KU LEUVEN

FACULTY OF
ENGINEERING SCIENCE

Computerised Production Planning Optimisation for Industrial Weaving Machines

Thomas Sergeys

Thesis voorgedragen tot het behalen
van de graad van Master of Science
in de ingenieurswetenschappen:
computerwetenschappen, hoofdoptie
Mens-machinecommunicatie

Promotor:
Prof. dr. ir. Tias Guns

Evaluatoren:
Dimosthenis Tsouros
Andreas Nuyts

Begeleider:
Dimosthenis Tsouros

Academiejaar 2022 – 2023

Preface

This thesis represents the final chapter of my studies towards becoming an engineer. Hundreds of hours have gone into creating the text now lying before you. The final result could not have been achieved without the help of many individuals, for who I would like to present this message of gratitude.

I would like to thank my supervisor Prof. dr. ir. Tias Guns for giving me the opportunity of working on the presented research topic, for bringing me in touch with the lovely people in his research group, and for the kind words of encouragement throughout our meetings.

I would like to thank my assistant-supervisor Dimosthenis Tsouros for guiding me throughout this journey, for giving me constructive feedback, for getting me motivated again when I couldn't see the wood for the trees, and for generally making this thesis a pleasant experience.

I would like to thank Vandewiele nv for providing the theme of this research: industrial weave planning. A special thanks goes to Loes Brabant and Joren Reynaert for their warm welcome, for the guided tour of their Experience Center, and for bringing me up-to-date with the industry.

A more personal sign of appreciation goes to my parents and family, for always believing in me and supporting me throughout the last five years of hard work. For being there for me throughout all the ups and downs. Without them, I would never have been able to come as far as I am today and be able to present you this work.

I would like to thank my sports club *Roll & Skate Leuven* for providing the necessary physical activity as variation in between all the hard work. A thanks goes to all its members for accepting me with open arms and for generally being good friends.

A final thank-you goes to my 5^{th} grade maths teacher, Mrs. Ingrid Verlinden, for inspiring me to aim high and become a civil engineer.

Thomas Sergeys

Contents

Abstract

This thesis introduces the first formal description of the Weave Planning and Scheduling (WPAS) problem, an optimisation problem originating from the textile industry. Multiple constrained programming models get proposed, each tackling a sub-problem towards the complete problem formulation. The CP-Anchor model gets introduced for the Single Large Object Placement Problem (SLOPP) at the center of WPAS. With its novel anchor-based set-of-points formulation, it serves as a performant and extensible alternative to the typical guillotine formulations from literature. Several experiments demonstrate its suitability for large-scale packing problems found in practical applications, approximating the guillotine's performance whilst allowing for greater flexibility. To this end, a new set of benchmark suites gets proposed. The notion of due dates gets added using an iterative approach, allowing for the optimisation of a complete production schedule. It is compared against a one-shot global model, demonstrating its performance and practical advantages. At last, an implementation gets proposed for the domain-specific constraints associated with a weaving machine. Experiments further demonstrate the value of the novel CP-Anchor formulation.

Samenvatting

Deze thesis richt zich op het optimalisatieprobleem geassocieerd met de creatie van een productieplanning. Specifieker zal het gaan over de planning voor industriële weefmachines binnen de textielindustrie. De probleemomschrijving werd aangeboden door Vandewiele nv, een Belgische textielmachineproducent opgericht in 1880 en gelegen in Kortrijk. Om geweven textielen op een industriële schaal te kunnen produceren, wordt er gebruikgemaakt van grootschalige weefgetouwen. Deze kunnen meerdere stukken stof tegelijkertijd fabriceren, zowel langs elkaar overheen de breedte van de machine als achter elkaar in sequentie. Achteraf zal dit lappendeken worden uitgesneden om de beoogde losse textielen te bekomen. Bij het plannen van deze productie komt er echter een tal van regels kijken geassocieerd met onder andere de fysische beperkingen van het weefgetouw, de gewoontes binnen een weefbedrijf en de voorkeuren van de experts die de planning moeten goedkeuren. Traditioneel wordt dit werk manueel uitgevoerd, vaak startende van een eerder gebruikte en bewezen planning waarop de nodige aanpassingen worden toegepast. Dit kan echter leiden tot suboptimale resultaten. Dit optimalisatieprobleem is een ideale kandidaat voor automatisaties. Met behulp van algoritmische ondersteuningen kunnen er verbeteringen worden gemaakt op het gebied van efficiëntie, winstmarge, concurrentievermogen, enzovoort.

Het verrichte onderzoek had twee doelen tot zijn eind: **I)** het introduceren van de eerste formele omschrijving in literatuur van het Weave Planning and Scheduling (WPAS) optimalisatieprobleem en **II)** de creatie van nieuwe wiskundige modellen om het probleem op te lossen op een schaalbare en uitbreidbare manier. WPAS is ontbindbaar in verschillende lagen, met het Single Large Object Placement Problem (SLOPP) dat de kern van het probleem vormt. Het behoort tot de familie van de Cutting and Packing (C&P) problemen, waarbij kleinere rechthoeken orthogonaal moeten gepositioneerd worden binnen een grotere rechthoek zonder te overlappen. Hiervoor wordt een nieuw constraint programming model geïntroduceerd: CP-Anchor. Het model is gebaseerd op de set-of-points encodering die erg prominent is binnen het onderzoeksdomein van C&P. CP-Base formuleert op basis van deze encodering een naïeve implementatie, wat resulteert in schaalbaarheidsproblemen. Het aantal constraints die nodig zijn om de overlap van de rechthoeken te vermeiden neemt exponentieel toe met de grootte van het probleem. De performantie van dit model dient als referentiekader. CP-Guillotine, een model van Salem et al. (2023), lost het schaalbaarheidsprobleem op via een guillotineassumptie op de positioneringspatronen van de oplossingen. Dit maakt de formulering van niet-overlap overbodig doordat deze intrinsiek wordt aan de encodering. Het aantal constraints wordt sterk verminderd in ruil voor een significante toename in het aantal variabelen. Hoewel deze de performantie (in termen van rekentijd) verbetert, zorgt de assumptie voor restricties op de uitbreidbaarheid van de formulering. Deze uitbreidbaarheid is juist hetgeen dat nodig is om het complete WPAS probleem te kunnen representeren. De assumptie maakt ook enkele oplossingen niet langer mogelijk. CP-Anchor neemt een andere aanpak. Het maakt geen assumpties, net als CP-Base, maar introduceert een grid-structuur met "*anker-cellen*" voor de positionering van de rechthoeken. Het grote aantal aan niet-overlap constraints viel deels te wijten aan de degeneratie van de set-of-points encodering door lokale symmetrieën. De nieuwe grid-structuur limiteert elke rechthoek tot een kleine lokale regio (anker-cel), wat de lokale symmetrieën volledig wegwerkt. CP-Anchor dient als alternatief voor de populaire guillotine gebaseerde modellen uit de literatuur, zonder assumpties op de oplossingsruimte en met toegang tot de exacte posities van de rechthoeken. Dit laat toe om constraints veel lokaler to formuleren. Experimenten tonen aan dat de performantie erg dicht in de buurt komt van CP-Guillotine en het een grote verbetering is ten opzichte van CP-Base.

Als eerste uitbreiding bovenop SLOPP wordt de notie van tijd toegevoegd. Meerdere rechthoeken worden nu ingepakt om zo goed mogelijk aan de vraag van een productieschema te voldoen. Een one-shot globaal model (CP-OneShot) en een iteratieve aanpak (CP-DLNS) worden hiervoor geïntroduceerd. DLNS staat voor Delayed Large Neighbourhood Search. Het is een techniek die inspiratie neemt van Large Neighbourhood Search (LNS) en Delayed Column Generation (DCG). Experimenten tonen de voordelen van deze iteratieve aanpak aan, met betere performantie, ondersteuning voor grootschaligere problemen en tal van voordelen gerelateerd aan de meer dynamische aanpak. Zo heeft CP-DLNS ondersteuning voor interactief oplossen in een human-in-the-loop setting waarbij een expert feedback kan geven op de resultaten en het algoritme kan begeleiden.

Als laatste wordt CP-WPAS geïntroduceerd als de finale uitbreiding bovenop SLOPP, wat toelaat om het volledige WPAS probleem op te lossen. De modellen tot nu toe implementeren een generiek productieplannings-probleem. Het model wordt uitgebreid met de fysische restricties van weefgetouwen. Experimenten tonen hierbij het voordeel van de uitbreidbaarheid van CP-Anchor aan, gezien CP-Guillotine al haar performantievoorsprong hierbij verliest. De uitbreiding formuleren bovenop het grote aantal variabelen zorgt nu voor een explosie in aantal constraints.

Voor alle bovenvermelde experimenten werden nieuwe benchmark suites gecreëerd op basis van de SLOPPGEN probleemgenerator van Wäscher et al. (2014). Bestaande suites konden niet gebruikt worden, gezien deze niet ontworpen zijn voor de speciale Hoge Multipliciteit (HM) variant van SLOPP die in WPAS van toepassing is. Met experimenten op basis van deze nieuwe suites doet deze thesis onderzoek naar onder andere de meerwaarde van de nieuwe formuleringen, opzoek naar alternatieve

modellen om C&P problemen in realistische toepassingen op te lossen.

List of Figures

List of Tables

List of Abbreviations

Abbreviations

COP	Constrained Optimisation Problem
CP	Constrained Programming
CSP	Cutting Stock Problem
C&P	Cutting and Packing
DM	Decision Maker
HM	High Multiplicity
MLOPP	Multiple Large Object Placement Problem
MPM	Maximal Packing Multiplicity
MPS	Master Production Schedule
SLOPP	Single Large Object Placement Problem
OR	Operations Research
PAS	Planning and Scheduling
R_2D_SLOPP	Rectangular Two-Dimensional Single Large Object Placement Problem
WPAS	Weave Planning and Scheduling

Chapter 1

Intro

Textiles can be found all around us, starting from the moment we are born and most likely until our last breath. For centuries they have played important roles in human society, history, culture, and so on. They satisfy many use cases: clothing, furniture, reinforcements for car tyres [61] and even more exotic as part of the fibreglass substrate of a Printed Circuit Board or PCB [2].

The word "*textile*" comes from the Latin "*textilis*" (woven) which in turn is derived from "*textere*" (to weave). Whilst the original definition only refers to woven fabrics, the modern domain of *textile science* uses the term to refer to a larger variety of fiber-based materials. [41]

The first textiles are estimated to originate from the stone age, giving this old craft a rich history [41]. The invention of the powered loom took place in 1785 during the Industrial Revolution and was one of the most significant developments during this time period [8]. The weaving process, a technique of interlacing yarns at right angles in established sequences [41], was brought from homes and cottages to the factory. It quickly grew to become one of the largest industrial sectors and the machine's inventor Edmund Cartwright was made a Fellow of the Royal Society for his contributions to the British industry [17]. Today, it is one of the oldest still existing industries. Flanders Belgium, the region of origin of this thesis' author, is home to one of the largest textile producing clusters in Europe [1].

Problem Statement The target of most companies, including those in the weaving sector, is to satisfy the real or forecasted demand. In order to meet the requirements as effectively as possible, they need to plan and control their production accordingly. To this end, a ***Master Production Schedule*** (MPS) typically gets created. An MPS formulates the amount of each product that needs to be produced over a planning horizon together with a collection of concrete deadlines or due dates. This MPS subsequently serves as the basis for different production related plannings within the company, like material management, machine planning, labour planning, inventory management, and so on [14]. The objective of each of these plannings is to produce products in time as to satisfy the MPS, alongside other performance metrics utilised within the company.

Oftentimes the creation of these schedules and plannings gets performed manually by expert planners, according to a private conversation with Vandewiele nv[1]. The experts create a schedule/planning based on the current customer demand, practical restrictions, personal preference, etc. on a rolling horizon basis. As repeatedly creating a schedule from scratch is a quite tedious and time consuming task, planners often start from a previous schedule (which has proven to work well) on which they make the necessary changes[1]. Whilst reuse allows for a reduced workload, this can also lead to suboptimal schedules. As these schedules have an immediate impact on the company's efficiency, productivity, profit, competitiveness, etc. this is less than ideal.

This is where automation and computerised scheduling can play an important role. According to Satchell, automation describes "*the replacement of human activity by machine activities*" [68]. It can help in significantly reducing the workload of the expert planners whilst at the same time help manufacturers improve on-time delivery, reduce costs, allow for quick response to changing customer demands, reduce human error and much more [14]. Overall, automation can improve the consistency and quality of the production plannings.

The problem statement has graciously been provided by Vandewiele nv, a Belgian textile machine manufacturer. Founded in 1880 [3], they have grown to a worldwide market leader position in the design and fabrication of state-of-the-art machines for the textile industry [4]. Their customers utilise their weaving machines to produce textiles on an industrial scale. This thesis will focus on the challenge of creating an automated algorithmic approach to production planning. It is a valuable research topic as the problem can be found in many industrial sectors and has an immediate impact on a company's daily operations. Many of the existing solutions in literature are rather limited in their capabilities.

Approach One of the big challenges related to production planning is keeping the solution scalable and extensible. Many similar problem statements have already been tackled in the literature but their scope is rather limited. As will be discussed in *chapter 3 Related Literature*, they often tackle simplified variants called "*dummy problems*" instead of a complete real-world problem. Many of the preexisting approaches make assumptions on the solution space, significantly reducing the size of the search space. Whilst this improves the scalability and performance of the approach, it can significantly hinder its extensibility. Many of the textile factories, the clients of Vandewiele, have their own unique set of requirements when creating a production planning. A one-size-fits-all solution would not be suitable. To prevent designing a new approach for each and every customer, a general (without any assumptions) and flexible representation is needed. To the best of the author's knowledge, no such solutions currently exists within the literature for the creation of a weaving machine's production planning.

[1]Vandewiele nv, personal communication, October 25, 2022

The type of problem that needs to be solved will be formalised in this thesis as the ***Weave Planning and Scheduling*** (WPAS) problem. Following the typology of Wäscher, this problem is a slight derivation from the well-known ***Cutting Stock Problem*** (CSP) [76]. The problem will be solved using the mathematical programming technique called ***Constrained Programming*** (CP). This technique is a good fit, as CSP is a ***Constrained Optimisation Problem*** (COP). Additionally, CP supports extensibility by adding variables and constraints to previous model formulations. This flexibility could also support an interactive solving approach. An expert planner solves the optimisation problem in collaboration with a computer algorithm by providing feedback on intermediate solutions.

Contributions To the author's knowledge, this thesis is the first publicly available source in literature to formally define the proposed WPAS optimisation problem and to provide a mathematical model capable of solving it. This formulation and the accompanying basic implementation can serve as the basis for future research towards weave planning or other related industrial applications.

At the core of the WPAS problem lies the ***Single Large Object Placement Problem*** (SLOPP). In this regard, the thesis proposes a new mathematical model called ***CP-Anchor***. It serves as an alternative to the more traditional guillotine-based model formulations. It is a general model, making no assumptions on the solution space, with performance closely approximating an efficient and assumption-rich formulation from literature, dubbed ***CP-Guillotine***. This thesis will demonstrate the model's suitability as a foundation for the complete WPAS problem, with its novel representation allowing for unrestricted extensibility with improved scalability.

Overview Chapter 2 will start with a collection of relevant concepts which serve as the needed background information for this thesis. In chapter 3 the related literature will be described, giving both a historic point of reference and an overview of some of the more recent research serving as inspiration for this thesis. Chapter 4 formally describes the problem statement of creating a weave production planning. Chapter 5 outlines both the newly proposed and the preexisting mathematical formulations for the WPAS's subproblems in detail. In chapter 6 these models get put to the test through a series of experiments on newly proposed benchmark suites. Finally, this thesis gets concluded in chapter 7 with a summary of the gained insights.

Chapter 2

Concepts

This chapter includes some of the background information necessary when reading the thesis text. It can be used as a glossary. When the reader already has great knowledge of the covered concepts, this chapter can safely be omitted.

Mathematical modelling When going through our day, each and every one of us will most likely encounter a variety of different problems needing to be solved. Some of them might be quite mundane and trivial, others quite complex and challenging. When the complexity of these problems starts to increase, one typically begins to rely on certain (structured) techniques in order to tackle them with greater success. One such technique, which throughout history has proven humanity well, is the use of mathematics. Real-world problems can be represented with abstract models, expressed using mathematical concepts. This technique is quite popular in the world of natural sciences, engineering, operations research, and so on. As the real world can be seen as infinitely complex, some assumptions and simplifications will have to be made, resulting in the extraction of the core of the problem at hand. In doing so, one has to be careful not to lose some of the (nontrivial) essential aspects of the problem [63]. Once a model is created, it can be solved and the solution can be translated back from its abstract mathematical representation to its concrete real-world consequences.

Combinatorial Optimisation Problems Many different problems can be represented and solved using the technique of mathematical modelling. One example is the class of ***Combinatorial Optimisation Problems*** (COP). It consists of finding an optimal object in a finite collection of possible objects, with optimality defined with respect to a given objective function [69]. A possible object of the problem is often represented as a specific allocation of concrete values to a collection of free variables. Subsequently, the optimisation comes down to finding the most optimal assignment for those variables. The range of all possible objects, within which the most optimal has to be found, is called the search space. For real-world problems this search space can be rather large, making them practically impossible to (optimally) solve by hand. Luckily, these mathematical models lean themselves quite well to being solved with the help of a computer.

Cutting and Packing ***Cutting and Packing*** (C&P) is an umbrella term that refers to a specific category of related Combinatorial Optimisation Problems. In its most general form, C&P involves the positioning of a (sub)set of relatively smaller items inside a collection of one or more larger objects. The undetermined positions can be thought of as the free decision variables of the problem at hand. The solving of the problem then translates to the assignment of concrete values to those variables and thus the positioning of the smaller items inside the larger objects. Some examples include the cutting of metal shapes from a larger sheet [72, 75], the positioning of cardboard boxes on a pallet [29] and the scheduling of tasks across the cores of a multicore CPU [20]. According to a recent survey [38], it is quite an active and diverse domain of research with many open problems left to be solved. In practice, C&P often gets combined with other optimisation-related problem formulations. Combining it with routing problems can for example be used to minimise transportation costs whilst subjected to certain loading constraints [51]. Combining it with time scheduling can be used for the assignment of ships to locks, subjected to a lock schedule [74].

Constraint Satisfaction Problem In C&P the placement of the smaller items, and thus the assignment of the variables, gets bound by a multitude of constraints. Some more general and rudimentary, like the non-overlap of the packed items, others quite complex and more application specific, like a balanced weight distribution of the entire packing [55]. The problem thus belongs to the more general category of ***Constrained Satisfaction Problems*** (CSP). These types of problems can more formally be defined using a triplet $< X, D, C >$: [66]

- X: the set of variables $\{X_1, \ldots, X_n\}$
- D: the domains of all variables $\{D_1, \ldots, D_n\}$
- C: the set of constraints formulated over X

An assignment of values to all free variables $(X_1 = v_1 \in D_1, \ldots, X_n = v_n \in D_n)$ can be called a configuration or a state of the modelled problem. A configuration becomes a solution to that same problem when it satisfies all formulated constraints C. A solution is said to be *consistent* with the modelled problem formulation. Depending on how restrictive the constraints are formulated, there can exist multiple alternative solutions. A complete assignment occurs when every variable is assigned a value, whilst a partial assignment can still have some free variables [66].

Global Constraint The simplest type of constraint is a ***unary constraint***. It defines a restriction on the value of a single variable. For example that a value should be even. A ***binary constraint*** relates the values of two variables. This can for example represent that one variable should always be larger than another. Even higher order constraints can be devised. What all these constraints have in common is that the number of input variables is fixed. A constraint with an arbitrary number of variables is called a ***global constraint***. A very popular example is the *Alldiff* constraint, demanding that all provided variables (no restriction on the amount) should have a unique value. [66]

Constrained Optimisation Problem Apart from constraints, a C&P typically entails one or more objectives in order to distinguish between the different solutions. This further categorises the problem as a ***Constrained Optimisation Problem*** (COP), where the solution has to be optimised with respect to a given objective function. More formally, a COP can be written as follows:

$$\begin{aligned} \min \quad & f(x) \\ \text{s.t.} \quad & g_i(x) = c_i \qquad for \quad i = 1, ..., n \\ & h_j(x) \geq d_j \qquad for \quad j = 1, ..., m \end{aligned} \tag{2.1}$$

In this equation, $f(x)$ represents the objective to minimise, $g_i(x) = c_i$ represents a collection of n equality constraint, and $h_j(x) \geq d_j$ a collection of m inequality constraint. The above formulated constraints are so-called hard constraints; for a configuration to be a solution, these constraints must be satisfied. Apart from hard constraints, there can also exist soft constraints. A solution does not have to satisfy these but any degree of violation will get punished inside the objective function. As both minimisation and maximisation are optimisations, the objective can equivalently be formulated as a function $-f(x)$ that has to be maximised.

Operations Research One of the driving factors behind the popularity of C&P's is its numerous real-world applications, often related to the domain of Operations Research (OR). According to the Britannica dictionary [5], OR is the

> " *application of scientific methods to the management and administration of organised military, governmental, commercial, and industrial processes*"

Some examples include the cutting of glass out of larger sheets [62], the loading of boxes onto pallets [29], the placement of items inside an air cargo volume [58], and so on. As can be seen from these few examples, many variants of the problem exist. Differentiating themselves in terms of the number of dimensions, type of constraints, type of objectives and other aspects.

Cutting Stock Problem Following the typology of Wäscher [76], this paper will go deeper into the ***Cutting Stock Problem*** (CSP) and will make small derivations from it. This problem received its name from the conceptual process of cutting a set of smaller items (demands) from a set of larger object (stocks). Both the small items and the larger objects have fixed dimensions and the small items have an accompanying demand (multiple copies of the same item will have to be packed). A typical objective function consists in cutting a maximal subset of the demand from the collection of larger objects whilst minimising the leftover waste, sometimes called the trim loss. A solution to a problem instance can then be called a (cutting) pattern. This thesis will look at the two-dimensional case with both rectangular items and objects and with the orthogonal placement of these items (each side of an item must be parallel with one of the encompassing objects' sides).

Imperative vs Declarative One aspect with which different solving techniques can be distinguished from one another is whether they belong to the paradigm of imperative programming or rather to that of declarative programming. ***Imperative programming*** is the more classical approach, where programs give precise, step-by-step instructions on how to solve a particular problem. ***Declarative programming*** on the other hand limits itself to the declaration of a high level representation of the problem, without specifying how to solve it. A general purpose program, called a solver, can then take this representation as input and compute a solution. In the ideal case this would allow someone to solely focus on correctly formulating all aspects of a problem, without having to know or implement the techniques needed to solve that problem.

Symmetry breaking One of the key problems in constraint programming based implementations is the concept of symmetry. It refers to the symmetry inside the search space, causing the solver to revisit multiple times equivalent solutions, wasting precious computation time. This problem is often inherent to the chosen representation. The goal of symmetry breaking is to make certain changes so as to never explore two configurations that are symmetric to each other. [65] A toy problem where this is the case is the "*Factorisation game*". In this game, the player is given a number. The challenge is to figure out which two other numbers, when multiplied together, result in the given number. The most straightforward way to model this game is to introduce two variables a and b, representing the hidden numbers, and a single constraint $a * b = c$ with c the given number. This representation unfortunately suffers from symmetry, where the values of a and b could be interchanged due to the commutative property of multiplication. The result would still be the same. A common technique to solve this problem is to add an additional symmetry-breaking constraint to the model, for example $a \leq b$. Another approach is to reformulate the problem to reduce or completely remove the symmetry. A new representation can consist of variable a and a new variable t signifying how much larger b is compared to a. This representation makes the inherent assumption that b is always larger than a, which can be made without any loss in generality. The singular constraint now becomes $a + b = a + (a + t) = 2 * a + t = c$. This new representation no longer suffers from symmetry.

Large Neighbourhood Search The idea of ***Large Neighbourhood Search*** (LNS) is to iteratively transform the current solution into a different (improved) solution by searching in its neighbourhood. This typically includes two subsequent operations: a destroy operation and a repair operation. Certain aspects of the current solution get destroyed, resulting in only a subset of the solution's variables being turned back into free decision variables, whilst the rest gets fixed. By resolving the partially destroyed solution, a new solution gets found in the neighbourhood of the original one. Both solutions are close to one another due to their shared subset of fixed variables. When a large section of the solution gets destroyed, the neighbourhood can become very large and exponential in size, resulting in the case of ***Very Large-scale Neighbourhood Search*** (VLNS).

Delayed Column Generation The idea behind ***(Delayed) Column Generation*** ((D)CG) is to not try to solve an entire model at once. Often times problems are too large to consider all variables at once. With ***Column Generation***, the algorithm starts with solving over a limited subset of the variables, iteratively adding specific variables to the problem formulation as needed. The hope is to find a satisfiable solution before having to ever define all variables of the total problem. The original problem gets split up into two: the *master problem* and the *subproblem*. Using the *subproblem*, improving variables get found and added to the *master problem*. Once the *master problem* produces a satisfactory solution, the iterations stop.

Chapter 3

Related Literature

This chapter will highlight some literature regarding what is already known and what still remains an open problem with respect to the Weave Planning and Scheduling problem. Some historical research will be discussed and the motivation behind certain upcoming model design decisions will be highlighted.

3.1 Historically important sources for C&P

The origin of the research domain related to C&P is often linked to the publication by Brooks et al. in the year 1940 [15]. The paper covers the combinatorial problem of dividing a rectangle into a number of non-overlapping, uniquely shaped items. The problem formulation is still a simplified version, as the items were assumed to be square and the number of times an item could be packed was limited to one. The authors utilised a network-based representation to model the problem. They developed a technique for solving this network by borrowing from the theory of electrical networks, such as Kirchhoff's laws.

In 1958, Kantorovich published a paper highlighting the practical use cases of C&P problems [44]. He was doing research towards improvements in organisational planning to increase efficiency under limited resources such as material, labour, and equipment. Doing so, he identified that many problems related to the scientific organisation of production are quite similar. He thus grouped them together under the category of "*extremal problems*". The goal of these problems is to find the most advantageous solution from a huge collection of possible solutions. The existing techniques of mathematical analysis were not sufficient, as it would require solving thousands or even millions of systems of equations. Kantorovich formulated a simple and general technique for solving these problems under practical conditions via the method of "*resolving multipliers with successive approximations*". This technique is also known as "*Kantorovich's method*", the first algorithm for solving linear programs.

