_h is always no greater than $|J_h|$. The variables used are as follows:

$x_{ijh} \coloneqq$ (integer) # of partitions on level j of stock type h assigned to order i

$s_{jh} \coloneqq$ (binary) 1 iff level j of stock type h is used

$$
\begin{aligned}
\min\ & \alpha \sum_{h \in H} \sum_{j \in J_h} W_h s_{jh} - \beta \sum_{h \in H} \sum_{j \in J_h} \sum_{i \in N_g} w_i x_{ijh} && && (4.13a)\\
\text{s.t.}\ & \sum_{i \in N_g} x_{ijh} w_i \le W_h s_{jh} && \forall j \in J_h, h \in H && (4.13b)\\
& \sum_{i \in N_g} x_{ijh} \le \eta s_{jh} && \forall j \in J_h, h \in H && (4.13c)\\
& \sum_{h \in H} \sum_{j \in J_h} \rho_g^{min} x_{ijh} \le q_i^{max} / w_i && \forall i \in N_g && (4.13d)\\
& \sum_{h \in H} \sum_{j \in J_h} \rho_g^{max} x_{ijh} \ge q_i^{min} / w_i && \forall i \in N_g && (4.13e)\\
& \sum_{i \in N_g} x_{ijh} w_i \ge \sum_{i \in N_g} x_{i(j+1)h} w_i && \forall h \in H, j \in J'_h && (4.13f)\\
& x_{ijh} \in \mathbb{Z}^+ && \forall i \in N_g, j \in J_h, h \in H && (4.13g)\\
& s_{jh} \in \mathbb{B} && \forall j \in J_h, h \in H && (4.13h)
\end{aligned}
$$

Objective (4.13a) minimizes the width loss of each level. Constraint (4.13b) restricts the width of the levels of different stock types. Constraint (4.13c) reflects the limit on the number of widthwise knives of the slitter. Constraints (4.13d) and (4.13e) impose lower and upper bounds on the area fulfilled assuming the level lengths are at their minimum and maximum. Constraint (4.13f) eliminates symmetry between levels of the same stock type. The rest of the constraints define the variables.

4.4.2.2 Constraint Programming Formulation

In CP, the 1D bin packing problem substructure can be efficiently modelled using the Pack constraint, from which we derive our CP model, CP^{S2}. We let x_{ip} be an integer variable representing level indices and Ω_{jh} be an integer variable denoting the used-up width. As not all items need to be present, like in Section 4.2.2.3, we use a dummy level indexed $D = |\overline{J}| + 1$ to absorb any unused items and a corresponding dummy leftover width variable Ω_D.

$$
\begin{aligned}
\min\ & \alpha \sum_{h \in H} \sum_{j \in J_h} (\Omega_{jh} > 0) W_h && && (4.14a)\\
& -\beta \sum_{i \in N_g} \sum_{p \in C_i} w_i (x_{ip} \neq D) && && (4.14b)\\
\text{s.t.}\ & \textsc{Pack}(\widetilde{\Omega}, x, w) && && (4.14c)\\
& \textsc{Count}(x, j) \leq \eta && \forall j \in \overline{J} && (4.14d)\\
& \sum_{j \in \overline{J}} \rho_g^{min} (x_{ip} == j) \leq q_i^{max} / w_i && \forall i \in N_g && (4.14e)\\
& \sum_{j \in \overline{J}} \rho_g^{max} (x_{ip} == j) \geq q_i^{min} / w_i && \forall i \in N_g && (4.14f)\\
& x_{ip} \leq x_{i(p+1)} && \forall i \in N_g, p \in C_i && (4.14g)\\
& x_{ip} \in \overline{J} \cup \{D\} && \forall i \in N_g, p \in C_i && (4.14h)\\
& \Omega_{jh} \in \{0, \dots, W_h\} && \forall j \in J_h, h \in H && (4.14i)\\
& \Omega_D \in \{0, \dots, \max_{h \in H} W_h\} && && (4.14j)
\end{aligned}
$$

Objective (4.14a) minimizes the width loss. Constraint (4.14c) uses the constraint Pack to pack orders into levels. Like in Section 4.2.2.3, $\widetilde{\Omega}$ is the combined set of actual and dummy stock width variables, and x and w is a one-dimensional vector flattened from variables x_{ip} and width parameters w_i, respectively. Constraint (4.14d) uses the constraint Count to limit the number of partitions within each level. Constraints (4.14e) and (4.14f) impose lower and upper bounds on the area fulfilled assuming the level lengths are approximately at their minimum and maximum. Constraint (4.14g) reduces the symmetry between partitions of the same order. The rest of the constraints define the variables.

4.4.3 Assigning Levels To Stocks

Finally, given a set of level patterns S from the second stage, we want to assign them to stocks. First, we let the number of items belonging to order i on level $j \in S$ be n_{ij}. As these levels have a fixed width, we only consider packing them lengthwise. The overlap among all of the length intervals of items on the level is often still an interval, so, instead

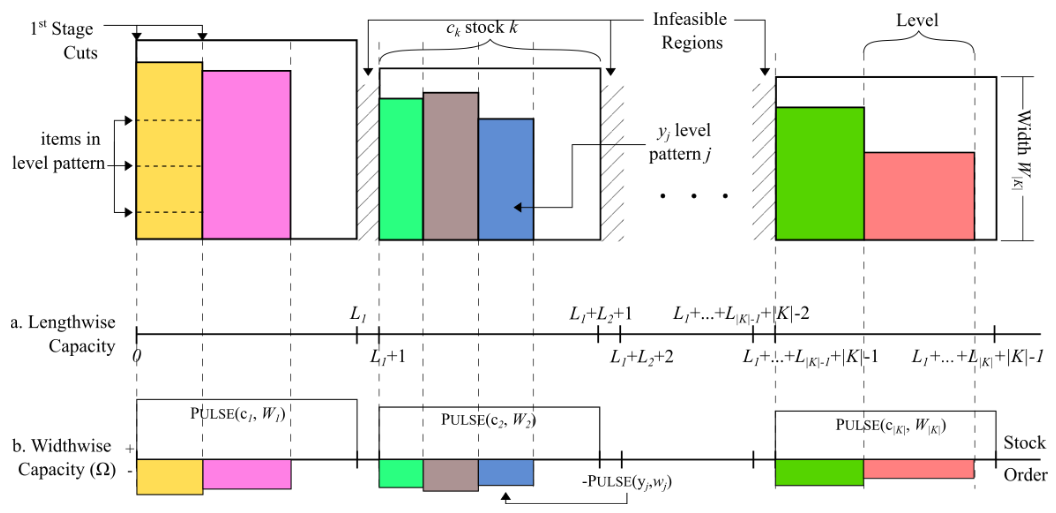


Figure 4.4: Single resource CP formulation of the third step in the sequential heuristic framework.

of having to solve a 1D packing problem, we solve a 1D Generalized Assignment Problem with Flexible Jobs [215], which is NP-hard. For this problem, we develop both monolithic and decomposition optimization models. First, we propose a CP formulation and two MILP formulations. For each MILP, we also investigate two classic Benders decomposition approaches, tackling the level length in the subproblem.

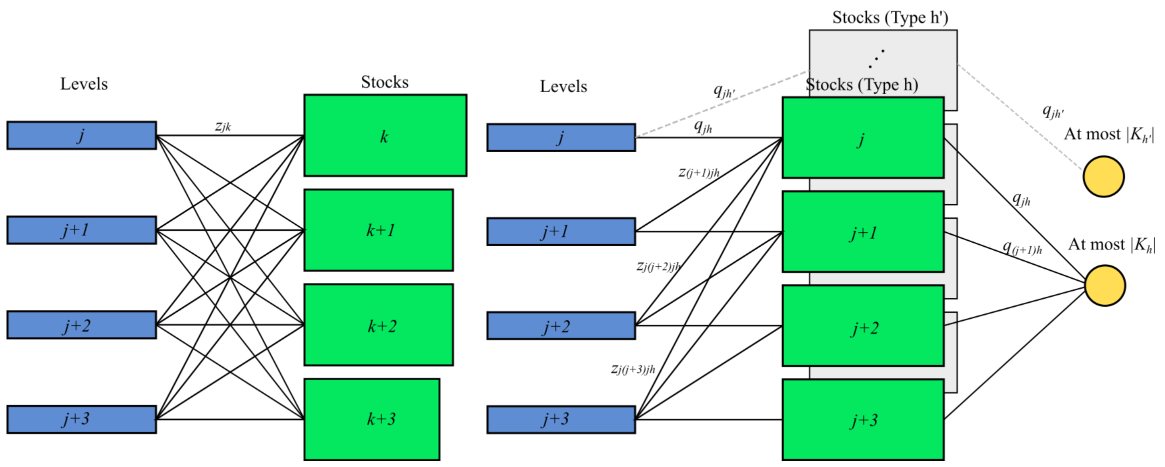
4.4.3.1 Monolithic Models

4.4.3.1.1 Constraint Programming Formulation In CP, the one-dimensional packing substructure suggests the usage of Pack, but this constraint cannot handle items of flexible sizes. Instead, we construct a scheduling-based CP formulation, CP^{S3}, that leverages the single resource transformation by concatenating the stocks lengthwise into a unified resource (Figure 4.4). We initialize an optional interval variable y_j for each level j obtained from the previous step. Slightly abusing the notation, we let ρ_j^{min} and ρ_j^{max} be the minimum and maximum allowable length of level j given the items assigned to it, and let w_j be the width of level pattern j. A level pattern cannot be placed in a stock whose width is greater than the level's width, so we can reduce the search space of y_j by first sorting the stocks by their widths in a decreasing order and then setting the maximum end time of y_j to be $\sum_{k\in K|w_j\le W_k}(L_k+1)$, the rightmost coordinate of the last stock that level j can fit into. The definition of the infeasible region $\bar{F}$ remains the same as in Section 4.2.1. Differing from the monolithic model CP_{SR}, we need an extra NoOverlap constraint to restrict the levels' positions, whereas CP_{SR} uses a guillotine state function.

The decision variables are as follows:

y_j := (interval) lengthwise interval of level pattern $j \in S$.

c_k := (interval) lengthwise interval representing stock k.



(a) Assignment scheme of MIP^{S3}_{simp}, where items are directly matched with the available stocks.

(b) Assignment scheme of MIP^{S3}_{comp}, where items are first assigned to a set of hypothetical stocks with a limiting assignment scheme.

Figure 4.5: Assignment schemes of MILP models for the third stage of the sequential heuristic.

$$
\begin{aligned}
&\min\ \alpha \sum_{k\in K} L_k W_k \textsc{PresenceOf}(c_k) - \beta \sum_{j\in S} w_j \textsc{SizeOf}(y_j) && (\mathrm{CP}^{S3}) && (4.15a)\\
&\text{s.t.}\ \ \textsc{NoOverlap}(y) && && (4.15b)\\
&\qquad \Omega = \sum_{k\in K} \textsc{Pulse}(c_k, W_k) - \sum_{j\in S} \textsc{Pulse}(y_j, w_j) && && (4.15c)\\
&\qquad \textsc{ForbidExtent}(y_j, \bar{F}) && \forall j \in S && (4.15d)\\
&\qquad \sum_{p\in C_i} n_{ij} \textsc{SizeOf}(y_j) \geq q_i^{min}/w_i && \forall i \in N && (4.15e)\\
&\qquad \sum_{p\in C_i} n_{ij} \textsc{SizeOf}(y_j) \leq q_i^{max}/w_i && \forall i \in N && (4.15f)\\
&\qquad y_j \in \textsc{IntervalVar}([0, \sum_{k\in K|w_j\leq W_k} (L_k+1)], [\rho_j^{min}, \rho_j^{max}]) && \forall j \in S && (4.15g)\\
&\qquad (4.1l), (4.1c) && && (4.15h)
\end{aligned}
$$

Objective (4.15a) describes the trim loss. Constraint (4.15b) uses NoOverlap to ensure the length intervals of levels do not overlap. Constraint (4.15c) describes a cumulative expression for the net width usage, and, in conjunction with Constraint (4.1c), ensures that c_k must be present if y_j is assigned to stock k. Constraint (4.15d) restricts the forbidden regions. Constraints (4.15e) and (4.15f) ensures that the total area fulfilled is within tolerances. The rest define the variables.

4.4.3.1.2 Mixed-integer Programming - Simple We first construct a mixed-integer linear model, MIP^{S3}_{simp}, that assigns levels to stocks and concurrently determines the levels' length (Figure 4.5a). We notice that the input level patterns may contain duplicates. To reduce symmetry, we reduce the set S to $\hat{S}$, a set containing unique level patterns only. Next, we introduce integer variables z_{jk} to count the number of the j^{th} unique level pattern matched to stock k. To determine the length of the level pattern j if assigned to stock k, we use continuous variables d_{jk}. We only need to declare z_{jk} and d_{jk} variables where the width of the j^{th} level does not exceed the width of the stock k. This subset of level patterns is denoted via a subscript of the stock's index k (e.g. $S_k = \{s \in S \mid \text{width of level } s \leq W_k\}$). Lastly, to maximize feasibility, we do not impose additional constraints to restrict the uniqueness of each level pattern in any solutions.

Overall, the variables used are as follows:

z_{jk} := (integer) # of unique level j is present on stock k

c_k := (binary) 1 iff stock k is used

d_{jk} := (continuous) surplus length of unique level pattern j on stock k

Objective (4.16a) defines the trim loss function. Constraint (4.16b) defines the lower and upper bounds of level patterns assigned to a stock. Constraint (4.16c) restricts the stock length. Constraints (4.16d) and (4.16e) ensure orders are fulfilled within tolerances. Constraint (4.16f) eliminates symmetry between stocks. The rest define the decision variables.

$$\begin{aligned}
\min\quad & \alpha \sum_{k \in K} L_k W_k c_k - \beta \sum_{k \in K} \sum_{j \in \hat{S}_k} w_j(\rho_j^{min} z_{jk} + d_{jk}) && (MIP^{S3}_{simp}) && (4.16a)\\
\text{s.t.}\quad & d_{jk} \leq (\rho_j^{max} - \rho_j^{min}) z_{jk} && \forall j \in \hat{S}_k, k \in K && (4.16b)\\
& \sum_{j \in \hat{S}_k} (\rho_j^{min} z_{jk} + d_{jk}) \leq L_k c_k && \forall k \in K && (4.16c)\\
& w_i \sum_{k \in K} \sum_{j \in \hat{S}_k} n_{ij}(\rho_j^{min} z_{jk} + d_{jk}) \leq q_i^{max} && \forall i \in N && (4.16d)\\
& w_i \sum_{k \in K} \sum_{j \in \hat{S}_k} n_{ij}(\rho_j^{min} z_{jk} + d_{jk}) \geq q_i^{min} && \forall i \in N && (4.16e)\\
& c_k \geq c_{k+1} && \forall k \in K'_h, h \in H && (4.16f)\\
& z_{jk} \in \mathbb{Z}^+ && \forall j \in \hat{S}_k, k \in K && (4.16g)\\
& c_k \in \mathbb{B} && \forall k \in K && (4.16h)\\
& d_{jk} \in \mathbb{R}^+ && \forall j \in \hat{S}_k, k \in K && (4.16i)
\end{aligned}$$

4.4.3.1.3 Mixed-integer Programming - Compact Inspired by Furini and Malaguti [86], we present an alternative mixed-integer programming formulation, denoted as MIP^{S3}_{comp}, that explicitly reduces the symmetry between levels by using a limiting assignment scheme (Figure 4.5b). For each level pattern $s \in S$, we initialize a hypothetical stock that is used in a solution if and only if that level is also present on the stock. In addition, a hypothetical stock with index k can only take on levels with indices greater than or equal to its index. Conversely, a level can only be assigned to hypothetical stocks with either the same index or smaller indices. Here, each hypothetical stock is essentially a possible combination of a level-to-stock assignment, and by restricting the assignment scheme, we reduce redundant combinations. Since the width of a level pattern limits the types of stock to which it can be assigned, given a level j with width w_j, we denote the set of viable stock types as $H_j = \{h \in H | w_j \leq W_h\}$, and the set of hypothetical stocks of type $h \in H_j$ as $K^h_j = \{j\} \bigcup \xi^h_j$, where $\xi^h_j = \{k \in S | w_j \leq W_h \wedge k < j\}$. For a given hypothetical stock k of type h, the set of assignable levels is denoted as $\zeta^h_k = \{j \in S | w_j \in W_h \wedge j \geq k\}$.

Overall, the following decision variables are used:

$q_{jh} :=$ (binary) 1 iff level j and the hypothetical stock j of type h are used

$z_{jkh} :=$ (binary) 1 iff level j is assigned to hypothetical stock k of type h

$d_{jkh} :=$ surplus length of level j assigned to hypothetical stock k of type h

$$\begin{aligned}
\min\ & \alpha \sum_{j \in S} \sum_{h \in H_j} ((L_h - \rho^{min}_j) W_h q_{jh} - W_h d_{jjh}) && (MIP^{S3}_{comp}) && (4.17a)\\
& -\beta \sum_{j \in S} \sum_{h \in H_j} \sum_{k \in \xi^h_j} w_j (z_{jkh} \rho^{min}_j + d_{jkh}) && &&\\
\text{s.t.}\ & \rho^{max}_j z_{jkh} \geq z_{jkh} \rho^{min}_j + d_{jkh} && \forall j \in S, h \in H_j, k \in K^h_j && (4.17b)\\
& \rho^{min}_j z_{jkh} \leq z_{jkh} \rho^{min}_j + d_{jkh} && \forall j \in S, h \in H_j, k \in K^h_j && (4.17c)\\
& d_{jjh} \leq (\rho^{max}_j - \rho^{min}_j) q_{jh} && \forall j \in S, h \in H_j && (4.17d)\\
& \sum_{j \in \zeta^h_k} (z_{jkh} \rho^{min}_j + d_{jkh}) \leq (L_h - \rho^{min}_k) q_{kh} - d_{kkh} && \forall k \in S, h \in H_k && (4.17e)\\
& \sum_{j \in S} \sum_{h \in H_j} \sum_{k \in \xi^h_j} n_{ij} w_i (z_{jkh} \rho^{min}_j + d_{jkh}) \leq q^{max}_i && \forall i \in N && (4.17f)\\
& \sum_{j \in S} \sum_{h \in H_j} \sum_{k \in \xi^h_j} n_{ij} w_i (z_{jkh} \rho^{min}_j + d_{jkh}) \geq q^{min}_i && \forall i \in N && (4.17g)\\
& \sum_{j \in S} q_{jh} \leq |K_h| && \forall h \in H_j && (4.17h)\\
& q_{jh} \in \mathbb{B} && \forall j \in S, h \in H_j && (4.17i)\\
& z_{jkh} \in \mathbb{B} && \forall j \in S, h \in H_j, k \in \xi^h_j && (4.17j)\\
& d_{jkh} \in \mathbb{R}^+ && \forall j \in S, h \in H_j, k \in \xi^h_j && (4.17k)
\end{aligned}$$

Objective (4.17a) defines the trim loss function. In the first term, we subtract ρ_j^{min} from L_h because if the j^{th} hypothetical stock of type h is used, then we always assign the j^{th} level to it. Constraints (4.17b) and (4.17c) define the upper and lower bounds of the j^{th} level pattern assigned to stock k if the indices do not match. In the case that they do match, the bounds are restricted by constraint (4.17d). Constraint (4.17e) restricts the length of each stock. Constraints (4.17f) and (4.17g) ensure orders are fulfilled within tolerances. Constraint (4.17h) ensures that the number of hypothetical stocks used does not exceed the number of available ones for a given stock type. The rest define the decision variables.

4.4.3.2 Decomposition-based Models

In both MIP_{simp}^{S3} and MIP_{comp}^{S3}, the length of each level pattern is determined concurrently with the actual assignment. In this section, we decompose each monolithic model using Benders Decomposition [25]. For each model, we derive two such decompositions: one from the upper bounds of the length intervals and another from the lower bounds. For all of these approaches, we ask the master problem to assign levels to stocks, and the subproblem to determine the levels' length. Since the subproblem can be formulated as a linear program, solving it is easy.

4.4.3.2.1 Benders Decompositions of MIP_{simp}^{S3}

4.4.3.2.1.1 From Lower Bound The first Benders reformulation, denoted as $BD_{simp}^{S3,lb}$, approaches the length of the level j assigned to stock k from its lower bound. In each iteration, we solve the master problem assuming the lengths of the levels are at their minimum. Then, in the subproblem, we calculate how much more area can we fulfill by increasing the length of assigned levels. In the case that the solution from the master problem is feasible for the subproblem, we add optimality cuts to the master problem to bound the estimation. If the master problem solution is infeasible in the subproblem, we add a feasibility cut to the master problem. The iterations repeat until either the solution of the master problem converges or the time limit is exceeded.

In addition to the decision variables in MIP_{simp}^{S3}, we introduce a continuous variable θ_{simp}^{lb} to estimate the area described in the subproblem. The master problem is expressed in Model (4.18). Objective (4.18a) expresses the trim loss, assuming levels' lengths are all at their minimum, less θ_{simp}^{lb}. Constraints (4.18b), (4.18c), and (4.18d) are valid inequalities determined using the bounds of the length intervals. Constraint (4.18b) ensures the minimum total length of the levels do not exceed the stock length. Constraints (4.18c) and (4.18d) ensure that the maximum total area fulfilled is not less than the minimum area tolerance, and that the minimum total area fulfilled is not more than the maximum area tolerance, respectively. Constraint (4.18e) ensures that the master problem is feasible on the first iteration.

$$\begin{aligned}
\min\ & \alpha \sum_{k\in K} W_k L_k c_k - \beta \sum_{k\in K}\sum_{j\in J_k} w_j \rho_j^{min} z_{jk} + \theta_{simp}^{lb} && && (4.18a)\\
\text{s.t.}\ & \sum_{j\in \hat{S}_k} \rho_j^{min} z_{jk} \le L_k c_k && \forall i \in N && (4.18b)\\
& w_i \sum_{k\in K}\sum_{j\in \hat{S}_k} n_{ij}\rho_j^{max} z_{jk} \ge q_i^{min} && \forall i \in N && (4.18c)\\
& w_i \sum_{k\in K}\sum_{j\in \hat{S}_k} n_{ij}\rho_j^{min} z_{jk} \le q_i^{max} && \forall i \in N && (4.18d)\\
& \theta_{simp}^{lb} \ge -\sum_{k\in K} W_k L_k && && (4.18e)\\
& \theta_{simp}^{lb} \in \mathbb{R} && && (4.18f)\\
& (4.16f) - (4.16i) && && (4.18g)
\end{aligned}$$

Given solutions $\hat{z}$ and $\hat{c}$ to the variables z and c in the master problem, we want to determine how much more order can we fulfill without violating the order tolerances by increasing the level length in the subproblem. The formulation is expressed in Model (4.19). Objective (4.19a) maximizes the total area induced by the additional length. Constraint (4.19b) limits total length of the levels assigned to each stock. Constraint (4.19c) enforces an upper bound on the total area of each order. Here, we do not need to limit its lower bound because it is implicitly satisfied by Constraint (4.19d), which bounds the surplus in a level's length, and Constraint (4.18c). We also associate a dual variable for the c^{th} set of constraints denoted as π_c.

$$\begin{aligned}
Q(\hat{z},\, \hat{c}) = \min\ & -\beta \sum_{k\in K}\sum_{j\in \hat{S}_k} w_j d_{jk} && && (4.19a)\\
(\pi_{1k}) \quad & \sum_{j\in \hat{S}_k} (\rho_j^{min}\hat{z_{jk}} + d_{jk}) \le L_k \hat{c_k} && \forall k \in K && (4.19b)\\
(\pi_{2i}) \quad & w_i \sum_{k\in K}\sum_{j\in \hat{S}_k} n_{ij}(\rho_j^{min}\hat{z_{jk}} + d_{jk}) \le q_i^{max} && \forall i \in N && (4.19c)\\
(\pi_{3jk}) \quad & d_{jk} \le (\rho_j^{max} - \rho_j^{min})\hat{z_{jk}} && \forall j \in \hat{S}_k, k \in K && (4.19d)\\
& (4.16i) && && (4.19e)
\end{aligned}$$

We add an optimality cut (Equation 4.20) to the master problem after solving the subproblem. The dual variables used in the cut are optimal to the subproblem in the same iteration.

$$\theta^{lb}_{simp} \geq \sum_{k\in K}(\sum_{j\in\hat{S}_k}\rho_j^{min}z_{jk} - L_k c_k)\pi_{1k} + \sum_{i\in N}(\sum_{k\in K}\sum_{j\in\hat{S}_k} n_{ij}\rho_j^{min}z_{jk} - q_i^{max}/w_i)\pi_{2i} + \sum_{k\in K}\sum_{j\in\hat{S}_k}(\rho_j^{min} - \rho_j^{max})\pi_{3jk} \quad (4.20)$$

If the subproblem is infeasible, we add a feasibility cut (Equation 4.21) using the extreme rays of the infeasible dual problem. For each set of dual variables π_c, we denote the corresponding extreme ray as ϕ_c. The feasibility cut is as follows:

$$0 \geq \sum_{k\in K}(\sum_{j\in\hat{S}_k}\rho_j^{min}z_{jk} - L_k c_k)\phi_{1k} + \sum_{i\in N}(\sum_{k\in K}\sum_{j\in\hat{S}_k} n_{ij}\rho_j^{min}z_{jk} - q_i^{max}/w_i)\phi_{2i} + \sum_{k\in K}\sum_{j\in\hat{S}_k}(\rho_j^{min} - \rho_j^{max})\phi_{3jk} \quad (4.21)$$

4.4.3.2.1.2 From Upper Bound The second Benders reformulation, denoted as $BD^{S3,ub}_{simp}$, approaches the length of the level j assigned to stock k from its upper bound, with it being expressed as $\rho_j^{max}z_{jk} - d_{jk}$ in the Benders Reformulation. The intuition is similar to the MIP^{S3}_{simp}'s decomposition from the lower bound. In the master problem, we assume all lengths are at their maximum length. In the subproblem, we determine the total area exceeding the stocks and add optimality and feasibility cuts as required. In the master problem, we denote the estimation of the area exceeded as θ^{ub}_{simp}.

