UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Head Office: Università degli Studi di Padova
Department of Mathematics "Tullio Levi-Civita"

Ph.D. Course in Mathematical Sciences
Curriculum: Computational Mathematics
Series XXXVII

Two-Dimensional Cutting and Packing Problems in the Sheet Metal Industry

Thesis written with the financial contribution of Salvagnini Italia S.p.A. through a Higher Education and Research Apprenticeship contract.

Coordinator: Prof. Giovanni Colombo
Academic Supervisor: Prof. Luigi De Giovanni
Industrial Supervisor: Nicola Gastaldon, Ph.D.

Ph.D. Candidate: Chiara Turbian

November 30th, 2024

Abstract

This thesis addresses real-world optimization problems in the sheet metal industry, where decision-making involves various operations to process raw material into finished objects. These processes can include cutting operations on material plates, as well as bending, assembly, and storage of the cut objects. Optimization goals typically aim to minimize material waste, reduce processing time, or maximize the efficiency of downstream operations.

Salvagnini Italia S.p.A., the headquarters of the Salvagnini Group, is a leading company in the sheet metal industry, providing cutting and bending machines that are often integrated into manufacturing systems. These machines are equipped with specialized software that offers automated solutions to the aforementioned problems.

This thesis focuses on Salvagnini's cutting machines, and, specifically, on the problem of generating cutting layouts for rectangular objects. This involves placing items to be cut into available material sheets in a way that minimizes waste. The problem incorporates both general and specific constraints arising from the technologies used by the reference company, creating a complex scenario that extends beyond standard formulations in the literature. In this regards, the problem presents specific attributes such as the presence of optional items to cut, production precedence among items, and required safety distances to maintain cut quality or optimize shared cuts. Some of these features have been addressed in the literature, while others, to the best of our knowledge, have not been explored either individually or in combination. Thus, defining the problem under study constitutes a primary contribution of the thesis, along with the mathematical formulation we devised to describe it.

The aim of our research is to develop a solution method for the problem under study that is applicable to real-world scenarios. Moreover, we seek to ensure that this method is competitive with the current third-party software used by the company, offering a viable alternative solution procedure. Various approaches have been explored, including exact methods, heuristics, and matheuristics. Among these, a heuristic approach based on a beam search algorithm proved itself to be particularly effective, delivering high-quality solutions with the required speed for practical industrial use. The beam search algorithm's performance makes it a valid alternative to the existing software, balancing execution time and solution quality. Concerning the other methods explored in this thesis, they provided valuable insights into the problem's specific challenges. These methods offer different perspectives on the problem and can be effective in particular scenarios, underscoring the need for diverse solutions tailored to varying conditions. All the proposed approaches result from an original combination of concepts drawn from the literature, which have been extended or adapted to address the practical attributes of our problem, thereby advancing known techniques and making

them applicable to a broader range of situations.

Additionally, this research investigates an irregular variant of the problem, where the items to be cut are irregularly shaped. In this context, we explore the integration of a recently proposed framework to handle the geometric aspects of the problem, comparing it with more established geometric tools from the literature. Although this extension is still under development, preliminary results are promising and induce significant industrial interest. The irregular variant introduces new challenges, especially related to managing the geometric complexity of the shapes, but it also holds potential for enhancing flexibility and efficiency in production processes.

Future work will focus on further refining the beam search algorithm, including leveraging more parallel computing to better optimize computational efficiency. Moreover, additional efforts will be directed towards testing and improving the irregular variant, integrating it with real-world attributes, and comparing it with existing methods to ensure practical applicability.

The thesis contributes to the field of cutting and packing problems by addressing real-world attributes and developing advanced solution methods to be integrated into real cutting environments. The thesis also provides practical benchmarks from the reference company, offering new valuable resources for evaluating solution approaches in realistic settings. Results highlight the beam search algorithm as a robust tool for industrial applications. The exploration of the irregular variant paves the way for future research and practical implementation, offering valuable implications for both academic research and industrial purposes.

Contents

Acknowledgments

This thesis is the result of the work, experience, and presence of many people.

I would like to thank my academic supervisor, Prof. Luigi De Giovanni, who provided me with the right amount of freedom over these three years to explore the world of research, guiding me when necessary, and always making himself available to clarify doubts and discuss new ideas.

A big thank you goes to my industrial supervisor, (Gentilissimo Dottor) Nicola Gastaldon, whose enthusiasm and energy fueled my passion for Operations Research, knowing how to motivate me in the most difficult moments, and celebrating with me the small goals we reached together.

I thank Salvagnini Italia for making this project possible and all the people within the company who took part in it. In particular, my colleagues and superiors, who always showed interest and were supportive of the research.

I would like to thank Prof. Tony Wauters, Jeroen Gardeyn, and the entire CODeS research group at KU Leuven – Technology Campus of Gent, for welcoming me during a collaboration period that was very interesting and stimulating for me.

Thank you to all the people I met at conferences. Both personally and professionally, you all contributed to enriching this PhD experience.

I thank all my friends, those closest and long-known, those living around the world, and the most recent friendships. You have been, and continue to be, fundamental in standing by my side during the toughest times, bringing lightheartedness and understanding when needed, and sharing important moments like this one with me.

Finally, over these three years, the support of my family has never been missing, especially my mom and dad, who always know how to help me both with words and actions, whether near or far. The biggest thanks goes to you, for always being there to support me in all my decisions, you are my safe harbor.

One day, someone told me, "Research is like Direct Search: you try different paths, evaluate them, and discover along the way whether you are improving or not." And it's so true. Thanks to each one of you, let's keep trying, evaluating, and discovering together.

Chapter 1

Introduction

In the sheet metal industry, companies typically process raw materials to produce various objects and products for customers. These can range from small items, such as metal brackets or hinges, to large products like industrial panels or enclosures. Each production area has its unique characteristics, but the general process usually involves major steps such as cutting the material, bending it, and assembling the pieces. Throughout these stages, numerous decision-making problems arise, and workers must answer questions like: Where should the items to be cut be placed on the raw material to maximize material utilization? How should the cutting and bending technology be applied to minimize processing time? In which sequence should the items, and their corresponding sheets, be cut to minimize the number of unloading and assembly operations, while respecting the predefined production order within each batch? Traditionally, workers have relied on shared experience to address these issues. However, in recent decades, software has been developed to support these decision-making processes, helping to save both time and material costs by providing high-quality solutions.

This thesis approaches the problem from the perspective of these software tools, which are now integrated into computer numerical control (CNC) machines. Specifically, we refer to Salvagnini Italia S.p.A., a leading company in the sheet metal industry that produces and commercializes CNC cutting and bending machines, along with dedicated software, flexible manufacturing systems, and additional services for the production process [81].

1.1 Technological context

We focus on cutting machines, that play a crucial role in the production process of final objects. The process consists of several macro stages, as illustrated in Figure 1.1. The first step involves having sheets of raw materials available to be loaded, one at a time, into the cutting machine (as shown in Figure 1.1 (a)). These sheets can vary in characteristics, such as size and

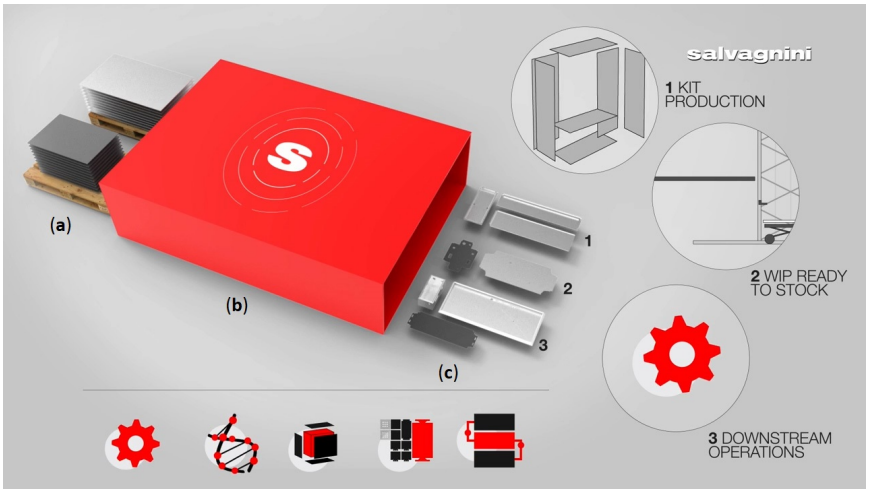


Figure 1.1: Macro stages involving Salvagnini cutting machines [81].

thickness, to meet the specific requirements of the objects being produced.

Once a sheet is loaded, the cutting machine extracts items from the plates according to a specified cutting layout. This process is depicted as a large red box in Figure 1.1 (b). In fact, Salvagnini does not offer a unique type of cutting machine, but a variety of them, each equipped with distinct technologies, tools, features, and requirements, providing for different production needs. Depending on the technology, the cutting phase may require practical constraints aimed at preserving item quality or guaranteeing a correct use of materials.

After the cut, the items are ready to be unloaded from the machine (Figure 1.1 (c)). Depending on the context, items may be grouped into kits, each containing the items that will form part of a same final object (for example, when producing a locker, the door and sides would be grouped in the same kit). Notice that, to facilitate the unloading process, the items must be cut in a specific order so that, e.g., items of the same kit are produced consecutively. If not, several kits would be simultaneously active during the cutting process, thus requiring substantial space to stack all the items of different kits, and necessitating further operators to correctly gather the items themselves together. The work-in-progress (WIP) items are then ready to be stored in depots, awaiting subsequent downstream operations such as bending and assembly, as required for the current production batch, or for future use if items are produced in excess.

Alongside the machines, Salvagnini provides software solutions to assist clients in addressing the decision-making challenges involved in these processes. The thesis investigates the problem of generating cutting layouts (an example is provided in Figure 1.2), which involves placing items on available material sheets with the aim of minimizing material waste while adhering

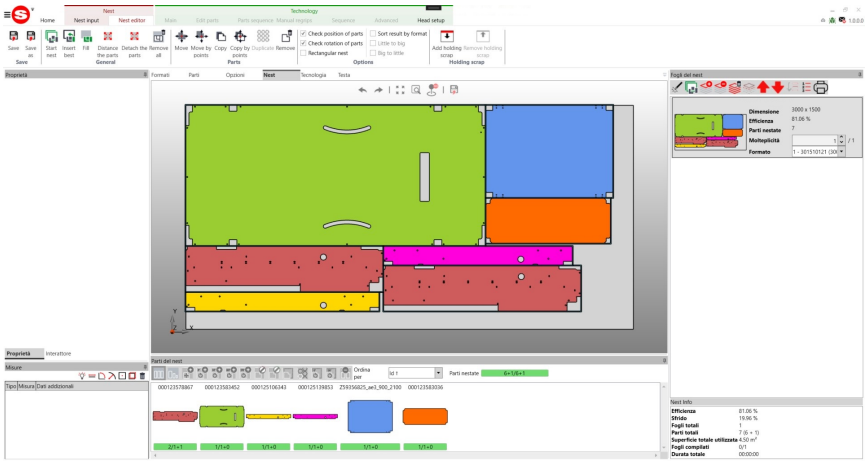


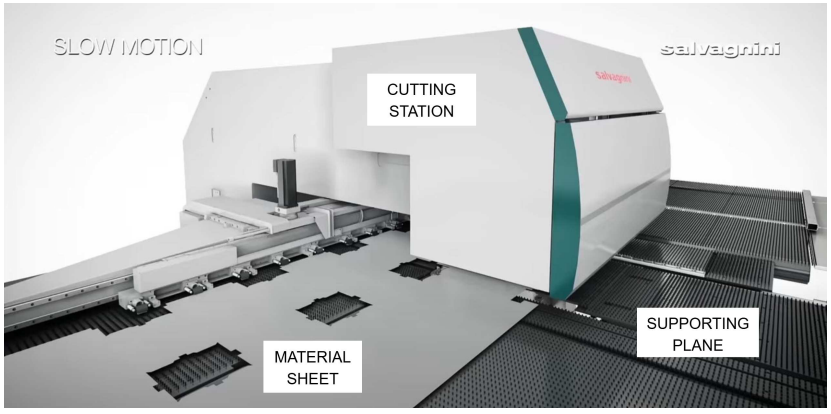
Figure 1.2: Example of cutting layout generated with Salvagnini's software.

to technological constraints. Constraints depend on the cutting process as well as on the cutting technology adopted. In this thesis, we will focus on the *punching technology*. Punching machines have an internal supporting plane where the material sheet lies, and, above it, a cutting station, as shown in Figure 1.3a. The cutting station, detailed in Figure 1.3b, includes a punching head equipped with different punching tools and an unloading shear. The punching process involves pressing the punching tools into the sheet metal to create holes, shapes, or cut-outs by removing small pieces of the material. The punching head holds and operates a variety of rotating punching tools, allowing quick changes between punch shapes and sizes as needed. Finally, the unloading shear is used to separate finished parts from the main sheet by cutting their bounding rectangle, ensuring clean edges and precise shapes. For a detailed video explanation of the process, the reader can refer to [82].

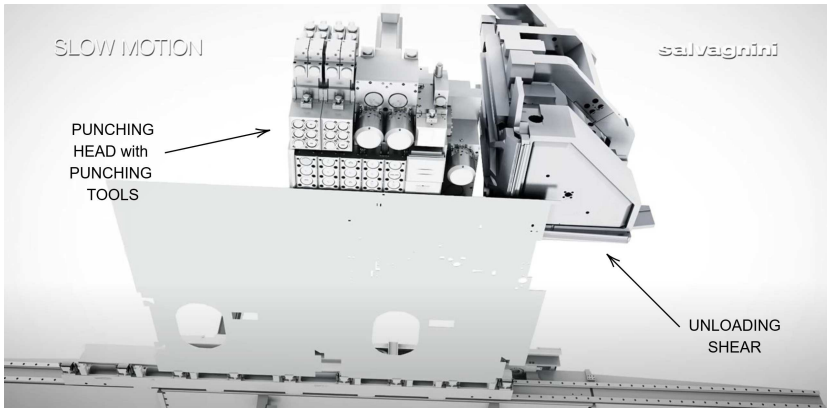
Concerning the software solutions, currently, the company employs an outsourced software that has consistently delivered high-quality results across different machines, despite varying technological requirements. The goal of this research is to thoroughly examine the problem and its specific constraints, focusing on the development of an efficient solution method applicable to real-world contexts. In particular, we aim to propose this method as a competitive alternative to the third-party software currently used by the reference company.

1.2 Problem overview

The problem of generating cutting layouts is well known in the Operations Research field, falling into the wide class of *Cutting and Packing Problems*



(a) Punching machine - supporting plane, cutting station [82].



(b) Cutting station - punching head with punching tools, unloading shear [82].

Figure 1.3: Key elements of a cutting machine with punching technology.

(C&PP). These problems arise in a variety of industries, such as, for example, paper, glass, or marble industries, in logistics, or freight transportation. Although they can differ significantly with each others due to specific constraints, they share a common goal: assigning small items to larger objects in a way that optimizes a given objective function.

In our case, the problem consists of placing rectangular items (representing the boxes that bound the small objects to be produced) onto available material sheets in such a way that material waste after cutting is minimized. As previously mentioned, depending on the machine and the overall production process, additional requirements may need to be considered, such as production precedence between items, minimum distances between them to ensure cutting quality, and other limitations specific to the machine in use.

Although the basic definition of this problem has been extensively studied in the literature, the problem addressed in this thesis involves numerous attributes that, to the best of our knowledge, limit or preclude the use of state-of-the-art solution methods. Some of these peculiarities have been considered in previous studies, while others are novel, coming from advances in cutting technology or specific features of interest to Salvagnini.

Along with the problem with rectangular items, the thesis proposes a preliminary study on a variant concerning items with irregular shapes. To this end, we extend one of the solution methods devised for the problem with rectangular items, and we integrate a recently proposed geometric tool to handle the challenges associated with the irregularity of the items. This extension offers the possibility to improve flexibility and efficiency in production processes involving different cutting technologies.

Our research aims to contribute to the scientific literature on C&PP by integrating new attributes and potentially introducing new classes of problems of broader interest. The thesis develops techniques to address the rectangular problem, incorporating a significant variety of attributes, and presents and extension to irregular shaped items. Moreover, thanks to the collaboration with Salvagnini, we have devised a benchmark of real and realistic instances to test the proposed approaches and compare them with the algorithm currently used by the company.

1.3 Structure of the thesis and contributions

The remainder of the thesis is structured as follows.

- **Chapter 2** provides an overview of the theoretical and methodological tools used throughout the thesis. It introduces mathematical programming, with a particular focus on linear and mixed-integer linear programming. The chapter reviews modeling techniques and the main exact methods for solving such problems (namely, simplex method, cutting planes, branch-and-bound). It also presents the main heuristic approaches investigated in this work (namely, greedy algorithms, pilot method, beam search). Finally, the chapter briefly discusses the possibility of optimizing multiple objective functions in a given order, known as lexicographic optimization.

- **Chapter 3** starts by giving a more detailed introduction to the general class of two-dimensional C&PP, which serves as the framework for this study. It then presents the real-world context of the problem, with particular emphasis on the technologies that define the problem's attributes. A formal definition of the Two-Dimensional Cutting and Packing Problem with Punching Technology (2DC&PP-PT), main

subject of the thesis, is provided, outlining its key elements and practical requirements. Some of these attributes are familiar from the literature, though we highlight specific peculiarities that, to the best of our knowledge, have not been previously considered, as well as the combination of all of these features. In this regard, the problem definition contributes to the existing literature by introducing and formalizing a new, complex variant of a well-known class of packing problems.

- **Chapter 4** reviews the Operations Research literature related to C&PP, to contextualize the specific problem tackled in this work. The most common mathematical formulations are discussed, along with references to the specific attributes of our problem. Since not all of the attributes are present in the literature, we refer to similar concepts where applicable and point out possible missing research. The chapter concludes with an overview of heuristic methods commonly used in this field.

- **Chapter 5** presents a mathematical formulation of the problem through two mixed-integer linear programming models, which defines another contribution of the thesis. The first model optimizes the primary objective, waste minimization, while the second model extends the former to address soft constraints related to precedence relations as a secondary objective, incorporating all the attributes of the problem. We emphasize the devised original models representing the peculiar combination of constraints, and the formulations developed to represent all the required technological attributes.

- **Chapter 6** discusses the solution methods designed to solve the problem. These include two model-based approaches (an exact method and a matheuristic) and three heuristic methods. The exact method provides optimality bounds for the computational results, while the matheuristic implements a decomposition approach to handle the precedence constraints by an iterative strategy. The heuristics aim to generate competitive results in terms of both execution time and solution quality. The contribution of the chapter relies in an original combination of algorithmic techniques taken from literature (like, e.g., beam search, pilot method, extreme points), that have been also adapted for our problem, to include all the specific attributes into their components. Moreover, the beam search based approach that we propose in this chapter represents a competitive method with respect to the software currently in use at the reference company, as shown by the computational results reported in Chapter 8.

- **Chapter 7** explores a preliminary extension of the most promising heuristic approach to a variant of cutting and packing problems in-

volving irregularly shaped items. The chapter locates the extension of the problem within the existing literature and motivates the interest in this variant. A recently proposed geometric tool is integrated in the solving procedure to handle a critical part of this class of problems, concerning the irregularity of the items. To the best of our knowledge, this is the first time the tool has been applied in an optimization algorithm, aside from the original work by the tool's authors, which provides an insight into the potential of the tool and adds theoretical interest to its comparison with existing literature.

- **Chapter 8** outlines the implementation of the proposed solution approaches and the results of their test. A comprehensive computational campaign was conducted on an extensive benchmark of real-world instances devised in collaboration with the reference company, which stands as a further contribution of this work. The chapter presents the experiments and detailed computational results, comparing the proposed solutions with the company's current procedure. Our findings show that model-based methods deliver high-quality solutions, but with execution times that are impractical for real-world applications. However, they still provide valuable insights into optimality bounds. The three heuristic approaches are evaluated, each offering progressively better solution quality at a correspondingly longer execution time, though still feasible for industrial purposes. In particular, the beam search approach proved to be competitive with the company's existing procedure. Preliminary results for the irregular problem variant are also included, showing promising early-stage findings.

- **Chapter 9** concludes the thesis with overall observations and suggestions for future research perspectives.

Chapter 2

Methodologies

Operations Research deals with optimization tasks, providing formulations of the involved problems, along with a wide range of solving techniques. An optimization problem generally aims at finding an *optimal solution* among the feasible ones, which minimizes or maximizes the *objective function* of the problem. A *feasible solution* is defined by values assigned to a set of *decision variables*, and it satisfies a set of *constraints*, that describes the domain of the variables in the search space. In this thesis, we always refer to optimization problems as minimization ones, unless stated otherwise. Nevertheless, all the reasoning that we present holds for the symmetric case of maximization.

To find a solution, there are several approaches that can be used, whose choice depends on the characteristics of the problem itself, and on potential requirements coming with the problem. If a guarantee of optimality is needed, *exact algorithms* must be used. These methods usually require high computational resources and execution times, thus, with large size instances, or problems with a complex internal structure, they could rapidly become impractical. A valid alternative in these cases is given by *heuristic algorithms*. These methods find feasible solutions that are possibly close to the optimum in terms of objective function, in a reasonably short amount of time. In real-world applications, it is often the case that memory and time resources are limited, therefore heuristic approaches are frequently used.

A significant class of exact algorithms works with problems formulated via mathematical programming, in particular using linear models, as discussed in Section 2.1 and 2.2. Among the most important exact approaches, there are the *Cutting Planes* and the *Branch-and-Bound* methods. There are commercial and non-commercial solvers implementing and exploiting these algorithms to solve problems to optimality (e.g., CPLEX [46], Gurobi [41], Xpress [33], SCIP [1]). As regards heuristic approaches, this chapter gives, in Section 2.3, a description of some widely used algorithms of relevance for the thesis, namely the *Greedy Algorithm*, the *Pilot Method*, and the *Beam Search*. Heuristics generally do not involve mathematical programming for-

mulations, but when they do, they constitute *matheuristics*, which is a fairly recent class of algorithms, gaining a lot of research interest for its results [63].

An optimization problem may have more than one minimization or maximization goal, in this case the problem belongs to the multi-objective optimization category. If the objective functions are sorted with respect to a priority, the problem can be solved by single-objective optimization techniques, such as in the *Lexicographic Method*, that will be presented in Section 2.4.

To have a more extensive description of the notions and the methodologies that will be presented in the next sections, the reader can refer to the sources used throughout this chapter, in particular, [16, 21, 53] for mathematical programming and exact methods, [84] for heuristic and metaheuristic approaches, and to [77] for lexicographic optimization.

2.1 Linear Programming

Mathematical programming represents one approach to solve an optimization problem through a mathematical model. A mathematical programming model has the following general form:

$$\begin{aligned} \min \quad & f(x) \\ \text{s.t.} \quad & g_i(x) \leq 0 \quad \forall i \in I \\ & h_j(x) = 0 \quad \forall j \in J \\ & x \in X \subseteq \mathbb{R}^n. \end{aligned} \tag{2.1}$$

A model describes the characteristics of the optimal solution by means of mathematical relations and consists of the following elements: the *objective function* $f(x)$, that is the quantity to minimize written as a function of the vector of *decision variables* $x \in \mathbb{R}^n$, and the inequality and equality *constraints* ($g_i(x)$ and $h_j(x)$, respectively) that, together with the *domain* X, describe the feasibility conditions of the solutions, defining the *feasible region* Ω of the problem

$$\Omega = \{x \in X \subseteq \mathbb{R}^n \mid g_i(x) \leq 0 \; \forall i \in I, \quad h_j(x) = 0 \; \forall j \in J\}. \tag{2.2}$$

Therefore, a *feasible solution* coincides with any point in the feasible region, representing the values of the decision variables in the solution. The feasible region defines the status of the minimization problem, which can be *infeasible* (if $\Omega = \emptyset$), *unbounded* (if $\forall \alpha \in \mathbb{R}$, $\exists x \in \Omega$ such that $f(x) < \alpha$), or *feasible* (otherwise). A feasible solution is an *optimal solution* if there are no better solutions in the feasible region, that is, $x^* \in \Omega$ is optimal if $\nexists x \in \Omega$ such that $f(x) < f(x^*)$.

Mathematical programming models in which both the objective function and the constraints are a linear expression of the decision variables belong to *linear* mathematical programming, and, depending on the nature of the decision variables, they can be classified into three categories:

- *Linear Programming* (LP), where all variables can take real values;
- *Integer Linear Programming* (ILP), where all variables can take integer values only;
- *Mixed Integer Linear Programming* (MILP), where some variables can take real values and others are allowed to take integer values only.

As we will see in the following, this distinction, based on the nature of the domain, is crucial to select the algorithm that can solve an optimization problem.

2.1.1 The Simplex Method

To solve an LP problem to optimality, there are several methods that can be used, and, among them, the *Simplex Method* is the most relevant in practice. This algorithm is based on geometrical observations related to the feasible region, which defines if an LP problem is either infeasible, unbounded, or it admits an optimal solution. In LP problems, thanks to the linearity of the constraints, the feasible region results in the intersection of a finite number of half-spaces in the domain of the decision variables, therefore resulting in a *polyhedron*. If the problem has a feasible region that is non empty and bounded, the corresponding polyhedron P is characterized by its *vertices*, and each point in P can be expressed as a *convex combination* of these vertices. This observation, together with the linearity of the objective function, brings to the following theorem:

Theorem 2.1.1 *Given an LP problem* $min\{c^T x : x \in P\}$ *with* P *non empty and bounded, the problem is feasible and there exists at least one optimal solution corresponding to a vertex.*

This theorem is one of the key points of the Simplex Method, since it allows the search of the optimal solution to be restricted to the vertices of the polyhedron.

A similar reasoning can be made from an algebraic point of view, starting from the fact that an LP problem can always be formulated in the *standard form*

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t.} \quad & Ax = b \\ & x \geq 0, \end{aligned} \tag{2.3}$$

where, as in (2.1), $x \in \mathbb{R}^n$ is the vector of decision variables, $c \in \mathbb{R}^n$ is the vector of *costs*, $b \in \mathbb{R}^m$ is the vector of *constant terms*, and $A \in \mathbb{R}^{n \times m}$ is the *matrix of constraint coefficients*.

Moreover, the standard form can be manipulated into another formulation through the notion of *basis of a matrix*. Assuming $m \leq n$, we define a basis of the matrix A as a subset of decision variable indices $B \subseteq \{1, \ldots, n\}$ of size m defining a sub-matrix $A_B \in \mathbb{R}^{m \times m}$ with full rank. Thus, we can retrieve the non-basis $N = \{1, \ldots, n\} \setminus B$, and rewrite the system of linear constraints as

$$Ax = \begin{bmatrix} A_B & A_N \end{bmatrix} \begin{bmatrix} x_B \\ x_N \end{bmatrix} = A_B x_B + A_N x_N = b,$$

meaning that a solution x is feasible if and only if $x_B = A_B^{-1} b - A_B^{-1} A_N x_N$. We define as the *basic solution* associated with the basis B the only solution of $Ax = b$ such that $x_N = 0$ and $x_B = A_B^{-1} b$. Similarly, the objective function $z := c^T x$ can be expressed as function of the variables out of the basis:

$$0 = z - c^T x = z - c_B^T x_B - c_N^T x_N \iff z - (c_N^T - c_B^T A_B^{-1} A_N) x_N = c_B^T A_B^{-1} b.$$

Therefore we can write the LP problem is its canonical form as follows

$$\begin{aligned} \min \quad & z \\ \text{s.t.} \quad & z - \bar{c}_N^T x_N = \bar{z} \\ & x_B + \bar{A}_N x_N = \bar{b} \\ & x \geq 0, \end{aligned} \tag{2.4}$$

where

$$\begin{aligned} &\bar{A}_N := A_B^{-1} A_N, \\ &\bar{b} := A_B^{-1} b, \\ &\bar{c}_N := c_N - A_N^T (A_B^{-1})^T c_B = c_N - \bar{A}_N^T c_B, \\ &\bar{z} := c_B^T A_B^{-1} b. \end{aligned}$$

The interest in the canonical form is motivated by the algebraic characterization of the vertices of a polyhedron described in the following theorem.

Theorem 2.1.2 *Given an LP problem with non empty and bounded feasible region, represented by the polyhedron P defined by the system of linear equations $Ax = b$, $x \geq 0$, we have that x is a feasible basic solution of the system of equations if and only if it is a vertex of the polyhedron P.*

The Simplex Method combines the results provided by Theorem 2.1.1 and Theorem 2.1.2: it starts from a feasible basic solution and searches for an optimal solution to an LP problem by visiting basic solutions until some

optimality or unboundness condition is met. Given a feasible solution and its related basis B, the algorithm looks for another basis obtained by switching a variable in B with a variable in N, in such a way that the new basic solution could improve the value of the objective function. From the canonical form associated with B in (2.4), we can observe that if every element of $\bar{c}_N$ is non-negative, then any variable entering the basis will not decrease the objective value (*optimality condition*). Otherwise, we select one variable x_k, $k \in N$ out of the basis associated with a negative cost $\bar{c}_k < 0$, and in order to let it enter the basis we increase its value as much as possible until one basic variable exits the basis (i.e., it resets to zero in the corresponding feasible solution). The value that x_k can reach is bounded by the positivity constraint on x_B, if no such threshold exists (that is when $\bar{a}_{ik} \leq 0$, $\forall i = 1, \ldots, m$), then the problem is unbounded (*unboundness condition*). When this condition is not satisfied, it can be proven that the best value for the variable to enter the basis is

$$x_k = \min_i \left\{ \frac{\bar{b}_i}{\bar{a}_{ik}} \mid i = 1, \ldots, m,\ \bar{a}_{ik} > 0\ \forall i \right\}.$$

The Simplex Method is summarized in Algorithm 1.

Algorithm 1 Simplex Method

```
1: procedure SimplexMethod(A, b, c, B)
2:     compute the canonical form and the basic solution x̄ related to B;
3:     if c̄ ≥ 0 then
4:         STOP: x̄ is an optimal solution.
5:     end if
6:     select k ∈ N such that c̄_k < 0;
7:     if ā_ik ≤ 0, ∀i = 1, ..., m then
8:         STOP: the problem is unbounded.
9:     end if
10:    select h ∈ B such that ā_hk > 0 and h = argmin{ b̄_h / ā_hk };
11:    update the basis: replace h with k in B;
12:    go to Step 2;
13: end procedure
```

2.2 Mixed Integer Linear Programming

If a problem can be formulated via LP, we have seen that one possibility to solve it is by using the Simplex Method. Nevertheless, in many situations we need to model and optimize discrete quantities, thus needing some decision variables to be subject to *integrality constraints*. These kind of problems are

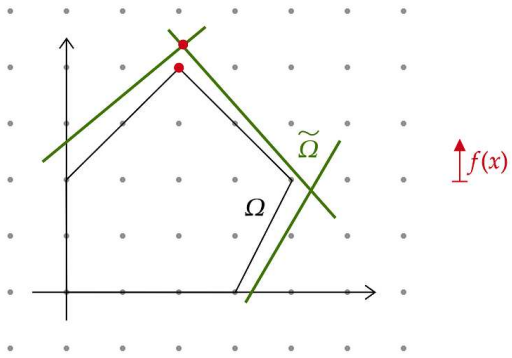


Figure 2.1: Example of the convex hull of the feasible region of a MILP problem (Ω, in black) and the LP relaxation of an associated formulation ($\tilde{\Omega}$, in green).

represented by MILP models, that can be formulated as

$$\begin{aligned} min \quad & c^T x \\ s.t. \quad & Ax \leq b \\ & x \geq 0 \\ & x_i \in \mathbb{Z}, \quad i \in I \end{aligned} \tag{2.5}$$

where, following the previous notation, $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$, and $I \subseteq \{1, \ldots, n\}$ is the index set of the integer decision variables (i.e., x_i with $i \in I$ is an integer variable, while x_i with $i \notin I$ is a continuous variable).

The *continuous relaxation* of a MILP model is obtained by removing the integrality constraints, retrieving an LP model that can be solved, for example, with the Simplex Method. Notice that the solution of the relaxed problem might have fractional values for integer decision variables in the original problem, leading to infeasibility. Nevertheless, there are some situations in which the optimal solution of an LP is integral, for example this happens when the feasible region has integer vertices. This geometrical property is guaranteed when the constraint matrix A satisfies the following algebraic property:

Definition 2.2.1 *A matrix A is a* Totally Unimodular Matrix (TUM) *if every square non-singular sub-matrix of A has determinant either* 1 *or* -1.

When the TUM condition is not satisfied, we can still hope to retrieve a feasible region characterized by integer vertices by exploiting the following observation. Let Ω be the smallest (in the sense of inclusion) convex set including the feasible region of a MILP problem, and let $\tilde{\Omega}$ be the feasible region of the LP relaxation of an associated formulation, by construction we have that $\Omega \subseteq \tilde{\Omega}$. Through Figure 2.1, we can observe that Ω can be seen as

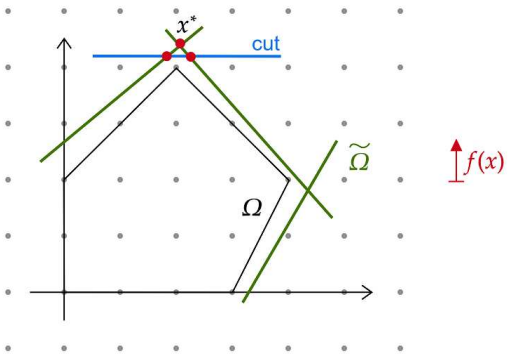


Figure 2.2: Example of a cutting plane (in blue) that cuts off x^*.

the *convex hull* of all the integer points in $\tilde{\Omega}$, that are all the integer-feasible solutions of the original problem.

2.2.1 Cutting Planes

To reproduce the geometrical conditions to exploit the Simplex Method to solve MILP problems, one possibility is to provide a MILP formulation of the problem such that $\tilde{\Omega} = \Omega$: this is the core idea of the *Cutting Planes* algorithm.

Definition 2.2.2 *A* cut *is an inequality $\alpha^T x \leq \beta$ which is valid for each $x \in \Omega$, and that is not valid for at least one point in $\tilde{\Omega} \setminus \Omega$ (i.e., $\exists \tilde{x} \in \tilde{\Omega} \setminus \Omega$ such that $\alpha^T \tilde{x} > \beta$).*

Algorithm 2 Cutting Planes

1: **procedure** CUTTINGPLANES($A, b, c, \Omega, \tilde{\Omega}$)
2: solve the LP relaxation $\min_{x \in \tilde{\Omega}} \{c^T x \mid Ax \leq b\}$ and get the solution x^*;
3: **if** $x^* \in \Omega$ **then**
4: STOP: x^* is an optimal solution to the original problem.
5: **end if**
6: compute a cut that is valid for Ω and cuts off x^*, and add it to $\tilde{\Omega}$;
7: go to Step 2;
8: **end procedure**

The Cutting Planes method (reported in Algorithm 2) is an iterative procedure that, at each iteration, adds a cut to the current set of constraints of the LP relaxation. A cut is a further constraint that is violated by the current optimal non-integer solution, but is valid for Ω. The aim of the iterations is to tighten the feasible region $\tilde{\Omega}$ (see Figure 2.2), until an optimal

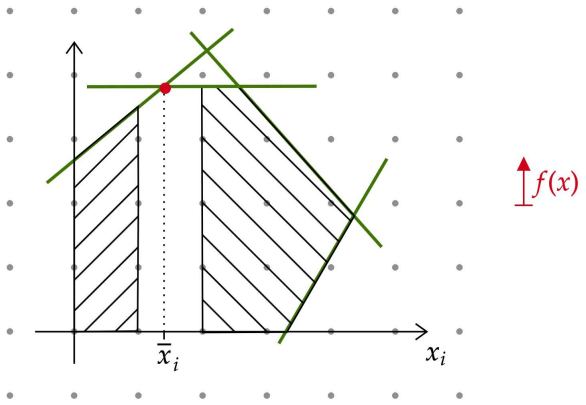


Figure 2.3: Example of feasible region branched on the fractional value $\bar{x}_i$ of the integer variable x_i .

solution is found among the vertices of Ω. Notice that the real challenge of this method consists in the separation problem, which is the problem of finding a suitable cut at each iteration.