Gilmore & Gomory (1961) published the first source in the literature regarding cutting stock problems [25]. It was limited to the one-dimensional case and was solved using a linear programming approach. Later, in 1965, the same authors extended their formulation bringing C&P to a two-dimensional setting [26].

3.2 Problem complexity

Ivkovi & Lloyd (1997) published the first source which demonstrated that bin packing, a variant of C&P, is an NP-hard problem [39]. Later, Jansen & Prdel (2014) expanded this insight to the two-dimensional case [40]. Fekete et al. (2007) discussed some techniques which could be used in order to deal with this complexity [24]. As a first technique, a problem can be converted into a different problem with better complexity characteristics. This is not always applicable as in practical applications one often does need to solve the original problem. A second technique is looking for special properties of the problem at hand and utilising expert knowledge to relax certain constraints. As a last technique, one can look for a good solution instead of the most optimal one. This results in the research domain of approximating algorithms.

3.3 Approximation techniques

Due to the complexity associated with solving C&P problems, many approximating techniques have appeared in the literature. A popular technique is the use of ***normal patterns***. By making assumptions about the solution's cutting pattern, the search space can be reduced in size. A well-known issue with C&P problems is that a large number of solutions are, with regards to the basic problem formulation, equivalent to each other. Normal patterns can help limit the number of these equivalent solutions by selecting only one and making the rest no longer valid. The first sources to utilise normal patterns were published by Herz in 1972 and by Christofides and Whitlock in 1977 [35, 18]. They independently observed that any solution (feasible packing) can be converted into an equivalent solution by shifting all items as much as possible to the left and to the bottom. Normal coordinates were introduced, reducing the set of possible positions of an item based on this shift assumption. The former source proved that for the basic problem formulation, this technique preserves optimality. Côté & Iori (2018) proposed a new pattern assumption based on the ***meet-in-the-middle*** principle and performed extensive computational experiments to demonstrate the efficiency of this technique [21]. Terno et al. (1987) proposed an improvement on the set of normal patterns, called ***Reduced Raster Points***, further reducing the set of possible points by eliminating redundant ones [73]. Another often-used category of approximation techniques, according to Pardalos et al. (2002), is the ***greedy sequential algorithms*** [59]. These include next-fit, first-fit, and best-fit. Lodi & Monaci (2003) proposed two new integer linear programming models for the two-dimensional knapsack problem [60]. Whilst these models fall under the umbrella of exact techniques, they heavily lean into the assumption of ***two-stage guillotine*** cuts, where each item must be reached with a maximum of two edge-to-edge cuts.

Whilst each of these sources demonstrated improvements in algorithm performance by utilising approximation techniques, Junqueira et al. (2012) criticise these types of results [42]. The authors highlight the lack of studies that consider more realistic problems with realistic constraints, requiring solving techniques that are not hyperefficient implementations of just a single toy problem. Whilst many approximating techniques exist in the literature, most are not flexible and can't handle the additional constraints of realistic problems. Most limit themselves to solely the formulation of non-overlap. The authors proposed a new model for the container loading problem, which is extensible and can cope with additional constraints. Whilst results looked promising, the proposed 0-1 integer linear programming model suffers from its lacklustre scalability. It is a completely ground-out set-of-points representation, where each possible placement of an item gets its own Boolean variable, resulting in many variables and subsequently a large number of constraints. What this representation does provide is many preprocessing opportunities as each possible placement gets a unique variable, giving inspiration to this thesis' newly proposed anchor model.

Similarly, Hermenier et al. (2011) argue that traditional techniques do not consider flexibility and generality enough in order to support additional user requirements [33]. The authors propose an extensible model for bin packing for the application of distributing applications over data centre servers. The model is formulated using constraint programming techniques, which due to its declarative nature and use of constraint propagation has support for composability. Many additional "*side placement constraints*" can be added to the base model formulation with ease. As a demonstration of this flexibility, they reported being able to add a new complex requirement to their model in just three hours without having to modify any of the previous formulations. They even propose adding a search algorithm on top of the CP model to handle even more complex problems. They hint at the integration of user feedback by repairing a partially fixed solution in an iterative fashion.

3.4 High multiplicity

This thesis will cover the ***High Multiplicity*** (HM) variant of a C&P problem. In a standard formulation of C&P, the length of the input is polynomially dependent on the problem scale. Hochbaum & Shami (1991) proposed an alternative encoding, based on the multiplicity of items, as a more efficient technique to input data [36]. Instead of listing all items to pack as individual instances, items get grouped by type. The multiplicity of each type serves as input together with a description of each type. Kan R. (1975) designated this encoding style as "*id-encoding*" [43]. Clifford & Posner (2001) analysed HM in the context of machine scheduling and argue that "*id-encoding*" is especially desirable when there are many jobs but few different types of jobs [19]. Additionally, their paper determines that switching to this new encoding style, even though it can significantly reduce the size of the input, does not result in a change in the complexity class. This encoding can also be used for C&P in the case of many items but few item types. Hansen et al. (2022) demonstrate this by applying the HM encoding to a strip packing problem [12].

3.5 Due date

In order to support the full WPAS problem formulation, the notion of due dates will have to be added to the solving algorithm. Hendry et al. (1996) investigated new methods for creating production plans [32]. The optimisation problem was categorised as a combination of scheduling and cutting stock, having to minimise the cost whilst meeting demand. Their solution does not however solve the complete problem, as manually designed cutting patterns had to be provided as input. Sezer & Muter (2017) highlight some of the difficulties of combining C&P with scheduling [70]. It becomes a multi-objective optimisation problem, requiring a technique to scalarise and balance the different sub-objectives. Additionally, both sub-problems employ a different perspective. Packing is formulated in terms of items and patterns, whilst scheduling in terms of time and collections of patterns to be produced in sequence. Classical C&P problems have no information on planning. It is still important to combine the two as Reinertsen & Vossen (2010) argue that in many industrial applications, delivering on time might be much more important than minimising waste [64]. As a solution, the authors discretise the planning horizon into "*production periods*" where time intervals get expressed in terms of multiples of the machine capacity. Arbib & Marinelli (2014) improve upon this idea by linking the packing model to the scheduling model using inventory levels [7]. Earlier representations used the notion of deadline time and completion time of each order. If the completion time is the largest of the two, tardiness is present. The old representation resulted in this tardiness being overcalculated. Using inventory levels, where tardiness is represented as negative inventory, this is no longer an issue. The new representation allows for earlier production to satisfy later demand, giving the ability to plan ahead. Gramani & Frana (2006) highlight the importance of producing in advance through an economic perspective [27]. Anticipating later demands increases production costs but often results in an overall cost decrease due to better compositions in the cutting patterns and a reduction of setup costs. Haessler (1975) is the first source to associate additional setup costs with pattern changes [31].

3.6 Large Neighbourhood Search

According to Middleton et al. (2022), Large Neighbourhood Search (LNS) is a very popular technique for solving combinatorial optimisation problems [54]. They unfortunately have a dentency to get stuck in local optima. To escape from these optima, generally two solutions are available; using heuristic methods or increasing the size of the neighbourhood. Blum et al. (2008) gives an overview of many heuristic approaches which could be utilised, but similar to approximation techniques they lack in flexibility and extensibility [13].

A problem associated with the second approach is that the design of an appropriate neighbourhood often requires expert knowledge, which is problem specific and not generalisable. Some sources, like Hojabri et al. (2018) and Ancion & Dardenne (2016), have found success in exploiting CP capabilities in an LNS framework to solve this neighbourhood problem [37, 6]. CP appears to be a natural fit for LNS, using a model as part of the reconstruction operator. The problem gets tackled in reverse; instead of destroying part of the solution as to create a neighbourhood, the other variables of the solution get fixed.

3.7 Similar problems

Within the literature, only a limited number of papers can be found tackling a problem similar to this thesis. Some have many similarities, but are situated in a completely different industrial sector. Sousa et al. (2015) developed a decision support tool for cutting patterns out of metal sheets for the production of animal cages [72]. The developed method utilises a two-phase approach, one where the cutting patterns are defined and a second where the patterns get chosen towards a total production objective using a linear programming model.

The number of sources specifically related to the planning of textile production are even more limited. Silva et al. (2015) propose a MIP model to create production plannings for a Portuguese home textile manufacturer [71]. The model makes a guillotine assumption and uses delayed column generation to solve the linear relaxation of the Restricted Master Problem (RMP), which limits the search space to a reasonable subset of the possible cutting patterns. A faster variant of the classical delayed column generation of Gilmore & Gomory, Oliveira & Ferreira (1994) [57], is utilised.

Salem et al. (2023) use an approximating approach in the context of the home textile industry [30]. It proposes two models. First, an extension upon the guillotine formulation of Lodi & Monaci (2003) [60] with additional inequalities as to break some of the symmetries within the formulation. Second, an adaptation of the symmetry-less formulation of Hadj Salem & Kieffer (2020) [67] to the two-dimensional setting.

Aside from limiting itself to guillotine cuts, it also makes the assumption that the demand for each item to pack is small, which does not suit the HM case of this thesis. Whilst the first model will serve as a performance reference for the newly proposed model of this thesis, the paper also serves as inspiration for the choice of evaluation technique. It makes use of the performance profiles from Dolan & Moré (2002) [22], a robust technique for comparing the metrics of different solvers and/or models.

Chapter 4

Problem statement

This thesis consists of research towards automated solutions for the ***Weave Planning and Scheduling*** (WPAS) problem. It is a real-world optimisation problem originating from the textile industry and concerns both the production planning and scheduling for industrial-sized weaving machines:

> "*Which textiles should be produced when? How should the they be laid out on the machine? How should the machine itself be configured? ...*"

By utilising computer-assisted techniques, the solutions to the WPAS problem can be made more consistent, of higher quality and more flexible with regard to unexpected schedule changes. In this chapter, the problem will get decomposed and reduced to its core components. Using standardised topology from the literature, the different subproblems get categorised. In the end, the problem will be formally defined from the bottom up, adding extensions and complexity in the process.

4.1 Formal specification

By creating a formal specification of an optimisation problem, one can start using analytical methods to improve decision-making. This often involves the creation of a mathematical model as an abstract representation of the problem at hand. Once such a model is created, different methods can be used to search for optimal or near-optimal solutions. This technique is most often associated with Operations Research (OR), a discipline in pursuit of improved decision-making and efficiency with strong ties to computer science and analytics. OR provides a systematic process to break down a problem into its core components and to solve it step by step: [46]

1. Identify the problem.
2. Construct an abstract model that represents the core of the problem.
3. Use the model to design solutions.
4. Test the proposed solutions on the model and analyse its performance.

5. Implement one as a solution to the actual problem.

Once a formal specification of the problem has been defined, a logical next step would be the creation of algorithmic automations to reduce the burden of manually creating new production plannings. Such algorithms can improve the reliability, consistency, and quality of the solution. More alternatives can be considered and can be presented to the expert planner. Changes can be made quickly and their impact can be analytically verified. These algorithms work as a support tool for the expert planner, can drive down production costs, and can create a competitive advantage.

4.2 Weaving process

As this thesis regards the optimisation problem of designing a production schedule and planning for a weaving machine, also called a loom, a better understanding its workings is required. Industrial sized looms are capable of producing a plethora of different textiles. These include products like carpets, rugs, towels, and so on. On an abstract level, the production process can be divided into two phases; the *weaving phase* and the *cutting phase*. Most knowledge presented in this chapter has been based upon a private conversation with Vandewiele[1] and upon a tour of their production facility.

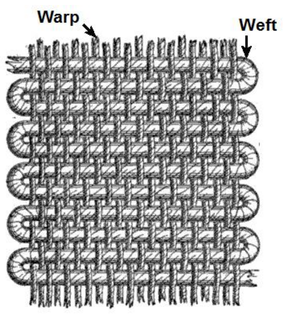
4.2.1 Weaving phase

During the weaving phase, different coloured yarns with possibly different material characteristics (as for example to create different surface textures) get used to weave one large rectangular textile piece. Weaving is the process of interlacing two sets of yarns, usually orientated orthogonal to each other, to create a larger piece of fabric [41]. This process can be done mechanically using a loom. Yarns orientated along the width of the machine are called wefts and those orthogonal along the length of the produced fabric are called warps (*figure 4.1*) [46]. The warps are held stationary under tension, whilst the wefts get interwoven. The produced rectangular piece often has the same width as the entire loom. It is possible to leave some warp yarns out but it should remain a continuous piece. The textile's length continuously increases during the weaving process.

The main material of a woven fabric is the utilised **yarns**. To manage the "piles of yarns" that are needed to create a fabric, they get wound on **bobbins**. An industrial-sized loom can quickly need hundreds or even thousands of these bobbins, containing different colours and materials for different sections across the width of the loom. To manage all these bobbins they get installed on a **creel**, a large mounting frame positioned next to a loom. The installed bobbins can rotate freely so that yarn can be pulled from them during the weaving process. The configuration of such a creel, and thus which bobbins get placed where, determines which yarns (which colours) are available across the width of the machine.

[1]Vandewiele nv, personal communication, October 25, 2022

Figure 4.1: **Warp and weft**



Note. Warp and weft in plain weaving. from ”*The History and Principles of Weaving By Hand and By Power (Classic Reprint)*” by A. Barlow, 2017. CC BY-SA 3.0

A loom has both a width-wise and a length-wise weaving resolution, determining the detail with which the patterns of a fabric can be produced. Length-wise the unit of resolution gets called a **pick** and across the width it is called a **dent**. For each dent, a loom can use a combination of multiple yarns to create a desired pattern. The number of these yarns and thus the number of colours to create patterns with, gets limited by both the capabilities of the loom and the size of the creel. The number of bobbin positions that a creel provides for each dent position directly determines the number of different colours to produce patterns with. These colours do not have to be the same for every dent and can thus accommodate for varying requirements across the width of the textile. They do however have to accommodate for all required colours at that dent position along the entire length of the produced piece of fabric. It would be too impractical and labour-intensive to change out colours mid-production.

4.2.2 Cutting phase

When producing on an industrial scale, a loom can get quite large. Quickly reaching a width of 4 metres and supporting a total of 8 colours per dent. This increase in size allows for the production of multiple textiles at once across its width (besides the sequential production across its length). The second stage consequently consists of a cutting phase. Production happens in sections of limited length between which the produced large rectangular fabric, consisting of a patchwork of many textiles, gets cut off for further processing. The different textiles which have been woven together in this larger segment have to be cut out to their individual sizes to be freed from each other. To accommodate for this step, a small spacing has to be foreseen between the different textiles. A horizontal spacing is often called a **free zone**, whilst a vertical spacing gets called a **cut bar**. Due to how the process of weaving works, even these empty spacings have to be woven. This results in the waste of yarn and thus its size has to be kept to a minimum. With unfortunate combinations of textile shapes to be produced, this empty space can sometimes be larger than what is strictly needed for the cutting process. This is called a **rest**

zone. These zones result in an additional increase in waste, as they often have to be woven using a wasteful **fill pattern** to prevent the skewing of the total fabric. This skewing is the result of the picks of empty space being smaller than those of a patterned section of fabric. This highlights the importance of a good production planning with a good combination of textile shapes to be produced together.

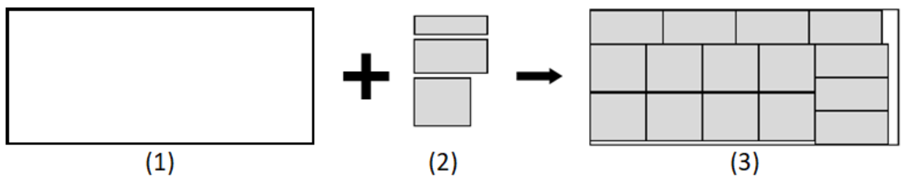
4.3 Production planning

Now that an understanding of weaving on an industrial scale has been established, a closer look can be taken at the encompassing production scheduling and planning problem, from now on simply referred to as the planning problem. As mentioned before, a good abstract model of the problem combined with a suitable solving algorithm can serve as a valuable tool to support the expert planners. For a weaving machine, production planning has both a *spatial* and a *temporal* aspect.

4.3.1 Spatial

Spatial refers to the positioning of different textiles within a single production segment (the larger produced textile, consisting of a patchwork of items). Placing textiles next to each other along the width of the loom will result in their simultaneous production. Placing them after each other along the length of the segment results in their sequential production. Any space within a segment that is not used, will result in wasted material. Planning comes down to the optimal placement within the segment of the textiles to be produced. This includes their selection and the choice of their position and rotation (*figure 4.2*).

Figure 4.2: **Spatial planning**

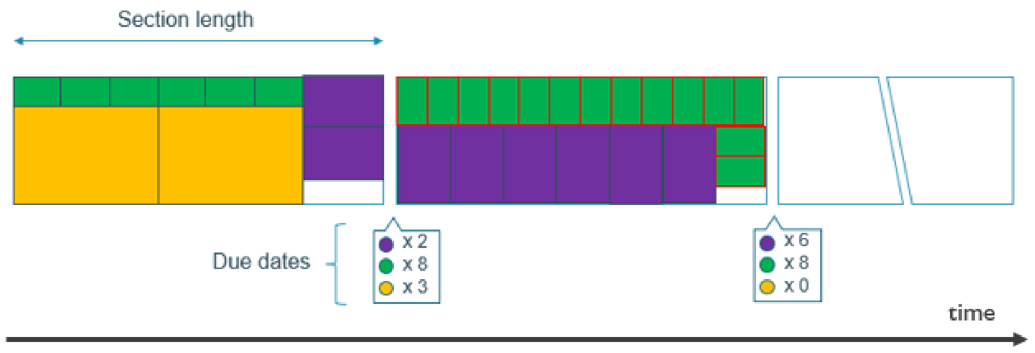


Selection and positioning of *textile items* (2) inside a larger *production segment* (1) to create a *spatial packing plan* (3).

4.3.2 Temporal

As one production segment of limited length is often not large enough to produce all demand, multiple segments will have to be designed and planned (*figure 4.3*). The temporal planning of these segments is needed to guarantee a timely delivery of the textiles. By looking at the demand for each type of textile and their respective due date, a production schedule can be created. It determines what to produce and when. Both producing too little, resulting in the missing of a deadline, and too much, resulting in unnecessary storage costs, are to be avoided. In practise this is often more nuanced, as producing too much can serve as a preparation for a later deadline. As these production plannings are often created in a rolling horizon fashion, future orders can still be added later on. Producing as early as possible, with additional storage costs, keeps production capacity available for later deadlines. Additionally, by mixing different orders more optimal combinations of textiles can be found. This improves the spatial planning aspect, reducing material waste.

FIGURE 4.3: **Temporal planning**



Creating a temporal planning based on demand and due dates.

4.3.3 Interdependence

There exists an interaction between the aforementioned spatial and temporal aspects of the weave planning problem. Each textile that has to be produced can have different requirements in terms of the needed yarn. These are mostly related to the availability of certain colours at the different dent positions across the width of a textile. As there is a limit on the number of creel positions for each dent, careful considerations have to be made when producing multiple textiles sequentially within one segment and across multiple segments.The reconfiguration of a creel requires a lot of manual labour and must be avoided as much as possible since it halts production. Planning the configuration of a creel thus spans across multiple segments (temporal aspect) and is dependent upon all their spatial configurations.

4.3.4 Objective

The combination of the previous two aspects results in an unclear definition of the ”optimal” production planning. Different objectives have to be simultaneously optimised, with some of these contradicting each other. The main objectives are:

- minimise the total waste
- minimise deviation from the demand (both underproduction and overproduction)

Incorrectly combining both objectives and ignoring their influence on each other can lead to suboptimal results. A solution with minimal waste is one where nothing gets produced, whilst satisfying the exact demand can result in sections being left rather empty. The objectives have to be balanced against each other depending on their relative importance. An additional complexity lies in the interaction with the planning expert. The balance between objectives is often not static. It can differ between planners (depending on their preferences) and between planning sessions of the same planner. It can even go so far as changing during an interactive solving session. The objective is rather situationally dependent.

4.4 Problem decomposition

Now that both the process of weaving and the planning of said weaving have been explained, the optimisation problem can get decomposed and generalised to reveal its underlying sub-problems.

4.4.1 Domain specific extensions

At the surface of the optimisation problem, there are some domain-specific aspects. These convert a more general planning problem into a WPAS problem. The textiles to be produced all require a collection of different yarns at different positions in order to create their respective pattern designs. The notion of these colours, made available at different dent positions by the configuration of the creel, is specific to weave planning. When creating a production plan, one or more creel configurations have to be designed in order to provide coloured yarns to a sequence of production segments whilst minimising the number of these configurations. Each change of configuration brings the loom to a halt, wasting precious production time.

4.4.2 Due dates

By removing these weaving-specific aspects, a more general optimisation problem gets revealed; a production planning and scheduling problem with due dates. Given a collection of different production segments (often called *patterns*) with differently positioned (textile) items, an optimal schedule for the sequential production of these segments has to be found. The input for this schedule is comprised of the list of available patterns and of the demands for all the different items across all deadlines. The optimisation problem is to find the optimal sequential combination of these patterns to meet the demand whilst satisfying certain constraints. Within the literature, this problem often gets called a machine scheduling problem with capacity constraints. A loom can only produce one segment at a time with a makespan dependent on the length of that segment. The limited size of a segment implies a capacity limitation of the machine.

4.4.3 SLOPP

As a final decomposition, even the notion of time can be removed. This reduces the optimisation problem to the repeated design of a single production segment or a single pattern. This categorises as a ***Single Large Object Placement Problem*** (SLOPP). Different rectangular items have to be positioned inside a larger rectangular object. These items can not overlap each other or be outside of the larger object. Copies of the different items must be positioned to optimally use the available space. SLOPP can be seen as the core of the entire WPAS problem, with all the previous layers seen as extensions build on top. It is of high importance to ensure that the solving of this sub-problem is implemented efficiently and that it has support for the previous extensions.

4.5 Formal definition

As a continuation of the problem decomposition, a formal definition of each sub-problem can be formulated. The combination of all the following sections results in a specification of the WPAS problem and its first introduction to the literature.

4.5.1 Cutting and Packing

Cutting and Packing (C&P) is a term used within the literature to refer to a rather large family of related optimisation problems. Some examples include the well-known ***Bin Packing Problem***, the ***Cutting Stock Problem***, the ***Knapsack problem***, and so on., all of which share a common core optimisation problem. Throughout the years, many of these problems received quite a lot of attention due to their link with OR and their subsequent many practical applications.

As this research domain is getting quite large, different topologies have been proposed in order to organise and categorise existing and future literature. These topologies base themselves upon the unique characteristics of the different problems falling under the umbrella of C&P. In this thesis, the improved topology proposed by Wäscher et al. [76] will be used.

At the core of every C&P problem, there is the notion of two sets of elements. A set of large objects (often called the *input* or the *supply*) and a set of smaller items (often called the *output* or the *demand*). Within OR, these elements often represent respectively the available resources, like large sheets of metal or empty cardboard boxes, and the required end-product, like shapes to be cut and items to be packaged. The problem to be solved comes down to the selection of a subset of these smaller items, the grouping of them into a collection of subsets, and the assignment of these subsets to one or more of the larger objects. Additional complexity gets associated with these assignments as they are subjected to a collection of geometric conditions. These conditions define how the smaller items must be laid out on their assigned larger object. In the most general case, the imposed geometric conditions come down to the following two requirements:

1. Every small item, if assigned, must lie entirely within its larger object.
2. Small items may not overlap when being laid out

The satisfaction of these two conditions is a requirement for an assignment to produce a valid solution. Within many practical applications, this notion of an item being ”laid out” on an object often refers to a spatial aspect. Smaller items must then be placed inside the boundaries of a larger object. Other interpretations also exist, like the scheduling of different tasks within a planning horizon.

As C&P is an optimisation problem, an objective must be specified. According to the topology of Wäscher, five sub-problems can be distinguished, all of which have to be solved simultaneously for an optimal solution to be reached [76]. These sub-problems include:

1. selecting the larger objects
2. selecting the smaller objects
3. grouping the smaller items into a collection of subsets
4. assigning subsets to larger objects
5. laying out the smaller items onto their assigned objects

Many different optimisation problems fall under the category of C&P. They differ from each other in terms of the formulation of their items, their larger objects, their additional properties and conditions, and so on.

4.5.2 Problem categorisation

In the typology of Wäscher, five criteria get used to distinguish the different C&P problem types [76].

I) The first aspect is the **dimensionality** of the optimisation problem. Traditionally up to three dimensions get considered but problems with even higher dimensions do exist [47]. For the problem of placing an assortment of different textiles inside the boundaries of a larger rectangular production segment, two dimensions suffice.

II) The second aspect concerns the kind of **assignment** when positioning smaller items inside larger objects. There exist two situations; output maximisation and input minimisation. As previously mentioned, ”*output*” is sometimes used to refer to the collection of smaller items, and ”*input*” refers to the larger objects. With ***output maximisation*** the set of larger objects is not sufficient to contain all demanded smaller items. A subset of the items must then be chosen with a combined maximal value or revenue. The value of an individual item is often associated with its surface area but other formulations are possible. With ***input minimisation*** the set of larger objects is sufficient for all smaller items, removing the need to select a subset of them. The problem now shifts to the selection of a subset of the larger objects with minimal total value or total cost, whilst still providing space for all items. Some problems contain a selection process for both the items and the objects, resulting in a so-called ***waste minimisation*** problem. In the context of finding new patterns for weave planning, a single production segment can never pack the entire item demand of a realistic problem setting. A subset of textile items must be chosen to optimally fill that segment, minimising wasted material. The single segment problem can thus be categorised as an output maximisation problem. For the complete production planning, multiple segments can be used and their number must be kept to a minimum to limit waste and to preserve production time. It then becomes a waste minimisation problem.

III) & IV) The third and fourth aspects concern the **assortment** of respectively the small items and the large objects. The small items can either all be identical, be weakly heterogeneous or be strongly heterogeneous. With weave planning, the production demand can be grouped into a limited set of unique textile types, with each type having a larger demand. This falls under the category of a weakly heterogeneous assortment of items. For the large object, the options are one or several large objects. Whilst the entire production will consist of multiple patterns, the creation of a new pattern will be limited to a single large object at a time.

