Optimizing 2D Cutting: A Bin Packing Approach to Minimize Scraps and Maximize their Reusability

Manuel Chastenay ✉ (ORCID)
Lab-Usine - Joint Research Unit for Advanced Manufacturing, Université Laval, Québec, Canada
SOKÏO Industrie, Saint-Augustin-de-Desmaures, Canada

Xavier Zwingmann ✉ (ORCID)
Lab-Usine - Joint Research Unit for Advanced Manufacturing, Université Laval, Québec, Canada
SOKÏO Industrie, Saint-Augustin-de-Desmaures, Canada

Claude-Guy Quimper ✉ (ORCID)
Lab-Usine - Joint Research Unit for Advanced Manufacturing, Université Laval, Québec, Canada

Jonathan Gaudreault ✉ (ORCID)
Lab-Usine - Joint Research Unit for Advanced Manufacturing, Université Laval, Québec, Canada

─── **Abstract** ───

In industrial settings, cutting predefined pieces from one or multiple sheets of material is a common optimization challenge. This problem can be formulated as a variant of the 2D bin packing problem, where the edges of the pieces define the cut lines. This paper presents a constraint programming model developed in collaboration with an industrial partner in construction to minimize scrap waste generated when cutting insulation pieces. The model introduces an objective function designed to maximize the reusability of leftover material. To fully leverage the model's efficiency, an initial process transforms irregular insulation pieces into rectangles using one of four processing methods. A comparative analysis is conducted to evaluate the impact of these methods, as well as to benchmark the model's results against the partner's manual approach.

1 Introduction

Efficient material utilization is a critical concern in modern manufacturing. Minimizing waste directly translates into environmental benefits and helps industries reduce costs. In the construction industry, cutting insulation pieces from material sheets presents a complex but common challenge. At its core, this problem can be seen as a variant of the two-dimensional bin packing problem, in which the goal is to pack irregular shapes onto sheets while minimizing the unused area in said sheets. Unlike classical packing problems where no cutting is done, the unused area here results in waste. Thus, the insulation nesting task further demands that scraps not only be minimized but also have a shape that favors reusability for cutting future pieces. For example, a single rectangular scrap is preferable to many long and thin pieces, even if their combined area is the same.

In practice, insulation pieces are not always rectangles. Instead, they are often made up of multiple different shapes predefined in a way that fits the factory that produces them. Moreover, the pieces can be rotated or flipped to fit better on the sheet, which increases

the combinatorial complexity of the problem. This increased complexity, coupled with the heterogeneous nature of the available sheets (new sheets and recycled scraps), makes the already NP-Hard 2D bin packing problem [14, 15] even more complex and requires the use of sophisticated heuristics and exact methods.

This work presents two novel constraint models that integrate one distinct objective, each. First, the total area of the sheets that are used to nest the pieces is minimized. This reduces the number of sheets used and prioritizes the use of scraps over new sheets. Second, using the sheets found using the first model, the placement of all pieces is optimized to enhance the reusability of any resultant scrap. To manage the inherent computational difficulty, irregular insulation pieces are transformed into rectangles by a preprocessing phase that occurs before the optimization phase. The preprocessing uses one of the four new algorithms that we introduce. These methods aim to reduce the search space of the optimization process, thus improving computational efficiency.

The proposed approach is compared with the manual strategies traditionally used by our collaborating industrial partner. The analysis highlights improvements in material utilization and scrap reusability, which offer a significant step toward automated and sustainable production practices that can be applied to many industries.

2   Preliminary concepts

2.1   Cutting and Packing Problems

Cutting and packing problems [6] are part of a category of problems that have been studied well before computers. The simplicity of these problems, paired with the fact that they arise naturally in a lot of domains, makes them good candidates for research in optimization. Since Gilmore and Gomory [16] laid the foundation of modern computational approaches to the cutting stock problem, the field grew much, and many subproblems were officially classified. Wäscher et al. [27] proposed the (currently) most popular of these classifications, splitting cutting and packing problems into two distinct categories. The first category, **output maximization**, considers problems in which items have to be packed in a single container of fixed size, maximizing the number of items packed (or their value). The second, **input minimization**, considers problems where all pieces must be packed in one or many containers, minimizing the number of containers used. This category contains problems such as the open dimension problem (ODP), also known as the strip packing problem (SPP) [18]. Schutt et al. proposed a great example of the use of the SPP for industrial purposes in [24]. The SPP considers a single container of infinite length, which differentiates it from the cutting stock problem (CSP) [8, 17] and the bin packing problem (BPP) [1, 3, 9, 18, 22]. These two problems consider the use of one or many containers of fixed sizes. Although both subproblems manage the use of homogeneous and heterogeneous containers, their main difference comes from the items to pack. Where the CSP uses predefined cutting patterns to determine how to cut the different pieces, the BPP uses no pattern and instead works with highly heterogeneous items as input. Our problem is a specific implementation of the BPP for which we present a model that acts on a problem that is not well documented in the literature: maximizing the potential reusability of scraps. While minimizing scraps is common, maximizing their reusability is not and justifies the implementation of a new model.

2.2 Constraint Programming

Constraint programming (CP) is a programming paradigm specialized in solving satisfaction and optimization problems, such as those presented in Section 2.1 and in [25]. These problems are defined as mathematical models, where the decision variables are linked together with constraints that restrict the possible assignments to the solution space. Many predefined constraints already exist in popular solvers, often under different names. This disparity (and the fact that solvers are often implemented in different programming languages) led, in 2007, to the creation of MiniZinc [21], which is a CP modeling language that can seamlessly translate the defined problem to work with multiple solvers, such as Chuffed [10] and CP-SAT [23]. The presented MiniZinc models incorporate two important constraints: DIFFN and CUMULATIVE.