2.2.2 Branch-and-Bound

An alternative approach to solve a MILP problem through the Simplex Method is the *Branch-and-Bound* (B&B) algorithm. This solving procedure is based on the idea of dividing the relaxed feasible region with respect to the integer variables of the original problem, and then optimize over the obtained sub-regions. This is supported by the fact that, given a set X and a partition of it $\{X_1, \ldots, X_k\}$, the following holds true:

$$\min_{x \in X} f(x) = \min_{i=1,\ldots,k} \left(\min_{x \in X_i} f(x) \right).$$

The B&B can be seen on a binary tree structure, where each node corresponds to an LP problem that is solved by the Simplex Method. If the solution found at one node has a fractional value for a variable that is integer in the original problem (e.g., $x_i = \bar{x}_i \notin \mathbb{Z}$), the *branching* operation is performed on the feasible region by splitting it into two polyhedra with respect to the value of the variable (e.g., one where $x_i \leq \lfloor \bar{x}_i \rfloor$, and the other one where $x_i \geq \lceil \bar{x}_i \rceil$), as represented in Figure 2.3. The branching process is recursively repeated until in one node we obtain an integer solution. To limit the search, B&B employs a *bounding* process that determines, for minimization problems, a lower bound to the value of the best integer feasible solution in the sub-tree rooted at a given node. For each node, a lower bound is determined by solving the relaxed problem. Moreover, a general upper

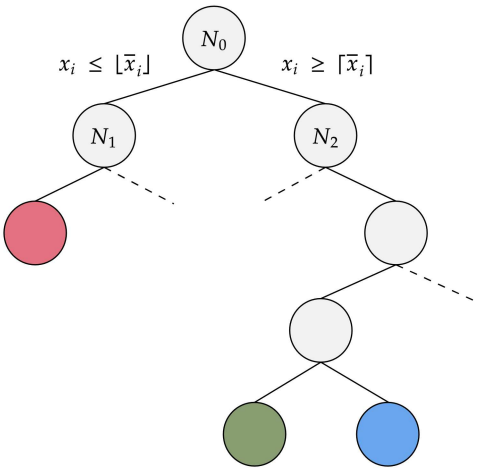


Figure 2.4: Branch-and-Bound search-tree scheme. The colored nodes are pruned by infeasibility (in red), bound (in green), and integrality (in blue).

bound is provided by the value of the best integer solution found during the search. Therefore, we can *prune* a node by:

- *infeasibility*: when the relaxed (and, hence, the integer) problem of the node is infeasible;

- *bound*: when the lower bound of the node is greater than or equal to the best upper bound found so far;

- *integrality*: when the solution of the relaxed problem is integer, thus feasible for the original problem and optimal for the sub-tree.

The algorithm terminates when all the leaves have been pruned, and the best integer solution found is the optimal solution to the MILP original problem.

The pseudo-code of the procedure can be found in Algorithm 3, and a schematic example of the search tree is given in Figure 2.4.

In addition, the B&B framework can be integrated with various extensions. For example, the Branch-and-Cut (B&C) algorithm combines the B&B with the Cutting Planes method, leveraging both branching strategies and cutting planes to strengthen the bounds. The key idea behind B&C is that, when solving the sub-problem associated with a node of the B&B tree, valid inequalities can be generated and added to the sub-problem to tighten the corresponding bound, thereby improving the pruning process, and facilitating the convergence of the algorithm.

Algorithm 3 Branch-and-Bound

procedure BRANCH-AND-BOUND(A, b, c)
  let (2.5) be the problem at the root node N_0;
  let X be the feasible region;
  let x^*, z^* be the current best solution and objective value;
  while there are unexplored nodes **do**
    select one unexplored node N_i;
    if the problem in N_i is infeasible **then**
      prune N_i by infeasibility;
    else
      solve the LP relaxation in N_i and get the objective value $\bar{z}$;
      if $\bar{z} \geq z^*$ **then**
        prune N_i by bound;
      else if $\bar{x}$ is integer **then**
        prune N_i by integrality;
        if $\bar{z} < z^*$ **then**
          update $z^* = \bar{z}$ and $x^* = \bar{x}$;
        end if
      else
        select a variable $x_i = \bar{x}_i \notin \mathbb{Z}$ and create two child nodes:
        N_{i+1} with feasible region $X \cap \{x_i \leq \lfloor \bar{x}_i \rfloor\}$;
        N_{i+2} with feasible region $X \cap \{x_i \geq \lceil \bar{x}_i \rceil\}$;
      end if
    end if
  end while
  STOP: x^* is an optimal solution to the original problem.
end procedure

2.3 Heuristic methods

The solution approaches that we have seen so far are able to provide an optimal solution to a problem, which is a feasible solution that minimize the objective function, and they fall into the category of exact methods. In some cases, the optimality certificate of the solution is required disregarding the cost to compute the optimal solution in terms, for example, of execution times. In fact, the drawback of exact methods is that generally they have exponential computational complexity, therefore due to the inner complexity of a problem, or due to the large size of the instance to solve, execution times can be extremely high. In real world applications, it is often the case that implementation resources and the available computational time are limited, while the instance to solve presents large scale data. In these cases, the optimality of the solution could be sacrificed in order to reduce the execution times. The solution approaches providing good feasible solutions

in a reasonable amount of time are called *heuristic methods*.

In the last decades, heuristic methods have gained a lot of interest, thus a wide range of algorithms of this type have been developed. We refer to heuristic method when the algorithm is devised for a specific problem, and to metaheuristic when the procedure is general enough to be applied to solve different problems.

Heuristic methods can be broadly divided in *constructive heuristics*, that build a solution incrementally, and *improving heuristics*, that start from one solution and iteratively try to improve it. On the other hand, metaheuristics can be classified with respect to their solution strategy. We cite here *search-based methods* (that explore the solution space looking for better solution than the current best one found), *population-based methods* (that iteratively start from a large set of solutions, i.e., population, and combines them to improve the quality of the population for the next generation), and *hybrid methods* (that combine aspects of different methodologies).

Among hybrid methods, we highlight the presence of the *matheuristics*, which combine a general metaheuristic with mathematical programming solution approaches (a detailed description can be found in [34, 63]). We have a dual advantage in combining these two techniques: we enhance the metaheuristic by solving to optimality one or more of its components, while discharging to the general framework the control of some constraints that are difficult or heavy to handle with the model.

Moreover, regardless its nature, a metaheuristic method represents a balance between two intertwined features of the search strategy: *intensification* (i.e., focus the search on promising regions) and *diversification* (i.e., explore different regions to sample more parts of the solution space).

2.3.1 Greedy algorithms

A *Greedy Algorithm* is a constructive heuristic that builds the solution one element at the time, by choosing at each step the best option to be added to the partial solution. At each iteration, the algorithm selects the option that provides the highest immediate benefit, without reconsidering previous choices, while typically preserving the feasibility of the partial solution. This type of algorithms represents one of the most intuitive practice to build a solution, since it focuses exclusively on expanding the partial solution at each step with the locally best additional element. Greedy algorithms are typically simple to implement, require low computational resources, and normally run in very short time, making them suitable for large-scale problems when rapid approximations are required. Despite their advantages, greedy algorithms often lack global insight into the solution space, as they focus on improving the current partial solution without exploring alternative paths to expand and complete a solution. As a result, the solutions obtained by a Greedy Algorithm are, in general, not guaranteed to be optimal, especially

in cases where the problem presents dependency relations among elements or constraints.

To overcome this issue, a *Randomized Greedy Algorithm* can introduce diversification by selecting the next element not strictly based on the best immediate option, but instead randomly choosing among the best $k \in \mathbb{Z}_{>0}$ options left. This randomized selection, combined with the repetition of the procedure, allows the algorithm to explore alternative solution expansion paths, enhancing the chances of finding better solutions.

2.3.2 Pilot method

In the (randomized) greedy algorithm, the ranking of the next element is based on the evaluation of the partial solution up to the current step, with respect to some criteria. Therefore, this kind of evaluation is considering only the elements that have already been chosen and the one under evaluation, ignoring the remaining elements. The *Pilot Method* aims at improving this aspect by introducing a look-ahead on the final solution during the evaluation [28].

The Pilot Method follows a structure similar to the one of a greedy algorithm but differs in its approach to evaluate candidates: instead of selecting the element that minimizes the immediate impact on the current partial solution, it evaluates each candidate based on its projected impact on the final solution, e.g., by considering a possible path to a complete solution, rather than the related partial solution. This look-ahead evaluation is achieved by temporarily incorporating the candidate element into the partial solution and conducting a deeper evaluation, often by applying a heuristic to approximate the effect of the choice on the corresponding solution quality. By simulating how each candidate would influence the final solution, the algorithm gains information to better build the partial solution at each step.

Notice that, in a certain sense, the look-ahead gives diversification to the search in the solution space, since the choice of the next element to add is not just guided by the partial solution and by the element itself, but it is based on some exploration of different prospective solutions. This method effectively balances intensification and diversification by blending a greedy selection with limited exploration of alternative future outcomes. Through this mechanism, the Pilot Method provides a more comprehensive search of the solution space and can outperform purely greedy approaches in terms of solution quality, especially in complex combinatorial problems.

2.3.3 Beam search

The *Beam Search* is a constructive heuristic that allows to search in parallel multiple solutions through a tree structure. In particular, the solution space is, in principle, represented by a tree where the root is the empty solu-

tion, and each leave represents one possible complete solution. Intermediate nodes correspond to partial solutions. The children of each (root or intermediate) node correspond to the expansion of the related partial solution with any possible additional element. Unlike greedy algorithms, which, in this representation, follow a single path from the root to a leaf to construct a solution, Beam Search evaluates and expands multiple paths in parallel at each level, increasing the chances of finding a high-quality solution. The algorithm is characterized by two parameters: the *filter width* α, which defines the number of child nodes kept from each parent, and the *beam width* β, which controls the number of nodes at the same level that are expanded in the next level of the search tree.

Assuming that a solution consists of a combination of elements in a given set, each node of the tree is obtained by adding one element, and the path from it up to the root node corresponds to the partial solution obtained by adding the elements along the path itself, or to a complete solution if the node is a leaf. The tree structure is built by a *breadth-first strategy*: at each level, every node is expanded according to the feasible remaining elements, which are the ones that are not in the path from the root, and that preserve the feasibility of the related partial solution. All child nodes are evaluated by the *filtering evaluation* criterion and the best α for each parent node are kept, while the others are discarded. This process, known as *filtering*, is optional and is based on a relatively quick evaluation of each child node, in order to keep limited the number of partial solutions that will be further processed.

Subsequently, all the filtered nodes at the current level are subject to a normally more accurate *beam selection evaluation*: the best β child nodes of the current level are called the beam nodes, and they are the ones that will be considered in the next level, while the other nodes are pruned by *beam selection*. This selective expansion ensures that only the most promising solutions are explored and avoids an exponential increase in the number of processed tree nodes, thus balancing computational efficiency with solution quality. The algorithm stops when the beam nodes of the last level are all leaves, that is when there are no more elements to be inserted in the explored solutions. Each leaf of the partially explored search tree represents one of the generated complete solutions, and the best one according to the objective function of the problem is returned.

Notice that the Beam Search can be tuned with respect to its usage by adjusting α and β, thus controlling the trade-off between exploration and computational resources, as larger beam widths yield a more exhaustive search, but at a higher computational cost.

A graphic example of the tree construction is given in Figure 2.5, where the filter width is $\alpha = 3$, and the beam width is $\beta = 2$.

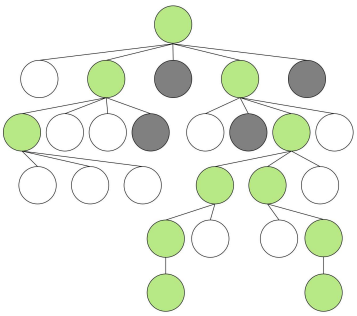


Figure 2.5: Example of beam search tree with filter width $\alpha = 3$, and beam width $\beta = 2$: beam nodes in green, nodes pruned by filtering in gray, and nodes pruned by beam selection in white.

2.4 The lexicographic method

In the previous sections, we have seen how we can compute a solution to a problem characterized by one objective function, and a set of constraints on the domain of the decision variables. Nevertheless, in many problems, we can have more than one objective to optimize over the feasible region, in these cases we talk about *Multi-Objective Optimization*. The different objectives could be in conflict with each other, for example, when buying a computer, we want to minimize the cost and maximize the performance, thus the optimal solution represents a trade-off between them. In 2DC&PP-PT considered in this thesis, the goal is to generate a solution that minimizes the material waste and minimizes the number of soft precedence violations, which could potentially increase the waste.

We focus on the specific case where the objectives can be sorted with respect to their importance in the optimization, in the previous example, for instance, we could want to first minimize the cost and then, among the computers at the same minimum price, buy the one that maximize the performance. Similarly, in 2DC&PP-PT, the request is to select, among solutions with the same minimum material waste, the one that minimizes the soft precedence violations. This kind of optimization is called *Lexicographic Optimization*.

The *Lexicographic Method* [77] is a general algorithm to solve a lexicographic optimization problem that is formulated as

$$\begin{aligned} min \quad & f_1(x), \ldots, f_k(x) \\ s.t. \quad & x \in X \end{aligned} \tag{2.7}$$

where $f_1(x), \ldots, f_k(x)$ are the objective functions to minimize sorted from the most important to the least important. The core idea of the algorithm is to solve an ordered sequence of k single-objective problems by constraining

Algorithm 4 Lexicographic Method

procedure LEXICOGRAPHICMETHOD($f_1, \ldots, f_k, X$)
  let (2.7) be the problem to solve;
  let x^*, $z^* \in \mathbb{R}^k$ be the vector of the best solutions and values;
  for $i = 1, \ldots, k$ **do**
    solve the single-objective problem

$$\begin{aligned} min \quad & f_i(x) \\ s.t. \quad & x \in X \\ & f_j(x) \leq z_j^* \ \ \forall j = 1, \ldots, i-1; \end{aligned} \tag{2.6}$$

    if (2.6) is infeasible or unbounded **then**
      STOP: no solution to the original problem.
    else
      let x_i^*, z_i^* be a solution and objective value of (2.6);
    end if
  end for
  STOP: x_k^* is a solution to the original problem.
end procedure

the solution of each iteration to consider the optimal value obtained in the previous iterations. In other words, at iteration t we minimize the objective f_t, and we bound the solution to the optimal values of the higher importance objectives (i.e., $f_i \ \forall i < t$), as shown in Algorithm 4.

Chapter 3

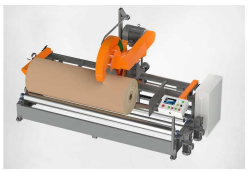
Problem definition

Cutting and Packing Problems (C&PP) arise in different real-world contexts, such as in sheet metal, wood, or glass industry, as well as in freight transportation and logistics. We call C&PP all the problems where a set of small items has to be assigned to larger objects by optimizing some objective function (e.g., minimize the number of used large objects). Based on the application field, there can be several extra requirements that must be satisfied by the assignment, and they generally are related to the shape of both the small items and the large objects, or to the mutual position of the small items within the large objects.

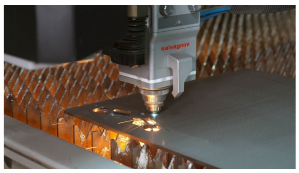
In this thesis, we consider a *Two-Dimensional Cutting and Packing Problem* (2DC&PP) representative of a common challenge in the sheet metal industry. Referring to Salvagnini Italia S.p.A., we will consider a problem related to punching machines, one of the cutting technologies developed by the company. The chapter is dedicated to an introduction to the general 2DC&PP (Section 3.1) and to the application context of this work in the sheet metal industry (Section 3.2), allowing for a detailed description and definition of the problem object of the thesis (Section 3.3).

3.1 Introduction to the Two-Dimensional Cutting and Packing Problem (2DC&PP)

In the field of Operations Research, C&PP form a well-known class of combinatorial optimization problems of practical relevance. They involve the assignment of smaller items to larger objects, generally with the aim of maximizing the utilization of the large objects or minimizing their waste. C&PP can differ from each other by a huge variety of features. The dimension of both small items and large objects, for instance, distinguishes between *One-*, *Two-*, and *Three-Dimensional C&PP* (Figure 3.1). One-dimensional problems aim at minimizing, e.g., the number of rolls, or rods, used to ac-



(a) 1DC&PP



(b) 2DC&PP

(c) 3DC&PP

Figure 3.1: Examples of one-, two-, and three-dimensional C&PP in the paper, sheet metal, and freight transportation industries, respectively.

commodate items of varying dimension, as seen in the paper industry where rolls are cut to obtain the required lengths. The two-dimensional case deals with cutting items from material sheets by reducing leftovers, such as in the wood, glass, and sheet metal industries. Three-dimensional problems concern, e.g., efficiently loading items into the minimum number of containers, and they are commonly faced in freight transportation where the goal is to reduce the number of containers needed for shipping.

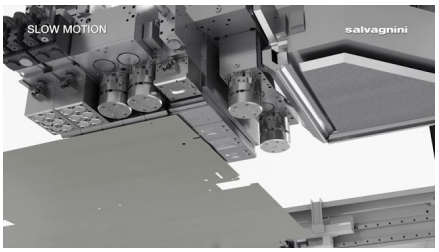
Hereinafter, we will concentrate on the two-dimensional class of C&PP, which is the focus of this thesis. In most 2DC&PP, a direct consequence of the fact that items are cut from the material sheets is that, when building a cutting layout by placing the items within the sheet, items cannot overlap with each other and must be fully contained in the sheet. Therefore, the optimization task of a 2DC&PP generally brings along the geometric challenge of avoiding overlap between shapes in the plane. The geometric constraints and the combinatorial nature of these kind of problems make them hard to be solved to optimality in most cases [32]. Moreover, depending on the application, the size of the instances can widely vary and become very large, thus several metaheuristic approaches have been developed to tackle this class of problems, as well as some problem-specific heuristics which exploit the practical characteristics of the application.

In the following section, we will describe the context in which the problem addressed in this thesis arises.

3.2 Real-world cutting problems in the sheet metal industry

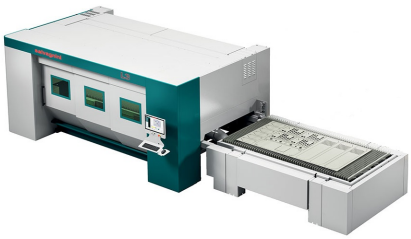
The problem under analysis in this work arises in the sheet metal industry. In this field of application, we specifically refer to Salvagnini Italia S.p.A. (shortened in Salvagnini from now on), which is the headquarter of the multinational corporation represented by Salvagnini Group [81]. The company, based in the north of Italy, designs, produces and commercializes *Flexible Manufacturing Systems* (FMS) consisting of *Computerized Numerical Control* (CNC) machines to cut and bend the sheet metal, along with

(a) Punching machine.

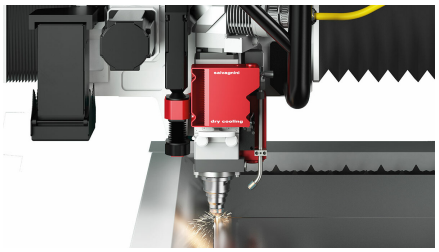


(b) Punching head and unloading shear.

Figure 3.2: Punching technology [81].



(a) Laser cutting machine.



(b) Laser cutting head.

Figure 3.3: Laser cutting technology [81].

dedicated software. The processes in a Salvagnini FMS are tailored to ensure customized production, while maximizing productivity and minimizing material waste and consumption, therefore in this context lots of optimization problems arise and are handled by the *Research and Development* (R&D) department. On the one hand, machines are equipped with an increasing number of automated processes that eliminate production downtime and reduce the chances of error. On the other hand, a big part of the optimization is handled at the software level, through the development of CAD (*Computer-Aided Design*) and CAM (*Computer-Aided Manufacturing*) applications. These applications assist operators with automatic tools to address challenging problems, such as how a machine should process metal sheets, the order in which operations should be performed, and so on.

In this thesis, we focus on cutting machines, and Salvagnini produces essentially two types of cutting machines, that are equipped with different cutting technologies: *punching* and *laser cutting*, shown in Figure 3.2a and in Figure 3.3a, respectively. Punching technology involves using a machine provided with a *punching head* that houses several rotating tools to punch holes in the sheet metal by applying high pressure with the *punching tools*. Moreover, Salvagnini punching machines present an integrated *unloading shear* composed of two blades, which are independent and orthogonal to each other. The shear is able to cut any length and it is used to detach

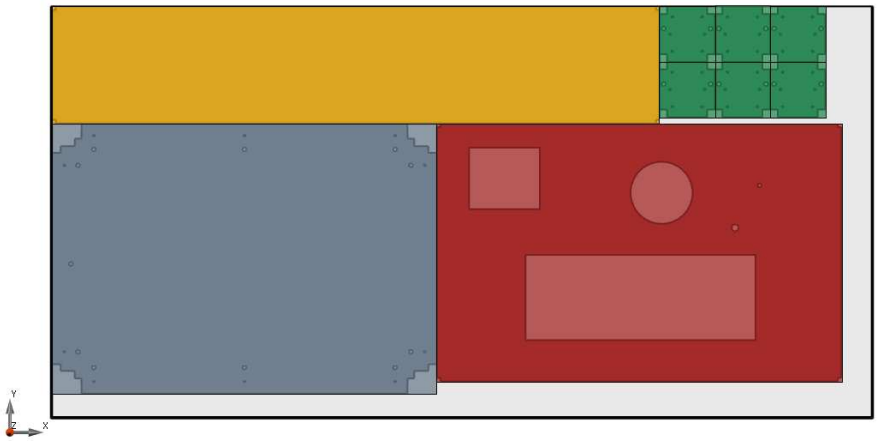


Figure 3.4: Example of a cutting layout for punching machines.

each processed part from the sheet, as will be detailed in the following (see Figure 3.2b for a bottom view of the punching head and the unloading shear). This cutting technology is highly efficient for producing different types of part. On the other hand, laser cutting machines have a single optical head integrated with a dry-cooling system to control the temperature of the optics (represented in Figure 3.3b) to guarantee extremely precise cuts. The precision of the laser cutting allows to produce intricate shapes with fine details with high accuracy.

We provide a brief overview of the production process where punching machines are involved in. Notice that a different process would be required when laser cutting machines are used. The cutting process is triggered by the client orders, including a list of parts that must be produced. These parts have to be cut from a specified set of material sheets available in stock. We consider sheets with a rectangular shape, which can be of different sizes. The sheets are loaded into the machine, one at a time, to be processed for cutting the required parts. When a sheet is loaded, it is centered within the machine, establishing a coordinate system where the axes align with the sheet's edges.

For the cutting process, a set of *cutting layouts* must be created (an example is provided in Figure 3.4), arranging all the parts on the available sheets while meeting both geometrical and technological constraints, as the ones that will be described in the following. Each layout is optimized to fit as many parts as possible, maximizing material use and minimizing waste. Once all required parts are placed, any remaining material can be used to produce extra parts, included in a list of optional parts useful for future orders or for internal use.

Each part is produced in two stages. First, the punching tools shape the

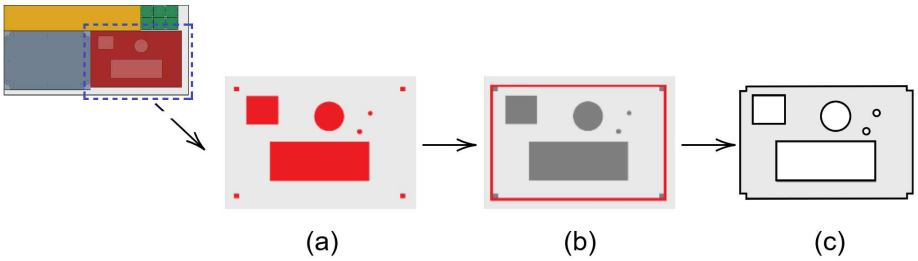


Figure 3.5: Example of the cutting process of one part - the punching tools shape the part (a), then the unloading shear cuts the minimum rectangle containing the part (b) to detach it from the sheet and obtain the final object (c).

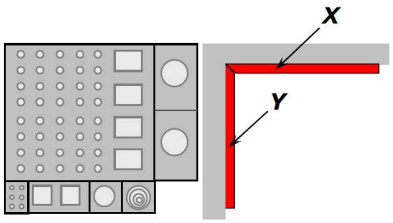
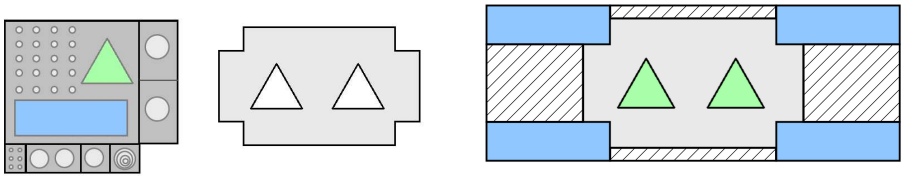


Figure 3.6: Representation of the cutting station - the punching head in dark gray with the punching tools in light gray (on the left) and the unloading shear in red (on the right).

part, defining its inner contours (if any) and any outer detail that falls inside the minimum (in the sense of inclusion) axis-aligned rectangle containing the part. In the second stage, the unloading shear cuts along the sides of this minimum rectangle to detach the part. This cutting method requires each part to have at least the space defined by its rectangle reserved on the sheet, ensuring that no overlap occurs with the rectangle reserved for other parts.

An example of the two-stage cutting process for a single part is shown in Figures 3.5 and 3.6. This example refers to the cutting layout shown in Figure 3.4 and reported in the top-left corner of Figure 3.5. In particular, the example considers the dark red part included in the cutting layout and highlighted by the dotted blue rectangle in Figure 3.5. The cutting station used in this process is illustrated in Figure 3.6. On the left side of the figure, the punching head (in dark gray) is shown, housing the punching tools (in light gray) of various shapes and sizes. On the right side, the unloading shear is outlined in red, with the blades aligned according to the sheet's coordinate system. The cutting process begins in Figure 3.5a, where holes (highlighted in red) are punched into the sheet metal, defining the contours and interior details within the bounding rectangle of the part. In Figure 3.5b, the unloading shear cuts along the axis-aligned edges of the bounding



(a) (b)

Figure 3.7: Example of buffer zones of a part - given the punching head and the part to produce in (a), the highlighted punching tools are required. When shaping the part, as in (b), the blue punching tool exceeds the bounding rectangle of the part, thus forming the buffer dashed zones.

rectangle (highlighted in red), allowing the part to be detached from the sheet and unloaded from the machine. This produces the final object, as shown in Figure 3.5c.

During the punching process, the tools may extend beyond the bounding rectangle of the part, necessitating a buffer zone to maintain cut quality and avoid damaging other parts. In Figure 3.7a, we see an example of a punching head (on the left) and a part to be cut with it (on the right). To shape the corners of the part, the only usable punching tool is the one highlighted in blue. As shown in Figure 3.7b, the punching tool extends beyond the bounding rectangle of the part, meaning that no other parts can be obtained from the exceeding zones (marked as the dashed areas). When the buffer zone is not required, the rectangles can be positioned adjacent to each other, allowing the shear to cut along their shared side in one pass. To prevent deformation of the sheet metal, the shear is used for detaching parts with minimum dimensions that are related to the material's thickness, avoiding bending.

Furthermore, the reference cutting machines allows cutting rotated items. Because of the oriented shears, parts can only be placed on the sheets with 90 degrees rotations to maintain the axis-alignment of their minimum enclosing rectangles. Additionally, the punching tools on the machine's punching head can rotate to a set of predefined angles, which may further restrict the allowed rotations for each part. Specifically, the rotation of a part depends on the orientations supported by the punching tools required for its cut. Moreover, while the punching head moves across the sheet, it cannot rotate itself, meaning not all tools can reach every area, which further limits the placement of the parts. In the previous example shown in Figure 3.7, we can observe that, to produce the inner holes of the part, the punching tool highlighted in green is strictly required. In this case, if the punching tool cannot be rotated, as a consequence, the part cannot be rotated too, as otherwise its rotation would prevent the ability to cut the inner triangular

contours.

Once the cutting layouts are defined, they are elaborated to decide the operations that the cutting machine should perform to produce the parts in each sheet. The actual production begins with the loading of material sheets into machines, proceeds with cutting the sheet with respect to the related layout, and ends with the unloading of the finished parts. To this end, we notice that each part may belong to a kit, i.e., a set of parts that will be later processed together to obtain a same final product (e.g., the components of a shelf). Once unloaded, parts that belong to the same kit are grouped and stored for downstream operations, like bending and assembly. The definition of kits has impact on both the design of the cutting layouts and on the sequence they are processed by the machine, since parts of the same group must be cut and unloaded consecutively, according to a given order. On the other hand, even parts that are not grouped (i.e., single-part products) may be prioritized to ensure the production of certain items (e.g., the ones belonging to a priority order) in view of potential interruptions or delays in the cutting process. However, these priorities are less strict than those related to part grouping, and have to be satisfied only when this does not increase material waste. More in general, the cutting process may require hard precedence or soft precedence among parts. The hard precedence notion is driven by operations that follow the cutting phase (e.g., unloading, storage, or assembly of the items), while soft precedence relations are motivated by the need to prioritize the production of certain items (e.g., belonging to a priority order) to account for potential interruptions or delays in the cutting process.

In the multitude of decision and optimization challenges that emerge in this production process, we focus on the problem of creating the cutting layouts and defining their processing sequence, accounting for all the related relevant features. It is indeed a multi-attribute 2DC&PP, and its definition will be specified in the following section.

To address this problem, the reference company currently relies on an outsourced commercial software under license. This software is capable of generating cutting layouts for both punching and laser cutting technologies, while effectively managing the associated technological constraints. Although the software operates as a black box, it offers user-configurable parameters, that allow to generate solutions according to the user's needs or preferences in terms of, for example, execution times. The software has proven to be highly reliable, consistently delivering high-quality solutions even with short time limits, as will be discussed in Chapter 8.

3.3 The 2DC&PP with Punching Technology (2DC&PP-PT)

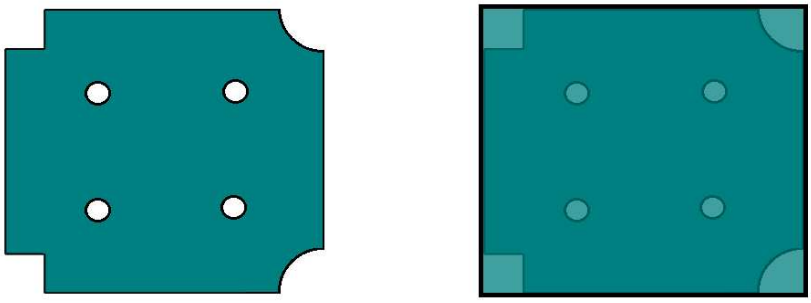
The 2DC&PP we consider in this thesis is related to the punching machines described in the previous section, and aims at efficiently placing a set of parts to be produced in the available material sheets by minimizing the waste, while respecting the technological constraints imposed by the cutting machine. Notice that, as mentioned in the previous section, the same general problem applied to a different cutting machine would require different additional constraints to guarantee the operation of the machine itself, and the desired quality of the final products.

Sheet types

In our case, the cutting machine can accommodate rectangular *sheets* of different sizes (with width between $370mm$ and $4064mm$, and height between $300mm$ and $1650mm$) and different thickness, which can range from $0.5mm$ to a maximum that depends on the material (e.g., $5mm$ if the material to cut is aluminum). Each sheet, once loaded into the machine, induces a coordinate system for the placement, where the axes are aligned with the sheet's edges. A limited or unlimited number of available sheets can be assigned to each sheet type: in the first case the user already dispose of that quantity, while in the second case they will consider the number of used sheets of that type in the final solution as an indication on the quantity of sheets to purchase for the current production batch.

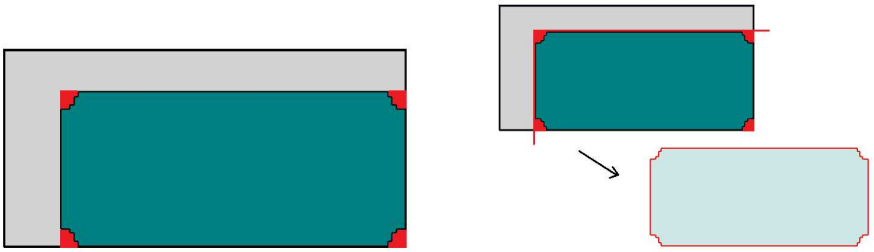
Item shape

Due to the presence of the integrated unloading shear, in our problem we have to represent each part to be produced with the minimum axis-aligned



(a) (b)

Figure 3.8: Example of a part (a), and its corresponding item (b).



(a) Shape the part on the sheet. (b) Detach the part and finalize it.

Figure 3.9: Example - production of a part: first the sheet is punched to shape the part (a), and then the part is detached by axis-aligned shear cuts (b), punches and cuts are highlighted in red.

rectangle containing it, i.e., its axis-aligned bounding box. An *item* is the axis-aligned bounding box of a *part*. A graphic example of the item related to a part is given in Figure 3.8.

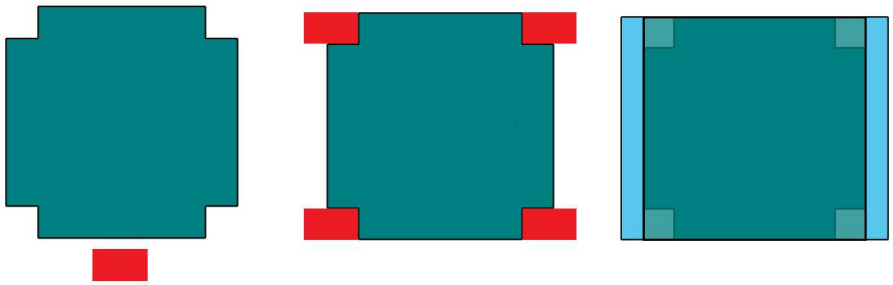
As previously discussed, this concept is necessary because, to produce a part, first the sheet metal is processed with the punching tools to shape the part itself, and then the shear detaches it from the sheet with axis-aligned cuts defined by its bounding box, which corresponds to the edges of the related item (see Figure 3.9 for a graphic example of the two steps). Therefore the axis-aligned bounding box of the parts cannot overlap, which requires us to consider each part as its corresponding rectangular item, defined by its width and height.

Compulsory and optional items

An item can be either compulsory or optional to cut: when it is compulsory, it is strictly required in the current production batch, while, if it is optional, it means that it belongs to future orders, but it can be produced in advance to exploit the material that would be wasted. Notice that no sheet can be used to produce only optional items, since, by definition, they must be considered exclusively to reduce the waste generated by the compulsory items production.