The master problem is given in Model (4.22).

$$\begin{aligned} \min\ & \alpha\sum_{k\in K} W_k L c_k - \beta\sum_{k\in K}\sum_{j\in\hat{S}_k} w_j\rho_j^{max}z_{jk} + \theta^{ub}_{simp} && (4.22a)\\ \text{s.t.}\ & \theta^{ub}_{simp} \in \mathbb{R}^+ && (4.22b)\\ & (4.16f) - (4.16i), (4.18b) - (4.18d) && (4.22c) \end{aligned}$$

Next, we want to determine the minimum amount of area that makes the master problem solution $(\hat{z}, \hat{c})$ feasible in the subproblem. Objective (4.23a) describes the trim loss. Constraint (4.23b) restricts the stock length. Constraint (4.23c) ensures the area fulfilled is within order tolerance. Constraint (4.23d) defines the maximum deficit of a level's length.

$$\begin{aligned}
&\mathrm{Q}(\hat{z},\, \hat{c}) = \min\ \beta \sum_{k\in K}\sum_{j\in\hat{S}_k} w_j d_{jk} && && (4.23a)\\
&(\pi_{1k}) \qquad \sum_{j\in\hat{S}_k}(\rho_j^{max}\hat{z_{jk}} - d_{jk}) \le L_k\hat{c_k} && \forall k\in K && (4.23b)\\
&(\pi_{2i}) \qquad w_i\sum_{k\in K}\sum_{j\in\hat{S}_k} n_{ij}(\rho_j^{max}\hat{z_{jk}} - d_{jk}) \ge q_i^{min} && \forall i\in N && (4.23c)\\
&(\pi_{3jk}) \qquad d_{jk} \le (\rho_j^{max} - \rho_j^{min})\hat{z_{jk}} && \forall j\in\hat{S}_k, k\in K && (4.23d)\\
&\qquad\qquad (4.16i) && && (4.23e)
\end{aligned}$$

Accordingly, we add the optimality cut (Equation 4.24) to the master problem.

$$\theta_{simp}^{ub} \ge \sum_{k\in K}(\sum_{j\in\hat{S}_k}\rho_j^{max}\hat{z_{jk}} - L_k\hat{c_k})\pi_{1k} + \sum_{i\in N}(q_i^{min}/w_i - \sum_{k\in K}\sum_{j\in\hat{S}_k} n_{ij}\rho_j^{max}\hat{z_{jk}})\pi_{2i} + \sum_{k\in K}\sum_{j\in\hat{S}_k}(\rho_j^{min} - \rho_j^{max})\pi_{3jk} \quad (4.24)$$

If the subproblem is infeasible, we add a feasibility cut using extreme rays ϕ_c (Equation 4.25).

$$0 \ge \sum_{k\in K}(\sum_{j\in\hat{S}_k}\rho_j^{max}\hat{z_{jk}} - L_k\hat{c_k})\phi_{1k} + \sum_{i\in N}(q_i^{min}/w_i - \sum_{k\in K}\sum_{j\in\hat{S}_k} n_{ij}\rho_j^{max}\hat{z_{jk}})\phi_{2i} + \sum_{k\in K}\sum_{j\in\hat{S}_k}(\rho_j^{min} - \rho_j^{max})\phi_{3jk} \quad (4.25)$$

4.4.3.2.2 Benders Decompositions of MIP_{comp}^{S3}

4.4.3.2.2.1 From Lower Bound Similar to $BD_{simp}^{S3,lb}$, we decompose MIP_{comp}^{S3} from the lower bound of the length interval, denoted as $BD_{comp}^{S3,lb}$. The intuition behind the decomposition is identical to $BD_{simp}^{S3,lb}$, and we denote the subproblem estimation in the master problem as θ_{comp}^{lb}. The master problem is described by Model (4.26). Objective (4.26a) describes the trim loss objective assuming a minimum level length. Constraint (4.26b) restricts the stock length. Constraint (4.26c) and (4.26d) ensure that the upper and lower area tolerances can be satisfied by the minimum and the maximum length, respectively.

$$\min \ \alpha \sum_{j\in S}\sum_{h\in H_j} W_h(L_h-\rho_j^{min})q_{jh} \tag{4.26a}$$
$$-\beta \sum_{j\in S}\sum_{h\in H_j}\sum_{k\in\xi_j^h} w_j\rho_i^{min}z_{jkh}+\theta_{comp}^{lb}$$
$$\text{s.t.} \ \sum_{j\in\zeta_k^h}\rho_j^{min}z_{jkh}\le (L-\rho_k^{min})q_{kh} \qquad \forall k\in S, h\in H_k \tag{4.26b}$$
$$\sum_{j\in S}\sum_{h\in H_j}\sum_{k\in\xi_j^h} n_{ij}w_iz_{jkh}\rho_i^{min}\le q_i^{max} \qquad \forall i\in N \tag{4.26c}$$
$$\sum_{j\in S}\sum_{h\in H_j}\sum_{k\in\xi_j^h} n_{ij}w_iz_{jkh}\rho_i^{max}\ge q_i^{min} \qquad \forall i\in N \tag{4.26d}$$
$$(4.17h)-(4.17j) \tag{4.26e}$$

The subproblem in Model (4.27) solves for the decrease in objective value from the flexible length given a master problem solution $(\hat{z},\hat{q})$. Objective 4.27a describes the aforementioned surplus area. Constraints (4.27b) and (4.27c) define the upper bound on the lengthwise surplus. Constraint (4.27d) restricts the stock length. Constraints (4.26c) and (4.26d) satisfy the area tolerances of each order.

$$\text{Q(q, z)} = \min \ -\beta\sum_{j\in S}\sum_{h\in H_j}\sum_{k\in\xi_j^h} w_jd_{jkh} \tag{4.27a}$$
$$(\pi_{1jkh}) \qquad (\rho_j^{max}-\rho_j^{min})\hat{z_{jkh}}\ge d_{jkh} \qquad \forall j\in S, h\in H_j, k\in K_j^h \tag{4.27b}$$
$$(\pi_{2jh}) \qquad (\rho_j^{max}-\rho_j^{min})\hat{q_{jh}}\ge d_{jjh} \qquad \forall j\in S, h\in H_j \tag{4.27c}$$
$$(\pi_{3kh}) \qquad \sum_{j\in\zeta_k^h}(\hat{z_{jkh}}\rho_j^{min}+d_{jkh})\le (L_h-\rho_k^{min})\hat{q_{kh}} \qquad \forall k\in S, h\in H_j \tag{4.27d}$$
$$(\pi_{4i}) \qquad \sum_{j\in S}\sum_{h\in H_j}\sum_{k\in\xi_j^h} n_{ij}w_i(\hat{z_{jkh}}\rho_i^{min}+d_{jkh})\le q_i^{max} \qquad \forall i\in N \tag{4.27e}$$
$$(\pi_{5i}) \qquad \sum_{j\in S}\sum_{h\in H_j}\sum_{k\in\xi_j^h} n_{ij}w_i(\hat{z_{jkh}}\rho_i^{min}+d_{jkh})\ge q_i^{min} \qquad \forall i\in N \tag{4.27f}$$
$$(4.17k) \tag{4.27g}$$

Given a solution $(\hat{z},\hat{q})$, Expressions (4.28) and (4.29) describe the optimality and feasibility cut, respectively.

$$\theta^{lb}_{comp} \geq \sum_{j\in S}\sum_{h\in H_j}\sum_{k\in K^h_j}(\rho^{min}_j - \rho^{max}_j)\hat{z_{jkh}}\pi_{1jkh} + \sum_{j\in S}\sum_{h\in H_j}(\rho^{min}_j - \rho^{max}_j)\hat{q_{jh}}\pi_{2jh} +$$
$$\sum_{k\in S}\sum_{h\in H_k}(\hat{z_{jkh}}\rho^{min}_j - (L_h - \rho^{min}_k)\hat{q_{kh}})\pi_{3kh} + \sum_{i\in N}(\sum_{j\in S}\sum_{h\in H_j}\sum_{k\in\xi^h_j} n_{ij}w_i\rho^{min}_i\hat{z_{jkh}} - q^{max}_i)\pi_{4i} +$$
$$\sum_{i\in N}(q^{min}_i - \sum_{j\in S}\sum_{h\in H_j}\sum_{k\in\xi^h_j} n_{ij}w_i\rho^{min}_i\hat{z_{jkh}})\pi_{5i} \quad (4.28)$$

$$0 \geq \sum_{j\in S}\sum_{h\in H_j}\sum_{k\in K^h_j}(\rho^{min}_j - \rho^{max}_j)\hat{z_{jkh}}\phi_{1jkh} + \sum_{j\in S}\sum_{h\in H_j}(\rho^{min}_j - \rho^{max}_j)\hat{q_{jh}}\phi_{2jh} +$$
$$\sum_{k\in S}\sum_{h\in H_k}(\hat{z_{jkh}}\rho^{min}_j - (L_h - \rho^{min}_k)\hat{q_{kh}})\phi_{3kh} + \sum_{i\in N}(\sum_{j\in S}\sum_{h\in H_j}\sum_{k\in\xi^h_j} n_{ij}w_i\rho^{min}_i\hat{z_{jkh}} - q^{max}_i)\phi_{4i} +$$
$$\sum_{i\in N}(q^{min}_i - \sum_{j\in S}\sum_{h\in H_j}\sum_{k\in\xi^h_j} n_{ij}w_i\rho^{min}_i\hat{z_{jkh}})\phi_{5i} \quad (4.29)$$

4.4.3.2.2.2 From Upper Bound Similarly, we decompose MIP^{S3}_{comp} from the upper bound of the length interval, denoted as $BD^{S3,ub}_{comp}$. The intuition behind the decomposition is identical to $BD^{S3,ub}_{simp}$, and we denote the estimation variable in the master problem as θ^{ub}_{comp}. The master problem is described by Model (4.30).

$$\min \ \alpha\sum_{j\in S}\sum_{h\in H_j} W_h(L_h - \rho^{max}_j)q_{jh} \quad (4.30a)$$
$$-\beta\sum_{j\in S}\sum_{h\in H_j}\sum_{k\in\xi^h_j} w_j\rho^{max}_i z_{jkh} + \theta^{ub}_{comp}$$
$$(4.30b)$$
$$\text{s.t.} \quad (4.17h) - (4.17j), (4.26b) - (4.26d) \quad (4.30c)$$
$$(4.30d)$$

The subproblem in Model (4.31) solves for the lengthwise surplus area given a master problem solution $(\hat{z}, \hat{q})$. Objective (4.31a) describes the decrease in trim loss. Constraints (4.31b) and (4.31c) define the upper bound on the lengthwise surplus. Constraint (4.31d) restricts the stock length. Constraints (4.31e) and (4.31f) satisfy the area tolerances of each order.

$$\text{Q(q, z)} = \min \ \beta \sum_{j\in S}\sum_{h\in H_j}\sum_{k\in\xi_j^h} w_j d_{jkh} \tag{4.31a}$$

$$(\pi_{1jkh}) \qquad (4.27b) \tag{4.31b}$$

$$(\pi_{2jh}) \qquad (4.27c) \tag{4.31c}$$

$$(\pi_{3kh}) \qquad \sum_{j\in\zeta_k^h}(z_{\hat{jkh}}\rho_j^{max} - d_{jkh}) \le (L_h - \rho_k^{max})q_{\hat{kh}} + d_{kkh} \quad \forall k\in S, h\in H \tag{4.31d}$$

$$(\pi_{4i}) \qquad \sum_{j\in S}\sum_{h\in H_j}\sum_{k\in\xi_j^h} n_{ij}w_i(z_{\hat{jkh}}\rho_j^{max} - d_{jkh}) \le q_i^{max} \quad \forall i\in N \tag{4.31e}$$

$$(\pi_{5i}) \qquad \sum_{j\in S}\sum_{h\in H_j}\sum_{k\in\xi_j^h} n_{ij}w_i(z_{\hat{jkh}}\rho_j^{max} - d_{jkh}) \ge q_i^{min} \quad \forall i\in N \tag{4.31f}$$

$$(4.17k) \tag{4.31g}$$

Given a solution $(\hat{z}, \hat{q})$, Inequalities (4.32) and (4.33) describe the optimality and feasibility cut, respectively.

$$\theta_{comp}^{ub} \ge \sum_{j\in S}\sum_{h\in H_j}\sum_{k\in K_j^h}(\rho_j^{min} - \rho_j^{max})z_{\hat{jkh}}\pi_{1jkh} + \sum_{j\in S}\sum_{h\in H_j}(\rho_j^{min} - \rho_j^{max})q_{\hat{jh}}\pi_{2jh} +$$
$$\sum_{k\in S}\sum_{h\in H_k}(z_{\hat{jkh}}\rho_j^{max} - (L_h - \rho_k^{max})q_{\hat{kh}})\pi_{3kh} + \sum_{i\in N}(\sum_{j\in S}\sum_{h\in H_j}\sum_{k\in\xi_j^h} n_{ij}w_i\rho_i^{min}z_{\hat{jkh}} - q_i^{max})\pi_{4i} +$$
$$\sum_{i\in N}(q_i^{min} - \sum_{j\in S}\sum_{h\in H_j}\sum_{k\in\xi_j^h} n_{ij}w_i\rho_i^{min}z_{\hat{jkh}})\pi_{5i} \tag{4.32}$$

$$0 \ge \sum_{j\in S}\sum_{h\in H_j}\sum_{k\in K_j^h}(\rho_j^{min} - \rho_j^{max})z_{\hat{jkh}}\phi_{1jkh} + \sum_{j\in S}\sum_{h\in H_j}(\rho_j^{min} - \rho_j^{max})q_{\hat{jh}}\phi_{2jh} +$$
$$\sum_{k\in S}\sum_{h\in H_k}(z_{\hat{jkh}}\rho_j^{max} - (L_h - \rho_k^{max})q_{\hat{kh}})\phi_{3kh} + \sum_{i\in N}(\sum_{j\in S}\sum_{h\in H_j}\sum_{k\in\xi_j^h} n_{ij}w_i\rho_i^{min}z_{\hat{jkh}} - q_i^{max})\phi_{4i} +$$
$$\sum_{i\in N}(q_i^{min} - \sum_{j\in S}\sum_{h\in H_j}\sum_{k\in\xi_j^h} n_{ij}w_i\rho_i^{min}z_{\hat{jkh}})\phi_{5i} \tag{4.33}$$

4.5 Numerical Results

We first examine the representational efficiency of the exact approaches. Then, we select the best configurations of the sequential heuristic and make an overall comparison on feasibility, solution quality, and runtime performance. The full results can be found in Appendix B.

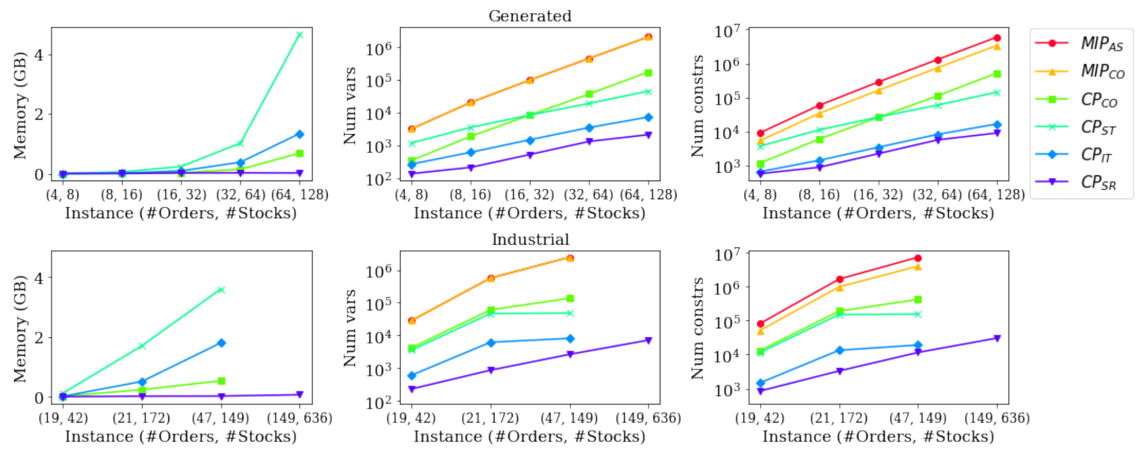


Figure 4.6: Comparison of the number of variables, the number of constraints, and the model memory before search over instance sizes. Note the scales of the y-axes.

4.5.1 Monolithic Model Size Comparison

Figure 4.6 compares the model sizes of the exact approaches based on the number of variables, the number of constraints, and the model memory (before search). The memory usage of MILP models is not accessible from the solver. A data point is omitted if the corresponding model fails to initialize in memory within the one-hour time limit. The CP_{SR} formulation is significantly smaller than the other models across all three measures, especially as the instances scale up. For the largest industrial instances, no other models could be loaded before timing out. For the largest generated instances, $(64, 128)$, the CP_{CO} model requires about 800MB of memory, while CP_{SR} only needs 20MB.

The CP_{ST} model used more memory than any other model for all instances. Interestingly, for the generated instances with $|N| \geq 16$ and the industrial instances, it used fewer variables and constraints than CP_{CO} and both MILP models. Typically using fewer variables and constraints is associated with using less memory, but here the size of the constraints is determinant. Notably, CP_{ST} adds expressions over possible partitions on all stocks to obtain the total area fulfilled for each order. As the number of stocks increases, this sum becomes especially hefty.

CP_{IT} used the second-fewest variables and constraints, trailing only behind CP_{SR}. However, CP_{IT} consumed the second-most memory, and the marginal increase over the next smallest model, CP_{CO}, is significant. Just like with CP_{ST}, the size of the constraints is again crucial. CP_{IT} iterates over all items to obtain the total area fulfilled for an order, while CP_{CO} iterates only over orders and levels. This difference is insignificant in smaller instances but becomes pronounced as the instance size increases.

	Generated		Industrial	
	OptGap	# Feasible	OptGap	# Feasible
$S : CP^{S1} + MIP^{S2} + MIP^{S3}_{simp}$	29.3	50	62.9	4
$S : CC^{S1} + MIP^{S2} + MIP^{S3}_{simp}$	29.8	50	63.4	4
$S : CC^{S1} + MIP^{S2} + BD^{S3,ub}_{simp}$	31.3	50	61.9	4
$S : CP^{S1} + MIP^{S2} + BD^{S3,ub}_{simp}$	31.7	49	61.6	4
$S : CC^{S1} + MIP^{S2} + BD^{S3,lb}_{simp}$	32.5	50	62.4	4
$S : CP^{S1} + MIP^{S2} + BD^{S3,lb}_{simp}$	33.2	49	65.6	3
$S : CP^{S1} + MIP^{S2} + CP^{S3}$	35.2	50	65.1	4

Table 4.2: Optimality gap and number of feasible solutions of the top 7 sequential heuristic configurations by their gaps on the generated instances.

4.5.2 Sequential Heuristic Selection

We select top performing configurations of the sequential heuristic that will be compared with the rest of the approaches. Table 4.2 displays the optimality gap (OptGap) and the number of feasible solutions of the top sequential heuristic configurations by OptGap. The OptGap is calculated from Equation (4.34), where $z(n;i)$ is the objective value of approach n for instance i, and $lb(i)$ is the best lower bound of instance i across all approaches. For a given solution approach, we penalize an unsolved instance by the worst objective value found for the instance. We base the selection on the lowest optimality gaps on the generated instances to avoid biasing towards the industrial instances, which are more limited in quantity.

$$\%\ \text{OptGap}(n;\ i) = 100\frac{z(n;i) - lb(i)}{lb(i)} \tag{4.34}$$

Overall, configurations $S : CC^{S1}+MIP^{S2}+MIP^{S3}_{simp}$ and $S : CP^{S1}+MIP^{S2}+MIP^{S3}_{simp}$ are the most preferred, as they found the lowest OptGap for the generated instances. Their only difference is in the first stage solvers, with the CP^{S1} configuration yielding slightly better solutions than the one with CC^{S1}. All top configurations use MIP^{S2} and an approach related to MIP^{S3}_{simp}. In particular, the performance of configurations using Benders Decomposition on MIP^{S3}_{simp} is lacking compared to using just the standalone formulation, suggesting that the Benders cuts are ineffective and the problem geometry complex. Configurations using CP^{S3} are not preferred, as they produced weaker solutions than alternative approaches. For the subsequent analysis, we select $S : CC^{S1} + MIP^{S2} + MIP^{S3}_{simp}$ and $S : CP^{S1} + MIP^{S2} + MIP^{S3}_{simp}$ to represent the performances of the sequential heuristic. For conciseness, we henceforth denote $S : CC^{S1} + MIP^{S2} + MIP^{S3}_{simp}$ as S_{CC} and $S : CP^{S1} + MIP^{S2} + MIP^{S3}_{simp}$ as S_{CP}.

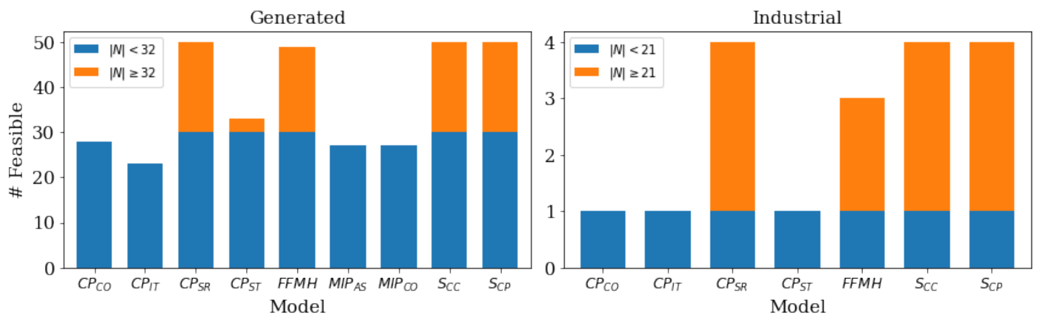


Figure 4.7: Number of generated and industrial instances with a feasible solution by each model for 2SCSP-FF.

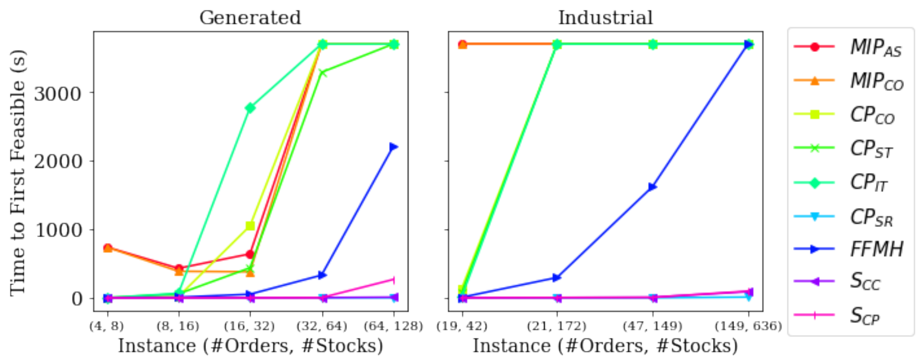


Figure 4.8: The average run time required to find a feasible solution for a given model at an instance size for 2SCSP-FF.

4.5.3 Feasibility Analysis

Figures 4.7 and 4.8 report the number of instances for which a feasible solution was found and the average time to feasibility or termination, respectively. Only CP_{SR}, S_{CC}, and S_{CP} found a feasible solution to all instances. In particular, CP_{SR} reached feasibility the fastest amongst all methods, requiring less than 100 seconds for the largest industrial instance. The short time-to-feasibility, however, differs from the results for routing problems [36], where the model struggled to find feasible solutions quickly. We suspect that this disparity is due to the looser constraints on interval variables for our problem compared to the routing formulation. Slightly worse than CP_{SR} are S_{CC} and S_{CP}, using 50 more seconds for the largest industrial instance and 160 more seconds for the largest generated instance, respectively. *FFMH* found a feasible solution to all but one generated instance and the largest industrial instance, while other exact approaches struggled with the large generated and industrial instances. Notably, neither MILP models found a feasible solution to the industrial instances within the runtime.

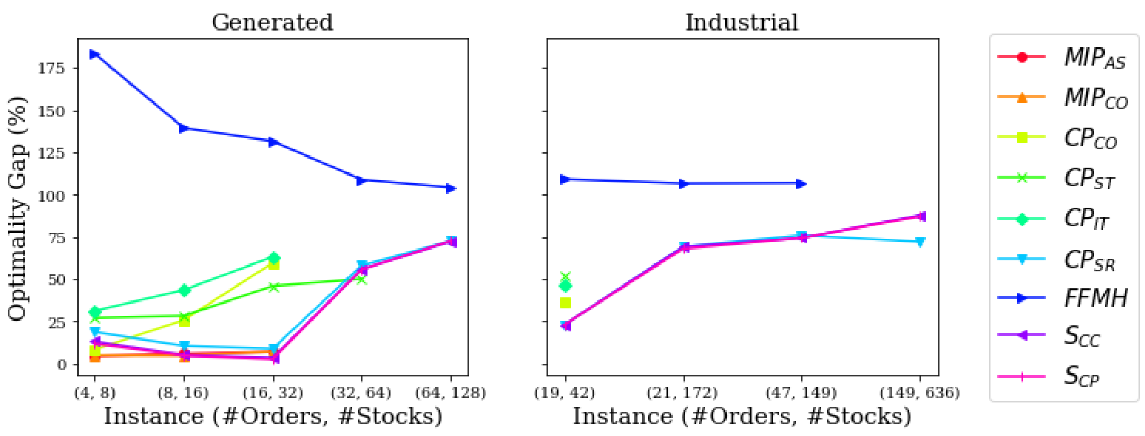


Figure 4.9: % optimality gap for generated and industrial instances for 2SCSP-FF. Instances that are not solved by an approach are not included in that approach's measure.