The DIFFN constraint [4, 26] considers a set of two-dimensional rectangles and constrains their relative placements so that they do not overlap. The constraint is formally expressed as: $\text{DIFFN}([X_1, \ldots, X_n], [Y_1, \ldots, Y_n], [width_1, \ldots, width_n], [height_1, \ldots, height_n])$, where X_i and Y_i represent the positions of the rectangles and $height_i$ and $width_i$ their dimensions. This constraint is especially useful for cutting and packing problems.

The CUMULATIVE constraint [2, 26] is used in scheduling problems to limit the number of tasks that can be executed simultaneously, but it can also be used efficiently in cutting and packing problems. Since it encodes the placement of tasks on a timeline, it can also encode the position of pieces over one dimension. Formally, it is expressed as: $\text{CUMULATIVE}([s_1, \ldots, s_n], [d_1, \ldots, d_n], [r_1, \ldots, r_n], B)$, where s_i represents the position of a piece, d_i its length, and r_i its height. B represents the maximum height allowed. It ensures that at any position along the dimension, the total height of the overlapping pieces does not exceed B.

Adding the CUMULATIVE to the DIFFN is considered redundant. In our case, this redundancy improves the solver convergence speed by pruning invalid placements early and more efficiently than by using only DIFFN [20].

3 Case study

SOKÏO is a wood manufacturer that is developing a highly customizable construction system. Its products are buildings whose components, such as walls, roofs, and floors, are made from cross-laminated timber (CLT) and are assembled directly in the factory in a fully automated way. For example, walls are made from layering the main CLT panel, insulation, doors, windows, and cladding. After being layered, all panels are brought to the site to be assembled, making it possible for the installation to be completed in one or two days at most.

Built with automation in mind, the different systems fueling the factory's production are seamlessly linked together and can communicate in a digital twin (DT) [12] developed in the Unity® game engine. Clients and architects eager to design their own dream building can freely do so in the DT, while the back-end of the app built using constraint programming ensures that the building fully respects all the different constraints of the factory.

The system in question in this paper concerns the nesting of insulation pieces. The shapes and dimensions of these pieces are calculated in real time during the product configuration process [28], and sent to the nesting module. As explained in the next section, this system computes an optimal way to cut the different input pieces into predefined sheets while optimizing key metrics, such as scrap reusability. This process saves a lot of time for SOKÏO, which used to do it by hand. It also saves critical quantities of insulation material by facilitating its reusability and reduces operations and labor costs.

(a) Specifications

(b) Examples
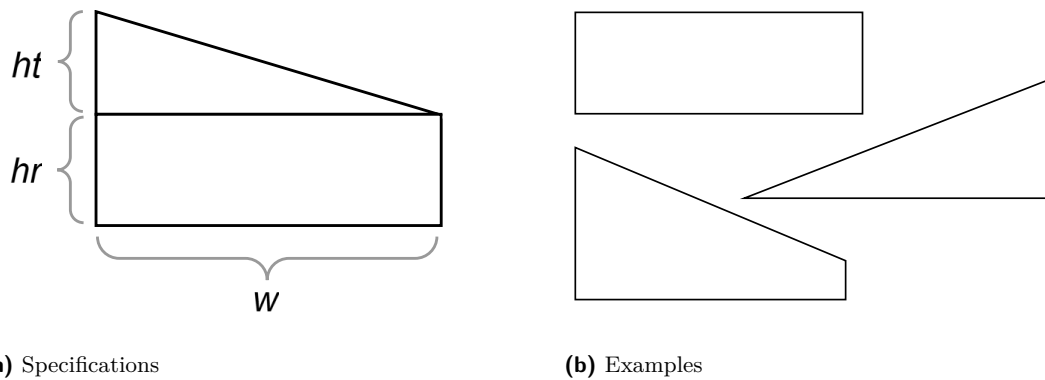
Figure 1 Insulation pieces

4   Problem presentation

To provide context for the proposed algorithms, it is important to understand the basic 2D bin packing problem [18, 22]. This problem involves two components: **i)** a set of rectangular items, and **ii)** a set of rectangular bins in which we need to fit all the items. The bins typically have the same predefined dimensions. The items also have their dimensions set in advance, with the added assumption that they can fit in the bins. These items can rotate 0° (without rotation) or 90°. Using these definitions, the goal is to find the smallest number of bins in which it is possible to fit all the items.

4.1   Insulation nesting

When applying these concepts to the insulation nesting problem, the bins are represented by insulation sheets, and the precomputed insulation pieces are the items. All insulation sheets come in a standard size of 8 feet by 4 feet, and the precision of the tools used to cut by SOKÏO is $\frac{1}{16}$ of an inch. All pieces are cut individually from the sheets, without common or guillotine cuts constraints [22]. We now add to the complexity of this problem by refining it with these new modifications/additions:

1. The shape of the precomputed pieces is now defined by the following constraints, represented in Figure 1: **i)** The piece must have a rectangular component of width $w > 0$ and height $hr \geq 0$. **ii)** It must have a right triangle component, whose width is equal to w and height $ht \geq 0$. **iii)** The sum of heights hr and ht is strictly greater than 0. These shape definitions allow for rectangular, triangular, and trapezoidal insulation pieces.

2. We use two rotational values (0° and 90°) when working with rectangular elements. These rotations offer the possibility for the sides of the items to align in an orthogonal way to the edges of the bins. Although this is not enough to obtain optimality in all cases [7], this is a great compromise between simplicity and results that are more than sufficient for our particular domain. For non-rectangular elements defined by the shape presented in point 1, we are required to use more than two rotations. Since at most one side of the trapezoidal shape has two right angles, two possible rotational values (180° and 270°) are added to ensure that it can correctly align to the bin. Additionally, to account for the possible placements of the right triangle in the sheet, it is possible to flip the piece over. Thus, we go from two possible rotations to eight (four rotations if not flipped, four if flipped) when working with a shape that has a triangle height $ht > 0$.