Item positioning

Items can be placed at any (or some) 90 degrees rotation, depending on the rotation angles assigned to the punching tools required for their cut. In addition, they can be assigned with requirements on the placement zone within the sheet. Namely, forbidden or mandatory placement zones can be imposed to the items due to production needs. For example, if a tool of the punching head can reach only certain areas of the sheet, the items that need



(a) (b) (c)

Figure 3.10: Example - punching margins of a part: to produce the teal part with the red punching tool in (a), we necessarily perform four holes as in (b), thus the part, and the corresponding item, must have left and right punching margins as represented in light blue in (c).

that tool must be placed accordingly. As regards the reciprocal position of the items within the sheet, as usual in the 2DC&PP, items cannot overlap with each other.

Punching margins

Due to the punching technology, as discussed in Section 3.2, a different distance margin may be required by each side of an item, representing the minimum distance between that side and any other item in the sheet. This attribute is intended to guarantee the quality of the products, since it can happen that to shape the side of a part, the used punching tool could exceed the side itself, potentially damaging other close parts. We refer to this distance as the *punching margin*.

Looking at the example in Figure 3.10, we can see that to produce the left and the right side of the part with the given punching tool, we necessarily exceed the sides of the axis-aligned bounding box (i.e., the related item), thus a punching margin corresponding to exceeding depth of the punching tool will be assigned to the left and to the right side of the item. We highlight that each punching margin is strictly related to its side, indirectly due to the punching tools required to process that side of the original part. Therefore, if an item is rotated, its punching margins are rotated accordingly, as the related punching tools. Thanks to the motivations behind the definition of the punching margins, we can overlap the punching margins of different items as long as we maintain the distance specified by the largest punching margin between the sides of the adjacent items. This explains why we cannot simply consider items inflated with respect to their punching margins, since it would prevent the possibility of overlapping the punching margins to reduce the material waste.

Common cut and safety margins

If two adjacent items have no assigned punching margins, they are allowed to either share a side with each other (in this case we say that they *share a common cut*), or they must respect a safety distance (called *safety margin*) that depends on the sheet type (normally, three times the sheet's thickness). The safety margin is necessary to avoid the risk of creating thin fillets between items, that would cause deformation of the material when detaching them with the shear cuts.

Hard and soft precedence

Due to the production process sketched in Section 3.2, each compulsory item may be assigned with a different level of precedence, imposing that items with higher precedence cannot be placed on sheets that are cut after those containing items with lower precedence. Notice that, due to this attribute, our solutions will be formed by a sequence of sheets, each with its own set of placed items, establishing a production order among them. We recall that, as explained in Section 3.2, in our framework, the precedence constraint can be either *hard*, meaning it must be strictly adhered to, or *soft*, meaning it should be satisfied only if it does not increase material waste. Notice that, since optional items are produced to utilize material waste and are not part of the required batch, they are not subject to precedence relations.

Objective function

The goal of the optimization is to find a feasible solution, defined as a sequence of cutting layouts (i.e., a sheet type with the related set of placed items), that minimizes the total amount of material waste. The material waste can be defined in various ways, depending on the attributes considered in the optimization process. To ensure a fair comparison of the results, we have adopted the definition used internally by the reference company. Waste is computed as the total waste from the used sheets, which is defined as the area of each sheet minus the area occupied by the placed items. Even if margins are mandatory and can overlap, we do not to include their area directly in the material waste. This approach aligns with the reference company's method, where margins are handled indirectly by the overall optimization process. In fact, in general, greater overlap tends to increase the chances of placing more items onto a single sheet.

Problem statement

The attributes described above yield the definition of the *Two-Dimensional Cutting and Packing Problem with Punching Technology* (2DC&PP-PT). A schematic definition of the 2DC&PP-PT is given in the following.

- **Input.**
  - Sheet types data: width, height, thickness, safety margin, available quantity.
  - Compulsory items data: width, height, allowed rotations, forbidden/mandatory placement zones, punching margins, hard/soft precedence level.
  - Optional items data: width, height, allowed rotations, forbidden/mandatory placement zones, punching margins.
- **Output.** A solution defined as a sequence of cutting layouts.
- **Hard constraints.** A feasible solution must respect:
  - sheet types maximum quantity constraints;
  - non-overlapping constraints;
  - punching margins constraints;
  - common cut or safety margin constraints;
  - compulsory items assignment constraints;
  - optional items assignment constraints;
  - hard precedence constraints;
  - allowed rotations constraints;
  - forbidden zones constraints;
  - mandatory zones constraints.
- **Soft constraints.** A feasible solution should, the amount of material waste being the same, respect the soft precedence constraints.
- **Objective function.** Minimization of the total amount of material waste.

Chapter 4

State of the Art

The problem described in Chapter 3 is a specific variant of a C&PP that presents additional attributes deriving from the application context. In this chapter, in Section 4.1, we will describe C&PP in general, together with the main variants that can be found in the literature. In particular, a possible classification of this set of problems, as proposed in [88], will be presented, and the specific class of *Two-Dimensional Bin Packing Problems* (2DBPP) will be further discussed in Subsection 4.1.1. In fact, the problem object of this thesis, at its core, falls into this 2DBPP class and, in Section 4.2, related basic mathematical formulations will be reviewed, along with the formulation of some specific constraints relevant to our application. Throughout Section 4.1 and 4.2, we will locate 2DC&PP-PT, the problem object of the thesis, in the Operations Research literature on 2DBPP, highlighting the differences between our problem and the ones in the cited works. In the end, we give an overview of the most common heuristic methods (Section 4.3), that are widely used to solve this class of problems.

4.1 Cutting and packing problems: definition and typology

C&PP define a well-known area in combinatorial optimization, with numerous real-world applications from logistics to manufacturing and materials processing, as we have introduced in Section 3.1. At their core, these problems share a common structure. We consider two sets of elements: a set of *large objects* (representing input or supply) and a set of *smaller items* (representing output or demand). The task is to select some or all of the small items, group them into subsets, and *assign* each subset to one of the large objects. This assignment must satisfy specific geometric constraints, such as ensuring that all small items in a subset fit entirely within their assigned large object without overlapping. The common elements and processes of

C&PP can have different characteristics based on the application context, and this leads to several variants of the problem belonging to this class. Moreover, a given *objective function* must be optimized.

From about 1980, C&PP started to gain a lot of attention in the scientific literature due to their versatility in the real-world applications, that could go from Computer Science, Engineering Industry, up to Management Science. The need for a classification of this class of problems was first satisfied by Dyckhoff in [29], where he proposed a typology of cutting and packing problems. This work let the previous literature, as well as the future research work, related to C&PP to be organized by a common terminology, thus gathering approaches used to solve the same problem in different fields of application. Nevertheless, with the passing of years, new characteristics arose, bringing new developments that were no longer fitting into Dyckhoff's typology. To account for this evolution, Wäscher et al. in 2007 [88], proposed an improved typology of C&PP, based on the previous one by Dyckhoff. Here we present the main criteria of the later typology which defines the basic problem types of cutting and packing problems, details can be found in the original work [88].

- **Kind of assignment**

  The assignment of small items to large objects can follow two alternative policies. On the one hand, there is the *output maximization*, where the available large objects are insufficient to accommodate all small items, so the goal is to assign a selection of items with the highest value to the large objects. Therefore, there is no need to decide which large objects to use since all of them are utilized. On the other hand, we can aim to pursue the *input minimization*, where the large objects can accommodate all the small items. The objective in this case is to assign all small items to a selected subset of large objects with the minimum value, thus all small items are used, so there is no selection problem regarding them. The terms *output maximization* and *input minimization* are used broadly and can represent various measures, such as cost, revenue, or material quantities. Notice that, in real-world situations or academic literature, there may be cases where both the large objects and small items are subject to selection, as well as problems which may involve multiple objectives, all these situations are considered as problem variants in this typology.

- **Assortment of small items**

  Small items are distinguished in three categories. First, *identical small items*, where all small items have the same shape and size across relevant dimensions (e.g., length, width, height). Second, *weakly heterogeneous assortment*, where the small items can be grouped into a few distinct classes, where items within each class share the same

shape and size. Notice that different orientations of the same shape are considered separated item types. Third, *strongly heterogeneous assortment*, in this scenario, most small items differ in shape and size from each other, with only a few identical elements, thus items are treated as individual elements, each with a demand of one. In basic types definition, these three categories are mutually exclusive, problems with a mixed set of weakly and strongly heterogeneous items are considered as variants.

- **Assortment of large objects**

  Large objects are distinguished based on their cardinality, there can be *one large object*, whose size can either be fixed or variable in all relevant dimensions, or *several large objects*, in this case only fixed dimensions are considered in most literature. Similar to the assortment of small items, large objects can be classified as identical, weakly heterogeneous, or strongly heterogeneous. In Wäscher et al. basic typology, large objects are assumed to be rectangular (e.g., rectangles and cuboids in two- and three-dimensional cases, respectively), while non-rectangular large objects are considered as variants.

The basic types of C&PP are characterized by a combination of these criteria. The possible combinations are displayed in Figure 4.1 and lead to the following categories: Identical Item Packing Problem, Placement Problem, Knapsack Problem, Open Dimension Problem, Cutting Stock Problem, and Bin Packing Problem.

These basic problem types can be further classified by the following criterion.

- **Dimensionality**

  The problem can be one-, two-, or three- dimensional (denoted by 1D, 2D, and 3D, respectively), imposing the dimensionality of the small items and of the large objects, which can be the same for consistency. Rarely in the literature the case with more than three dimensions is considered (e.g., [57]).

We focus on the two-dimensional case, of which a detailed discussion can be found in the work by Lodi et al. [58]. Furthermore, we draw the attention to the category of *Two-Dimensional Bin Packing Problems* (2DBPP) [59], since the problem object of this thesis, and presented in Chapter 3, belongs to it. In the following subsection will be given an overview on the possible variants of this problem that have been explored in the literature and are relevant to our study.

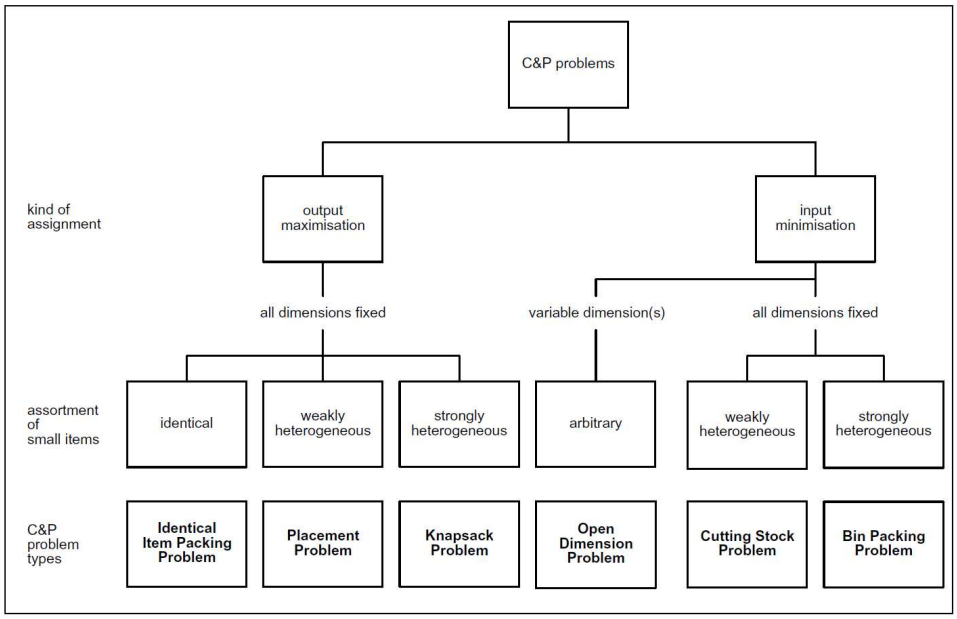


Figure 4.1: Basic C&PP typology proposed by Wäscher et al. [88].

4.1.1 Variants of Two-Dimensional Bin Packing Problem

The 2DBPP is, as previously described, a special class of cutting and packing problems where the aim of the optimization is the input maximization, small items (here *items*) are two-dimensional and strongly heterogeneous, and there are several large objects (here *bins*) with two fixed relevant dimensions. Several variants can be devised from this class of problems, we now describe separately the most common traits that defines these variant, but notice that they could be combined together yielding more complex situations.

Wäscher et al. [88] proposed another criterion to be considered in the classification of C&PP, which holds for every basic type, but that we now present restricted to the 2DBPP category.

- **Shape of items**

  Items can be either regular (e.g., rectangles, or circles) or irregular (i.e., non-regular, mostly general polygons). In the two-dimensional case, regular items are sometimes further classified as rectangular, circular, or other specific shapes.

Whenever the items are classified with respect to their shape, the indication of the shape is generally specified between the dimensionality and the basic type (e.g., Two-Dimensional *Rectangular* Bin Packing Problem). A thorough review on research topics, applications, and scientific literature on the Rectangular Bin Packing Problem up to 2023 can be found in [65].

As previously mentioned in Section 4.1, the assortment of the bins can lead to different definitions of the same problem. In particular, there can be only one unique type of bin (i.e., *identical*), or more than one type, each with a certain multiplicity (i.e., *weakly heterogeneous*), or almost all bins can be different from one another (i.e., *strongly heterogeneous*). This attribute is usually highlighted in the name of the problem, and can be rephrased with respect to the category. For example, in the case of 2DBPP with weakly heterogeneous bins, we refer to the problem as *Two-Dimensional Variable Sized Bin Packing Problem* (2DVSBPP).

With the increasing number of applications of cutting and packing problems, researchers have been challenged to account for practical constraints that arise in real-world scenarios and to explore ways to enhance the efficiency of the solution approaches. These constraints reflect relationships either between the bins and items or within the packing process itself. In the following, we review additional 2DRBPP attributes that typically arise in practical applications.

- Optional items

  The presence of optional items is generally intended to fill in the free spaces left by the required items (as in, e.g., [7]), which would otherwise result in material waste, by extra items. These extra items, e.g., could be needed in future productions. Not every application can allow optional items (e.g., when there is not enough storage space in the production site), or sometimes their presence can not be convenient (e.g., when the leftover material is refunded). Nevertheless, in most real-world situations optional items are allowed and they increase the efficiency of the solutions.

- Rotation

  The rotation constraint refers to whether items have a fixed orientation or can be rotated by 90 degrees (to preserve the axis-alignment of the items). In the literature, most studies focus on cases where the items must remain in a fixed orientation (see, for example, [14]). However, variants that allow rotation have been explored in a small number of papers (see [15, 85]), despite this constraint is very recurrent in applications.

- Guillotine cuts

  The guillotine constraint requires that all packed items can be reproduced through a series of edge-to-edge cuts parallel to the edges of the bin. These constraints are necessary, for example, when the cutting tool must perform a continuous cut, parallel to the bin's edges, without possible interruptions, therefore splitting the bin into two parts.

This constraint has been addressed by researchers, such as, e.g., in [17, 61]).

- Precedence relations

  Precedence constraints generally refer to the order of items, where precedence imposes the relative ordering of bins. The *Bin Packing Problem with Precedence Constraints* (BPP-P) is a variant that has gained increasing attention in recent years. This constraint introduces an additional requirement: items must be packed into bins in an order (i.e., in a *sequence*) that satisfies a given precedence. BPP-P has specific applications in assembly (Assembly Line Balancing Problem [9]) and scheduling (Multiprocessing Scheduling [38]) tasks. Several studies have focused on the BPP-P, including [25, 19], even if, to the best of our knowledge, they all refer to the one-dimensional case.

- Bin-Item conflicts

  Incompatibility constraints between bins and items are commonly encountered in real-world scenarios, when a specific item cannot be assigned to a particular bin (e.g., Server Consolidation Problem [40]). Moreover, the bin-items conflicts can represent the fact that an item cannot be placed in certain zones of the bin, or, on the contrary, must be placed in some others (e.g., the placement must consider different quality areas of the bin [43]).

- Item-Item conflicts

  Following the terminology introduced by Mezghani et al. [65], we can distinguish two types of item-item conflicts.

  - The first is *total conflict*, where conflicting items cannot be placed in the same bin (e.g., [68]). Most commonly, this kind of conflicts is encountered in the three-dimensional case, such as in storing problems, where items cannot share the same storage place, or in container loading, when commodities cannot be shipped together.
  - The second is *partial conflict*, where partially conflicting items can be packed in the same bin, but must be separated by a designated safe space (as in [42]). This problem has several practical applications, like in scenarios involving geographical location constraints, or the management of hazardous waste, where items may be partially incompatible and must be separated by a safety distance.

Notice that the 2DC&PP-PT problem described in Chapter 3 can be included in the Rectangular 2DVSBPP category. The items are indeed strongly heterogeneous rectangles, while the bins (also referred to as *sheets*)

are weakly heterogeneous rectangles with fixed dimensions. Moreover, we consider the presence of optional items, and we allow all, or some, orthogonal rotations for the items. Precedence constraints are considered in two versions, both as hard and soft constraints: for the hard version, this thesis extends the literature to the two-dimensional case, while the soft precedence constraints are, to the best of our knowledge, newly introduced by this work. In addition, our problem presents bin-item conflicts related to the forbidden and mandatory placement zones in the sheets, which can be different for each item. This attribute, to the best of our knowledge, have not been addressed for the rectangular packing problem in the available literature. We also consider partial item-item conflicts that reflects our notion of punching margin, which differ from the present literature due to the fact that each item in our case can have different punching margins on each side, making this attribute more complex to handle. Moreover, the alternative situations between two items of sharing a common cut or keeping a safety margin, to the best of our knowledge, has not been accounted in the literature so far, as well as the combination of all these additional technological constraints that characterize the 2DC&PP-PT. As a consequence, the 2DC&PP-PT can be considered, to the best of our knowledge, as a new variant of C&PP in the Operations Research literature.

4.2 Mathematical formulations

C&PP have been extensively studied through various mathematical models, which are used to represent and solve a wide range of practical applications, as well as to develop academic research in this field. Mathematical formulations for C&PP are often constructed using MILP models, as introduced in Section 2.2. MILP models provide the basis for exact solution approaches that allow for the optimization of cutting and packing operations while considering the geometric and operational constraints deriving from the problem.

In this field, MILP models are particularly useful because problems involve both binary and continuous decision variables, such as deciding whether to use a certain bin or not, and determining the exact position of items within a bin. These models offer a flexible framework for incorporating complex constraints, but they are computationally challenging to solve, especially for large-scale instances.

For classical surveys on mathematical formulations refer to [87] regarding one-dimensional bin packing and cutting stock problems, to [58, 60, 59] and [55] for reviews dedicated to two-dimensional problems with rectangular and irregular items, respectively.

4.2.1 Formulations for Two-Dimensional Rectangular Bin Packing Problem (2DRBPP)

The 2DRBPP is a specific variant of C&PP, where the goal is to pack a set of rectangular items into a minimum number of rectangular bins, as presented in Subsection 4.1.1. Several mathematical formulations of the problem have been proposed to tackle it. These formulations often include additional constraints that reflect practical considerations, such as variable bin sizes, optional items, precedence, and partial conflicts.

Pisinger and Sigurd in [72] presented a basic formulation of 2DRBPP with variable sized bins and costs. In many practical situations, bins of different sizes and costs are available, and the goal is not only to minimize the number of used bins, but also to minimize the total cost associated with using different bins. The authors formulated this problem as a MILP model, highlighting the importance of accounting for variable bin sizes, which adds another layer of complexity to the problem, making it more suitable for real-world packing operations.

In [85], Tang et al. provide a MILP formulation that accounts for orthogonal rotations. The problem is symmetric, thus only two orientations of the items are considered (i.e., 0 and 90 degrees rotations). Their modeling choice was to introduce a rotation binary variable for each item, and to embed it in the non-overlapping constraints so that the dimensions of the items are properly considered (i.e., swapped when the item is rotated by 90 degrees).

One way of formulating the presence of optional items in a MILP model can be found in [7]. The authors address this attribute for the one-dimensional problem, however the key point of the modeling involves assignment variables of the items to the bins, therefore making this formulation directly applicable to problems with different dimensions.

In [25], a mathematical formulation for the bin packing problem with precedence constraints has been proposed. In this variant, a partial order is imposed on the items, meaning that certain items must be placed in a specific order relative to others. The authors formulated the problem as a MILP model that incorporates this attribute in the one-dimensional problem, nevertheless, as discussed for the modeling of optional items in [7], the formulation of precedence constraints can be directly extended to higher dimensions. In this work the authors demonstrated that incorporating precedence constraints into the bin packing problem significantly increases its complexity in practice, but, once again, it also allows for more realistic modeling of real-world scenarios.

The presence of partial conflicts in 2DRBPP has been discussed, to the best of our knowledge, only by Hamdi-Dhaoui et al. in [42]. In this work the authors consider a version of the problem where a uniform distance must be kept between some items in the same sheet. They provide a formulation

of the specific case through a MILP model, which adapts the classical non-overlapping constraints to the presence of a conditional distance between the items.

The mentioned specific formulations for 2DRBPP demonstrate the diversity of mathematical models that have been developed to address different practical constraints. While each formulation is tailored to a particular set of constraints, they all share the common goal of optimizing the use of available space in the most efficient possible way. We also remark that, to the best of our knowledge, a complete model representing the 2DC&PP-PT has not yet been introduced in the literature. While certain attributes of the problem have been addressed with dedicated mathematical formulations, integrating these into a comprehensive linear model raises new challenges.

A review of exact methods to solve the models that have been presented for the 2DRBPP can be found in the work by Iori et al. [49], where the authors report and expand on previous surveys presented in [60, 58, 59].

4.3 Heuristic methods for 2DRBPP

Heuristic methods have become essential tools in solving large instances of the 2DRBPP. Due to the computational complexity of exact methods, heuristics provide a way to obtain good solutions in a reasonable amount of time, although without a guarantee of optimality. The flexibility and efficiency of heuristic approaches make them particularly well-suited for industrial applications where decision-makers must balance solution quality and computation time. In this section, we will explore the various heuristic techniques applied to 2DRBPP, including metaheuristics and matheuristics.

Greedy heuristics (see Subsection 2.3.1) are among the most widely used heuristic approaches for 2DRBPP due to their simplicity and speed. These are constructive algorithms that make decisions based on a local expansion criterion at each step, aiming to build a feasible solution incrementally. The most common greedy heuristics for 2DRBPP include the following.

- **First-Fit**

  In the First-Fit heuristic [26], items are packed into the first available bin that can accommodate them. The bins are tested in the order they were opened, meaning the heuristic checks from the first bin to the most recently opened one. If no existing bin can accommodate the item, a new bin is opened. This method is straightforward and efficient, often used as a baseline for more complex algorithms.

- **Best-Fit**

  The Best-Fit heuristic [27], attempts to place each item in the bin where it leaves the minimum amount of remaining space. This strategy

can lead to better space utilization compared to First-Fit, though it may require more computation since all the bins must be evaluated for placement at each expansion step.

- **Worst-Fit**

  Contrary to Best-Fit, the Worst-Fit heuristic [20] places items in the bin with the most remaining space. While this may seem counterintuitive, it can sometimes lead to improved distribution of items across bins, therefore reducing the tendency of prematurely opening new bins. This strategy avoids overfilling any single bin early on, leaving more room for larger items that may arrive later. The computational effort is similar to that of the Best-Fit heuristic.

These greedy algorithms have been widely studied in the literature, because of their effectiveness in handling simple instances of C&PP along with their variants. Despite their simplicity, they often serve as a foundation for more sophisticated approaches, including more complex heuristics and matheuristics. However, in many industrial applications, greedy algorithms are the only viable approach, due to the inner complexities of practical problems or the need to meet strict execution time limits.

More complex heuristic algorithms, such as pilot method, beam search, and genetic algorithms, offer more advanced strategies for exploring the solution space of 2DRBPP. These approaches provide a mechanism to escape local optima and explore a broader range of potential solutions.

- **Genetic Algorithm**

  Genetic Algorithm mimics the process of natural evolution, using operators such as selection, crossover, and mutation to evolve a population of candidate solutions. In the context of 2DRBPP, Genetic Algorithm has been applied to optimize the placement of rectangular items, taking into account constraints such as orientation and packing efficiency [85, 56]. Works considering this approach demonstrates its effectiveness even in complex application contexts.

- **Pilot Method**

  A recent work by Arruda et al. [4] proposes to enhance the greedy algorithm by introducing an alternative evaluation for the placement of the items, where the evaluation accounts for future placements resulting in a Pilot Method (see Subsection 2.3.2). The results that have been presented suggest that the quality of the solutions heavily depends on the geometry of the items.

- **Beam Search**

  The Beam Search (see Subsection 2.3.3) is a constructive algorithm that allows to explore more than one solution at the same time. It

has been applied to tackle not only the 2DRBPP [90], but also the one-dimensional case with variable sized bins and optional items [8], the circular variant [2], as well as the irregular one [10].

These heuristic approaches are highly adaptable and have been successfully employed in various 2DRBPP applications, from logistics to manufacturing, where achieving near-optimal solutions in a limited time frame is crucial.

On the other hand, matheuristic algorithms combine the strengths of both heuristic and exact methods. These approaches aim to leverage the computational efficiency of heuristics while benefiting from the optimization frameworks provided by exact algorithms. There can be several ways to hybridize heuristics with exact methods and vice versa, we present the two most common ones:

- *Heuristic-Exact combination.* One common approach is to use a greedy heuristic to generate an initial solution before running B&B. This two-stage process generates a primal bound that may allow for early node pruning and fasten the convergence to an optimal solution, which helps the application of B&B to larger instances. For example, Dell'Amico et al. [25] explored the combination of heuristics with exact techniques for solving bin packing problems with precedence constraints;

- *Heuristic-Exact integration.* Another option for hybridization is integrating heuristics with mathematical programming models to perform optimization sub-tasks. In this context, heuristics are employed to explore the solution space, while integer programming models handle the local optimization within selected regions of the space. This approach is particularly effective when dealing with complex constraints, such as variable bin sizes and costs [72].

Matheuristics offer a promising alternative for solving large-scale 2DRBPP instances, as they balance the trade-off between solution quality and computation time. Recent developments have also explored the use of machine learning techniques within matheuristic frameworks, further enhancing their adaptability and performance in dynamic environments [89].

Chapter 5

Mathematical modeling

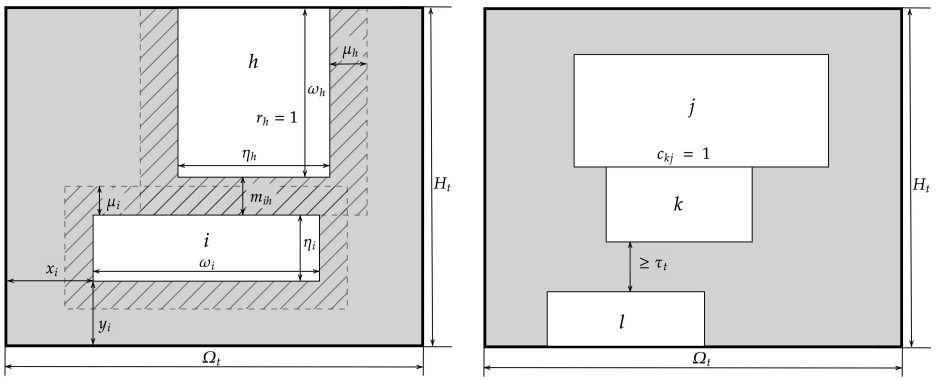
In this chapter, we propose a mathematical formulation of the 2DC&PP-PT described in Chapter 3. We model the problem in two consecutive steps via mathematical programming. We devise two MILP models, together representing all the attributes of the problem: in the first one, we minimize the material waste (primary objective of 2DC&PP-PT); in the second one, the number of soft precedence violations is minimized (secondary objective of 2DC&PP-PT) while satisfying a maximum waste constraint. These formulations refer to the model presented in [72] for the basic constraints, while integrating and extending the modeling of optional items in [7], orthogonal rotation in [85], punching margins in [42], and precedence relations in [25].

For ease of notation, the models discussed hereinafter are initially based on a simplified version of the problem described in Chapter 3. In this simplification, we assume that each item has the same punching margin on all sides (i.e., a uniform punching margin around the perimeter). To apply these models to the original problem, it would be necessary to expand the variables and parameters to account for different punching margins and rotations for each item on all four sides, as well as modify the corresponding constraints. While this would make the notation more complex, the underlying principles of the formulations would remain the same. The extension of the models to the original problem is discussed at the end of the chapter and a complete MILP model for 2DC&PP-PT is reported in Appendix A.

The models presented in this chapter have been published in [24].

5.1 Two-step modeling

An important attribute of the problem presented in Chapter 3 is the presence of soft precedence relations. These relations can only be violated when satisfying them would result in increased material waste. In other words, while we aim to minimize the number of soft precedence violations, we cannot prioritize these relations if doing so leads to higher waste. As such,



(a) (b)

Figure 5.1: Graphic representation of the notation for a sheet of type t containing items i, h with punching margins in (a), and items j, k, l without punching margins in (b).

soft precedence relations represent soft constraints, treated as a secondary objective in the problem. This creates a clear optimization hierarchy: the *primary objective* is to minimize material waste, while minimizing soft precedence violations as a *secondary objective*.

To address both objectives, following the principles of the lexicographic method (see Section 2.4), we chose to model the problem using two consecutive MILP formulations. Each model accounts for the problem's attributes, but focuses on a different objective: the first model minimizes material waste, while the second model minimizes soft precedence violations, subject to a maximum waste constraint.

5.2 Notation

Consider the set of compulsory items $I_c = \{1, \ldots, n_c\}$, the set of optional items $I_o = \{n_c + 1, \ldots, n_c + n_o = n\}$, and their union $I = I_c \cup I_o$. Each item $i \in I$ is characterized by its horizontal size (width) ω_i and its vertical size (height) η_i (as illustrated in Figure 5.1a for item i). Additionally, each item $i \in I$ may have an associated punching margin μ_i to maintain a specified distance from other items (e.g., the punching margin μ_i, μ_h for item i, h, respectively, in Figure 5.1a). Each item i also has placement constraints defined by two parameters: ϵ_i and ϕ_i, which limit the placement of i to be within a maximum and minimum distance from the sheet's border, respectively (as shown for item j in Figure 5.2). For items in the compulsory set I_c, the hard precedence level is denoted by an integer ρ_i, and the set of all pairs of items with a hard precedence relationship is given by $P = \{(i, j) \mid i, j \in I_c, \rho_i < \rho_j\}$. Similarly, to account for soft precedence rela-

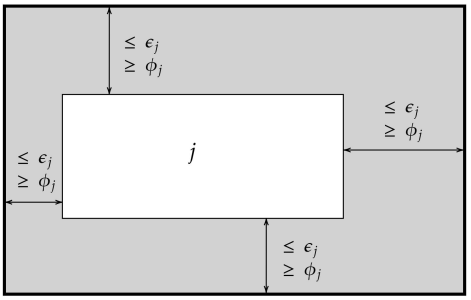


Figure 5.2: Example of placement limits for item j - maximum (ϵ_j) and minimum (ϕ_j) distance from the sheet's border.

tions, we define the soft precedence level for each compulsory item $i \in I_c$ as $\tilde{\rho}_i$, with the set of soft precedence pairs being $\tilde{P} = \{(i,j) \mid i,j \in I_c, \tilde{\rho}_i < \tilde{\rho}_j\}$. Notice that, in our context, lower the precedence level, higher the priority.

Let n_s represent the number of available sheets, $S = \{1,\ldots,n_s\}$ the set of positions in the production order, and $T = \{1,\ldots,n_t\}$ the set of sheet types. Each sheet type $t \in T$ is defined by its width Ω_t and height H_t (as depicted in Figure 5.1). We denote the maximum width, height, and maximum dimension across all sheet types as $\overline{\Omega} = \max_{t\in T} \Omega_t$, $\overline{H} = \max_{t\in T} H_t$, and $\overline{\Delta} = \max\{\overline{\Omega}, \overline{H}\}$, respectively. Each sheet type t has a maximum available quantity β_t and a safety distance τ_t, that is the minimum distance to keep between items without punching margins when they are not sharing a common cut (as for items k and l in Figure 5.1b). The notation related to sets and parameters is summarized in Table 5.1.

For each $i,j \in I$, $s \in S$, and $t \in T$, the following variables are introduced: x_i and y_i represent the distances of the bottom left corner of item i from the left and bottom sides of the sheet, respectively (see item i in Figure 5.1a); r_i is a binary variable equal to 1 if item i is placed vertically (as shown for item h in Figure 5.1a) and 0 otherwise; l_{ij} and b_{ij} are binary variables equal to 1 if item i is located to the left of, or below, item j, respectively, and 0 otherwise; c_{ij} is a binary variable equal to 1 if item i shares a common cut on its right or top side with item j (as illustrated for items k and j in Figure 5.1b) and 0 otherwise; m_{ij} represents the margin between the right and top sides of item i and the corresponding sides of j (e.g., the margin m_{ih} between items i and h in Figure 5.1a); f_{is} is a binary variable equal to 1 if item i is cut from the sheet in position s and 0 otherwise; g_{st} is a binary variable equal to 1 if the sheet in position s is of type t and 0 otherwise; a_{ijs} is a binary variable equal to 1 if items i and j are both placed on the sheet in position s and 0 otherwise. Finally, for every pair $(i,j) \in \tilde{P}$ and every $s \in S$, we define the decision variable q_{ijs}, which takes the value 1 if placing item j in position s violates the soft precedence with item i, and 0 otherwise. The decision variables are summarized in Table 5.2.