V) The fifth and final criteria typically used to distinguish C&P problems concerns the **shape** of the smaller items. This paper will be limited to rectangular textile shapes.

With these five criteria, the core of the weave planning optimisation problem can be characterised as a ***Rectangular Two-Dimensional Single Large Object Placement Problem*** or R_2D_SLOPP.

4.5.3 Rectangular Two-Dimensional Single Large Object Placement Problem

At the centre of the WPAS problem lies R_2D_SLOPP as per the taxonomy of Wäscher. It is a C&P problem with a single large object and a collection of smaller items types $t \in T$ to be selected and packed. It is an output maximisation problem where a number of instances of some or all of the item types have to be selected as to maximise the total value of the solution. The large object has a length L and a width W whilst the smaller item types each have their own dimensions l_t and w_t and an associated value v_t $(t = 1, ..., n)$. Both the object and the items are rectangular. When positioned their edges must be parallel, resulting in an orthogonal pattern. Rotation over 90° is allowed for the items. In the case of WPAS and many other practical implementations of R_2D_SLOPP, the value of an item is chosen to be its surface area, i.e., $v_t = l_t * w_t$.

A) Input

An encoding for the input has to be chosen. As this paper concerns the ***High Multiplicity*** (HM) variant of SLOPP, an HM encoding scheme gets used to represent the items to pack. Traditionally with many C&P problems, each item gets defined individually; its width and its length. In the case of HM, many of these items are identical and can be interchanged without affecting the solution. There is only a limited number of unique items, but each has a large demand. As to encode this type of input, the notion of item types and item type instances gets used.

The problem to solve consists of a set T of unique item types t. Each type gets defined by its width w_t and its length l_t. Normally for C&P problems, a demand $\tilde{d_t}$ would be defined to represent how many instances of item type t should optimally be packed inside the object. As this paper concerns the HM case, this will not be defined as an input (of the SLOPP part of WPAS) but rather be derived from the problem itself. The demand is assumed to be larger than what can maximally fit into a single object and can thus be initialised to the Maximal Packing Multiplicity MPM_t for each item type t. With the additional input of the object's width W and length L, the upper bound that is the MPM_t can be set to $\left\lfloor \frac{W*L}{w_t*l_t} \right\rfloor$. This will, in turn, determine the number of instances N_t of each item type t that have to be prepared.

$$d_t >> N_t = MPM_t = \left\lfloor \frac{W*L}{w_t*l_t} \right\rfloor \tag{4.1}$$

B) Output

With z_t representing the number of instances of item type t packaged inside the larger object, the total value V of a pattern can be formulated as:

$$V = \sum_{t=1}^{n} v_t * z_t = \sum_{t=1}^{n} (l_t * w_t) * z_t \tag{4.2}$$

The optimisation problem can subsequently be formulated as:

$$\begin{aligned} &\max V \\ &\text{s.t. } (z_1, ..., z_n) \text{ is a feasible cutting pattern} \end{aligned} \tag{4.3}$$

A cutting pattern can thus be abstractly represented as a vector $(z_1, ..., z_n)$, formulating all the items it contains. What it means to be a feasible cutting pattern depends on the application at hand. In the most basic form, only non-overlap between the items and positioning within the object must be satisfied to ensure feasibility.

The vector $(z_1, ..., z_n)$ is not enough to completely represent a solution to an R_2D_SLOPP instance. Other information which must be encoded includes the positioning of each item instance and their chosen rotation. No concrete formulation can be given just yet, as multiple representations exist based on different design decisions. The chosen representation plays an important role for the algorithms, which have to solve the optimisation problem. A good design can significantly reduce the complexity of the problem, but it often introduces assumptions on the solution space which limits the representation's expressibility [34]. Thus the representation has to be adapted to the application at hand and must carefully balance the chosen assumptions. In *section 5.1 Approach*, multiple representations will be proposed and compared against each other.

4.5.4 Due Dates

The first extension upon R_2D_SLOPP is the addition of time. Following Reinertsen & Vossen (2010), the planning horizon can be discretised into "*production periods*" matching the capacity of a single machine [64]. Instead of designing the packing of a singular object, an entire production schedule should be created. Planning thus occurs at the resolution of produced objects, scheduled one after the other. This converts SLOPP to a ***Multiple Large Object Placement Problem*** (MLOPP). Discrete timepoints $tp \in [0, tp_{max}] = TP$ can be introduced. At each timepoint only a single object can be produced on a weaving machine. The objects must be designed and scheduled in such a way as to optimally fulfil the ***Master Production Schedule*** (MPS). The MPS consists of the formulation of multiple deadlines $d \in D$, each with its own timepoint and its own demands for certain items to be produced. The "optimality" considers both the wasted material in each object and the compliance towards the demand.

A) Input

Just like the SLOPP models, the input for this optimisation problem consists of the specification of the object and the different items to pack inside. Additionally, a set of deadlines is provided. Each deadline $d \in D$ consists of a due date $dd_d \in TP$ for when in time the production should be ready, together with the demands $\tilde{d}_{t,d}$ for each item type t.

B) Output

The output of this optimisation problem will be the production schedule; what should be produced when. More formally, $z_{t,tp}$ represents how much of item type t should be produced at timepoint tp, which gets determined by the planned object at tp. The value V_{tp} of the pattern at timepoint tp can still be formulated as before:

$$V_{tp} = \sum_{t=1}^{n} v_t * z_{t,tp} = \sum_{t=1}^{n} (l_t * w_t) * z_{t,tp} \tag{4.4}$$

The total value V can subsequently be calculated by summing over all timepoints:

$$V = \sum_{tp \in TP} V_{tp} \tag{4.5}$$

With the addition of time, demands and deadlines, the objective of the MLOPP with due dates becomes more complex. The problem gets converted from a single-objective optimisation problem to a multi-objective optimisation problem. The first objective remains the maximisation of the total value V. The second objective wishes to minimise the deviation from the MPS. Only items produced before a deadline can count towards that deadline, after subtracting the demands from the previous deadlines. Producing too little and missing deadlines should be punished. Similarly, producing too many resulting in leftovers at the last timepoint tp_{max} should also be punished. These two objectives sometimes contradict each other. How they should be scalarised and balanced together to obtain a single-objective optimisation problem is left open to the implementation.

4.5.5 Weaving

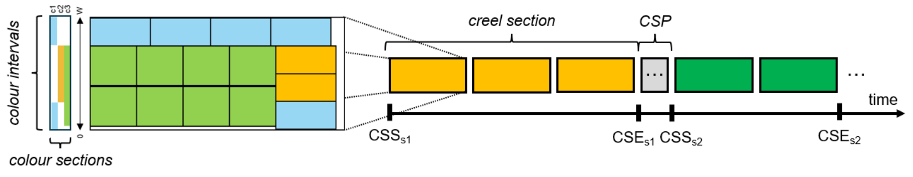
This section includes the last extension to achieve the complete WPAS formulation for the textile industry. The notion of packing, planning, and scheduling of (textile) items is already supported by the previous models up until this point. The last additions to the problem formulation should be the loom's creel and the production constraints that are associated. The extension can both be formulated on top of basic SLOPP (with a single object) and on top of the due-date extended SLOPP. In the latter case, the manual labour cost of retooling or reconfiguring the creel has to be taken into account.

Whilst yarns can have many different characteristics, WPAS will only be concerned with the colour of the material. The assignment of certain colours at certain dent positions based on the installation of bobbins can be called a *creel configuration* and is visualised in *figure 4.4*. The period in time that this configuration is used gets named a *creel section*. A production planning for WPAS can consist of multiple creel sections $s \in CS$. For every section s it must be defined when it starts (CSS_s) and when it ends (CSE_s).

Within each creel section, there is the notion of *colour sections*: regions across the width of the machine (at the different dent positions) where certain colours are present. For each colour $c \in C$, with C the collection of all available colours, a colour section can be defined. It is a group of intervals $ci_{s,c,j_c} \subseteq [0, W] \in CI_{s,c}$, $j_c \in J_c \subset \mathbb{N}$ which define where that colour c is present in section s.

Besides the additional constraints associated with the creel, the optimisation problem still has the same objective; optimally fulfil the MPS when extending MLOPP, otherwise minimal waste.

Figure 4.4: **Creel configuration**



A) Input

The input for the WPAS problem only differs slightly from that of (the due-date extension of) SLOPP. Additional information needs to be provided with regards to the practical limitations of the creel. First, there is the maximal number of colours CC_{max} that the creel supports. This represents the limited number of unique bobbins which can be mounted for each dent position along the width of the machine. It will impose a limitation with respect to which textiles (with different colours) can be produced in sequence one after the other without the need for reconfiguring the machine. Second, there is the maximal number of "*creel sections*" CS_{max}, continuous sequences of planned patterns that will be produced with the same creel configuration. At each transition of one creel section to another, a machine reconfiguration necessarily has to occur which results in downtime. Expert planners typically want to set a limit to the number of these costly reconfigurations $\#CS$. Third is the cost or penalty CSP for switching creel configuration, a duration of downtime expressed as a number of unusable production segments. Lastly, there is the colour information of each textile to be produced. This has to be expressed with regards to their usage of a subset of the colours provided by the available yarns.

B) Output

As mentioned, the objective of the WPAS problem is exactly the same as its underlying MLOPP or SLOPP problem. The output now also includes the configuration of the creel across the timeline; creel section (CSS_s, CSE_s) and creel configurations determined by colour sections and their intervals ($ci_{s,c,j}$).

Chapter 5

Approach

Now that the problem statements have been formalised, different algorithms and models to solve the optimisation problems can be discussed. In this chapter, some mathematical models for the different sub-problems will be presented. Starting from SLOPP and slowly working up towards a model for the complete WPAS problem. The models each utilise a different representation, which plays a significant role in their characteristics. Choosing an appropriate representation can transform a problem into one that is easier to solve, but might have hidden consequences like a reduced level of extensibility. In the case of WPAS, a good model should be both scalable and extensible: capable of solving large production plannings with added custom requirements.

As the sub-problems all belong to the family of constrained optimisation problems, the discussion of each model will include an overview of its variables (its chosen representation of the problem), its constraints, and its objective function. All models are represented using the principles of the *Constraint Programming* (CP) paradigm.

5.1 SLOPP models

In this section, three different mathematical programming formulations for the SLOPP optimisation problem will be presented. This includes **I)** a basic model as a simple baseline, **II)** a guillotine type model from literature and **III)** a novel anchor-based model as a contribution from this thesis. Each model utilises a different mathematical representation or encoding for the position and rotation of the packed item instances. The different sections explaining each formulation, together with the subsequent performance comparison in the next chapter, will demonstrate the importance of choosing an appropriate representation. It directly influences the solution quality, the model's performance, and the formulation's extensibility. This leads to a discussion on why the contribution of the newly proposed anchor model is of importance. All models have been implemented using CP and their performance will be compared later on.

5.1.1 Basic Model: CP-Base

The basic model entails a naive mathematical representation of SLOPP. It encodes all aspects of the optimisation problem without making any assumptions regarding its solution space or using any encoding tricks as to reduce the search space. Only a basic type of symmetry breaking gets utilised to make the upcoming benchmark problem instances more tractable. The basic nature of this model results in limited scalability, but makes it an ideal starting point from which to make further improvements. It also serves as a baseline against which to compare the performance of other models.

The definition of a mathematical model for a SLOPP optimisation problem consists of the following elements:

- representation of possible item position variables
- definition of the packing constraints
- optimisation objective

A) Variables

At the core of the mathematical model lies its set of variables. Searching for a solution to the optimisation problem corresponds to searching for an optimal value assignment to these variables. There is some flexibility in choosing which variables to use when representing the problem and thus how to encode its solutions. For SLOPP, the following elements of a solution have to be encoded:

- placement positions
- number of placed instances of each item type
- rotation of an item instance

Number of placed instances As the optimisation problem falls under the category of output maximisation [76], not all demanded small items will fit into a single large object. Additionally, as this paper concerns the special case of HM, a solution may fill the entire object with instances of just a single item type. This results in the model having to foresee enough decision variables as to allow for a maximal instantiation for each item type separately. With the previously defined MPM_t, the number of instances N_t of item type t to be initialised will be the upper bound MPM_t. As not all instances i_t of all item types t can simultaneously fit into the object, additional variables have to be defined representing the number of packed instances $n_t \in [0, N_t]$ of each item type.

Active instances For each instantiated instance i_t of item type t, an auxiliary variable a_{t,i_t} can be introduced. It signifies whether that instance is active, i.e. whether it is packed in the larger object.

Item selection As to simplify the further specification of the model, an additional type selection variable s_t gets introduced. It is a Boolean representing whether at least one instance of type t gets packed.

$$\begin{cases} s_t = 1 & \text{if } n_t > 0 \\ s_t = 0 & \text{if } n_t = 0 \end{cases} \tag{5.1}$$

Rotation The chosen SLOPP formulation allows for rotations by 90°, keeping all item sides parallel with those of the object. To encode these rotations, one Boolean variable r_{t,i_t} gets defined per item instance i_t of type t.

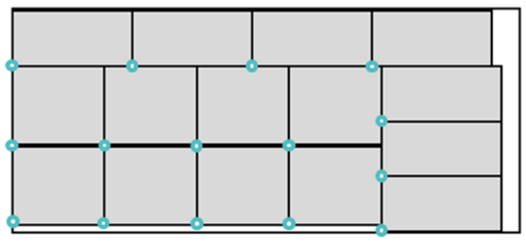
Position As the optimisation problem entails finding the optimal placement of a selection of smaller items within the available space of the larger object, a core element of its mathematical representation is determined by the formulation of possible item positions. Furthermore, when it comes down to implementing the model with CP, the chosen representation determines many of the auxiliary decision variables and the formulation of the constraints.

Different encoding techniques exist, like order encoding and set-of-points encoding. With ***order encoding***, the placement of the smaller items is defined relatively, i.e. whether one item is placed above, below, to the left, or to the right of some other item. A benefit of this representation is that it intrinsically enforces non-overlap of the items without the need for any further constraints. An apparent downside is the lack of absolute positional information. The basic model will be based on the ***set-of-points encoding***, a representation used by many sources in the literature. The packing of each item instance i_t of item type t can be encoded using a single point $p_{t,i_t} = (x_{t,i_t}, y_{t,i_t})$. It represents the position of the lower left corner of an item instance within the larger object, as shown in *figure 5.1*. Due to implementation constraints, only integer point coordinates will be considered. For each item, the coordinates x_{t,i_t} and y_{t,i_t} must thus be chosen from domains $X_t = \{0, 1, ..., W - min(w_t, l_t)\} \subset \mathbb{N}$ and $Y_t = \{0, 1, ..., L - min(w_t, l_t)\} \subset \mathbb{N}$ respectively, with W and L the dimensions of the larger object and w_t and l_t the dimensions of item type t. These domains determine the possible value ranges of the placement decision variables. The min-operation is needed as it is not known beforehand which rotation an instance will have. Additional constraints will thus be needed to guarantee that all packaged items stay within the object's boundaries.

TABLE 5.1: **SLOPP basic model variables**

Variable	Domain	Explanation
n_t	$[0, N_t] \subset \mathbb{N}$	Number of packed instances of type t
a_{t,i_t}	$(0, 1) = \mathbb{B}$	Active Boolean for instance i_t of type t
s_t	$(0, 1) = \mathbb{B}$	Packing Boolean of type t
r_{t,i_t}	$(0, 1) = \mathbb{B}$	Instance rotation by 90°
x_{t,i_t}	$[0, W - min(w_t, l_t)] \subset \mathbb{N}$	x-coordinate lower-left corner of instance i_t of type t
y_{t,i_t}	$[0, L - min(w_t, l_t)] \subset \mathbb{N}$	y-coordinate lower-left corner of instance i_t of type t

Figure 5.1: **Set-of-points encoding**



Visualisation of packaged items with their respective set-of-points in the lower-left corner.

B) Constraints

Now that the variables of the model have been defined, the necessary constraints can be formulated across them. These constraints restrict the solution space and define the satisfiability criteria of a feasible cutting pattern for the constraint satisfaction problem.

Selected item The first constraint is an exact implementation of the selection variable s_t's definition:

$$s_t \iff (n_t \neq 0) \tag{5.2}$$

Active instance The second constraint is also an exact implementation of a definition, namely that of the instance active variable a_{t,i_t}. Every instance i_t can be given a index number, i.e. i_t itself. Without loss of generality the assumption can be made that when packing instances of an item type, lower index numbers are always packed first. For each type t there thus exists an $\hat{i}_t$ such that all instances with a lower or equal index are active and those with a higher index are not.

$$\begin{cases} a_{t,i_t} = True & \text{if } i_t \leq \hat{i}_t \\ a_{t,i_t} = False & otherwise \end{cases} \tag{5.3}$$

Combined with variable n_t representing the number of placed instances, the constraint for a_{t,i_t} can be defined as follows:

$$a_{t,i_t} \iff (i_t < n_t) \tag{5.4}$$

Item count As to strengthen the model and reduce the search, an implied constraint for n_t can be added:

$$n_t = \sum_{i_t=0}^{N_t-1} a_{t,i_t} \tag{5.5}$$

Within object The domains of the set-of-points variables x_{t,i_t} and y_{t,i_t} already take somewhat into account the dimensions of the encompassing object. With the addition of rotation, these ranges have to be adapted. To more accurately restrict the domains based on the chosen rotation, the following two constraints get utilised:

$$\begin{aligned} x_{t,i_t} &\leq W - w_{t,i_t} * r_{t,i_t} - l_{t,i_t} * (1 - r_{t,i_t}) \\ y_{t,i_t} &\leq L - l_{t,i_t} * r_{t,i_t} - w_{t,i_t} * (1 - r_{t,i_t}) \\ &\text{with } r_{t,i_t} \text{ a Boolean variable} \end{aligned} \tag{5.6}$$

As the notion of "*width after rotation*" and "*length after rotation*" will also be needed in future constraints, two new symbols $\hat{w}_{t,i_t}$ and $\hat{l}_{t,i_t}$ can be introduced:

$$\begin{aligned} \hat{w}_{t,i_t} &= w_{t,i_t} * r_{t,i_t} - l_{t,i_t} * (1 - r_{t,i_t}) \\ \hat{l}_{t,i_t} &= l_{t,i_t} * r_{t,i_t} - w_{t,i_t} * (1 - r_{t,i_t}) \end{aligned} \tag{5.7}$$

The *within-object* constraint now simplifies to:

$$\begin{aligned} x_{t,i_t} &\leq W - \hat{w}_{t,i_t} \\ y_{t,i_t} &\leq L - \hat{l}_{t,i_t} \end{aligned} \tag{5.8}$$

Non-overlap The non-overlap constraints specify that no two packed item instances of any type can overlap. This results in the following constraints, one for each pair of instances:

$$\begin{aligned} (a_{t_1,i_1} \ \& \ a_{t_2,i_2}) \Rightarrow (\quad & (x_{t_1,i_1} + \hat{w}_{t_1,i_1} \leq x_{t_2,i_2}) \mid \\ & (x_{t_2,i_2} + \hat{w}_{t_2,i_2} \leq x_{t_1,i_1}) \mid \\ & (y_{t_1,i_1} + \hat{l}_{t_1,i_1} \leq y_{t_2,i_2}) \mid \\ & (y_{t_2,i_2} + \hat{l}_{t_2,i_2} \leq x_{t_1,i_1}) \ ) \\ \forall \ (t_1, t_2) \in T \times T, \ (i_1, i_2) \in \quad & ([0, N_{t_1} - 1] \times [0, N_{t_2} - 1]), \\ \text{s.t.} \quad \mathit{not}(t_1 = t_2 \ \& \ i_1 = i_2) & \end{aligned} \tag{5.9}$$

As the number of these constraints grows exponentially with the number of item instances, it is largely responsible for the size of the complete model. Any reduction in the number of these constraints can significantly improve the model's size and thus its performance. This is exactly what the later proposed anchor model will try to achieve.

Anti symmetry As mentioned before, in order to make the solving of this base model somewhat tractable a limited amount of symmetry breaking is added. For each item type, a partial spatial order within the item instances gets defined, greatly reducing the search space without any loss in generality:

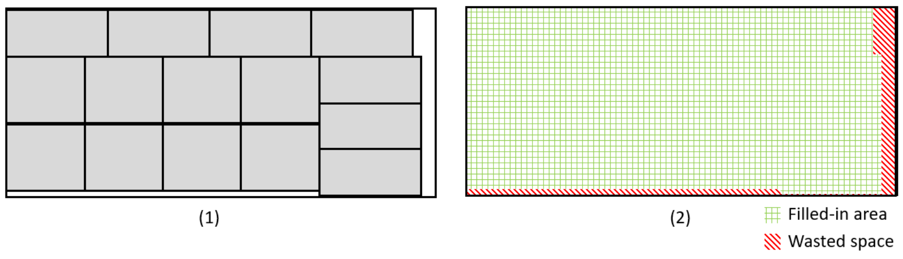
$$\begin{aligned} (a_{t,i_1} \ \& \ a_{t,i_2}) \Rightarrow (\quad & (x_{t,i_1} < x_{t,i_2}) \mid \\ & ((x_{t,i_1} = x_{t_1,i_2}) \ \ \& \ \ (y_{t,i_1} < y_{t,i_2}) \ ) \\ \forall \ t \in T, \ (i_1, i_2) \in ([0, \quad & N_t - 1])^2, \\ \text{s.t.} \quad i_1 \neq i_2 & \end{aligned} \tag{5.10}$$

C) Objective

The last component towards defining the base model is its objective function. Just like the constraints, it gets defined across the available decision variables. It allows for the distinction between the different solutions of the satisfaction problem by specifying a preference towards higher or lower values of the objective. In the case of SLOPP, which is an output maximisation problem, the filled in area of the object will be optimised. More specifically, the value V of the created packing should be optimised. This value is the sum of the values of the packed items (*equation 4.2*). As the input of the studied problem does not specify any preference towards a specific item type, the value of an item will correspond with its surface area $w_t * l_t$, visualised in *figure 5.2*. Without loss of generality, the maximisation of the filled-in area can be reformulated as the minimisation of the wasted space:

$$O_1 = W * L - \sum_{t \in T} w_t * l_t * n_t \tag{5.11}$$

Figure 5.2: **Filled-in area and wasted space**



(1) object with packed items, (2) objective O_1

When the solution contains wasted space, some items can move around freely without changing the objective. This results in an unnecessary increase in the size of the search space, as the current model does not define a preference amongst these equivalent solutions. As previously mentioned in *section 3.3 Related Literature*, there exist different techniques in the literature to reduce the set of possible positions whilst still preserving optimality, like normal patterns and reduced raster points. Whilst these techniques can improve the performance of this basic model, they introduce some structural assumptions. This can be restrictive for further extensions, which might need all possible placements. That is why this basic model tackles this problem via its objective function as a preference instead of a constraint. In order to improve the quality of the solution, a preference for the lower-left positioning of items gets added to the objective:

$$O_2 = \sum_{t \in T} \sum_{i_t = 0}^{N_t - 1} a_{t,i_t} * (x_{t,i_t} + y_{t,i_t}) \tag{5.12}$$

To prevent this objective from skewing the solution space and counteracting the original waste minimisation objective, a lexicographic ordering amongst the objectives gets defined. By scaling the original objective with a large value B, the second objective will only specify a preference amongst the optimal solutions of the first objective. For this SLOPP model, B will be chosen as $B = W * L$. Thus:

$$O = O_1 * B + O_2 \tag{5.13}$$

5.1.2 Guillotine Model: CP-Guillotine

Whilst the basic model serves as a simple baseline by utilising a naive problem representation, the guillotine model has a carefully designed encoding which improves the model's characteristics. The original formulation was first presented in Lodi & Monaci (2003) for the ***Two-Dimensional Knapsack*** (TDK) problem, which formulates the same optimisation problem as R_2D_SLOPP [60]. The extended version from Salem et al. (2023), a formulation specifically adapted for the textile industry and other industries with similar production processes [30], will be utilised in this chapter.

A) Guillotine assumption

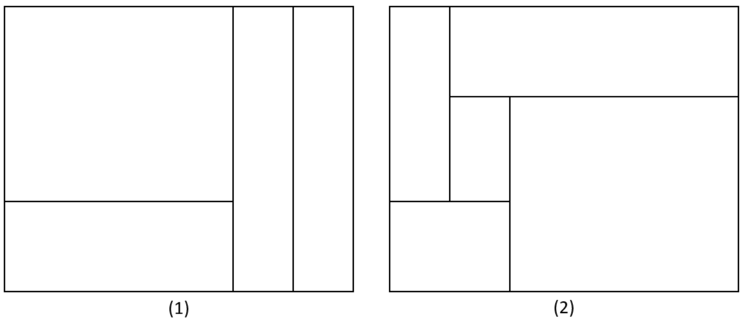
The model gains its performance improvements and reduced size from its reliance on the guillotine assumption. This assumption is inspired by the physical cutting process that often occurs in real industrial settings. In the case of weaving, after multiple textiles have been produced together in a large rectangular piece (the object), the individual textiles (the items) have to be cut out. When positioning the textiles, the physical limitations of the cutting process have to be taken into account. Assumptions that have already been used, are the rectangular shape of the items and their parallel positioning (allowing for 90° rotations) inside the object.

Another assumption that is typically related to the cutting machine is the usage of guillotine cuts, often called edge-to-edge cuts. It specifies that all pieces must be positioned such that the subsequent cutting process solely consists of parallel cuts from one side of the object (or a previously cut-out piece) to the complete other side. A cut can not stop somewhere within the object. *Figure 5.3* demonstrates the difference between guillotine cuts and non-guillotine cuts using an interlocking pattern. The cuts are performed in stages, giving the cutting pattern a hierarchical structure. A sequence of horizontal and vertical cuts gets performed up until a certain depth. The number of stages is often limited to just two or three, with an additional set of cuts allowing for the remaining waste material being separated from the pieces. The model of Salem et al. (2023) utilises a two-stage guillotine cutting pattern [30].

To realise this guillotine assumption, a totally different representation gets utilised. The entire structure of the model, together with the choice of its decision variables, is centred around these cuts. Instead of representing item positions with the coordinates of their lower left corners, the guillotine model formulates a collection of cutting sequences in order to produce the packed items.