3. Sheets are not all the same size. For SOKÏO and many other companies, the ability to reuse scraps and smaller sheets has great potential value. As a result, they need to be able to specify sheets that have dimensions smaller than the ones predefined for new sheets. These smaller sheets represent scraps that have been acquired by cutting insulation sheets in the past. The model has to be able to take these scraps into account so that they can be reused in the future. Following the above nomenclature, these different-sized bins are referred to as being **strongly** heterogeneous [27].

4. Although the main objective remains similar, we now aim to minimize the total area of the sheets in which the pieces are nested, rather than only their number. A second objective is also introduced. This is justified by the fact that the "creation" of scraps of sizes large enough to be reused later for nesting insulation pieces is critical, as implied by point 3. This second objective aims to position the insulation pieces in the sheet in a way that maximizes the reusability of its wasted area. This goal is complex to describe as a quantifiable metric due to its subjective nature, and this is part of the reason why the literature on this exact subject is rare, especially when working with constraint programming.

4.2 Instance specification

A formal specification of an instance of the described problem is defined as follows: We are given two sets: **i)** Let $\mathcal{P}$ be the set of indices of the insulation pieces. For each $p \in \mathcal{P}$, the piece p is defined by its width w_p, its rectangular height hr_p and its triangular height ht_p. **ii)** The set $\mathcal{R}$ contains the indices of rectangular sheets, which have predefined widths and heights (sw_r and sh_r $\forall$ $r \in \mathcal{R}$). These are ordered by non-decreasing area $sw_r \cdot sh_r$, and represent old heterogeneous insulation scraps that can be reused as bins. Using these sets, we define $\mathcal{S}$, which contains all the scrap sheets in $\mathcal{R}$ plus $|\mathcal{P}|$ new sheets that are added in sufficient quantities to make the instance feasible. The new sheets all have the same dimensions. Each sheet $s \in \mathcal{S}$ is defined with their width sw_s and height sh_s.

The goal is to assign each piece $p \in \mathcal{P}$ a sheet $s \in \mathcal{S}$, a position in the sheet (considering that the piece's origin is located at its bottom left corner), and a rotation such that none of the pieces overlap, while minimizing the total area of the used sheets. After finding optimality for the area of the sheets that are used, we also want to optimize the scrap reusability, further refining the solution of the problem. All this being said, we can consider the typology brought up by Wäscher et al. [27] to classify the problem as a variant of the residual bin packing problem (RBPP), where the dimensions of the residual pieces are considered an objective during the optimization process. Of course, a quantifiable metric is needed to optimize this value, which will be introduced in Section 5.3.2. Figure 2 shows two possible placements of the same pieces in a sheet. The example in Figure 2a is objectively worse than the one in Figure 2b when comparing the reusability of the scraps.

5 Methodology

5.1 Preprocessing - converting pieces to rectangles

The NP-Hardness of the problem makes it difficult for large instances to be computed to optimality in a reasonable time. First, having to deal with shapes that contain angles non-divisible by 90° adds an incredible layer of complexity, as computing the collisions between the shapes now relies on using trigonometric functions rather than simple edge placement comparisons. The eight possible rotations also make the solver's search tree
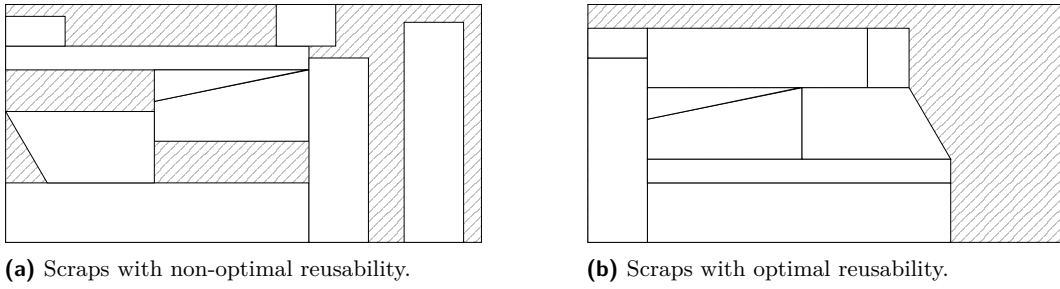
(a) Scraps with non-optimal reusability.   **(b)** Scraps with optimal reusability.

Figure 2 Packing examples: Pieces (white rectangles) are packed within the hatched sheet. Visible hatched areas are the resulting scraps.
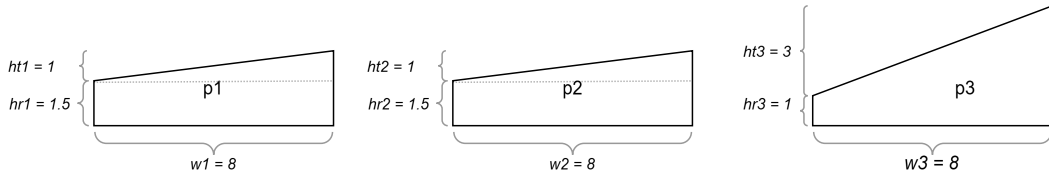

Figure 3 Instance being passed to the different preprocessing algorithms

exponentially larger than if only two were used. That being said, four different polynomial-time preprocessing algorithms will be introduced and compared. Figure 3 shows an example of a set of pieces, which is sent to the different preprocessing methods to be analyzed in Figure 4. The goal of these algorithms is to reduce the search space by transforming the pieces into rectangles before sending them to the solver. Each preprocessing method receives the same input (assuming that it is working with the same instance), this being the set of pieces $\mathcal{P}$ and their dimensions. They all also return an output of the same form: First, each preprocessing returns two arrays of the same length that contain the dimensions of the newly created rectangular shapes, W and H. Second, they also return an array named SUB, which will be explained in more detail in Section 5.1.2. Note that the preprocessing presented in Section 5.1.4 introduces a way to further decompose predefined insulation pieces, which has significant implications for the definition of the problem in itself. We allow the Slope mapping + preprocessing to decompose the non-rectangular pieces in such a way that they can be represented as a rectangle, reducing the losses induced from the piece's slope to zero. These pieces are subdivided into exactly three new smaller pieces. Subdividing further could allow for slightly better packing, but is not allowed since the factory wants to limit the necessary cutting to a minimum.