Symbol	Description
$I_c = \{1, \dots, n_c\}$	Set of compulsory items
$I_o = \{n_c + 1, \dots, n\}$	Set of optional items
$I = I_c \cup I_o$	Union of compulsory and optional items
$S = \{1, \dots, n_s\}$	Set of positions in the production order
$T = \{1, \dots, n_t\}$	Set of sheet types
$P = \{(i,j) \mid i,j \in I_c, \rho_i < \rho_j\}$	Set of hard precedence pairs
$\tilde{P} = \{(i,j) \mid i,j \in I_c, \tilde{\rho}_i < \tilde{\rho}_j\}$	Set of soft precedence pairs
ω_i	Width of $i \in I$
η_i	Height of $i \in I$
μ_i	Punching margin for $i \in I$
ϵ_i	Max distance of $i \in I$ from sheet's border
ϕ_i	Min distance of $i \in I$ from sheet's border
ρ_i	Hard precedence level for $i \in I_c$
$\tilde{\rho}_i$	Soft precedence level for $i \in I_c$
Ω_t	Width of sheet type $t \in T$
H_t	Height of sheet type $t \in T$
$\overline{\Omega} = \max_{t \in T} \Omega_t$	Max width across all sheet types
$\overline{H} = \max_{t \in T} H_t$	Max height across all sheet types
$\overline{\Delta} = \max\{\overline{\Omega}, \overline{H}\}$	Max size across all sheet types
β_t	Max available quantity of sheet type t
τ_t	Safety distance for sheet type t

Table 5.1: Summary of notation - sets and parameters

Variable	Description
(x_i, y_i)	Distance of the bottom left corner of $i \in I$ from the left/bottom side of the sheet
r_i	Binary variable, 1 if $i \in I$ is placed vertically
l_{ij}	Binary variable, 1 if $i \in I$ is placed to the left of $j \in I$
b_{ij}	Binary variable, 1 if $i \in I$ is located below $j \in I$
c_{ij}	Binary variable, 1 if $i \in I$ shares a common cut with $j \in I$
m_{ij}	Margin between the right and top sides of $i \in I$ and the corresponding sides of $j \in I$
f_{is}	Binary variable, 1 if $i \in I$ is cut from the sheet in position $s \in S$
g_{st}	Binary variable, 1 if the sheet in position $s \in S$ is of type $t \in T$
a_{ijs}	Binary variable, 1 if items $i, i \in I$ are both placed on the sheet in position s
q_{ijs}	Binary variable, 1 if placing $j \in I$ in position $s \in S$ violates soft precedence with $i \in I$

Table 5.2: Summary of decision variables

5.3 Optimizing the primary objective

The following MILP model includes all the hard constraints of the 2DC&PP-PT and minimizes the material waste (i.e., the primary objective). We refer to this formulation as *Model 1*.

$$\min \sum_{s\in S}\left[\sum_{t\in T}(\Omega_t H_t)g_{st} - \sum_{i\in I}(\omega_i\eta_i)f_{is}\right] \tag{5.1}$$

s.t.

$$l_{ij} + l_{ji} + b_{ij} + b_{ji} + (1-f_{is}) + (1-f_{js}) \geq 1 \qquad \forall i,j \in I, s \in S \tag{5.2}$$

$$x_i + (1-r_i)\omega_i + r_i\eta_i + m_{ij} \leq x_j + \overline{\Omega}(1-l_{ij}) \qquad \forall i,j \in I \tag{5.3}$$

$$x_i + (1-r_i)\omega_i + r_i\eta_i + m_{ij} \geq x_j - \overline{\Omega}(1-l_{ij}) \qquad \forall i,j \in I \tag{5.4}$$

$$y_i + (1-r_i)\eta_i + r_i\omega_i + m_{ij} \leq y_j + \overline{H}(1-b_{ij}) \qquad \forall i,j \in I \tag{5.5}$$

$$y_i + (1-r_i)\eta_i + r_i\omega_i + m_{ij} \geq y_j - \overline{H}(1-b_{ij}) \qquad \forall i,j \in I \tag{5.6}$$

$$m_{ij} \geq max\{\mu_i, \mu_j, \tau_t\}(1-c_{ij}) - \overline{\Delta}\,(2 - a_{ijs} - g_{st}) \qquad \begin{array}{l}\forall i,j \in I,\\ s \in S, t \in T\end{array} \tag{5.7}$$

$$m_{ij} \leq \overline{\Delta}(1-c_{ij}) \qquad \forall i,j \in I \tag{5.8}$$

$$a_{ijs} \geq f_{is} + f_{js} - 1 \qquad \forall i,j \in I, s \in S \tag{5.9}$$

$$\sum_{t\in T} g_{st} \leq 1 \qquad \forall s \in S \tag{5.10}$$

$$\sum_{s\in S} g_{st} \leq \beta_t \qquad \forall t \in T \tag{5.11}$$

$$\sum_{s\in S} f_{is} = 1 \qquad \forall i \in I_c \tag{5.12}$$

$$\sum_{s\in S} f_{is} \leq 1 \qquad \forall i \in I_o \tag{5.13}$$

$$\sum_{i\in I} f_{is} \leq n \sum_{t\in T} g_{st} \qquad \forall s \in S \tag{5.14}$$

$$f_{js} \leq \sum_{r=1}^{s} f_{ir} \qquad \begin{array}{l}\forall s \in S,\\ (i,j) \in P\end{array} \tag{5.15}$$

$$x_i + (1-r_i)\omega_i + r_i\eta_i \leq \sum_{t\in T}(\Omega_t - \phi_i)g_{st} + \overline{\Omega}(1-f_{is}) \qquad \forall i \in I, s \in S \tag{5.16}$$

$$y_i + (1-r_i)\eta_i + r_i\omega_i \leq \sum_{t\in T}(H_t - \phi_i)g_{st} + \overline{H}(1-f_{is}) \qquad \forall i \in I, s \in S \tag{5.17}$$

$$x_i \geq \phi_i - \overline{\Omega}(1-f_{is}) \qquad \forall i \in I, s \in S \tag{5.18}$$

$$y_i \geq \phi_i - \overline{H}(1-f_{is}) \qquad \forall i \in I, s \in S \tag{5.19}$$

$$x_i + (1-r_i)\omega_i + r_i\eta_i \leq \epsilon_i + \overline{\Omega}(1-f_{is}) \qquad \forall i \in I, s \in S \tag{5.20}$$

$$y_i + (1-r_i)\eta_i + r_i\omega_i \leq \epsilon_i + \overline{H}(1-f_{is}) \qquad \forall i \in I, s \in S \tag{5.21}$$

$$x_i \geq \sum_{t\in T}(\Omega_t - \epsilon_i)g_{st} - \overline{\Omega}(1-f_{is}) \qquad \forall i \in I, s \in S \tag{5.22}$$

$$y_i \geq \sum_{t \in T} (H_t - \epsilon_i) g_{st} - \overline{H}(1 - f_{is}) \qquad \forall i \in I, s \in S \quad (5.23)$$

$$x_i,\ y_i \geq 0 \qquad \forall i \in I \quad (5.24)$$

$$l_{ij},\ b_{ij},\ c_{ij} \in \{0,1\} \qquad \forall i,j \in I \quad (5.25)$$

$$r_i \in \{0,1\} \qquad \forall i \in I \quad (5.26)$$

$$m_{ij} \geq 0 \qquad \forall i,j \in I \quad (5.27)$$

$$f_{is} \in \{0,1\} \qquad \forall i \in I, s \in S \quad (5.28)$$

$$g_{st} \in \{0,1\} \qquad \forall s \in S, t \in T \quad (5.29)$$

$$a_{ijs} \in \{0,1\} \qquad \forall i,j \in I,\ s \in S \quad (5.30)$$

Model 1 is paired with a variable fixing pre-process that involves rotation and common cut requirements. As described in Section 3.3, each item has a set of allowed orthogonal rotations that, in the simplification adopted in this chapter, reflects the possibility of being placed horizontally or vertically. Therefore, we fix variables r of items that must be placed with a specific orientation:

$$r_i = \begin{cases} 0 & \forall i \in I \text{ that must be placed horizontally,} \\ 1 & \forall i \in I \text{ that must be placed vertically.} \end{cases} \quad (5.31)$$

Moreover, we recall that two items can share a side only if none of them has an assigned punching margin. Thus, we must prevent the possibility of sharing a common cut for any pair of items that involves punching margins:

$$c_{ij} = 0 \qquad \forall i,j \in I : \mu_i + \mu_j > 0. \quad (5.32)$$

In Model 1, we can see that the objective function (5.1) aims to minimize the total material waste across all used sheets, complying with the definition given in Section 3.3. In (5.2), we establish relationships between items: if two items are placed on the same sheet, they must be positioned one left or below the other. Constraints (5.3)-(5.6) prevent overlapping between items based on their mutual position, while accounting for the rotation of the items and their required margins. The margin between two items is defined by constraints (5.7)-(5.8): if they share a common cut, the margin is zero; otherwise, it must be at least the greater of the two items' punching margin and the safety one. To maintain the linearity of the model, variables a are introduced through constraints (5.9), which indicate whether two items are placed in the same sheet. In (5.10)-(5.11), we ensure that to each sheet is assigned at most one type without exceeding the available quantities. The assignment of the items to the sheets is regulated by constraints (5.12)-(5.13), which require that compulsory items must be placed on exactly one sheet, while optional items can be placed on at most one sheet. Notice that, according to the definition of the problem, the objective function prevents

the presence of sheets accommodating only optional items, since they would unnecessarily increase the material waste. Constraint (5.14) specifies that if an item is placed on a sheet, a type must be assigned to that sheet. Hard precedence relations between items are enforced through constraints (5.15), ensuring that if an item i has higher precedence than item j (i.e., if $(i, j) \in P$), then the sheet containing i is processed before the sheet containing j by having an earlier position in the solution sequence. Constraints (5.16)-(5.19) guarantee that each item is fully contained within a sheet, and they also prevent items from being placed in forbidden areas. Similarly, constraints (5.20)-(5.23) enforce that items are placed within their mandatory zones, if any are assigned. Finally, constraints (5.24)-(5.30) specify the domains of the variables.

With respect to the definition of the 2DC&PP-PT given in Section 3.3, both the objective function and all the hard constraints have been modeled trough linear expressions in Model 1. In the following section, we discuss the missing attribute concerning soft precedence relations, thus retrieving a complete formulation of the problem.

5.4 Optimizing the secondary objective

As previously discussed, we decide to model the soft constraint concerning the soft precedence attribute as the secondary objective of the problem. To this end we propose the following MILP model, referred to as *Model 2*, which minimizes the number of soft precedence violations under all the hard constraints and imposing an additional maximum waste usage constraint, to account for the primary objective of the problem.

$$\min \sum_{(i,j)\in\tilde{P}} \sum_{s\in S} q_{ijs} \tag{5.33}$$

s.t.

$$(5.2) - (5.30)$$

$$q_{ijs} \geq f_{js} - \sum_{r=1}^{s} f_{ir} \qquad \forall s \in S, (i,j) \in \tilde{P} \tag{5.34}$$

$$\sum_{s\in S} \left[\sum_{t\in T} (\Omega_t H_t) g_{st} - \sum_{i\in I} (\omega_i \eta_i) f_{is} \right] \leq v^* \tag{5.35}$$

$$q_{ijs} \in \{0, 1\} \qquad \forall s \in S, (i,j) \in \tilde{P}. \tag{5.36}$$

In Model 2, the objective function (5.33) minimizes the total number of violated soft precedence relations, which are counted through constraints (5.34), following the modeling approach used for the hard precedence constraints in (5.15). The hard constraints of the problem are (5.2)-(5.30),

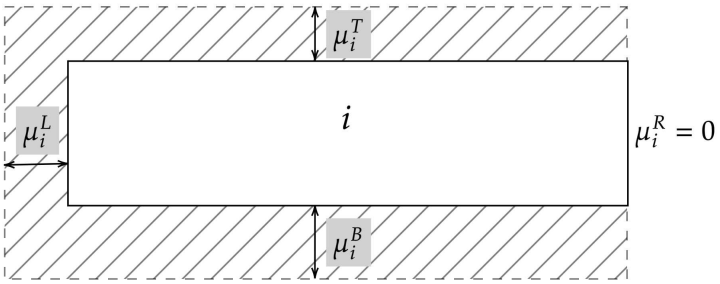


Figure 5.3: Example of four different punching margins for an item i.

taken from Model 1, and the additional constraint on the maximum total amount of material waste is guaranteed by (5.35).

5.5 Extension to the full-featured 2DC&PP-PT

As motivated at the beginning of the chapter, both models refer to a simplification of the problem where every item has a uniform punching margin assigned on each side, which can be zero if no punching margin is needed. In this section, we discuss how to extend the models to comply with the original 2DC&PP-PT defined in Chapter 3, where an item can have a different punching margin on each side, as well as limited rotation possibilities.

The notation must account for the presence of different punching margins, therefore the parameter μ_i is replaced by the family of parameters μ_i^L, μ_i^R, μ_i^B, and μ_i^T representing the punching margin assigned to the left, right, bottom, and top side of item i, respectively (see Figure 5.3 for a graphic representation). Moreover, to take possible forbidden rotations into account, for each orthogonal rotation $\theta \in \Theta = \{0, 90, 180, 270\}$, a binary parameter σ_i^θ is associated with every item $i \in I$, taking value 1 if the item can be placed at θ degrees, 0 otherwise.

In this extension, for each item i, the rotation decision variable r_i not only devises the size of the placed item (i.e., swapped width and height if it is placed vertically), but it also determines the punching margins of the placed item, since each punching margin is strictly related to its side. To this end, for every item $i \in I$, we consider four binary variables $r_i^\theta \in \{0, 1\}$, one for each orthogonal rotation $\theta \in \Theta$, that will be 1 if item i is placed at θ degrees, and 0 otherwise. Since each item must be placed with exactly one allowed orientation, the following constraints must be added to the models:

$$r_i^\theta \leq \sigma_i^\theta \qquad \forall i \in I, \theta \in \Theta \tag{5.37}$$

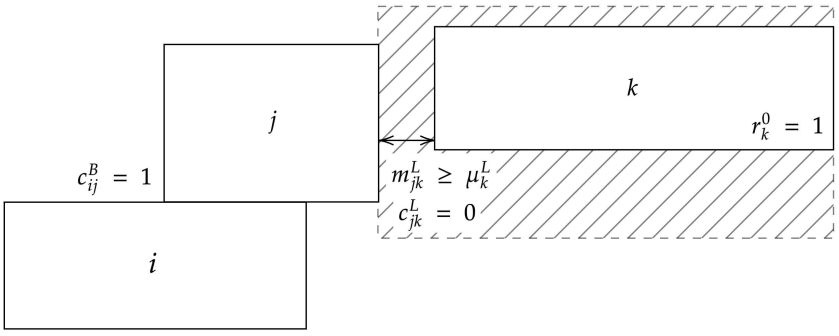


Figure 5.4: Example of extended rotation, common cut, and margin variables in a group of three items i, j, k.

$$\sum_{\theta \in \Theta} r_i^\theta = 1 \qquad \forall i \in I. \tag{5.38}$$

Where (5.37) extends and replaces the pre-process in (5.31). Moreover, both in Model 1 and Model 2, the variable r_i must be replaced with $(r_i^{90} + r_i^{270})$ in the constraints involving it to define the horizontal or vertical orientation of item i (namely, constraints (5.3)-(5.6), (5.16)-(5.17), and (5.20)-(5.21)).

The extension must concern also the decision variables and the constraints related to the common cuts and the margins. For each pair of items $i, j \in I$, we explode the corresponding decision variable c_{ij} in the right and top sides of i by considering the binary variable c_{ij}^L (resp. c_{ij}^B), that are 1 if item i shares its right (resp. top) side in common cut with the left (resp. bottom) side of item j, and 0 otherwise. Similarly, we replace, for each $i, j \in I$, the margin variable m_{ij} with the non-negative variables m_{ij}^L (resp. m_{ij}^B) that correspond to the distance margin between the right (resp. top) side of item i with the left (resp. bottom) side of item j. A graphic intuition of the extended variables is given in Figure 5.4. Consequently, constraints must be extended accordingly.

In (5.3)-(5.6), the first couple of constraints must consider m_{ij}^L instead of m_{ij}, while the second couple of constraints must replace m_{ij} with m_{ij}^B.

Moreover, the possibility of sharing a common cut between two items $i, j \in I$ has to account for their mutual positions, left to or below, separately and to the rotation of the items involved. Thus, the pre-process according to (5.32) must consider c_{ij}^L (instead of c_{ij}) when item i is placed left to item j, and its formulation must also be conditional to the combination of the items' rotations to properly consider the adjacent punching margins. There are sixteen possible combinations of the items' rotation, therefore we will have sixteen groups of constraints to add to the model instead of the

pre-process, one for each combination, of the following form:

$$\begin{aligned} c_{ij}^L &\leq 3 - l_{ij} - r_i^0 - r_j^0 && \forall i,j \in I : \mu_i^R + \mu_j^L > 0 \\ c_{ij}^L &\leq 3 - l_{ij} - r_i^0 - r_j^{90} && \forall i,j \in I : \mu_i^R + \mu_j^T > 0 \\ & \vdots \\ c_{ij}^L &\leq 3 - l_{ij} - r_i^{270} - r_j^{270} && \forall i,j \in I : \mu_i^T + \mu_j^B > 0. \end{aligned} \tag{5.39}$$

Analogous reasoning must be applied when item i is placed below item j, $\forall i,j \in I$. In this case, constraints involve variable c_{ij}^B and, again, are replicated for every items' rotation combination. For example, for the case with both items rotated at 0 degrees, we have:

$$c_{ij}^B \leq 3 - b_{ij} - r_i^0 - r_j^0 \qquad \forall i,j \in I : \mu_i^T + \mu_j^B > 0. \tag{5.40}$$

Constraints that consider margins' upper and lower bounds must also be extended. On the one hand, constraints related to margins' upper bounds (5.8), should address the extended decision variables, in the two possible mutual positions, as follows:

$$m_{ij}^L \leq \overline{\Omega}(1 - c_{ij}^L) \qquad \forall i,j \in I \tag{5.41}$$

$$m_{ij}^B \leq \overline{H}(1 - c_{ij}^B) \qquad \forall i,j \in I. \tag{5.42}$$

On the other hand, margins' lower bound constraints (5.7), since they directly involve the punching margin's parameters, should consider the possible combinations of the items' rotation both when items are placed one left to the other (as in (5.39)), or one below the other (as in (5.40)). We report, as an example, one out of the sixteen couples of constraints that extend (5.7), in the case where both items are placed at 0 degrees rotation, $\forall i,j \in I, s \in S, t \in T$:

$$m_{ij}^L \geq max\{\mu_i^R, \mu_j^L, \tau_t\}(1 - c_{ij}^L) - \overline{\Omega}\,(2 - a_{ijs} - g_{st}) - \overline{\Omega}(r_i^0 + r_j^0) \tag{5.43}$$

$$m_{ij}^B \geq max\{\mu_i^T, \mu_j^B, \tau_t\}(1 - c_{ij}^B) - \overline{H}\,(2 - a_{ijs} - g_{st}) - \overline{H}(r_i^0 + r_j^0). \tag{5.44}$$

In conclusion, we have seen that the models can be extended to represent the original problem at the cost of a more complex notation, and of a higher number of decision variables and constraints. The complete extension of Model 1 to the original 2DC&PP-PT can be found in Appendix A and, in a similar way, the one for Model 2 can be obtained.

Chapter 6

Solution methods

In the present chapter, we describe five solution approaches that we devised to tackle the 2DC&PP-PT problem presented in Section 3.3. We designed methods of different nature to comply with several needs that can vary depending on the application and on the purpose of the solution. We propose one exact method and one matheuristic, both based on the formulation of the problem devised through MILP models (Chapter 5), along with three heuristics that require increasing computational resources and provide, accordingly, solutions of increasing quality. All the proposed heuristics are characterized by the need to solve a common sub-problem as sub-routine, that is, given a partial layout, we have to place a new item by satisfying non-overlapping and technological constraints, and by optimizing a given placement evaluation criterion. To solve this sub-problem, we devise a placement heuristic, specialized for the 2DC&PP-PT, that is used by all the three proposed heuristics.

6.1 Mathematical model-based approaches

In this section, we present two approaches to solve the 2DC&PP-PT, both based on the mathematical models presented in Chapter 5. The first method is an exact approach, therefore providing us with an optimal solution of the problem. The second method is a matheuristic which exploits a formulation based on Model 1 only, and handles the secondary objective through an iterative heuristic on top.

Notice that, since the mathematical models have been devised under the assumption of having a uniform punching margin associated with each item, also the model-based approaches are presented with the same simplification of the problem. Even so, by extending the formulations to handle different punching margins over the sides of each item, as proposed at the end of Chapter 5, the exact approach and the matheuristic are automatically extended accordingly.

6.1.1 Exact lexicographic procedure

The problem under study includes a soft constraint related to the soft precedence attribute. As discussed in Section 5.1, this attribute is treated as a secondary objective, subordinated to the minimization of material waste, which is the primary objective. Based on this observation, we can approach the problem as having two objectives in a lexicographic order. Consequently, the optimal solution can be obtained using the lexicographic method described in Section 2.4.

We solve the problem to optimality in two steps. First, solve Model 1 and compute the optimal minimum quantity of the material waste (primary objective). Second, Model 2 is solved, where the number of soft precedence violations is minimized (secondary objective), and the solution is constrained to have material waste less than or equal to the optimal value v^* computed in the previous step (parameter v^*). The solution provided by Model 2 corresponds to an optimal solution of the problem. This solution method is called *Lexicographic Procedure* and it is summarized in Algorithm 5.

Algorithm 5 Lexicographic Procedure

```
1: procedure LexicographicProcedure
2:     solve Model 1, compute the optimal value v* of the material waste;
3:     solve Model 2 by constraining the material waste to be at most v*;
4:     STOP: the solution of Step 3 is the optimal solution to the problem.
5: end procedure
```

6.1.2 Iterative matheuristic

In this section, we propose an alternative way to handle the soft precedence constraint through an iterative heuristic that exploits a formulation based on Model 1 to minimize the material waste. We observed that, in the exact lexicographic procedure, solving the two models is time-consuming. In fact, in addition to a first MILP model, a second model has to be solved, which also includes a non-negligible number of further variables and constraints to handle soft constraints and the related objective function. We observe that, to guarantee optimality, the second model cannot inherit any of the decisions from the optimal solution of the first model. To avoid solving the second "complete" model, we developed a heuristic where only Model 1 is used, and the soft constraints are incrementally added to the formulation after having been transformed into hard constraints. This is done by progressively estimating upper bounds on the positions of prioritized items, by solving smaller and more constrained models. The core idea of the algorithm is to partition the compulsory items based on their soft precedence level (higher precedence first), and to iteratively determine their maximum

position within the sequence of sheets. This approach, called *Iterative Procedure*, results in a matheuristic, and it is summarized in Algorithm 6.

Let us consider the following partition of the compulsory items with respect to their soft precedence level. Let $\tilde{\rho}_{max}$ be the maximum level of soft precedence. Based on the notation introduced in Section 5.2, for each precedence level $k = 1, \dots, \tilde{\rho}_{max}$, let $I_k = \{i \in I_c \mid \tilde{\rho}_i = k\} \subseteq I_c$ be the subset of compulsory items with soft precedence k. Recall that, specific to our notation, higher the precedence level, lower the priority value, i.e., items with higher precedence are placed on sheets that are cut before those containing only items with lower precedence. In addition, we introduce an integer parameter δ_k, for $k = 1, \dots, \tilde{\rho}_{max}$, that represents the maximum sheet position in which items in I_k can be placed. This parameter is determined throughout iterations as follows. In the first iteration $k = 1$, we solve Model 1 in which we ignore optional items (that, we recall, have no associated priority) and we consider compulsory items in I_1. Throughout iterations, we ignore optional items because they are not involved in precedence relations, therefore their presence does not affect the number of soft precedence violations, which is regulated by the iterative scheme. The number of used sheets in the solution will be the value of δ_1, thus identifying an upper bound on the position of items in I_1 in the solution sequence. Similarly, starting from the second iteration (i.e., for $k \geq 2$), we solve Model 1 where we consider only compulsory items up to soft precedence k (i.e., belonging to $\bigcup_{\bar{k}=1,\dots,k} I_{\bar{k}}$) and we constrain the maximum position in the sheets' sequence of the items up to soft precedence $k-1$ by adding the following constraints:

$$\sum_{s \in \{1,\dots,\delta_{\bar{k}}\}} f_{is} = 1 \quad \forall i \in I_{\bar{k}}, \ \forall \bar{k} = 1, \dots, k-1. \tag{6.1}$$

Notice that at iteration k, the parameter $\delta_{\bar{k}}$ for $\bar{k} = 1, \dots, k-1$ has been computed in the previous steps, and let δ_k be set to the number of sheets in the current solution. Once all δ-parameters are computed, we perform one last iteration to place all the items. We solve Model 1 (considering also optional items), where compulsory items are constrained to be placed according to the upper bound given by the δ-parameters through the following additional constraints:

$$\sum_{s \in \{1,\dots,\delta_k\}} f_{is} = 1 \quad \forall i \in I_k, \ \forall k = 1, \dots, \tilde{\rho}_{max}. \tag{6.2}$$

The rationale behind this algorithm is that each iteration, through constraints (6.1), ensures that items with higher soft precedence are placed in the initial sheets, allowing items with lower soft precedence to be placed in the subsequent sheets or used to reduce waste in the earlier sheets, though this may lead to some precedence violations for the sake of material waste minimization.

Notice that, in absence of soft precedence relations, the algorithm performs only two iterations: first it solves the problem with compulsory items only, and then it solves the complete instance by constraining the maximum number of sheets in the solution.

Algorithm 6 Iterative Procedure

procedure IterativeProcedure
  let $I_k = \{i \in I_c \mid \tilde{\rho}_i = k\}$, and introduce δ_k for $k = 1, \ldots, \tilde{\rho}_{max}$;
  for $k = 1, \ldots, \tilde{\rho}_{max}$ **do**
    solve Model 1 by:
    - ignoring optional items;
    - considering only compulsory items up to soft precedence k;
    - constraining maximum position $\delta_{\bar{k}}$ for items of soft precedence $\bar{k} = 1, \ldots, k-1$ (for $k \geq 2$);
    fix δ_k = number of sheets in the solution;
  end for
  solve Model 1, with optional items, by constraining maximum position δ_k for items of soft precedence $k = 1, \ldots, \tilde{\rho}_{max}$;
  STOP: the solution of Step 10 is the optimal solution to the problem.
end procedure

6.2 Placement heuristic

This section presents an algorithm that is used by all the heuristic methods that will be described in Section 6.3 and 6.4. This algorithm, called *Placement Heuristic*, given an item and a partial layout, determines the best position and rotation to place the item in the current sheet, when possible, according to an evaluation criterion. The position is chosen in a finite set of positions related to the partial layout, and defined as we will discuss in Subsection 6.2.2. First, a *precedence-feasibility check* is performed to verify if all the items with higher hard precedence than the current one have already been placed, and, if not, the algorithm cannot place the current item, since it would break a hard precedence constraint.

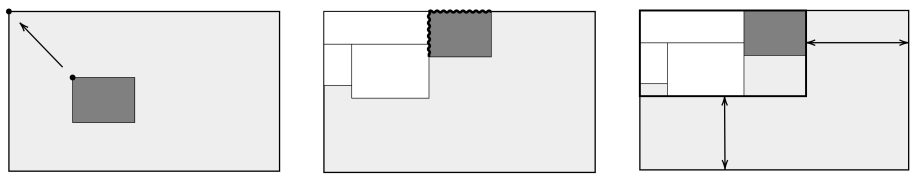
6.2.1 Placement evaluation criterion

If the feasibility check is passed, the best placement position of the item, for each of its allowed rotations, is computed, and the best fit among them is selected, based on the given evaluation function. The *evaluation* is a crucial component of the heuristic, and as such it can have a great impact on the final layout. Since the Placement Heuristic is used in constructive algorithms, we have considered evaluation criteria that prefer compact layouts because, in most cases, the more compact a partial layout is, higher the chances to

place the following items are. To this end, classical evaluation functions can be considered [51, 60], such as the one that pushes the items toward the top-left corner of the sheet by minimizing the x-coordinate and maximizing the y-coordinate of the placement position (Figure 6.1a), as well as the one which maximizes the contact perimeter of the item with the partial layout, that is the amount of perimeter of the item that touches either the sheet border or placed items (Figure 6.1b). Alternatively, the horizontal or vertical gap between the partial layout's axis aligned bounding box and the sheet border can be minimized or maximized to shape the partial layout accordingly (Figure 6.1c).

6.2.2 Margin-aware Extreme Points

Given the item to be placed with a chosen rotation, the Placement Heuristic tests the placement in some specific positions and selects the best one according to a chosen compactness criterion (e.g., the ones previously presented). To devise the positions to test, we start from the concept of *Extreme Points* (EPs) as defined by Crainic et al. in [22]. EPs are specific locations where an item can be placed to utilize the free space delimited by the items that are already placed in the sheet. As we can see in Figure 6.2, EPs are defined with respect to the partial layout, in fact the set of EPs is updated whenever a new item is placed in the sheet by projecting its bottom-left and top-right corner on the placed items to retrieve new positions for the next item to place. Notice that, since the set of EPs is updated whenever a new item is placed, given two partial layouts with the same items that have been placed in a different order, the corresponding set of EPs could be different, in other words, it depends not only on the placed items, but also on the order in which they have been placed. When placing an item with a chosen rotation, the heuristic evaluates all feasible EPs and selects the best one based on the compactness criterion. This criterion, along with the nature of the EPs, ensures that items are packed closely together, maximizing the use of the available space. Notice that, when positioning an item on an extreme point, a *geometry-feasibility check* is needed to verify if the new item overlaps the



(a) Top-left placement. (b) Contact perimeter. (c) Horizontal/vertical gap.

Figure 6.1: Example of evaluation criteria for an item (in dark gray) in a partial layout (in white).

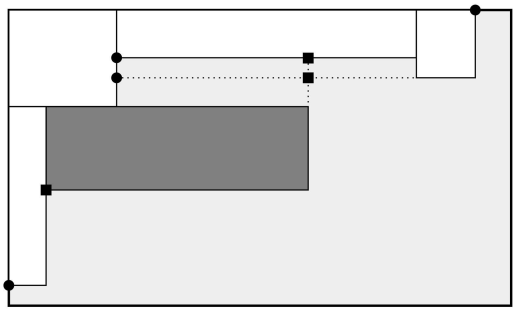


Figure 6.2: Example of extreme points. The dark gray item is added to the partial layout (in white), and the set of EPs (black circles) is updated with the projections of its bottom-left and top-right corner (black squares).

already placed ones, or if it exceeds the sheet.

Handling punching margins

In our framework, we have to consider that items can be assigned with a punching margin to keep from each others. To account for this attribute, we need to extend the concept of EPs, making it dependent not only on the partial layout, but also on the item to place. To this end, we propose the following procedure, which exploits the classical generation of EPs within a scheme that handles the presence of punching margins on top. Given a partial layout and an item $\hat{i}$ to place, we build the set of EPs, specific to $\hat{i}$, in the following way. We start from an empty set of EPs, and we consider the sheet to be empty. Following the order of placement of the partial layout, we place each item in its corresponding position in the partial layout, and we inflate it by the maximum between its punching margin and the one of the item to place $\hat{i}$. At each insertion, we update the EPs with the standard procedure, so that the final set will contain positions that account for overlapping punching margin (if any) between the item to place $\hat{i}$ and each of the items already in the partial layout.

An example of this procedure in given in Figure 6.3: we have a partial layout where some items have an assigned punching margin, and we want to place an item $\hat{i}$ that has a punching margin itself. We replace the placed items with their inflated version as described above (i.e, inflated with respect to the maximum punching margin between the ones of the placed item and $\hat{i}$), and we end up having the set of EPs related to $\hat{i}$. In Figure 6.4, we see that placing $\hat{i}$ in the inflated partial layout results in respecting the punching margins in the original one, allowing punching margins overlap to reduce the material waste. In the example, for the sake of clarity, a uniform punching margin is assigned to each item along its border. However, the procedure is able to handle different punching margins by inflating each side of the items

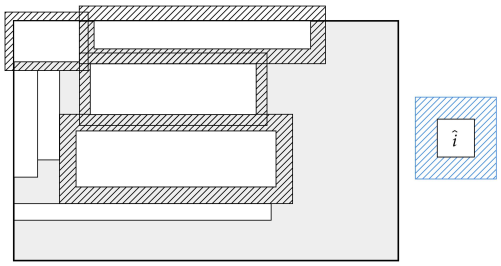
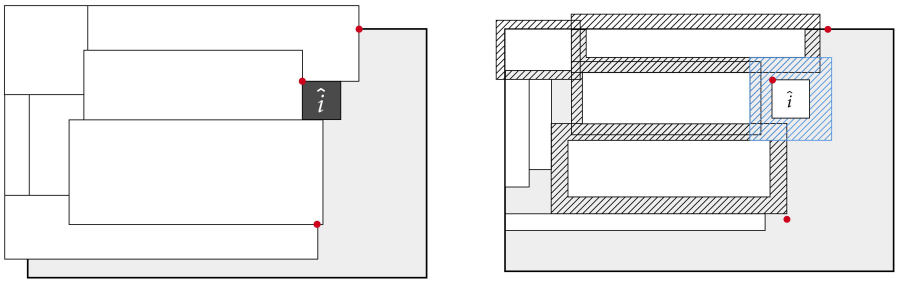


Figure 6.3: Example - placing item $\hat{i}$ in a partial layout where some items have an assigned punching margin (dashed area).



(a) (b)

Figure 6.4: Example - extreme points for item $\hat{i}$, considering overlapping punching margins. Placing $\hat{i}$ without punching margins in the inflated layout (a) is equivalent to place it in the original one with punching margins (b).

in the partial layout with respect to the maximum between their punching margin and the one of the opposite side in the item to place (e.g., inflate the left side on an item in the partial layout by the maximum between its left punching margin and the one assigned to the right side of the item to place $\hat{i}$).

Checking safety margins

When no punching margins are assigned between two adjacent items, they can either share a side or must maintain a safety distance. To handle this constraint, when placing an item without punching margins, the related extreme points on the border of items in the partial layout (i.e., those without assigned punching margins) should be duplicated by shifting the point by the safety margin. This way, the original EP accounts for the common cut scenario, while the duplicated one ensures the required safety margin. In our Placement Heuristic, we prioritize having items that share a common cut to maintain compactness and minimize material waste between items whenever possible. Therefore, we compute EPs as previously described, without

considering the safety margin (i.e., without duplicating common cut EPs), thus forcing shared cuts. However, during the geometry-feasibility check, when testing a position, we examine the relative placement of the item being tested against all other placed items. If the distance between them is greater than zero but less than the safety margin, the position is considered infeasible. This approach is supported by computational experiments, which show that very few EPs are discarded due to safety margin infeasibility, justifying the computational savings of not duplicating and testing EPs for safety margins.

Algorithm 7 Placement Heuristic

```
1: procedure PlacementHeuristic(item, sheet, place_eval)
2:     if hard precedence constraint is violated then
3:         return no feasible placement due to precedence constraints.
4:     end if
5:     let (best_rot, best_pos) be the current best placement option;
6:     initialize (best_rot, best_pos) to empty;
7:     for each rotation rot in item's allowed rotations do
8:         retrieve extreme points (EPs) for the rotated item on the sheet;
9:         for each position pos in EPs do
10:            if overlap with previously placed items is detected then
11:                skip to the next position;
12:            end if
13:            if safety margins are violated at pos then
14:                skip to the next position;
15:            end if
16:            if place_eval(item, rot, pos) improves current best then
17:                update (best_rot, best_pos) to (rot, pos);
18:            end if
19:        end for
20:    end for
21:    return (best_rot, best_pos), best placement found.
22: end procedure
```

The Placement Heuristic is summarized in Algorithm 7. The procedure is designed to identify a placement for a given item (*item*) within a layout (*sheet*), while adhering to predefined constraints. The algorithm systematically evaluates potential placements based on an input evaluation criterion (*place_eval*), ensuring that the selected placement maximizes suitability under the specified conditions.

Initially, the algorithm checks for hard precedence constraints (Steps 2–4), ensuring that any violation of these mandatory relations results in an immediate termination of the procedure. This early exit prevents unnecessary computations when an infeasible configuration is detected. Next, the

algorithm initializes the variables *best_rot* and *best_pos* (Steps 5–6) to represent the current best rotation and position for the item. At this stage, these variables are set to empty values, indicating that no feasible placement has yet been identified.

The main computation involves iterating over all allowed rotations (Step 7). For each rotation (*rot*), the algorithm retrieves the extreme points (EPs) on the layout, which represent potential candidate positions for placing the item (Step 8). Each candidate position (*pos*) is then evaluated (Step 9) with respect to several constraints. First, the overlap constraint (Steps 10–12) ensures that placing the item at the current position does not interfere with any previously placed items. If an overlap is detected, the position is immediately discarded. Next, the safety margin constraint (Steps 13–15) verifies that the required spacing around the item is satisfied. If these margins are violated, the position is skipped. Finally, the placement evaluation criterion (Steps 16–18) assesses the quality of the placement by applying *place_eval* to the current rotation and position. If this placement improves upon the current best option, the algorithm updates *best_rot* and *best_pos* accordingly. The nested loop structure ensures that all valid combinations of rotations and positions are systematically considered. By continuously updating the best placement option, the algorithm identifies the most suitable configuration based on the given evaluation criterion.

The procedure terminates by returning the best found placement option (*best_rot*, *best_pos*) (Step 21). If no feasible placement is identified due to constraint violations, the algorithm terminates and returns an empty result.