4.5.4 Overall Quality

Figure 4.9 displays the optimality gap of each approach calculated using Equation (4.34). Any unsolved instances are omitted from the visualization, so some approaches are penalized for finding solutions to harder instances compared to approaches that did not do so.

First, we compare the exact approaches. MIP_{AS} demonstrated the strongest performance for generated instances with $|N| \leq 16$. MIP_{AS} proved optimality for two generated instances, the only ones proven optimal across all models. For larger instances ($|N| \geq 32$), both MILP models scaled poorly, failing to find a feasible solution within the time limit. CP_{CO}, CP_{IT}, and CP_{ST} struggle to find competitive solutions to the generated instances past $N = 4$, eventually encountering loading time issues for larger instances. CP_{CO} found similar solutions to MIP_{AS} for the smallest generated instances, but could not prove optimality due to a weaker lower bound. CP_{SR} consistently outperformed the MILP and the integer-based CP models for all but the smaller instances. A similar trend is observed in the industrial instances, where CP_{SR} was the only model-based approach that found a feasible solution to more than one instance. We note that the lower bounds of the instances with $|N| \geq 21$ are generated by CP_{SR}, but their values are non-trivial, a rare feat for typical CP approaches. Here, CP Optimizer computes the lower bound using an automatic LP-based relaxation of the scheduling constraints [158], a feature not available in other CP solvers.

The sequential heuristic S_{CP} outperformed CP_{SR} for most of the generated and industrial instances. However, CP_{SR} is better than S_{CP} for the largest industrial instance, making it the preferred approach for solving large-scale industrial use cases. S_{CC} is marginally worse than S_{CP} for the smaller generated instances, but their solution qualities are approximately equal for the larger generated and industrial ones. $FFMH$ found weaker solutions to all other approaches for both generated and industrial instances on average.

Interestingly, although CP_{SR} is one of the best-performing monolithic formulations, us-

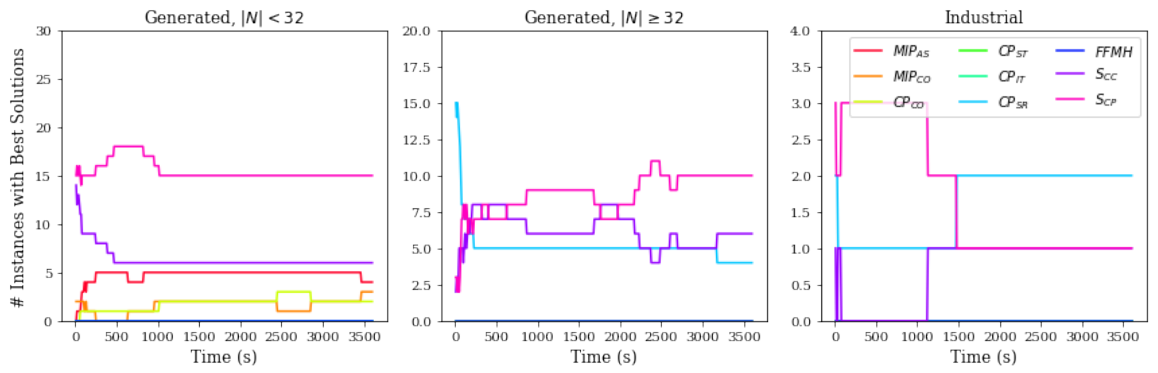


Figure 4.10: Number of instances for which an approach found the best solution over runtime.

ing the single resource model CP^{S3} in the last subproblem of the sequential heuristic yielded poorer solution quality than using MIP^{S3}_{simp} (Table 4.2). This is consistent with our results in Figure 4.9, where both monolithic MILP models found higher quality solutions than CP_{SR} did for the smaller instances. In other words, the value of CP_{SR} (and hence CP^{S3}) is more in its representational efficiency, enabling it to solve industrial instances that the monolithic MILP models could not. However, the decomposed subproblems in the sequential heuristic framework are less complex than the full 2SCSP-FF, which allows MIP^{S3}_{simp} to solve the larger instances that MIP_{AS} and MIP_{CO} could not. So, the advantage of CP^{S3} is muted, as the heuristic can effectively exploit MILP's reasoning capabilities without encountering tractability issues. Nonetheless, there may still exist be larger instances that only CP^{S3} can solve.

4.5.5 Runtime Performance

Finally, we examine the performance of the proposed approaches over their runtime by averaging the number of best solutions every 10 seconds (Figure 4.10). For the small generated instances, S_{CP} found the best solution for more instances than any other approaches from start to finish, with S_{CC} being a distant second. After the 500th second, S_{CC}'s superiority diminished, undermined mostly by MIP_{AS}. Interestingly, the quality of solutions found by CP_{SR}, which yielded low optimality gaps for these instances, was dominated by S_{CP}, S_{CC}, and MIP_{AS} throughout the runtime. For the large generated instances, CP_{SR} found the most number of best solutions in the first 200 seconds, after which the heuristics generally are better. The dynamics of S_{CP} and S_{CC} almost reflect one another, suggesting that CC^{S1} is competitive with CP^{S1}. For the industrial instances, S_{CP} found the most number of best solutions between the 100th and 1500th second, after which CP_{SR} is preferred. In addition, CP_{SR}'s solution to the largest industrial instance is stronger than the others' solutions throughout the runtime, again attesting to its suitability for large-scale use cases.

4.6 Conclusion

In this chapter, we proposed a single resource CP model, three integer-based CP models, two MILP models, a first-fit-based heuristic, and a sequential heuristic framework. Out of these exact approaches, CP_{SR} has significant advantages in its problem representation and overall performance. The memory efficiency of CP_{SR} results from the compact representation of complicated substructures and variables. To represent the guillotine cuts, CP_{SR} is the only model that does not enumerate the set of levels, instead using just a state function and an AlwaysConstant constraint. Similarly, CP_{SR} restricts the widthwise capacities and the partition counts without levelwise constraints. By using the fewest variables and constraints, the CP_{SR} model has at least an order-of-magnitude advantage in its memory usage. As the instances scale up, this advantage increases. Including the heuristics, our empirical results show that S_{CC} is generally the preferred approach, having found higher quality solutions than CP_{SR}, the best exact approach, in less time within a one-hour runtime. Nonetheless, CP_{SR} remains the most suitable for the largest industrial instances.

Chapter 5

2SCSP with Flexible Item Length, Flexible Demand, and Marriageability

In this chapter, we extend the top-performing models and heuristics proposed for 2SCSP-FF (the single resource CP model, the assignment-based MILP model, the counting-based MILP model, the first-fit based heuristic and the top-performing sequential heuristic configurations) to tackle the full Two-Stage Cutting Stock Problem with Flexible Item Length, Flexible Demand, and Order-to-order Marriageability (2SCSP-FFM). In 2SCSP-FFM, a more constrained version of 2SCSP-FF, stocks need to be treated so that the cut items have desired properties. As a result, items from orders requesting different properties cannot be cut from the same stock, creating order-to-order marriageability requirements. For the exact approaches, we develop the formulations from different modelling perspectives, and, for the heuristics, we modify the algorithms surgically. We conclude this chapter with empirical experiments.

5.1 Problem Description

The 2SCSP-FFM restricts order-to-order marriageability: stocks are treated once before cutting so that the cut partitions have desired properties; consequently, orders requesting different properties cannot be cut from the same stock. There are no restrictions on the type of treatment that is applied a stock, but only one treatment can be applied to a given stock. We recall that G is the set of combinations of order properties and N_g is the set of orders with treated properties $g \in G$. Equivalently, we can construct a binary marriageability matrix M such that $M_{ii'} = 1$ if and only if items from order i and i' have identical properties (i.e. $\exists g \in G \mid i, i' \in N_g$) and hence can be cut from the same stock.

Model	**Conflict-based**	**Treatment-based**
CP_{SR}		x
MIP^{C}_{AS}	x	
MIP^{T}_{AS}		x
MIP^{C}_{CO}	x	
MIP^{T}_{CO}		x

Table 5.1: Perspectives taken by the marriageability formulations of the top-performing exact approaches on 2SCSP-FFM.

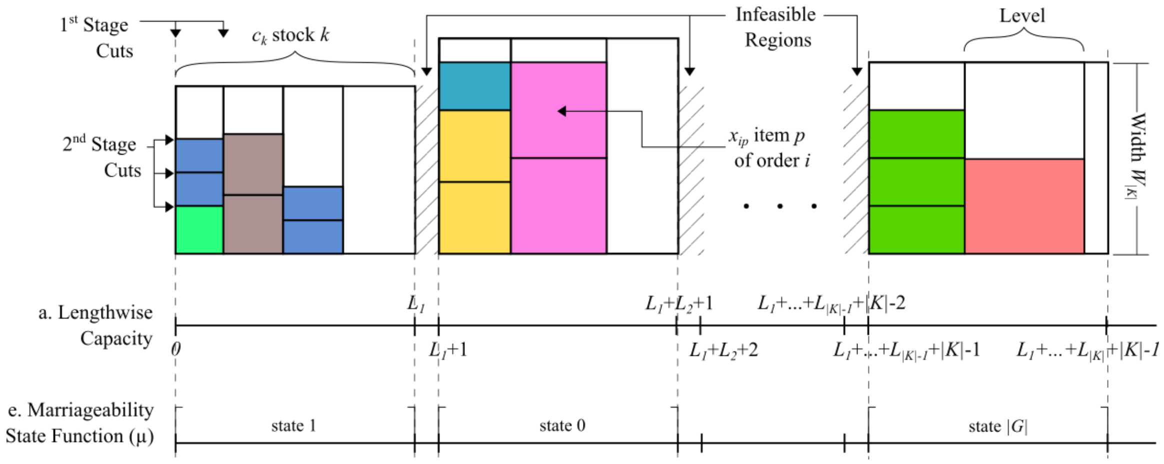


Figure 5.1: Illustration of the CP_{SR} model with order-to-order marriageability. The marriageability state values in this example are arbitrarily assigned.

The guillotine state function, widthwise capacity, and the partition number capacity are the same as in the 2SCSP-FF formulation (Figure 4.1 b, c, and d) and are not shown.

5.2 Exact Approaches

In this section, we extend the top exact approaches proposed in Chapter 4: the single resource model (CP_{SR}), the assignment-based MILP model (MIP_{AS}), and the counting-based MILP model (MIP_{CO}). For the MILP models, we formulate their marriageability requirements from two perspectives: a conflict-based perspective where conflicting orders cannot be assigned to the same stock, and a treatment-based perspective where each stock can only take on one set of treatments. Table 5.1 provides a summary for the perspectives.

Notably, from a graphical perspective, the conflict-based formulation is derived from the conflict graph, where orders are nodes and edges represent the conflicting relationships; the combination-based formulation is obtained from its complement, the marriageability graph, where orders with identical properties form a clique.

5.2.1 The Single Resource CP Formulation

Items of different orders assigned to the same stock are treated to have the same properties, so a finite number of orders implies a finite number of treatments for each stock. Here, we associate each type of treatment with a state value in the state function μ (Figure 5.1e). A feasible solution to the 2SCSP-FFM then requires the state values of items to be the same as the state value of the stock, which is formalized using the following two sets of constraints. Constraint (5.1b) restricts a constant state value over each stock interval c_k, thereby ensuring that each stock is treated to take on only one set of order properties. Next, constraint (5.1a) restricts the state value over the item interval variable x_{ip} to always equal $\gamma \in G \mid i \in N_\gamma$. Thus, item p belonging to order i can only be cut from stocks that are treated to have its desired properties. We note that the boolean arguments in both constraints are important. In constraint (5.1b), since there can be multiple levels of items in a stock, the starting and ending coordinates of the item intervals do not need to align with coordinates of the state intervals. In constraint (5.1a), however, both the starting and ending coordinates of the stock interval need to be aligned with the coordinates of a state interval in the marriageability state function so that the treatment throughout the stock is consistent. Since these constraints restrict how the stock is treated, we classify this model as a treatment-based formulation.

$$
\begin{array}{lll r}
\min & (4.1a) & (CP_{SR}) & \\
\text{s.t.} & \textsc{AlwaysEqual}(\mu, x_{ip}, \gamma, False, False) & \forall i \in N_\gamma, \gamma \in G, p \in C_i & (5.1a) \\
& \textsc{AlwaysConstant}(\mu, c_k, True, True) & \forall k \in K & (5.1b) \\
& \mu : \textsc{StateFunction}() & & (5.1c) \\
& (4.1b) - (4.1m) & &
\end{array}
$$

5.2.2 Assignment-based MILP Model

We investigate two marriageability formulations extending MIP_{AS}, one from the conflict-based perspective and another from the treatment-based perspective. In both formulations, we recall the definition of the binary decision variables x_{ijkl}, which equals 1 if the l^{th} piece of the j^{th} level of the k^{th} stock is assigned to i^{th} order. The base formulation of MIP_{AS} is in Section 4.6.

Conflict-based Formulation The conflict-based formulation, denoted as MIP^C_{AS}, explicitly models the conflicts between non-marriageable orders. It introduces binary variables z^c_{ik}, which, in conjunction with Constraints (5.2b) and (5.2d), detect the presence of order i on stock k. Notably, constraint (5.2b) only needs to include the first partition on a level because of the symmetry-reducing constraint (4.5a), which requires the smaller indexed partition to be assigned first. Finally, each stock, after treatment, can only be used to fulfill at most one of two orders with conflicting properties. In other words, given two conflicting orders i and i', at most one of z^c_{ik} and $z^c_{i'k}$ can be present in a given stock k. Formalized in Constraint (5.2c), this relation can be captured using the $SOS1$ set, which ensures that at most one of the variables in the set is non-zero.

$$\begin{aligned}
\min \quad & (4.6a) && (MIP^C_{AS}) && (5.2a)\\
\text{s.t.} \quad & z^c_{ik} \geq x_{ijk0} && \forall i \in N, j \in J_k, k \in K && (5.2b)\\
& SOS1(z^c_{ik}, z^c_{i'k}) && \forall (i, i') \notin M, k \in K && (5.2c)\\
& z^c_{ik} \in \{0,1\} && \forall i \in N, k \in K && (5.2d)\\
& (4.6b) - (4.6o) && &&
\end{aligned}$$

Treatment-based Formulation The treatment-based formulation of MIP_{AS}, denoted as MIP^T_{AS}, ensures that each stock is only treated once. Since there is a finite number of possible treatments, we introduce binary variables z^t_{gk} that only equals 1 if stock k is treated to take on a set of properties represented by $g \in G$. Constraint (5.3b) and (5.3d) associates z^t_{gk} with x_{ijkl} so that $z^t_{gk} = 1$ only if some item belonging to an order $i \in N_g$ with properties g is assigned to stock k. In particular, $p_g = \max_{i \in N_g} |P_i|$ is an upper bound on the number of items belonging to an order demanding properties g in a level, $|J_k|$ is the maximum number of levels in stock k, and $p_g|J_k|$ is an upper bound on the number of items demanding properties g across all levels in stock k. Constraint (5.3c) captures the uniqueness of treatment on each stock. In particular, each stock can only take on one combination of treated properties, so there can only be at most one non-zero z^t_{gk} for each stock k. For each stock, we can characterize the set of the corresponding z^t_{gk} variables as a $SOS1$ set.

$$\begin{aligned}
\min \quad & (4.6a) && (MIP^T_{AS}) && (5.3a)\\
\text{s.t.} \quad & p_g|J_k|z^t_{gk} \geq \sum_{i \in N_g, j \in J_k} x_{ijk0} && \forall k \in K, g \in G && (5.3b)\\
& SOS1(z^t_{gk} \forall g \in G) && k \in K && (5.3c)\\
& z^t_{gk} \in \{0,1\} && \forall g \in G, k \in K && (5.3d)\\
& (4.6b) - (4.6o) && &&
\end{aligned}$$

5.2.3 Counting-based MILP Model

The counting-based MILP model uses binary decision variables x_{ijkl}, which equal 1 if and only if l partitions belonging to order i is assigned to level j of stock k. Its formulation is similar to that of MIP_{AS}, so we can reuse the variables and constraints proposed to address marriageability from both the conflict-based and the treatment-based perspectives. The base formulation of MIP_{CO} is described in Section 4.8.

Conflict-based Formulation The conflict-based formulation, denoted as MIP^C_{CO}, again explicitly restricts the assignment of conflicting orders on the same stock. Its formulation is identical to MIP^C_{AS}, except for Constraint (5.4b), which relates z^c_{ik} and MIP_{CO}'s x_{ijkl}. Here, for each order i in level j of stock k, x_{ijkl} is non-zero for at most one of the $l \in P_i$ counts of partitions, so the sum $\sum\limits_{l \in P_i} x_{ijkl}$ can only be 0 or 1.

$$\begin{aligned}
&\text{minimize} && (4.6a) && (MIP^C_{CO}) && (5.4a)\\
&\text{s.t.} && z^c_{ik} \geq \sum_{l \in P_i} x_{ijkl} && \forall i \in N, j \in J_k, k \in K && (5.4b)\\
& && (5.2c), (5.2d) && &&\\
& && (4.6d) - (4.6h), (4.6l) - (4.6o), (4.8b) - (4.8g) && &&
\end{aligned}$$

Treatment-based Formulation The treatment-based formulation, MIP^T_{CO}, again leverages the fact that each stock can only treated once. The definition of the binary variable z^t_{gk} is identical to those in MIP^T_{AS}. Constraint (5.5b) replaces Constraint (5.3b) due again to the difference in variable definitions.

$$\begin{aligned}
&\text{minimize} && (4.6a) && (MIP^T_{CO}) && (5.5a)\\
&\text{s.t.} && p_g |J_k| z^t_{gk} \geq \sum_{i \in N_g, j \in J_k, l \in P_i} l \times x_{ijkl} && \forall k \in K, g \in G && (5.5b)\\
& && (5.3c), (5.3d) && &&\\
& && (4.6d) - (4.6h), (4.6l) - (4.6o), (4.8b) - (4.8g) && &&
\end{aligned}$$

5.3 The First-fit based Heuristic

The two-stage first-fit based heuristic iteratively packs items into open levels with the lowest available stock index and then solves a MILP to determine the packed items' lengths. The base heuristic, before packing an item, checks if it can fit into a level geometrically. With the marriageability requirements, before verifying the geometric fit, the heuristic examines if the item's properties conflict with properties of any items previously packed into the stock. If there is a conflict, the heuristic moves on to the next stock. If an item of an order cannot be packed into any stocks and other already-packed items belonging to the order cannot satisfy the minimum total area demanded, the heuristic terminates and assumes infeasibility. Like in the base heuristic, after all orders are packed into the stocks, the

Step	**Models**	**Modifications**
S1 (Congruent Groups)	CC^{S1} and CP^{S1}	Include marriageability in the set of cannot-link relations $\mathcal{C}$
S2 (Group to Levels)	MIP^{S2}	None
S3 (Levels to Stocks)	MIP^{S3}_{simp}	Add constraints expressing the cannot-link relations

Table 5.2: Changes made to select sequential heuristic configurations to tackle 2SCSP-FFM.

length of each item is determined by solving a linear program defined in Model 4.9. We do not need to modify the this model because the solution from the first stage requires the items packed in the same stock to be marriageable with each other.

5.4 The Sequential Heuristics

The sequential heuristic framework aims to construct high-quality solutions by sequentially solving three subproblems: finding congruent orders, packing grouped orders into level patterns, and assigning level patterns to stocks (Figure 4.3). We investigated different approaches for each step in Section 4.4, and found the top-performing configurations to be $S : CP^{S1} + MIP^{S2} + MIP^{S3}_{simp}$ and $S : CC^{S1} + MIP^{S2} + MIP^{S3}_{simp}$. Here, we modify these configurations to tackle the marriageability requirement. The summary of these changes is shown in Table 6.1.

5.4.1 Finding Congruent Groups

The first step in the base heuristic finds congruent orders with overlapping length intervals by computing a lower bound on the number of clusters and solving a clustering problem using either constrained clustering (CC^{S1}) or constraint programming (CP^{S1}). Both procedures rely on the set of cannot-link pairwise relations, $\mathcal{C}$, which, in the base heuristic, represents pairs of orders that have non-overlapping length intervals. As the marriageability requirements introduce additional conflicts between orders, we augment $\mathcal{C}$ to also include pairs of orders (i, i') that belong to different marriageable groups (i.e. $M_{ii'} = 0$). Then, with the updated $\mathcal{C}$, we can simply initialize the number of clusters using the Hoffman's lower bound, update the cannot-link constraints in CP^{S1}, and check for the order-to-cluster assignments in CC^{S1} according to the constrained algorithm in Section 4.4.

5.4.2 Dividing Groups into Levels

Given the congruent groups found by the first step, the second step of the heuristic distributes items in each order in the same group into levels of varying width. Here, no modification is needed for the second-step models proposed in Section 4.4, since orders in each congruent group must be marriageable with each other.

5.4.3 Assigning Levels To Stocks

Given a set of level patterns, the third step of the heuristic packs the levels to stocks. The top two configuration of 2SCSP-FF both used MIP_{simp} as their third-step solver, so, in this section, we only extend this MILP model. Because the number of level patterns is typically much larger than the number of orders, a conflict-based formulation that restricts pairs of conflicting levels introduces significantly more constraints than it would in MIP^{T}_{AS}. Instead, we investigate a treatment-based formulation of MIP_{simp} that ensures each stock can only be treated to take on one set of order properties. We recall that z_{jk} is a binary variable from Section 4.4.3.1.2 which equals 1 if level pattern j is assigned to stock k, and z^{t}_{gk} is a binary variable from Section 5.2 which equals 1 if stock k takes on properties represented by $g \in G$. We reuse these variables and all constraints, except that Constraint (5.2b) is replaced by Constraint (5.6b) to associate z_{jk} with z^{t}_{gk}.

$$\min\ (4.16a) \qquad (MIP^{S3}_{simp}) \tag{5.6a}$$

$$|J_k| z^{t}_{gk} \geq \sum_{j \in S_g} z_{jk} \qquad \forall k \in K, g \in M \tag{5.6b}$$

$$(5.2c) - (5.2d), (4.16b) - (4.16i)$$

5.5 Numerical Results

We present our experimental results on 2SCSP-FFM. For each MILP model, we only display the marriageability formulation with the superior average solution quality after penalizing the instances for which the approach failed to find a feasible solution within time and memory limits. We again abbreviate $S : CP^{S1} + MIP^{S2} + MIP^{S3}_{simp}$ and $S : CC^{S1} + MIP^{S2} + MIP^{S3}_{simp}$ as S_{CP} and S_{CC}.

Model	# New Variables	# New Constraints
CP_{SR}	1	$\sum_{g \in G, i \in N_g} \lvert C_i\rvert + \lvert K\rvert = \sum_{i \in N} \lvert C_i\rvert + \lvert K\rvert$
MIP^{C}_{AS}	$\lvert N\rvert \lvert K\rvert$	$\sum_{k \in K} \lvert N\rvert \lvert J_k\rvert + (\lvert N\rvert \lvert N-1\rvert/2 - \lvert M\rvert)\lvert K\rvert$
MIP^{T}_{AS}	$\lvert G\rvert \lvert K\rvert$	$\lvert G\rvert \lvert K\rvert + \lvert K\rvert$
MIP^{C}_{CO}	$\lvert N\rvert \lvert K\rvert$	$\sum_{k \in K} \lvert N\rvert \lvert J_k\rvert + (\lvert N\rvert \lvert N-1\rvert/2 - \lvert M\rvert)\lvert K\rvert$
MIP^{T}_{CO}	$\lvert G\rvert \lvert K\rvert$	$\lvert G\rvert \lvert K\rvert + \lvert K\rvert$

Table 5.3: Exact number of variables and constraints additionally introduced to describe the marriageability requirement.

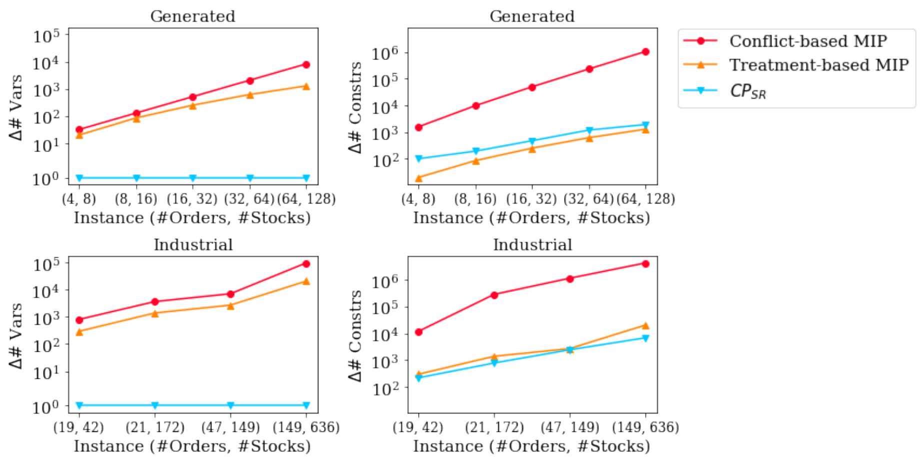


Figure 5.2: Empirical number of variables and constraints additionally introduced to describe the marriageability requirement per instance size.