In order to define the model, the notion of patterns, strips, and cuts will be introduced and will be visualised in *figure 5.4*. The larger object, often referred to as

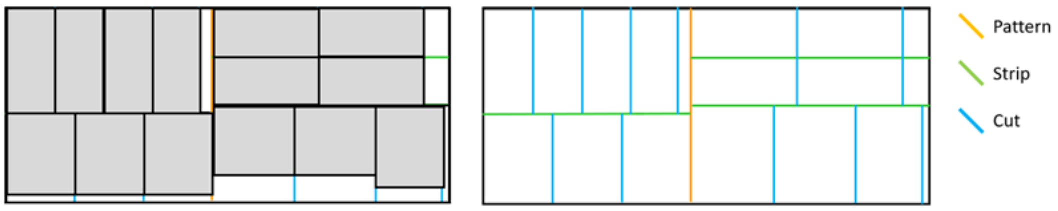
Figure 5.3: **Guillotine cuts**



(1) guillotine cuts, (2) non-guillotine cuts

the material stock in industrial applications, will be cut into smaller pieces with a sequence of cutting stages. Every stage consists of a collection of parallel cuts, both with each other and with the edges of the object, alternating between horizontal and vertical across the different stages. After the cuts of the zeroth stage, the object gets divided into a collection of ***patterns***. These patterns subsequently get split into ***strips*** after the first stage. At the second stage, so-called ***cuts*** get created.

Figure 5.4: **Guillotine cutting pattern**



A packing and its underlying guillotine cutting pattern.

The final result is a two-stage guillotine cutting pattern. By restricting to two stages, an assumption on the solution space gets made. This can result in certain optimal solutions of SLOPP no longer being feasible. One example is the circular interlocking of four items, visible in *figure 5.3 (2)*. The reason that this assumption often occurs within the literature is twofold. Firstly, the link with the real-world cutting machine itself. Due to practical constraints, many machines can only make cuts with a limited number of guillotine stages. Even if a SLOPP model allows for any type of cutting pattern, in practise guillotine cuts are often preferred. Secondly, the performance implication. Thanks to its completely different representation of SLOPP, with an entirely different variable encoding for its solutions, the model has been significantly reduced in size. As seen for both the basic model (*subsection 5.1.1 Approach*) and as will be the case for the later introduced anchor model (*subsection 5.1.3 Approach*), the non-overlap constraints account for the largest source of the models' size. By no longer representing a solution as a collection of points, but rather as a limited number of cuts, non-overlap constraints are no longer needed as they have become implicit.

The reliance on the guillotine assumption does pose a chicken-and-egg problem. The assumption is often made due to the physical restriction of most cutting machines. At the same time, most companies do not invest in the more complex machines due to restrictions imposed by the assumptions of the planning software. Besides the reduced solution space, an additional downside of the guillotine implementation is the loss of absolute positional information. The lower left position of the item instances is no longer directly available. If needed, it can be derived from the encoded cutting locations but at a hefty price in model size, undoing the original benefit of the guillotine assumption. This addition can be found in *appendix A*. Without introducing additional variables for each cutting location and each item type, no information can be derived regarding the position of empty space between items. For this basic SLOPP problem both limitations are not an issue, but when further extensions get built on top it will present a problem.

B) Variables

Variables have to be introduced to represent the locations of the different cuts. Each subsequent cut splits the object into *patterns*, *strips*, and lastly *cuts*. In order to support all possible cutting patterns, a maximal number of cuts have to be initialised, each with its own Boolean variable representing whether or not it is in use. The reason for the maximal initialisation of cuts is exactly the same as for the maximal initialisation of item instances in the set-of-points based models; support as many packing configurations as possible. Given the lengths of all items, a minimal length for a pattern can be determined. Each utilised pattern should hold at least one item, giving the pattern a minimal size of the smallest side of any item. Subsequently, the maximum number of patterns ($\#P$) can be determined:

$$\#P = \left\lfloor \frac{L}{\min\limits_{t \in T}(l_t)} \right\rfloor \tag{5.14}$$

Similarly, the maximal number of *strips* per *pattern* ($\#A$) can be determined:

$$\#A = \left\lfloor \frac{W}{\min\limits_{t \in T}(w_t)} \right\rfloor \tag{5.15}$$

Once again for the number of *cuts* per *strip* ($\#B$):

$$\#B = \#P \tag{5.16}$$

Given these metrics, the following domains can be defined:

$$\begin{aligned} P &= [0, \#P - 1] \subset \mathbb{N} \\ A &= [0, \#A - 1] \subset \mathbb{N} \\ B &= [0, \#B - 1] \subset \mathbb{N} \end{aligned} \tag{5.17}$$

With $\#T$ the number of item types ($T = [0, \#T - 1] \subset \mathbb{N}$) and $\#W$ the number of unique item widths ($W = [0, \#W - 1] \subset \mathbb{N}$), the variables of the model can be defined. The explanation can be found in *table 5.2*.

Table 5.2: **SLOPP guillotine model variables**

Variable	Domain	Index	Explanation
β_p	$(0,1) = \mathbb{B}$	$p \in P$	if the p^{th} cutting pattern is used
pl_p	$\left[\min_{t \in T}(l_t), L\right] \subset \mathbb{N}$	$p \in P$	length of the p^{th} cutting pattern
$\gamma_{p,a,w}$	$(0,1) = \mathbb{B}$	$p \in P, a \in A$, $w \in W$	if the a^{th} strip of the p^{th} pattern produces an item of the w^{th} width
$\sigma_{p,a,b,t}$	$(0,1) = \mathbb{B}$	$p \in P, a \in A$, $b \in B$, $t \in T$	if the b^{th} cut of the a^{th} strip of the p^{th} pattern produces the t^{th} item type

The standard guillotine formulation does not support item rotation. To circumvent this restriction, all item types t can be duplicated with one copy for each rotation (t and $\tilde{t}$). The combination of both will be referred to using $\bar{t}$.

As is immediately noticeable, this new representation trades a reduction in the number of constraints for a significant increase in the number of variables. Section 6 will demonstrate this observation on a set of benchmark problem instances. These four sets of variables completely specify a packing solution. As mentioned previously there is no knowledge of the absolute position of the items, only their relative position inside the cutting pattern.

C) Constraints

As mentioned before, the guillotine model does not have to formulate non-overlap constraints as the non-overlap property is inherent to this representation. The remaining constraints are all related to the definition of the chosen variables, to the guarantee that the bounds of the object are respected, and to some performance-improving additions.

To prevent degeneracy within the representation, precedence between the *patterns* p is enforced:

$$\beta_{p-1} \geq \beta_p \tag{5.18}$$

All *patterns* together should not exceed the object's length L:

$$\sum_{p \in P} pl_p \leq L \tag{5.19}$$

To prevent degeneracy within the representation, precedence between the *strips* a is enforced:

$$\sum_{w \in W} \gamma_{p,a-1,w} \geq \sum_{w \in W} \gamma_{p,a,w} \tag{5.20}$$

A *strip* can only have one width:

$$\sum_{w \in W} \gamma_{p,a,w} \leq 1 \tag{5.21}$$

If a *pattern* is in use, so should the first *strip* within that pattern:

$$\sum_{w \in W} \gamma_{p,1,w} \geq \beta_p \tag{5.22}$$

With w_w the w^{th} unique item width, the combined width of all *strips* within a *pattern* should not exceed the width W of the object:

$$\sum_{a \in A} \sum_{w \in W} w_w * \gamma_{p,a,w} \leq W \tag{5.23}$$

To prevent degeneracy within the representation, precedence between the *cuts b* is enforced:

$$\sum_{t \in T} \sigma_{p,a,b,t} \leq \sum_{t \in T} \sigma_{p,a,b-1,t} \tag{5.24}$$

Each *cut* can produce in only one item type t:

$$\sum_{t \in T} \sigma_{p,a,b,t} \leq 1 \tag{5.25}$$

If a *strip* is in use, only *cuts* resulting in items smaller than or equal to the *strip*'s width can be located inside it:

$$\sum_{t \in T} \sigma_{p,a,b,t} \leq \gamma_{p,a,w} \tag{5.26}$$

The sum of all *cut*-lengths of each *strip* must be less then its *pattern*'s length:

$$\sum_{t \in T} \sum_{b \in B} l_t * \sigma_{p,a,b,t} \leq pl_p \tag{5.27}$$

For added symmetry breaking and improved performance, the following three constraints are added:

Patterns are ordered by decreasing lengths:

$$pl_p \leq pl_{p-1} \tag{5.28}$$

Strips are ordered by decreasing widths:

$$\sum_{\substack{\hat{w} \in W \\ s.t. \hat{w} \geq w}} \gamma_{p,a-1,\hat{w}} \geq \gamma_{p,a,w} \tag{5.29}$$

Cuts are ordered by decreasing lengths:

$$\sum_{\substack{\hat{t} \in T \\ s.t. l\hat{t} \geq l_t}} \gamma_{p,a,b-1,\hat{t}} \geq \gamma_{p,a,b,t} \tag{5.30}$$

At last, a small addition to the model links the guillotine formulation to the previous concept of item counts $n_{\bar{t}}$:

$$n_{\bar{t}} = \sum_{p \in P} \sum_{a \in A} \sum_{b \in B} \sigma_{p,a,b,t} + \sum_{p \in P} \sum_{a \in A} \sum_{b \in B} \sigma_{p,a,b,\tilde{t}} \tag{5.31}$$

D) Objective

As the degeneracy preventing constraints already enforce a lower-left positioning of the items, the sole objective comes down to the minimisation of the wasted material: *equation 5.11*.

5.1.3 Anchor Model: CP-Anchor

Now that the two models to compare against (CP-Base as baseline and the CP-Guillotine from the literature) have been discussed, this section can cover the anchor model as a contribution of this thesis.

A) Motivation

As mentioned before, one aspect where the basic model can significantly be improved is in terms of the number of constraints needed to formulate the non-overlap condition. CP-Base used a rather naive implementation where one additional constraint gets specified for every pairwise combination of items instances. This results in an exponential increase in model size with respect to the problem size. CP-Guillotine does not share this fate as it utilises a different representation style based on a guillotine-cuts encoding instead of the more typical set-of-points encoding. This prevents the need for non-overlap constraints as it becomes implicit, thus improving the model's scalability. Unfortunately this representation technique is based upon the guillotine assumption, removing valid patterns from the solution space. Not every packing can be encoded using this representation, in some cases leaving the most optimal solution unreachable. Additionally, as will later be discussed in *chapter 6 Experiments*, the guillotine model does not perform so well with respect to extensibility.

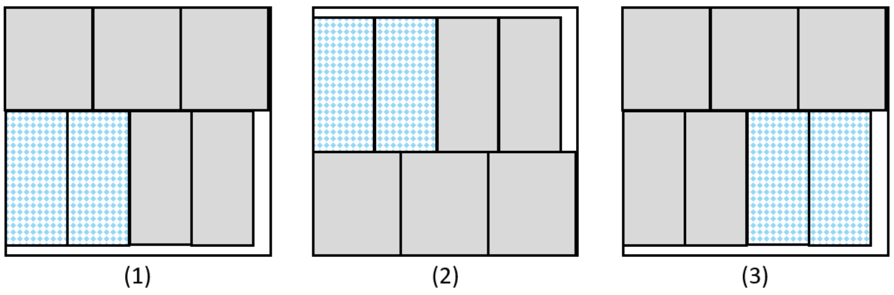
The anchor model introduces a novel anchor based packing representation. It is based upon the set-of-points encoding. It allows for a strong reduction in the model size for large-scale problems, without losing any kind of generality with respect to the basic model. The anchor model has the exact same solutions space as the basic model, but at a significantly reduced solving complexity.

B) Origin of unscalability

Many different techniques already exist to reduce the model complexity of SLOPP. Oftentimes, domain knowledge gets utilised to make certain assumptions on the solution space. For example in the metal industry, by looking at the physical process of cutting material from a larger sheet, guillotine cuts can be assumed as to make the cutting process practically viable [72, 75]. The anchor model stands in contrast with these types of techniques, as it makes no assumptions and produces the exact same results as the basic model but at a lower computational cost.

One of the main problems behind the unscalability of the basic model is its amount of symmetry, causing degeneracy in the solution space. Not only can (part of) the contents of an object be mirrored and still result in an equivalent solution (*global symmetry*), but due to the high multiplicity of the items, different instances can be interchanged without impact upon the solution (*local symmetry*). Both can be seen in *figure 5.5*. This degeneracy makes the search space unnecessarily large and makes the problem more difficult for the solver to optimise. Whilst the *global symmetry* is best left untouched to keep the model as general as possible by not filtering out any of these symmetries, the *local symmetry* is purely an artefact of the chosen representation. Two instances of the same item type can be interchanged, switching all variable assignments around, with no noticeable change to the solution. This is the largest source of degeneracy in the solution space for the basic model and thus the main culprit for its unscalability.

FIGURE 5.5: **Symmetry**



(1) original, (2) global symmetry, (3) local symmetry

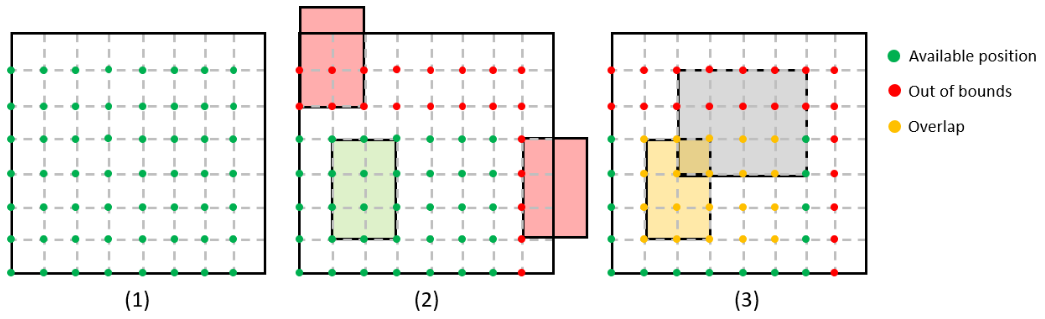
This degeneracy also has a direct impact on the size of the model. Every item type has as many instances initialised as its MPM_t, all of which can be placed anywhere inside the object. As there is no knowing beforehand which item instances will be neighbours, non-overlap has to be formulated for every possible pairwise combination resulting in an exponential model size. Part of this problem was already tackled by a limited amount of symmetry breaking in the basic model. Whilst this reduced the search space by defining a partial spatial order, all non-overlap constraints still had to be specified.

The contribution of the anchor model is to make clever use of patterns for the possible item positions, allowing instances to be bound to specific anchor locations inside the object. This introduction of structure significantly reduces the model's degeneracy, improving both its formulation size and its efficiency whilst solving.

C) Ground-out model

The anchor model can be seen as a stage between the basic set-of-points representation and a fully ground-out representation. The former has been explained in a previous section as the basic model formulation (*subsection 5.1.1 Approach*). With the latter, for every possible integer position of an item, a Boolean placement variable gets created. This variable indicates whether an instance of the respective item type is packed at that position (the lower-left corner) in the object. This can be seen in *figure 5.6*. The ground-out representation removes the need for positional variables, as every instance is bound to a single fixed location. Non-overlap can now be defined as a simple NAND-condition between placement variables that fall in each other's item regions.

Figure 5.6: **Ground-out placement variables**



(1) ground-out placement variables, (2) out of bounds, (3) non-overlap

This ground-out representation allows for:

1. Total ordering between instances of the same type (reducing search space).
2. Selectively specifying non-overlap with neighbouring instances.
3. Allowing extensions to selectively refer to instances in certain regions.

This beneficial structure inside the representation comes at the cost of a significant increase in the number of item instances and thus the number of variables. For each item type t, $(W - w_t) \times (L - l_t)$ Boolean placement variables are needed, instead of the previously $2 * N_t$ variables for the x_{t,i_t} and y_{t,i_t} coordinates. For large problem instances with a large object size this results in many more variables. The anchor model has a similar philosophy, allowing for the same benefits, but without this large increase.

D) Anchor structure

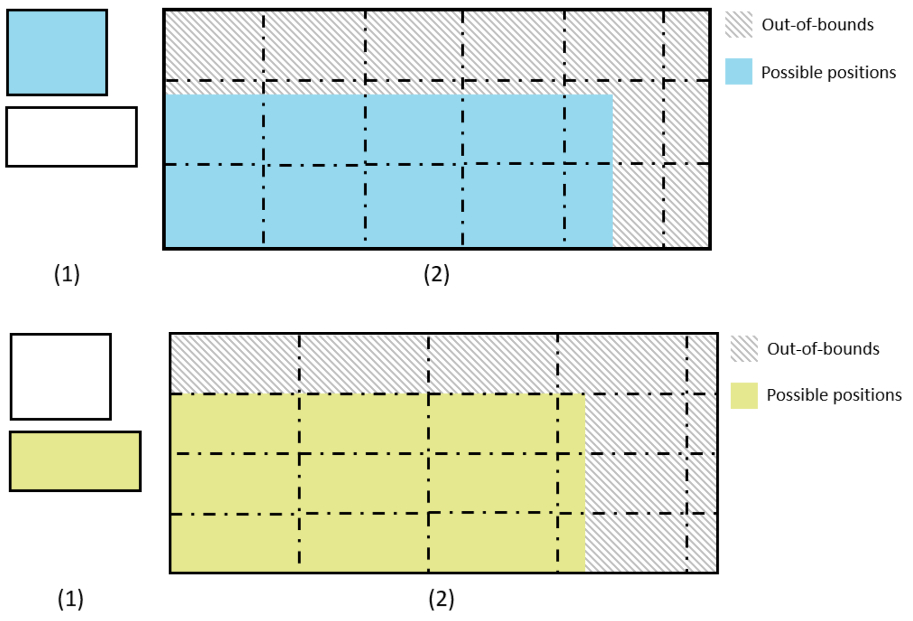
To solve the problem of degeneracy due to interchangeable item instances of the same type, the anchor model imposes a positional structure to these instances. To support every possible packing, MPM_t item instances of each type t should be initialised. The set-of-points representation of the basic model will still be used, representing the position of each instance using coordinate variables. Instead of allowing every instance to be placed anywhere inside the object or fixing each instance to a singular location, a placement pattern with anchor positions gets created. This limits the range of motion of each instance to a small region, gaining similar benefits as the ground-out model without the large increase in the number of item instances.

Given a certain item type t, based on its shape a grid pattern can be overlayed on top of the object. With an object shape of $W \times L$ and an item shape of $w_t \times l_t$, the grid pattern has a cell count of $ccw_t = \lfloor \frac{W}{w_t} \rfloor$ along the width and $ccl_t = \lfloor \frac{L}{l_t} \rfloor$ along the length of the object. The grid pattern itself can then be defined as $\{w_t, 2 * w_t, ..., n_{w_t} * w_t, ..., ccw_t * w_t\} \times \{l_t, 2 * l_t, ..., m_{l_t} * l_t, ..., ccl_t * l_t\}$. This grid structure is visualised for two item types in *figure 5.7*. Certain regions are labelled as *out-of-bounds*, since positioning an item instance here would result in leaving the boundaries of the object. Each cell $[n_{w_t}, n_{w_t} + 1) \times [m_{l_t}, m_{l_t} + 1)$ within the grid defines a possible anchor position (n_{w_t}, m_{l_t}). Linking an item instance to one such anchors limits the movable range of its *placement point* (lower-left corner) to the region of that cell:

$$w_t * \begin{cases} [n_{w_t}, n_{w_t} + 1)) & if\ n_{w_t} \neq ccw_t - 1 \\ [n_{w_t}, \frac{W}{w_t} - 1) & otherwise \end{cases} \times l_t * \begin{cases} [m_{l_t}, m_{l_t} + 1) & if\ m_{l_t} \neq ccl_t - 1 \\ [m_{l_t}, \frac{L}{l_t} - 1) & otherwise \end{cases} \tag{5.32}$$

This is visualised in *figure 5.8*.

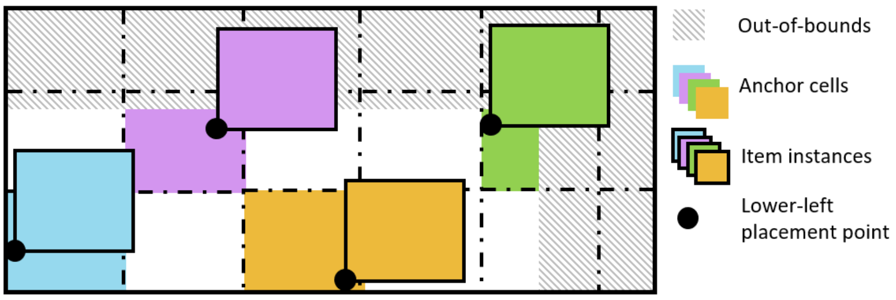
FIGURE 5.7: **Grid structure**



(1) selected item type, (2) grid pattern

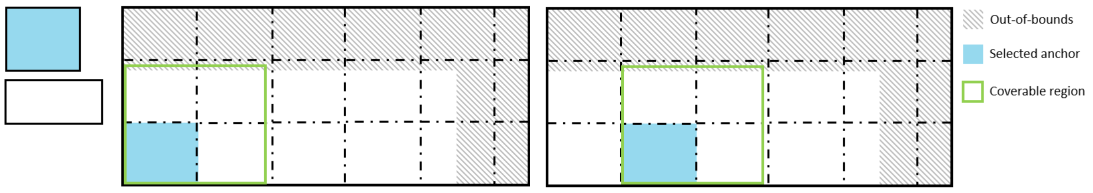
Figure 5.9 visualises the *coverable region* for the instance bound to one of the corners anchor (the space that the instance can occupy depending on its position) and for an instance bound to its neighbouring anchor at the right. Due to the open bounds of each cell region, the *coverable region* does not spread across its neighbouring cells entirely. In a practical implementation, the region will be short of one integer resolution to entirely cover its neighbouring cells. Every instance i_t of a certain item type t gets bounded to one of such anchor positions and every anchor position gets assigned exactly one item instance. As the *coverable regions* of

Figure 5.8: **Grid with anchor positions**



a cell never fully cover their neighbouring cells, each instance position that a cell can represent is completely unique to that cell. The anchor representation thus completely eliminates the local symmetries and instances can no longer be interchanged without representing a different solution. More formally, every possible solution packing p_{SAT} has a unique set of a_{t,i_t} representing the active instances and thus the active anchor positions. A different combination of a_{t,i_t}'s necessarily represents a different packing with a different combination of items.

Figure 5.9: **Anchor coverable regions**



This technique has the same spirit as the *ground-out* representation, resulting in all the same desired properties, whilst preventing an increase in the number of variables. Item instances are still able to move, preventing the need for a new instance for every possible position, but are at the same time restricted to a local region designed in such a way as to prevent local symmetries. No assumptions on the solution space are made and thus the generality of the basic model is preserved, i.e. every possible solution of a SLOPP can still be represented.

Besides an improvement in the representation of packings, the anchor encoding also has benefits when it comes down to the constraints of the model. As each instance has a limited range of motion, location-bounded constraints can selectively be formulated over a limited number of item instances. A simple example would be a constraint that is only applicable in the first half of the object, limiting the formulation to only the first half of anchor positions. An important collection of constraints that can benefit from this rigid structure are the non-overlap constraints. Due to the limited motion, item instances can only overlap with their direct neighbours within the grid structure. This insight can significantly reduce the model size. The use of the set-of-points encoding also has benefits when it comes down to location-bounded constraints. It for example allows for the formulation of restrictions on the relative position of two items, which would be much more complex for guillotine-based models.

FIGURE 5.10: **Neighbours of same item type**

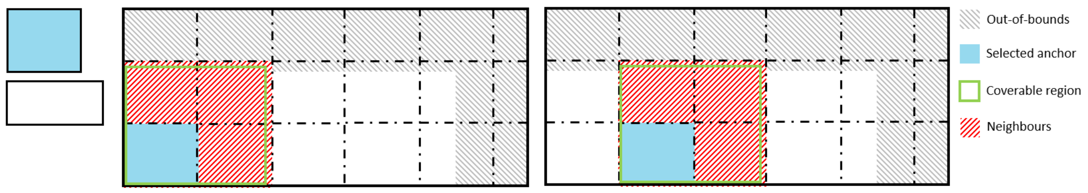
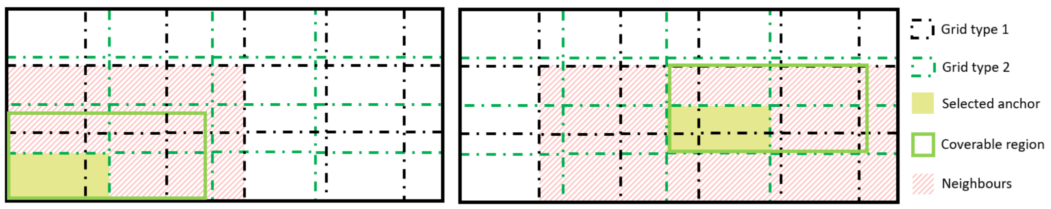


FIGURE 5.11: **Neighbours of different item type**



Between items of the same type, the neighbours are just the next cell over (*figure 5.10*). Between different types it can be determined by looking at the overlap of the respective grid patterns (*figure 5.11*). Cells with overlapping *coverable regions* hold item instances that might overlap depending on their positioning. As this instance position within a cell is not known beforehand, the non-overlap constraint must be formulated between every cell with overlapping coverable regions.

The anchor representation assumes fixed item shapes and subsequently does not directly support rotatable items. As with the guillotine model, this can be circumvented by duplicating all items. One copy for every orientation.

E) Variables

A solution of the anchor model has the exact same encoding as one of the basic model. All the changes take place in the internal structure of the model and in the generation of the constraints. The chosen decision variables are exactly the same aside from the rotation variables being removed. By linking every item instance to a single anchor location, the domain of the positional variables shrink by limiting to the anchor's cell region. Instances will no longer be referred to using a single index i_t, but rather a combination of u_t and v_t to represent the respective column and row within the grid structure where the cell is located. So a_{t,i_t} now becomes $a_{t,(u_t,v_t)}$. To avoid cluttering the notations, u_t and v_t will be represented as u and v but are in reality still dependant upon the item type t

F) Constraints

The ***Item count*** (*eq. 5.5*) and ***Selected item*** (*eq. 5.2*) constraints are exactly the same as the formulations from the basic model. ***Active instance*** (*eq. 5.4*) is removed as the "activeness" of an instance is no longer bound to n_t, but rather the positional placement of the instance. ***Within object*** (*eq. 5.8*) is also no longer needed with the removal of the rotation variable r_{t,i_t}, just like the ***Anti symmetry*** (*eq. 5.10*) constraint due to the removal of local symmetries. The only thing that remains is a new definition of non-overlap.