6.3 Constructive heuristics

The problem addressed in this thesis is inherently complex. As highlighted in the formulation presented in Chapter 5, the feasible region is defined by a set of constraints that grows exponentially with the number of items. This complexity makes mathematical model-based approaches impractical for finding solutions in large instances. To address this challenge, we propose three heuristics tailored to different needs: a fast procedure for generating a reasonable feasible solution (*Greedy Algorithm*), a method that produces high-quality solutions at the cost of longer execution times (*Beam Search*), and an intermediate approach that strikes a balance between running times and solution quality (*Pilot Method*).

To focus on the most challenging and significant aspects of the problem for the reference company, we simplify the problem in all three heuristics by assuming that all sheets have the same size (i.e., only one sheet type is available), and that items are not assigned with any forbidden or mandatory placement zones. In addition, the simplification is justified by the fact that, although these two attributes are not directly addressed, the methods can

be extended to generate feasible solutions for the original problem. Since a constructive heuristic, in our context, generates solutions on a sheet-by-sheet basis, with items placed one at a time, we outline here possible extensions to the simplified attributes accordingly, before proceeding to the description of the heuristics. When working with multiple sheet types, we can initially choose the largest available sheet to fit as many items as possible, then check if the same configuration can be accommodated in a smaller sheet to reduce material waste. Additionally, to handle forbidden zones, we can treat these zones as dummy items in the current partial solution, since no overlaps are allowed. For mandatory zones, the placement area for a given item can be restricted accordingly. This is one possible extension, but more sophisticated adaptations can be developed.

In the following subsections we present the two constructive heuristics that we devised for the 2DC&PP-PT (i.e., the greedy algorithm, and the pilot method), while the beam search approach will be discussed in Section 6.4.

6.3.1 Greedy algorithm

We introduce a constructive heuristic designed to generate a sequence of cutting layouts such that the constraints, both hard and soft, related to the problem under study are considered. We call it *Greedy Algorithm*, because it operates on a sheet-by-sheet basis by placing items sequentially in a greedy way. This method constructs one sheet at a time, subject to the availability of sheets, using the specialized Placement Heuristic, presented in Section 6.2, to place the items according to a given order.

The placement sorting of the items is a key aspect of the algorithm, since it handles the presence of optional items and the soft precedence relations. On the one hand, the sequence begins with compulsory items followed by the optional ones. In this way, for each sheet, the placement of optional items is tested after all compulsory items that fit in the layout have already been positioned, thus filling in the remaining spaces to reduce the waste. On the other hand, compulsory items are prioritized based on their soft precedence levels. By doing so, we ensure that an item of lower soft precedence is placed before items of higher soft precedence, only if these last ones do not fit the sheet. In other words, a soft precedence relation is violated only when it reduces the material waste. Moreover, within the same soft precedence level, items are further sorted according to a specific sorting heuristic to guide the placement. Classical sorting heuristics are, for example, the ones based on the following characteristics of the items [51, 70]: area, shorter or longer side, perimeter, difference or ration between sides (for all these features the sort can be considered both ascending or descending).

The Greedy Algorithm is summarized in Algorithm 8. This procedure aims to construct a feasible solution to the placement problem by iteratively

Algorithm 8 Greedy Algorithm

```
1: procedure GreedyAlgorithm(items, sort_eval, place_eval)
2:     let solution be an empty set of cutting layouts;
3:     while |items| > 0 do
4:         if no available sheets remain then
5:             return solution, partial solution found (out of sheets).
6:         end if
7:         select a new empty sheet;
8:         sort items by (compulsory first, soft precedence, sort_eval);
9:         for each item in items do
10:            (rot, pos) ← PlacementHeuristic(item, sheet, place_eval);
11:            if feasible placement found then
12:                place item on the sheet at (rot, pos);
13:            else
14:                continue to the next item;
15:            end if
16:        end for
17:        add the sheet (with placed items) to solution;
18:        update items deleting the placed ones;
19:    end while
20:    return solution, final solution completed.
21: end procedure
```

filling cutting layouts with the given items (*items*) while adhering to constraints and optimizing the solution based on heuristic evaluations. The algorithm relies on two key parameters: the criterion adopted by the sorting evaluation heuristic (*sort_eval*), that determines the order in which items are processed, and the placement evaluation criterion (*place_eval*), which guides the placement decisions within the layouts.

The algorithm initializes an empty set of cutting layouts (*solution*) to store the final result (Step 2). The main loop (Step 3) iterates until all items have been placed. At the beginning of each iteration, the algorithm checks whether any sheets remain available for use (Steps 4–6). If no sheets are left, the procedure terminates early, returning the *solution* found so far as a partial result. Otherwise, the algorithm selects a new empty *sheet* for use in the current iteration (Step 7). Before attempting to place items on the current *sheet*, the items are sorted using a composite sorting strategy (Step 8). This strategy prioritizes compulsory items and items with higher soft precedence, ensuring that these constraints are addressed first. After accounting for these priorities, the sorting evaluation heuristic *sort_eval* is applied in a lexicographic manner to determine the final order of items. This step ensures that the order in which items are considered aligns with both the problem constraints and the heuristic guidance.

The algorithm then processes each *item* in the defined order (Step 9). For each item, it invokes the `PlacementHeuristic` (Step 10), passing the current *item*, *sheet*, and the placement evaluation criterion *place_eval*. This subroutine attempts to identify a feasible placement for the item, returning a rotation (*rot*) and a position (*pos*) if the placement is found. If a feasible placement is identified (Step 11), the item is placed on the *sheet* with the specified configuration (Step 12). Otherwise, the algorithm skips the current item and moves to the next one (Step 14), ensuring that no infeasible placements are taken into account.

Once all items have been considered for the current *sheet*, the algorithm finalizes the *sheet* by adding it, along with the placed items, to the *solution* (Step 17). The *items* list is then updated to remove the items that have been successfully placed (Step 18), ensuring that only unplaced items are carried forward to subsequent iterations. The process continues until all items have been placed or no sheets remain. Upon completion, the algorithm returns the final *solution*, which consists of a sequence of cutting layouts (Step 20).

6.3.2 Pilot method

We propose an alternative version of the greedy algorithm presented in Subsection 6.3.1, which results in a *Pilot Method* (Subsection 2.3.2). One of the parameters of Algorithm 8 is the evaluation criterion by which the Placement Heuristic positions each item in the sheets. This parameter can have a great impact on the solutions, both in terms of quality as well as execution times. In our case we designed the greedy algorithm with the aim of developing an extremely fast method to solve our problem, therefore we decided to use evaluation criteria that are fast to compute, rather than accurate in the evaluation. With the pilot method, on the other hand, we are willing to allow longer execution times in favor of better quality of the solutions.

The criteria considered in Algorithm 8 are related to the compactness of the current solution, in fact they measure the impact on the partial solution induced by placing one item. This can draw to solutions which are well packed in the firsts sheets, but poorly handled in the last ones, because the locality of the evaluation criteria, combined with the greediness of the algorithm, leads to account only for placed items, ignoring the ones that still need to be accommodated. To prevent these kind of situations, one possibility is to introduce an evaluation criterion that has a look-ahead on the final solution, that is to involve also future items to place in the evaluation. In this direction, we make use of the pilot method approach. We measure the placement of one item by the total material waste of the solution composed by the current partial solution with the current item placed in the position under evaluation, and then completed via the greedy algorithm (thus with the compactness criteria, e.g. contact perimeter) applied to the remaining items. This measure enables decisions that are locally worse in terms of

compactness, but better with respect to the material waste of the final solution, which is the primary objective of our problem. In other words, we can say that the pilot method soften the greediness of the greedy algorithm.

In the pilot method evaluation, we have the option of considering or not the remaining optional items in the execution of the look-ahead greedy algorithm. The inclusion of optional items may bring both advantages and drawbacks. Excluding them allows the look-ahead to consider purely on minimizing material waste from compulsory items, which is indeed the focus of the optimization. However, this can result in configurations where, although waste is minimized for the compulsory items, their placement and rotation may prevent the placement of optional items, since they are not considered, which also impacts the objective function. Including optional items in the evaluation prevents this issue, but it can also lead to misleading results, where solutions with fewer compulsory items but bigger optional ones are favored, distorting the waste minimization goal. After conducting computational experiments, we decided to account for optional items in the evaluation, as this approach on average led to better outcomes by guiding the search more effectively.

As regards all the other attributes specific to our problem, the pilot method handles them in the same way as the greedy algorithm: optional items and soft precedence relations are managed by the sorting procedure of the items, orthogonal rotations and punching margins are considered in the generation of extreme points, while hard precedence relations and the safety margin constraint are guaranteed by the feasibility check.

6.4 Beam search heuristics

The beam search method, as presented in Subsection 2.3.3, is a constructive tree search based algorithm that allows to explore multiple solutions simultaneously. There are several components that lead the search, and that can be tailored to the specific problem. We now describe the tree search that we designed to tackle the problem under study, namely the 2DC&PP-PT.

6.4.1 Beam search on one sheet

We apply the beam search method on a sheet-by-sheet basis, therefore let us now focus on the generation of one cutting layout. To handle the presence of optional items, we decided to first perform a tree search on the compulsory items only, and then, starting from the generated cutting layout, to fill it as much as possible with optional items by running another tree search on the set of optional items. This mechanism embraces again the definition of optional items, that are meant to fill the free space where compulsory items cannot fit.

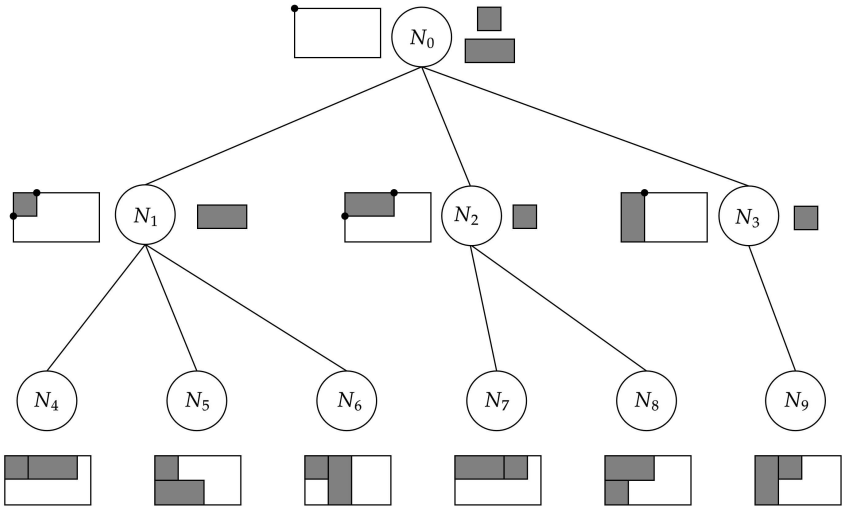


Figure 6.5: Example of a tree search. The root N_0 represents the initial state where the sheet (in white) is empty and there are two items (in gray) to place. Children nodes are generated selecting the combinations $(item, rotation, position)$. In the last level there are six feasible solutions with different configurations.

The algorithm starts building the tree search from a dummy root node representing the initial state (i.e., empty sheet and items to place, compulsory or optional). In the tree, every node represents a partial solution of the current cutting layout, along with a *placing option* corresponding to the combination of the item to place, its orthogonal rotation and position. Therefore, given a node N_i, all its children nodes will have the same partial solution, corresponding to the set of placing options of the nodes in the path from the root node down to N_i, and will differ from each other by the next placing option to be added in the partial solution. Notice that, at each level of the tree, exactly one item is added to the cutting layout. The tree leaves are reached when there are no more remaining items, or when the partial solution cannot accommodate any of the remaining items in any rotation and position combination (i.e., the remaining placing options are all infeasible). A graphic visualization of the tree search is given in Figure 6.5.

Branching process

The expansion of each node in its children nodes is called *branching*. Given a node N_i, we consider all the pairs consisting of the remaining items to place and their allowed rotations. To evaluate the possible placement positions for each combination, we exploit the extreme points notion in the following

way: given the partial solution, and a pair $(item, rotation)$, we retrieve the set of extreme points corresponding to the rotated item, and each one of them will be considered as a potential placement position. The extreme points are recovered by the same procedure used in the Placement Heuristic, previously explained in Section 6.2. Thus, the node N_i will have as many children as the number of combinations $(item, rotation, position)$ given by the remaining items, their allowed rotations, and the corresponding extreme points in the partial solution.

Feasibility check

After the branching phase, all the child nodes are subject to a *feasibility check*. This verifies if the expansion given by the placing option of the node respects the non-overlapping constraints, the containment within the sheet, the distance with respect to the safety margin, and the hard precedence relations, with the same rationale described for the Placement Heuristic (Section 6.2). If a child node results to be infeasible, it is discarded from the search tree.

Filtering

Once all the infeasible children have been deleted, the remaining feasible ones are further selected through the *filtering* process. The filtering is regulated by the *filter width* α parameter of the algorithm, that is the maximum number of children per node that will be considered after the filtering. This step is performed to reduce the computational cost of the algorithm in terms of execution times, while preserving the most promising nodes to explore based on some criteria. In our beam search, we perform the filtering evaluation by relying on compactness measures, namely the amount of contact perimeter of the placing option with respect to the partial solution, and ties are broken by preferring the most top-left placement in the sheet. Moreover, to account for soft precedence relations, children nodes are first ranked with respect to their soft precedence level, and then, in a lexicographic manner, by the compactness criteria. This leads the search to explore solutions that, when it is feasible and convenient, respect the soft precedence relations. If the number of children is smaller than the filter width, this step is skipped, because all children would be considered anyway, so the computational cost of the filtering evaluation is avoided.

Beam selection

The search tree is built by a breadth-first strategy, thus, at each level, every node is expanded by following the steps we have just described (i.e., branching, feasibility check, and filtering). To focus the search on the most promising regions of the solution space, when moving from one level to the

next one, the *beam selection* is performed. This process involves all the filtered nodes of the current tree level, independently from their parent node. These nodes are ranked by the beam selection evaluation, and only the best β will be branched in the next level, where β is the *beam width* parameter, while the other ones are discarded. In beam selection, as in the filtering, if the number of nodes involved is less than the beam width, the evaluation is skipped and all the nodes are kept. The choice of the beam selection evaluation has a great impact on the final solution, since it decides on which nodes to expand, thus guiding the search to some areas of the solution space, while excluding others. We decide to adopt the idea underneath of the pilot method for this delicate step. Recall that, in the pilot method, we rank the placement of one item according to the overall material waste of the partial solution that has been completed via the greedy algorithm. In the beam search, on the other hand, we are building one sheet at the time, so as beam selection evaluation we consider the material waste of the cutting layout that is currently being generated, completed with the greedy algorithm. Limiting the completion of the solution has also a positive impact on the required execution time for the beam selection evaluation, that, since it is called many times throughout the algorithm, is important to be reasonably short.

When there are no more nodes to branch, the best leaf is selected according to the primary objective of the problem, that is the node representing the cutting layout with the lower amount of material waste. Notice that the leaf nodes do not necessarily correspond to the beam nodes at the final level, although they do include them. Each level represents the placement of a new item in the layout, therefore layouts at different levels contain a different number of items. As a result, depending on the placed items, leaf nodes can appear at earlier levels than the final one in the search.

Including optional items

When the tree search is completed on the set of compulsory items, the best cutting layout is then filled with optional items, if any, by a similar tree search.

Summary of the one-sheet procedure

The one-sheet procedure is summarized in Algorithm 9. This algorithm focuses on constructing a cutting layout (*best_layout*) for a single sheet by applying a beam search strategy. It processes both compulsory items (*items*) and optional items (*optionals*, if any), and its parameters include the filter width (α) and beam width (β). The layout (*layout*) provided to the algorithm can either be empty (when starting with compulsory items) or partially filled (when refining the layout by adding optional items).

The algorithm begins by initializing two sets, *leaf_nodes* and *beam_nodes*, both of which are initially empty (Steps 2–3). The *leaf_nodes* set will store terminal nodes representing completed layouts, while the *beam_nodes* set holds the nodes being actively explored at each level of the search tree. The given *layout* is added as the initial node to *beam_nodes* (Step 4), forming the root of the search tree.

The core of the algorithm is the while-loop (Step 5), which iterates as long as *beam_nodes* is not empty. At the start of each iteration, an empty set, *level_nodes*, is initialized to store the nodes generated at the current level of the search tree (Step 6). The algorithm then iterates over each *node* in *beam_nodes* (Step 7) and applies the branching process (Step 8) to generate *children_nodes*. If a *node* generates no children, it is considered terminal and is added to *leaf_nodes* (Steps 9–12). To manage the search's complexity, the algorithm filters *children_nodes* to retain only the α best nodes based on a filtering evaluation (Step 13). The filtered nodes are then added to *level_nodes* (Step 14). If *level_nodes* becomes empty (Steps 16–18), the algorithm terminates the current beam search iteration early. After completing the evaluation of all nodes in *beam_nodes*, the algorithm selects the β best nodes from *level_nodes* using a beam selection evaluation (Step 19). The selected nodes replace the current *beam_nodes* (Step 20), forming the basis for the next level of the search tree.

When the while-loop concludes, the algorithm identifies the best node in *leaf_nodes* and extracts the corresponding cutting layout as *best_layout* (Steps 22–23). If optional items remain to be placed, the algorithm is repeated (Step 25) with these items, the partially filled *best_layout*, and the same parameters α and β. If no optional items remain, the algorithm returns the *best_layout* as the final solution (Step 27).

Notice that the constraints specific to the 2DC&PP-PT under study are considered by the algorithm. In particular, the presence of optional items is handled through the two consecutive tree searches, while the allowed orthogonal rotations are accounted in the branching, where also the the punching margins of each item are considered indirectly thanks to the generation of extreme points. Moreover, the feasibility check ensures that hard precedence relations (for compulsory items only) and the safety margin constraint are satisfied. Finally, soft precedence relations are addressed directly in the filtering evaluation, and indirectly in the beam selection evaluation, where the sorting of the items is primary based on soft precedence levels.

6.4.2 Beam search for 2DC&PP-PT

The *Beam Search* (BS) method that we propose to solve the 2DC&PP-PT implements the tree search presented in Subsection 6.4.1 to build each cutting layout in the solution, one at a time. We embed Algorithm 9 in a greedy scheme, resulting in Algorithm 10.

Algorithm 9 Beam Search for one sheet

```
1: procedure BeamSearchOnSheet(items, optionals, layout, α, β)
2:     let leaf_nodes be the empty set of leaf nodes;
3:     let beam_nodes be the empty set of beam nodes;
4:     add the node representing the layout to beam_nodes;
5:     while |beam_nodes| > 0 do
6:         let level_nodes be the empty set of the current level nodes;
7:         for each node in beam_nodes do
8:             generate children_nodes of node (branching);
9:             if |children_nodes| = 0 then
10:                add node to leaf_nodes;
11:                continue;
12:            end if
13:            filter α best children_nodes using filtering evaluation;
14:            add children_nodes to level_nodes;
15:        end for
16:        if |level_nodes| = 0 then
17:            break;
18:        end if
19:        select β best level_nodes using beam selection evaluation;
20:        replace beam_nodes with selected level_nodes;
21:    end while
22:    let best_node be the best node among leaf_nodes;
23:    let best_layout be the cutting layout corresponding to best_node;
24:    if |optionals| > 0 then
25:        return BeamSearchOnSheet(optionals, ∅, best_layout, α, β).
26:    else
27:        return best_layout.
28:    end if
29: end procedure
```

Given the items to place, both compulsory (*items*) and optional (*optionals*), and the BS parameters, namely the filter (α) and the beam (β) widths, the algorithm begins by initializing an empty set, *solution*, which will store the sequence of cutting layouts forming the final solution (Step 2). It operates within a while-loop (Step 3) that iterates until all compulsory items have been placed. This stopping condition prevents having cutting layouts consisting only of optional items, which is not allowed by the definition of the problem.

At each iteration, the algorithm first checks whether any material sheets remain available (Steps 4–6). If all sheets have been used, the procedure terminates and returns the partial *solution* constructed so far. Otherwise, a new empty *sheet* is selected (Step 7), and the `BeamSearchOnSheet` subrou-

tine is invoked (Step 8), which generates a feasible cutting layout (*layout*) for the current set of items, starting from the selected empty sheet. The layout is then added to the *solution* (Step 9), and the *items* and *optionals* sets are updated by removing the items successfully placed on the current *layout* (Step 10). Upon completion of the while-loop, the algorithm returns the *solution* (Step 12), which represents the complete sequence of cutting layouts.

Algorithm 10 Beam Search

```
1: procedure BS(items, optionals, α, β)
2:     let solution be an empty set of cutting layouts;
3:     while |items| > 0 do
4:         if no available sheets remain then
5:             return solution, partial solution found (out of sheets).
6:         end if
7:         select a new empty sheet;
8:         let layout ← BeamSearchOnSheet(items, optionals, sheet, α, β);
9:         add layout to solution;
10:        update items and optionals deleting the placed ones;
11:    end while
12:    return solution, final solution completed.
13: end procedure
```

6.4.3 Beam search with double search effort

Since the problem under study arises from an industrial application, the related instances to be solved can have variable size (e.g., number of items, number of cutting layouts to produce, etc), as well as different execution time limitations. To account for these aspects, we propose to embed the beam search algorithm presented in Subsection 6.4.2 into an iterative procedure that stops subject to a time limit condition, and that varies the beam width throughout iterations. The procedure is sketched in Algorithm 11.

The algorithm begins by initializing a timer (*stop_watch*) to zero (Step 2). The initial solution is computed using the Greedy Algorithm (Step 5), which quickly provides a feasible solution that may be further improved. The set of all items, consisting of both compulsory (*items*) and optional (*optionals*) items, is combined into *all_items* (Step 4). This initial solution serves as a baseline, ensuring that the algorithm returns a result even for large instances or short time limits.

The next step initializes the beam width β to 1 (Step 6). The outer loop runs until the time limit τ is reached (Step 7). In each iteration, the BS procedure (Algorithm 10) is invoked with the current value of β to generate

a *solution* (Step 8). If the solution obtained improves over the current best solution (*best_solution*), it is stored as the new best solution (Steps 9–11).

The core feature of this algorithm is the dynamic adjustment of the beam width, obtained by following the idea proposed by Araya and Riff in [3]. After each iteration, β is updated by multiplying it by $\sqrt{2}$, and then rounding it up to the nearest integer (Step 12). Increasing the beam width at each iteration by a $\sqrt{2}$ factor is called *double search effort mechanism*, and that is because the search tree will, in principle, simultaneously consider double the number of solutions with respect to the previous iteration. In this way, we do not have to set the parameter β, which may be a hard task, in principle, but we just need to set a time limit (τ), which is more directly related to the need of the user in our context. We refer to this variant as the *Beam Search with Double Search Effort* (BS-DSE).

The algorithm terminates when the elapsed time reaches the specified limit τ, thus the best solution found during the search is returned (Step 14).

Algorithm 11 Beam Search With Double Search Effort

1: **procedure** BS-DSE(*items*, *optionals*, α, τ, *sort_eval*, *place_eval*)
2: start timer: let *stop_watch* be 0;
3: let *best_solution* be the current best solution;
4: let *all_items* be the union of *items* and *optionals*;
5: *best_solution* $\leftarrow$ `GreedyAlgorithm`(*all_items*, *sort_eval*, *place_eval*);
6: initialize beam width $\beta \leftarrow 1$;
7: **while** *stop_watch* $< \tau$ **do**
8: let *solution* $\leftarrow$ `BS`(*items*, *optionals*, α, β);
9: **if** *solution* improves over *best_solution* **then**
10: update *best_solution* to *solution*;
11: **end if**
12: increase beam width $\beta \leftarrow \lceil \sqrt{2}\beta \rceil$;
13: **end while**
14: **return** *best_solution*.
15: **end procedure**

Chapter 7

Towards irregular shapes

In this chapter, we analyze a different variant of two-dimensional packing problems. Referring to the typology introduced by Wäscher et al. [88], that has been discussed in Section 4.1, this problem belongs to the basic type of *Open Dimension Problem* (ODP). In most of the cases treated in literature, ODP considers a single large object that has one or more variable dimensions, and a set of small items that must be placed in it. The general goal of the optimization is to place all the small items in the large object by minimizing a measure of its variable dimension(s), e.g., the required area, while respecting the classical geometric constraints of non-overlapping of the items and non-exceeding the large object boundaries. We focus on the two-dimensional problem where the large object is indeed unique and it is called a *strip*, since it has a rectangular shape characterized by a fixed height and an infinite width. The small items to be placed in the strip are general *simple polygons*, thus irregularly shaped objects without holes, that have an assigned finite set of allowed rotations. The objective of the problem is to place all the items in the strip by minimizing its width, such that the overall material waste is minimized accordingly. This variant of the ODP is commonly referred to as the *Irregular Strip Packing Problem* (ISPP) or as *the Nesting Problem* (respectively as in, e.g., [44, 69]). A straightforward and common field of application of the ISPP is the textile industry (see Figure 7.1), where irregular garment's pieces must be cut out of fabric rolls. Nevertheless, other real-world scenarios involving ISPP can be found, such as, for instance, in the sheet metal industry, as will be discussed in the following.

We investigate the ISPP in this chapter of the thesis for two intertwined reasons, which are of practical and theoretical interest, respectively. We recall that the reference company of this work operates in the sheet metal industry, and, among other services, produces cutting machines for the sheet metal. As presented in Section 3.2, the company produces essentially two types of cutting machines, that differ from each other by the adopted cutting

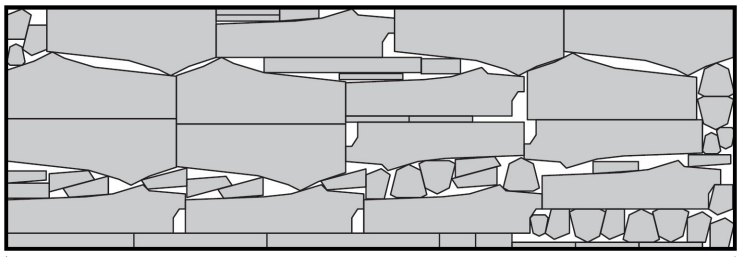


Figure 7.1: Example of irregular C&PP in the textile industry [30].

technology, which induces two different problems related to the generation of cutting layouts. On the one hand, punching machines refer to the problem defined in Section 3.3 as a 2DRBPP with practical constraints specific to the punching technology (i.e., 2DC&PP-PT), where the parts can be represented by their axis-aligned bounding box due to the unloading process of the parts through the integrated shear. On the other hand, laser cutting machines are not equipped with the unloading shear, thus the approximation of the parts with the corresponding rectangles is no longer valid. As a result, generating cutting layouts for laser cutting machines requires handling irregularly shaped parts, which are typically polygons and may include holes. In line with the terminology used in the literature, in this chapter, the term *item* will refer to the simple polygon representing each part, defined by its outer contour. The cutting machines used by the reference company are designed to load material sheets. However, some clients prefer to buy metal in coils and cut them into sheets using third-party machines, as purchasing coils is often cheaper than buying pre-cut sheets. For this reason, the company has a strategic interest in exploring the strip packing problem. By offering this service, they can help clients to nest pieces and to cut metal coils according to their needs, without being limited by the standard dimensions of pre-cut sheets.

The decision to focus on the strip packing problem, rather than the bin packing one, is also motivated by the following theoretical question. The solution method that we propose as an alternative to the procedure currently adopted by the company, competitive both in terms of running times and solution quality, is the BS-DSE algorithm described in Subsection 6.4.3. At its core, this method relies on a beam search strategy, which in the literature has been successfully applied to tackle the ISPP by Bennell et al. [13]. Therefore, our research intends to investigate whether the beam search algorithm that we propose can be extended to the irregular strip packing variant. Moreover, we investigate a new way of integrating, into a heuristic approach for ISPP, the recent geometric framework, `jagua-rs` [35], to deal with collision detection and other geometry-related tasks.

In this chapter, we provide an overview of the literature on the ISPP

(Section 7.1) and discuss the geometry tools commonly used to address the irregularity of items (Section 7.2). We then introduce our constructive and beam search-based solution approaches for solving the ISPP (Section 7.3), with particular attention to a strategy we developed for handling overlapping items in our specific context (Section 7.4), a task induced by the use of the new geometric framework `jagua-rs`.

7.1 State of the Art on the Irregular Strip Packing Problem (ISPP)

In the Operations Research literature, the ISPP is a variant of the C&PP that has been extensively studied due to its versatility in several application fields. Both model-based exact methods and heuristics have been proposed by researchers to tackle this problem, as well as, in recent years, hybridizations of the two (i.e., matheuristics). All the solution approaches deal with the geometric challenge that comes with the irregularity of the items, in particular, the check for non-overlapping constraints involves challenging theoretical elements and generally requires a great computational effort, which strongly affects the devised methods. The tools to handle the geometric aspect of the problem will be discussed in the next section.

Several mathematical formulations of the problem have been proposed over the years. A thorough review on mathematical models is given by Leao et al. [55], where it can be seen how different modeling approaches have been studied. Discretization-based formulations are introduced, e.g., in [86, 79], where the strip is discretized in a *dotted-board*, thus representing an approximation of the real problem, since the position of the items is chosen over a finite set of discrete points. On the other hand, e.g., in [18], exact methods are proposed based on MILP formulations of the problem, where the placement variables of the items are continuous. A combination of these two approaches is proposed in [54], where the y-axis is discretized, while the x-axis is not, therefore allowing continuous translations in such direction. Although mathematical formulations are now considered also in heuristic algorithms, yielding matheuristic approaches (e.g., [47]), model-based methods are usually suitable for small to medium sized instances, thus not always applicable to real-world scenarios.

In this direction, heuristic approaches are the most commonly used in application contexts, and a detailed overview on the ones related to the ISPP can be found in [12]. Heuristic methods, as discussed in Section 2.3, can be grouped into two classes based on how the search for the solution is performed. One class gathers the improvement heuristics, which start from a complete solution and iteratively try to improve it. The way in which the solution is perturbed from one iteration to the next can follow different strategies. In some approaches, overlap between the items and exceeding the

strip boundaries are allowed to let the algorithm better search the solution space and to escape local minima. In these cases, generally, a post process in performed on the infeasible solution to redeem its feasibility. The other class involves constructive heuristics, where items are sorted in a sequence and placed one at a time through a placement procedure. The sequence can be either static, where the order is fixed at the beginning of the algorithm, or dynamic, where the remaining sequence can be modified accordingly to the current partial solution. The placement procedure typically selects in a greedy way the best position of the item according to some criteria (e.g., bottom-left placement).

Among heuristics, we can find the beam search method (Subsection 2.3.3). In the cutting and packing literature, this method has been applied to the ISPP, to the best of our knowledge, only by Bennell et al. [13]. In this work, the authors build the beam search tree by associating nodes to partial solutions and by branching on the remaining items. They filter the child nodes with local evaluation (related to the current partial solution), and they select the beam nodes through a global evaluation (that accounts for the remaining items). To face the geometrical aspects, the placement heuristic adopted in [13] is a revised version of the TOPOS algorithm [69].

In the work by Elkeran [30], a different heuristic method for ISPP is proposed, based on cuckoo search combined with guided local search algorithms, and integrated by pairwise clustering techniques for irregular items that aim at finding compact configurations for any given pair of items. The results presented in [30] represents, to the best of our knowledge, the state-of-the-art on a wide range of classical benchmarks for ISPP.

Regarding improvement heuristics, we report, among others, the recent work by Gardeyn et al. [37], which implements a ruin-and-recreate iterative strategy that destroys and rebuilds the solution at each iteration. The solution improvement is guided by a local search that dynamically ranks items, where these rankings are used to select the items to remove and replace within the ruin-and-recreate mechanism. This heuristic fully relies on `jagua-rs` [35], proposed by the same authors, for representing irregular items and handling geometry-related tasks.

7.2 Geometry tools

As anticipated in Section 7.1, in irregular C&PP, the presence of simple polygons asks for the development of geometric tools to represent the problem and, in particular, to perform the overlapping check. A comprehensive review on the notions that have been devised to handle this common challenge is given by Bennell and Oliveira in [11]. When dealing with real-world applications, the needs and limitations that come with the problem can be several (e.g., limited time or resources to find a solution, or the need for

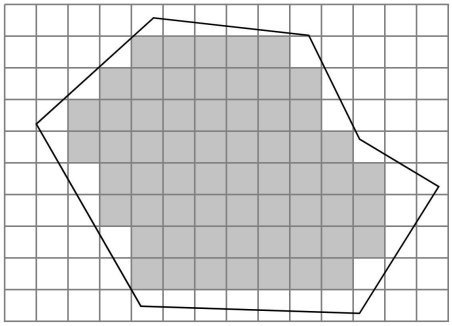


Figure 7.2: Example of a raster representation of a simple polygon.

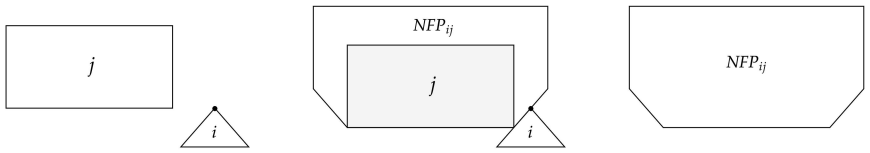
high quality solutions), and the geometric tool should be selected accordingly. The representation of the items affects the corresponding technique to check for their overlapping, which is the ultimate aim of the representation and the great common challenge when dealing with irregular problems. We briefly report the geometric tools typically adopted for the ISPP, namely the *raster method*, the *no-fit polygon*, and the *direct geometry* (a detailed discussion can be found in [11]). We also introduce a recently proposed geometric framework, `jagua-rs` [35], able to address the geometric aspects of irregular C&PP, with a specific focus on collision detection.

Raster method

The *raster method* is based on the discretization of the items and the strip through a grid of pixels (see Figure 7.2), where each pixel has an associated value that expresses if the related area is free, occupied by an item, or occupied by more than one item, thus identifying overlap (see, e.g., [83, 75]). This method is robust and simple to implement, but it heavily depends on the resolution of the grid. On the one hand, small resolution grids are fast to be computed, but they lead to lack of precision in the representation, thus generally leaving unnecessary free space among items. On the other hand, with higher resolutions the grid is more accurate, but the required computational effort badly affects the efficiency of the method.

No-fit polygon

The *no-fit polygon* (NFP) concept was introduced by Art [5] and it represents the area on the strip where positioning an item would result in overlapping other items or exceeding the strip boundaries. Notice that the NFP depends on the item to place, therefore for each item, a NFP must be computed. In this representation, each item is identified with a reference point, and checking if placing it causes overlapping is done by verifying if its reference



(a) (b) (c)

Figure 7.3: Example of two items i and j (a), the generation of the no-fit polygon of i with respect to j (b), resulting in NFP_{ij} (c).

point lies inside the respective NFP. An example of NFP is provided in Figure 7.3. Moreover, the NFP provides information on positions where the item touches the already placed items, which is a valuable indication when looking for compact solutions. Although the NFP approach is the most used to handle the geometry of irregular C&PP due to its strengths, it requires a medium-high computational effort, and in some situations can lack of robustness.

Direct geometry

Methods based on *direct geometry* (i.e., trigonometry) offer precision, flexibility, and robustness, and several strategies have been developed in this direction (e.g., D-function [52], circle-based representations [78], as the one shown in Figure 7.4). Nevertheless, these approaches are hard to be efficiently implemented and are associated with high computational costs in terms of execution times, which would make them not suitable for practical use.

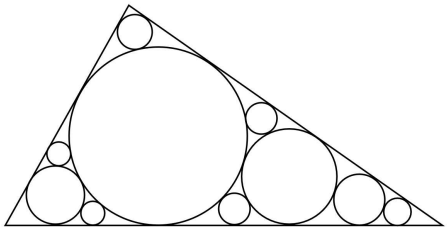


Figure 7.4: Example of a circle-based internal representation of a simple polygon.