5.5.1 Monolithic Model Comparison

Table 5.3 formalizes the number of variables and constraints introduced to describe the marriageability requirement. CP_{SR} adds only one new variable, the state function, but the number of its marriageable constraints is dependent on both the maximum possible number of items and the number of stocks. The change in the number of variables and constraints of the MILP models is the same for each marriageability formulation, regardless of its base model. The conflict-based formulation uses at least as many variables as the treatment-based formulation, since the number of orders is always greater than the number of treatment properties, except possibly when the properties of orders are all different. The constraints in the conflict-based formulation can be segmented into two functionalities: those relating different decision variables (Constraints 5.2b and 5.4b) and those forbidding conflicts (Constraint 5.2c). Counting the former is straightforward, as it is the product of the number of orders and the number of possible levels on all stocks. The number of the latter can be described by the number of missing edges in the marriageability graph times the number of stocks. The number of missing edges is calculated by the number of edges, $|N||N-1|/2$, in a fully-connected graph whose vertices are orders, minus the number of edges in the marriageability graph, $|M|$. The number of constraints in the treatment-based formulation is much simpler. Notably, $|G|$, the number of types of treatments, is equivalent to the number of cliques in the marriageability graph. In contrast to the conflict-based formulation, the size of the treatment-based formulation is not affected by the increase in the number of orders or in the number of possible levels in a stock. The number of variables and constraints in both formulations is reliant on the number of stocks. CP_{SR} is the only model whose number of marriageability variables is independent of the instance size.

Figure 5.2 describes the empirical number of additional variables and constraints instantiated for each instance size. The number of variables introduced in CP_{SR} is indeed

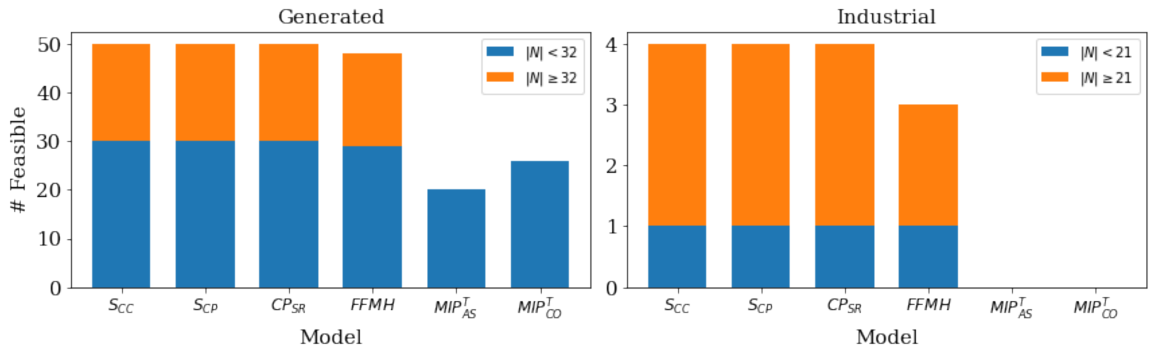


Figure 5.3: Number of generated and industrial instances with a feasible solution by each model for 2SCSP-FFM. None of the MILP formulations found a feasible solution to the industrial instances.

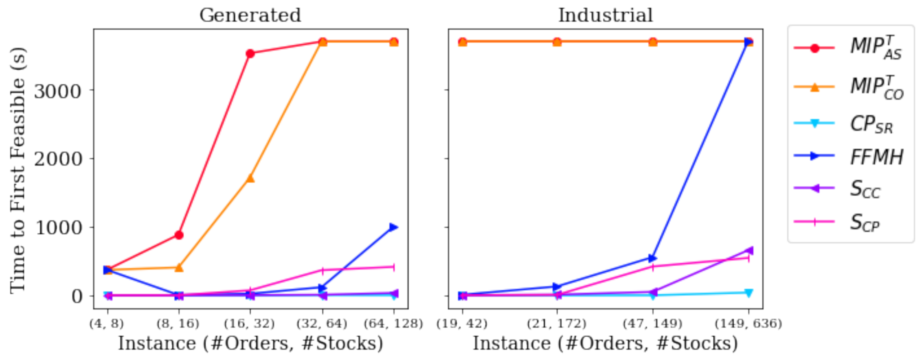


Figure 5.4: The average run time required to find a feasible solution for a given model at an instance size for 2SCSP-FFM.

orders of magnitude fewer than the MILP models. However, CP_{SR} does not always use fewer constraints than the treatment-based MILP formulations, as the total possible number of items can be large and is independent of the number of cliques in the marriageability graph. If the number of unique types of stock treatments is substantial, CP_{SR} is preferable; if orders demand many items, the treatment-based MILP formulations may still be desirable. Both CP_{SR} and the treatment-based MILP models use significantly fewer variables and constraints than the conflict-based MILP models. Nonetheless, including the rest of the formulation, CP_{SR} still uses significantly fewer variables and constraints than the alternative models.

We do not compare the memory usage of the models, because we can not access the memory usage of the MILP models via the CPLEX solver. The increase in CP_{SR}'s model size in memory before search averaged across the generated instances is under 10 MB, and that averaged across the industrial ones is around 120 MB.

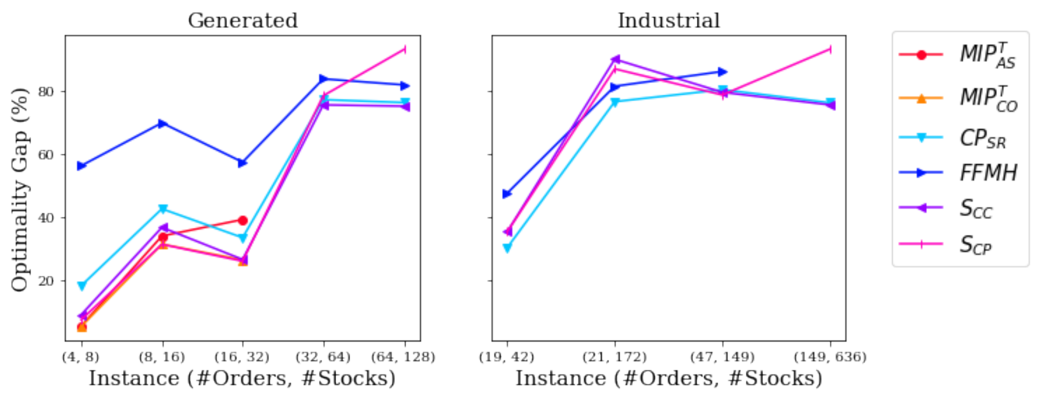


Figure 5.5: % optimality gap for generated and industrial instances for 2SCSP-FFM. Instances that are not solved by an approach are not included in that approach's measure.

5.5.2 Feasibility Analysis

Figures 5.3 and 5.4 display the number of feasible solutions found, and the time to feasibility, respectively. Only CP_{SR}, S_{CC}, and S_{CP} found a feasible solution to all generated and industrial instances. Notably, out of the model-based approaches, CP_{SR} is the only one that found a feasible solution to the generated instances with $|N| \geq 16$ and the industrial ones with $|N| \geq 21$. CP_{SR}'s ability to quickly find a feasible solution for 2SCSP-FF was also observed here, as it required less than 150 seconds to find its first feasible solution for the largest industrial instance. S_{CP} and S_{CC}, the second and third fastest for that instance, needed at least 400 seconds. With the solution space being more constrained, $FFMH$ did not find a feasible solution to two of the generated instances and timed-out solving the largest industrial instance. MIP^T_{CO} found more feasible solutions than any other MILP formulations for the generated instances, and none of the MILP models found a feasible solutions for the industrial ones.

5.5.3 Overall Quality

Figures 5.5 displays the optimality gap of the final solution found by the solution approaches computed using Equation (4.34). Instances that an approach did not find a feasible solution are not included. Out of the exact approaches (red, orange, and cyan), MIP^T_{CO} found the highest quality solutions for the generated instances with $|N| \leq 16$, and CP_{SR} for the generated instances with $|N| \geq 32$ and all industrial instances.

Comparing the heuristics, the solution qualities of S_{CC} and S_{CP} are similar for all instances except the largest generated and the largest industrial instances, where S_{CC} is superior, thereby affirming our initial hypothesis that the randomized clustering process aids the discovery of high-quality solutions, especially when the instance size scales up. For all instances, $FFMH$ found worse solutions than CP_{SR}, S_{CC}, and S_{CP}. The gap between their solutions, however, gradually tightened as the instance size increased. This behaviour

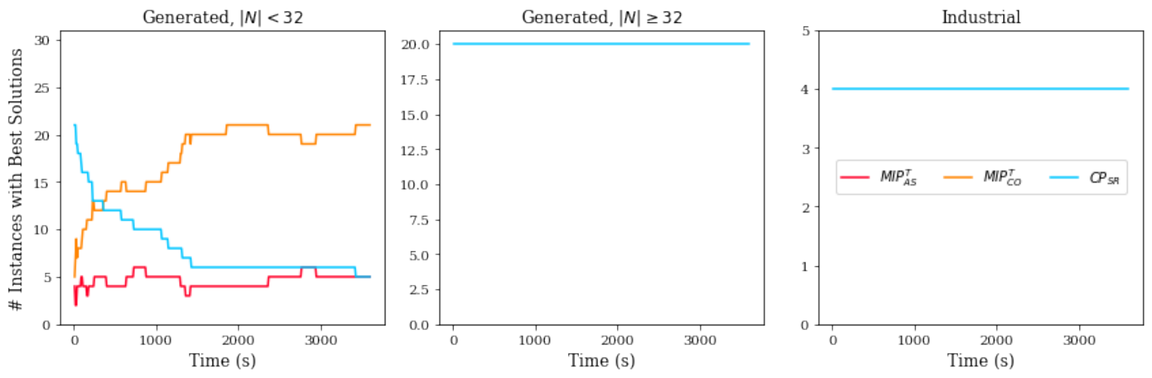


Figure 5.6: Number of instances for which an exact method found the best solution over runtime for 2SCSP-FFM.

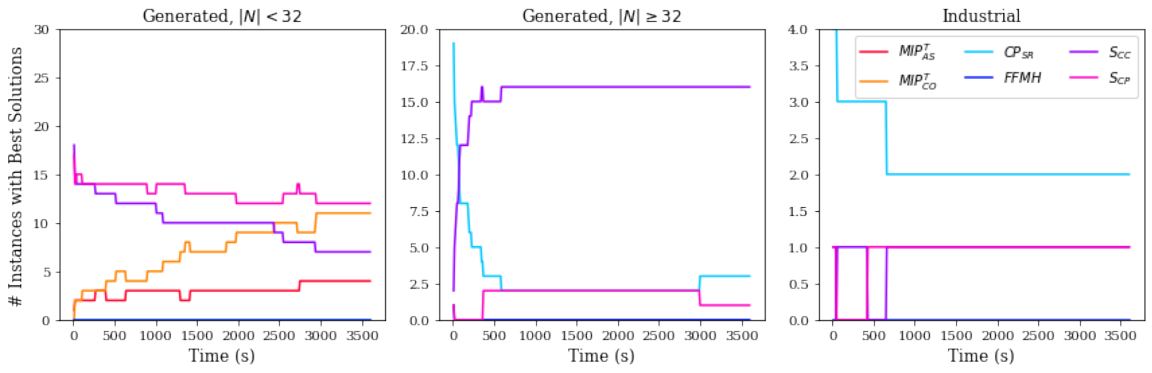


Figure 5.7: Number of instances for which any approach found the best solution over runtime for 2SCSP-FFM.

is also present in the industrial instances. CP_{SR} found the high-quality solutions for the industrial instances with $|N| \leq 21$. For the rest of the instances, its solutions remain competitive with the best performing approach, S_{CC}.

The lower bound used for all instances with $|N| \geq 19$ are produced from CP_{SR}. Like in Section 4.5.4, these bounds are non-trivial and are attributed to using the CP scheduling tools.

5.5.4 Runtime Analysis

Figures 5.6 and 5.7 display the relative performance of the solution approaches over the runtime. Out of the exact approaches, CP_{SR} found the highest number of best solutions in the first 400 seconds, but was superseded by MIP^T_{CO} in the remaining runtime. For the larger generated and industrial instances, CP_{SR} remains the only viable model-based approach. Comparing the exact approaches with the heuristic methods, for the smaller instances, we observe that S_{CC} and S_{CP} found the highest number of best solutions before the 2500th second, after which MIP^T_{CO} overtook S_{CC} and finished with approximately the same number as S_{CP}. Like in Section 4.5.5, MILP models require longer runtime to achieve similar performance with the sequential heuristics. For the large generated instances, CP_{SR} found the highest quality solutions before the 200th second, after which S_{CC} dominated. For the industrial instance, the relative solution quality between CP_{SR}, S_{CC}, and S_{CP} is steady after 1000 seconds, with CP_{SR} finding the highest number of best solutions.

5.6 Conclusion

In this chapter, we extended select models and heuristics to solve the 2SCSP with Flexible Item Length, Flexible Demand, and Marriageability. For the single resource constraint programming formulation, we develop a concise augmentation using a state function spanning the entire single resource horizon. We also investigate two marriageability formulations of the mixed-integer programming models, one from a conflict-based perspective and another from a treatment-based perspective, and modify both the first-fit-based heuristic and the algorithms in the sequential heuristics. In our experiment, the single resource model CP_{SR} introduces orders-of-magnitude fewer variables than MILP models to describe the marriageability requirement and remains the only model-based approach to scale effectively to the larger generated and industrial instances. It yielded solutions slightly weaker with the best sequential heuristic for the generated instances, but outperformed all other approaches for two of the four industrial instances. With the additional marriageability requirement, MILP models struggled to scale up, which can be attributed to the considerable number of variables and constraints that are introduced.

Chapter 6

2SCSP with Flexible Item Length, Flexible Demand, Marriageability, and Scheduling Costs

This chapter extends the top-performing models and heuristics proposed for 2SCSP-FF for the full Two-Stage Cutting Stock Problem with Flexible Item Length, Flexible Demand, Order-to-order Marriageability, and Scheduling Costs (2SCSP-FFMS). We develop expressions for each model to describe the scheduling costs and conduct experiments on this packing-scheduling hybrid use case. Our results show that, with this more complicated objective function, the single resource CP model performed worse than it did on previous problems, while the performance of the sequential heuristics remained steady.

6.1 Problem Description

In 2SCSP-FFMS, stock k can only be cut after its availability date a_k, and order i has a due date d_i. Any items cut before their due dates incur an inventory cost c_{inv} for every unit area per unit time; any items cut after their due dates incur tardiness cost c_{tard} for every unit area per unit time. Typically, $c_{tard} >> c_{inv}$, so tardiness is more heavily penalized. To simplify the objective expression, we consider an approximation of the scheduling cost: we assume all stocks are cut as soon as they are available. Formally, the approximated inventory cost per unit area per unit time is $c_{inv}w_il_i \times \max(0, d_i - a_k)$, and the approximated tardiness cost per unit area per unit time is $c_{tard}w_il_i \times \max(0, a_k - d_i)$. Since an item of order i assigned to stock k can only incur either inventory cost or tardiness cost, its scheduling cost per unit area is in fact a constant, which we denote as $\mathcal{T}_{ik}$ and is formally defined in Equation 6.1.

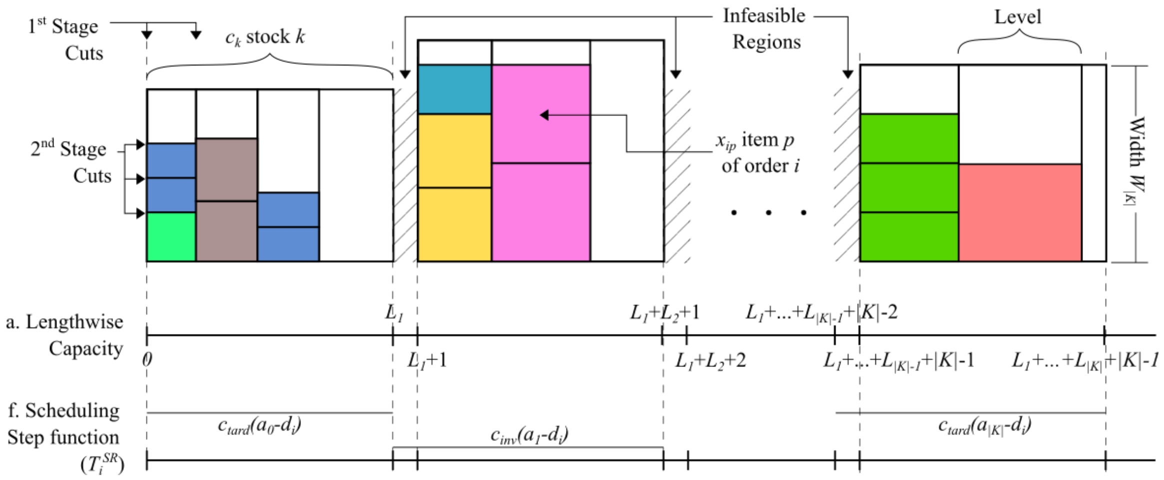


Figure 6.1: Illustration of the CP_{SR} model with a step function representing the scheduling costs associated with a given order. The guillotine state function, widthwise capacity, the partition number capacity, and the marriageability state function are the same as in the 2SCSP-FFM formulation (Figure 4.1 b, c, and d and Figure 5.1 e) and are not shown.

$$\mathcal{T}_{ik} = \begin{cases} c_{tard}(a_k - d_i) & \text{if } a_k > d_i \\ c_{inv}(d_i - a_k) & \text{if } a_k < d_i \\ 0 & \text{otherwise} \end{cases} \tag{6.1}$$

6.2 Exact Approaches

6.2.1 The Single Resource CP Formulation

To quantify the scheduling costs, we need information about both the order an item belongs to and the index of the stock. In the single resource model, while the former can be easily obtained via the index of the item's interval variable, the index of each stock is implied in the coordinates of the variables. Consequently, we define a new parameter, $\mathcal{T}_{i\tau}^{SR}$, that represents the scheduling cost per unit area incurred by assigning order i to stock κ such that, on the single resource horizon, the coordinate τ is within the lengthwise interval defined by stock κ. We formally define $\mathcal{T}_{i\tau}^{SR}$ in Equation 6.2. We note that, given an item of order i and its positioning τ, $\mathcal{T}_i^{SR}(\tau)$ yields a constant.

$$\mathcal{T}_i^{SR}(\tau) = \mathcal{T}_{i\kappa} \quad \text{, where } \kappa \in K \mid \sum_{k=1}^{\kappa-1} L_k \leq \tau \leq \sum_{k=1}^{\kappa} L_k \tag{6.2}$$

The scheduling costs incurred for item p of order i is then $\mathcal{T}_i^{SR}(\text{StartOf}(x_{ip}))$, where x_{ip} is the item's interval variable on the single resource horizon and $\text{StartOf}(x_{ip})$ is its leftmost lengthwise coordinates. As we vary τ along the single resource horizon horizon, $\mathcal{T}_i^{SR}(\tau)$ becomes a step function (Figure 6.1 f), and can be natively expressed in CP Optimizer using the PIECEWISELINEAR functionality [134].

The total scheduling costs incurred by items over all orders, denoted as $\mathcal{T}_{total}$, is expressed in Equation 6.3 and is added to Objective Function (4.1a) when solving instances of 2SCSP-FFMS. Here, $w_i\textsc{SizeOf}(x_{ip})$ indicates the total area of each item p of order i.

$$\mathcal{T}_{total} = \sum_{i \in N} \sum_{p \in C_i} w_i \textsc{SizeOf}(x_{ip}) \mathcal{T}_i^{SR}(\textsc{StartOf}(x_{ip})). \tag{6.3}$$

6.2.2 Assignment-based MILP Model

The total scheduling cost over all items, denoted as $\mathcal{T}_{total}$, is described in Equation 6.4. We recall that variable a_{ijkl} representing the area fulfilled for each partition l belonging to order i on level j of stock k is indexed by both order and stock, and so we can compute the total cost by multiplying the a_{ijkl} with the per-unit-area scheduling costs. We added $\mathcal{T}_{total}$ to the packing-based objective function (4.6a) when solving instances of 2SCSP-FFMS. The marriageability formulations of MIP_{AS} without scheduling costs are found in Section 5.2.2.

$$\mathcal{T}_{total} = \sum_{k \in K} \sum_{j \in J_k} \sum_{l \in P_i} \sum_{i \in N} a_{ijkl} \mathcal{T}_{ik} \tag{6.4}$$

6.2.3 Count-based MILP Model

Like the assignment-based MILP model, adding the scheduling costs to the counting-based MILP model involves multiplying the per-unit-area scheduling cost with the item areas. Since the definition of the item-area variable a_{ijkl} is the same as the assignment-based model, we can reuse the definition of $\mathcal{T}_{total}$ from Equation 6.4 and add it to the original packing objective function (4.6a). The marriageability formulations of MIP_{CO} without scheduling costs are described in Section 5.2.2.

6.3 The First-fit-based Heuristic

In model-based approaches, expressing scheduling cost involves extending the objective functions; however, the first-fit-based heuristic, $FFMH$, lacks an declarative objective function, so instead, we need to preprocess the input parameters to guide the solution construction. Since incurring inventory costs is much cheaper than incurring tardiness costs, the heuristic should avoid the latter. A simple solution is to cut orders from stocks as early as possible, so that, in the worst case, cut orders are stored in the inventory for a long time. Translating this strategy to the first-fit-based heuristic, we can sort the orders by their due dates and the stocks by their availability dates, both in ascending order. As a result, orders with an early due date are packed into the earliest available stocks.

As the base heuristic in Section 4.3 already sorts the orders and stocks by their physical dimensions, the lexicographic order to which the parameters are sorted directly contributes to the lexicographic importance of the objectives. In our experiments, we find that solutions typically incur more scheduling costs than the cost of wasteful packing. So, we first sort the orders and stocks by their schedule-related parameters. Then, orders with identical due dates are sorted by their widths and then length interval sizes, and stocks with identical availability dates are sorted by their widths and then lengths. This sorting scheme results in a greater emphasis on reducing scheduling costs.

6.4 The Sequential Heuristics

In this section, we modify the algorithms used in the steps of the top-performing configurations in 2SCSP-FF, $S : CP^{S1} + MIP^{S2} + MIP^{S3}_{simp}$ and $S : CC^{S1} + MIP^{S2} + MIP^{S3}_{simp}$. The summary of these changes is shown in Table 6.1.

Step	**Component**	**Modifications**
S1 (Congruent Groups)	Initial # of Clusters	$k = \|N\|$ clusters are initialized. k is decremented over time.
	CC^{S1}	Cluster using order due dates as input features
	CP^{S1}	New model clustering orders based on their due dates
S2 (Group to Levels)	MIP^{S2}	None
S3 (Levels to Stocks)	MIP^{S3}_{simp}	Add scheduling cost to objective function.

Table 6.1: Changes made to select sequential heuristic configurations to tackle 2SCSP-FFMS.

6.4.1 Finding Congruent Groups

The first step segments orders into congruent groups. Adding scheduling costs does not alter the definition of congruency, so we can reuse the set of cannot-link relations $\mathcal{C}$ from Section 5.4, which considers both overlapping length intervals and order-to-order marriageability. However, orders in a congruent group should have similar due dates, as they will be packed into the same level in the subsequent step. Thus, we modify our algorithms accordingly.

Finding the Initial Number of Clusters

The first step of the heuristic starts with finding an initial number of clusters, k. In the base heuristic, we initialized a lower bound in the hope that fewer groups can lead to larger groups, which, in turn, increases the packing efficiency. However, with the scheduling costs, having as few clusters as possible does not necessarily yield good solutions, as orders with varying due dates may be forced together. In our experiments, we find that the costs associated with scheduling are often higher than those with packing, so we let the initial k be as large as possible (i.e. $k = |N|$), thereby assigning each order to its own cluster. The hope is that, in the later steps of the heuristic, the due dates of orders in the level patterns are identical, so matching them to a stock having an appropriate available date is easier.

When the base heuristic increases k by 1, we decrement k by 1 to form congruent groups containing more orders.

Constrained Clustering

The Constrained K-Means algorithm, CC^{S1}, clusters orders based on the similarity of their features while respecting a set of cannot-link relations, which, in 2SCSP-FFM, concerns non-overlapping length intervals and order-to-order marriageability. With the scheduling costs, we cluster orders around their due dates (Figure 6.2), hypothesizing that orders with similar due dates are more likely to be assigned to the same stock. Since the cannot-link relations did not change, we reuse its definition in Section 5.4.1.

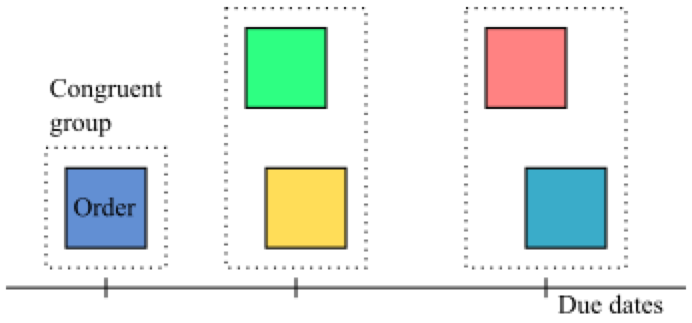


Figure 6.2: Finding congruent groups by clustering around order due dates.

Constraint Programming Formulation

To tackle the 2SCSP-FFMS, we adapt CP^{S1} so that orders with similar due dates are clustered together. The new formulation is expressed in Model 6.5, in which we introduce an array of integer variables, t, where the r^{th} element is the integer variable t_r representing the centroid of cluster r. Instead of the total number of clusters, we minimize the residual sum of squares of the orders' due dates, penalizing clusters with large variance in Objective Function 6.5a. With the updated $\mathcal{C}$, we can also reuse Constraint (4.11b) to express marriageability requirements.

$$\begin{aligned}
\min \quad & \sum_{i \in N} (d_i - t_{x_i})^2 && && \text{(6.5a)} \\
\text{s.t.} \quad & t_r \in \{0, \ldots, \max_{i' \in N} d_{i'}\} && \forall r \in \{1, \ldots, k\} && \text{(6.5b)} \\
& t_r \geq t_{r+1} && r \in \{1, \ldots, k-1\} && \text{(6.5c)} \\
& (4.11b), (4.11c) && && \text{(6.5d)}
\end{aligned}$$

We also reuse the solution cut in Expression 4.12, as the new variables t_r are unrelated to the order-to-cluster assignment.