5.1.1 Bounding box

The Bounding box preprocessing is depicted in Figure 4a. Each piece $p \in \mathcal{P}$ is transformed into the smallest rectangle that can fully contain it. The width of the new rectangle is equal to w_p, while its height is set to $hr_p + ht_p$. Of course, while being fast and easy to comprehend, this preprocessing generates a substantial amount of scraps when working with trapezoid pieces, since treating them as their bounding rectangles takes up more space than they really occupy. Pieces with a triangular height ht_p of 0 do not create additional losses.

(a) Bounding box   **(b)** Staircase

(c) Slope mapping   **(d)** Slope mapping +
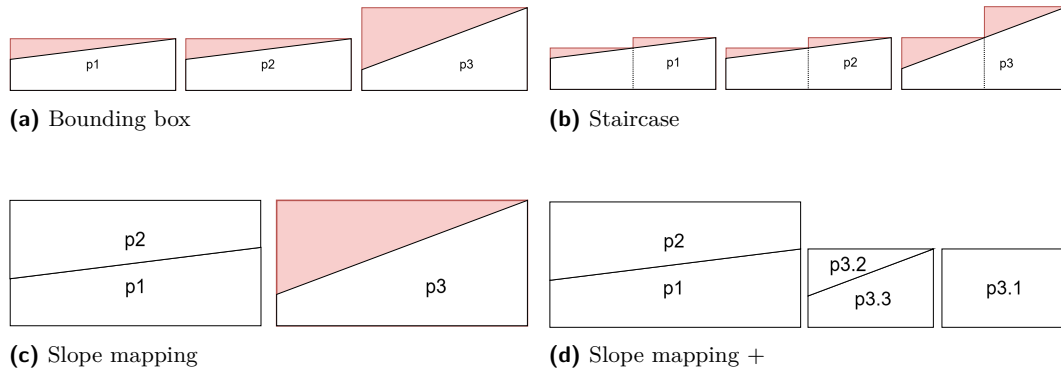
Figure 4 Preprocessing algorithms examples

5.1.2 Staircase

The Staircase preprocessing goes one step further in the same direction as Bounding box did and creates multiple bounding boxes for each piece that has a triangular height component $ht_p > 0$, reducing the amount of wasted space. Figure 4b shows this reduction by highlighting the way it forces the model to consider pieces. Each of these pieces have their width w_p divided into sub subdivisions of equal width $\left\lfloor \frac{w_p}{sub} \right\rfloor$, with any remainder $w_p \bmod sub$ added to the last segment. The algorithm calculates the bounding box for each subdivision using the original dimensions of the shape. For each subdivision, the starting and ending points relative to the original width of the shape are known. It is possible to use the heights of the piece hr_p and ht_p together with this information to calculate their respective heights. This calculation is shown in the following:

Suppose that the triangular height ht_p of a piece is always pointing upward and that its bottom left corner is at the origin $(0,0)$. The piece is also flipped so that its upper corners are at coordinates $(0, hr_p)$ and $(w_p, hr_p + ht_p)$. Each subdivision has the x position of their right boundary referred to as rb, which always has a value between 0 and w_p (included). The following expression is then used to calculate the height of each subdivision: $\left\lceil hr_p + \frac{rb}{w_p} \cdot ht_p \right\rceil$. The bounding boxes are always high enough to contain their entire subdivision, as the value of rb is always greater than the value of the x position of the left boundary for each subdivision.

Rectangular pieces that have a triangular height ht_p of 0 are treated differently. These pieces are not subdivided, so their bounding box is the same as the piece itself. The SUB array encodes the subdivision information for the model. In short, SUB is an array of length $|\mathcal{P}|$, where SUB_p represents the number of subdivisions in which the piece p has been split, $\forall p \in \mathcal{P}$. Since the pieces are always divided into the same number of subdivisions, the only two possible values in SUB are sub, if the piece is a triangle or a trapezoid, or 1 if the piece is a rectangle. Also note that an important constraint is added to the model when working with subdivisions. Since the shape is not split into smaller parts but is only represented as a combination of bounding boxes, it is important that these boxes remain connected when being translated or rotated. These connections between the different subdivisions make it so that the piece can be rotated in the eight possible rotations mentioned in Section 4 (each individual subdivision only truly rotates 0 or 90 degrees as they are rectangular, but the connection constraints model the eight feasible rotations).

5.1.3   Slope mapping

The Slope mapping preprocessing acts as an extension of the simple Bounding box approach by attempting to merge pieces that have an identical triangular height and width into one rectangular piece. Figure 4c shows an example of two pieces that are matched together as a singular rectangle, and the other piece, being alone, is instead considered as its bounding box. For each piece $p_1 \in \mathcal{P}$, if $ht_{p_1} \neq 0$, the algorithm searches for another piece p_2 with the same width and triangular height that has not yet been matched. If a match is found between the pieces p_1 and p_2, they are stacked to create one larger rectangle, provided that their combined height $hr_{p_1} + hr_{p_2} + ht_{p_1}$ fits within the predefined dimensions of a new sheet. If no match is found, p_1 is transformed in the same way as it would have been by the Bounding box approach. This approach has the potential to remove the wasted space from two pieces that have the same triangular height component each time a match is found.