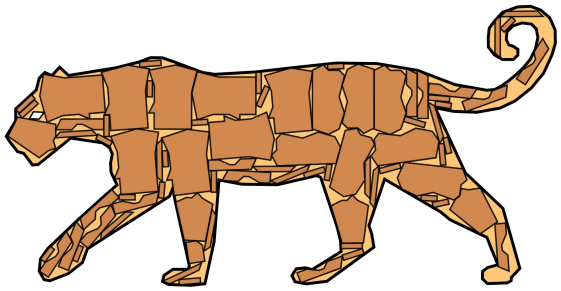


Figure 7.5: `jagua-rs` framework logo [35].

The `jagua-rs` framework

A new geometric tool to address the overlap of irregular items has been recently proposed [36]. It is an open-source framework, called `jagua-rs` (see Figure 7.5), designed to detect collisions (i.e., overlap, intersections) between general simple polygons, and it is publicly available at [35]. The framework is based on trigonometry notions to ensure precision and robustness, while the authors focused on the design and the implementation of the tool to guarantee computational efficiency. Entirely written in Rust [80], an emerging high-performance compiled programming language, `jagua-rs` implements a fast-fail strategy in two consecutive steps: a broad phase that detects severe collisions through rough and inexpensive checks, followed, when no collision is detected, by a more expensive narrow phase which validates the overlap with precision. Thanks to its structure, this geometric tool proves itself to be extremely fast and reliable. The framework can be queried by any algorithm using it to check if, given a partial solution, placing one further item in a given position would cause a collision. Moreover, thanks to the inner structures implemented in `jagua-rs`, other valuable information can be retrieved concerning the collision (e.g., where and how it occurs). On the other hand, designing the way in which exploiting the framework to look for valid placements is not trivial. In fact, the framework does not provide direct information on where can be convenient to test the placement, unlike, for example, the indications provided by the boundary of the NFP.

The features of the framework, including both its advantages and drawbacks, make designing heuristics based on `jagua-rs` challenging, e.g., straightforward extensions of previous approaches in literature are not viable, due to the lack of some functionalities (although replaced by new query opportunities). To the best of our knowledge, only one such method has been proposed so far by the same framework's authors [37].

7.3 Heuristics for ISPP based on the `jagua-rs` geometric framework

In this section, we propose an extension of the constructive and beam search heuristic methods presented in Section 6.3 and 6.4 to solve the ISPP. To handle the geometric aspect of the problem, we adopt the `jagua-rs` framework, that has been previously presented. In the following, we discuss one possible way to exploit the recently proposed framework, instead of the more classic geometric tools (e.g., NFP).

7.3.1 A `jagua-rs`-based placement heuristic

In analogy with the rectangular case, the heuristics that we aim to extend to irregular shapes present the sub-problem of placing one item into a given partial layout. The algorithm that we propose to solve this sub-problem will be referred to as the *Placement Heuristic for ISPP* (IPH).

Given a partial solution and an irregular item to be added with a fixed orientation, we want to find the best feasible position according to the minimization of a cost function. As mentioned in the previous section, `jagua-rs` does not provide straightforward indications on feasible or promising placement areas within the partial solution, instead, given a candidate position, the framework can be efficiently queried to check if the placement is feasible (i.e., no collision is detected). Therefore we need to adopt a procedure to select promising candidate positions, and, to this end, we start by following the idea proposed by the authors of the framework [37]. In a pre-processing phase, a *pairwise clustering table* is generated, where, for each pair of items, the best mutual positions are stored, according to compactness criteria (refer to [30] for further details on the clustering process). Thus, the positions provided by the clustering table of the item to place with respect to the items in the partial solution, are promising candidate positions to test for feasibility. Moreover, reference [37] suggests to uniformly sample in the two coordinates a reasonably high number of positions in the strip, to search also areas that are not necessarily close to any item already in the partial solution. Then, the best feasible position sampled at uniform is translated towards the boundary of the partial solution through a coordinate descent walk [62]. The descent is meant to refine the position, aiming at making the item touch the partial solution for the sake of compactness. We remark that, as we will discuss in Section 7.4, the search of a refined position may even star from an infeasible sampled position, using a more sophisticated descent that further exploits the functionality of `jagua-rs` to reduce and possibly escape temporary item overlaps. The refined position is finally compared to the best feasible one provided by the clustering table (if any), and the best one among the two is fixed for the new item in the partial solution. The feasibility of the tested positions is checked by `jagua-rs`, and the quality of

the placements, as well as the comparison between one another, is measured by the minimization of a cost function. As cost function we propose the sum of the following local compactness criteria: width of the partial solution, normalized with the width of the current item's bounding box; area of the partial solution's bounding box, normalized with the area of the current item's bounding box; minus the overlapping area between the current item's bounding box and the bounding box of the other placed items, normalized with the area of the current item's bounding box. The bounding box in these criteria is always considered to be axis-aligned with the coordinate system induced by the strip's edges. Ties are broken by the weighted combination of the maximum x-coordinate and y-coordinate of the current item's bounding box. The compactness criteria that are summed in the cost function are taken from [13]. Due to the locality of these measures, we will refer to this cost function as the *local evaluation*, which is intended to be minimized.

7.3.2 A greedy algorithm for ISPP

The IPH can be exploited in a straightforward way within a greedy procedure. Similarly to Algorithm 8, items to place can be sorted according to some criterion and placed one at a time. Sorting heuristics are, again, generally related to characteristics of the items themselves or of their convex hull or rectangle enclosure [69]. When placing one item, all its allowed rotations are evaluated, the corresponding placement positions are computed with the IPH, and the best rotation-position combination is chosen based on placement evaluation criterion. Recall that, in the ISPP, the strip does not have fixed size, therefore the placement evaluation commonly refers to the impact of the current item to the partial configuration of the items, measured by evaluating how the configuration changes or measuring the compactness of the new placement [69]. The algorithm stops when all items have been placed. We refer to this procedure as the *Greedy Algorithm for ISPP* (IGA).

7.3.3 A beam search for ISPP

In this subsection, we extend the beam search algorithm presented in Subsection 6.4.1 to solve the ISPP. We recall that the search is limited by the two classical parameters that are the filter width and the beam width, which are related to the filtering and beam selection processes, respectively (see Subsection 2.3.3). The proposed method is structured as Algorithm 9, where the placement heuristic, the branching operation and the evaluation criteria related to the filtering and the beam selection have to be adapted to the tackle the ISPP.

To place items with fixed rotation in a partial solution, we refer to the placement heuristic IPH presented in Subsection 7.3.1. As regards branching process, given a partial solution represented by a node, and the remaining

items to place, we generate one child node for every combination defined by one remaining item and one of its allowed rotation. Therefore, the branching option *(item,rotation)* expresses the willingness to place the item with a fixed orientation into the partial solution of the parent node. The placement position of the branching option is computed through the IPH, and, if no feasible position is found, the node is discarded.

We propose two versions of the algorithm, that differ from each other by the pruning evaluation criteria that are used in the filtering and in the beam selection processes. The first version is a faster version that includes inexpensive evaluation criteria, while the second one implements the intuition of having a look-ahead on the final material waste when building the solution, resulting in a more accurate version. We now define the two versions of the algorithm by specifying the corresponding evaluation criteria.

Basic beam search for ISPP

The *Beam Search for ISPP* (IBS), filters the tree nodes by item area, where the items with a bigger area are preferred. The rationale is that placing bigger items first, which are harder to fit in partial solutions, leaves free spaces that could be filled by smaller items. The beam selection is performed by minimizing the same fast local evaluation criterion used by the IPH, among the ones presented in Subsection 7.3.1. We recall that this criterion accounts not only for the item under evaluation, but also for its overall compactness that can be achieved with the partial solution.

Beam search with global evaluation for ISPP

In the *Beam Search with Global Evaluation for ISPP* (IBSGE), the filtering is performed based on the local evaluation (smaller values first), namely the cost of placing the branching option in the partial solution of the parent node via the IPH. In other words, the filtering criterion here is the criterion that the basic IBS adopts for beam selection. As beam selection evaluation, similarly to Algorithm 9, we compute the waste percentage of the solution given by the partial solution of the node (including the branching option), that has been completed via the IGA (Subsection 7.3.2) that sorts the items by descending area, and places them through the IPH. We call this kind of metric *global evaluation*, since it considers a simulated complete solution. The IBSGE is similar to the algorithm proposed by Bennell et al. [13] in the evaluation criteria, since both algorithms adopt local measures for the filtering process, and global ones for the beam selection.

7.3.4 A beam search with double search effort for ISPP

Both versions of the beam search algorithm for ISPP have been implemented as presented above, and, similarly to Algorithm 11, the corresponding vari-

ants based on double search effort mechanism, and therefore subject to time limit, have also been developed and tested. We recall that the adaptation of the algorithms to flexible time limits arise from the industrial application context, since the limitations in execution times can sensibly vary with the needs of different clients. We refer to the procedure integrating the IBS (resp., the IBSGE) in an iterative scheme implementing the double search effort mechanism as the *Beam Search* (resp, *with Global Evaluation*) *for ISPP with Double Search Effort* (IBS-DSE, and, reps., IBSGE-DSE).

7.4 Redeeming overlapping items

During the preliminary computational tests on the proposed algorithms, and, in particular, on the placement heuristic IPH, we have drawn the following observations on using the `jagua-rs` framework as a substitute of classical geometric tools.

The framework has a complex structure, composed of different components that contribute all together to the collision detection, but that can also be further exploited to retrieve valuable information on the packing status of a partial solution. Despite the elaborated nature of the framework, the open-source code is polished and well documented, making it accessible to new users and allowing them to directly integrate it into their algorithms, as well as to refine it with additional improvements, tailored for their use.

The element of `jagua-rs` dedicated to testing overlap is called *Collision Detection Engine* (CDE), and it can be modulated through a set of parameters based on the solution requirements [36]. The CDE, in our computational tests, has proven itself to be precise and reliable. Moreover, querying the CDE is extremely fast, allowing for a high number of collision checks for each placement.

We observe that the integration of the CDE into an optimization algorithm should depend on the nature of the algorithm. This comment is motivated by the fact that the type of the algorithm used on top of the framework highly affects the characteristics of the partial solutions on which the CDE is queried. As presented in Subsection 7.3.1, to place an item by using `jagua-rs`, our IPH directly adopts the procedure suggested by the authors of the framework in [37] (i.e., choose the best position among pairwise clustering samples, and uniform samples refined via coordinate descent). However, in [37], the procedure supports an improvement heuristic that has a ruin-and-recreate strategy at its core, where, during the search, partial solutions are likely to be almost filled, leaving small and well-defined areas for placing a new item. On the contrary, in a constructive heuristic (e.g., IBS or IBSGE), the partial solutions are almost empty at the beginning, thus presenting vast placement areas. Therefore, given the same number of uniform samplings for placing one item, in the first kind of heuristic the

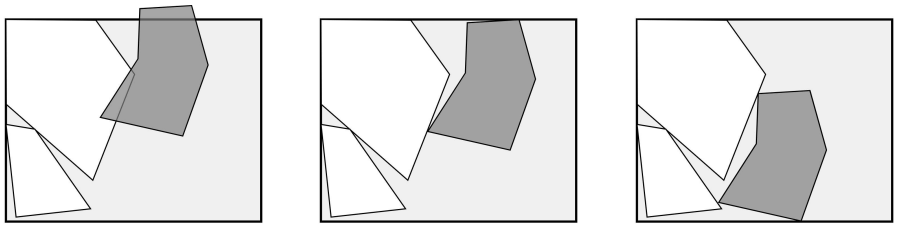
chances of getting a good placement position are higher than in the second one. Moreover, with the ruin-and-recreate heuristic, each item has the possibility to be removed and placed again multiple times, proportionally increasing the number of samples that are tested for each item, and, consequently, the success rate of the sampling. Notice that this possibility of rehashing the placement of one item, in a one-pass constructive heuristic, as well as in IBS or IBSGE, is not given.

Based on these observations, to better interact with the CDE in our constructive heuristic, we propose to accept infeasible positions during the sampling process, and exploit the overlapping information provided by the inner components of the `jagua-rs` framework to guide a descent method that moves the sampled point towards the profile of the partial solution up to a feasible position. In addition, once a feasible position is found, a local search could be performed along the directions given by the edges of the placed items that are touching the current item, to slide it along the profile of the partial solution looking for a better placement position. These observations are inspired by the concept of the no-fit polygon and aim at reproducing the indications given by its border.

We have investigated the first idea by devising a procedure to redeem infeasible samples to feasible ones. To this end, we introduce an *overlap metric* based on an internal approximation of the polygons provided by `jagua-rs`. In the framework, each polygon is approximated by a set of fully contained non-overlapping circles (called *poles*) that cover a configurable high ratio of the area, and by a set of fully contained lines that capture the long and narrow sections of the polygon.

The overlap metric measures the extent of interaction of the current item with the placed items or the strip boundary of the partial solution, which we will consider as *obstacles* in the following. This metric evaluates the degree of overlap between the item and surrounding obstacles relying on the circle-based approximation, considering both potential collisions (maximum overlap) and non-overlapping distances (minimum separation).

For each pole representing the item, the distances to the poles of all obstacles are calculated. If the distance between the centers of two poles is smaller than the sum of their radii, then the poles overlap. In cases where one pole is fully contained within another, the overlap is defined as twice the radius of the smaller pole. Otherwise, the overlap is determined by the difference between the sum of the radii and the distance between the centers. This process is repeated for all pairs of poles, tracking the maximum overlap encountered. In situations where the poles do not overlap, the calculation captures the separation as a negative value. This negative overlap is defined as the distance between the poles, minus the sum of their radii, representing the non-overlapping distance. The smallest negative overlap is recorded to reflect the closest possible separation. The final overlap metric is derived by considering both the positive and negative overlap results. If any positive



(a) (b) (c)

Figure 7.6: Example of the redeeming-infeasibility process - starting from overlapping position (a), a feasible position is retrieved (b), and further refined (c).

overlap exists, the metric reflects the maximum overlap. If no overlap is detected, the metric is based on the smallest negative overlap, indicating the closest separation between the item and the obstacles. This metric provides a concise measure of interaction, where positive values indicate overlap and negative values represent separation.

The procedure to redeem infeasible samples aims to adjust the item's position so that it does not overlap with any obstacles, while keeping the separation between them as minimal as possible. Ideally, the metric should be negative but as close to zero as possible, reflecting a position where the item is just avoiding overlapping with the obstacles.

To achieve this, a coordinate descent algorithm is employed, based on the general algorithm detailed in [62]. The algorithm we propose iteratively adjusts the item's position by moving it along axis-aligned directions and diagonals. Each adjustment aims to reduce the overlap metric, guiding the item towards a feasible position. The algorithm is parametrized by a step size, which controls the magnitude of the position adjustments.

The step is dynamically adjusted based on the performance of the algorithm. On the one hand, when the overlap metric improves, the step size is doubled, allowing for more significant adjustments in the item's position. On the other hand, if the overlap metric worsens, the step size decreases, enabling finer adjustments. The algorithm tracks consecutive worsening steps, and when the step size becomes too small after many consecutive failures to improve, the process terminates, since further improvement is unlikely.

This iterative process tries to redeem an initially infeasible position (Figure 7.6a), leveraging the overlap metric as a guide to refine the item's position until one close to the partial solution boundary is found (Figure 7.6b). Moreover, once infeasibility is redeemed, a further refinement can be performed, based on the compactness criteria (Figure 7.6c). Nevertheless, at the end of the procedure, the position may still be infeasible, in this case the sample is discarded.

Chapter 8

Computational results

This chapter presents the results of an extensive computational campaign, conducted on real-world and realistic instances, to assess the solving procedures proposed in this work. A critical component of this study involved devising realistic benchmarks that accurately reflect the diversity of production among the reference company's clients. This proved to be a challenging task due to the wide range of production fields in which these clients operate, each with its own unique requirements and limitations.

The primary goal of the computational tests was to generate optimality bounds and evaluate the efficiency of the algorithms proposed to tackle the 2DC&PP-PT in comparison to the company's current outsourced software solutions. In addition, preliminary results are provided for the irregular problem (ISPP), which was tested using benchmarks from the literature. The tests also helped in fine-tuning the parameters of the algorithms, leading to insightful observations and further refinement of the approach.

Moreover, at the beginning of the chapter, we introduce the tools employed throughout the implementation and the computational testing processes.

8.1 Implementation tools

In this section, we present the technologies adopted for the implementation of the algorithms devised for the problems addressed in this thesis (i.e., 2DC&PP-PT and ISPP), along with the tools used for the following computational analysis.

Version Control System

Version control systems (VCS) are essential tools in modern software development, enabling multiple developers to collaborate efficiently on codebases. These systems track and manage changes to source code over time, allowing

Figure 8.1: Version Control System - Git.

developers to work on different parts of a project simultaneously without conflict. Git [39] (see Figure 8.1) is one of the most widely used VCS, it is a free and open-source software, representing the foundation of many development workflows. It can be accessed through a command line interface, although an increasing number of tools offer a graphical interface for managing Git repositories (e.g., Sourcetree [6]). By maintaining a history of revisions, VCS facilitate easy rollback to previous versions, improving code stability and reducing the risk of errors. Additionally, they provide a structured way to merge contributions from various developers, ensuring that the most up-to-date and stable version of the code is always accessible.

Programming languages

The proposed algorithms for tackling the 2DC&PP-PT were first implemented in Python [74] (see Figure 8.2) for prototyping. Python is a high-level programming language known for its simplicity and readability, which makes it an ideal choice for prototyping and experimentation. The decision to use Python in the beginning, was driven by the need for a fast and flexible language that allowed us to explore various solution methods, algorithmic variants, and parameter configurations with ease.

Python's extensive ecosystem of libraries played an important role in this process. Libraries such as `json` [50] and `re` [76] were used for parsing input and output data, while `matplotlib` [64] provided graphical representations of the solutions. Additionally, `pandas` [71] was used to analyze the computational results efficiently. Despite Python's strengths, one of its main drawbacks is efficiency, particularly in terms of execution speed.

To address efficiency issues, we ported the most promising procedure to C++ [48] (see Figure 8.2), which is a lower-level programming language that offers significant performance advantages. The shift to C++ not only optimized the code but also enabled the smoothly integration of the procedure into the company's software codebase, making the implementation competitive with their existing solutions. Moreover, C++ facilitated the exploitation of parallel computing, resulting in a significant further speed-up in execution times.

The implementation of the algorithms for ISPP was done in Rust [80]

Figure 8.2: Programming languages - Python, C++, Rust.

(see Figure 8.2), a modern systems programming language. Rust is known for its strong emphasis on safety and performance, making it an excellent choice for computationally intensive tasks. Rust was selected to naturally integrate with the `jagua-rs` library [35], which is also written in Rust. By leveraging Rust's safety features and memory management, we ensured a robust and efficient implementation capable of handling the complexities of the problem.

CPLEX

IBM ILOG CPLEX Optimization Studio [46], commonly referred to as CPLEX (see Figure 8.3), is a commercial optimization software package currently owned and developed by IBM. It provides a set of high-performance solvers for optimization problems, along with specialized tools: the CPLEX Optimizer for mathematical programming (supporting linear, integer, and mixed-integer programming problems), the CP Optimizer dedicated to constraint programming, the Optimization Programming Language (OPL) as a modeling language for mathematical optimization models, and an integrated IDE. Additionally, the CPLEX Optimizer can be accessed through various modeling systems (e.g., AMPL) and supports modeling interfaces for C++, C#, Java, and Python. To facilitate interaction with CPLEX through Python, IBM offers both a Python application programming interface (API) and an integration library called IBM Decision Optimization CPLEX Modeling for Python [45], commonly known as DOcplex.

In this work, we utilized CPLEX to solve the models in the model-based procedures, leveraging its powerful optimization capabilities. To interact with CPLEX in Python, we employed the DOcplex library, which provided an effective interface for implementing and solving the optimization models.

Integrated Development Environments

The development of both Python and Rust code was carried out using Visual Studio Code [67] (see Figure 8.4), a popular and versatile code editor.

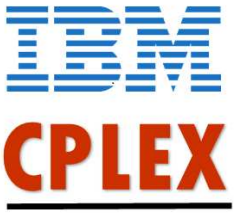


Figure 8.3: CPLEX.

Visual Studio Code is widely used by developers due to its lightweight nature, moreover it is easy to customize, and has many useful extensions and packages that enhance productivity. These features made it an ideal choice for our work, as it allowed for seamless integration with the necessary tools and libraries for both Python and Rust development.

The C++ implementation was developed using the Visual Studio 2022 environment [66] (see Figure 8.4), under a Professional license provided by the company. Visual Studio offers a comprehensive suite of tools that facilitate development and debugging processes. It supports efficient code navigation, an integrated debugger, and powerful profiling tools, which were particularly beneficial for optimizing the performance-critical parts of the implementation. Moreover, its built-in support for parallel computing further enhanced the development experience, making it a suitable choice for the high-performance requirements of the development of this project.

For computational analysis and experimentation, Jupyter Notebook [73] (see Figure 8.4) was used. Jupyter Notebook is an interactive environment that supports the creation and execution of code in a cell-based format. It is particularly useful for data analysis, visualization, and sharing results in a clear and organized way. This tool allowed us to run Python code, visualize the results, and document the process in a way that made it easier to explore and improve our algorithms.

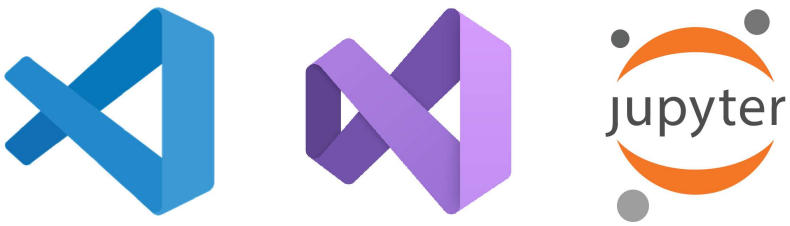


Figure 8.4: Development environments - Visual Studio Code, Visual Studio, Jupyter Notebook.

8.2 Definition of benchmarks for the 2DC&PP-PT

Given the practical focus of this thesis, we aim to test the proposed procedures for solving the 2DC&PP-PT on real-world benchmarks that reflect the production environments of an important sample of the reference company's clients. To define these benchmarks, we first collaborated with the company to understand the common features of the problem that appear in their clients' production processes. As detailed in Section 3.3, the key characteristics identified include the use of multiple sheet types, the presence of optional items, the need to account for punching margins, and precedence constraints among items. Based on the problem definition, we collected instances that exhibit these features directly from the company's database.

To create a structured and representative benchmark, we decided to randomly generate instances while ensuring that they adhered to the characteristics observed in real-world scenarios, as well as the guidelines provided by the company. As a result, we devised a benchmark consisting of 96 different classes of instances, with 20 instances per class, resulting in a total of 1920 instances (available at [23]). Each class represents one of the combinations of the following traits:

- number of sheet types: either 1 or 3. All sheets consist of iron plates with a thickness of 0.8mm, leading to a safety margin of 2.4mm (three times the thickness). The quantity of sheets was considered unlimited;
- number of compulsory items: 5, 10, 15, or 20;
- number of optional items: none, a few (one third of the number of compulsory items), or many (two thirds of the number of compulsory items) optional items are added to the compulsory set;
- items with punching margins: none, half, or all items, with punching margins that are greater or equal to the safety margin. A uniform punching margin was assigned to each side of the item;
- items with precedence: none or all. In the latter case, one precedence level was assigned to the items, and this precedence could be treated as either soft or hard.

The specific features of each class of instances are detailed in the table reported in Appendix B.

To maintain compatibility with the attributes handled by the outsourced optimization software, one combination of attributes was excluded, specifically the one in which both optional items and items with precedence constraints are considered, thus leading to 96 classes (instead of 144, as the set of all possible combinations of the previous traits would suggest). Although

our problem definition allows for both attributes to coexist, with optional items not involved in precedence relations, this combination was excluded for consistency with the company's procedure. In practice, this coexistence can be useful in scenarios where precedence relations are motivated by the order of production, such as, for example, due to item assembly after the cutting phase. In such cases, producing optional items to save material does not negatively impact production, instead, they are simply stored for future use. Our procedures are designed to handle both attributes simultaneously to accommodate these scenarios.

Furthermore, for comparison with the company's procedure, we considered that the level of precedence of the items (if any is assigned), could be treated as either soft or hard for all items. However, it is important to note that our procedures are capable of handling instances where some items have hard precedence constraints assigned and some other come with soft precedence preferences.

Finally, to maintain consistency with the simplifications adopted in this thesis, we assigned a uniform punching margin to the items, even though the implemented procedures are able to address different punching margins on each side of an item.

The results presented in the following sections adhere to these restrictions and simplifications. However, all algorithms were thoroughly tested for correctness in their more general implementations.

8.3 Results on 2DC&PP-PT

This section presents the results of the computational analysis conducted to evaluate the proposed procedures for solving the 2DC&PP-PT. We begin by examining the model-based procedures, with a focus on finding optimality bounds using the exact lexicographic approach. These results are then compared with both the matheuristic solutions and the solutions currently used by the company.

Next, we report on the heuristic approaches, which were developed to be fast methods that provide high-quality solutions in comparison to the company's current software.

In all the results presented here, we consider only soft precedence relations. Indeed, as mentioned in Section 8.2, the company's software does not allow a mix of hard and soft precedence. As a consequence, the presence of hard precedence would disable the possibility to evaluate performance in terms of the secondary objective of the problem, which is minimizing the violation of soft precedence relations. This would limit the extent of our analysis, thus we focus on soft precedence. However, the procedures were also tested for correctness when treating precedence levels as hard constraints. From this point on, when we refer to precedence, we mean soft

precedence, unless stated otherwise.

Computational results have been analyzed with respect to the instance size, by grouping the instances by the number of compulsory items (5, 10, 15, and 20, corresponding to Groups A, B, C, and D respectively). Average results for each analyzed method are also presented and identified by the term *Total* in the results' tables reported in this chapter. Table 8.1 provides a summary of the column definitions used in the following computational results tables. This summary serves as a reference throughout the chapter, allowing the reader to easily revisit the meaning of each metric as needed.

All the computational tests were performed on a machine with an Intel Core i7-10850H processor at 2.7GHz and 32GB of RAM.

Metric	**Description**
Err	Percent error relative to the best waste achieved by any of the compared procedures, computed on the amount of instances solved by the analyzed method.
Time	Running time in seconds.
F	Percentage of instances that the analyzed method fails to solve within the time limit.
W	Percentage of instances where the analyzed method provides worse results compared to the company's procedure.
S	Percentage of instances where the analyzed method provides the same results as the company's procedure.
B	Percentage of instances where the analyzed method provides better results compared to the company's procedure.
ΔP	Average precedence violation gap, comparing the company's procedure with the analyzed method on instances with precedence relations that achieve the same waste as the company's solution. Positive values indicate better handling of precedence by the analyzed method.
Total	Average results for the analyzed method.

Table 8.1: Definition of the metrics used in computational results tables. The *analyzed method* refers to one of the procedures developed in this thesis and subject to performance evaluation.

8.3.1 Model-based algorithms

The model-based approaches discussed in Section 6.1 consist of two main methods. The first is an exact method, the *lexicographic procedure* (Subsection 6.1.1), which provides optimal solutions. The second is an iterative matheuristic approach, referred to as the *iterative procedure* (Subsection 6.1.2). The analysis reported below follows the lines of the one presented in [24], related to a previous version of the instance benchmark.

Group	Company	Lexicographic				Iterative					
	Err	Time	F	S	B	Time	F	W	S	B	Err
A	10.0	0.7	0.0	78.3	21.7	0.6	0.0	0.0	79.6	20.4	0.4
B	25.8	107.3	10.4	62.9	26.7	51.7	3.8	3.3	64.6	28.3	8.2
C	19.9	364.0	54.2	32.1	13.7	279.7	39.6	2.5	40.4	17.5	12.7
D	9.7	470.1	71.2	21.3	7.5	410.3	62.1	0.0	30.0	7.9	9.8
Total	*16.3*	*235.5*	*34.0*	*48.6*	*17.4*	*185.6*	*26.4*	*1.4*	*53.7*	*18.5*	*7.8*

Table 8.2: Comparative results of lexicographic and iterative procedures with the company's solutions - instances grouped by size.

For the computational analysis, we focused on a subset of instances from the devised benchmark. We excluded instances with *many* optional items, as we aimed to investigate the impact of a smaller number of optional items on the solution quality and execution time without significantly increasing the instance size. Additionally, we excluded instances where *half* of the items had assigned punching margins. This allowed us to better assess the net effect of including or excluding punching margins by activating or deactivating the related constraints, particularly in terms of how they influence running times. We observe that, for some of the excluded classes, especially the ones with a high number of compulsory items, our model-based methods show excessively long execution times. As a result, the tests were conducted on 48 different classes, corresponding to a total of 960 instances.

The models were solved using CPLEX 12.8 as the MILP solver, when needed. All procedures were implemented in Python and executed with a time limit of 600 seconds. This time limit was chosen as a balance between the operational needs of the company and the computational time required by our methods to produce quality results.

The comparative results shown in Table 8.2 focus on waste minimization, which is the primary objective of the optimization. The table presents the average results for the company's procedure, the lexicographic procedure, and the iterative procedure. For the company's procedure, we report the percent error (*Err*) relative to the best waste achieved by any of the three methods. It is important to note that the company's procedure always runs until the time limit is reached, which serves as its stopping criterion, so running times are not included in the table. The company's procedure is designed to quickly find a solution and improve upon it, consistently providing a feasible solution. Additionally, for all the tested instances, no significant improvement was observed after the first few seconds, indicating that the percent error would remain the same even if the time limit were significantly reduced. For the iterative procedure, the table includes not only the percent error, but also the running time in seconds (*Time*). It also shows the percentage of instances where the procedure either fails to

solve all involved models within the time limit (F) or produces worse (W), the same (S), or better (B) results compared to the company's procedure. In this case, the reported percent error only includes instances where the iterative procedure does not fail. The lexicographic procedure has similar columns, except for W and Err, which are both zero by definition.

Compared to the company's procedure, the exact lexicographic method provides better results in 17.4% of the instances, on average. However, its longer execution time limits its ability to handle larger instances, and it fails to solve 71.2% of the instances in Group D within the time limit. In contrast, the iterative matheuristic has lower failure rates but at the cost of some error relative to the optimal or best-known waste. Despite this, the error remains limited (12.7% in the worst-performing group) and is consistently much lower than the error from the company's procedure. Furthermore, the iterative procedure outperforms the company's method in a comparable or even greater number of instances than the lexicographic procedure, as shown in the B column.

Table 8.3 analyzes the effect of having precedence relations among items by comparing average results between instances with and without this feature. An additional column, ΔP, shows the average precedence violation gap between the company's solutions and the ones obtained by the proposed model-based procedures. To properly reflect the secondary optimization objective, the gap is computed, for both procedures, on the related subset of instances that reach the same waste of the corresponding company's solution. A positive (resp., negative) gap identifies a better (resp., worse) handling of the precedence attributes by the lexicographic or the iterative method with respect to the current company's procedure. The metric ΔP is computed as follows:

$$\Delta P = \frac{1}{|I_{same}|} \sum_{i \in I_{same}} \left(P_i^{\text{Company}} - P_i^{\text{Method}} \right)$$

where I_{same} represents the set of instances with precedence relations where the analyzed method achieves the same waste as the company's solution. The term P_i^{Company} (resp., P_i^{Method}) is the precedence violation percentage of the company's procedure (resp., the analyzed method) for instance $i \in I$.

For instances that include precedence relations, the decomposition approach used in the iterative procedure significantly reduces both running times and failure rates, in instances with 10 or more items. However, this comes with a trade-off, as errors in waste minimization can reach up to 24.2% (compared to almost no error in instances without precedence). Despite this, the company's procedure is even more impacted by the presence of precedence relations in terms of waste minimization. As a result, the iterative matheuristic still manages to produce either the same (57.8% of cases) or better (26.6% of cases) results for most instances, with exceptions

	Group	Company	Lexicographic					Iterative						
		Err	Time	F	S	B	ΔP	Time	F	W	S	B	Err	ΔP
no precedence	A	8.0	0.7	0.0	80.6	19.4	–	0.6	0.0	0.0	81.2	18.8	0.3	–
	B	13.3	139.4	15.6	65.6	18.8	–	73.9	5.6	0.0	67.5	26.9	0.0	–
	C	2.7	394.4	60.0	31.9	8.1	–	379.2	56.9	0.0	34.4	8.7	0.0	–
	D	1.4	480.6	74.4	22.5	3.1	–	474.4	73.1	0.0	23.1	3.8	0.0	–
	Total	*6.4*	*253.8*	*37.5*	*50.1*	*12.4*	–	*232.0*	*33.9*	*0.0*	*51.5*	*14.6*	*0.1*	–
precedence	A	14.0	0.6	0.0	73.8	26.2	6.1	0.7	0.0	0.0	76.2	23.8	0.8	2.8
	B	50.9	43.3	0.0	57.5	42.5	11.5	7.4	0.0	10.0	58.8	31.2	23.7	3.2
	C	54.2	303.3	42.5	32.5	25.0	14.2	80.7	5.0	7.5	52.5	35.0	24.2	3.8
	D	26.2	449.2	65.0	18.8	16.2	15.8	282.1	40.0	0.0	43.8	16.2	18.6	5.9
	Total	*36.3*	*199.1*	*26.9*	*45.6*	*27.5*	*11.9*	*92.7*	*11.2*	*4.4*	*57.8*	*26.6*	*16.8*	*3.9*

Table 8.3: Comparative results of lexicographic and iterative procedures with the company's solutions - instances grouped by size, precedence.

being rare (4.4% on average). Furthermore, the iterative method is able to provide a solution within the time limit with higher frequency than the lexicographic one, especially on larger instances with precedence, reducing the failure rate from 26.9% to 11.2% in Group D. From the perspective of precedence violation rates, we can see that both procedures better handle the secondary objective, in fact we have positive precedence violation gaps for both model-based methods. Reported results suggest that the company's procedure has certain limitations in handling precedence relations. First, the procedure does not support a mix of soft and hard precedence relations, nor does it allow the inclusion of items with precedence constraints and optional items in the same instance (as discussed in Section 8.2). Furthermore, even when precedence constraints are handled, as in the considered benchmark, the procedure's performance is notably affected. Specifically, the margin of improvement is significant: in instances with precedence relations, the company's procedure exhibits an average error rate of 36.6%, compared to 16.8% achieved by the iterative procedure. Results confirm that the iterative procedure offers a solid trade-off: while its average performance in all instance groups is slightly worse than the lexicographic procedure, it consistently outperforms the company's method, making it a strong option in balancing both the waste minimization and the precedence satisfaction objectives.