Non-overlap The new formulation of non-overlap will make use of the ability to selectively formulate constraints based on spatial locality. For instances of the ***same item type***, simple locality based on neighbouring cells can be utilised:

$$\begin{aligned}
\forall t \in T,\quad & (u,v) \in [0, u_{max}-1] \times [0, v_{max}]: \\
& (a_{t,(u,v)} \;\;\&\;\; a_{t,(u+1,v)}) \implies (x_{t,(u,v)} + w_t \leq x_{t,(u+1,v)}), \\
\forall t \in T,\quad & (u,v) \in [0, u_{max}] \times [0, v_{max}-1]: \\
& (a_{t,(u,v)} \;\;\&\;\; a_{t,(u,v+1)}) \implies (y_{t,(u,v)} + l_t \leq y_{t,(u,v+1)}), \\
\forall t \in T,\quad & (u,v) \in [1, u_{max}] \times [0, v_{max}-1]: \\
& (a_{t,(u,v)} \;\;\&\;\; a_{t,(u-1,v+1)}) \implies \begin{pmatrix} (x_{t,(u-1,v+1)} + w_t \leq x_{t,(u,v)}) \mid \\ (y_{t,(u,v)} + l_t \leq y_{t,(u-1,v+1)}) \end{pmatrix} \\
\forall t \in T,\quad & (u,v) \in [0, u_{max}-1] \times [0, v_{max}-1]: \\
& (a_{t,(u,v)} \;\;\&\;\; a_{t,(u+1,v+1)}) \implies \begin{pmatrix} (y_{t,(u,v)} + l_t \leq y_{t,(u,v+1)}) \mid \\ (y_{t,(u,v)} + l_t \leq y_{t,(u+1,v+1)}) \end{pmatrix}
\end{aligned} \tag{5.33}$$

The first two equations of 5.33 formulate the non-overlap of neighbouring cells in respectively the width-wise and the length-wise direction. The next two equations are for the cells neighbouring in a diagonal direction. As can be seen, the equations are quite a bit shorter since the rigid structure also allows for selective formulation of the exact direction in which instances should move as to not overlap. For example, two instances with horizontally neighbouring anchor positions (u, v) and $(u+1, v)$ must be positioned next to each other $(x_{t,(u,v)} + w_t \leq x_{t,(u+1,v)})$, as any other relative positioning is not possible due to the limited movement of each instance.

For the non-overlap between instances of ***different item types***, some preprocessing has to be done to determine the neighbours based on overlapping *coverable regions* of the grid patterns. *Algorithm 1* demonstrates how this neighbourhood gets determined. Given two item types, one gets chosen to be t_1 and the other t_2. Two types are never compared against each other twice. Whilst going over the anchor positions (u_{t_1}, v_{t_1}) of t_1, the neighbourhood region of possibly overlapping t_2 instances gets determined using upper and lower bounds on u_{t_2} and v_{t_2}. Every anchor (u_{t_2}, v_{t_2}) of t_2 is subsequently used to formulate a non-overlap condition with t_1. All these conditions get put together, resulting in the collection C.

In *algorithm 2*, the function enforcing the required non-overlap between neighbouring instances gets defined. Once again, the constraint can be defined more selectively using some additional preprocessing at model definition time. Moving work to definition time reduces the complexity of the model at solve time.

Algorithm 1
Preprocessed neighbourhood determination for non-overlap formulation between different item types

Input: object and item dimensions: w_t, l_t, W, L
Output: collection of constraints C

```
1: procedure NonOverlapDifferentItemType
2:     Initialise $C = \emptyset$
3:     for $u_{t_1} \leftarrow 0$ to $u_{t_1,max}$ do
4:         for $v_{t_1} \leftarrow 0$ to $v_{t_1,max}$ do
5:             $u_{t_2,lower} \leftarrow \max(\left\lceil \frac{u_{t_1} * w_{t_1} - 2 * w_{t_2} + 1}{w_{t_2}} \right\rceil, 0)$
6:             $u_{t_2,upper} \leftarrow \min(\left\lfloor \frac{(u_{t_1} + 2) * w_{t_1} - 1}{w_{t_2}} \right\rfloor, u_{t_2,max})$
7:             $v_{t_2,lower} \leftarrow \max(\left\lceil \frac{v_{t_1} * l_{t_1} - 2 * l_{t_2} + 1}{l_{t_2}} \right\rceil, 0)$
8:             $v_{t_2,lower} \leftarrow \min(\left\lfloor \frac{(v_{t_1} + 2) * l_{t_1} - 1}{l_{t_2}} \right\rfloor, v_{t_2,max})$
9:             for $u_{t_2} \leftarrow u_{t_2,lower}$ to $u_{t_2,upper}$ do
10:                for $v_{t_2} \leftarrow v_{t_2,lower}$ to $v_{t_2,upper}$ do
11:                    $C \leftarrow C \cup$ NonOverlapDifferentType$(u_{t_1}, v_{t_1}, u_{t_2}, v_{t_2})$
12:                end for
13:            end for
14:        end for
15:    end for
16: end procedure
```

An additional preprocessing formulation can be added in each loop of *algorithm 1* to remove unavoidable overlaps caused by the limited range of movement of each item instance (*algorithm 3*). This is especially the case for items located close to the edge of the object, limiting their range of motion even more. Based on the anchor cell space that is not out-of-bounds, a rectangular section of the *coverable region* gets determined that the instance of that cell always covers. When these rectangular sections of two item instances overlap, the instances themselves must by definition overlap. An example is given in *figure 5.12*.

Object capacity This is an additional implied constraint meant to strengthen the model. It formulates that the total area capacity of the object may not be exceeded.

$$\sum_{t \in T} \sum_{(u_t, v_t)} (a_{t,(u_t,v_t)} * v_t) \leq W * L \tag{5.34}$$

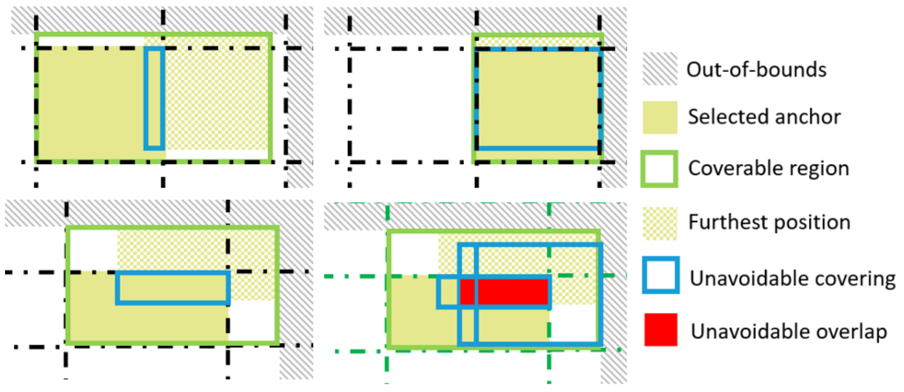
Algorithm 2 Selective non-overlap formulation for items of different type

```
1: function NonOverlapDifferentType($u_{t_1}, v_{t_1}, u_{t_2}, v_{t_2}$)
2:     Initialise $C = \emptyset$
3:     if $u_{t_1} * w_{t_1} \leq u_{t_2} * w_{t_2}$ then
4:         $C \leftarrow C \cup \left[x_{t_1,(u_{t_1},v_{t_1})} + w_{t_1} \leq x_{t_2,(u_{t_2},v_{t_2})}\right]$
5:     else if $u_{t_2} * w_{t_2} \leq u_{t_1} * w_{t_1}$ then
6:         $C \leftarrow C \cup \left[x_{t_2,(u_{t_2},v_{t_2})} + w_{t_2} \leq x_{t_1,(u_{t_1},v_{t_1})}\right]$
7:     end if
8:     if $v_{t_1} * l_{t_1} \leq v_{t_2} * l_{t_2}$ then
9:         $C \leftarrow C \cup \left[y_{t_1,(u_{t_1},v_{t_1})} + l_{t_1} \leq y_{t_2,(u_{t_2},v_{t_2})}\right]$
10:    else if $v_{t_2} * l_{t_2} \leq v_{t_1} * l_{t_1}$ then
11:        $C \leftarrow C \cup \left[y_{t_2,(u_{t_2},v_{t_2})} + l_{t_2} \leq y_{t_1,(u_{t_1},v_{t_1})}\right]$
12:    end if
13:    return $C$
14: end function
```

Algorithm 3 No-co-occurence constraint when overlap is inevitable

```
1: function RemoveGaruanteedOverlap($u_{t_1}, v_{t_1}, u_{t_2}, v_{t_2}$)
2:     Initialise $C = \emptyset$
3:     $a_1 \leftarrow x_{t_1,(u_{t_1},v_{t_1}),max}$
4:     $c_1 \leftarrow (u_{t_1} + 1) * w_{t_1} - (u_{t_1} * w_{t_1} - x_{t_1,(u_{t_1},v_{t_1}),min})$
5:     $b_1 \leftarrow y_{t_1,(u_{t_1},v_{t_1}),max}$
6:     $d_1 \leftarrow (v_{t_1} + 1) * l_{t_1} - (v_{t_1} * l_{t_1} - y_{t_1,(u_{t_1},v_{t_1}),min})$
7:     $a_2 \leftarrow x_{t_2,(u_{t_2},v_{t_2}),max}$
8:     $c_2 \leftarrow (u_{t_2} + 1) * w_{t_2} - (u_{t_2} * w_{t_2} - x_{t_1,(u_{t_2},v_{t_2}),min})$
9:     $b_2 \leftarrow y_{t_2,(u_{t_2},v_{t_2}),max}$
10:    $d_2 \leftarrow (v_{t_2} + 1) * l_{t_2} - (v_{t_2} * l_{t_2} - y_{t_2,(u_{t_2},v_{t_2}),min})$
11:    if $(\min(c_1, c_2) > \max(a_1, a_2))$ & $(\min(d_1, d_2) > \max(b_1, b_2))$ then
12:        $C \leftarrow C \cup \left[\neg\left(a_{t_1,(u_{t_1},v_{t_1})} \ \& \ a_{t_2,(u_{t_2},v_{t_2})}\right)\right]$
13:    end if
14:    return $C$
15: end function
```

FIGURE 5.12: **Unavoidable overlap**



G) Objective

Besides small adaptations to the indexing ((u, v) instead of i), the objective is exactly the same as for the basic model (*Eq. 5.13*).

5.2 Due date models

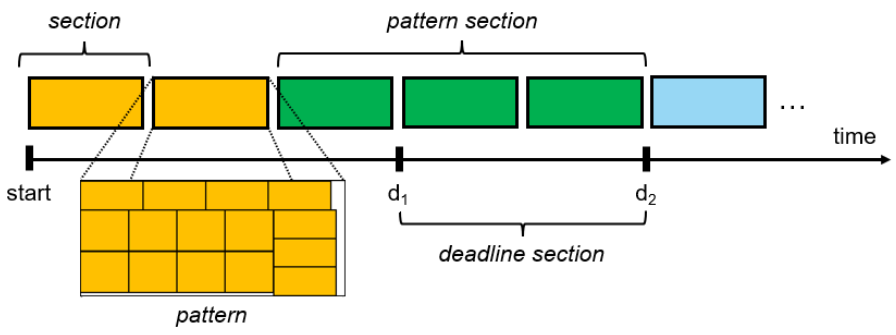
This section contains the first model extension of the standard SLOPP optimisation problem towards the weave planning and scheduling problem. Whilst packing a collection of smaller items inside a single larger object represents the core of the WPAS problem, a production planning will require the packing of many objects. In the case of weaving machines, these objects represent the different sections in which textiles get produced. In a normal setting, the demand for a collection of textiles is so high that a single section will not suffice. In order to produce the entire demand, many objects will have to be packed with varying combinations of items. Besides needing to solve multiple SLOPP optimisation problems for a certain demand, known as MLOPP or *Multiple Large Object Placement Problem*, a production planning usually specifies multiple deadlines. These deadlines each define their own demand, combined with a due date. This further increases the number of SLOPP problems to solve and adds the extension of time management. An optimal solution should satisfy these deadlines as good as possible whilst minimising wasted material.

5.2.1 One-shot model: CP-OneShot

The one-shot model serves as a baseline implementation to support the extension of due dates. One global model gets defined containing all elements of the problem. As the name suggests, the one-shot model solves the problem in a single shot, all at once. This representation serves as a simple step in explaining how the due-date extension should be modelled. The next section will make improvements to increase the model's scalability. The impact of these improvements can subsequently be measured by comparing the two.

To ease the explanation of this model, the concept of *sections*, *pattern sections*, and *deadline sections* get introduced (*figure 5.13*). A ***section*** is a single object that gets packed with items, just like the SLOPP formulation. A ***pattern section*** is a collection of consecutive *sections* all packed in the exact same way with the exact same items. This unique packing that gets repeated is called a *pattern*. A ***deadline section*** is the collection of all *sections* produced or planned between two deadlines.

FIGURE 5.13: **Different types of sections**



A) Variable definitions

Ideally, every possible *section* should be an individual solution to a single SLOPP problem, allowing for each *section* to be uniquely packaged. For realistic production plannings with high item demands and spread-out deadlines, the large number of *sections* would unfortunately make the problem intractable. To resolve this issue, the notion of *patterns* can be employed. Instead of allowing each *section* to contain a unique packing, a limited number of reusable packings (called *patterns*) get created. The specification of a *section* then comes down to a reference towards one of the limited number of *patterns*.

Instead of encoding each *section*'s *pattern* choice individually, a per-deadline run-length encoding gets used to further improve the scalability. For the current problem definition, two solutions with the same choice of *patterns* in each *deadline section* but with a different order, are completely equivalent. *Sections* with the same *pattern* can thus be grouped in continuous sections and be encoded together using run-length encoding techniques. This significantly reduces the number of decision variables inside the model, at the cost of an assumption upon the solution space; *patterns* always get produced in continuous *pattern sections* with no breaks in between. Whilst this assumption makes the model less general, it can be argued that it matches well with a realistic production setting. Often times there is a certain cost associated with switching between the production of different *patterns*. For example, the reconfiguration of a machine, added storage costs, and so on. For many production planning problems, including weave planning, it is beneficial to create solutions with continuous *pattern sections*.

Within each *deadline section* $d \in D$ and for each *pattern* $p \in P$, the following variables get defined:

- $O_{d,p}$: the order number of *pattern p* in *deadline section d*
- $RL_{d,p}$: the run-length of ...
- $DB_{d,p}$: the delay before ...
- $DA_{d,p}$: the delay after ...

$O_{d,p}$ encodes the order between the *pattern sections* whilst $RL_{d,p}$ encodes how many times *pattern p* gets repeated inside the *deadline section.* $DB_{d,p}$ and $DA_{d,p}$ respectively encode a possible delay before and after *pattern p*, as a time buffer until the run-length of another pattern can occur. All are defined for each *deadline section d* separately. The run-length encoding can not cross the deadline boundaries. The domain of $O_{d,p}$ is spanned by $[0, NP-1]$, with NP the number of unique *patterns.* The domain of $RL_{d,p}$ gets defined by $[0, DC_d]$ with DC_d the capacity of *deadline section d.* DC_d can be defined as:

$$DC_d = \begin{cases} dd_d & if\ d = 0 \\ dd_d - dd_{d-1} & otherwise \end{cases}$$

With dd_d being the time of deadline d measured since the start of production, expressed in the number of *sections* that can be produced within that time. It can thus be interpreted as: between deadlines $d-1$ and d, there is enough time to produce DC_d *sections.* The domains for $DB_{d,p}$ and $DA_{d,p}$ can both be set to $\left[0, \max\limits_{d \in D}(dd_d) - 1\right]$, as in the most extreme case the machine waits for the entire planning duration only to produce a single packing at the very last moment in time.

As this model formulates an extension to the previously defined SLOPP models, all variables from those models must be brought over. In the case of this basic due-date model, where all NP patterns get solved at once, NP copies of all the selected SLOPP model's variables get created. Two auxiliary variables $PS_{d,p}$ and $PE_{d,p}$ can be introduced, signifying the start and end timepoints for each of the *patterns* in each of the *deadline sections.* Lastly, the auxiliary variable $A_{d,p}$ gets added to the model, signifying whether *pattern p* gets used in *deadline section d.*

B) Constraints

All the constraints can be separated into two groups. First, there are the constraints coming from NP copies of a SLOPP model formulation. The one-shot model tries to solve NP SLOPP-instances at once in order to find a collection of *patterns* with which to plan a production schedule. Second, there are the new constraints specific to the extension of due dates, linking the NP SLOPP-models together to model a single optimisation problem. The latter group will now be discussed.

Table 5.3: **One-shot due-date model variables**

Variable	Domain	Explanation
$O_{d,p}$	$[O, NP] \subset \mathbb{N}$	the order number of *pattern p* in *deadline section d*
$RL_{d,p}$	$[0, DC_d] \subset \mathbb{N}$	the run-length of *pattern p* in *deadline section d*
$DB_{d,p}$	$\left[0, \max\limits_{d\in D}(dd_d) - 1\right] \subset \mathbb{N}$	the delay before *pattern p* in *deadline section d*
$DA_{d,p}$	$\left[0, \max\limits_{d\in D}(dd_d) - 1\right] \subset \mathbb{N}$	the delay after *pattern p* in *deadline section d*

Pattern section start & end The following constraint formulates the definition of the auxiliary variables $PS_{d,p}$ and $PE_{d,p}$:

$$\begin{aligned} PS_{d,p} &= \begin{cases} (\sum_{\hat{d}=0}^{d-1} DC_{\hat{d}}) + DB_{d,p} & if\ O_{d,p} = 0 \\ \left(\sum_{\substack{\hat{p}\in P,\\ s.t. \hat{p}\neq p}} \delta_{p,\hat{p}} * (PE_{d,\hat{p}} + DA_{d,\hat{p}}) \right) + DB_{d,p} & otherwise \end{cases} \\ \delta_{p,\hat{p}} &\iff (O_{d,\hat{p}} = O_{d,p} - 1) \\ PE_{d,p} &= PS_{d,p} + (RL_{d,p} - 1) \end{aligned} \tag{5.35}$$

If a *pattern section* is in the first position of a *deadline section*, its start must be the start of that *deadline section* plus a possible delay $DB_{d,p}$. When it is not in the first position, it starts with a possible delay $DA_{d,\hat{p}} + DB_{d,p}$ after the ending of the previous *pattern section* ($O_{d,\hat{p}} = O_{d,p} - 1$). The ending of a *pattern section* is its start plus its run-length $RL_{d,p}$ minus 1 (the end is inclusive).

Bin active This constraint formulates the definition of the auxiliary variable $A_{d,p}$:

$$A_{d,p} \iff (RL_{d,p} \neq 0) \tag{5.36}$$

A *pattern section* can only be active if it has a nonzero run-length.

Bin order For the bin order variable to work in its intended way, a constraint should be included that implies its uniqueness within each *deadline section d*.

$$O_{d,p_1} \neq O_{d,p_2}\ , \forall d \in D, (p_1, p_2) \in P \times P,\ \text{c.t.}\ p_1 \neq p_2 \tag{5.37}$$

Within a constraint programming framework, this can more concretely be implemented using an *AllDifferent* global constraint:

$$\underset{p\in P}{AllDifferent}(O_{d,p})\ , \forall d \in D \tag{5.38}$$

Additionally, to reduce some degeneracy, the condition gets added that active *patterns* must always come before inactive ones:

$$(A_{d,p_1}\ \&\ !A_{d,p_2}) \Rightarrow (O_{d,p_1} < O_{d,p_2}) \tag{5.39}$$

Deadline capacity Between every pair of subsequent deadlines, there is a limited amount of time (expressed as a number of *sections*) to produce items. The created production planning must adhere to these limits.

$$PE_{d,p} + DA_{d,p} \leq \sum_{\hat{d}=0}^{d} DC_{\hat{d}} \tag{5.40}$$

Unique bin Each packing in the list of *patterns* should be unique. This uniqueness will be defined on the level of the included items. Different positioning of the same items will be seen as the same packing, as the problem only concerns the packing densities. As mentioned previously, all variables of the underlying SLOPP model get copied NP times, one copy for each *pattern*. In the case of the number of packed instances DC_t, this variable now becomes $DC_{t,p}$. It can now be used to express the uniqueness of a *pattern*:

$$(\forall t \in T : DC_{p_1,t} = DC_{p_2,t}) \implies (\sum_{t \in T} DC_{p_1,t} = 0) \tag{5.41}$$

This constraint expresses that when two *patterns* are identical, i.e., contain the exact same items, the *patterns* should be empty. Just stating that each *pattern* should be unique would not work when the number of unique *patterns* for a given problem instance is less than the number of *patterns* that the model tries to solve.

Symmetry breaking The model contains a degeneracy with regards to the numbering of the *patterns*. Multiple SLOPP instances get solved at once and each should have a unique solution and thus create a unique *pattern*. Switching around which instance finds which *pattern* will still result in the same solution. Some symmetry breaking can be achieved by assigning an order to the found *patterns*. With $A_p = \sum_t w_t * l_t * DC_{p,t}$ the packed surface area of a *pattern* p, the following condition can be added:

$$A_{p_1} \leq A_{p_2} \leq \cdots \leq A_{p_{NP}} \tag{5.42}$$

C) Objective

With the addition of time, demands and deadlines, the objective of the MLOPP with due dates becomes more complex. The problem gets converted from a single-objective optimisation problem to a multi-objective optimisation problem.

O_1: Waste The first objective is still the minimalization of wasted material but now across many objects / *sections*. The SLOPP objective (without the lower left preference) of each *pattern* can be scaled by its total run-length and afterwards be summed together, resulting in the total amount of wasted material across the entire planning.

$$O_1 = \sum_p \left[\left(W * L - \sum_t w_t * l_t * DC_{t,p} \right) * \left(\sum_d RL_{d,p} \right) \right] \tag{5.43}$$

O_2 & O_3**: Demand** The second and third objective both concern the fulfilment of the item demands. These fulfilments only get defined at the resolution of entire *deadline sections* based on how much each underlying *pattern* is repeated. On the one hand, enough items should be produced as to fulfil the demand of each deadline before its due date, keeping in mind what has already been fulfilled by previous deadlines. On the other hand, producing too much creates unnecessary waste because of the empty spaces between the items. If the customer does not want the surplus items, the overproduction too can be classified as waste. It is thus important to find the correct balance between overproduction and underproduction. This takes shape as the following two objectives:

$$\begin{aligned} O_2 &= \sum_t \left[\sum_{s=0}^{|D|-1} \max \left(\sum_{d=0}^{s} \left[\tilde{d}_{t,d} - \sum_p (DC_{t,p} * RL_{d,p}) \right], 0 \right) * v_t \right] \\ O_3 &= \sum_t \left[\max \left(\sum_{d=0}^{|D|-1} \left[\sum_p (DC_{t,p} * RL_{d,p}) - \tilde{d}_{t,d} \right], 0 \right) * v_t \right] \end{aligned} \tag{5.44}$$

Objective $\boldsymbol{O_2}$ formulates the underproduction penalty. Its expression will now be explained starting from the centre and going outwards. For each *deadline section*, the difference gets calculated between the demand $\tilde{d}_{t,d}$ and the number of items that have been produces in that section alone. With the summation over d, the cumulative difference across multiple sections gets determined. For example, with $s = 2$ the summation results in the cumulative difference between demand and production across the first three deadlines. It is necessary to use cumulative metrics as earlier *deadline section* can already start producing for later sections. As O_2 concerns underproduction, negative values get suppressed. The cumulative underproduction gets summed over all deadlines and over all item types to get the total objective.

Objective $\boldsymbol{O_3}$ has a very similar structure and formulates the overproduction penalty. The order of the subtraction has been reversed and the cumulative overproduction only gets determined for the last deadline. Only the item instances which are not in demand across the entire production result in actual overproduction. At all but the last deadline, overproduction is allowed as a means to plan ahead.

O_4**: Lower left** The lower left preference objective is once again introduced as to produce cleaner results. This can simply be formulated as the sum of the SLOPP model's lower-left formulation (if applicable) across all *patterns*.

Linearisation Now that all objectives have been defined, they can get linearised into a single objective that the solver can optimise. In the previous SLOPP models, a lexicographical ordering was utilised as to create an importance difference between the objectives. This will again be used to differentiate the lower-left preference objective from the rest. For the balancing of the remaining objectives, a normalised weighting will be used. The lexicographical ordering is not applicable as these sub-objectives are of equal order of importance and should be co-optimised. The normalised weighting allows for formulation of preference between the objectives. The total, linearised objective can be defined as:

$$O = \frac{1}{7} * (4 * O_1 + O_2 + 2 * O_3) * M + O_4 \tag{5.45}$$

The normalised weighting was chosen based on experimentational results and insights in the real world problem of production planning. The amount of wasted material directly influences the profit margins of the production and should be prioritised. Underproduction is more important than overproduction, as not meeting the customers' demands could result in additional losses besides the purely monetary. Depending on the application at hand, these weights can get rebalanced. The above formula thus serves as a demonstrative example.

5.2.2 Iterative model: CP-DLNS

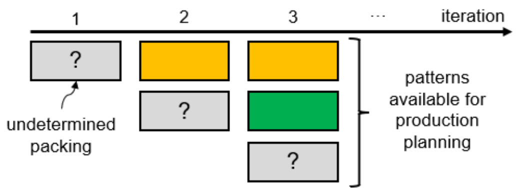
Once again, similar to the SLOPP chapter, a new model gets proposed as to improve upon the basic problem encoding of the one-shot model. Whilst some assumptions have been made regarding the uniqueness of each produced *section*, by leveraging a run-length encoding of a limited set of unique packings, the number of *patterns* to solve simultaneously is still rather high. The more unique *patterns* are requested, the higher quality the solution is likely to get. Unfortunately, the basic model quickly becomes untractable with increasing number of *patterns*.