6.4.2 Dividing Groups into Levels

Given congruent groups found by the first step, the second step of the heuristic distributes items in each order in the same group to a set of levels of varying width. Here, no modification is needed for the second-step models proposed in Section 4.4, since orders in a congruent group share similar due dates.

6.4.3 Assigning Levels To Stocks

MIP^{S3}_{simp} is the preferred approach for both of the top-performing configurations to assign level patterns generated from the previous step to stocks, so we add the scheduling cost expressed in Equation (6.6) to its objective function (4.16a). We recall that z_{jk} is a binary variable that equals 1 if level pattern j is assigned to stock k, and d_{jk} is a continuous variable that represents the length surplus exceeding ρ_j^{min} for level pattern j assigned to stock k. The additional scheduling cost added to the objective function is expressed as the following:

$$\mathcal{T}_{total} = \sum_{k \in K} \sum_{j \in S_k} \sum_{i \in N} n_{ij} w_i T_{ik} (\rho_j^{min} z_{jk} + d_{jk}) \tag{6.6}$$

where n_{ij} is the number of items belonging to order i present in level pattern j.

6.5 Numerical Results

We present our experimental results on 2SCSP-FFMS. As in Chapter 5, we only display the marriageability formulation with the superior average solution quality after penalizing the instances for which the approach failed to find a feasible solution within time and memory limits. Top-performing sequential heuristic configurations $S : CP^{S1} + MIP^{S2} + MIP^{S3}_{simp}$ and $S : CC^{S1} + MIP^{S2} + MIP^{S3}_{simp}$ are abbreviated as S_{CP} and S_{CC}.

6.5.1 Feasibility Analysis

The number of feasible solutions (Figure 6.3) found for each approach and their time to feasibility (Figure 6.4) with the additional scheduling costs are similar to those without the scheduling costs. Only S_{CP}, S_{CC}, and CP_{SR} found a feasible solution to all generated and industrial instances, all of which needed significantly less time than the other approaches. Notably, S_{CC} improved on its results for 2SCSP-FFM, and we suspect this is caused by the change in the input features in the first step of the heuristic. CP_{SR} again uses significantly less time than all other approaches to find a feasible solution for the largest industrial instance, resulting in a 500-second margin over the second-fastest approach, S_{CP}. Both

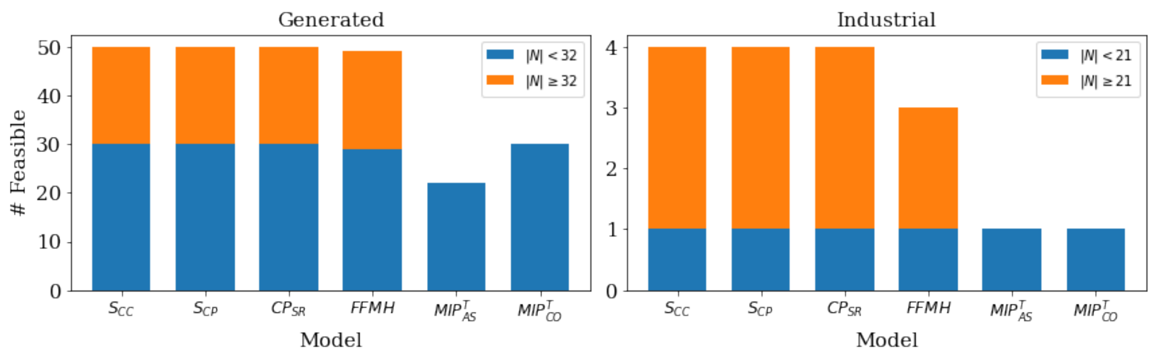


Figure 6.3: Number of generated and industrial instances with a feasible solution by each model for 2SCSP-FFMS.

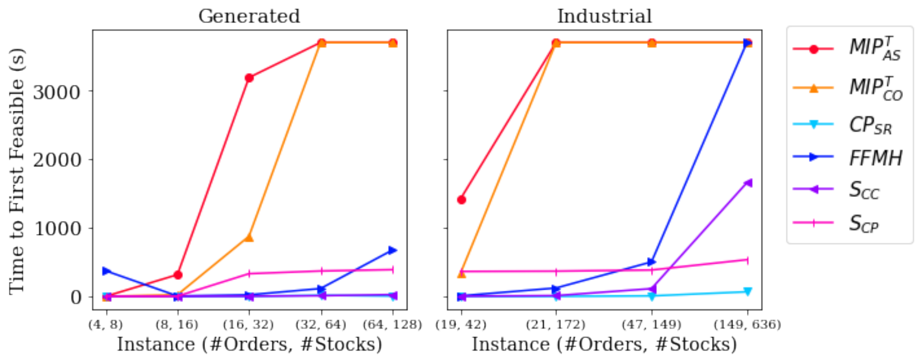


Figure 6.4: The average run time required to find a feasible solution for a given model at an instance size for 2SCSP-FFMS.

MIP^T_{AS} and MIP^T_{AS} found a feasible solution to one of the industrial instances, which is an improvement considering it could not find any solving 2SCSP-FFM.

6.5.2 Overall Quality

Figure 6.5 displays the average optimality gap of a solution compared with the best lower bound found across all approaches. Any unsolved instances are omitted from the visualization. With the additional objective term, different approaches behaved differently, yielding trends in optimality gaps inconsistent with those observed in 2SCSP-FFM. For the small generated instances, MIP^T_{AS} and MIP^T_{CO} yielded the best solutions and commanded a significant margin over the next best approaches, S_{CC} and S_{CP}. Both CP_{SR} and $FFMH$ struggled to find solutions that are competitive with the sequential heuristics. Increasing the size of the generated instances again leads to all model-based approaches, except CP_{SR}, becoming intractable. $FFMH$, which was lacking in 2SCSP-FFM, found solutions competitive with CP_{SR}. Both sequential heuristics exhibited strong performances on the industrial instances, outperforming both CP_{SR} and $FFMH$.

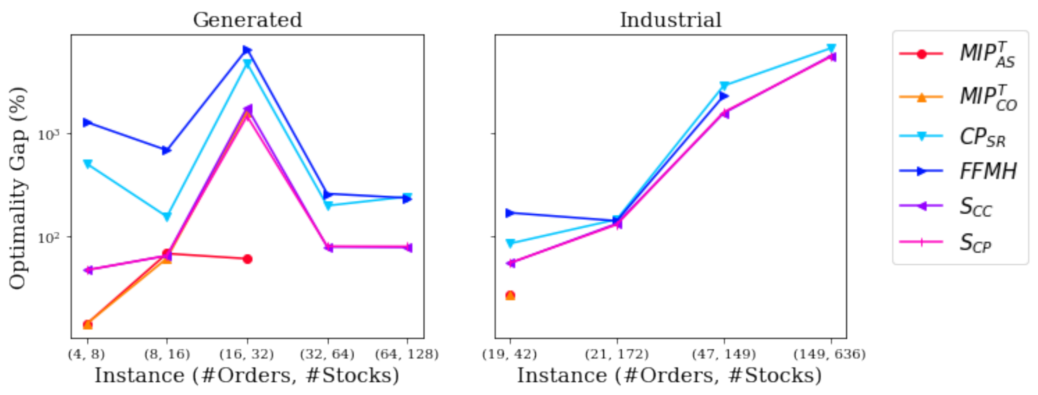


Figure 6.5: % optimality gap for generated and industrial instances for 2SCSP-FFMS. Instances that are not solved by an approach are not included in that approach's measure. The y-axis are log scaled.

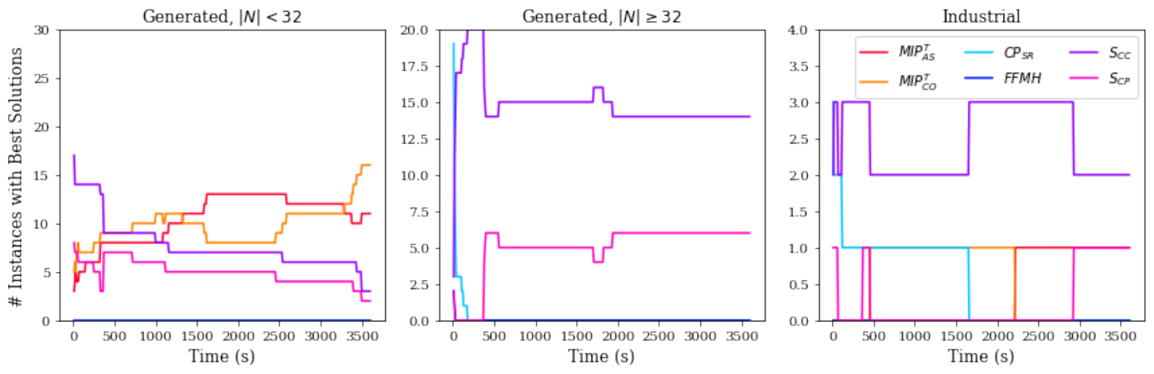


Figure 6.6: Number of instances for which an approach found the best solution over runtime for 2SCSP-FFMS.

6.5.3 Runtime Analysis

Figure 6.6 records the number of best solutions found by each approach throughout the runtime. For the smaller instances, S_{CC} was dominant for the first 500 seconds, after which the number of best solutions from both S_{CC} and S_{CP} dwindled, replaced instead by MIP^T_{AS} and MIP^T_{CO}. MILPs appear to be more suitable with this more complex objective function, as for the remainder of the runtime, they dominated this metric. For the larger generated instances, S_{CC} is superior to all other approaches for most of the runtime. S_{CP} undermined the dominance of S_{CC} after the 400th second by finding 5 solutions better than S_{CC}. CP_{SR}, the only model-based approach to find a feasible solution, was bested after the 50th second. The industrial instances followed a similar trend, where S_{CC} was better for most of the solutions.

6.6 Conclusion

In this chapter, we extended the exact models and the heuristics to reflect the industrial use case with scheduling costs (2SCSP-FFMS). The single resource CP model remains the most viable model-based approach, being the only one to effectively scale to the larger industrial instances. Its ability to quickly find feasible solutions translated from 2SCSP-FF and 2SCSP-FFM, as it required less time than even the heuristics. However, the single resource model struggled to find high-quality solutions with this more difficult objective function. In comparison, MIP^{T}_{AS} and MIP^{T}_{CO} are the preferred approach for the smaller generated and industrial instances, and S_{CC} and S_{CP} are the preferred ones for the larger instances. The fact that MILPs were much better than S_{CC} and S_{CP} for the smaller generated instances suggests that the sequential heuristic's modifications may be improved, which we leave for future work.

Chapter 7

Conclusion and Next Steps

7.1 Summary and Contributions

In this thesis, we explored applying scheduling-based constraint programming tools to solve variations of the novel Two-Dimensional Two-Stage Cutting Stock Problem with Flexible Length, Flexible Demand, Marriageability, and Scheduling Costs (2SCSP-FFMS). Stemming from the aluminum-metal industry, this problem is situated at the intersection of many hard problems. In solving large-scale instances of this problem, we showed that tools developed for scheduling in general-purpose constraint programming solvers can achieve state-of-the-art performance among model-based approaches and competitive performance with customized heuristics.

In Chapter 4, we examined a variant of 2SCSP-FFMS that ignores stock treatment, order properties, and scheduling costs. For the resulting problem, the Two-Dimensional Two-Stage Cutting Stock Problem with Flexible Length and Flexible Demand (2SCSP-FF), we develop a novel scheduling-based constraint programming model that adapted the recently proposed single resource transformation [34]. In particular, we noticed a connection between guillotine cuts and batch scheduling and introduced the use of state functions to model these cuts, a new idea to the literature. For comparison purposes, we also developed exact methods (three integer-based constraint programming models and two mixed-integer linear models) and heuristics (a first-fit-based heuristic and a three-stage sequential heuristic framework with 28 different configurations). Over the other exact methods, the single resource model demonstrates an order-of-magnitude advantage in memory usage before the search and the number of variables and constraints used. Accordingly, the single resource model is the only viable model-based approach to solving the industrial instances. Compared with the heuristics, the single resource model outperforms the simple first-fit-based heuristic and is competitive with the best sequential heuristic. Nonetheless, the simplicity and scalability of the single resource model coupled with its high-quality solutions remain desirable in the industrial setting.

Chapter 5 investigates a second variation of 2SCSP-FFMS that ignores any scheduling costs. In addition to 2SCSP-FF, this problem considers treating the stocks once before cutting so that the resulting partitions have desired properties. Items belonging to two orders with conflicting properties cannot be cut from the same stock, leading to order-to-order marriageability constraints. We augment the best performing approaches from Chapter 4: the single-resource constraint programming model, the mixed-integer programming models, and the sequential heuristic framework. We also adapt the first-fit heuristic as a baseline. The representational efficiency of the single resource model remains prominent. The single resource model outperforms other exact models for the larger instances and is competitive with the best sequential heuristic for some instances.

Chapter 6 studied the full 2SCSP-FFMS problem, which adds scheduling costs to the problem studied in Chapter 5. While the modelling efficiency of the single resource constraint programming model carried over, its solution quality is generally weaker than the best sequential heuristic. This noticeable struggle may be attributed to the more complex objective function but deserves further investigation.

In conclusion, our primary contribution, the scheduling-based single resource constraint programming model, is able to consistently outperform other exact approaches for the industrial-sized instances of variations of 2SCSP-FFMS. Its performance on these instances is also competitive with an ad-hoc multi-step heuristic depending on the objective function.

7.2 Future Work

We anticipate two main directions for future work: improvements to the proposed approaches and extensions to problems beyond 2SCSP-FFMS.

7.2.1 Improvements to Existing Work

This study is motivated by real-life industrial applications, so the instances used in the experiments are designed to reflect these use cases. While we have examined the performance of various approaches on these instances, another important research direction is to understand problem bottlenecks. For instance, it would be interesting to generate phase-transition diagrams while varying problem parameters to characterize the solubility of different instances [90]. Understanding these bottlenecks can lead to a more efficient and tailored search process for high-quality solutions.

We noted in Chapter 2 that problems with marriageability requirements cannot be directly decomposed into smaller problems due to a limiting number of stocks. Nonetheless, assuming an infinite number of stocks may be reasonable in scenarios where new stocks can be easily and timely procured. Under this assumption, 2SCSP-FFM and 2SCSP-FFMS can be decomposed into smaller subproblems, each of which assigns subsets of orders with identical demanded properties to an infinite set of stocks of varying types and can be solved independently. From this observation, we believe developing heuristics that relax the stock cardinality requirement to take advantage of these independent subproblems will be interesting. The solution to the relaxed problem can then serve as a starting point for post-processing algorithms, such as a neighbourhood search.

Another idea not explored in this thesis is the set-cover MILP formulation and column-generation-based heuristics. Traditionally, applying column-generation approaches to the set-cover formulation works well for optimizing 2D packing problems of fixed item dimensions: the master problem assigns cutting patterns to stocks, and the subproblem generates new patterns to improve the objective value of the master problem [53]. For our problem, the lengths of the items are flexible, so a secondary decision is required to localize the items' length values. This decision needs to be resolved in either the master problem or the subproblem, and the resulting effects on the overall computation deserve future investigation.

For large industrial instances, while the lower bounds computed by the single resource constraint programming model in Chapter 4, Chapter 5, and Chapter 6 are usable, externally developing a stronger lower bound can reduce the search space and help the solver prove optimality. Various lower bound procedures have been developed for two-dimensional packing problems, including dual feasible functions, contiguous relaxation, and state space relaxations [137], but extending them to the problem with guillotine cuts or flexible item sizes is nontrivial and lacking in literature. We leave the development of a stronger bound as future work.

For all variations of 2SCSP-FFMS, the mixed-integer programming models excelled for small instances, but experienced considerable scalability issues. A possible approach to circumvent these issues is to use a rounding heuristic on the linear relaxation of the mixed-integer linear programming models, as rounding heuristics have a track record alleviating computation strain [77].

An alternative approach to improve the scalability of the mixed-integer programming models is to integrate them in a large neighbourhood search. Large neighbourhood search essentially removes a set of values from a solution and asks a solver, such as a mixed-integer linear program or a constraint program, to reoptimize the partial solution and produce a feasible solution. Here, the large neighbourhood search can provide a smaller scope in which the mixed-integer linear programming models may be able to deliver strong performance. Unfortunately, this potential performance gain will be at the expense of losing a global lower bound and hence optimality guarantees. Examples of successful hybrids of large neighbourhood search and exact methods to solve packing-related problems include the work by Hojabri et al. [119] and by Lodi et al. [175].

Finally, the results in all three chapters showed that the best sequential heuristic configuration can at times outperform the single resource constraint programming model. Hence, we believe that further investigation into solving the subproblems within the heuristic, especially the problem of assigning levels to stocks, is warranted. Possible methods include

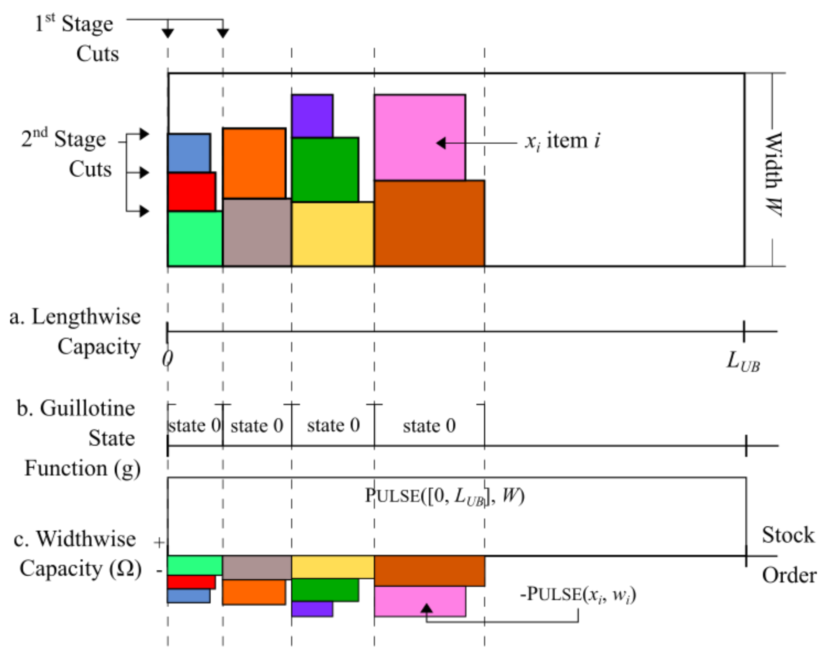


Figure 7.1: Proposed constraint programming model for 2D-LSPP.

rounding heuristics, local search, and large neighbourhood search.

7.2.2 Extensions to Related Problems

Another direction of future work is to extend the scheduling-based constraint programming model to other problems.

Two-Dimensional Level Strip Packing Problem In Two-Dimensional Level Strip Packing (2D-LSPP), a rectangular stock of fixed width and infinite length needs to be cut into a set of items using at most three stages of guillotine cuts, and the goal is to cut the shortest length possible [174]. An important nuance between 2D-LSPP and other guillotine packing problems is that two items cannot be placed side-by-side within a level. We can easily translate this packing problem to a batch scheduling problem by interpreting items as operations without any precedence requirements and the stock as a single machine with capacity equalling the stock width. The desire to minimize length can then interpreted as one to minimize the makespan.

Here, we adapt our single resource constraint programming formulation to the 2D-LSPP (Figure 7.1). Given a set of items N, for each item $i \in N$ with width w_i and length ρ_i, we introduce an interval variable x_i with duration ρ_i. We also declare a state function g to represent the guillotine cuts (Figure 7.1b). We use a cumulative function Ω to track the amount of stock width not used at any given length. Different from 2SCSP-FF, 2D-LSPP allows three-stages of guillotine cuts, which can be represented by not enforcing the end times of each items to align with a state in the state function. Then, for stock of width W

and infinite length, we can compute an upper bound L_{UB} for the stock length using some simple heuristic and denote the lengthwise horizon as $\mathcal{H} = [0, L_{UB}]$ (Figure 7.1a). Finally, we again use a cumulative function Ω to track the widthwise usage of the stock (Figure 7.1c). Now, we can define the scheduling-based constraint program as follows.

$$\begin{aligned}
\min \quad & \max_{i \in N} \textsc{EndOf}(x_i) && && (7.1a)\\
\text{s.t.} \quad & \textsc{AlwaysConstant}(g, x_i, True, False)\ \forall i \in N && && (7.1b)\\
& \Omega = \textsc{Pulse}(\mathcal{H}, W) - \sum_{i \in N} \textsc{Pulse}(x_i, w_i) && && (7.1c)\\
& \Omega \geq 0 && && (7.1d)\\
& x_i : \textsc{IntervalVar}(\mathcal{H}, \rho_i) && \forall i \in N && (7.1e)\\
& g : \textsc{StateFunction}() && && (7.1f)
\end{aligned}$$

Objective (7.1a) defines the maximum stock length required, which equals the latest end time of all interval variables. Constraint (7.1b) represents the guillotine cuts. In particular, we recall that the last two arguments to the constraint AlwaysConstant is true only if the start and the end times of interval variable x_i are aligned with the start and end times of a state in the state function g, respectively. With three stages of guillotine cuts, the rightmost coordinates of items on the same level does not need to align, so we set that latter argument to false. Constraints (7.1c) and (7.1d) ensure that the widthwise capacity of the stock is always satisfied. The rest define the variables.

Comparing with the single resource Model 4.1 in Chapter 4, we observe that Model 7.1 does not declare any infeasible regions, since only one stock is present, and that all item interval variables are non-optional. These should allow a stronger temporal relaxation, improve the search process, and reduce the search space. Furthermore, if the stock length is interpreted as a time horizon, the objective of 2D-LSPP is equivalent to minimizing the makespan, an objective function which constraint programming solvers are typically very good at optimizing. Hence, we believe extending the scheduling concepts to this problem is very promising.

Two-Dimensional Bin Packing Problem with Size Changeable Items Introduced by Lee et al. [163], the Two-Dimensional Bin Packing Problem with Size Changeable Items generalizes 2SCSP-FF in that the width of items is also flexible within a specified range. In other words, item $p \in C_i$ of order $i \in N$ is characterized by a flexible width and length within intervals $\bar{w}_i = [w_i^{min}, w_i^{max}]$ and $\bar{\rho}_i = [\rho_i^{min}, \rho_i^{max}]$, respectively. To adapt the single resource model, we recognize that the magnitude of Pulse can in fact be an interval, so we simply need to redefine the cumulative function representing the net usage of stock width to be $\Omega = \sum_{k \in K} \textsc{Pulse}(c_k, W_k) - \sum_{i \in N} \sum_{p \in C_i} \textsc{Pulse}(x_{ip}, \bar{w_i})$. Here, the only change is the substitution of w_i with $\bar{w_i}$. Because the change is subtle, the representational efficiency of the single resource model exhibited in solving 2SCSP-FF is likely to carry over to this problem, making this an interesting direction for further investigation.

Two-Dimensional Packing Problems with Defects Introduced by Afsharian [3], the Two-Dimensional Cutting Stock Problem with Defects and Guillotine Cuts asks for rectangular items to be cut out of a larger rectangular stock using guillotines while avoiding rectangular defects on the stocks. Without loss of generality, assuming the length of stocks is unified into a single resource, the length of a defective area can easily modelled by introducing an interval variable with start and end times representing the its leftmost and rightmost length coordinates. However, representing the width of the defective area using cumulative functions can be difficult, as the pulses generated by this interval variable does not represent the absolute widthwise positioning of the defective area. One alternative would be to declare two sets of interval variables for every item and for every defective area, one set representing the length and another representing the width, similar to the work by Clautiaux et al. [55]. The resulting effect on the performance of the model, however, is unclear.

Appendix A

Parameter Notations

Table A.1 summarizes the notations used in this thesis.

Parameter	**Description**
	$\forall$ order $i \in N$
q_i^{max}	Upper bound on the total area demanded of order i.
q_i^{min}	Lower bound on the total area demanded of order i.
w_i	Width of items belonging to order i.
ρ_i^{max}	Upper bound on the item length of order i.
ρ_i^{min}	Lower bound on the item length of order i.
d_i	Due date of order i.
P_i	The index set of items/partitions belonging to order i on a level.
A_i	The index set of necessary items belonging to order i that must be present in all solutions.
B_i	The index set of optional items belonging to order i.
C_i	The index set of all possible items belonging to order i.
	$\forall$ order properties $\gamma \in G$
$N_\gamma \subseteq N$	The subset of orders demanding order properties γ.
p_γ	Maximum number of items demanding order properties γ on a level.
	$\forall$ stock $k \in K$
W_k	Width of stock k.
L_k	Length of stock k.
a_k	Availability date of stock k.
J_k	The set of possible numbers of levels of stock k.

Table A.1: Notations describing or derived from the problem inputs.

Table A.2 summarizes the notations specific to the sequential heuristic.

Parameter	**Description**
$\mathcal{C}$	The set of cannot-link relations.
S	The set of level patterns obtained by solving the first second step.
S_k	The set of level patterns obtained by solving the first second step that can fit widthwise into stock k.

Table A.2: Notations used in the sequential heuristic.

Table A.3 describes the notations specific to the 2SCSP-FFMS in Chapter 6.

Parameter	**Description**
c_{tard}	Tardiness cost per unit area per unit time.
c_{inv}	Inventory cost per unit area per unit time.
$\mathcal{T}_{ik}$	Scheduling cost per unit area of order i assigned to stock k.
$\mathcal{T}_i^{SR}(\tau)$	Scheduling cost per unit area of order i assigned to coordinate τ in the single resource.
$\mathcal{T}_{total}$	The total scheduling costs for all orders.

Table A.3: Notations used in Chapter 6.