Aggregating the instances by size and the number of sheet types, as in Table 8.4, reveals that handling multiple sheet types makes the problem significantly more challenging. The exact procedure's running times quickly become prohibitive, often failing on larger instances. In fact, with more than one sheet type, all instances of Group D failed to provide a solution within the time limit, against the 42.5% corresponding rate with only one sheet type. Moreover, only on smaller instances the procedure provides notable

	Group	Company	Lexicographic					Iterative						
		Err	Time	F	S	B	ΔP	Time	F	W	S	B	Err	ΔP
1 type	A	2.4	0.3	0.0	99.2	0.8	8.3	0.3	0.0	0.0	99.2	0.8	0.0	3.4
	B	11.5	4.6	0.0	85.8	14.2	12.8	3.9	0.0	0.0	89.2	10.8	4.5	2.8
	C	13.5	145.2	16.7	62.5	20.8	14.2	128.2	15.8	0.0	73.4	10.8	10.6	3.6
	D	11.5	340.2	42.5	42.5	15.0	15.8	247.4	31.7	0.0	60.0	8.3	10.9	5.9
	Total	*9.7*	*122.6*	*14.8*	*72.5*	*12.7*	*12.8*	*94.9*	*11.9*	*0.0*	*80.4*	*7.7*	*6.5*	*3.9*
3 types	A	17.6	1.1	0.0	57.5	42.5	2.1	1.0	0.0	0.0	60.0	40.0	0.9	1.9
	B	40.1	210.1	20.8	40.0	39.2	7.3	99.6	7.5	6.7	40.0	45.8	12.3	5.3
	C	26.2	582.9	91.7	1.6	6.7	–	431.2	63.3	5.0	7.5	24.2	17.4	5.9
	D	7.9	600.0	100.0	0.0	0.0	–	573.2	92.5	0.0	0.0	7.5	0.0	–
	Total	*23.0*	*348.5*	*53.1*	*24.8*	*22.1*	*4.7*	*276.2*	*40.8*	*2.9*	*26.9*	*29.4*	*7.6*	*4.4*

Table 8.4: Comparative results of lexicographic and iterative procedures with the company's solutions - instances grouped by size,sheet types.

improvements with more that one sheet type. This difficulty is reduced by the iterative procedure, which not only decreases running times and failure rates but also continues to offer considerable improvements compared to the company's procedure, especially for mid-sized instances. The impact of single versus multiple sheet types does not significantly affect the percentage of precedence violations for this procedure.

	Group	Company	Lexicographic					Iterative						
		Err	Time	F	S	B	ΔP	Time	F	W	S	B	Err	ΔP
no optional	A	9.3	0.6	0.0	79.4	20.6	6.1	0.6	0.0	0.0	80.6	19.4	0.4	2.8
	B	28.6	36.2	0.0	73.1	26.9	11.5	17.6	0.0	5.0	73.8	21.2	11.9	3.2
	C	27.4	310.3	44.3	41.9	13.8	14.2	192.9	24.4	3.8	53.0	18.8	15.2	3.8
	D	13.1	429.1	63.7	28.1	8.1	15.8	349.9	51.9	0.0	40.0	8.2	11.6	5.9
	Total	*19.6*	*194.0*	*27.0*	*55.6*	*17.3*	*11.9*	*140.2*	*19.1*	*2.2*	*61.9*	*16.9*	*9.8*	*3.9*
optional	A	11.4	0.9	0.0	76.2	23.8	–	0.7	0.0	0.0	77.5	22.5	0.5	–
	B	20.3	249.5	31.2	42.6	26.2	–	120.0	11.2	0.0	46.2	42.6	0.0	–
	C	4.7	471.5	73.8	12.4	13.8	–	453.2	70.0	0.0	15.0	15.0	0.0	–
	D	2.9	552.2	86.2	7.4	6.2	–	531.1	82.5	0.0	10.0	7.5	0.0	–
	Total	*9.8*	*318.5*	*47.8*	*34.7*	*17.5*	–	*276.2*	*40.9*	*0.0*	*37.2*	*21.9*	*0.1*	–

Table 8.5: Comparative results of lexicographic and iterative procedures with the company's solutions - instances grouped by size, optional items.

Optional items have a similar, though milder, effect, as can be seen in Table 8.5. Their presence leads to longer running times and higher failure rates, primarily due to the increased number of items within the same instance class. Nonetheless, both the lexicographic and iterative procedures outperform the company's procedure more markedly in terms of waste min-

imization, especially the iterative one that increases the average rate of getting better results than the company's ones from 16.9% to 21.9%. Notice that, due to the simplification presented in Section 8.2, in instances where optional items are involved, precedence relations are not present.

	Group	Company	Lexicographic					Iterative						
		Err	Time	F	S	B	ΔP	Time	F	W	S	B	Err	ΔP
no margins	A	11.7	0.7	0.0	75.0	25.0	3.2	0.6	0.0	0.0	75.8	24.2	0.1	2.7
	B	26.4	99.5	10.0	65.8	24.2	10.1	44.3	3.3	4.2	66.7	25.8	8.8	2.6
	C	21.0	356.0	53.4	35.8	10.8	14.1	276.0	39.2	2.5	43.3	15.0	13.3	2.3
	D	15.2	465.4	71.7	19.1	9.2	13.3	394.8	59.2	0.0	30.0	10.8	13.5	5.6
	Total	*18.6*	*230.4*	*33.8*	*49.0*	*17.3*	*10.2*	*178.9*	*25.4*	*1.7*	*54.0*	*19.0*	*8.9*	*3.3*
margins	A	8.3	0.7	0.0	81.7	18.3	9.1	0.6	0.0	0.0	83.3	16.7	0.8	2.9
	B	25.2	115.2	10.8	60.0	29.2	13.2	59.2	4.2	2.5	62.5	30.8	7.6	3.9
	C	18.8	372.1	55.0	28.3	16.7	14.4	283.4	40.0	2.5	37.5	20.0	12.1	5.5
	D	4.2	474.8	70.8	23.4	5.8	17.5	425.8	65.0	0.0	30.0	5.0	5.5	6.1
	Total	*14.1*	*240.7*	*34.2*	*48.3*	*17.5*	*13.5*	*192.2*	*27.3*	*1.2*	*53.3*	*18.1*	*6.5*	*4.6*

Table 8.6: Comparative results of lexicographic and iterative procedures with the company's solutions - instances grouped by size, items with punching margins.

Finally, the inclusion of items with required punching margins has a relatively minor impact on the performance of the proposed procedures, as reported in Table 8.6, showing only slight differences between instances with punching margins and those without.

As a general note, we have seen that the model-based procedures frequently fail to identify a feasible solution, even within a generous time limit of $600s$. This limit has been set for testing purposes, but it already exceeds the constraints typically imposed by the application domain, where acceptable time limits range from a few seconds to a couple of minutes. Potential improvements include enhancing the problem formulation by incorporating tightening inequalities and customized cutting planes, as well as providing the models with a warm-start solution, such as one generated by a heuristic algorithm. However, we observed large gaps at the root nodes, and we do not expect that such expedients lead to acceptable running times. We expect that, given the industrial nature of the application, the execution times of exact methods, while valuable for assessing the quality of other approaches, are incompatible with the strict production requirements, making them impractical for real-world use. Consequently, our efforts have focused on the development of heuristic methods tailored for practical implementation.

8.3.2 Heuristic algorithms

In this section, we present the results of the main heuristic approaches described in Section 6.3 and 6.4 to solve the 2DC&PP-PT, namely the *greedy algorithm* (Subsection 6.3.1), the *pilot method* (Subsection 6.3.2), and the *beam search with double search effort* (Subsection 6.4.3).

As discussed in Section 6.3, we focus on a simplified version of the problem where only one sheet type is considered. Accordingly, we conducted tests on half of the instances in the devised benchmark (Section 8.2), specifically those without multiple sheet types, which corresponds to 48 classes, resulting in a total of 960 instances.

In the model-based methods, we focused primarily on the quality of the solutions. However, with the heuristic methods, our goal was to still achieve high-quality solutions while working within a limited time budget. To this end, we set a time limit of $4s$ for the company's procedure, which, as we will detail in the following, ensures a fair comparison to our proposed heuristics.

The tables reporting on the computational analysis for the heuristic methods are structured in a similar way to the ones presented for the model-based procedures in the previous subsection, a summary of the considered metrics can be found in Table 8.1. Results focus on waste minimization, precedence violation rate, and execution times. For the company's procedure, the table reports the percent error (Err) relative to the best waste achieved by any of the methods considered in this subsection (i.e., greedy algorithm, pilot method, beam search with double search effort, and company's algorithm). Since the company's procedure runs up to the time limit, its running time is not explicitly presented in the table. In the case of the constructive methods, the tables include the execution time in seconds ($Time$). Additionally, the tables show the percentage of instances where the heuristic methods produced worse (W), the same (S), or better (B) results compared to the company's procedure. Furthermore, the tables report the average difference in precedence violation (ΔP) between the company's solutions and those obtained by the heuristic methods, which reflects performance with respect to the secondary objective of minimizing precedence violations. We recall that the precedence violation gap is computed on the instances with precedence relations that achieve the same material waste of the corresponding company's solution.

Constructive heuristics

In the following we present and discuss the computational results related to the greedy algorithm (Subsection 6.3.1) and the pilot method (Subsection 6.3.2), both implemented in Python. For both procedures, we use an area descending sorting heuristic for the items and evaluate their placement based on the amount of contact perimeter. This configuration is commonly used

in the literature, and has proven to be suitable for our context.

The results presented in Table 8.7 compare the greedy algorithm, the pilot method, and the company's solutions, and they highlight key differences in performance, especially regarding the trade-off between solution quality and computational speed. The greedy algorithm, stands out for its extremely fast execution times, consistently solving instances in a fraction of a second. However, this speed comes at a cost, as the solutions produced by the greedy algorithm are often of lower quality compared to both the pilot method and the company's software. This is reflected in its high error rates across all groups, with an average error of 31.0%. Additionally, the greedy algorithm shows a significant portion of instances (28.1%) where it provides worse solutions than the company's procedure, and only marginal improvements where it performs better (6.8% of cases). The precedence violation gap is small and often denotes a slight improvement with respect to the company software (except in class C), but this is secondary to the larger waste observed in its solutions.

In contrast, the pilot method strikes a better balance between speed and solution quality. While still very fast, with an average execution time of $0.25s$, it improves upon the greedy algorithm in terms of solution accuracy. The total average error drops to 19%, indicating a noticeable improvement compared to the corresponding 33% of the greedy algorithm. Moreover, the pilot method outperforms the company's procedure in 9.6% of cases, and provides the same solution quality in 72.2% of the instances. This method also demonstrates a more balanced performance across different instance groups, maintaining a competitive edge even as the problem size increases.

Overall, while the greedy algorithm is faster, it sacrifices solution quality, whereas the pilot method offers a more refined balance, still operating efficiently but producing better results than the greedy approach. However, both heuristics remain less effective than the company's solutions in minimizing waste, especially in larger instances.

Group	Company	Greedy Algorithm						Pilot Method					
	Err	Time	W	S	B	Err	ΔP	Time	W	S	B	Err	ΔP
A	3.4	0.00	9.6	89.2	1.2	18.8	0.4	0.02	1.7	96.6	1.7	4.0	0.7
B	9.7	0.01	26.2	69.2	4.6	35.9	0.5	0.11	16.3	75.4	8.3	19.6	0.6
C	13.6	0.01	33.3	55.4	11.3	36.5	-0.9	0.32	22.9	62.5	14.6	23.6	-0.8
D	16.8	0.02	43.3	46.7	10.0	40.6	0.6	0.55	32.1	54.2	13.8	28.8	0.2
Total	*10.9*	*0.01*	*28.1*	*65.1*	*6.8*	*33.0*	*0.2*	*0.25*	*18.2*	*72.2*	*9.6*	*19.0*	*0.2*

Table 8.7: Comparative results of greedy algorithm and pilot method with the company's solutions - instances grouped by size.

Beam search heuristic

We report the computational results related to the BS-DSE, presented in Subsection 6.4.3. This method was initially implemented in Python for prototyping and preliminary testing, allowing us to calibrate the algorithm's parameters. The sorting heuristic and placement evaluation criterion are consistent with those used for testing the greedy algorithm and the pilot method. Specifically, we adopt the area descending sorting heuristic and evaluate placements based on the contact perimeter.

Regarding the filter width, which determines how many branches of each node are evaluated in the beam selection, we follow the intuition that it should vary depending on the problem instance. The idea is to filter a number of nodes proportional to the number of branches. Inspired by Bennell et al. [13], we propose setting the filter width to the minimum between the number of items to be placed and five times the maximum number of allowed rotations. After testing various filter widths, results confirmed that Bennell et al.'s rule of thumb is the most suitable for our application.

As for the beam width, in the BS-DSE, it is no longer a parameter to be set, as it auto-regulates through the double search effort mechanism. Instead, the time limit parameter must be set. Given the promising preliminary results in Python, the algorithm was ported to C++ for a competitive comparison with the company's outsourced software. The following results refer to the latter implementation, where we imposed the same time limit of $4s$ as the one set for the company's software. Since the BS-DSE runs until the time limit is reached, execution times will not be reported.

Group	Company	BS-DSE				
	Err	W	S	B	Err	ΔP
A	3.4	0.4	96.7	2.9	0.3	0.9
B	9.7	6.7	76.6	16.7	2.5	-1.5
C	13.6	8.7	62.1	29.2	3.5	-4.6
D	16.8	12.1	55.0	32.9	3.8	-6.0
Total	*10.9*	*7.0*	*72.6*	*20.4*	*2.5*	*-2.8*

Table 8.8: Comparative results of BS-DSE algorithm with the company's solutions - instances grouped by size.

The results from the BS-DSE, shown in Table 8.8, demonstrate that this method is a highly competitive alternative to the company's software, particularly in terms of the primary objective of minimizing material waste. The beam search approach consistently delivers strong performance across all instance groups. It achieves better results in 20.4% of the cases, while matching the company's solution in an impressive 72.6% of instances. These results highlight the effectiveness of the beam search in producing high-quality solutions within a very reasonable time budget.

In terms of error rates, the beam search maintains a solid performance, with an average error of just 2.5%, thus outperforming the performance of the company's software, whose average error is almost 11%. This represents a significant improvement over the previously discussed heuristics and highlights its capability of handling even larger and more complex instances. The relative performance of BS-DSE, compared to the reference software, improves with instance size: for example, in Group D, which includes the larger instances, the beam search still outperforms the company's solution in 32.9% of cases, a notable achievement given the difficulty of this problem class.

However, there is still room for improvement regarding the secondary objective of minimizing precedence violations. While the primary objective is met with great success, the beam search exhibits a negative ΔP value overall, indicating that it slightly struggles with precedence relations compared to the company's procedure. This is an area where further enhancements could be beneficial, especially for instances that involve complex precedence constraints.

In analogy with the computational analysis conducted for the model-based methods, we have investigated the results with respect to the attributes involved in the tested instances, namely the presence of precedence relations, optional items, and items with punching margins.

	Group	Company	BS-DSE				
		Err	W	S	B	Err	ΔP
no precedence	A	3.0	0.6	96.1	3.3	0.3	–
	B	6.6	8.9	73.3	17.8	3.3	–
	C	9.7	11.7	58.9	29.4	4.7	–
	D	6.3	16.1	55.0	28.9	5.1	–
	Total	*6.4*	*9.3*	*70.8*	*19.8*	*3.4*	–
precedence	A	4.8	0.0	98.3	1.7	0.0	0.9
	B	19.1	0.0	86.7	13.3	0.0	-1.5
	C	25.5	0.0	71.7	28.3	0.0	-4.6
	D	48.1	0.0	55.0	45.0	0.0	-6.0
	Total	*24.4*	*0.0*	*77.9*	*22.1*	*0.0*	*-2.8*

Table 8.9: Comparative results of the BS-DSE algorithm with the company's solutions - instances grouped by size, precedence.

Table 8.9 presents comparative results focusing on instances grouped by the precedence attribute. In instances without precedence relations, the BS-DSE outperforms the company's method by producing low error rates (3.4%) and a high percentage of better results (19.8%). However, in cases with precedence constraints, the BS-DSE proves itself to be the best method among the tested heuristics, including the company's one, with 0% rate

	Group	Company	BS-DSE				
		Err	W	S	B	Err	ΔP
no optional	A	2.4	0.0	99.2	0.8	0.0	0.9
	B	11.9	0.0	92.5	7.5	0.0	-1.5
	C	12.7	1.7	84.1	14.2	2.9	-4.6
	D	24.1	3.3	74.2	22.5	4.0	-6.0
	Total	*12.8*	*1.2*	*87.5*	*11.2*	*1.7*	*-2.8*
optional	A	4.4	0.8	94.2	5.0	0.5	–
	B	7.5	13.3	60.9	25.8	4.9	–
	C	14.5	15.8	40.0	44.2	4.2	–
	D	9.5	20.8	35.9	43.3	3.6	–
	Total	*9.0*	*12.7*	*57.7*	*29.6*	*3.3*	–

Table 8.10: Comparative results of the BS-DSE algorithm with the company's solutions - instances grouped by size, optional items.

of error for the primary objective (waste minimization). This impressive performance comes at the cost, as previously observed, of a poorer handling of the secondary objective, showing negative rates for ΔP.

In Table 8.10, instances are grouped by the presence of optional items. When there are no optional items, the BS-DSE demonstrates solid performance, achieving a high percentage of same results as the company's method (87.5% on average), and shows good but limited improvement (11.2% better results). For instances with optional items, the BS-DSE exhibits again a strong performance, significantly outperforming the company's method, with 29.6% of instances showing better results, even if the number of instances where BS-DSE provides worse results increases with instance size. However, in all the classes, BS-DSE still provides same or better results than the company's software in a significant majority of instances (almost 80% in the worst case, the class D). The total error rate for the company is notably lower for instances with optional items (9%) with respect to instances without them (12.8%), however, it remains fairly above the error rate of the beam search approach (respectively 1.7% and 3.3%). Therefore, we can say that the BS-DSE algorithm handles optional items more effectively, leading to favorable outcomes. As regards the secondary objective, we recall that the simplification adopted in Section 8.2 excludes instances where both optional items and precedence relations are present. Column ΔP reports the same values in Table 8.8, 8.9 (with precedence), and 8.10 (with no optional items), thus, observations concerning precedence violations made for Table 8.8 holds true for instances without optional items.

Both for the company's procedure and the BS-DSE, the presence of punching margins favors the optimization of the first objective, as shown by the decreasing error rates presented in Table 8.11 for both the company's

	Group	Company	BS-DSE				
		Err	W	S	B	Err	ΔP
no margins	A	5.5	0.0	96.2	3.8	0.0	2.6
	B	7.5	7.6	76.2	16.2	3.8	0.9
	C	13.0	10.0	62.5	27.5	7.9	-6.5
	D	25.0	10.0	51.2	38.8	4.0	-7.2
	Total	*12.8*	*6.9*	*71.5*	*21.6*	*3.9*	*-2.6*
margins	A	2.3	0.6	96.9	2.5	0.4	0.0
	B	10.9	6.2	76.9	16.9	1.8	-2.8
	C	14.0	8.1	61.9	30.0	1.4	-3.5
	D	12.6	13.1	56.9	30.0	3.7	-5.5
	Total	*10.0*	*7.0*	*73.2*	*19.8*	*1.8*	*-3.0*

Table 8.11: Comparative results of the BS-DSE algorithm with the company's solutions - instances grouped by size, items with punching margins.

software and BS-DSE. In particular, BS-DSE improves the error rate from 3.9% to 1.8%, while obtaining better results than the company's software in a significant number of instances (19.8% on average), especially in larger instances. However, as expected, the company's method maintains a better handling in terms of precedence violations, as BS-DSE performs similarly with or without punching margins involved, showing negative values of ΔP in both cases for all instance classes, but the smallest ones.

In summary, the beam search is a highly promising heuristic that offers a strong balance between speed and solution quality, particularly excelling in material waste minimization. While it could benefit from improvements in handling precedence violations, its overall performance makes it a viable and competitive alternative to the company's software for addressing the 2DC&PP-PT investigated in this work.

8.4 Preliminary results for the ISPP

Preliminary results are presented for the solving procedures devised for the ISPP, as described in Chapter 7. These results are meant to assess the viability of a beam search approach based on the geometric framework `jagua-rs`, and they focus on analyzing key parameters, such as filter width, beam width, and time limit. Additionally, we compare the two versions of the proposed algorithms for ISPP (Subsection 7.3.3): one that includes global evaluation (IBSGE) and one that does not (IBS). Finally, we examine the effectiveness of the infeasibility redeeming strategy.

The tests were performed on a classical benchmark consisting of five instances (i.e., shapes0, shapes1, shapes2, shirts, and trousers) that were also used by Bennell et al. [13], who proposed a similar approach, and by

Elkeran [30], that presents, to the best of our knowledge, the related state-of-the-art results. These instances are available on the European Working Group on Cutting and Packing (ESICUP) website [31].

Single run beam search

We tested different filter widths, confirming the rule of thumb proposed by Bennell et al. in [13], which suggests, for each instance, to set the filter width to the minimum between the number of items to place and five times the maximum number of allowed rotations over the items.

To evaluate the impact of global evaluation in the beam search, we tested both IBS and IBSGE on the same set of instances with fixed filter width (as previously defined) and beam width (set to 2). Table 8.12 reports the material waste percentage of the final solution ($Waste$), to be minimized, and the corresponding computational time in seconds ($Time$). Results indicate that incorporating global evaluation significantly increases execution times, as expected. However, the improvements in solution quality are substantial, with IBSGE outperforming IBS in four out of five instances, and with no performance degradation observed in the other one. Therefore, while global evaluation proves to be beneficial, it comes at a high computational cost.

In addition to IBS and IBSGE, we also evaluated the performance of the greedy algorithm for the ISPP (i.e., the IGA presented in Subsection 7.3.2). This heuristic, which was employed to set a baseline for comparison, provides a feasible, but often less optimized, solution compared to the beam search methods. The IGA achieves competitive results in terms of efficiency, as its performance in terms of computational time is considerably better, although it lacks the enhanced solution quality provided by the beam search methods, which leverage more complex search strategies and evaluations.

Overall, the comparison highlights that while IGA provides a faster solution, the beam search methods, especially IBSGE, offer superior solution quality at the cost of increased computational time.

Instance	IGA		IBS		IBSGE	
	Waste	Time	Waste	Time	Waste	Time
shapes0	44.52	0.7	40.56	47	38.65	598
shapes1	41.16	0.7	38.70	87	35.75	674
shapes2	36.70	0.4	29.17	40	26.76	317
shirts	17.67	2.3	17.67	889	17.67	9421
trousers	17.68	1.5	17.68	326	13.48	3069

Table 8.12: Comparative results for the three proposed procedures IGA, IBS, IBSGE, where IBS and IBSGE share the same filter width and beam width.

Double search effort mechanism

The execution time of the beam search is proportional to the beam width. Given that our goal is to develop an algorithm suitable for industrial applications, running times are a critical factor. To address this, we introduced a time limit option in both procedures, following the approach suggested by Araya et al. [3]. As discussed in Subsection 7.3.4, this approach leads to algorithms IBS-DSE and IBSGE-DSE, and it involves solving the instance iteratively with increasing beam widths until the time limit is reached, returning the best solution found.

Instance	Time limit 1200s		Time limit 300s	
	IBS-DSE	IBSGE-DSE	IBS-DSE	IBSGE-DSE
shapes0	41.37	39.57	41.50	42.18
shapes1	36.74	35.23	36.74	41.16
shapes2	25.27	23.63	27.28	24.57
shirts	17.67	17.67	17.67	17.67
trousers	17.06	17.68	17.65	17.68

Table 8.13: Waste percentage results for IBS-DSE and IBSGE-DSE.

We applied this method to the benchmark with a time limit of 1200s (as proposed in [30]). The material waste rates reported in Table 8.13 suggest that with a generous time limit, such as 1200s, global evaluation enhances the algorithm, leading to better solutions compared to IBS-DSE. However, with a shorter time limit of 300s, IBSGE-DSE inhibits the algorithm's ability to expand the beam width, as more time is allocated to computing the global evaluation. In contrast, IBS-DSE allows for greater beam width expansion, which compensates for the lack of global evaluation and improves diversification within the shorter time budget.

Instance	IBS-DSE	IBSGE-DSE	Elkeran [30]	Bennell et al. [13]	
	Waste	Waste	Waste	Waste	Time
shapes0	41.37	39.57	31.21	35.65	1119
shapes1	36.74	35.23	23.27	28.75	1410
shapes2	25.27	23.63	15.16	18.71	20784
shirts	17.67	17.67	11.04	10.31	32616
trousers	17.06	17.68	9.00	9.62	28631

Table 8.14: Comparative results for the IBS-DSE and IBSGE-DSE with results from Elkeran [30] and Bennell et al. [13], where IBS-DSE, IBSGE-DSE, and Elkeran's procedure are subject to a time limit of 1200s.

Table 8.14 presents the results of the IBS-DSE and IBSGE-DSE methods with a time limit of 1200s (as shown in Table 8.13), and compares them with results reported by Elkeran [30] and Bennell et al. [13]. The referenced

studies express results in terms of material utilization percentage, which is calculated as the ratio of the area of placed items to the total area of the final strip. To facilitate comparison with our procedures, we present the complementary percentage of material waste.

It is important to note that execution times are not shown for our procedures as well as for Elkeran's results, as these methods run up to the specified time limit of $1200s$. While our preliminary results demonstrate a reasonable performance, they do not yet match the benchmarks established in the literature. This disparity indicates a significant opportunity for further refinement and improvement of our algorithms. Future work will focus on enhancing the methods to achieve more competitive results and better align with the high standards set by previous studies.

Infeasibility redeeming strategy

We performed tests to assess the impact of the new placement heuristic IPH we propose in Subsection 7.3.1 as integrated in Section 7.4 to allow for starting from infeasible sampled positions. In particular, we conducted a specific analysis of the sampling process involved in constructing solutions using the IGA procedure, which incorporates the IPH and the redeeming strategy. By profiling 100 samples per item, we recorded the number of infeasible samples and tracked those that were subsequently redeemed. On average, 58.6% of the samples were found infeasible, highlighting significant potential for redeeming infeasibility. Additionally, we recorded that, on average, 21.8% of the infeasible samples were successfully redeemed. These results demonstrate the effectiveness of the proposed procedure in redeeming infeasibility, potentially improving the overall solution process by recovering better placement positions. We remark that not only our beam search procedures, but also different constructive or search heuristics, may benefit from the redeeming procedure.

Overall, the preliminary experiments provided insightful feedback on the procedures devised to solve the ISPP. In particular, we were able to evaluate the integration of global evaluation in the beam search scheme, which resulted in a positive impact on solution quality. Additionally, we measured the rates of redeemed infeasible samples, indicating that the proposed strategy for handling infeasibility holds promise as an effective addition to the search-based algorithms for ISPP.

Chapter 9

Conclusions

This thesis has addressed a real-world optimization problem arising in the sheet metal industry, namely the 2DC&PP-PT, specifically related to the generation of cutting layouts for punching-based cutting machines produced by Salvagnini Italia S.p.A., the reference company of this work. The complexity of the problem lies not only in its theoretical challenges as a rectangular cutting and packing problem, but also in the practical attributes that arise from the application context, related to the technologies used by the reference company. In addition to classical objective function, aimed at waste minimization, and geometrical constraints, such as ensuring that items do not overlap, the problem also incorporates further attributes, among which the ones related to multiple sheet types, optional cutting items, necessary margins between items to preserve quality and prevent material deformation, limitations in the placement of the items within the sheets, different production priorities leading to hard and soft precedence constraints. These attributes added a layer of complexity that distinguishes this work from other formulations commonly found in the literature. As such, this thesis presents, to the best of our knowledge, an original problem that has not been previously addressed by scientific literature.

The ultimate goal of our research was to develop a competitive alternative to the current solution employed by the reference company to solve the 2DC&PP-PT, which is provided by heuristics embedded in a third-party commercial software. To achieve this, we devised several solution approaches, including exact methods, heuristics, and a matheuristic, each designed to address different aspects of the study.

Both the exact approach and the matheuristic are based on the MILP formulations that we devised to model the problem. In addition to integrating several modeling components taken from literature, the thesis extends the formulation to take all the new attributes into account, in particular the ones related to a secondary objective concerning soft preferences, and to the definition of different cut margins. The devised exact proce-

dure utilizes a lexicographic method to handle the two objectives, while the matheuristic employs a dedicated iterative process to solve the models. The exact lexicographic procedure has been useful to provide bounds to assess the quality of the solutions output by the company's software, at least for small- and medium-size instances, showing potential for improvements in heuristic solution algorithms. The matheuristic decomposes the problem by soft-precedence priority and iterates the solution of smaller-size and more constrained models where all precedence are considered as hard, gaining in computational efficiency and extending the set of results to compare to the company's software ones.

For the heuristic methods, we propose algorithms of varying complexity and nature, including a fast constructive greedy algorithm, a related variant that incorporates the pilot method, and algorithms based on beam search. Among the proposed methods, the beam search approach has proven to be the most effective. Beam search-based algorithms not only provided high-quality solutions, but also performed with the speed necessary for practical industrial use. The balance between execution time and solution quality makes it competitive with the company's existing software, and thus a valuable alternative for future integration. In order to achieve these results, the thesis develops a new definition of extreme point, that extends the one known in the literature for classical packing problems to the new attributes of 2DC&PP-PT, especially the ones that involve the different types of required margins between items in a cutting layout. Moreover, the right setting of the different beam search component has been achieved by combining and adapting the algorithmic components taken from literature.

The other faster methods explored in this thesis to solve the 2DC&PP-PT, though not as competitive in all scenarios, were useful in understanding the nature and the challenges of the problem. These approaches provide deeper insights into specific features of the problem and can be effective in certain cases, especially when dealing with specific machine configurations or requirements. This reinforces the intuition that no single solution fits all instances of the problem, and multiple methods can coexist, each tailored to different scenarios.

A significant part of this research also involved investigating the irregular variant of the problem, where items to be cut are no longer regular in shape. To address this problem, we extended the beam search approach to the ISPP, while integrating `jagua-rs`, a recently proposed geometric framework, to handle the irregularity of the items. Although this part of the research is still in its early stages, the initial results are promising and demonstrate high industrial interest. The irregular variant introduces new challenges, especially in handling the geometric complexity of irregular shapes, but it also offers substantial potential for further exploration. In our preliminary study, in part conducted in collaboration with the authors of `jagua-rs`, we exploited the features of the framework to explore a new placement heuristic

that recover the infeasibility of sampled positions, which may be beneficial to beam search as well as other constructive or improvement heuristics for ISPP. The implemented beam search variant, once fully developed and tested on real-world benchmarks, could provide additional flexibility and efficiency for the company's production processes.

Looking forward, several future research directions arise from this work. First, the beam search algorithm for the 2DC&PP-PT, while already performing well, can be further enhanced. In particular, future work will focus on refining the implementation of the algorithm in a more efficient way, aiming at further reducing the running time required by computationally intensive tasks. Additionally, further tests and refinements will be conducted on the components of the algorithm, especially on those attributes that were simplified during the computational experiments.

As regards the extension to the irregular variant, future work will focus on testing the proposed algorithm for the ISPP on real-world benchmarks and refining it with more realistic and complex attributes of practical relevance. Once sufficiently developed, this method can be integrated into industrial applications, providing both flexibility and efficiency in handling irregular shapes.

In conclusion, this thesis has made several contributions to the field of cutting and packing problems, particularly in addressing the challenges posed by real-world constraints and advancing solution methods that are competitive with existing industrial practices. The beam search algorithm stands out as a robust and efficient tool for solving the rectangular problem, while the exploration of the irregular variant opens new perspectives for research and industrial application. With the outlined future improvements and further exploration, the work presented here has the potential to further significantly contribute to both the academic literature and the practical aspects of industrial cutting processes.