Instead of trying to optimise all the *patterns* at once in a one-shot model, a promising alternative would be to use an iterative approach solving one *pattern* at a time. The modelled problem formulation becomes much smaller by only considering a small aspect of the complete problem at a time, greatly improving the performance of the solver. An additional benefit of solving the problem iteratively, is the ability to make dynamic decisions regarding the flow of the algorithm. One example is to dynamically decide how many *patterns* are needed in order to get a solution of sufficient quality. Once the usage of newer generated *patterns* falls below a certain threshold, indicating that they do not contribute much to the final result and that the best *patterns* have already been found, the algorithm can stop.

A) Relation to literature

The model can be seen as the combination of two techniques from the literature: Large Neighbourhood Search (LNS) and Delayed Column Generation (DCG). The technique utilised in this section sits somewhere in between and takes inspiration from both. It gets dubbed as the Delayed LNS (DLNS). In each iteration, the configuration of only a single new packing gets solved. Just like the basic model, besides the SLOPP model aspects for the creation of the pattern, additional production planning related modelling gets added. The DLNS model can be seen as solving the basic model with just a single pattern ($NP = 1$). Once a solution is found, the pattern gets fixed and a subsequent iteration searches for an improving solution within its neighbourhood. The neighbourhood gets defined by the destroyed production planning variables (LNS), together with newly introduced variables to allow for the creation of an additional *pattern* (DCG). Once again the packing of only a single pattern gets solved, but now in the context of the previously found packings and with a brand new production planning. This process is visualised in *figure 5.14*.

Figure 5.14: **LNSDCG**



These changes to the basic one-shot formulation should allow the model to be much more scalable, combined with all the other benefits of an iterating algorithm design (intermediate solutions, dynamic number of iterations, collection of easier-to-solve problems, interactivity, etc.).

B) Variables

The variables are exactly the same as those of the one-shot model (*subsection 5.2.1 Approach*).

C) Constraints

As the DLNS-model is actually an iterative solving of CP-OneShot model instances of increasing size with a singular new *pattern* to create, most of the model formulation is exactly the same. Some slight changes were made to improve the overall performance.

Unique pattern As the iterative approach does not necessarily have to fix the required number of *patterns* beforehand, its formulation for the required uniqueness of the patterns can be simplified. The most straightforward formulation can be used:

$$!\underset{t\in T}{All}(C_{t,p_1} = C_{t,p_2}) \tag{5.46}$$

This causes the model to be unsatisfiable if the iteration counter surpasses the number of unique solutions (which is only likely in very small-scaled problem instances). This is not a problem, as the iteration can detect this unsatisfiability and break out of its loop, preventing any further iterations and terminating with the last found satisfiable solution.

Useful pattern In order to prevent the model from generating useless *patterns*, a constraint gets added requiring the newly found *pattern* to be used at least once in the production planning.

$$\underset{d \in D}{Any}(A_{d,p}) \tag{5.47}$$

D) Objective

The objective is almost the same as for the one-shot model. The different iterations can be split into two groups: **I)** those focused on finding new *patterns* and **II)** those focused on improving the production planning. The former will not utilise the overproduction penalty as it has a tendency to result in less densely packed *patterns*. As each iteration of the model does not know of the later iterations that follow, which can still add additional *patterns*, it will try to get the most optimal solution with the *patterns* that it has so far. By taking overproduction into account, the balancing of all objectives can result in sparser *patterns*. The solution is to temporarily disable that sub-objective, solving a few iterations to get a collection of dense *patterns* and in the end run the latter iteration type. In this last iteration no new *patterns* get created, only the production planning gets solved with the inclusion of overproduction.

E) Algorithm

Now that the optimisation model has been explained, the encompassing iterative algorithm (*algorithm 4*) can be formulated. It simply runs a number of iterations creating new *patterns*, after which the entire collection gets used to solve for the final planning.

Whilst this algorithm utilises simple and fixed iterations, more dynamic techniques could be utilised with the proposed models. For example, changing the weights of the objective function linearisation in between the iterations to better guide the solver. It thus serves as a good basis for an interactive solving approach, where an expert planner can give feedback to further improve results. This thesis will not go into further detail.

5.3 Weaving model: CP-WPAS

In this section, a mathematical model for the creel will be created. A collection of constraints will be introduced to make the production planning adhere to the additional restrictions. Due to the extensible nature of the constraint programming formulations seen so far, these additions can simply be added on top of the current models without any changes to the previous expressions. The practical application of weaving serves as a good example to demonstrate the effectiveness and extensibility of the proposed CP-Anchor model for SLOPP. This might serve as an inspiration for future research in its application for solving other industrial use cases.

Algorithm 4 Iterative DLNS algorithm for the due-date extension to SLOPP

```
1:  Inputs:
        $M_{pattern}$, model for finding new pattern
        $M_{planning}$, model to solve planning with inclusion of
        overproduction penalty
2:  Output: Complete production planning

3:  procedure DLNS
4:      Initialise $P \leftarrow \emptyset$, list of found patterns
5:      for $i \leftarrow 1$ to $\#iterations$ do
6:          Solve $M_{pattern}$ for new pattern $p$ with $P$ as context
7:          if Not SAT then
8:              Break
9:          end if
10:         $P \leftarrow P \cup [p]$
11:     end for
12:     Solve $M_{planning}$ for final production planning with all patterns $P$
13: end procedure
```

5.3.1 Variables

The weaving model serves as an extension upon either a SLOPP model or a due-date extended model. All the variables from this foundation will be brought over to the CP-WPAS model. The new variables are related to the configuration of the creel. As mentioned before (*subsection 4.5.5 Problem statement*), there is the notion of *creel sections* (CS). These are continuous, multiple production segments in length and characterised by a singular *creel configuration*. The collection of all *creel sections* together will be called a *creel packing* (CP). A *creel packing* is characterised by the number of *creel sections* within it. This is represented by the variable $\#CS$ and bounded by $\#CS_{max}$. For every *creel section* $s \in CS$ it must be defined when it starts (CSS_s) and when it ends (CSE_s). An auxiliary variable CSA_s can once again be introduced to represent whether a certain *creel section* s is active or not.

Within each *creel section* there are *colour sections*. For each colour $c \in C$ a *colour section* can be defined. It is a group of intervals which define where that colour c is present across the width of the machine. Given all item types $t \in T$, a subset $t_c \in T_c$ can be defined with all items requiring colour c for their production. By taking the smallest item dimension $dim_{t_c,min}$ within T_c, it can be determined how many discrete intervals are maximally needed for that colour:

$$\#CI_{c,max} = \left\lfloor \frac{W+1}{dim_{t_c,min}+1} \right\rfloor \tag{5.48}$$

The variables for the $\#CI_{c,max}$ *colour intervals* for the *colour section* of c can now be defined. $\#CI_{s,c}$ represents how many intervals are actually in use and is upper bounded by $\#CI_{c,max}$. Auxiliary variable CIA_{s,c,j_c} represents if the j_c^{th} interval

($j_c \in J_c = [0, \#CI_{c,max} - 1]$) of colour c in section s is active. Each interval has a variable CIS_{s,c,j_c} and CIE_{s,c,j_c} representing respectively the start and the end of the interval across the width of the machine.

TABLE 5.4: **Weaving model variables**

Variable	Domain	Index	Explanation
$\#CS$	$[1, \#CS_{max}] \subset \mathbb{N}$		number of *creel sections*
CSS_s	$[0, dd_{max}] \subset \mathbb{N}$	$s \in CS$	start of *creel section*
CSE_s	$[0, dd_{max}] \subset \mathbb{N}$	$s \in CS$	end of *creel section*
CSA_s	$(0, 1) \subset \mathbb{B}$	$s \in CS$	if section s is active
$\#CI_{s,c}$	$[0, \#CI_{c,max}] \subset \mathbb{N}$	$s \in CS$, $c \in C$	number if *creel intervals*
CIA_{s,c,j_c}	$(0, 1) \subset \mathbb{B}$	$s \in CS$, $c \in C, j_c \in J_c$	if interval j_c is active
CIS_{s,c,j_c}	$[0, W - \min\limits_{t \in T_c}(\min(w_t, l_t))] \subset \mathbb{N}$	$s \in CS$, $c \in C, j_c \in J_c$	start of interval j_c
CIE_{s,c,j_c}	$[\min\limits_{t \in T_c}(\min(w_t, l_t)), W] \subset \mathbb{N}$	$s \in CS$, $c \in C, j_c \in J_c$	end of interval j_c

5.3.2 Constraints

The constraints can be subdivided into those modelling **A)** the creel, which consists of **I)** the *colour sections*, **II)** the *creel sections* and **III)** the *creel packing* and **B)** the interaction between this extension and the underlying (due-date extended) SLOPP model.

A) Creel

Colour section The definition of the auxiliary variable variable CIA_{s,c,j_c} can be defined as follows:

$$CIA_{s,c,j_c} \iff (j_c < \#CI_{s,c}) \tag{5.49}$$

It must first be specified that the *colour intervals* of each colour c can not overlap, as they could otherwise be merged together as a single, continuous interval. They must all be disjoint. Combined with the symmetry-breaking assumption that the intervals j are defined according to their spatial order, the following constraint can be specified:

$$CIA_{s,c,j_c+1} \implies (CIE_{s,c,j_c} < CIS_{s,c,j_c+1}) \tag{5.50}$$

This enforces that when an interval is active, it must come before an inactive one and its start must come after the previous end. Next, the width of all sections combined can not be greater than the width W of the machine. The last active section can not go beyond W:

$$CSEs, c, (\#CI_{s,c} - 1) \leq W \tag{5.51}$$

Creel section Once again the definition of the auxiliary "active" variable CSA_s can be formulated:

$$CSA_s \iff (s < \#CS) \tag{5.52}$$

The limitation to the number of colours at each dent position is formulated as:

$$\begin{aligned} CIA_{s,c,j_c} &\implies \left(\sum_{\substack{\hat{c}\in C,\\ s.t.\hat{c}\neq c}} IsHere(s,\hat{c},CIS_{s,c,j_c}) \right) < CC_{max} \\ IsHere(s,\hat{c},x) &= \underset{\hat{j}_{\hat{c}}\in \hat{J}_{\hat{c}}}{Any} \left[(CIS_{s,\hat{c},\hat{j}_{\hat{c}}} \leq x) \;\&\; (x < CIE_{s,\hat{c},\hat{j}_{\hat{c}}}) \;\&\; CIA_{s,\hat{c},\hat{j}_{\hat{c}}} \right] \end{aligned} \tag{5.53}$$

$IsHere(s,\hat{c},x)$ returns a Boolean value indicating whether x lies within the *colour section* of $\hat{c}$ inside *creel section* s, thus within one of its *colour intervals* $[CIS_{s,\hat{c},\hat{j}_{\hat{c}}}, CIE_{s,\hat{c},\hat{j}_{\hat{c}}}]$. By summing over all the colours, the total number of colours present at a certain position can be determined. It is sufficient to check the condition at the start of each *colour interval* when trying to limit the number of colours at each dent position. If at a certain dent position the maximum is succeeded and the condition is not satisfied, it is trivial to see that this unsatisfiable situation must be within a continuous region with a start and an end. The start is the location where the colour threshold gets surpassed for the first time. As an increase within the number of colours can only occur at the presence of the start of a new *colour interval*, the region start must coincide with one. It is sufficient to only check the start positions of these regions and thus the beginning of the colour intervals. If the condition is not satisfied at a certain position x, the condition is necessarily also not satisfied at the start of the last added *colour interval* when going from left to right. If supported, the "*Cumulative (Scheduling)*" global constraint could be used as an alternative formulation.

Finally, the end of a creel section must always come after its start:

$$CSE_s \geq CSS_s \tag{5.54}$$

Creel packing The creel sections within the packing must not overlap. Combined with basic spatial-order-based symmetry breaking, the following set of constraints can be added:

$$CSA_{s+1} \implies CSS_{s+1} = (CSE_s + 1 + CSP) \tag{5.55}$$

B) Interaction

The interaction between the creel representation and the underlying model can also be further subdivided. For the extension upon SLOPP, adapted formulations for both the CP-Guillotine and the CP-Anchor model will be presented. Additional constraints for the extension upon the due-date model will follow suit.

CP-Guillotine Somehow the positions of the packed item instances must be linked with the positions of the different colour sections. With a small trick, this can be formulated on the level of *strips* instead of the individual *cuts*, reducing the required number of constraints. The presence of an item $t_c \in T_c$ in a certain *strip*, together with the location of that *strip*, is enough to derive the colour location for c along the width of the machine. The creel only depends on the width-wise positioning of the items, not on the length-wise position. Since the width-wise position is identical for all items within a *strip*, just the knowledge that any instance of an item is present suffices. This is implemented in *algorithm 5*, where the position of each *strip* is iteratively constructed and the linking constraints are formulated. If an item is present in a certain *strip*, *IsHere2* (*algorithm 6*) will handle the restrictions for the creel configuration. It formulates that for a given interval $[x_1, x_2]$ (which will be set to the *strip*'s position), there must be a *colour interval* of c that encompasses it.

Algorithm 5 Function to link creel formulation to CP-Guillotine

Input: creel section: s

Output: interaction constraint

```
1:  function InteractionGuillotine($s$)
2:      Initialise $cc = \emptyset$
3:      $pattern\_height \leftarrow 0$
4:      for each $p \in P$ do
5:          $strip\_width \leftarrow 0$
6:          for each $a \in A$ do
7:              for each $t \in T$ do
8:                  for each $c \in C_t$ do
9:                      $cc = cc \ \cup \ \Big[ \underset{b}{Any}\{\sigma_{p,a,b,t}\} \implies$
10:                             $IsHere2(s, c, strip\_width, strip\_width + w_t) \Big]$
11:                 end for
12:             end for
13:             $strip\_width \mathrel{+}= \sum_{w \in W} (\gamma_{p,a,w} * w)$
14:         end for
15:         $pattern\_height \mathrel{+}= pl_p$
16:     end for
17:     return $All(cc)$
18: end function
```

CP-Anchor For CP-Anchor, a similar approach can be taken (*algorithm 7*). Thanks to the anchor grid structure, item instances are bound to specific cells limiting their range of movement. For each length-wise column in this grid, similar to the *strips*, the presence of only a single instance of item t_c already reveals a lot of information regarding the location of colour c. Taking the min and max positions

Algorithm 6 Decomposable expression checking if a colour section is present in a certain interval

```
1: Inputs:
       creel section: $s$
       colour: $c$
       interval start: $x_1$
       interval end: $x_2$
2: Output: decomposed IsHere2 constraint

3: function IsHere2($s, c, x_1, x_2$)
4:     return $\underset{j_c \in J_c}{Any}\left\{(CIS_{s,c,j_c} \le x_1) \;\&\; (x_2 \le CIEs, c, j_c) \;\&\; CIA_{s,c,j_c}\right\}$
5: end function
```

of all active instances of type t_c in that column results in an interval where c must be present. This interval is necessarily continuous since the cells do not allow for width-wise movement greater than the width of the item.

Algorithm 7 Function to link creel formulation to CP-Anchor

```
Input: creel section: $s$
Output: interaction constraint

1: function InteractionAnchor($s$)
2:     $cc \leftarrow \emptyset$
3:     for each $t \in T$ do
4:         for each $u_t \leftarrow 0$ to $u_{t,max}$ do
5:             $ccc \leftarrow \emptyset$
6:             for each $c \in C_t$, $C_t$ = colours of item type $t$ do
7:                 $ccc = ccc \;\cup$
8:                     $\left[IsHere2(s, c, \underset{v_t \in [0, v_{t,max}]}{min}(x_{t,(u_t,v_t)}), \underset{v_t \in [0, v_{t,max}]}{max}(x_{t,(u_t,v_t)}) + w_t)\right]$
9:             end for
10:            $cc = cc \;\cup\; \left[\left(\underset{v_t \in [0, v_{t,max}]}{Any} a_{t,(u_t,v_t)}\right) \implies All(ccc)\right]$
11:        end for
12:    end for
13:    return $All(cc)$
14: end function
```

Due-date extension The last additions to the model are the interaction constraints when extending upon a due-date formulation. A link must be made between the run-length encodings of both the pattern sections and the creel sections. Each pattern section must be located inside a creel section that supports all colours of that pattern. A creel section can hold multiple pattern sections to minimise the number of creel reconfigurations. *Algorithm 8* iterates over all deadline sections d, all patterns p and all creel sections s. cc are the interaction constraint for the underlying SLOPP model. The procedure subsequently formulates that when s is active and when within deadline section d, the pattern section of p lies within s, the interaction constraints cc between s and that pattern section should be added to the model Y. Line 8 also stipulates that every pattern section should always lie within a creel section. *Algorithm 9* implements the "*lies within*" check between pattern sections and creel sections.

Algorithm 8 Formulate interaction with due-date extended model

Input: model Y
Output: model Y with added interaction constraints

```
1: procedure InteractionDueDate
2:     for each $d \in D$ do
3:         for each $p \in P$ do
4:             for each $s \in S$ do
5:                 $cc \leftarrow Interaction < Guillotine|Anchor > (s)$
6:                 $Y \leftarrow Y \cup [CSA_s \implies (LiesWithinCreelSection(s,d,p) \implies cc)]$
7:             end for
8:             $Y \leftarrow Y \cup [A_{d,p} \implies \underset{s \in S}{Any}(CSA_s \ \& \ LiesWithinCreelSection(s,d,p))]$
9:         end for
10:    end for
11: end procedure
```

Algorithm 9 Decomposable expression linking continuous pattern sections with creel sections

```
1: Inputs:
       creel section: $s$
       deadline: $d$
       pattern: $p$
2: Output: decomposed LiesWithinCreelSection constraint

3: function LiesWithinCreelSection($s, d, p$)
4:     return $(CSS_s \leq PS_{d,p}) \ \& \ (PE_{d,p} \leq CSE_s)$
5: end function
```

5.3.3 Objective

The weaving extension only adds variables and constraints to the model as to formulate additional restrictions on the packings and plannings. The optimisation objective stays exactly the same as the underlying model, either *section 5.1 Approach* or *section 5.2 Approach*.

Chapter 6

Experiments

Several experiments have been performed to evaluate the performance and other characteristics of the proposed models. These experiments utilise newly introduced benchmark suites for the ***Weave Planning and Scheduling*** problem and its subproblems, serving as a reference for future research. Each sub-problem of WPAS, with increasing layers of extensions built upon SLOPP, is analysed separately. The new formulations of this thesis are compared against baseline implementations and a model from the literature, assessing how well they improve the scalability whilst preserving generality. The influence of different factors related to the problem instances is evaluated, like the dimensions of the object and the number of smaller items. From these results, the contributions of this thesis can be substantiated.

6.1 Implementation

All mathematical models have been implemented using constraint programming. *Python 3.10* was used as the programming language together with the *CPMpy* library. As mentioned on the official documentation webpage[1], *CPMpy* is a "*Constraint Programming and Modeling library in Python, based on numpy, with direct solver access*" [28]. It follows the declarative programming paradigm, allowing for optimisation problems to be represented as a collection of discrete decision variables, constraints, and an objective function. These models can then automatically be translated to the input for state-of-the-art general-purpose solvers which then handle the solving to optimality.

For the upcoming experiments, the following setting were used:

- *OR-Tools* solver
- default CPMpy parameters
- 4 minutes solving timeout

The open-source and award-winning *OR-Tools* optimisation suite [45] was used. All solver parameters have been left at their default values as set by the *CPMpy*

[1]https://cpmpy.readthedocs.io/en/latest/

library. Each problem instance received a maximum run-time of 240 seconds (4 minutes) before aborting the solver and continuing with the best solution found at that point (which may not be globally optimal). All experiments have been conducted on a desktop PC running Debian 5.10.70 with an Intel® core i3-10100F processor at 3.60GHz and 16 GB of ram. All four available cores were used.

6.2 Experiment questions

The experiments are intended to evaluate the performance and other characteristics of the newly proposed models of this thesis; CP-Anchor, CP-DLNS, and CP-WPAS. The purpose of this evaluation is to formulate answers to the the following experiment questions:

[**Q1**] What are the model size implications of the different SLOPP formulations?

[**Q2**] How do the different SLOPP models perform under various problem instance parameters?

[**Q3**] How does an iterative approach improve the results, compared to a one-shot model, for the due-date extension?

[**Q4**] How does an exemplar problem extension, like the creel, impact the performance of the SLOPP formulations?

6.3 Problem instances

To evaluate and compare the effectiveness and performance of the models under various conditions, a diverse collection of problem instances at multiple difficulty scales is needed. Together, they will compose a benchmark suite. Many of such suites for the SLOPP optimisation problem exist in the literature, like Berkey & Wang (1987) [11], Martello & Vigo (1998) [52] and the collection of OR-Library [10, 9]. Unfortunately, they are not designed for the High Multiplicity case. Due to the lack of appropriate problem instances, this paper proposes a new benchmark suite. As an unbiased basis to generate these instances, the ***SLOPPGEN*** problem generator [56] has been utilised. It allows for the generation of problem data for R_2D_SLOPP and enables a more systematic testing approach. The generator is configurable, allowing for the creation of a large number of problem instances with the required properties. The generated problem instances contain all inputs for SLOPP. For the extensions towards WPAS, additional parameters get added to each problem instance.

6.3.1 SLOPP benchmark

For the proposed SLOPP benchmark suite, the following parameters for *SLOPPGEN* have been selected:

Number of small items This parameter is quite important for the SLOPP optimisation problem, as it is often used as a measure of problem size. It can be expected that a placement problem becomes more difficult to solve with a larger number of available item types to pack. The more types, the more cutting patterns are possible, and the larger the search space. On the other hand, more item types has a positive influence on the solution quality. Having more shapes available allows for denser packings with less wasted space (when sufficient time is given to solve the more complex optimisation problem). To analyse a variety of problem sizes, the considered number of items is $n_{item} \in \{4, 7, 10\}$.

Dimensions of larger object The large object is the rectangle within which the different smaller items have to be optimally positioned. It is also a measure for the problem size or complexity. In real settings, this object often represents a larger piece of material to cut the smaller items from or a larger container in which to pack the items. As mentioned within the accompanying paper of *SLOPPGEN*, the problems can be divided into two classes depending on the shape of the object; quadratic problems and rectangular problems [56]. ***Quadratic problems*** use equal values for each dimension, resulting in square objects for the two-dimensional case. These problems test the different models in their handling of large open spaces. ***Rectangular problems*** use a rectangular object shape, allowing for long and narrow problem instances. These problems often occur in industrial applications, where smaller items have to be cut from a larger, rectangular sheet of material. This is also the case for the WPAS problem, where many textiles get produced together in large rectangular sections after which they get cut out. The absolute size of the larger object is not that important, but rather its aspect ratio and its size relative to the smaller items. All dimensions can be multiplied by the same constant factor without changing the optimalisation problem. As the implementations using constraint programming make use of discrete decision variables, an integer scale should be used. The chosen problem instances include an object size of 100×100, 200×200, and 300×300 for the *quadratic problems*. For the *rectangular problems*, to allow for comparisons, objects with equivalent surface area have been chosen. All variants of the object's dimensions are listed in *table 6.1*.

TABLE 6.1: SLOPPGEN problem object sizes

100^2	100×100		
200^2	200×200	100×400	
300^2	300×300	200×450	100×900

Dimensions of small items As previously mentioned, only the relative size of the smaller items is important for SLOPP (besides for the integer discretisation). In the *SLOPPGEN* generator, the item size gets defined by its relative area with respect to the object. Using a lower and an upper bound, item sizes can be sampled to create problem instances. As to create similarly sized items, regardless of the increasing object size of the benchmark instances, the bounds have been proportionally adjusted. For an object with an area of 100^2, the relative size range is defined as $[0.08, 0.5]$. Any increase in area is compensated by an inverse scaling on the bounds, i.e., $[\frac{1}{b} * 0.08, \frac{1}{b} * 0.5]$. These adjustments allow for a better analysis of the impact of object size on the model's performance. *Table 6.2* gives an overview.

Table 6.2: SLOPPGEN problem item sizes

Problem size	Item relative area range
100^2	$[0.08, 0.5]$
200^2	$[0.02, 0.125]$
300^2	$[0.0089, 0.0556]$

Given a chosen area (relative item size), a random aspect ratio gets selected. The algorithm includes a safeguard to prevent the generation of degenerate items, i.e., long and narrow slithers. The original *SLOPPGEN* implementation imposes a range of $[0.1, 0.9]$ on the aspect ratio. This still produces rather narrow items. In order to make the benchmark more representative for the later extensions of WPAS, the range was further restricted to $[0.3, 0.7]$.

Defects The *SLOPPGEN* generator allows for the creation of problem instances for the extended problem with defects in its object. These are regions inside the object where no small item can be positioned. As WPAS does not consider defects, these parameters were disabled.

Benchmark suite Using all the above-mentioned parameters, the benchmark suite has been created by repeated calls to the *SLOPPGEN* generator. A total of 18 problem configurations (6 object sizes, 3 item type amounts) can be identified. For each configuration, 10 instances were created. The total suite thus consists of 180 problem instances. Whilst this suite only contains the parameters for SLOPP, it will serve as the basis for the suites of the different extensions. Each extension adds certain parameters to the instances.

Table 6.3: SLOPP benchmark parameters

Parameter	Values
# small items	$\{4, 7, 10\}$
object sizes	$\{$**100^2**: $[100 \times 100]$, **200^2**: $[200 \times 200, 100 \times 400]$, **$300^2$**: $[300 \times 300, 200 \times 450, 100 \times 900]\}$
item relate size	$\{$**100^2**: [0.08, 0.5], **200^2**: [0.02, 0.125], **300^2**: [0.0089, 0.0556]$\}$
item aspect ratio	[0.3, 0.7]
defects	disabled
timeout	4 minutes

6.3.2 Due-dates benchmark

The problem instances for the due-date extension will be derived from a subset of the SLOPP instances. n_{item} is still $\{4, 7, 10\}$ and 10 instances are still created per configuration, but only the rectangular object dimension of 100×400 is selected. Since the impact of the object shape will already be studied using the SLOPP benchmark suite, it is sufficient to select a single object shape. As previously mentioned, rectangular problems are more appropriate for the process of weaving. The parameters that are added on top are the formulation of the MPS: due dates dd_d for each deadline $d \in D$, together with the demand $\tilde{d}_{t,d}$ for each item type t. The number of deadlines is selected from $\{1, 3, 6\}$. As to create an achievable MPS, the demands should be realistic with respect to the due dates. To this end, an additional CP model has been created to generate realistic problem instances. The NP or number of *patterns* for the MLOPP algorithms has been set to 5. With 4 minutes per pattern, each model received 20 minutes to find the most optimal solution.