5.1.4   Slope mapping +

The Slope mapping + preprocessing refines the basic Slope mapping approach by further addressing pieces that still inherently create losses when sent to the model, those being pieces with triangular components that cannot be paired with a matching piece. Each piece $p \in \mathcal{P}$ that matches this condition is decomposed by the algorithm into three smaller shapes. In other words, each trapezoidal shape is converted into one rectangle, one triangle, and another triangular or trapezoidal shape. The pieces *p3.1*, *p3.2*, and *p3.3*, as shown in Figure 4d, are the result of the decomposition of piece *p3* in Figure 3.

Applying the same assumptions made for the Staircase cutting phase, the piece can be decomposed in a way that makes it reconfigurable as a rectangle, preventing any left losses. First, two cuts are made, one at the coordinate $(\lfloor \frac{w_p}{2} \rfloor, 0)$ going up to $(\lfloor \frac{w_p}{2} \rfloor, hr_p + \lceil \frac{ht_p}{2} \rceil)$, and the other at $(\lfloor \frac{w_p}{2} \rfloor, hr_p + \lceil \frac{ht_p}{2} \rceil)$ going right to $(w_p, hr_p + \lceil \frac{ht_p}{2} \rceil)$. Following these cuts, the new pieces are considered to be two distinct rectangles. The first rectangle *p3.1* has a width of $\lceil \frac{w_p}{2} \rceil$ and a height of $hr_p + \lceil \frac{ht_p}{2} \rceil$. The second is made up of the pieces *p3.2* and *p3.3* stacked, creating a rectangle that has the same dimensions as the first. Although it might not be ideal to cut a piece into three smaller pieces, the insulation pieces context allows it. The Slope mapping + preprocessing is the only one among the four methods we propose that leads to the cutting of a piece into smaller ones.

5.2   Preprocessing - Cutting margins

Before sending the data to the first optimization model, the dimensions of the sheets and pieces are increased so that the cutting margins are taken into account. Insulation pieces are cut using a $\frac{1}{8}$" wide water jet. This is modeled by increasing the width and height of all (now rectangular) insulation pieces and insulation sheets by $\frac{1}{8}$". All pieces now contain half the width of the jet ($\frac{1}{16}$") as margins on each of their edges. Two touching pieces will always at least be distanced by their combined margins (equal to the width of the jet), so the water jet can directly cut between them and never impact the true sizes of the pieces. Similarly, pieces on the edge of a sheet benefit from its increased dimensions. The placement finished, the added dimensions of the sheet are removed, effectively removing the margins of the pieces that were placed on its side. Figure 5 shows the effect of the added margins, where 5a shows the addition of the margins to the pieces and sheets, 5b shows two pieces with their margins touching and touching the edges of the resized sheet, and 5c shows the final placement of the pieces, after the removal of the margins of the sheets.

(a) Addition of the margins   **(b)** Effects of the margins   **(c)** Sheet margins removal
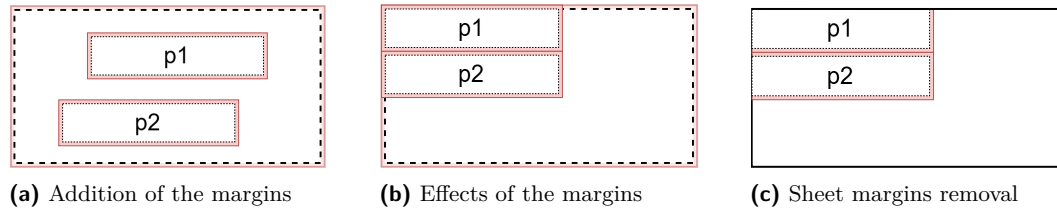
Figure 5 Preprocessing - Adding the margins to the pieces and sheets

5.3 Objective functions / models

The preprocessing step finished, it is possible to send the dimension values of the pieces and sheets as well as the subdivision data to the first constraint model, thus starting the optimization process. As mentioned earlier, two models are used sequentially. The first minimizes the total area of the used sheets, prioritizing the use of scraps over new sheets. The second optimizes the shape of the scraps. All preprocessing output their data in a way that can be used by the first constraint model. This model, in turn, outputs its data in a way that is usable by the second model.

The use of two sequential models is justified by the computation time. In essence, the solver used by the first model computes the optimal sheets to use before sending them to the next one. The second model then optimizes the reusability of the scraps without having to think about reducing the total area of the used sheets again. Optimizing those two objectives at the same time in a unique model does not work as efficiently because the solver spends a lot of time optimizing the reusability of scraps even before the optimal sheets are found.

5.3.1 Sheets minimization model

From the selected preprocessing output, we receive the following: **i)** The array SUB of length $|\mathcal{P}|$ indicates the number of subdivisions for a piece. For each piece $p \in \mathcal{P}$, we have $SUB_p = sub$ subdivisions if the piece p was subdivided by Staircase and $SUB_p = 1$ if not. We define the set $\mathcal{D} = \{1, \ldots, \sum_{p \in \mathcal{P}} SUB_p\}$ to be the indices of all subdivisions. **ii)** The arrays W and H, each of length $|\mathcal{D}|$, give the width and height of each subdivision. Pieces that were not subdivided are still considered a subdivision, which means that the dimensions of these pieces will also be present in W and H as a single subdivision. We also have **iii)**, the sheet widths SW and the sheet heights SH which are arrays of length $|\mathcal{S}|$. As stated earlier, the sheets in $\mathcal{S}$ are ordered by non-decreasing area, meaning that $\forall s \in \mathcal{S}$ where $s > 1, SW_s \cdot SH_s \geq SW_{s-1} \cdot SH_{s-1}$. The first model is described below and uses the constants and variables presented in tables 1 and 2.