Appendix B

2SCSP-FF: Detailed Results

The tables in this appendix present the detailed results for solving the generated and industrial instances of 2SCSP-FF. In total, there are 50 generated instances and 4 industrial instances. Table B.1 describes the number of feasible solutions found, Table B.2 describes the average optimality gap calculated using Equation (4.34), and Table B.3 describes the average time taken for an approach to find the first feasible solution. For Table B.2 and Table B.3, if an approach did not find a feasible solution for an instance, we penalize it with the worst solution found for that instance. The lower bound used in the optimality gap calculation is the best lower bound found across all approaches.

	Generated					Industrial			
$\lvert N\rvert$	4	8	16	32	64	19	21	47	149
$\lvert K\rvert$	8	16	32	64	128	42	172	149	636
$S: CC^{S1} + CP^{S2} + BD^{S3,lb}_{comp}$	10	9	5	2	0	1	1	0	0
$S: CC^{S1} + CP^{S2} + BD^{S3,ub}_{comp}$	10	10	1	0	0	1	0	0	0
$S: CC^{S1} + CP^{S2} + BD^{S3,lb}_{simp}$	9	10	10	9	7	1	1	1	1
$S: CC^{S1} + CP^{S2} + BD^{S3,ub}_{simp}$	9	10	10	9	6	1	1	0	1
$S: CC^{S1} + CP^{S2} + CP^{S3}$	7	8	6	0	0	0	0	0	0
$S: CC^{S1} + CP^{S2} + MIP^{S3}_{comp}$	9	10	9	5	0	1	1	0	0
$S: CC^{S1} + CP^{S2} + MIP^{S3}_{simp}$	10	10	10	9	9	1	1	1	0
$S: CC^{S1} + MIP^{S2} + BD^{S3,lb}_{comp}$	10	10	10	7	1	1	1	0	0
$S: CC^{S1} + MIP^{S2} + BD^{S3,ub}_{comp}$	10	10	6	0	0	1	1	0	0
$S: CC^{S1} + MIP^{S2} + BD^{S3,lb}_{simp}$	10	10	10	10	10	1	1	1	1
$S: CC^{S1} + MIP^{S2} + BD^{S3,ub}_{simp}$	10	10	10	10	10	1	1	1	1
$S: CC^{S1} + MIP^{S2} + CP^{S3}$	10	10	9	10	10	1	1	1	1
$S: CC^{S1} + MIP^{S2} + MIP^{S3}_{comp}$	10	10	10	9	3	1	1	1	0
$S: CC^{S1} + MIP^{S2} + MIP^{S3}_{simp}$	10	10	10	10	10	1	1	1	1
$S: CP^{S1} + CP^{S2} + BD^{S3,lb}_{comp}$	9	10	8	5	1	1	1	0	0

$S : CP^{S1} + CP^{S2} + BD^{S3,ub}_{comp}$	9	10	4	1	0	1	1	0	0
$S : CP^{S1} + CP^{S2} + BD^{S3,lb}_{simp}$	10	10	10	10	5	1	1	0	1
$S : CP^{S1} + CP^{S2} + BD^{S3,ub}_{simp}$	10	10	10	10	5	1	1	0	1
$S : CP^{S1} + CP^{S2} + CP^{S3}$	8	4	4	0	0	0	0	0	0
$S : CP^{S1} + CP^{S2} + MIP^{S3}_{comp}$	9	10	9	7	2	1	1	0	0
$S : CP^{S1} + CP^{S2} + MIP^{S3}_{simp}$	10	10	10	10	5	1	1	0	1
$S : CP^{S1} + MIP^{S2} + BD^{S3,lb}_{comp}$	10	10	10	7	1	1	1	1	0
$S : CP^{S1} + MIP^{S2} + BD^{S3,ub}_{comp}$	10	10	7	0	0	1	1	0	0
$S : CP^{S1} + MIP^{S2} + BD^{S3,lb}_{simp}$	10	10	10	10	9	1	1	1	0
$S : CP^{S1} + MIP^{S2} + BD^{S3,ub}_{simp}$	10	10	10	10	9	1	1	1	1
$S : CP^{S1} + MIP^{S2} + CP^{S3}$	10	10	10	10	10	1	1	1	1
$S : CP^{S1} + MIP^{S2} + MIP^{S3}_{comp}$	10	10	10	10	2	1	1	1	0
$S : CP^{S1} + MIP^{S2} + MIP^{S3}_{simp}$	10	10	10	10	10	1	1	1	1
$FFMH$	9	10	10	10	9	1	1	1	0
CP_{CO}	10	10	8	0	0	1	0	0	0
CP_{IT}	10	10	3	0	0	1	0	0	0
CP_{SR}	10	10	10	10	10	1	1	1	1
CP_{ST}	10	10	10	3	0	1	0	0	0
MIP_{AS}	8	9	10	0	0	0	0	0	0
MIP_{CO}	8	9	10	0	0	0	0	0	0

Table B.1: The number of instances for which an approach found a feasible solution for 2SCSP-FF for all approaches.

	Generated					Industrial			
$\|N\|$	4	8	16	32	64	19	21	47	149
$\|K\|$	8	16	32	64	128	42	172	149	636
$S : CC^{S1} + CP^{S2} + BD^{S3,lb}_{comp}$	29.1	33.6	136.6	110.0	111.6	29.1	215.3	107.1	88.0
$S : CC^{S1} + CP^{S2} + BD^{S3,ub}_{comp}$	32.1	26.1	148.3	112.4	111.6	29.4	236.9	107.1	88.0
$S : CC^{S1} + CP^{S2} + BD^{S3,lb}_{simp}$	27.7	15.8	12.7	63.8	94.8	28.5	69.4	89.9	87.6
$S : CC^{S1} + CP^{S2} + BD^{S3,ub}_{simp}$	26.4	17.2	11.7	63.7	94.1	28.5	69.2	107.1	85.1
$S : CC^{S1} + CP^{S2} + CP^{S3}$	47.6	62.3	79.6	112.4	111.6	116.0	236.9	107.1	88.0
$S : CC^{S1} + CP^{S2} + MIP^{S3}_{comp}$	33.5	55.5	112.0	103.4	111.6	38.1	236.9	107.1	88.0
$S : CC^{S1} + CP^{S2} + MIP^{S3}_{simp}$	22.8	13.4	12.5	62.0	89.2	29.9	73.1	85.9	88.0
$S : CC^{S1} + MIP^{S2} + BD^{S3,lb}_{comp}$	13.9	9.1	48.1	108.5	109.2	24.5	71.9	107.1	88.0
$S : CC^{S1} + MIP^{S2} + BD^{S3,ub}_{comp}$	14.8	9.2	43.7	112.4	111.6	26.2	73.2	107.1	88.0
$S : CC^{S1} + MIP^{S2} + BD^{S3,lb}_{simp}$	13.8	5.2	2.9	58.1	82.3	21.3	67.7	74.9	85.9
$S : CC^{S1} + MIP^{S2} + BD^{S3,ub}_{simp}$	13.8	5.3	3.0	58.1	76.6	20.8	67.6	75.2	83.9
$S : CC^{S1} + MIP^{S2} + CP^{S3}$	16.4	9.6	37.9	61.3	76.4	21.4	73.4	82.0	79.4

$S : CC^{S1} + MIP^{S2} + MIP^{S3}_{comp}$	16.7	15.2	93.4	109.8	111.5	116.0	236.7	74.4	88.0
$S : CC^{S1} + MIP^{S2} + MIP^{S3}_{simp}$	13.0	4.9	3.3	55.7	72.0	22.9	68.9	74.2	87.7
$S : CP^{S1} + CP^{S2} + BD^{S3,lb}_{comp}$	29.3	25.6	114.4	105.3	111.6	28.9	223.9	107.1	88.0
$S : CP^{S1} + CP^{S2} + BD^{S3,ub}_{comp}$	30.7	27.6	97.0	111.9	111.6	30.0	72.7	107.1	88.0
$S : CP^{S1} + CP^{S2} + BD^{S3,lb}_{simp}$	23.3	14.3	11.1	64.5	102.2	28.7	68.9	107.1	83.7
$S : CP^{S1} + CP^{S2} + BD^{S3,ub}_{simp}$	23.3	13.9	11.2	64.5	96.8	28.7	68.8	107.1	83.9
$S : CP^{S1} + CP^{S2} + CP^{S3}$	44.8	76.1	122.3	112.4	111.6	116.0	236.9	107.1	88.0
$S : CP^{S1} + CP^{S2} + MIP^{S3}_{comp}$	35.2	29.9	118.8	95.9	111.6	32.6	70.3	107.1	88.0
$S : CP^{S1} + CP^{S2} + MIP^{S3}_{simp}$	23.3	14.2	11.5	64.1	97.2	28.7	72.0	107.1	88.0
$S : CP^{S1} + MIP^{S2} + BD^{S3,lb}_{comp}$	11.7	9.8	53.7	108.3	111.0	22.9	130.7	107.1	88.0
$S : CP^{S1} + MIP^{S2} + BD^{S3,ub}_{comp}$	13.5	8.3	37.7	112.4	111.6	21.7	70.5	107.1	88.0
$S : CP^{S1} + MIP^{S2} + BD^{S3,lb}_{simp}$	11.6	4.4	2.9	59.5	88.8	20.3	68.0	86.3	88.0
$S : CP^{S1} + MIP^{S2} + BD^{S3,ub}_{simp}$	11.6	4.3	2.6	59.3	84.3	20.1	67.6	74.6	84.2
$S : CP^{S1} + MIP^{S2} + CP^{S3}$	14.8	10.4	11.4	61.9	77.4	24.3	74.2	82.6	79.4
$S : CP^{S1} + MIP^{S2} + MIP^{S3}_{comp}$	14.5	15.0	44.9	85.9	111.4	114.7	229.5	74.2	88.0
$S : CP^{S1} + MIP^{S2} + MIP^{S3}_{simp}$	11.7	4.5	2.5	55.5	72.4	22.8	67.8	74.2	87.0
CP_{CO}	8.2	25.7	64.9	112.4	111.6	35.9	236.9	107.1	88.0
CP_{IT}	31.1	43.4	112.3	112.4	111.6	46.2	236.9	107.1	88.0
CP_{SR}	18.6	10.4	8.7	57.9	72.4	22.8	68.9	75.8	71.9
CP_{ST}	26.9	28.1	45.8	99.5	111.6	51.7	236.9	107.1	88.0
MIP_{AS}	15.2	13.3	6.8	112.4	111.6	116.0	236.9	107.1	88.0
MIP_{CO}	15.4	12.3	6.9	112.4	111.6	116.0	236.9	107.1	88.0

Table B.2: % optimality gap for generated and industrial instances for 2SCSP-FF for all approaches.

	Generated					Industrial			
$\|N\|$	4	8	16	32	64	19	21	47	149
$\|K\|$	8	16	32	64	128	42	172	149	636
$S : CC^{S1} + MIP^{S2} + BD^{S3,lb}_{comp}$	2.6	3.4	2.0	2520.5	TO	3.0	5.0	TO	TO
$S : CC^{S1} + MIP^{S2} + BD^{S3,ub}_{comp}$	3.7	3240.0	3240.5	TO	TO	TO	TO	TO	TO
$S : CC^{S1} + MIP^{S2} + BD^{S3,lb}_{simp}$	1.9	2.7	3.2	5.9	2.3	3.0	1.0	15.0	20.0
$S : CC^{S1} + MIP^{S2} + BD^{S3,ub}_{simp}$	2.9	2.3	1.7	4.3	2.9	1.5	2.0	12.8	22.6
$S : CC^{S1} + MIP^{S2} + CP^{S3}$	2.7	3.8	362.6	12.9	177.0	4.0	2.0	12.0	25.0
$S : CC^{S1} + MIP^{S2} + MIP^{S3}_{comp}$	2.1	2.2	3.4	362.2	3240.4	4.0	3.0	21.0	TO
$S : CC^{S1} + MIP^{S2} + MIP^{S3}_{simp}$	2.0	1.1	2.1	4.0	10.3	2.0	5.0	9.0	91.0
$S : CP^{S1} + MIP^{S2} + BD^{S3,lb}_{comp}$	3.4	2.3	2.1	1441.2	TO	2.0	2.0	11.0	TO
$S : CP^{S1} + MIP^{S2} + BD^{S3,ub}_{comp}$	361.9	3240.3	TO	TO	TO	TO	TO	TO	TO
$S : CP^{S1} + MIP^{S2} + BD^{S3,lb}_{simp}$	3.0	2.8	3.5	2.3	793.5	1.0	4.0	13.0	TO

$S : CP^{S1} + MIP^{S2} + BD^{S3,ub}_{simp}$	2.2	2.2	2.3	2.1	409.0	1.0	5.0	10.0	21.0
$S : CP^{S1} + MIP^{S2} + CP^{S3}$	3.6	2.2	5.7	31.7	334.3	1.0	4.0	12.0	21.0
$S : CP^{S1} + MIP^{S2} + MIP^{S3}_{comp}$	2.5	2.4	2.8	2.3	2881.0	4.0	1.0	53.0	TO
$S : CP^{S1} + MIP^{S2} + MIP^{S3}_{simp}$	2.9	2.9	1.9	2.9	268.2	1.0	5.0	8.0	98.0
CP_{CO}	3.4	20.1	1029.2	TO	TO	137.0	TO	TO	TO
CP_{IT}	3.3	67.0	2701.1	TO	TO	17.0	TO	TO	TO
CP_{SR}	2.0	3.4	3.3	5.7	3.7	2.0	3.5	2.0	12.0
CP_{ST}	3.4	58.4	438.1	3217.9	TO	79.0	TO	TO	TO
MIP_{AS}	721.9	420.7	641.7	TO	TO	TO	TO	TO	TO
MIP_{CO}	722.2	378.1	380.4	TO	TO	TO	TO	TO	TO

Table B.3: Time to feasibility in seconds for generated and industrial instances for 2SCSP-FF for all approaches. TO denotes all instances of the approach timed out.

Appendix C

2SCSP-FFM: Detailed Results

The content in this appendix pertains to 2SCSP-FFM and is similarly structured as Appendix B.

	Generated					Industrial			
$\|N\|$	4	8	16	32	64	19	21	47	149
$\|K\|$	8	16	32	64	128	42	172	149	636
$S : CC^{S1} + MIP^{S2} + MIP^{S3}_{simp}$	10	10	10	10	10	1	1	1	1
$S : CP^{S1} + MIP^{S2} + MIP^{S3}_{simp}$	10	10	10	10	10	1	1	1	1
CP_{SR}	10	10	10	10	10	1	1	1	1
$FFMH$	9	10	10	10	9	1	1	1	0
MIP^{T}_{AS}	9	9	2	0	0	0	0	0	0
MIP^{C}_{AS}	9	9	0	0	0	0	0	0	0
MIP^{T}_{CO}	9	10	8	0	0	0	0	0	0
MIP^{C}_{CO}	9	10	8	0	0	0	0	0	0

Table C.1: The number of instances for which an approach found a feasible solution for 2SCSP-FFM for all approaches.

	Generated					Industrial			
$\|N\|$	4	8	16	32	64	19	21	47	149
$\|K\|$	8	16	32	64	128	42	172	149	636
$S : CC^{S1} + MIP^{S2} + MIP^{S3}_{simp}$	9.0	36.6	26.5	75.5	75.1	35.3	89.9	79.5	75.5
$S : CP^{S1} + MIP^{S2} + MIP^{S3}_{simp}$	7.4	31.3	25.9	78.4	93.0	35.3	86.9	78.6	93.1
CP_{SR}	18.2	42.5	33.4	77.1	76.2	30.2	76.5	80.3	76.1
$FFMH$	51.2	69.8	57.3	83.7	81.6	47.5	81.4	86.1	93.1
MIP^{T}_{AS}	11.9	35.7	101.3	88.7	97.6	90.6	89.9	99.5	93.1
MIP^{C}_{AS}	11.2	35.0	122.3	88.7	97.6	90.6	89.9	99.5	93.1
MIP^{T}_{CO}	11.7	29.9	44.0	88.7	97.6	90.6	89.9	99.5	93.1
MIP^{C}_{CO}	11.7	29.9	44.1	88.7	97.6	90.6	89.9	99.5	93.1

Table C.2: % optimality gap for generated and industrial instances for 2SCSP-FFM for all approaches.

	Generated					Industrial			
$\|N\|$	4	8	16	32	64	19	21	47	149
$\|K\|$	8	16	32	64	128	42	172	149	636
$S : CC^{S1} + MIP^{S2} + MIP^{S3}_{simp}$	4.3	2.1	3.1	9.5	35.3	2.0	11.0	51.0	657.0
$S : CP^{S1} + MIP^{S2} + MIP^{S3}_{simp}$	2.1	2.1	73.7	367.4	415.8	3.0	7.0	421.0	546.0
CP_{SR}	2.2	2.8	3.6	6.2	3.7	4.0	5.0	2.0	41.0
$FFMH$	362.8	6.2	24.1	118.9	987.3	9.0	131.0	557.0	TO
MIP^{C}_{AS}	363.0	862.2	TO	TO	TO	TO	TO	TO	TO
MIP^{T}_{AS}	362.8	874.1	3449.9	TO	TO	TO	TO	TO	TO
MIP^{C}_{CO}	361.3	38.2	1696.5	TO	TO	TO	TO	TO	TO
MIP^{T}_{CO}	362.7	397.6	1698.3	TO	TO	TO	TO	TO	TO

Table C.3: Time to feasibility in seconds for generated and industrial instances for 2SCSP-FFM for all approaches. TO denotes all instances of the approach timed out.

Appendix D

2SCSP-FFMS: Detailed Results

The content in this appendix pertains to the 2SCSP-FFMS and is similarly structured as Appendix B.

	Generated					Industrial			
$\|N\|$	4	8	16	32	64	19	21	47	149
$\|K\|$	8	16	32	64	128	42	172	149	636
$S : CC^{S1} + MIP^{S2} + MIP^{S3}_{simp}$	10	10	10	10	10	1	1	1	1
$S : CP^{S1} + MIP^{S2} + MIP^{S3}_{simp}$	10	10	10	10	10	1	1	1	1
CP_{SR}	10	10	10	10	10	1	1	1	1
$FFMH$	9	10	10	10	10	1	1	1	0
MIP^{T}_{AS}	10	10	2	0	0	1	0	0	0
MIP^{C}_{AS}	10	10	5	0	0	1	0	0	0
MIP^{T}_{CO}	10	10	10	0	0	1	0	0	0
MIP^{C}_{CO}	10	10	10	0	0	1	0	0	0

Table D.1: The number of instances for which an approach found a feasible solution for 2SCSP-FFMS for all approaches.

	Generated					Industrial			
$\|N\|$	4	8	16	32	64	19	21	47	149
$\|K\|$	8	16	32	64	128	42	172	149	636
$S : CC^{S1} + MIP^{S2} + MIP^{S3}_{simp}$	47	65	1765	79	78	55	133	1553	5542
$S : CP^{S1} + MIP^{S2} + MIP^{S3}_{simp}$	47	65	1454	80	80	55	130	1587	5443
CP_{SR}	504	155	4716	198	242	85	146	2863	6589
$FFMH$	1749	681	6503	259	235	169	141	2278	7534
MIP^{T}_{AS}	14	63	7292	337	313	29	470	3341	7534
MIP^{C}_{AS}	14	68	7460	337	313	27	470	3341	7534
MIP^{T}_{CO}	14	62	1589	337	313	26	470	3341	7534
MIP^{C}_{CO}	14	61	1614	337	313	27	470	3341	7534

Table D.2: % optimality gap for generated and industrial instances for 2SCSP-FFMS for all approaches.

	Generated					Industrial			
$\|N\|$	4	8	16	32	64	19	21	47	149
$\|K\|$	8	16	32	64	128	42	172	149	636
$S : CC^{S1} + MIP^{S2} + MIP^{S3}_{simp}$	2.6	1.9	3.1	10.3	26.7	3.0	12.0	113.0	1658.0
$S : CP^{S1} + MIP^{S2} + MIP^{S3}_{simp}$	3.2	3.4	331.4	370.5	390.0	364.0	368.0	385.0	535.0
CP_{SR}	8.4	2.6	2.7	18.6	5.4	5.0	2.0	8.0	67.0
$FFMH$	362.8	6.0	23.1	114.5	672.8	9.0	123.0	503.0	TO
MIP^{C}_{AS}	3.6	392.8	3040.9	TO	TO	2627.0	TO	TO	TO
MIP^{T}_{AS}	2.5	317.2	3109.4	TO	TO	1425.0	TO	TO	TO
MIP^{C}_{CO}	2.5	20.5	830.0	TO	TO	289.0	TO	TO	TO
MIP^{T}_{CO}	2.6	24.0	876.2	TO	TO	345.0	TO	TO	TO

Table D.3: Time to feasibility in seconds for generated and industrial instances for 2SCSP-FFMS for all approaches. TO denotes all instances of the approach timed out.

Bibliography

[1] Hernan Abeledo, Ricardo Fukasawa, Artur Pessoa, and Eduardo Uchoa. "The Time Dependent Traveling Salesman Problem: Polyhedra and Branch-Cut-and-Price Algorithm". In: vol. 5. May 2010, pp. 202–213.

[2] Tobias Achterberg, Thorsten Koch, and Alexander Martin. "Branching rules revisited". In: *Operations Research Letters* 33.1 (2005), pp. 42–54.

[3] Mohsen Afsharian. "The Two-Dimensional, Rectangular, Guillotineable-Layout Cutting Problem with Defects". PhD thesis. Otto von Guericke University Magdeburg, 2013.

[4] Mohsen Afsharian, Ali M. Niknejad, and Gerhard Wäscher. "A heuristic, dynamic programming-based approach for a two-dimensional cutting problem with defects". In: *OR Spectr.* 36.4 (2014), pp. 971–999.

[5] Dwi Agustina, C.K.M. Lee, and Rajesh Piplani. "Vehicle scheduling and routing at a cross docking center for food supply chains". In: *International Journal of Production Economics* 152 (2014). Sustainable Food Supply Chain Management, pp. 29–41.

[6] Filipe Alvelos, T. M. Chan, Paulo Vilaça, Tiago Gomes, Elsa Silva, and José Carvalho. "Sequence based heuristics for two-dimensional bin packing problems". In: *Engineering Optimization* 41 (July 2009), pp. 773–791.

[7] Adriana C. F. Alvim and Celso C. Ribeiro. "A Hybrid Bin-Packing Heuristic to Multiprocessor Scheduling". In: *Experimental and Efficient Algorithms, Third International Workshop, WEA 2004, Angra dos Reis, Brazil, May 25-28, 2004, Proceedings.* Ed. by Celso C. Ribeiro and Simone L. Martins. Vol. 3059. Lecture Notes in Computer Science. Springer, 2004, pp. 1–13.

[8] Hatem Ben Amor, Jacques Desrosiers, and José M. Valério de Carvalho. "Dual-Optimal Inequalities for Stabilized Column Generation". In: *Oper. Res.* 54.3 (2006), pp. 454–463.

[9] D. Applegate, R. Bixby, V. Chvatal, and B. Cook. *Finding Cuts in the TSP (A Preliminary Report).* Tech. rep. 1995.

[10] Egon Balas. “Disjunctive Programming”. In: *Discrete Optimization II.* Ed. by P.L. Hammer, E.L. Johnson, and B.H. Korte. Vol. 5. Annals of Discrete Mathematics. Elsevier, 1979, pp. 3–51.

[11] Egon Balas. “Facets of the knapsack polytope”. In: *Math. Program.* 8.1 (1975), pp. 146–164.

[12] Egon Balas and Eitan Zemel. “Facets of the Knapsack Polytope From Minimal Covers”. In: *SIAM Journal on Applied Mathematics* 34.1 (1978), pp. 119–148.

[13] Michel Balinski and Richard E. Quandt. “On an Integer Program for a Delivery Problem”. In: *Operations Research* 12 (1964), pp. 300–304.

[14] Philippe Baptiste, Philippe Laborie, Claude Le Pape, and Wim Nuijten. “Constraint-Based Scheduling and Planning”. In: *Handbook of Constraint Programming.* Ed. by Francesca Rossi, Peter van Beek, and Toby Walsh. Vol. 2. Foundations of Artificial Intelligence. Elsevier, 2006, pp. 761–799.

[15] Cynthia Barnhart, Ellis L. Johnson, George L. Nemhauser, Martin W. P. Savelsbergh, and Pamela H. Vance. “Branch-and-Price: Column Generation for Solving Huge Integer Programs”. In: *Oper. Res.* 46.3 (1998), pp. 316–329.

[16] E. Beale and John Tomlin. “Special facilities in a general mathematical programming system for nonconvex problems using ordered sets of variables”. In: *Operational Research* 69 (Jan. 1969), pp. 447–454.

[17] E. M. L. Beale and John J. H. Forrest. “Global optimization using special ordered sets”. In: *Math. Program.* 10.1 (1976), pp. 52–69.

[18] J.C. Beck, Andrew Davenport, Eugene Davis, and Mark Fox. “The ODO project: Toward a unified basis for constraint-directed scheduling”. In: *J. Sched.* 1 (1998).

[19] Henrique Becker, Olinto Araujo, and Luciana S. Buriol. “Extending an Integer Formulation for the Guillotine 2D Bin Packing Problem”. In: *Proceedings of the XI Latin and American Algorithms, Graphs and Optimization Symposium, LAGOS 2021, Online Event / São Paulo, Brazil, May 2021.* Ed. by Carlos E. Ferreira, Orlando Lee, and Flávio Keidi Miyazawa. Vol. 195. Procedia Computer Science. Elsevier, 2021, pp. 499–507.

[20] N Beldiceanu and E Contejean. “Introducing global constraints in CHIP”. In: *Mathematical and Computer Modelling* 20.12 (1994), pp. 97–123.