Benchmark suite Using all the above-mentioned parameters, the benchmark suite has been created by making small additions to a subset of the SLOPP benchmark suite. A total of 9 problem configurations (1 object size, 3 item type amounts, 3 deadline options) can be identified. For each configuration, 10 instances were created. The total suite thus consists of 90 problem instances.

Table 6.4: Due-date benchmark parameters

Parameter	Values
# small items	$\{4, 7, 10\}$
object sizes	100×400
item relate size	[0.02, 0.125]
item aspect ratio	[0.3, 0.7]
defects	disabled
# deadlines	$\{1, 3, 6\}$
# patterns	5
timeout	20 minutes (4 per pattern)

6.3.3 WPAS benchmark

Similarly to the model formulations, the benchmark suite can be formulated as an extension to both the SLOPP suite and the due-date suite. The parameters that are added concern the colour and creel information: the creel colour capacity CS_{max} is selected from $\{1, 2, 3, 4, 5\}$, the item types t are assigned one of the five colours in a round-robin fashion, and the creel switch penalty CSP is set to 10.

Benchmark suite Using all the above-mentioned parameters, the benchmark suite has been created by making small additions to a subset of the SLOPP or due-date benchmark suite. A total of 15 problem configurations (1 object size, 3 item type amounts, 5 creel capacities) can be identified. For each configuration, 10 instances were created. The total suite thus consists of 150 problem instances.

Table 6.5: WPAS benchmark parameters

Parameter	Values
# small items	$\{4, 7, 10\}$
object sizes	100×400
item relate size	[0.02, 0.125]
item aspect ratio	[0.3, 0.7]
defects	disabled
# deadlines	$\{1, 3, 6\}$
# patterns	5
creel colour capacity	$\{1, 2, 3, 4, 5\}$
timeout	20 minutes (4 per pattern)

6.4 Measurement methodology

To compare the different representations, the performance profiles from Dolan & Moré are utilised [22]. These profiles define for each model a distribution function concerning a selected performance metric. They serve as a tool for comparing optimisation software, including different model representations of a singular optimisation problem. This allows for the different model representations to be ranked against each other based on the chosen metric.

The benchmark results are collected by running a set of solvers S (or in this case models) on a set of problem instances P and recording certain performance metrics. These metrics cannot simply be averaged in order to get a global score of each solver, as not all benchmark instances are of equal difficulty. The more difficult benchmark instances will tend to dominate the metric. Using global scoring techniques, the models can only be compared on a per-problem instance basis.

Instead, a ***performance ratio*** gets determined. For each problem $p \in P$ and solver $s \in S$, a performance metric $t_{p,s}$ can be measured. The performance ratio $r_{p,s}$ is defined as:

$$r_{p,s} = \frac{t_{p,s}}{\min\limits_{s \in S}(t_{p,s})} \tag{6.1}$$

When a solver failed to find a solution for a certain problem instance p, its ratio gets fixed to a value r_M with $r_M \geq r_{p,s}$ for any choice of p and s.

The ***performance profile*** for a solver s assesses its overall performance with respect to the other solvers via a cumulative distribution function:

$$\rho_s(\tau) = \frac{1}{n_p} * |r_{p,s} : r_{p,s} \leq \tau| \tag{6.2}$$

where τ is the closeness factor to the best possible performance ratio.

Whilst the cumulative distribution function based on performance ratios works well for minimisation metrics (performance metrics for which a more optimal solution has a lower value), it does not work well in the case of maximisation. Reformulating the performance ratio as:

$$r_{p,s} = \frac{t_{p,s}}{\max\limits_{s \in S}(t_{p,s})} \tag{6.3}$$

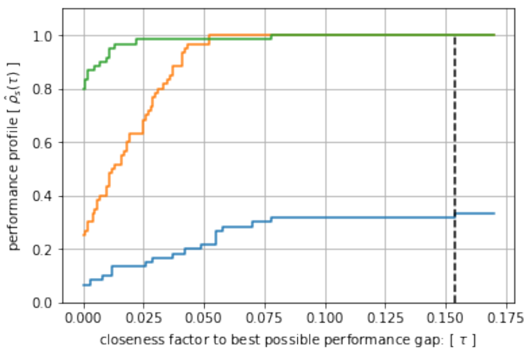
results in a range $[0, 1]$ instead of the previous $[1, +\infty)$. The worst solutions of each problem will be closer to 0, whilst the more optimal ones closer to 1. This new ordering of the ratio values stands in contrast with the minimisation metric, where the more optimal solutions are located closer to 1 and the less optimal ones in the direction of $+\infty$. As the calculation of the cumulative distribution depends on this ordering, a small change to the ratio has to be made. As the range is limited to $[0, 1]$, taking $1 - r_{p,s}$ restores the order.

$$1 - r_{p,s} = 1 - \frac{t_{p,s}}{\max\limits_{s}(t_{p,s})} = \frac{\max\limits_{s}(t_{p,s}) - t_{p,s}}{\max\limits_{s}(t_{p,s})} = \frac{\hat{t}_p - t_{p,s}}{\hat{t}_p} = GAP_{p,s} \tag{6.4}$$

This new ratio often gets referred to as the ***performance gap***, a relative difference between the best found solution and the current solution. The performance profiles based on this ratio will be identified using $\hat{\rho}_s(\tau)$.

Figure 6.1 shows an example of a performance profile. Taking a single vertical slice from the graph at a certain τ gives the cumulative probability of each model for producing a gap or ratio of at most τ. Taking a horizontal slice from the graph at a certain ρ, gives for each model the upper limit for which can be guaranteed that with a probability of ρ the model will produce a gap or ratio under that limit. Both are for a random problem instance sampled from the same population of problem instances.

FIGURE 6.1: **Example performance profile**



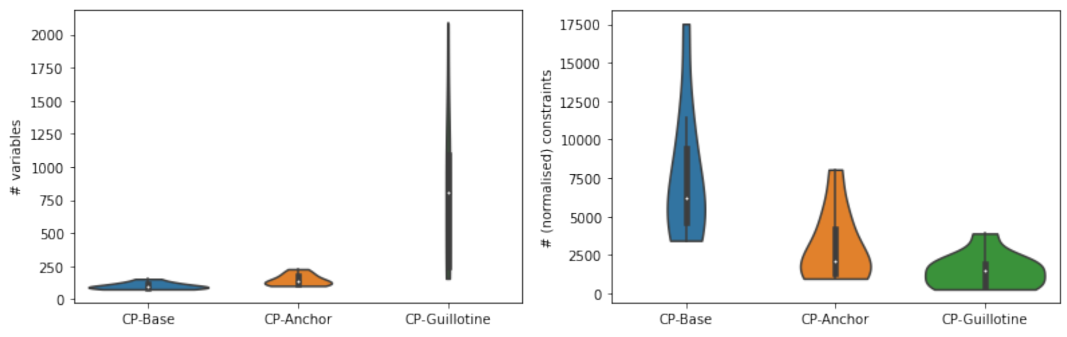
6.5 Results and discussion

Now that every organisational aspect regarding the experiments has been explained, such as the formulation of the experiment questions, the creation of the benchmark suites, and the establishment of the measurement methodology, the result can get analysed. They should support the previously made claims throughout this thesis regarding the position of the newly proposed models with respect to the baselines and the model from the literature.

6.5.1 [Q1] SLOPP model size

section 5.1 Approach introduced the different models for the SLOPP sub-problem; CP-Base, CP-Guillotine, and CP-Anchor. Each of these models utilise a different representation technique for the optimisation problem, resulting in different model characteristics. One of these characteristics is the model size. Both the number of variables and the number of constraints can be compared. The measurement for the variables is straightforward. For the constraints the "flat normalisation form" is utilised, as provided by CPMpy's "*flatten_model*" transformation. Without this transformation, the metric would only measure how compact the model's implementation was formulated. A one-liner consisting of a single *And* across all constraints of the entire model would otherwise count as a single constraint.

Figure 6.2: **SLOPP model size**



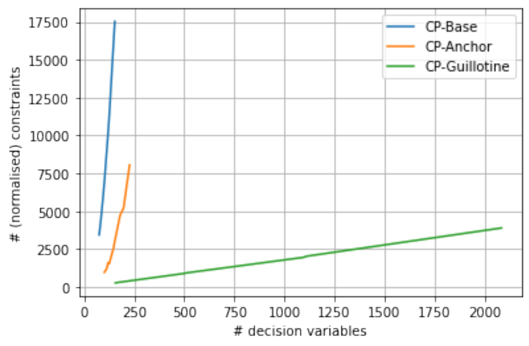
(A) Variables (B) Constraints

The measurements have been performed on the benchmark instances of size 100^2 with 7 item types. The result of these measurements can be seen in *figure 6.2* for both the variables and the constraints. *Figure 6.3* visualises the combination of both metrics for the selected problem instances.

CP-Base (*subsection 5.1.1 Approach*) was introduced as a baseline. A naive implementation of SLOPP, serving as a reference for later models. It was mentioned that this model receives its unscalability from its exponential number of constraints; non-overlap has to be formulated between every pair of item instances.

CP-Guillotine (*subsection 5.1.2 Approach*) takes an alternative approach to SLOPP using a guillotine representation. Non-overlap has become an implicit

FIGURE 6.3: **Constraints vs. Variables**



(A) Variables

property of the model, reducing the number of constraints. Whilst the model size has decreased in that regard, the number of variables has significantly increased. For the basic SLOPP problem formulation this is a worthwhile trade-off. Upcoming experiments will demonstrate its improved performance. Unfortunately, besides not being a general model, the large number of variables restricts the scalability for any further extension. New additions to the model associated with the items will have to be formulated over all these variables, resulting in a large increase in the number of constraints. When looking back at the critique of Junqueira et al. (2012) mentioned in *section 3.3 Related Literature*, this is an example of a model that only works for basic toy problems and can not be extended to more realistic settings. These scalability issues will be demonstrated in [Q4].

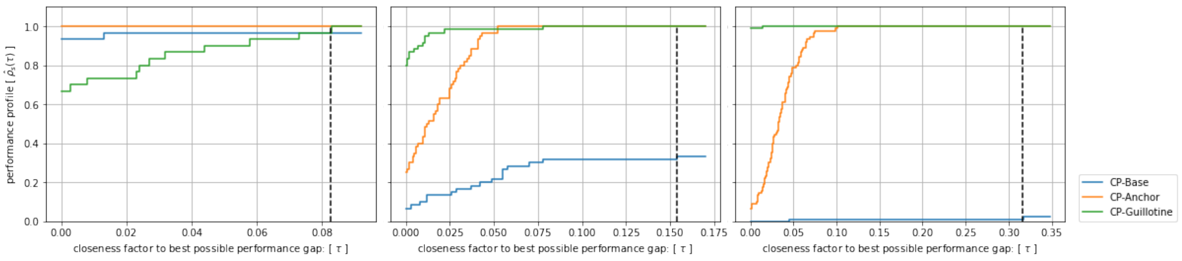
CP-Anchor (*subsection 5.1.3 Approach*) is the newly proposed model of this thesis. It utilises the same set-of-points formulation and similar non-overlap constraints as CP-Base. With the help of the anchor structure, the constraints can be formulated more selectively. There is a minimal increase in the number of variables and the explosive nature of the non-overlap constraints has been resolved. The combination of both leads to a formulation that is general, scalable, and can support further extensions.

6.5.2 [Q2] Evaluation of SLOPP models

In the case of SLOPP, the main objective when placing items into the larger object is output maximisation. As the standard problem formulation does not specify a preference amongst the items, each item gets as value v_t its surface area $w_t * l_t$. More specifically, the objective comes down to the maximisation of the packing density. The packing density of the different SLOPP models will be discussed under various problem instance parameters, analysing their impact on the models' performance.

Object size Three object sizes have been defined: 100^2, 200^2, and 300^2. For each size respectively, *figure 6.4* shows the performance profiles for all models.

Figure 6.4: **SLOPP density performance profile w.r.t. object size**



(a) 100^2 (b) 200^2 (c) 300^2

At $\mathbf{100^2}$, the problem instances are still rather small in size. The upper bound on the solve time of 4 minutes is not yet a limiting factor. The two most general models, CP-Base and CP-Anchor, are outperforming CP-Guillotine. Denser packings are possible thanks to the models' generality and larger solution space.

At $\boldsymbol{\tau} = \mathbf{0}$, the probability $\hat{\rho}$ that a model produces the best packing seen so far (with respect to the given time limit) can be read. CP-Guillotine can only produce the best packing in 67% of the problem instances. This is a direct consequence of its two-stage guillotine cut assumption on the solution space. CP-Anchor, being a general model, can find all optimal solutions. CP-Base shows a probability of 93%, even though it is also a general model and should have the exact same solution space as CP-Anchor. 93% is $\frac{2}{30}^{th}$ short of a perfect score, indicating that the model was not able to find the optimal solution to two problem instance within the 4-minute time window. For one problem instance it was not even able to find a single solution, causing its profile to never reach 100%. This is already the first indication that CP-Anchor is more scalable than CP-Base.

At the other end of the graph, at $\boldsymbol{\tau} = \mathbf{0.083}$, the largest relative deviation from the optimal packing can be observed. Whilst CP-Guillotine never performs better than CP-Anchor (as it has 100% probability throughout), its maximal relative deviation from the best found solution is at most 0.083.

At $\mathbf{200^2}$, the performance graph significantly changes and takes on the shape which will be prevalent for the remainder of the benchmarks. The complexity of the problem instances has increased enough to cause the 4-minute limit to also become a bottleneck for CP-Anchor. As CP-Anchor is more general, every solution of CP-Guillotine is also a solution of CP-Anchor. Theoretically, the performance profile of CP-Anchor should always be at 100%. Due to its larger model size with respect to CP-Guillotine, it was not able to find the same solution quality within the given time. This behaviour is to be expected for the two general models, as they define a larger search space. Given more time the results should improve.

From this graph, many of the previously made claims about the relative effectiveness and performance of CP-Anchor are visible. Even tough CP-Anchor's probability of a zero gap lies at a mere 25%, it still manages to produce some optimal solutions which the guillotine model can not approximate (its zero-gap does not lie at 100%). It takes a gap of 0.072 before CP-Guillotine reaches a cumulative probability of 100%. CP-Anchor's maximal gap is 0.059.

When some gap on the solution is allowed, it is clearly visible that CP-Anchor quickly catches up and approximates the CP-Guillotine solutions quite well. It reaches a cumulative probability of 90% at a gap of just 0.036. This is a fine demonstration of how, whilst CP-Guillotine is a simpler representation to solve and is thus less bottlenecked by the time limit, the general CP-Anchor with its novel anchor pattern representation is able to approximate its performance quite well. This is a very important insight, as the generality of the model does not pose any restrictions on future extensions, which CP-Guillotine does suffer from. When the assumptions on the solution space do become a problem for the extensions, CP-Anchor serves as a promising and well-performing alternative.

When comparing with CP-Base, the improvements due to the anchor encoding are now clearly visible. The zero-gap probability has more than doubled from 6.6% to 25%. The cumulative probability rises much faster with increasing gap size. More noticeably, CP-Base was only able to solve $\frac{1}{3}^{rd}$ of all the problem instances within the 4 minute limit. For $\frac{2}{3}^{rds}$ not even a single satisfiable solution was found.

At $\mathbf{300^2}$ the complexity of the optimisation problem has increased even more. The same insights can be gained as before, but now with even more extreme differences. CP-Base can only solve a mere 3.3% of the problem instances. CP-Guillotine has a zero-gap probability of close to 100%, indicating that the model almost always outperforms the others. Whilst the complexity of the problem has dramatically increased going from an object size of 200^2 to 300^2, the performance difference between CP-Anchor and CP-Guillotine did not change that drastically. CP-Anchor still approximates CP-Guillotine, but now with an increased gap size. This can not be said for CP-Base. This once again demonstrates the improved scalability of the anchor pattern representation. Whilst its approximation of CP-Guillotine has slightly degraded, the model still has the benefit of being general, allowing for any extension being formulated on top.

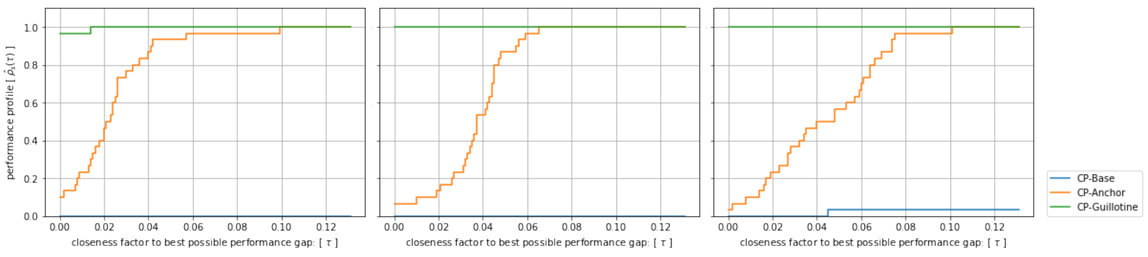
Object aspect ratio An additional aspect that has to be analysed is the impact of the object's aspect ratio. Whilst the results seen so far were categorised in different problem complexities in terms of surface area, the shape of that surface has a different effect on each of the models. The problem size of 300^2 can be divided into 300×300, 200×450, and 100×900 to represent both quadratic and rectangular problem instances. As the surface area stays the same, each change in one dimension causes a disproportionate and nonlinear change in the other.

In order to take full advantage of the limited two-stage cuts of CP-guillotine, the $zero^{th}$ cut of the different patterns should be parallel to the shortest edge (width) whilst positioned along the longest edge (length). The number of patterns $\#P$ (or NP) equals how many times the shortest item dimension fits along the object's length. Inside each pattern, multiple strips can be positioned. The upper limit $\#A$

equals the number of times the shortest item dimension fits along the object's width. Lastly, within each strip there can be at most $\#B$ cuts with $\#B$ once again the repeats along the object's length. The total number of cut variables can then be calculated as $\#P * \#A * \#B$. By making a problem instance more rectangular, both $\#P$ and $\#B$ will increase and $\#A$ will decrease as the object gets narrower. Due to the non-linear relationship, the decrease will be at a lower rate than the increase. Overall, CP-Guillotine gets an increase in its number of variables for rectangular problem instances and the model thus increases in size.

CP-Anchor on the other hand gets a benefit from the rectangular shape. As the surface area stays the same, so does the total number of anchor positions (besides some fluctuations due to the discretisation of the area with respect to the item shapes). What does change when a square becomes rectangular, is its circumference. More items will lay close to an edge, limiting the number of neighbours inside its movable region and in turn reducing the amount of non-overlap constraints. The model reduces in size.

Figure 6.5: **SLOPP density performance profile w.r.t. aspect ratio**



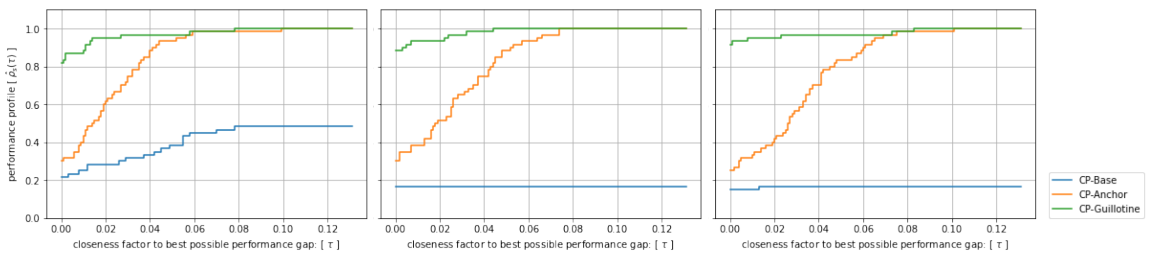
(a) 100×900 (b) 200×450 (c) 300×300

This insight matches with the observations: with increased rectangularity, CP-Guillotine increases in solving complexity whilst CP-Anchor decreases. This effect is clearly visible in *figure 6.5*. As the eventual purpose of this paper is to propose a model for the WPAS problem, where production happens in long rectangular segments, CP-Anchor was purposefully designed for the case of rectangular problems.

Number of item types The last influence to be analysed is the number of item types. As expected, more item types make for more complex models, resulting in more extreme differences in *figure 6.6*. At a certain problem area size, whichever model is more performant only gains relative performance difference when increasing the number of item types.

As has often been suggested throughout this analysis, the approximation of CP-Guillotine by CP-Anchor is only limited by the configured solve time limit. To verify this claim, that in the limit CP-Anchor should always dominate due to its general nature, some further experiments have been conducted with different time limits. The problem instances with object shape 100×400 were selected. The models were run with both a 4-minute and 16-minute time limit and the results are visible

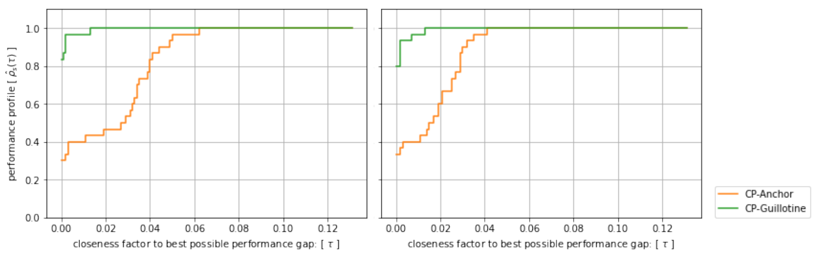
FIGURE 6.6: **SLOPP density performance profile w.r.t. # item types**



(A) 4 item types (B) 7 item types (C) 10 item types

in *figure 6.7*. It can be identified that indeed an increase in solve time improves the proximity of CP-Anchor to CP-Guillotine. The performance profile becomes steeper, shifting the probability distribution towards smaller gap values.

FIGURE 6.7: **SLOPP density performance profile w.r.t. solve time**



(A) 4 minutes (B) 16 minutes

6.5.3 [Q3] Improvements of iterative approach to Due-date

The previously explained benchmark suite for the due-date extension has been run for both the CP-OneShot and the CP-DLNS model. The performance profiles for the entire suite are visible in *figure 6.8*. The new optimisation objective balances multiple targets and is formulated as a minimisation problem. This stands in contrast with the previous maximisation of the packing density. The standard performance ratio can now be used instead of the reformulated performance gap.

As predicted, the CP-DLNS model mostly outperforms the baseline CP-OneShot model, with an identity-ratio probability of 86% and a maximal ratio of 1.56. CP-OneShot can only find the best solution for 12% of the problem instances and requires a maximal ratio of 4.40.

The suite introduced a new parameter: the number of deadlines. Splitting up the performance profiles results in *figure 6.9*. It can be determined that there is no observable correlation, CP-DLNS consistently outperforms CP-OneShot independent of the number of deadlines.

Figure 6.8: **Due-date objective performance profile**

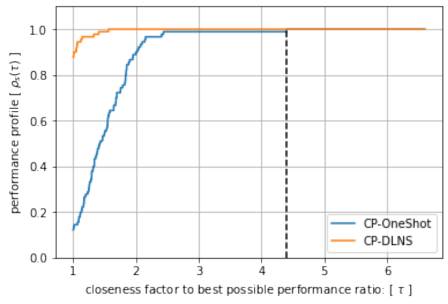
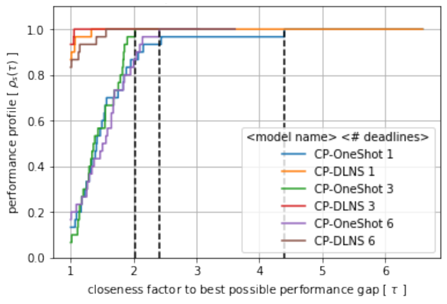


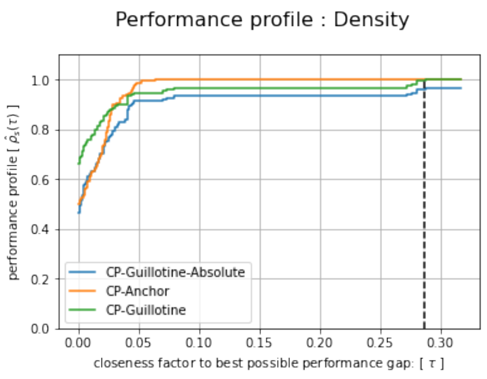
Figure 6.9: **Due-date objective performance profile w.r.t. # deadlines**



6.5.4 [Q4] Creel extension

Figure 6.10 visualises the performance profiles of the SLOPP models with the creel extension, adding support for textile colours and machine creel capacity to the problem formulation. CP-Base is not included, since **[Q1]** already demonstrated its unscalability. The creel extension can also be added on top of the due-date extension, resulting in the complete WPAS formulation. Since the impact of due dates has already been analysed, the easier-to-solve SLOPP models were chosen for this experiment

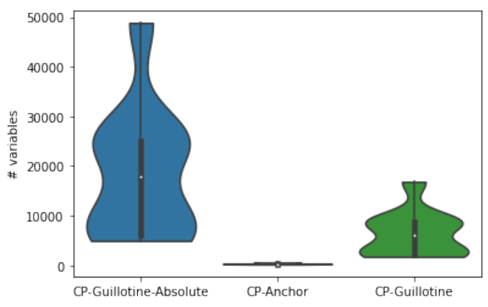
Figure 6.10: **Creel density performance profile**



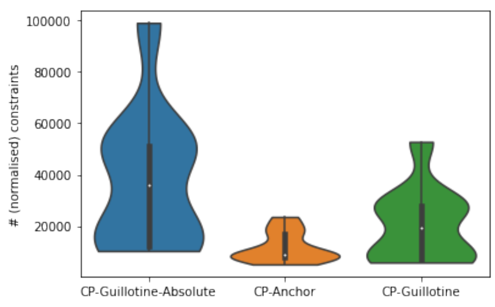
What is immediately apparent for the complete benchmark suite is the strong reduction in the performance lead of CP-Guillotine. The zero-gap probability is now 66% from the previous 80% for problem instances of scale 200^2. Starting from a gap of 0.0276 and probability of 89%, CP-Anchor consistently outperforms CP-Guillotine. If the application at hand requires the best-performing model for 90% or more of the problem instances, CP-Anchor is the best option.