Name	Value	Description
SA_s	$SW_s \cdot SH_s, \forall s \in \mathcal{S}$	Area of the sheet s (already ordered by non-decreasing area)
SL_s	$\sum_{i=1}^{s-1} SW_i, \forall s \in \mathcal{S}$	X position of the left edge of sheet s
I_p	$1 + \sum_{i=1}^{p-1} SUB_i, \forall p \in \mathcal{P}$	Index of the first subdivision of the piece p
R	$|\{s \in \mathcal{S} \mid SA_s < \max(SA)\}|$	The number of sheets that are scraps

Table 1 Names, values and descriptions of model constants

Name	Domain	Description
$sheet_d$	$\mathcal{S}, \forall d \in \mathcal{D}$	The sheet in which the d subdivision is nested
l_d	$\{0, \ldots, \sum_{s \in \mathcal{S}} SW_s\}, \forall d \in \mathcal{D}$	X position of the left edge of subdivision d
r_d	$\{0, \ldots, \sum_{s \in \mathcal{S}} SW_s\}, \forall d \in \mathcal{D}$	X position of the right edge of subdivision d
b_d	$\{0, \ldots, \max(SH)\}, \forall d \in \mathcal{D}$	Y position of the bottom edge of subdivision d
t_d	$\{0, \ldots, \max(SH)\}, \forall d \in \mathcal{D}$	Y position of the top edge of subdivision d
rw_d	$\{0, \ldots, \max(SW)\}, \forall d \in \mathcal{D}$	Width of subdivision d after rotation
rh_d	$\{0, \ldots, \max(SH)\}, \forall d \in \mathcal{D}$	Height of subdivision d after rotation
rot_d	$\{1, \ldots, 8\}, \forall d \in \mathcal{D}$	Rotation of subdivision d
ta	$\{0, \ldots, \sum_{s \in \mathcal{S}} SA_s\}$	Total area of the used sheets.

Table 2 Names, domains and descriptions of model variables

5.3.1.1 Constants

We position the sheets on the Cartesian plane, side by side from left to right, in non-decreasing order of area (in the order in which they are in $\mathcal{S}$). For each sheet $s \in \mathcal{S}$, SA_s denotes the area of s, and SL_s holds the x position of its left edge. For each piece $p \in \mathcal{P}$, I_p is the index of the first subdivision belonging to p. The constant R contains the number of sheets that are scraps, i.e. sheets whose area is smaller than the area of the largest sheet. The constants and their values are summarized in Table 1.

5.3.1.2 Variables

For each subdivision $d \in \mathcal{D}$, we have the decision variables l_d and b_d that are the coordinates of the lower left corner of the subdivision and the variable rot_d that encodes one of the eight possible rotations (four if d is flipped, four if not). In addition to the decision variables, there are *functional variables*. The values of these variables can be fully determined once the values of the decision variables are known. These variables are, for each subdivision $d \in \mathcal{D}$, the coordinates (r_d, t_d) of the upper-right corner of the subdivision, the sheet number $sheet_d$ on which lies the subdivision, the width rw_d and height rh_d of the subdivision after being rotated. Furthermore, the variable ta is used to represent the total area of the used sheets in the solution. The variables and their domains are summarized in Table 2.

5.3.1.3 Constraints

The following constraints ensure that subdivisions are fully nested inside a single sheet and that subdivisions do not overlap each other.

Constraints (1) to (3) apply to each piece $p \in \mathcal{P}$ and its subdivisions d. Constraint (1) applies to pieces that have a single subdivision and, therefore, are rectangular. It ensures that the rotation is either 0° or 90°. Constraints (2) and (3) consider the current rotation of subdivision d and fix the width and height of the subdivision.

$$SUB_p = 1 \implies rot_d \leq 2 \qquad \forall p \in \mathcal{P}, d \in \{I_p, \ldots, I_p + SUB_p - 1\} \qquad (1)$$

$$rot_d \bmod 2 = 0 \implies \begin{cases} rw_d = H_d \\ rh_d = W_d \end{cases} \qquad \forall d \in \mathcal{D} \qquad (2)$$

$$rot_d \bmod 2 \neq 0 \implies \begin{cases} rw_d = W_d \\ rh_d = H_d \end{cases} \qquad \forall d \in \mathcal{D} \qquad (3)$$

Constraint (4) links the left-side coordinate, the right-side coordinate and the width of a subdivision together. The same is true for the top and bottom coordinates and the height of the subdivision. This constraint takes into account the rotation of the piece.

$$r_d = l_d + rw_d \wedge t_d = b_d + rh_d \qquad\qquad \forall d \in \mathcal{D} \qquad\qquad (4)$$

Constraints (5) to (7) have two purposes: 1) to force each subdivision $d \in \mathcal{D}$ to be fully contained in one sheet and 2) to assign the sheet in $\mathcal{S}$ on which subdivision d lies to the variable $sheet_d$.

$$l_d \geq SL_{sheet_d} \qquad\qquad \forall d \in \mathcal{D} \qquad\qquad (5)$$

$$r_d \leq SL_{sheet_d} + SW_{sheet_d} \qquad\qquad \forall d \in \mathcal{D} \qquad\qquad (6)$$

$$t_d \leq SH_{sheet_d} \qquad\qquad \forall d \in \mathcal{D} \qquad\qquad (7)$$