Appendix A

A complete MILP model for 2DC&PP-PT

In Section 5.3, we have presented a MILP formulation of a simplified version of 2DC&PP-PT, where every item has a uniform punching margin assigned to each side. This allowed us to simplify the notation and focus on the principles of the proposed modeling approach. In Section 5.5, we also discussed on how the formulation can be extended to comply with all the features of 2DC&PP-PT as defined in Section 3.3. In this appendix, we provide the complete model for 2DC&PP-PT by adopting the notation presented in Section 5.2 and following the extension outlined in Section 5.5.

$$\min \sum_{s \in S} \left[\sum_{t \in T} (\Omega_t H_t) g_{st} - \sum_{i \in I} (\omega_i \eta_i) f_{is} \right]$$

s.t.

$$l_{ij} + l_{ji} + b_{ij} + b_{ji} + (1 - f_{is}) + (1 - f_{js}) \geq 1 \qquad \forall i, j \in I, s \in S$$

$$x_i + (1 - (r_i^{90} + r_i^{270}))\omega_i + (r_i^{90} + r_i^{270})\eta_i + m_{ij}^L \leq x_j + \overline{\Omega}(1 - l_{ij}) \qquad \forall i, j \in I$$

$$x_i + (1 - (r_i^{90} + r_i^{270}))\omega_i + (r_i^{90} + r_i^{270})\eta_i + m_{ij}^L \geq x_j - \overline{\Omega}(1 - l_{ij}) \qquad \forall i, j \in I$$

$$y_i + (1 - (r_i^{90} + r_i^{270}))\eta_i + (r_i^{90} + r_i^{270})\omega_i + m_{ij}^B \leq y_j + \overline{H}(1 - b_{ij}) \qquad \forall i, j \in I$$

$$y_i + (1 - (r_i^{90} + r_i^{270}))\eta_i + (r_i^{90} + r_i^{270})\omega_i + m_{ij}^B \geq y_j - \overline{H}(1 - b_{ij}) \qquad \forall i, j \in I$$

$$c_{ij}^L \leq 3 - l_{ij} - r_i^0 - r_j^0 \qquad \forall i, j \in I : \mu_i^R + \mu_j^L > 0$$

$$c_{ij}^B \leq 3 - b_{ij} - r_i^0 - r_j^0 \qquad \forall i, j \in I : \mu_i^T + \mu_j^B > 0$$

$$c_{ij}^L \leq 3 - l_{ij} - r_i^0 - r_j^{90} \qquad \forall i, j \in I : \mu_i^R + \mu_j^T > 0$$

$$c_{ij}^B \leq 3 - b_{ij} - r_i^0 - r_j^{90} \qquad \forall i, j \in I : \mu_i^T + \mu_j^L > 0$$

$$c_{ij}^{L} \leq 3 - l_{ij} - r_i^0 - r_j^{180} \qquad \forall i,j \in I : \mu_i^R + \mu_j^R > 0$$

$$c_{ij}^{B} \leq 3 - b_{ij} - r_i^0 - r_j^{180} \qquad \forall i,j \in I : \mu_i^T + \mu_j^T > 0$$

$$c_{ij}^{L} \leq 3 - l_{ij} - r_i^0 - r_j^{270} \qquad \forall i,j \in I : \mu_i^R + \mu_j^B > 0$$

$$c_{ij}^{B} \leq 3 - b_{ij} - r_i^0 - r_j^{270} \qquad \forall i,j \in I : \mu_i^T + \mu_j^R > 0$$

$$c_{ij}^{L} \leq 3 - l_{ij} - r_i^{90} - r_j^0 \qquad \forall i,j \in I : \mu_i^B + \mu_j^L > 0$$

$$c_{ij}^{B} \leq 3 - b_{ij} - r_i^{90} - r_j^0 \qquad \forall i,j \in I : \mu_i^R + \mu_j^B > 0$$

$$c_{ij}^{L} \leq 3 - l_{ij} - r_i^{90} - r_j^{90} \qquad \forall i,j \in I : \mu_i^B + \mu_j^T > 0$$

$$c_{ij}^{B} \leq 3 - b_{ij} - r_i^{90} - r_j^{90} \qquad \forall i,j \in I : \mu_i^R + \mu_j^L > 0$$

$$c_{ij}^{L} \leq 3 - l_{ij} - r_i^{90} - r_j^{180} \qquad \forall i,j \in I : \mu_i^B + \mu_j^R > 0$$

$$c_{ij}^{B} \leq 3 - b_{ij} - r_i^{90} - r_j^{180} \qquad \forall i,j \in I : \mu_i^R + \mu_j^T > 0$$

$$c_{ij}^{L} \leq 3 - l_{ij} - r_i^{90} - r_j^{270} \qquad \forall i,j \in I : \mu_i^B + \mu_j^B > 0$$

$$c_{ij}^{B} \leq 3 - b_{ij} - r_i^{90} - r_j^{270} \qquad \forall i,j \in I : \mu_i^R + \mu_j^R > 0$$

$$c_{ij}^{L} \leq 3 - l_{ij} - r_i^{180} - r_j^0 \qquad \forall i,j \in I : \mu_i^L + \mu_j^L > 0$$

$$c_{ij}^{B} \leq 3 - b_{ij} - r_i^{180} - r_j^0 \qquad \forall i,j \in I : \mu_i^B + \mu_j^B > 0$$

$$c_{ij}^{L} \leq 3 - l_{ij} - r_i^{180} - r_j^{90} \qquad \forall i,j \in I : \mu_i^L + \mu_j^T > 0$$

$$c_{ij}^{B} \leq 3 - b_{ij} - r_i^{180} - r_j^{90} \qquad \forall i,j \in I : \mu_i^B + \mu_j^L > 0$$

$$c_{ij}^{L} \leq 3 - l_{ij} - r_i^{180} - r_j^{180} \qquad \forall i,j \in I : \mu_i^L + \mu_j^R > 0$$

$$c_{ij}^{B} \leq 3 - b_{ij} - r_i^{180} - r_j^{180} \qquad \forall i,j \in I : \mu_i^B + \mu_j^T > 0$$

$$c_{ij}^{L} \leq 3 - l_{ij} - r_i^{180} - r_j^{270} \qquad \forall i,j \in I : \mu_i^L + \mu_j^B > 0$$

$$c_{ij}^{B} \leq 3 - b_{ij} - r_i^{180} - r_j^{270} \qquad \forall i,j \in I : \mu_i^B + \mu_j^R > 0$$

$$c_{ij}^{L} \leq 3 - l_{ij} - r_i^{270} - r_j^{0} \qquad \forall i,j \in I : \mu_i^T + \mu_j^L > 0$$

$$c_{ij}^{B} \leq 3 - b_{ij} - r_i^{270} - r_j^{0} \qquad \forall i,j \in I : \mu_i^L + \mu_j^B > 0$$

$$c_{ij}^{L} \leq 3 - l_{ij} - r_i^{270} - r_j^{90} \qquad \forall i,j \in I : \mu_i^T + \mu_j^T > 0$$

$$c_{ij}^{B} \leq 3 - b_{ij} - r_i^{270} - r_j^{90} \qquad \forall i,j \in I : \mu_i^L + \mu_j^L > 0$$

$$c_{ij}^{L} \leq 3 - l_{ij} - r_i^{270} - r_j^{180} \qquad \forall i,j \in I : \mu_i^T + \mu_j^R > 0$$

$$c_{ij}^{B} \leq 3 - b_{ij} - r_i^{270} - r_j^{180} \qquad \forall i,j \in I : \mu_i^L + \mu_j^T > 0$$

$$c_{ij}^{L} \leq 3 - l_{ij} - r_i^{270} - r_j^{270} \qquad \forall i,j \in I : \mu_i^T + \mu_j^B > 0$$

$$c_{ij}^{B} \leq 3 - b_{ij} - r_i^{270} - r_j^{270} \qquad \forall i,j \in I : \mu_i^L + \mu_j^R > 0$$

$$m_{ij}^{L} \geq max\{\mu_i^R, \mu_j^L, \tau_t\}(1 - c_{ij}^{L}) - \overline{\Omega}\,(2 - a_{ijs} - g_{st}) - \overline{\Omega}(r_i^0 + r_j^0) \qquad \forall i,j \in I,\ s \in S, t \in T$$

$$m_{ij}^{B} \geq max\{\mu_i^T, \mu_j^B, \tau_t\}(1 - c_{ij}^{B}) - \overline{H}\,(2 - a_{ijs} - g_{st}) - \overline{H}(r_i^0 + r_j^0) \qquad \forall i,j \in I,\ s \in S, t \in T$$

$$m_{ij}^{L} \geq max\{\mu_i^R, \mu_j^T, \tau_t\}(1 - c_{ij}^{L}) - \overline{\Omega}\,(2 - a_{ijs} - g_{st}) - \overline{\Omega}(r_i^0 + r_j^{90}) \qquad \forall i,j \in I,\ s \in S, t \in T$$

$$m_{ij}^{B} \geq max\{\mu_i^T, \mu_j^L, \tau_t\}(1 - c_{ij}^{B}) - \overline{H}\,(2 - a_{ijs} - g_{st}) - \overline{H}(r_i^0 + r_j^{90}) \qquad \forall i,j \in I,\ s \in S, t \in T$$

$$m_{ij}^{L} \geq max\{\mu_i^R, \mu_j^R, \tau_t\}(1 - c_{ij}^{L}) - \overline{\Omega}\,(2 - a_{ijs} - g_{st}) - \overline{\Omega}(r_i^0 + r_j^{180}) \qquad \forall i,j \in I,\ s \in S, t \in T$$

$$m_{ij}^{B} \geq max\{\mu_i^T, \mu_j^T, \tau_t\}(1 - c_{ij}^{B}) - \overline{H}\,(2 - a_{ijs} - g_{st}) - \overline{H}(r_i^0 + r_j^{180}) \qquad \forall i,j \in I,\ s \in S, t \in T$$

$$m_{ij}^{L} \geq max\{\mu_i^R, \mu_j^B, \tau_t\}(1 - c_{ij}^{L}) - \overline{\Omega}\,(2 - a_{ijs} - g_{st}) - \overline{\Omega}(r_i^0 + r_j^{270}) \qquad \forall i,j \in I,\ s \in S, t \in T$$

$$m_{ij}^{B} \geq max\{\mu_i^T, \mu_j^R, \tau_t\}(1 - c_{ij}^{B}) - \overline{H}\,(2 - a_{ijs} - g_{st}) - \overline{H}(r_i^0 + r_j^{270}) \qquad \forall i,j \in I,\ s \in S, t \in T$$

$$m_{ij}^{L} \geq max\{\mu_i^B, \mu_j^L, \tau_t\}(1 - c_{ij}^L) - \overline{\Omega}\,(2 - a_{ijs} - g_{st}) - \overline{\Omega}(r_i^{90} + r_j^0) \qquad \forall i,j \in I,\ s \in S, t \in T$$

$$m_{ij}^{B} \geq max\{\mu_i^R, \mu_j^B, \tau_t\}(1 - c_{ij}^B) - \overline{H}\,(2 - a_{ijs} - g_{st}) - \overline{H}(r_i^{90} + r_j^0) \qquad \forall i,j \in I,\ s \in S, t \in T$$

$$m_{ij}^{L} \geq max\{\mu_i^B, \mu_j^T, \tau_t\}(1 - c_{ij}^L) - \overline{\Omega}\,(2 - a_{ijs} - g_{st}) - \overline{\Omega}(r_i^{90} + r_j^{90}) \qquad \forall i,j \in I,\ s \in S, t \in T$$

$$m_{ij}^{B} \geq max\{\mu_i^R, \mu_j^L, \tau_t\}(1 - c_{ij}^B) - \overline{H}\,(2 - a_{ijs} - g_{st}) - \overline{H}(r_i^{90} + r_j^{90}) \qquad \forall i,j \in I,\ s \in S, t \in T$$

$$m_{ij}^{L} \geq max\{\mu_i^B, \mu_j^R, \tau_t\}(1 - c_{ij}^L) - \overline{\Omega}\,(2 - a_{ijs} - g_{st}) - \overline{\Omega}(r_i^{90} + r_j^{180}) \qquad \forall i,j \in I,\ s \in S, t \in T$$

$$m_{ij}^{B} \geq max\{\mu_i^R, \mu_j^T, \tau_t\}(1 - c_{ij}^B) - \overline{H}\,(2 - a_{ijs} - g_{st}) - \overline{H}(r_i^{90} + r_j^{180}) \qquad \forall i,j \in I,\ s \in S, t \in T$$

$$m_{ij}^{L} \geq max\{\mu_i^B, \mu_j^B, \tau_t\}(1 - c_{ij}^L) - \overline{\Omega}\,(2 - a_{ijs} - g_{st}) - \overline{\Omega}(r_i^{90} + r_j^{270}) \qquad \forall i,j \in I,\ s \in S, t \in T$$

$$m_{ij}^{B} \geq max\{\mu_i^R, \mu_j^R, \tau_t\}(1 - c_{ij}^B) - \overline{H}\,(2 - a_{ijs} - g_{st}) - \overline{H}(r_i^{90} + r_j^{270}) \qquad \forall i,j \in I,\ s \in S, t \in T$$

$$m_{ij}^{L} \geq max\{\mu_i^L, \mu_j^L, \tau_t\}(1 - c_{ij}^L) - \overline{\Omega}\,(2 - a_{ijs} - g_{st}) - \overline{\Omega}(r_i^{180} + r_j^0) \qquad \forall i,j \in I,\ s \in S, t \in T$$

$$m_{ij}^{B} \geq max\{\mu_i^B, \mu_j^B, \tau_t\}(1 - c_{ij}^B) - \overline{H}\,(2 - a_{ijs} - g_{st}) - \overline{H}(r_i^{180} + r_j^0) \qquad \forall i,j \in I,\ s \in S, t \in T$$

$$m_{ij}^{L} \geq max\{\mu_i^L, \mu_j^T, \tau_t\}(1 - c_{ij}^L) - \overline{\Omega}\,(2 - a_{ijs} - g_{st}) - \overline{\Omega}(r_i^{180} + r_j^{90}) \qquad \forall i,j \in I,\ s \in S, t \in T$$

$$m_{ij}^{B} \geq max\{\mu_i^B, \mu_j^L, \tau_t\}(1 - c_{ij}^B) - \overline{H}\,(2 - a_{ijs} - g_{st}) - \overline{H}(r_i^{180} + r_j^{90}) \qquad \forall i,j \in I,\ s \in S, t \in T$$

$$m_{ij}^{L} \geq max\{\mu_i^L, \mu_j^R, \tau_t\}(1 - c_{ij}^L) - \overline{\Omega}\,(2 - a_{ijs} - g_{st}) - \overline{\Omega}(r_i^{180} + r_j^{180}) \qquad \forall i,j \in I,\ s \in S, t \in T$$

$$m_{ij}^{B} \geq max\{\mu_i^B, \mu_j^T, \tau_t\}(1 - c_{ij}^B) - \overline{H}\,(2 - a_{ijs} - g_{st}) - \overline{H}(r_i^{180} + r_j^{180}) \qquad \forall i,j \in I,\ s \in S, t \in T$$

$$m_{ij}^{L} \geq max\{\mu_i^L, \mu_j^B, \tau_t\}(1 - c_{ij}^L) - \overline{\Omega}\,(2 - a_{ijs} - g_{st}) - \overline{\Omega}(r_i^{180} + r_j^{270}) \qquad \forall i,j \in I,\ s \in S, t \in T$$

$$m_{ij}^{B} \geq max\{\mu_i^B, \mu_j^R, \tau_t\}(1 - c_{ij}^B) - \overline{H}\,(2 - a_{ijs} - g_{st}) - \overline{H}(r_i^{180} + r_j^{270}) \qquad \forall i,j \in I,\ s \in S, t \in T$$

$$m_{ij}^{L} \geq max\{\mu_i^T, \mu_j^L, \tau_t\}(1 - c_{ij}^L) - \overline{\Omega}\,(2 - a_{ijs} - g_{st}) - \overline{\Omega}(r_i^{270} + r_j^0) \qquad \forall i,j \in I,\ s \in S, t \in T$$

$$m_{ij}^{B} \geq max\{\mu_i^L, \mu_j^B, \tau_t\}(1 - c_{ij}^B) - \overline{H}\,(2 - a_{ijs} - g_{st}) - \overline{H}(r_i^{270} + r_j^0) \qquad \forall i,j \in I,\ s \in S, t \in T$$

$$m_{ij}^{L} \geq max\{\mu_i^T, \mu_j^T, \tau_t\}(1 - c_{ij}^L) - \overline{\Omega}\,(2 - a_{ijs} - g_{st}) - \overline{\Omega}(r_i^{270} + r_j^{90}) \qquad \forall i,j \in I,\ s \in S, t \in T$$

$$m_{ij}^B \geq max\{\mu_i^L, \mu_j^L, \tau_t\}(1 - c_{ij}^B) - \overline{H}\,(2 - a_{ijs} - g_{st}) - \overline{H}(r_i^{270} + r_j^{90}) \quad \forall i,j \in I,\ s \in S, t \in T$$

$$m_{ij}^L \geq max\{\mu_i^T, \mu_j^R, \tau_t\}(1 - c_{ij}^L) - \overline{\Omega}\,(2 - a_{ijs} - g_{st}) - \overline{\Omega}(r_i^{270} + r_j^{180}) \quad \forall i,j \in I,\ s \in S, t \in T$$

$$m_{ij}^B \geq max\{\mu_i^L, \mu_j^T, \tau_t\}(1 - c_{ij}^B) - \overline{H}\,(2 - a_{ijs} - g_{st}) - \overline{H}(r_i^{270} + r_j^{180}) \quad \forall i,j \in I,\ s \in S, t \in T$$

$$m_{ij}^L \geq max\{\mu_i^T, \mu_j^B, \tau_t\}(1 - c_{ij}^L) - \overline{\Omega}\,(2 - a_{ijs} - g_{st}) - \overline{\Omega}(r_i^{270} + r_j^{270}) \quad \forall i,j \in I,\ s \in S, t \in T$$

$$m_{ij}^B \geq max\{\mu_i^L, \mu_j^R, \tau_t\}(1 - c_{ij}^B) - \overline{H}\,(2 - a_{ijs} - g_{st}) - \overline{H}(r_i^{270} + r_j^{270}) \quad \forall i,j \in I,\ s \in S, t \in T$$

$$m_{ij}^L \leq \overline{\Omega}(1 - c_{ij}^L) \quad \forall i,j \in I$$

$$m_{ij}^B \leq \overline{H}(1 - c_{ij}^B) \quad \forall i,j \in I$$

$$a_{ijs} \geq f_{is} + f_{js} - 1 \quad \forall i,j \in I, s \in S$$

$$\sum_{t \in T} g_{st} \leq 1 \quad \forall s \in S$$

$$\sum_{s \in S} g_{st} \leq \beta_t \quad \forall t \in T$$

$$\sum_{s \in S} f_{is} = 1 \quad \forall i \in I_c$$

$$\sum_{s \in S} f_{is} \leq 1 \quad \forall i \in I_o$$

$$\sum_{i \in I} f_{is} \leq n \sum_{t \in T} g_{st} \quad \forall s \in S$$

$$f_{js} \leq \sum_{r=1}^{s} f_{ir} \quad \forall s \in S,\ (i,j) \in P$$

$$x_i + (1 - (r_i^{90} + r_i^{270}))\omega_i + (r_i^{90} + r_i^{270})\eta_i \leq \sum_{t \in T} (\Omega_t - \phi_i) g_{st} + \overline{\Omega}(1 - f_{is}) \quad \forall i \in I, s \in S$$

$$y_i + (1 - (r_i^{90} + r_i^{270}))\eta_i + (r_i^{90} + r_i^{270})\omega_i \leq \sum_{t \in T} (H_t - \phi_i) g_{st} + \overline{H}(1 - f_{is}) \quad \forall i \in I, s \in S$$

$$x_i \geq \phi_i - \overline{\Omega}(1 - f_{is}) \quad \forall i \in I, s \in S$$

$$y_i \geq \phi_i - \overline{H}(1 - f_{is}) \quad \forall i \in I, s \in S$$

$$x_i + (1 - (r_i^{90} + r_i^{270}))\omega_i + (r_i^{90} + r_i^{270})\eta_i \leq \epsilon_i + \overline{\Omega}(1 - f_{is}) \quad \forall i \in I, s \in S$$

$$y_i + (1 - (r_i^{90} + r_i^{270}))\eta_i + (r_i^{90} + r_i^{270})\omega_i \leq \epsilon_i + \overline{H}(1 - f_{is}) \quad \forall i \in I, s \in S$$

$$x_i \geq \sum_{t \in T} (\Omega_t - \epsilon_i) g_{st} - \overline{\Omega}(1 - f_{is}) \quad \forall i \in I, s \in S$$

$$y_i \geq \sum_{t \in T} (H_t - \epsilon_i) g_{st} - \overline{H}(1 - f_{is}) \quad \forall i \in I, s \in S$$

$$r_i^\theta \leq \sigma_i^\theta \quad \forall i \in I, \theta \in \Theta$$

$$\sum_{\theta \in \Theta} r_i^\theta = 1 \quad \forall i \in I$$

$x_i,\ y_i \geq 0$	$\forall i \in I$
$l_{ij},\ b_{ij},\ c^L_{ij},\ c^B_{ij} \in \{0,1\}$	$\forall i,j \in I$
$r^\theta_i \in \{0,1\}$	$\forall i \in I, \theta \in \Theta$
$m^L_{ij},\ m^B_{ij} \geq 0$	$\forall i,j \in I$
$f_{is} \in \{0,1\}$	$\forall i \in I, s \in S$
$g_{st} \in \{0,1\}$	$\forall s \in S, t \in T$
$a_{ijs} \in \{0,1\}$	$\forall i,j \in I,$ $s \in S$

Appendix B

Features of the benchmark defined in Section 8.2

This appendix provides a detailed table outlining the characteristics of the 96 classes of instances that form the benchmark defined in Section 8.2 (publicly available at [23]). Each class corresponds to a unique combination of the traits described in the main text, and the table serves as a reference for the specific configurations used in the benchmark.

Table B.1: Characteristics of benchmark classes.

Class	Number of compulsory items	Number of optional items	Number of sheet types	Number of items with margin	Number of items with precedence
0	5	0	1	0	0
1	5	0	1	0	5
2	5	0	1	2	0
3	5	0	1	2	5
4	5	0	1	5	0
5	5	0	1	5	5
6	5	1	1	0	0
7	5	1	1	2	0
8	5	1	1	5	0
9	5	2	1	0	0
10	5	2	1	2	0
11	5	2	1	5	0
12	10	0	1	0	0
13	10	0	1	0	10
14	10	0	1	5	0
15	10	0	1	5	10

Table B.1: Characteristics of benchmark classes (continued).

Class	Number of compulsory items	Number of optional items	Number of sheet types	Number of items with margin	Number of items with precedence
16	10	0	1	10	0
17	10	0	1	10	10
18	10	3	1	0	0
19	10	3	1	5	0
20	10	3	1	10	0
21	10	6	1	0	0
22	10	6	1	5	0
23	10	6	1	10	0
24	15	0	1	0	0
25	15	0	1	0	15
26	15	0	1	7	0
27	15	0	1	7	15
28	15	0	1	15	0
29	15	0	1	15	15
30	15	5	1	0	0
31	15	5	1	7	0
32	15	5	1	15	0
33	15	10	1	0	0
34	15	10	1	7	0
35	15	10	1	15	0
36	20	0	1	0	0
37	20	0	1	0	20
38	20	0	1	10	0
39	20	0	1	10	20
40	20	0	1	20	0
41	20	0	1	20	20
42	20	6	1	0	0
43	20	6	1	10	0
44	20	6	1	20	0
45	20	12	1	0	0
46	20	12	1	10	0
47	20	12	1	20	0
48	5	0	3	0	0
49	5	0	3	0	5
50	5	0	3	2	0
51	5	0	3	2	5
52	5	0	3	5	0
53	5	0	3	5	5

Table B.1: Characteristics of benchmark classes (continued).

Class	Number of compulsory items	Number of optional items	Number of sheet types	Number of items with margin	Number of items with precedence
54	5	1	3	0	0
55	5	1	3	2	0
56	5	1	3	5	0
57	5	2	3	0	0
58	5	2	3	2	0
59	5	2	3	5	0
60	10	0	3	0	0
61	10	0	3	0	10
62	10	0	3	5	0
63	10	0	3	5	10
64	10	0	3	10	0
65	10	0	3	10	10
66	10	3	3	0	0
67	10	3	3	5	0
68	10	3	3	10	0
69	10	6	3	0	0
70	10	6	3	5	0
71	10	6	3	10	0
72	15	0	3	0	0
73	15	0	3	0	15
74	15	0	3	7	0
75	15	0	3	7	15
76	15	0	3	15	0
77	15	0	3	15	15
78	15	5	3	0	0
79	15	5	3	7	0
80	15	5	3	15	0
81	15	10	3	0	0
82	15	10	3	7	0
83	15	10	3	15	0
84	20	0	3	0	0
85	20	0	3	0	20
86	20	0	3	10	0
87	20	0	3	10	20
88	20	0	3	20	0
89	20	0	3	20	20
90	20	6	3	0	0
91	20	6	3	10	0

Table B.1: Characteristics of benchmark classes (continued).

Class	Number of compulsory items	Number of optional items	Number of sheet types	Number of items with margin	Number of items with precedence
92	20	6	3	20	0
93	20	12	3	0	0
94	20	12	3	10	0
95	20	12	3	20	0

References

[1] T. Achterberg. Scip: solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, 2009.

[2] H. Akeb, M. Hifi, and R. M'Hallah. A beam search algorithm for the circular packing problem. *Computers & Operations Research*, 36(5):1513–1528, 2009.

[3] I. Araya and M.-C. Riff. A beam search approach to the container loading problem. *Computers & Operations Research*, 43:100–107, 2014.

[4] V.P.R. Arruda, L.G.B. Mirisola, and N.Y. Soma. Rectangle packing with a recursive pilot method. *Computers & Operations Research*, 161:106447, 2024.

[5] R.C. Art. An approach to the two-dimensional irregular cutting stock problem. *Technical report 36-Y08, IBM Cambridge Science Centre*, 1966.

[6] Atlassian. SourceTree. `https://www.sourcetreeapp.com/`.

[7] M.M. Baldi, T.G. Crainic, G. Perboli, and R. Tadei. The generalized bin packing problem. *Transportation Research Part E: Logistics and Transportation Review*, 48(6):1205–1220, 2012.

[8] M.M. Baldi, T.G. Crainic, G. Perboli, and R. Tadei. Branch-and-price and beam search algorithms for the variable cost and size bin packing problem with optional items. *Annals of Operations Research*, 222:125–141, 2014.

[9] I. Baybars. A survey of exact algorithms for the simple assembly line balancing problem. *Management Science*, 32(8):909–932, 1986.

[10] J. Bennell and J.F. Oliveira. A beam search approach to solve the convex irregular bin packing problem with guillotine cuts. *European Journal of Operational Research*, 270(1):89–102, 2018.

[11] J.A. Bennell and J.F. Oliveira. The geometry of nesting problems: A tutorial. *European Journal of Operational Research*, 184(2):397–415, 2008.

[12] J.A. Bennell and J.F. Oliveira. A tutorial in irregular shape packing problems. *Journal of the Operational Research Society*, 60:S93–S105, 2009.

[13] J.A. Bennell and X. Song. A beam search implementation for the irregular shape packing problem. *Journal of Heuristics*, 16:167–188, 2010.

[14] M.A. Boschetti and A. Mingozzi. The two-dimensional finite bin packing problem. Part I: New lower bounds for the oriented case. *Quarterly Journal of the Belgian, French and Italian Operations Research Societies (4OR)*, 1:27–42, 2003.

[15] M.A. Boschetti and A. Mingozzi. The two-dimensional finite bin packing problem. Part II: New lower and upper bounds. *Quarterly Journal of the Belgian, French and Italian Operations Research Societies (4OR)*, 1:135–147, 2003.

[16] S.P. Bradley, A.C. Hax, and T.L. Magnanti. *Applied Mathematical Programming*. Addison-Wesley, 1977.

[17] C. Charalambous and K. Fleszar. A constructive bin-oriented heuristic for the two-dimensional bin packing problem with guillotine cuts. *Computers & Operations Research*, 38(10):1443–1451, 2011.

[18] L.H. Cherri, L.R. Mundim, M. Andretta, F.M. Toledo, J.F. Oliveira, and M.A. Carravilla. Robust mixed-integer linear programming models for the irregular strip packing problem. *European Journal of Operational Research*, 253:570–583, 2016.

[19] W. Ciscal-Terry, M. Dell'Amico, and M. Iori. Bin packing problem with general precedence constraints. *IFAC-PapersOnLine*, 48(3):2027–2029, 2015. 15th IFAC Symposium on Information Control Problems in Manufacturing.

[20] E.G. Coffman, M.R. Garey, and D.S. Johnson. *Approximation Algorithms for Bin-Packing – An Updated Survey*, pages 49–106. Springer, Vienna, 1984.

[21] M. Conforti, G. Cornuéjols, and G. Zambelli. *Integer Programming*. Springer, Cham, 2014.

[22] T.G. Crainic, G. Perboli, and R. Tadei. Extreme point-based heuristics for three-dimensional bin packing. *INFORMS Journal on Computing*, 20:368–384, 2008.

[23] L. De Giovanni, N. Gastaldon, and C. Turbian. Instances of a 2D Bin Packing Problem in the Sheet Metal Industry. `https://github.com/ChiaraTurbian/Instances2DBPP.git`, 2024.

[24] L. De Giovanni, N. Gastaldon, and C. Turbian. An integer programming approach for a 2D bin packing problem with precedence constraints in the sheet metal industry. In *Optimization in Green Sustainability and Ecological Transition*, pages 131–145. Springer Nature Switzerland, 2024.

[25] M. Dell'Amico, J.C. Díaz Díaz, and M. Iori. The bin packing problem with precedence constraints. *Operations Research*, 60(6):1491–1504, 2012.

[26] G. Dósa and J. Sgall. First Fit bin packing: A tight analysis. In *30th International Symposium on Theoretical Aspects of Computer Science (STACS 2013)*, volume 20 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 538–549. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2013.

[27] G. Dósa and J. Sgall. Optimal analysis of best fit bin packing. In *Automata, Languages, and Programming*, pages 429–441. Springer Berlin, Heidelberg, 2014.

[28] C. Duin and S. Voß. The pilot method: A strategy for heuristic repetition with application to the steiner problem in graphs. *Networks*, 34(3):181–191, 1999.

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

[30] A. Elkeran. A new approach for sheet nesting problem using guided cuckoo search and pairwise clustering. *European Journal of Operational Research*, 231(3):757–769, 2013.

[31] EURO Special Interest Group on Cutting and Packing. ESICUP Data Sets. `https://www.euro-online.org/websites/esicup/data-sets/`.

[32] L. Faina. A survey on the cutting and packing problems. *Bollettino dell'Unione Matematica Italiana*, 13:567–572, 2020.

[33] FICO. FICO Xpress Optimization. `https://www.fico.com/en/products/fico-xpress-optimization`.

[34] M. Fischetti and M. Fischetti. *Matheuristics*, pages 1–33. Springer International Publishing, Cham, 2018.

[35] J. Gardeyn. jagua-rs. `https://github.com/JeroenGar/jagua-rs`.

[36] J. Gardeyn, G. Vanden Berghe, and T. Wauters. Decoupling geometry from optimization: an open-source collision detection engine for 2D irregular cutting and packing problems. 2024. 20th ESICUP Meeting, Guimarães.

[37] J. Gardeyn and T. Wauters. A new guided local search heuristic and fast-fail collision-detection system for 2D irregular cutting and packing problems. 2023. 19th ESICUP Meeting, Bologna.

[38] M.R. Garey, R.L. Graham, D.S. Johnson, and A.C. Yao. Resource constrained scheduling as generalized bin packing. *Journal of Combinatorial Theory, Series A*, 21(3):257–298, 1976.

[39] Git. `https://git-scm.com/`.

[40] R. Gupta, S.K. Bose, S. Sundarrajan, M. Chebiyam, and A. Chakrabarti. A two stage heuristic algorithm for solving the server consolidation problem with item-item and bin-item incompatibility constraints. In *2008 IEEE International Conference on Services Computing*, volume 2, pages 39–46. IEEE, 2008.

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

[42] K. Hamdi-Dhaoui, N. Labadie, and A. Yalaoui. Algorithms for the two dimensional bin packing problem with partial conflicts. *RAIRO - Operations Research*, 46(1):41–62, 2012.

[43] J. Heistermann and T. Lengauer. The nesting problem in the leather manufacturing industry. *Annals of Operations Research*, 57:147–173, 1995.

[44] E. Hopper and B.C.H. Turton. A review of the application of meta-heuristic algorithms to 2D strip packing problems. *Artificial Intelligence Review*, 16:257–300, 2001.

[45] IBM. IBM Decision Optimization CPLEX Modeling for Python (DOcplex). `https://ibmdecisionoptimization.github.io/docplex-doc/`.

[46] IBM. IBM ILOG CPLEX Optimizer. `https://www.ibm.com/products/ilog-cplex-optimization-studio`.

[47] T. Imamichi, M. Yagiura, and H. Nagamochi. An iterated local search algorithm based on nonlinear programming for the irregular strip packing problem. *Discrete Optimization*, 6:345–361, 2009.

[48] International Organization for Standardization (ISO). C++. `https://isocpp.org/`.

[49] 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.

[50] JSON library. `https://docs.python.org/3/library/json.html`.

[51] J. Jylänki. A thousand ways to pack the bin - a practical approach to two-dimensional rectangle bin packing. `http://clb.demon.fi/files/RectangleBinPack.pdf`, 2010. Technical report.

[52] M. Konopasek. Mathematical treatments of some apparel marking and cutting problems. *U.S. Department of Commerce, Washington DC*, 1981.

[53] B. Korte and J. Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer, Berlin, Heidelberg, 2018.

[54] A.A.S. Leao, F.M.B. Toledo, J.F. Oliveira, and M.A. Carravilla. A semi-continuous mip model for the irregular strip packing problem. *International Journal of Production Research*, 54:712–721, 2016.

[55] A.A.S. Leao, F.M.B. Toledo, J.F. Oliveira, M.A. Carravilla, and R. Alvarez-Valdés. Irregular packing problems: A review of mathematical models. *European Journal of Operational Research*, 282(3):803–822, 2020.

[56] L.S. Lee. A genetic algorithm for two-dimensional bin packing problem. *MathDigest. Research Bulletin of Institute for Mathematical Research*, 2:34–39, 2008.

[57] 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, 2002.

[58] A. Lodi, S. Martello, and M. Monaci. Two-dimensional packing problems: A survey. *European Journal of Operational Research*, 141(2):241–252, 2002.

[59] A. Lodi, S. Martello, M. Monaci, and D. Vigo. *Two-Dimensional Bin Packing Problems*, volume 2, pages 107–129. Wiley Blackwell, Hoboken, 2014.

[60] A. Lodi, S. Martello, and D. Vigo. Recent advances on two-dimensional bin packing problems. *Discrete Applied Mathematics*, 123(1):379–396, 2002.

[61] A. Lodi, M. Monaci, and E. Pietrobuoni. Partial enumeration algorithms for two-dimensional bin packing problem with guillotine constraints. *Discrete Applied Mathematics*, 217:40–47, 2017.

[62] I. Loshchilov, M. Schoenauer, and M. Sebag. Adaptive coordinate descent. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, GECCO '11, pages 885–892. Association for Computing Machinery, 2011.

[63] V. Maniezzo, M.A. Boschetti, and T. Stützle. *Matheuristics: Algorithms and Implementations*. Springer, Cham, 2021.

[64] Matplotlib Library. `https://matplotlib.org/stable/`.

[65] S. Mezghani, B. Haddar, and H. Chabchoub. The evolution of rectangular bin packing problem - a review of research topics, applications, and cited papers. *Journal of Industrial and Management Optimization*, 19(5):3329–3361, 2023.

[66] Microsoft Corporation. Visual Studio 2022. `https://visualstudio.microsoft.com/vs/`.

[67] Microsoft Corporation. Visual Studio Code. `https://code.visualstudio.com/`.

[68] A.E.F. Muritiba, M. Iori, E. Malaguti, and P. Toth. Algorithms for the bin packing problem with conflicts. *INFORMS Journal on Computing*, 22(3):401–415, 2009.

[69] J.F. Oliveira, A.M. Gomes, and J.S. Ferreira. TOPOS - A new constructive algorithm for nesting problems. *OR Spektrum*, 22:263–284, 2000.

[70] O. Oliveira, D. Gamboa, and E. Silva. An introduction to the two-dimensional rectangular cutting and packing problem. *International Transactions in Operational Research*, 30(6):3238–3266, 2023.

[71] Pandas Library. `https://pandas.pydata.org/`.

[72] D. Pisinger and M. Sigurd. The two-dimensional bin packing problem with variable bin sizes and costs. *Discrete Optimization*, 2(2):154–167, 2005.

[73] Project Jupyter. Jupyter Notebook. `https://jupyter.org/`.

[74] Python. `https://www.python.org/`.

[75] A. Ramesh Babu and N. Ramesh Babu. A generic approach for nesting of 2-d parts in 2-d sheets using genetic and heuristic algorithms. *Computer-Aided Design*, 33:879–891, 2001.

[76] re library. https://docs.python.org/3/library/re.html.

[77] M.J. Rentmeesters, W.K. Tsai, and K. Lin. A theory of lexicographic multi-criteria optimization. In *Proceedings of ICECCS '96: 2nd IEEE International Conference on Engineering of Complex Computer Systems (held jointly with 6th CSESAW and 4th IEEE RTAW)*, pages 76–79. IEEE, 1996.

[78] P. Rocha, R. Rodrigues, A.M. Gomes, F.M.B. Toledo, and M. Andretta. Circle covering representation for nesting problems with continuous rotations. *IFAC Proceedings Volumes*, 47(3):5235–5240, 2014. 19th IFAC World Congress.

[79] M.O. Rodrigues and F.M.B. Toledo. A clique covering mip model for the irregular strip packing problem. *Computers & Operations Research*, 87:221–234, 2017.

[80] Rust Programming Language. https://www.rust-lang.org/.

[81] Salvagnini Group. https://www.salvagninigroup.com/en-INT.

[82] Salvagnini punching machine - the punching process. https://youtu.be/7vLhy3BwfIM?si=UCYEUHKqrlLysc2x.

[83] S.A. Segenreich and L.M.P. Faria Braga. Optimal nesting of general plane figures: a monte carlo heuristical approach. *Computers & Graphics*, 10:229–237, 1986.

[84] E.G. Talbi. *Metaheuristics: From Design to Implementation*. John Wiley & Sons, 2009.

[85] H. Tang, X. Li, S. Guo, S. Liu, L. Li, and L. Huang. An optimizing model to solve the nesting problem of rectangle pieces based on genetic algorithm. *Journal of Intelligent Manufacturing*, 28(8):1817–1826, 2017.

[86] F.M.B. Toledo, M.A. Carravilla, C. Ribeiro, J.F. Oliveira, and A.M. Gomes. The dotted-board model: A new mip model for the nesting problem. *International Journal of Production Economics*, 145:478–487, 2013.

[87] J. M. Valério de Carvalho. LP models for bin packing and cutting stock problems. *European Journal of Operational Research*, 141(2):253–273, 2002.

[88] G. Wäscher, H. Haußner, and H. Schumann. An improved typology of cutting and packing problems. *European Journal of Operational Research*, 183(3):1109–1130, 2007.

[89] W. Wu, C. Fan, J. Huang, Z. Liu, and J. Yan. Machine learning for the multi-dimensional bin packing problem: Literature review and empirical evaluation. *arXiv:2312.08103*, 2023.

[90] X. Yin. A beam search approach based on action space for the 2D rectangular packing problem. In *Computational Intelligence and Intelligent Systems (ISICA 2017)*, volume 874, pages 165–174. Springer Singapore, 2018.