As was predicted in **[Q1]**, CP-Guillotine suffers from its large number of variables when trying to add an extension on top. The new model sizes can be found in

Figure 6.11: **Creel model size**



(A) Variables



(B) Constraints

figure 6.11. CP-Anchor is now the smallest model, in terms of both constraints and variables. Luckily, the creel is an extension that does not need the absolute positional information of the items. If an extension would require item positions, it could be derived from the cutting locations as mentioned in *subsection 5.1.2 Approach*. This results in the model CP-Guillotine-Absolute. Simply formulating the absolute positions, without any extensions built on top, already results in a doubling of the number of the constraints and more than a doubling of the number of the variables. Due to the practicalities of this derivation, CP-Guillotine-Absolute has many more item instances than would be required by MPM_t: one for every possible *cut*. This results in any additional extension increasing the size difference even more. Looking back at *figure 6.10*, it can be seen that all performance benefits of the guillotine assumption are completely gone for this more realistic problem setting with extensions built on top of SLOPP using absolute positions. As an example of the limited scalability for extensions, formulating the creel extension on top of these derived absolute positions results in an unsolvable model within the given time restriction. Many problem instances even surpassed the memory limit of 16GB.

Chapter 7

Conclusion

The research conducted in this thesis has two goals: **I)** creating a formal description of the previously undocumented Weave Planning and Scheduling (WPAS) problem and **II)** proposing a novel mathematical model capable of solving this problem in an extensible and scalable manner. WPAS originates from the textile industry and involves the creation of a production planning for industrial-sized weaving machines or looms. Whilst this industry is known to be one of the oldest in existence, no good solutions to the complete problem can be found in the literature. By combining pre-existing techniques and novel mathematical formulations, this thesis demonstrated new models with improved performance and favourable model characteristics.

7.1 Formal description

Before a solution could be proposed, a formal description of the constrained optimisation problem had to be introduced to the literature. In *chapter 4 Problem statement* this was achieved by first slowly decomposing the problem at hand, revealing all the layered sub-problems in the process, after which each sub-problem could be formalised. With SLOPP or the Single Large Object Placement Problem at its core, a due-date extension for the inclusion of temporal aspects formulated on top, and the domain-specific extension related to the limitations of the weaving machine at its surface. The first two layers could be constructed using previous knowledge from literature. It results in a generic production planning optimisation problem, which can also be found in many other industrial sectors. The contributions of this thesis can thus have an impact beyond the textile industry. The last layer, which is domain-specific, is a new contribution to the literature. It includes many additional restrictions related to the physical constraints of weaving. With a complete formulation of the WPAS problem, there now exists a starting foundation for future research about this sub-domain of Operations Research (OR).

7.2 Mathematical models

The second goal of this thesis was to create a mathematical formulation and its accompanying algorithmic implementation capable of solving the WPAS problem. Constrained Programming was chosen as the paradigm, allowing for great flexibility thanks to the declarative concepts of decision variables, constraints, and objective functions. One of the big challenges related to WPAS and any similar production planning problem, is keeping the solution scalable and extensible. From the literature review in *chapter 3* it became apparent that many of the existing solutions are rather limited in their capabilities. Whilst being hyperefficient for their respective optimisation problems, they proved to be restrictive in terms of the extensibility towards more realistic problem cases. Within the weaving sector, there is no standardised technique for creating a production planning. Each manufacturer could have its own set of rules, preferences and customs, making a one-size-fits-all solution unsuitable. The model should be capable of adapting to these different requirements and not force a single planning approach upon the textile companies. It should serve as a support tool for expert planners.

For each of the sub-problems of WPAS, a new model is proposed. To analyse their performance and other characteristics, extensive experimentations were performed. Since WPAS is built upon the High Multiplicity variant of SLOPP for which pre-existing research was lacking, new benchmark suites are proposed based on the SLOPPGEN problem instance generator (Wäscher et al. (2012)).

SLOPP A baseline implementation of SLOPP was first introduced: CP-Base. It served as a reference, both in terms of implementation technique and in terms of performance. Whilst the formulation was general and supported any possible packing, experiments quickly revealed an unscalable aspect: its non-overlap constraints. Degeneracy within the encoding, based on a set-of-points representation, resulted in a large number of symmetries within the solution space. Non-overlap had to be formulated between every possible pair of item instances, resulting in an exploding number of constraints. Within the literature, one of the techniques to solve the issue of non-overlap is utilising a guillotine assumption. This is an assumption on the allowed cutting patterns, resulting in a size reduction of the search space. An efficient encoding from literature (Salem et al. (2023)) was adapted to SLOPP (CP-Guillotine), which completely removed the need for non-overlap constraints by making it an inherent characteristic of the encoding. Whilst efficient in solving this sub-problem, it posed issues related to its extensibility. The model traded a decrease in the number of constraints for a large increase in the number of decision variables. Whilst being efficient for SLOPP, any extension would have to be formulated on top of these variables, once again resulting in a large number of constraints. Additionally, by not having access to the exact item positions, location-bounded extensions become more difficult to formulate. The formulation even loses its performance benefit when trying to reintroduce these item positions.

The newly proposed CP-Anchor model takes another approach to improve the scalability of the non-overlap constraints. It builds upon the set-of-points encoding of CP-Base but introduces a generality-preserving grid structure. Instead of allowing each item instance to be packed anywhere, instances get bounded to the grid cells, limiting their range of motion. The representation loses its degeneracy caused by local symmetries, resulting in a unique encoding for each possible packing. Constraints can now be formulated locally. This property is utilised for non-overlap, resulting in a significant decrease in model size. CP-Anchor has all the same generality benefits of CP-Base, whilst significantly improving scalability and performance. Using the spacial locality of the decision variables, work can be moved from the solving phase to the model definition phase, reducing the number of constraints and the size of the search space. Access to the exact item positions (due to the used set-of-points encoding) allows for the formulation of location-bounded constraints. This is useful for weaving when for example two textiles should not be produced next to each other.

Extensive experiments have demonstrated that this new contribution closely approximates the performance of the CP-Guillotine model adapted from Salem et al. (2023), whilst outperforming it when extensions are added. CP-Anchor serves as a promising alternative representation for SLOPP models when generality, extensibility, and exact item positions are of importance, as is the case for WPAS.

Due-date The subsequently researched formulations included models for the due-date extension to SLOPP. Both a single-shot (CP-OneShot) and an iterative (CP-DLNS) approach got proposed. Experiments demonstrated the performance benefits of splitting up the optimisation problem into a collection of smaller steps. Further arguments were made regarding its dynamic nature, allowing for human-in-the-loop optimisation with preference feedback from the expert planners.

Creel The last extension towards WPAS included a representation of the creel and other weaving machine related restrictions (CP-WPAS). This research further demonstrated the benefits of the CP-Anchor model, with the previous performance advantage of CP-Guillotine almost disappearing. When covering 90% or more of the proposed benchmark suite, CP-Anchor consistently outperforms CP-Guillotine.

Whilst results are promising, the researched WPAS problem is still a basic version of the real production planning problem in the textile industry. Many real-world and manufacturer-specific extensions are still missing and the approach is not yet interactive. Implementing these will require more improvements towards the scalability of the models. Other possible improvements will be discussed in ”future work”.

7.3 Future work

The research performed in this thesis introduces several possibilities for future work.

One of the most promising topics is the inclusion of interactive solving techniques to incorporate feedback from expert planners. According to Liu et al. (2022), there are two factors that affect user trust in an optimisation system; its performance and its transparency [50]. Whilst new models have been proposed in this thesis to improve the solving performance for automated production scheduling and planning, it is still perceived as a ”black box” system. Liu et al. (2018) formalised the concept of the problem solving loop, consisting of a model definition loop and a separate optimisation loop [48]. To prevent these loops from becoming a black box, Liu et al. (2020) propose the use of interactivity and visual analytics [49]. The authors formulate a collection of nine recommendations when designing interactive visualisation tools for problem solving loops. Besides increasing user trust, the survey of Meignan et al. (2015) justifies the need for user interaction to reduce the gap between the mathematical model and the real world [53]. The real world is infinitely complex, requiring models to simplify their targeted problem. Interaction allows users to guide the system towards expected solutions and adds flexibility when encountering unexpected situations. The result is an optimisation system build as a tool to assist the expert decision-maker, not one to make the decision itself. The proposed models of this thesis were designed to support such an interactive addition by choosing extensible model formulations and an interactive design for due dates.

With the support for interactivity, preference learning could also be added. Whilst allowing user feedback is beneficial, it could quickly lead to a cognitive burden for the Decision Maker (DM) when trying to determine the Most Preferred Solution (MPS). DM’s typically find it difficult to concretely formulate their preference. It gets limited by the DM’s knowledge, insight, memory, and so on. and it might change whilst exploring alternatives. Dragone et al. (2018) propose a general framework for systematically collecting user feedback in constructive settings using targeted queries [23].

Additionally, many companies often have historical data to learn from. Instead of trying to explicitly formulate manufacturer-specific planning requirements within the model, a learning approach can be used to make decisions similar to the historical data. Canoy & Guns (2019) employed this technique for the Vehicle Routing Problem by learning a first-order Markov model [16].

Besides additions to the model’s formulation, additional research could be conducted on the real-world performance of the proposed techniques. All experiments in this thesis were performed on artificial benchmarks due to the lack of real-world data. Validating the techniques on actual MPS data from the industry and comparing the result with manual plannings would be of high value. More challenging to formulate planning restrictions from real manufacturers could be explored. Lastly, the contributions could be validated in other industries with similar planning requirements.

Appendices

Appendix A

Guillotine absolute positions

Algorithm 10
Additional constraints for absolute item positions within guillotine formulation.

```
1: Inputs:
       item dimensions: $w_t$, $l_t$
       guillotine size variables: $P$, $A$, $B$, $T$
       guillotine decision variables: $\gamma$, $pl$
2: Output: collection of constraints $C$

3: procedure GuillotineAbsolutePosition
4:     Initialise $C = \emptyset$
5:     for each $p \in P$ do
6:         $strip\_width \leftarrow 0$
7:         for each $a \in A$ do
8:             $cut\_height \leftarrow 0$
9:             for each $b \in B$ do
10:                for each $t \in T$ do
11:                    $C \leftarrow C \cup [x_{p,a,b,t} = strip\_width]$
12:                    $C \leftarrow C \cup [y_{p,a,b,t} = pattern\_height + cut\_height]$
13:                    $cut\_height \leftarrow cut\_height + l_t$
14:                end for
15:            end for
16:            $strip\_width \leftarrow \sum_{t \in T} (\gamma_{p,a,t} * w_t)$
17:        end for
18:        $pattern\_height \leftarrow pattern\_height + pl_p$
19:    end for
20:    return $C$
21: end procedure
```

Bibliography

[1] Flanders textile industry weaves past and future. Accessed: 2023-07-10. URL: https://www.flandersinvestmentandtrade.com/invest/en/sectors/textile.

[2] Pcb dielectric material selection and fiber weave effect on high-speed channel routing. Technical report, Altera Corporation, 101 Innovation Drive San Jose, CA 95134, 1 2011.

[3] History of Vandewiele, 5 2023. Accessed: 2023-07-10. URL: https://vandewiele.com/about-vandewiele-group/history-of-vandewiele.

[4] ITMA Milan 2023, 5 2023. Accessed: 2023-07-10. URL: https://vandewiele.com/events/itma-milan-2023.

[5] R. L. Ackoff, W. K. Holstein, M. Tanenbaum, and S. Eilon. operations research. In *Encyclopedia Britannica*. Feb. 2023. URL: https://www.britannica.com/topic/operations-research.

[6] A. Ancion and B. Dardenne. Self-adapting LNS minimizing expected time to improvement, applied to the vehicle routing problem. *Computers Operations Research*, 2016.

[7] C. Arbib and F. Marinelli. On cutting stock with due dates. *Omega*, 46:11–20, 7 2014. doi:10.1016/j.omega.2014.01.004.

[8] E. Baines, Sir, H. Fisher, R. Fisher, and P. Jackson. *History of the Cotton Manufacture in Great Britain*. 1 1835.

[9] J. E. Beasley. OR-LIBRARY. Accessed: 2022-11-23. URL: https://www.brunel.ac.uk/~mastjjb/jeb/info.html.

[10] J. E. Beasley. OR-Library: Distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41(11):1069–1072, 11 1990. doi:10.1057/jors.1990.166.

[11] J. O. Berkey and P. Wang. Two-Dimensional finite Bin-Packing algorithms. *Journal of the Operational Research Society*, 38(5):423–429, 5 1987. doi:10.1057/jors.1987.70.

[12] A. Bloch-Hansen, R. Solis-Oba, and A. Yu. High multiplicity strip packing with three rectangle types. In I. Ljubić, F. Barahona, S. S. Dey, and A. R. Mahjoub, editors, *Combinatorial Optimization*, pages 215–227, Cham, 2022. Springer International Publishing.

[13] C. Blum, M. Aguilera, A. Roli, and M. Sampels. *Hybrid Metaheuristics: an Emerging Approach to Optimization.* 4 2008. URL: http://ci.nii.ac.jp/ncid/BA85675075.

[14] M. Bonney. Reflections on production planning and control (PPC). *Gesto produo*, 7(3):181–207, 12 2000. doi:10.1590/s0104-530x2000000300002.

[15] R. L. Brooks, C. Smith, A. H. Stone, and W. T. Tutte. The dissection of rectangles into squares. *Duke Mathematical Journal*, 7(1), 1 1940. doi:10.1215/s0012-7094-40-00718-9.

[16] R. Canoy and T. Guns. Vehicle routing by learning from historical solutions. In T. Schiex and S. de Givry, editors, *Principles and Practice of Constraint Programming*, pages 54–70, Cham, 2019. Springer International Publishing.

[17] M. Cartwright. The textile industry in the British Industrial Revolution. *World History Encyclopedia*, 3 2023. URL: https://www.worldhistory.org/article/2183/the-textile-industry-in-the-british-industrial-rev/.

[18] N. Christofides and C. Whitlock. An algorithm for two-dimensional cutting problems. *Operations Research*, 25(1):30–44, 2 1977. doi:10.1287/opre.25.1.30.

[19] J. L. Clifford and M. P. Posner. Parallel machine scheduling with high multiplicity. *Mathematical Programming*, 89(3):359–383, 2 2001. doi:10.1007/pl00011403.

[20] E. G. Coffman, M. R. Garey, and D. W. Johnson. An application of Bin-Packing to multiprocessor scheduling. *SIAM Journal on Computing*, 7(1):1–17, 2 1978. doi:10.1137/0207001.

[21] J.-F. Côté and M. Iori. The meet-in-the-middle principle for cutting and packing problems. *Informs Journal on Computing*, 30(4):646–661, 11 2018. doi:10.1287/ijoc.2018.0806.

[22] E. M. Dolan and J. J. Mor. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213, 1 2002. doi:10.1007/s101070100263.

[23] P. Dragone, S. Teso, and A. Passerini. Constructive preference elicitation. *Frontiers in Robotics and AI*, 4, 2018. URL: https://www.frontiersin.org/articles/10.3389/frobt.2017.00071, doi:10.3389/frobt.2017.00071.

[24] S. P. Fekete, J. Schepers, and J. T. Van Der Veen. An exact algorithm for higher-dimensional orthogonal packing. *Operations Research*, 55(3):569–587, 6 2007. doi:10.1287/opre.1060.0369.

[25] P. C. Gilmore and R. E. Gomory. A linear programming approach to the cutting-stock problem. *Operations Research*, 9(6):849–859, 12 1961. doi:10.1287/opre.9.6.849.

[26] P. C. Gilmore and R. E. Gomory. Multistage cutting stock problems of two and more dimensions. *Operations Research*, 13(1):94–120, 2 1965. doi:10.1287/opre.13.1.94.

[27] M. C. N. Gramani and P. M. Frana. The combined cutting stock and lot-sizing problem in industrial processes. *European Journal of Operational Research*, 174(1):509–521, 10 2006. doi:10.1016/j.ejor.2004.12.019.

[28] T. Guns. Increasing modeling language convenience with a universal n-dimensional array, cppy as python-embedded example. In *Proceedings of the 18th workshop on Constraint Modelling and Reformulation at CP (Modref 2019)*, volume 19, 2019.

[29] F. Gzara, S. Elhedhli, and B. Yildiz. The Pallet Loading Problem: Three-dimensional bin packing with practical constraints. *European Journal of Operational Research*, 287(3):1062–1074, 12 2020. doi:10.1016/j.ejor.2020.04.053.

[30] K. Hadj Salem, E. Silva, J. F. Oliveira, and M. A. Carravilla. Mathematical models for the two-dimensional variable-sized cutting stock problem in the home textile industry. *European Journal of Operational Research*, 306(2):549–566, 2023. URL: https://ideas.repec.org/a/eee/ejores/v306y2023i2p549-566.html, doi:10.1016/j.ejor.2022.08.01.

[31] R. W. Haessler. Controlling cutting pattern changes in one-dimensional trim problems. *Operations Research*, 23(3):483–493, 6 1975. doi:10.1287/opre.23.3.483.

[32] L. Hendry, K. K. Fok, and K. Shek. A cutting stock and scheduling problem in the copper industry. *Journal of the Operational Research Society*, 47(1):38–47, 1 1996. doi:10.1057/jors.1996.4.

[33] F. Hermenier, S. Demassey, and X. Lorca. *BiN repacking scheduling in virtualized datacenters.* 1 2011. doi:10.1007/978-3-642-23786-7_5.

[34] J. W. Herrmann. *A History of Production Scheduling*, pages 1–22. Springer US, Boston, MA, 2006. doi:10.1007/0-387-33117-4_1.

[35] J. C. Herz. Recursive computational procedure for two-dimensional stock cutting. *IBM journal of research and development*, 16(5):462–469, 9 1972. doi:10.1147/rd.165.0462.

[36] D. S. Hochbaum and R. Shamir. Strongly polynomial algorithms for the high multiplicity scheduling problem. *Operations Research*, 39(4):648–653, 8 1991. doi:10.1287/opre.39.4.648.

[37] H. Hojabri, M. Gendreau, J.-Y. Potvin, and L.-M. Rousseau. Large neighborhood search with constraint programming for a vehicle routing problem with synchronization constraints. *Computers Operations Research*, 92:87–97, 4 2018. doi:10.1016/j.cor.2017.11.011.

[38] M. Iori, V. L. de Lima, S. Martello, F. K. Miyazawa, and M. Monaci. Exact solution techniques for two-dimensional cutting and packing. *European journal of operational research*, 289(2):399–415, 2021.

[39] Z. Ivkovi and E. L. Lloyd. Partially dynamic bin packing can be solved within $1 + \in$ in (amortized) polylogarithmic time. *Information Processing Letters*, 63(1):45–50, 7 1997. doi:10.1016/s0020-0190(97)00092-6.

[40] K. Jansen and L. Prdel. New approximability results for two-dimensional bin packing. *Algorithmica*, 74(1):208–269, 10 2014. doi:10.1007/s00453-014-9943-z.

[41] M. L. Joseph. *Introductory Textile Science*. Holt McDougal, 3 edition, 1 1977.

[42] L. Junqueira, R. Morabito, and D. S. Yamashita. Three-dimensional container loading models with cargo stability and load bearing constraints. *Computers Operations Research*, 39(1):74–85, 1 2012. doi:10.1016/j.cor.2010.07.017.

[43] A. H. G. R. Kan. *Machine Scheduling Problems: Classification, complexity and computations*. 7 1976. URL: http://ci.nii.ac.jp/ncid/BA20678996.

[44] L. V. Kantorovich. Mathematical methods of Organizing and planning production. *Management Science*, 6(4):366–422, 7 1960. doi:10.1287/mnsc.6.4.366.

[45] P. Laurent and F. Vincent. Or-tools. URL: https://developers.google.com/optimization/.

[46] S. Lewis. operations research (OR). *WhatIs.com*, 9 2019. URL: https://www.techtarget.com/whatis/definition/operations-research-OR.

[47] L. Lins, S. Lins, and R. Morabito. An n-tet graph approach for non-guillotine packings of n-dimensional boxes into an n-container. *European Journal of Operational Research*, 141(2):421–439, 9 2002. doi:10.1016/s0377-2217(02)00135-2.

[48] J. Liu, T. Dwyer, K. Marriott, J. Millar, and A. Haworth. Understanding the relationship between interactive optimisation and visual analytics in the context of prostate brachytherapy. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):319–329, 2018. doi:10.1109/TVCG.2017.2744418.

[49] J. Liu, T. Dwyer, G. Tack, S. Gratzl, and K. Marriott. Supporting the problem-solving loop: Designing highly interactive optimisation systems, 09 2020.

[50] J. Liu, K. Marriott, T. Dwyer, and G. Tack. Increasing user trust in optimisation through feedback and interaction. *ACM Transactions on Computer-Human Interaction*, 29, 07 2022. doi:10.1145/3503461.

[51] I. Manuel and M. Silvano. An annotated bibliography of combined routing and loading problems. *Yugoslav Journal of Operations Research*, 23(3):311–326, 2013.

[52] S. Martello and D. Vigo. Exact solution of the Two-Dimensional finite bin packing problem. *Management Science*, 44(3):388–399, 3 1998. doi:10.1287/mnsc.44.3.388.

[53] D. Meignan, S. Knust, J.-M. Frayret, G. Pesant, and N. Gaud. A review and taxonomy of interactive optimization methods in operations research. *Transactions on Interactive Intelligent Systems*, 5:1–43, 10 2015. doi:10.1145/2808234.

[54] J. Middleton, R. T. Icarte, and J. A. Baier. Large Neighborhood Search with Decision Diagrams. 7 2022. doi:10.24963/ijcai.2022/659.

[55] M. Mongeau and C. Bes. Optimization of aircraft container loading. *IEEE Transactions on Aerospace and Electronic Systems*, 39(1):140–150, 1 2003. doi:10.1109/taes.2003.1188899.

[56] V. Neidlein, A. Scholz, and G. Wäscher. SLOPPGEN: a problem generator for the two-dimensional rectangular single large object placement problem with defects. *International Transactions in Operational Research*, 9 2014. doi:10.1111/itor.12119.

[57] J. Oliveira and J. Soeiro Ferreira. A faster variant of the gilmore and gomory technique for cutting stock problems. *Jorbel : Belgian Journal of Operational Research, Statistics and Computer Science*, 34:23–37, 01 1994.

[58] C. Paquay, M. Schyns, and S. Limbourg. A mixed integer programming formulation for the three-dimensional bin packing problem deriving from an air cargo application. *International transactions in operational research*, 23(1-2):187–213, 2016.

[59] P. M. Pardalos, S. Martello, and D. Vigo. Recent advances on two-dimensional bin packing problems. *Discrete Applied Mathematics*, 123(1-3):379–396, 11 2002. doi:10.1016/s0166-218x(01)00347-x.

[60] P. M. Pardalos and M. Monaci. Integer linear programming models for 2-staged two-dimensional Knapsack problems. *Mathematical Programming*, 94(2-3):257–278, 1 2003. doi:10.1007/s10107-002-0319-9.

[61] S. Parmar and T. Malik. Application of textiles in automobile application of textiles in automobile. 7 2018.

[62] F. Parreo, M. Alonso, and R. Alvarez-Valdes. Solving a large cutting problem in the glass manufacturing industry. *European journal of operational research*, 287(1):378–388, 2020.

[63] M. E. Rayner. An introduction to mathematical modelling. *The Mathematical Gazette*, pages 1–5, 3 1980. doi:10.2307/3615903.

[64] H. Reinertsen and T. Vossen. The one-dimensional cutting stock problem with due dates. *European Journal of Operational Research*, 201(3):701–711, 3 2010. doi:10.1016/j.ejor.2009.03.042.

[65] F. Rossi, P. Van Der Beek, and T. Walsh. *Handbook of Constraint Programming*. 1 2006. doi:10.1016/s1574-6526(06)x8001-x.

[66] S. D. Russell and P. Norvig. Artificial intelligence: a modern approach. *Choice Reviews Online*, 33(03):33–1577, 11 1995. doi:10.5860/choice.33-1577.

[67] K. H. Salem and Y. Kieffer. *New symmetry-less ILP formulation for the classical one dimensional Bin-Packing problem*. 1 2020. doi:10.1007/978-3-030-58150-3_34.

[68] P. M. Satchell. *Innovation and Automation*. Number 1. 12 1998.

[69] A. Schrijver. *Combinatorial optimization : polyhedra and efficiency*. Algorithms and combinatorics 24. Springer, Berlin, 2003.

[70] Z. Sezer and . Muter. *Two-Stage Cutting Stock Problem with Due Dates*. Springer Nature, 7 2017. doi:10.1007/978-3-319-55702-1_20.

[71] E. Silva, C. Vies, J. L. Oliveira, and M. A. Carravilla. *Integrated Cutting and Production Planning: a case study in a home textile manufacturing company*. 1 2015. doi:10.1007/978-3-319-24154-8_25.

[72] C. A. Sousa, E. Silva, M. J. Lopes, and A. J. Ramos. *The cutting stock problem: a case study in a manufacturer of pet Vivaria*. Springer International Publishing, 1 2015. doi:10.1007/978-3-319-24154-8_26.

[73] J. Terno, R. Lindemann, and G. Scheithauer. Zuschnittprobleme and ihre praktische losung. Technical report, 1987.

[74] J. Verstichel, J. Kinable, P. De Causmaecker, and G. Vanden Berghe. A combinatorial benders' decomposition for the lock scheduling problem. *Computers & operations research*, 54:117–128, 2015.

[75] J. Wy and B.-I. Kim. Two-staged guillotine cut, two-dimensional bin packing optimisation with flexible bin size for steel mother plate design. *International Journal of Production Research*, 48(22):6799–6820, 12 2009. doi:10.1080/00207540903317523.

[76] G. Wscher, H. Hauner, and H. Schumann. An improved typology of cutting and packing problems. *European Journal of Operational Research*, 183(3):1109–1130, 12 2007. doi:10.1016/j.ejor.2005.12.047.