[21] Nicolas Beldiceanu, Mats Carlsson, Sophie Demassey, and Thierry Petit. “Global Constraint Catalog: Past, Present and Future”. In: *Constraints* 12 (Mar. 2007), pp. 21–62.

[22] Nicolas Beldiceanu, Mats Carlsson, Pierre Flener, and Justin Pearson. “On the reification of global constraints”. In: *Constraints* 18 (2012), pp. 1–6.

[23] Mandell Bellmore and John C. Malone. “Pathology of Traveling-Salesman Subtour-Elimination Algorithms”. In: *Oper. Res.* 19.2 (1971), pp. 278–307.

[24] Gleb Belov, Guntram Scheithauer, and E. A. Mukhacheva. “One-dimensional heuristics adapted for two-dimensional rectangular strip packing”. In: *J. Oper. Res. Soc.* 59.6 (2008), pp. 823–832.

[25] J. F. Benders. “Partitioning procedures for solving mixed-variables programming problems”. In: *Numerische Mathematik* 4.1 (Dec. 1962), pp. 238–252.

[26] Michel Bénichou, Jean-Michel Gauthier, Paul Girodet, Gerard Hentges, Gerard Ribière, and O. Vincent. “Experiments in mixed-integer linear programming”. In: *Math. Program.* 1.1 (1971), pp. 76–94.

[27] Judith O. Berkey and P. Y. Wang. “Two-Dimensional Finite Bin-Packing Algorithms”. In: *Journal of the Operational Research Society* 38 (1987), pp. 423–429.

[28] Ksenia Bestuzheva, Mathieu Besançon, Wei-Kun Chen, Antonia Chmiela, Tim Donkiewicz, Jasper van Doornmalen, Leon Eifler, Oliver Gaul, Gerald Gamrath, Ambros Gleixner, Leona Gottwald, Christoph Graczyk, Katrin Halbig, Alexander Hoen, Christopher Hojny, Rolf van der Hulst, Thorsten Koch, Marco Lübbecke, Stephen J. Maher, Frederic Matter, Erik Mühmer, Benjamin Müller, Marc E. Pfetsch, Daniel Rehfeldt, Steffan Schlein, Franziska Schlösser, Felipe Serrano, Yuji Shinano, Boro Sofranac, Mark Turner, Stefan Vigerske, Fabian Wegscheider, Philipp Wellner, Dieter Weninger, and Jakob Witzig. *The SCIP Optimization Suite 8.0.* Technical Report. Optimization Online, Dec. 2021.

[29] Ksenia Bestuzheva, Mathieu Besançon, Wei-Kun Chen, Antonia Chmiela, Tim Donkiewicz, Jasper van Doornmalen, Leon Eifler, Oliver Gaul, Gerald Gamrath, Ambros Gleixner, Leona Gottwald, Christoph Graczyk, Katrin Halbig, Alexander Hoen, Christopher Hojny, Rolf van der Hulst, Thorsten Koch, Marco Lübbecke, Stephen J. Maher, Frederic Matter, Erik Mühmer, Benjamin Müller, Marc E. Pfetsch, Daniel Rehfeldt, Steffan Schlein, Franziska Schlösser, Felipe Serrano, Yuji Shinano, Boro Sofranac, Mark Turner, Stefan Vigerske, Fabian Wegscheider, Philipp Wellner, Dieter Weninger, and Jakob Witzig. *The SCIP Optimization Suite 8.0.* ZIB-Report 21-41. Zuse Institute Berlin, Dec. 2021.

[30] Andrea Bettinelli, Alberto Ceselli, and Giovanni Righini. “A branch-and-price algorithm for the two-dimensional level strip packing problem”. In: *4OR* 6.4 (2008), pp. 361–374.

[31] Vanessa M. R. Bezerra, Aline Aparecida de Souza Leão, José Fernando Oliveira, and Maristela Oliveira Santos. “Models for the two-dimensional level strip packing problem - a review and a computational evaluation”. In: *J. Oper. Res. Soc.* 71.4 (2020), pp. 606–627.

[32] Merve Bodur. *Lecture notes in Integer Programming.* Jan. 2021.

[33] Kyle Booth. “Constraint Programming Approaches to Electric Vehicle and Robot Routing Problems”. PhD dissertation. University of Toronto, 2021.

[34] Kyle Booth. “Constraint Programming Approaches to Electric Vehicle and Robot Routing Problems”. PhD thesis. University of Toronto, 2021.

[35] Kyle Booth, Tony Tran, Goldie Nejat, and J. Beck. “Mixed-Integer and Constraint Programming Techniques for Mobile Robot Task Planning”. In: *IEEE Robotics and Automation Letters* 1 (Jan. 2016).

[36] Kyle E. C. Booth and J. Christopher Beck. “A Constraint Programming Approach to Electric Vehicle Routing with Time Windows”. In: *Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 16th International Conference, CPAIOR 2019.* Vol. 11494. Lecture Notes in Computer Science. Springer, 2019, pp. 129–145.

[37] Andreas Bortfeldt. “A genetic algorithm for the two-dimensional strip packing problem with rectangular pieces”. In: *Eur. J. Oper. Res.* 172.3 (2006), pp. 814–837.

[38] Andreas Bortfeldt and Tobias Winter. “A genetic algorithm for the two-dimensional knapsack problem with rectangular pieces”. In: *Int. Trans. Oper. Res.* 16.6 (2009), pp. 685–713.

[39] Marco Antonio Boschetti and Lorenza Montaletti. “An Exact Algorithm for the Two-Dimensional Strip-Packing Problem”. In: *Oper. Res.* 58.6 (2010), pp. 1774–1791.

[40] Filipe Brandão and João Pedro Pedroso. “Bin packing and related problems: General arc-flow formulation with graph compression”. In: *Comput. Oper. Res.* 69 (2016), pp. 56–67.

[41] Edmund K. Burke, Matthew R. Hyde, and Graham Kendall. “A squeaky wheel optimisation methodology for two-dimensional strip packing”. In: *Comput. Oper. Res.* 38.7 (2011), pp. 1035–1044.

[42] Keith Butterworth. “Practical Application of Linear/Integer Programming in Agriculture”. In: *Journal of the Operational Research Society* 36 (1985), pp. 99–107.

[43] Eray Cakici, Scott J. Mason, John W. Fowler, and H. Neil Geismar. “Batch scheduling on parallel machines with dynamic job arrivals and incompatible job families”. In: *International Journal of Production Research* 51.8 (2013), pp. 2462–2477.

[44] Quentin Cappart and Pierre Schaus. “Rescheduling Railway Traffic on Real Time Situations Using Time-Interval Variables”. In: *Integration of AI and OR Techniques in Constraint Programming - 14th International Conference, CPAIOR 2017.* Vol. 10335. Lecture Notes in Computer Science. Springer, 2017, pp. 312–327.

[45] Alberto Caprara, Andrea Lodi, and Michele Monaci. “Fast Approximation Schemes for Two-Stage, Two-Dimensional Bin Packing”. In: *Math. Oper. Res.* 30.1 (2005), pp. 150–172.

[46] Alberto Caprara and Michele Monaci. “On the two-dimensional Knapsack Problem”. In: *Oper. Res. Lett.* 32.1 (2004), pp. 5–14.

[47] José M. Valério de Carvalho. “Exact solution of bin-packing problems using column generation and branch-and-bound”. In: *Ann. Oper. Res.* 86 (1999), pp. 629–659.

[48] José M. Valério de Carvalho. “Using Extra Dual Cuts to Accelerate Column Generation”. In: *INFORMS J. Comput.* 17.2 (2005), pp. 175–182.

[49] Sang-Yule Choi, Myong-chul Shin, and Jae Sang Cha. “Loss Reduction in Distribution Networks Using Cyclic Best First Search”. In: *Computational Science and Its Applications - ICCSA 2006, International Conference, Glasgow, UK, May 8-11, 2006, Proceedings, Part V.* Ed. by Marina L. Gavrilova, Osvaldo Gervasi, Vipin Kumar, Chih Jeng Kenneth Tan, David Taniar, Antonio Laganà, Youngsong Mun, and Hyunseung Choo. Vol. 3984. Lecture Notes in Computer Science. Springer, 2006, pp. 312–321.

[50] Christian Schulte and Guido Tack and Mikael Lagerkvist. *Modeling and programming with Gecode.* 2010.

[51] F. R. K. Chung, M. R. Garey, and D. S. Johnson. “On Packing Two-Dimensional Bins”. In: *SIAM Journal on Algebraic Discrete Methods* 3.1 (1982), pp. 66–76.

[52] Vasek Chvátal. “Edmonds polytopes and a hierarchy of combinatorial problems”. In: *Discret. Math.* 4.4 (1973), pp. 305–337.

[53] G. F. Cintra, Flávio Keidi Miyazawa, Yoshiko Wakabayashi, and E. C. Xavier. “Algorithms for two-dimensional cutting stock and strip packing problems using dynamic programming and column generation”. In: *Eur. J. Oper. Res.* 191.1 (2008), pp. 61–85.

[54] François Clautiaux, Jacques Carlier, and Aziz Moukrim. “A new exact method for the two-dimensional orthogonal packing problem”. In: *Eur. J. Oper. Res.* 183.3 (2007), pp. 1196–1211.

[55] François Clautiaux, Antoine Jouglet, Jacques Carlier, and Aziz Moukrim. “A new constraint programming approach for the orthogonal packing problem”. In: *Computers & Operations Research* 35.3 (2008). Part Special Issue: New Trends in Locational Analysis, pp. 944–959.

[56] W. F. Clocksin and Chris Mellish. *Programming in Prolog.* Springer, 1981.

[57] Alain Colmerauer. “Opening the Prolog III Universe”. In: *BYTE* 12.9 (Aug. 1987), pp. 177–182.

[58] Jean-François Côté, Michel Gendreau, and Jean-Yves Potvin. "An Exact Algorithm for the Two-Dimensional Orthogonal Packing Problem with Unloading Constraints". In: *Oper. Res.* 62.5 (2014), pp. 1126–1141.

[59] Harlan Crowder, Ellis L. Johnson, and Manfred Padberg. "Solving Large-Scale Zero-One Linear Programming Problems". In: *Operations Research* 31.5 (1983), pp. 803–834.

[60] Yaodong Cui, Yi Yao, and Yi-Ping Cui. "Hybrid approach for the two-dimensional bin packing problem with two-staged patterns". In: *Int. Trans. Oper. Res.* 23.3 (2016), pp. 539–549.

[61] G. B. Dantzig. "Maximization of a Linear Function of Variables Subject to Linear Inequalities". In: *[Activity Analysis of Production and Allocation, Cowles Commission Monograph]* 13 (1951).

[62] George B. Dantzig. "Linear programming and extensions". In: 1963.

[63] Rina Dechter and Judea Pearl. "Generalized Best-First Search Strategies and the Optimality of A*". In: *J. ACM* 32.3 (1985), pp. 505–536.

[64] Türkay Dereli and Gülesin Sena Das. "A Hybrid Simulated-Annealing Algorithm for Two-Dimensional Strip Packing Problem". In: *Adaptive and Natural Computing Algorithms, 8th International Conference, ICANNGA 2007, Warsaw, Poland, April 11-14, 2007, Proceedings, Part I*. Ed. by Bartlomiej Beliczynski, Andrzej Dzielinski, Marcin Iwanowski, and Bernardete Ribeiro. Vol. 4431. Lecture Notes in Computer Science. Springer, 2007, pp. 508–516.

[65] Guy Desaulniers, Jacques Desrosiers, Irina Ioachim, Marius Solomon, François Soumis, and Daniel Villeneuve. "A Unified Framework for Deterministic Time Constrained Vehicle Routing and Crew Scheduling Problems". In: vol. 57-93. Jan. 1998, pp. 57–93.

[66] M. Desrochers, J. Lenstra, Martin Savelsbergh, and F. Soumis. "Vehicle routing with time windows: Optimization and approximation". In: *Veh Rout Methods Stud* 16 (Jan. 1988).

[67] Jacques Desrosiers, François Soumis, and Martin Desrochers. "Routing with time windows by column generation". In: *Networks* 14.4 (1984), pp. 545–565.

[68] Mehmet Dincbas, Helmut Simonis, and Pascal Van Hentenryck. "Solving a cutting-stock problem with the constraint logic programming language CHIP". In: *Mathematical and Computer Modelling* 16 (1992), pp. 95–105.

[69] Michael Drexl. "Synchronization in Vehicle Routing - A Survey of VRPs with Multiple Synchronization Constraints". In: *Transp. Sci.* 46.3 (2012), pp. 297–316.

[70] Alessandro Druetto and Andrea Grosso. “Column generation and rounding heuristics for minimizing the total weighted completion time on a single batching machine”. In: *Computers Operations Research* 139 (2022), p. 105639.

[71] Harald Dyckhoff. “A New Linear Programming Approach to the Cutting Stock Problem”. In: *Oper. Res.* 29.6 (1981), pp. 1092–1104.

[72] Jens Egeblad and David Pisinger. “Heuristic approaches for the two- and three-dimensional knapsack packing problem”. In: *Comput. Oper. Res.* 36.4 (2009), pp. 1026–1049.

[73] Leah Epstein, Asaf Levin, and Rob van Stee. “Two-dimensional packing with conflicts”. In: *Acta Informatica* 45.3 (2008), pp. 155–175.

[74] Tomislav Erdelić and Tonci Caric. “A Survey on the Electric Vehicle Routing Problem: Variants and Solution Approaches”. In: *Journal of Advanced Transportation* 2019 (May 2019), pp. 1–48.

[75] Emanuel Falkenauer. “A hybrid grouping genetic algorithm for bin packing”. In: *J. Heuristics* 2.1 (1996), pp. 5–30.

[76] Dominique Feillet. “A tutorial on column generation and branch-and-price for vehicle routing problems”. In: *4OR* 8.4 (2010), pp. 407–424.

[77] Matteo Fischetti, Michele Monaci, and Domenico Salvagnin. “Mixed-integer linear programming heuristics for the prepack optimization problem”. In: *Discrete Optimization* 22 (2016). SI: ISCO 2014, pp. 195–205.

[78] Marshall L. Fisher and Ramchandran Jaikumar. “A generalized assignment heuristic for vehicle routing”. In: *Networks* 11.2 (1981), pp. 109–124.

[79] Krzysztof Fleszar and Christoforos Charalambous. “Average-weight-controlled bin-oriented heuristics for the one-dimensional bin-packing problem”. In: *Eur. J. Oper. Res.* 210.2 (2011), pp. 176–184.

[80] Krzysztof Fleszar and Khalil S. Hindi. “New heuristics for one-dimensional bin-packing”. In: *Comput. Oper. Res.* 29.7 (2002), pp. 821–839.

[81] L. R. Ford and D. R. Fulkerson. “A Suggested Computation for Maximal Multi-Commodity Network Flows”. In: *Manage. Sci.* 5.1 (Oct. 1958), pp. 97–101.

[82] John Forrest and Robin Lougee-Heimer. “CBC user guide”. In: *Emerging theory, methods, and applications* (2005), pp. 257–277.

[83] John W. Fowler and Lars Mönch. “A survey of scheduling with parallel batch (p-batch) processing”. In: *Eur. J. Oper. Res.* 298.1 (2022), pp. 1–24.

[84] J. B. G. Frenk and G. Galambos. “Hybrid Next-Fit Algorithm for the Two-Dimensional Rectangle Bin-Packing Problem”. In: *Computing* 39.3 (Dec. 1987), pp. 201–217.

[85] Alex S. Fukunaga and Richard E. Korf. "Bin Completion Algorithms for Multicontainer Packing, Knapsack, and Covering Problems". In: *J. Artif. Intell. Res.* 28 (2007), pp. 393–429.

[86] Fabio Furini and Enrico Malaguti. "Models for the two-dimensional two-stage cutting stock problem with multiple stock size". In: *Comput. Oper. Res.* 40.8 (2013), pp. 1953–1962.

[87] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.

[88] Jean-Michel Gauthier and Gerard Ribière. "Experiments in mixed-integer linear programming using pseudo-costs". In: *Math. Program.* 12.1 (1977), pp. 26–47.

[89] Ridvan Gedik, Emre Kirac, Ashlea Bennett Milburn, and Chase Rainwater. "A constraint programming approach for the team orienteering problem with time windows". In: *Computers & Industrial Engineering* 107 (2017), pp. 178–195.

[90] Ian P. Gent and Toby Walsh. "The TSP phase transition". In: *Artificial Intelligence* 88.1 (1996), pp. 349–358.

[91] Arthur Geoffrion. "Generalized Benders Decomposition". In: *Journal of Optimization Theory and Applications* 10 (Oct. 1972), pp. 237–260.

[92] Hamza Gharsellaoui and Hamadi Hasni. "An hybrid genetic algorithm for two-dimensional cutting problems using guillotine cuts". In: (Apr. 2012).

[93] P. C. Gilmore and R. E. Gomory. "A Linear Programming Approach to the Cutting-Stock Problem". In: *Operations Research* 9.6 (1961), pp. 849–859.

[94] P.C Gilmore and Ralph Gomory. "A Linear Programming Approach to the Cutting Stock Problem—Part II". In: *Operations Research* 11 (Dec. 1963).

[95] Bruce L. Golden, Thomas L. Magnanti, and H. Q. Nguyen. "Implementing vehicle routing algorithms". In: *Networks* 7.2 (1977), pp. 113–148.

[96] Solomon W. Golomb and Leonard D. Baumert. "Backtrack Programming". In: *J. ACM* 12.4 (1965), pp. 516–524.

[97] R.E Gomory. "Outline of an Algorithm for Integer Solutions to Linear Programs". In: *Bulletin of the American Mathematical Society* 64 (1958), pp. 275–278.

[98] José Fernando Gonçalves. "A hybrid genetic algorithm-heuristic for a two-dimensional orthogonal packing problem". In: *Eur. J. Oper. Res.* 183.3 (2007), pp. 1212–1229.

[99] José Fernando Gonçalves and Mauricio G. C. Resende. "A parallel multi-population genetic algorithm for a constrained two-dimensional orthogonal packing problem". In: *J. Comb. Optim.* 22.2 (2011), pp. 180–201.

[100] José Fernando Gonçalves and Gerhard Wäscher. “A MIP model and a biased random-key genetic algorithm based approach for a two-dimensional cutting problem with defects”. In: *Eur. J. Oper. Res.* 286.3 (2020), pp. 867–882.

[101] Stéphane Grandcolas and Cyril Pain-Barre. “A hybrid metaheuristic for the two-dimensional strip packing problem”. In: *Ann. Oper. Res.* 309.1 (2022), pp. 79–102.

[102] Ignacio E. Grossmann, Ignacio Quesada, Ramesh Raman, and Vasilios T. Voudouris. “Mixed-Integer Optimization Techniques for the Design and Scheduling of Batch Processes”. In: 1996.

[103] Oktay Günlük. *Cutting Planes for Mixed-Integer Programming: Theory and Practice.* Apr. 2018.

[104] Oktay Günlük and Yves Pochet. “Mixing mixed-integer inequalities”. In: *Math. Program.* 90.3 (2001), pp. 429–457.

[105] Jatinder N. D. Gupta and Johnny C. Ho. “A new heuristic algorithm for the one-dimensional bin-packing problem”. In: *Production Planning & Control* 10.6 (1999), pp. 598–603.

[106] Gurobi Optimization, LLC. *Gurobi Optimizer Reference Manual.* `https://www.gurobi.com`. 2022.

[107] Andy Ham. “Flexible job shop scheduling problem for parallel batch processing machine with compatible job families”. In: *Applied Mathematical Modelling* 45 (2017), pp. 551–562.

[108] Andy Ham, J.W. Fowler, and Eray Cakici. “Constraint Programming Approach for Scheduling Jobs with Release Times, Non-identical Sizes, and Incompatible Families on Parallel Batching Machines”. In: *IEEE Transactions on Semiconductor Manufacturing* PP (Aug. 2017), pp. 1–1.

[109] Andy M. Ham and Eray Cakici. “Flexible job shop scheduling problem with parallel batch processing machines: MIP and CP approaches”. In: *Computers & Industrial Engineering* 102 (2016), pp. 160–165.

[110] Sönke Hartmann. “Packing problems and project scheduling models: an integrating perspective”. In: *J. Oper. Res. Soc.* 51.9 (2000), pp. 1083–1092.

[111] David Hartvigsen and Yanjun Li. “Valid Inequalities for the 1-Restricted Simple 2-Matching Polytope”. In: *Aussois Combinatorial Optimization Workshop.* 2014.

[112] Joseph El Hayek, Aziz Moukrim, and Stéphane Nègre. “New resolution algorithm and pretreatments for the two-dimensional bin-packing problem”. In: *Comput. Oper. Res.* 35.10 (2008), pp. 3184–3201.

[113] Pascal Van Hentenryck and Jean-Philippe Carillon. “Generality versus Specificity: An Experience with AI and OR Techniques”. In: *Proceedings of the Seventh AAAI National Conference on Artificial Intelligence*. AAAI’88. Saint Paul, Minnesota: AAAI Press, 1988, pp. 660–664.

[114] “Heuristic and Metaheuristic Approaches for a Class of Two-Dimensional Bin Packing Problems”. In: *INFORMS J. on Computing* 11.4 (Nov. 1999), pp. 345–357.

[115] Sandy Heydrich and Andreas Wiese. “Faster Approximation Schemes for the Two-Dimensional Knapsack Problem”. In: *ACM Trans. Algorithms* 15.4 (2019), 47:1–47:28.

[116] Sandy Heydrich and Andreas Wiese. “Faster approximation schemes for the two-dimensional knapsack problem”. In: *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*. Ed. by Philip N. Klein. SIAM, 2017, pp. 79–98.

[117] Willem-Jan van Hoeve. *Introduction to Constraint Programming*. Sept. 2012.

[118] A.J. Hoffman. “On eigenvalues and colorings of graphs”. In: *Graph Theory and its Applications*. Academic Press, 1970, pp. 79–91.

[119] Hossein Hojabri, Michel Gendreau, Jean-Yves Potvin, and Louis-Martin Rousseau. “Large neighborhood search with constraint programming for a vehicle routing problem with synchronization constraints”. In: *Computers Operations Research* 92 (2018), pp. 87–97.

[120] John Hooker. *Benders Decomposition*. June 2016.

[121] John Hooker and Greger Ottosson. “Logic-based Benders’ Decomposition”. In: *Mathematical Programming, Series B* 96 (Apr. 2003), pp. 33–60.

[122] John N. Hooker. *Integrated methods for optimization*. Vol. 100. International series in operations research and management science. Springer, 2007.

[123] IBM. *Arithmetic constraints*. `https://www.ibm.com/docs/en/icos/22.1.0?topic=constraints-arithmetic`. Accessed: 2022-05-10.

[124] IBM. *Constraint propagation*. `https://www.ibm.com/docs/en/icos/22.1.0?topic=optimizer-constraint-propagation`. Accessed: 2022-05-10.

[125] IBM. *Count constraint*. `https://www.ibm.com/docs/en/icos/22.1.0?topic=functions-count`. Accessed: 2022-05-10.

[126] IBM. *Cumul functions in CP Optimizer*. `https://www.ibm.com/docs/en/icos/22.1.0?topic=c-cumul-functions-in-cp-optimizer`. Accessed: 2022-05-10.

[127] IBM. *Element constraint*. `https://www.ibm.com/docs/en/icos/22.1.0?topic=f-element`. Accessed: 2022-05-10.

[128] IBM. *ILOG CPLEX Optimization Studio.* `https://www.ibm.com/ca-en/products/ilog-cplex-optimization-studio`.

[129] IBM. *ILOG CPLEX Reference Manual.* Tech. rep. 2003.

[130] IBM. *Interval variables in CP Optimizer.* `https://www.ibm.com/docs/en/icos/22.1.0?topic=c-interval-variables-in-cp-optimizer`. Accessed: 2022-05-10.

[131] IBM. *Lexicographic constraint.* `https://www.ibm.com/docs/en/icos/22.1.0?topic=variables-lexicographic-constraint`. Accessed: 2022-05-10.

[132] IBM. *Logical constraints.* `https://www.ibm.com/docs/en/icos/22.1.0?topic=optimizer-logical-constraints`. Accessed: 2022-05-10.

[133] IBM. *Packing constraint.* `https://www.ibm.com/docs/en/icos/22.1.0?topic=variables-packing-constraint`. Accessed: 2022-05-10.

[134] IBM. *Piecewise linear and stepwise functions.* `https://perso.ensta-paris.fr/~diam/ro/online/cplex/cplex1271/OPL_Studio/opllangref/topics/opl_langref_scheduling_pwstep.html`.

[135] IBM. *State functions in CP Optimizer.* `https://www.ibm.com/docs/en/icos/22.1.0?topic=c-state-functions-in-cp-optimizer`. Accessed: 2022-05-10.

[136] Maksud Ibrahimov, Arvind Mohais, and Zbigniew Michalewicz. "Global Optimization in Supply Chain Operations". In: *Natural Intelligence for Scheduling, Planning and Packing Problems.* Ed. by Raymond Chiong and Sandeep Dhakal. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 1–28.

[137] Manuel Iori, Vinícius L. de Lima, Silvano Martello, Flávio K. Miyazawa, and Michele Monaci. "Exact solution techniques for two-dimensional cutting and packing". In: *European Journal of Operational Research* 289.2 (2021), pp. 399–415.

[138] Joxan Jaffar and Michael J. Maher. "Constraint Logic Programming: A Survey". In: *J. Log. Program.* 19/20 (1994), pp. 503–581.

[139] David Johnson. "Fast Algorithms for Bin Packing". In: *J. Comput. Syst. Sci.* 8.3 (1974), pp. 272–314.

[140] David Johnson. "Near-Optimal Bin Packing Algorithms". PhD thesis. Massachusetts Institute of Technology, 1968.