Constraint (8) links the subdivisions of a piece $p \in \mathcal{P}$. It ensures that the subdivisions are assigned to the same sheet and the same rotation. Finally, it ensures that the subdivisions are positioned side by side. The right edge of one subdivision coincides with the left edge of the other subdivision, or the top edge of one subdivision coincides with the bottom edge of the other subdivision. The constraint takes into account the eight combinations of rotations and flips.

$$\forall p \in \mathcal{P}, d \in \{I_p, \ldots, I_p + SUB_p - 2\} :$$

$$sheet_d = sheet_{d+1} \wedge rot_d = rot_{d+1} \wedge$$

$$\begin{pmatrix} rot_d \bmod 4 = 1 \rightarrow r_d = l_{d+1} \\ rot_d \bmod 4 = 2 \rightarrow t_d = b_{d+1} \\ rot_d \bmod 4 = 3 \rightarrow l_d = r_{d+1} \\ rot_d \bmod 4 = 0 \rightarrow b_d = t_{d+1} \end{pmatrix} \wedge \begin{pmatrix} rot_d \in \{1,7\} \rightarrow b_d = b_{d+1} \\ rot_d \in \{2,8\} \rightarrow r_d = r_{d+1} \\ rot_d \in \{3,5\} \rightarrow t_d = t_{d+1} \\ rot_d \in \{4,6\} \rightarrow l_d = l_{d+1} \end{pmatrix} \qquad (8)$$

The constraint DIFFN ensures that the subdivisions do not overlap on each other. This constraint has in its scope the positions of the subdivisions and their dimensions. We also use the CUMULATIVE constraint. It is redundant but comes with more filtering algorithms that improve the performance of the model. The CUMULATIVE constraint only considers the positions of the subdivisions on the x-axis, as the range defined by the y-axis is not large enough for its filtering algorithms to offer any improvements.

$$\text{DIFFN}(l, b, rw, rh) \qquad\qquad (9)$$

$$\text{CUMULATIVE}(l, rw, rh, \max(SH)) \qquad\qquad (10)$$

Constraint (11) constrains ta, which is the objective variable to minimize, to be equal to the total area of the sheets used by the solution. The constant R indicates the number of scrap sheets that are available for use. These scrap sheets have smaller areas than the new sheets, they are therefore labeled with indices smaller than or equal to R. The summation in (11) adds the area of the used scrap sheets. The second term computes the area of new sheets. Since all have the same area, we simply multiply the area of new sheets, $\max(SA)$, by the number of new sheets used, $\max(sheet) - R$. Using $\max(sheet) - R$ to encode the number of used new sheets forces the solver to break symmetries by assigning pieces to new sheets with lower indices. We found that using this formulation instead of explicitly specifying a value precedence constraint among the new sheets offered equivalent results while being slightly faster.

$$ta = \left(\sum_{\substack{s \in \{1, \ldots, R\} \\ \exists d \in \mathcal{D}, sheet_d = s}} SA_s \right) + \max(SA) \cdot (\max(sheet) - R) \qquad (11)$$
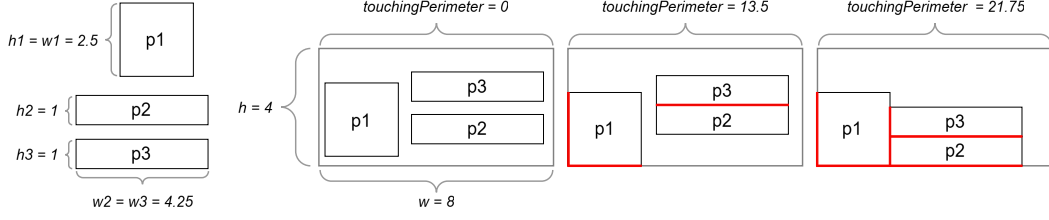
Figure 6 Computing the touching perimeter metric

Constraint (12) breaks symmetry between similar solutions where the pieces nested in a new sheet can be swapped for the pieces nested in another new sheet. This symmetry occurs less often with scrap sheets, as their area can differ. The constraint states that the first piece can only be nested on the first new sheet, the second piece on the first two sheets, and so on.

$$sheet_d \leq p + R \qquad\qquad \forall p \in \mathcal{P}, d \in \{I_p, \ldots, I_p + SUB_p - 1\} \qquad\qquad (12)$$

We implemented this model in MiniZinc using the Chuffed solver [10], which offers the priority search [13] annotation to model the branching heuristic.

```
solve::priority_search(l,
    [int_search([rot_d, l_d, b_d], input_order, indomain_min) | d ∈ D ],
    smallest, complete) minimize ta;
```

The priority search selects the subdivision that can be placed the farthest to the left on the sheet with the smallest index. Once the subdivision is selected, it fixes its rotation and position (in that order). The solver also uses solution-based phase saving [11] as it offers a significant speed increase in the time needed to compute optimality.

Using a contiguous coordinate system for packing the pieces on the sheets instead of the classic method where each sheet has its own coordinate system avoids the need for optional variables, hence simplifying the way the problem is modeled.

5.3.2   Scrap reusability maximization model

The subsequent model uses the same base as the first model (Section 5.3.1). We only change the objective function, some data given as input, and the branching heuristic.

We introduce the new objective function, which is inspired by the *touching perimeter* heuristic introduced by Lodi et al. [19]. The intuition is to maximize the length of the perimeter of the subdivisions that touches either the side of a sheet or the side of another subdivision. Maximizing this value is equivalent to maximizing the density of the pieces in the sheet. This also indirectly concentrates the waste in each sheet, increasing the size of the scraps and their reusability. Where Lodi et al. [19] use a greedy heuristic to try to maximize the touching perimeter, our objective maximizes it directly.