[141] C. Joncour, S. Michel, R. Sadykov, D. Sverdlov, and F. Vanderbeck. "Column Generation based Primal Heuristics". In: *Electronic Notes in Discrete Mathematics* 36 (2010). ISCO 2010 - International Symposium on Combinatorial Optimization, pp. 695–702.

[142] John J. Kanet, Sanjay Ahire, and Michael F. Gorman. "Constraint Programming for Scheduling". In: *Handbook of Scheduling - Algorithms, Models, and Performance Analysis.* Ed. by Joseph Y.-T. Leung. Chapman and Hall/CRC, 2004.

[143] Leonid Kantorovich. “The Mathematical Method of Production Planning and Organization”. In: *Management Science* 6 (1939).

[144] Gio K. Kao, Edward C. Sewell, and Sheldon H. Jacobson. “A branch, bound, and remember algorithm for the $1|r_i|\Sigma t_i$ scheduling problem”. In: *J. Sched.* 12.2 (2009), pp. 163–175.

[145] Konstantinos Kaparis and Adam N. Letchford. “Separation algorithms for 0-1 knapsack polytopes”. In: *Math. Program.* 124.1-2 (2010), pp. 69–91.

[146] Narendra Karmarkar. “A new polynomial-time algorithm for linear programming”. In: *Comb.* 4.4 (1984), pp. 373–396.

[147] Mitsutoshi Kenmochi, Takashi Imamichi, Koji Nonobe, Mutsunori Yagiura, and Hiroshi Nagamochi. “Exact algorithms for the two-dimensional strip packing problem with and without rotations”. In: *Eur. J. Oper. Res.* 198.1 (2009), pp. 73–83.

[148] Arindam Khan, Aditya Lonkar, Arnab Maiti, Amatya Sharma, and Andreas Wiese. “Tight Approximation Algorithms for Two Dimensional Guillotine Strip Packing”. In: *CoRR* (2022).

[149] Arindam Khan, Arnab Maiti, Amatya Sharma, and Andreas Wiese. “On Guillotine Separable Packings for the Two-Dimensional Geometric Knapsack Problem”. In: *37th International Symposium on Computational Geometry, SoCG 2021, June 7-11, 2021, Buffalo, NY, USA (Virtual Conference).* Ed. by Kevin Buchin and Éric Colin de Verdière. Vol. 189. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 48:1–48:17.

[150] Ali Khanafer, François Clautiaux, and El-Ghazali Talbi. “Tree-decomposition based heuristics for the two-dimensional bin packing problem with conflicts”. In: *Comput. Oper. Res.* 39.1 (2012), pp. 54–63.

[151] Joris Kinable, Willem-Jan van Hoeve, and Stephen F. Smith. “Optimization Models for a Real-World Snow Plow Routing Problem”. In: *Integration of AI and OR Techniques in Constraint Programming - 13th International Conference, CPAIOR 2016.* Vol. 9676. Lecture Notes in Computer Science. Springer, 2016, pp. 229–245.

[152] Richard E. Korf. “A New Algorithm for Optimal Bin Packing”. In: *Proceedings of the Eighteenth National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence, July 28 - August 1, 2002, Edmonton, Alberta, Canada.* Ed. by Rina Dechter, Michael J. Kearns, and Richard S. Sutton. AAAI Press / The MIT Press, 2002, pp. 731–736.

[153] Richard E. Korf. “An Improved Algorithm for Optimal Bin Packing”. In: *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003.* Ed. by Georg Gottlob and Toby Walsh. Morgan Kaufmann, 2003, pp. 1252–1258.

[154] Richard E. Korf. “Optimal Rectangle Packing: Initial Results”. In: *Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling (ICAPS 2003).* AAAI, 2003, pp. 287–295.

[155] Richard E. Korf. “Optimal Rectangle Packing: New Results”. In: *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS 2004).* AAAI, 2004, pp. 142–149.

[156] Sebastian Kosch and J. Christopher Beck. “A New MIP Model for Parallel-Batch Scheduling with Non-identical Job Sizes”. In: *Integration of AI and OR Techniques in Constraint Programming.* Ed. by Helmut Simonis. Cham: Springer International Publishing, 2014, pp. 55–70.

[157] Wen-Yang Ku and J. Christopher Beck. “Mixed Integer Programming models for job shop scheduling: A computational analysis”. In: *Computers Operations Research* 73 (2016), pp. 165–173.

[158] Philippe Laborie and Jerome Rogerie. “Temporal linear relaxation in IBM ILOG CP Optimizer”. In: *Journal of Scheduling* 19.4 (2016), pp. 391–400.

[159] Philippe Laborie, Jerome Rogerie, Paul Shaw, and Petr Vilím. “IBM ILOG CP optimizer for scheduling: 20+ years of scheduling with constraints at IBM/ILOG”. In: *Constraints* 23 (Mar. 2018).

[160] Philippe Laborie, Jerome Rogerie, Paul Shaw, and Petr Vilím. “Reasoning with Conditional Time-Intervals. Part II: An Algebraical Model for Resources.” In: Jan. 2009.

[161] A. H. Land and A. G. Doig. “An Automatic Method of Solving Discrete Programming Problems”. In: *Econometrica* 28.3 (1960), pp. 497–520.

[162] Gilbert Laporte. “The vehicle routing problem: An overview of exact and approximate algorithms”. In: *European Journal of Operational Research* 59.3 (1992), pp. 345–358.

[163] Jongsung Lee, Byung-in kim, and Andrew L. Johnson. “A two-dimensional bin packing problem with size changeable items for the production of wind turbine flanges in the open die forging industry”. In: *IIE Transactions* 45.12 (2013), pp. 1332–1344.

[164] Stephen C. H. Leung, Defu Zhang, and Kwang Mong Sim. “A two-stage intelligent search algorithm for the two-dimensional strip packing problem”. In: *Eur. J. Oper. Res.* 215.1 (2011), pp. 57–69.

[165] Stephen C. H. Leung, Defu Zhang, Changle Zhou, and Tao Wu. “A hybrid simulated annealing metaheuristic algorithm for the two-dimensional knapsack packing problem”. In: *Comput. Oper. Res.* 39.1 (2012), pp. 64–73.

[166] Stephen C. H. Leung, Xiyue Zhou, Defu Zhang, and Jiemin Zheng. “Extended guided tabu search and a new packing algorithm for the two-dimensional loading vehicle routing problem”. In: *Comput. Oper. Res.* 38.1 (2011), pp. 205–215.

[167] T. W. Leung, Chi Kin Chan, and Marvin D. Troutt. “Application of a mixed simulated annealing-genetic algorithm heuristic for the two-dimensional orthogonal packing problem”. In: *Eur. J. Oper. Res.* 145.3 (2003), pp. 530–542.

[168] Kunpeng Li, Hailan Liu, Yong Wu, and Xianhao Xu. “A two-dimensional bin-packing problem with conflict penalties”. In: *International Journal of Production Research* 52.24 (2014), pp. 7223–7238.

[169] Ching-Jong Liao and Li-Man Liao. “Improved MILP models for two-machine flowshop with batch processing machines”. In: *Mathematical and Computer Modelling* 48.7 (2008), pp. 1254–1264.

[170] Chang Liu, Dionne M. Aleman, and J. Christopher Beck. “Modelling and Solving the Senior Transportation Problem”. In: *Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 15th International Conference, CPAIOR 2018, Delft, The Netherlands, June 26-29, 2018, Proceedings.* Ed. by Willem-Jan van Hoeve. Vol. 10848. Lecture Notes in Computer Science. Springer, 2018, pp. 412–428.

[171] Qiang Liu, Jiawei Zeng, Hao Zhang, and Lijun Wei. “A Heuristic for the Two-Dimensional Irregular Bin Packing Problem with Limited Rotations”. In: *Trends in Artificial Intelligence Theory and Applications. Artificial Intelligence Practices - 33rd International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems, IEA/AIE 2020, Kitakyushu, Japan, September 22-25, 2020, Proceedings.* Ed. by Hamido Fujita, Philippe Fournier-Viger, Moonis Ali, and Jun Sasaki. Vol. 12144. Lecture Notes in Computer Science. Springer, 2020, pp. 268–279.

[172] Andrea Lodi, Silvano Martello, and Daniele Vigo. “Approximation algorithms for the oriented two-dimensional bin packing problem”. In: *European Journal of Operational Research* 112.1 (1999), pp. 158–166.

[173] Andrea Lodi, Silvano Martello, and Daniele Vigo. “Heuristic and Metaheuristic Approaches for a Class of Two-Dimensional Bin Packing Problems”. In: *INFORMS J. Comput.* 11.4 (1999), pp. 345–357.

[174] Andrea Lodi, Silvano Martello, and Daniele Vigo. “Models and Bounds for Two-Dimensional Level Packing Problems”. In: *J. Comb. Optim.* 8.3 (2004), pp. 363–379.

[175] Andrea Lodi, Silvano Martello, and Daniele Vigo. “Neighborhood Search Algorithm for the Guillotine Non-Oriented Two-Dimensional Bin Packing Problem”. In: 1999.

[176] Andrea Lodi, Silvano Martello, and Daniele Vigo. “TSpack: A Unified Tabu Search Code for Multi-Dimensional Bin Packing Problems”. In: *Annals of Operations Research* 131 (2004), pp. 203–213.

[177] Andrea Lodi and Michele Monaci. “Integer linear programming models for 2-staged two-dimensional Knapsack problems”. In: *Math. Program.* 94.2-3 (2003), pp. 257–278.

[178] Kok-Hua Loh, Bruce L. Golden, and Edward A. Wasil. “Solving the one-dimensional bin packing problem with a weight annealing heuristic”. In: *Comput. Oper. Res.* 35.7 (2008), pp. 2283–2291.

[179] Chris Loken, Daniel Gruner, Leslie Groer, Richard Peltier, Neil Bunn, Michael Craig, Teresa Henriques, Jillian Dempsey, Ching-Hsing Yu, Joseph Chen, L Jonathan Dursi, Jason Chong, Scott Northrup, Jaime Pinto, Neil Knecht, and Ramses Van Zon. “SciNet: Lessons Learned from Building a Power-efficient Top-20 System and Data Centre”. In: *Journal of Physics: Conference Series* 256 (Nov. 2010), p. 012026.

[180] Qiang Luo, Yunqing Rao, Xiaoqiang Guo, and Bing Du. “A biased genetic algorithm hybridized with VNS for the two-dimensional knapsack packing problem with defects”. In: *Appl. Soft Comput.* 118 (2022), p. 108479.

[181] Yiqing L. Luo and J. Christopher Beck. “Packing by Scheduling: Using Constraint Programming to Solve a Complex 2D Cutting Stock Problem”. In: *Integration of Constraint Programming, Artificial Intelligence, and Operations Research.* Ed. by Pierre Schaus. Cham: Springer International Publishing, 2022, pp. 249–265.

[182] Rita Macedo, Cláudio Alves, and J.M. Valério de Carvalho. “Arc-flow model for the two-dimensional guillotine cutting stock problem”. In: *Computers Operations Research* 37.6 (2010), pp. 991–1001.

[183] Alan K. Mackworth. “Consistency in Networks of Relations”. In: *Artif. Intell.* 8.1 (1977), pp. 99–118.

[184] Vicky H. Mak-Hau, John Yearwood, and William Moran. “Knowledge engineering mixed-integer linear programming: constraint typology”. In: *ArXiv* abs/2102.12574 (2021).

[185] Arnaud Malapert, Christelle Guéret, and Louis-Martin Rousseau. “A constraint programming approach for a batch processing problem with non-identical job sizes”. In: *European Journal of Operational Research* 221.3 (2012), pp. 533–545.

[186] Paolo Marocco, Domenico Ferrero, Emanuele Martelli, Massimo Santarelli, and Andrea Lanzini. “An MILP approach for the optimal design of renewable battery-hydrogen energy systems for off-grid insular communities”. In: *Energy Conversion and Management* 245 (2021), p. 114564.

[187] Silvano Martello and Paolo Toth. *Knapsack Problems: Algorithms and Computer Implementations*. USA: John Wiley Sons, Inc., 1990.

[188] Mateus Martin, Ernesto G. Birgin, Rafael D. Lobato, Reinaldo Morabito, and Pedro Munari. "Models for the two-dimensional rectangular single large placement problem with guillotine cuts and constrained pattern". In: *Int. Trans. Oper. Res.* 27.2 (2020), pp. 767–793.

[189] Mateus Martin, Pedro Henrique Del Bianco Hokama, Reinaldo Morabito, and Pedro Munari. "The constrained two-dimensional guillotine cutting problem with defects: an ILP formulation, a Benders decomposition and a CP-based algorithm". In: *Int. J. Prod. Res.* 58.9 (2020), pp. 2712–2729.

[190] Mateus Martin, Reinaldo Morabito, and Pedro Munari. "A top-down cutting approach for modeling the constrained two- and three-dimensional guillotine cutting problems". In: *J. Oper. Res. Soc.* 72.12 (2021), pp. 2755–2769.

[191] M. Mesyagutov, G. Scheithauer, and G. Belov. "LP bounds in various constraint programming approaches for orthogonal packing". In: *Computers & Operations Research* 39.10 (Oct. 2012), pp. 2425–2438.

[192] Andrew Miller and Laurence Wolsey. "Tight formulations for some simple mixed integer programs and convex objective integer programs". In: *Math. Program.* 98 (Sept. 2003), pp. 73–88.

[193] Juan José Miranda-Bront, Isabel Méndez-Díaz, and Paula Zabala. "Facets and valid inequalities for the time-dependent travelling salesman problem". In: *European Journal of Operational Research* 236.3 (2014). Vehicle Routing and Distribution Logistics, pp. 891–902.

[194] Michael D. Moffitt and Martha E. Pollack. "Optimal Rectangle Packing: A Meta-CSP Approach". In: *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling, (ICAPS 2006)*. AAAI, 2006, pp. 93–102.

[195] Michele Monaci and Paolo Toth. "A Set-Covering-Based Heuristic Approach for Bin-Packing Problems". In: *INFORMS J. Comput.* 18.1 (2006), pp. 71–85.

[196] Clyde L. Monma and Chris N. Potts. "On the Complexity of Scheduling with Batch Setup Times". In: *Operations Research* 37.5 (1989), pp. 798–804.

[197] David Morrison, Jason Sauppe, Wenda Zhang, Sheldon Jacobson, and Edward Sewell. "Cyclic best first search: Using contours to guide branch-and-bound algorithms: Cyclic Best First Search". In: *Naval Research Logistics (NRL)* 64 (Mar. 2017).

[198] David R. Morrison, Sheldon H. Jacobson, Jason J. Sauppe, and Edward C. Sewell. "Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning". In: *Discret. Optim.* 19 (2016), pp. 79–102.

[199] Narendra Jussien and Guillaume Rochart and and Xavier Lorca. *Choco: an open source Java constraint programming library.* 2008.

[200] Vera Neidlein, André Scholz, and Gerhard Wäscher. “SLOPPGEN: a problem generator for the two-dimensional rectangular single large object placement problem with defects”. In: *Int. Trans. Oper. Res.* 23.1-2 (2016), pp. 173–186.

[201] G.L. Nemhauser and L.A. Wolsey. “A recursive procedure to generate all cuts for 0–1 mixed integer programs”. In: *Mathematical Programming* 46 (1900), pp. 379–390.

[202] George L. Nemhauser and Laurence A. Wolsey. *Integer and Combinatorial Optimization.* Wiley interscience series in discrete mathematics and optimization. Wiley, 1988.

[203] FICO Xpress Optimization. *FICO Xpress Solver.* `https://www.fico.com/en/products/fico-xpress-solver`.

[204] Manfred Padberg and Giovanni Rinaldi. “A Branch-and-Cut Algorithm for the Resolution of Large-Scale Symmetric Traveling Salesman Problems”. In: *SIAM Review* 33.1 (1991), pp. 60–100. eprint: `https://doi.org/10.1137/1033004`.

[205] Célia Paquay, Sabine Limbourg, and Michaël Schyns. “A tailored two-phase constructive heuristic for the three-dimensional Multiple Bin Size Bin Packing Problem with transportation constraints”. In: *Eur. J. Oper. Res.* 267.1 (2018), pp. 52–64.

[206] Laurent Perron. “Operations Research and Constraint Programming at Google”. In: *Principles and Practice of Constraint Programming - CP 2011 - 17th International Conference, CP 2011, Perugia, Italy, September 12-16, 2011. Proceedings.* Ed. by Jimmy Ho-Man Lee. Vol. 6876. Lecture Notes in Computer Science. Springer, 2011, p. 2.

[207] Jean-Claude Picard and Maurice Queyranne. “The Time-Dependent Traveling Salesman Problem and Its Application to the Tardiness Problem in One-Machine Scheduling”. In: *Operations Research* 26.1 (1978), pp. 86–110.

[208] David Pisinger and Mikkel Sigurd. “The two-dimensional bin packing problem with variable bin sizes and costs”. In: *Discret. Optim.* 2.2 (2005), pp. 154–167.

[209] Riccardo Poli, John R. Woodward, and Edmund K. Burke. “A histogram-matching approach to the evolution of bin-packing strategies”. In: *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2007, 25-28 September 2007, Singapore.* IEEE, 2007, pp. 3500–3507.

[210] Sergey Polyakovskiy and Rym M’Hallah. “A lookahead matheuristic for the unweighed variable-sized two-dimensional bin packing problem”. In: *Eur. J. Oper. Res.* 299.1 (2022), pp. 104–117.

[211] Marcelo Ponce, Ramses van Zon, Scott Northrup, Daniel Gruner, Joseph Chen, Fatih Ertinaz, Alexey Fedoseev, Leslie Groer, Fei Mao, Bruno C. Mundim, Mike Nolta, Jaime Pinto, Marco Saldarriaga, Vladimir Slavnic, Erik Spence, Ching-Hsing Yu, and W. Richard Peltier. “Deploying a Top-100 Supercomputer for Large Parallel Workloads: the Niagara Supercomputer”. In: *Proceedings of the Practice and Experience in Advanced Research Computing on Rise of the Machines (learning), PEARC 2019, Chicago, IL, USA, July 28 - August 01, 2019.* ACM, 2019, 34:1–34:8.

[212] Florian A. Potra and Stephen J. Wright. “Interior-point methods”. In: *Journal of Computational and Applied Mathematics* 124.1 (2000). Numerical Analysis 2000. Vol. IV: Optimization and Nonlinear Equations, pp. 281–302.

[213] Chris N. Potts and Mikhail Y. Kovalyov. “Scheduling with batching: A review”. In: *European Journal of Operational Research* 120.2 (2000), pp. 228–249.

[214] Jakob Puchinger and Günther R. Raidl. “Models and algorithms for three-stage two-dimensional bin packing”. In: *Eur. J. Oper. Res.* 183.3 (2007), pp. 1304–1327.

[215] Chase Rainwater, Joseph Geunes, and H. Edwin Romeijn. “The generalized assignment problem with flexible jobs”. In: *Discrete Applied Mathematics* 157 (2009), pp. 49–67.

[216] Jean-Charles Régin. “Global Constraints and Filtering Algorithms”. In: *Constraint and Integer Programming: Toward a Unified Methodology.* Ed. by Michela Milano. Boston, MA: Springer US, 2004, pp. 89–135.

[217] Philipp Rohlfshagen and John A. Bullinaria. “A genetic algorithm with exon shuffling crossover for hard bin packing problems”. In: *Genetic and Evolutionary Computation Conference, GECCO 2007, Proceedings, London, England, UK, July 7-11, 2007.* Ed. by Hod Lipson. ACM, 2007, pp. 1365–1371.

[218] Armin Scholl, Robert Klein, and Christian Jürgens. “Bison: A fast hybrid procedure for exactly solving the one-dimensional bin packing problem”. In: *Comput. Oper. Res.* 24.7 (1997), pp. 627–645.

[219] Alexander Schrijver. “Theory of linear and integer programming”. In: *Computers and Mathematics with application* 36.8 (1998).

[220] Mahdi Sharifzadeh, Marti Cortada Garcia, and Nilay Shah. “Supply chain network design and operation: Systematic decision-making for centralized, distributed, and mobile biofuel production using mixed integer linear programming (MILP) under uncertainty”. In: *Biomass and Bioenergy* 81 (2015), pp. 401–414.

[221] Paul Shaw. “A Constraint for Bin Packing”. In: *Principles and Practice of Constraint Programming – CP 2004.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 648–662.

[222] Lei Shi, John Furlong, and Runxin Wang. “Empirical evaluation of vector bin packing algorithms for energy efficient data centers”. In: *2013 IEEE Symposium on Computers and Communications, ISCC 2013, Split, Croatia, 7-10 July, 2013.* IEEE Computer Society, 2013, pp. 9–15.

[223] Elsa Silva, Filipe Pereira Alvelos, and José M. Valério de Carvalho. “An integer programming model for two- and three-stage two-dimensional cutting stock problems”. In: *Eur. J. Oper. Res.* 205.3 (2010), pp. 699–708.

[224] Jefferson L. M. da Silveira, Eduardo C. Xavier, and Flávio Keidi Miyazawa. “Two-dimensional strip packing with unloading constraints”. In: *Discret. Appl. Math.* 164 (2014), pp. 512–521.

[225] Helmut Simonis and Barry O’Sullivan. “Almost Square Packing”. In: *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems - 8th International Conference, CPAIOR 2011, Berlin, Germany, May 23-27, 2011. Proceedings.* Ed. by Tobias Achterberg and J. Christopher Beck. Vol. 6697. Lecture Notes in Computer Science. Springer, 2011, pp. 196–209.

[226] Helmut Simonis and Barry O’Sullivan. “Search Strategies for Rectangle Packing”. In: *Principles and Practice of Constraint Programming, 14th International Conference, CP 2008, Sydney, Australia, September 14-18, 2008. Proceedings.* Ed. by Peter J. Stuckey. Vol. 5202. Lecture Notes in Computer Science. Springer, 2008, pp. 52–66.

[227] Alok Singh and Ashok Kumar Gupta. “Two heuristics for the one-dimensional bin-packing problem”. In: *OR Spectr.* 29.4 (2007), pp. 765–781.

[228] David J Sklan and I. Dariel. “Diet planning for humans using mixed-integer linear programming”. In: *British Journal of Nutrition* 70 (1993), pp. 27–35.

[229] Barbara Smith. *Modelling for Constraint Programming.* Sept. 2005.

[230] Takehide Soh, Katsumi Inoue, Naoyuki Tamura, Mutsunori Banbara, and Hidetomo Nabeshima. “A SAT-based Method for Solving the Two-dimensional Strip Packing Problem”. In: *Fundam. Informaticae* 102.3-4 (2010), pp. 467–487.

[231] Tomasz Szczygiel. “CLP Approaches to 2D Angle Placements”. In: *CoRR* cs.PL/0109066 (2001).

[232] Tanya Y. Tang and J. Christopher Beck. “CP and Hybrid Models for Two-Stage Batching and Scheduling”. In: *Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 17th International Conference, CPAIOR 2020.* Vol. 12296. Lecture Notes in Computer Science. Springer, 2020, pp. 431–446.

[233] Robert Endre Tarjan. “Depth-First Search and Linear Graph Algorithms”. In: *SIAM J. Comput.* 1.2 (1972), pp. 146–160.

[234] Edward P. K. Tsang. “Constraint Based Scheduling: Applying Constraint Programming to Scheduling Problems”. In: *J. Sched.* 6.4 (2003), pp. 413–414.

[235] Edward P. K. Tsang. *Foundations of constraint satisfaction.* Computation in cognitive science. Academic Press, 1993.

[236] Marius-Florinel Tudosoiu and Florin Pop. “Bin Packing Scheduling Algorithm with Energy Constraints in Cloud Computing”. In: *17th IEEE International Conference on Intelligent Computer Communication and Processing, ICCP 2021, Cluj-Napoca, Romania, October 28-30, 2021.* IEEE, 2021, pp. 77–84.

[237] Mustafa Kemal Tural. *Valid Inequalities for the Maximal Matching Polytope.* 2019.

[238] Darrell R. Ulm and Pearl Y. Wang. “Solving a Two-Dimensional Knapsack Problem on SIMD Computers”. In: *Proceedings of the 1992 International Conference on Parallel Processing, University of Michigan, An Arbor, Michigan, USA, August 17-21, 1992. Volume III: Algorithms & Applications.* Ed. by Quentin F. Stout. CRC Press, 1992, pp. 181–184.

[239] José Manuel Valério de Carvalho. “LP models for bin packing and cutting stock problems”. In: *European Journal of Operational Research* 141.2 (2002), pp. 253–273.

[240] Pamela H. Vance. “Branch-and-Price Algorithms for the One-Dimensional Cutting Stock Problem”. In: *Comput. Optim. Appl.* 9.3 (1998), pp. 211–228.

[241] Mathieu Van Vyve. “The Continuous Mixing Polyhedron”. In: *Math. Oper. Res.* 30.2 (2005), pp. 441–452.

[242] H. M Wagner. “The Dual Simplex Algorithm for Bounded Variables”. In: *Naval Research Logistics Quarterly* 5.3 (1958), pp. 257–261.

[243] Kiri Wagstaff, Claire Cardie, Seth Rogers, and Stefan Schrödl. “Constrained K-means Clustering with Background Knowledge”. In: *Proceedings of the Eighteenth International Conference on Machine Learning (ICML 2001), Williams College, Williamstown, MA, USA, June 28 - July 1, 2001.* Ed. by Carla E. Brodley and Andrea Pohoreckyj Danyluk. Morgan Kaufmann, 2001, pp. 577–584.

[244] Laurence A. Wolsey. “Faces for a linear inequality in 0-1 variables”. In: *Math. Program.* 8.1 (1975), pp. 165–178.

[245] Hao Zhang, Qiang Liu, Lijun Wei, Jiawei Zeng, Jiewu Leng, and Duxi Yan. “An iteratively doubling local search for the two-dimensional irregular bin packing problem with limited rotations”. In: *Comput. Oper. Res.* 137 (2022), p. 105550.