Figure 6 shows three pieces that need to be packed in the most efficient way. In the first sheet, those pieces float in the middle without touching any edges, resulting in an objective value of 0 and a suboptimal packing. In the second and last sheets, their placement is further optimized, and we can clearly see parts of their perimeter that come in contact. In this example, the last sheet shows the best way to place the three pieces, as it offers the best scrap reusability in terms of the pieces SOKÏO often produce.

The objective variable of the total used sheet area ta is discarded and replaced by the touching perimeter variable tp, whose domain is $\{0, \ldots, 2 \cdot \sum_{d \in \mathcal{D}}(W_d + H_d)\}$ and whose

value should be maximized. Constraint (13) constrains tp to be equal to the total touching perimeter of the subdivisions where the notation $[\![b]\!]$ returns 1 if b is *true* or 0 otherwise. Specifically, the first summation computes the perimeter of the subdivisions that are in contact with the edges of the sheet in which they are nested, while the second summation computes the perimeter of the subdivisions that are in contact with the edges of other subdivisions. The only pairs of subdivisions that are considered are those that are on the same sheet. Each pair is considered at most once ($d_1 < d_2$), which means that the touching perimeter will only be counted once, when both touching edges should be taken into account. This explains the multiplication by two, which makes it so that the touching edges of the subdivisions have their touching perimeter added correctly.

$$
tp = \sum_{d \in \mathcal{D}} \Big([\![l_d = SL_{sheet_d}]\!] \cdot rh_d + [\![r_d = SL_{sheet_d} + SW_{sheet_d}]\!] \cdot rh_d +
$$

$$
[\![b_d = 0]\!] \cdot rw_d + [\![t_d = SH_{sheet_d}]\!] \cdot rw_d \Big) +
$$

$$
2 \times \sum_{\substack{d_1, d_2 \in \mathcal{D} \\ d_1 < d_2 \\ sheet_{d_1} = sheet_{d_2}}} \Big([\![r_{d_1} = l_{d_2} \vee r_{d_2} = l_{d_1}]\!] \cdot \max(0, \min(t_{d_1}, t_{d_2}) - \max(b_{d_1}, b_{d_2})) +
$$

$$
[\![t_{d_1} = b_{d_2} \vee t_{d_2} = b_{d_1}]\!] \cdot \max(0, \min(r_{d_1}, r_{d_2}) - \max(l_{d_1}, l_{d_2})) \Big) \qquad (13)
$$

The input of this model is still a list of subdivisions and sheets, but we only feed this second model with the sheets that were assigned pieces by the first model. This reduces the search space and ensures that the total area of the used sheets remains optimal.

To solve this model, we use the CP-SAT solver [23] using four threads (using six threads did not improve performance in any way, and more than six worsened it). CP-SAT does not offer solution-based phase saving and does not offer a priority search. Despite that, we have found that CP-SAT is faster than Chuffed for this specific model. The second model uses this specific search annotation that selects the variable in the l and b arrays with the smallest value in its domain and branches on that value.

```
solve::int_search( l ∥ b, smallest, indomain_min) maximize tp;
```

6 Experimentations

6.1 Benchmarks

This section benchmarks both models presented in this paper and compares our solutions with those of SOKÏO, for a particular instance. All experiments were run on a computer with a 6-core Intel Core i7-10750H CPU @ 2.60 GHz and 16 GB of memory.

6.1.1 Sheets minimization model

The experiments consisted of four instances ranging from 15 to 188 pieces, which were analyzed with each of the preprocessing methods. For the Staircase preprocessing, we experimented with 2 and 4 as values for the number of subdivisions *sub*. Each of these tests was also performed with 3 different sets of scrap sheets $\mathcal{R}$. The first one contains no scrap, so only new sheets can be used. The other ones contain 30 and 60 scraps of different sizes.

We use three key metrics to compare the different experiments made with the first model: 1) the number of new sheets used, 2) the waste ratio, expressed as a percentage of the total
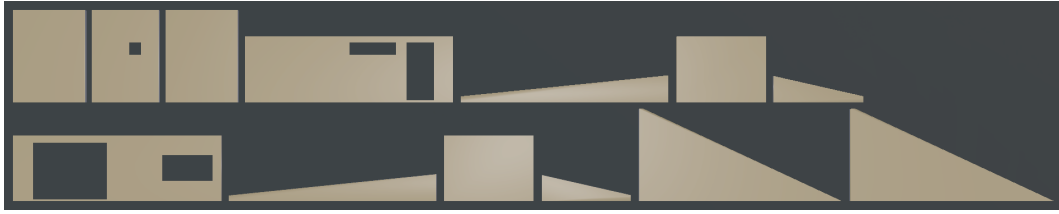
■ **Figure 7** CLT panels breakdown of instance 2

used sheets area. This is calculated using the following expression:

$$\left(1 - \frac{\text{Total area of the pieces (before the preprocessing)}}{\text{Total area of the used sheets}}\right) \cdot 100\%,$$

as well as 3) the time required to compute the solution, which is by far the most valuable resource for SOKÏO.

Our results will be compared with those obtained by SOKÏO on instance 2, which represents the only structure currently in production. This structure is composed of prefabricated panels, which are shown in Figure 7. SOKÏO, exactly as we now do in an automated way, had to compute the best shapes, sizes, placements, etc. of the insulation pieces so that they would completely cover the structure's panels, excluding the different openings. Their manual computation landed them with a set of pieces that were then nested in a total of 46 new sheets. Note that no scraps were used to create these pieces or kept after the cutting process due to their suboptimal sizes. SOKÏO's waste metric on this specific instance is 21.58%, and their manual nesting took an entire week to complete.

Table 3 shows our results for the instance 2 produced by SOKÏO, as well as for the others. For this model, all instances were subject to a 5-minute timeout. For instance 2, we were able to calculate a solution that is much more efficient than what SOKÏO achieved manually on site, with an optimal solution of 39 new sheets obtained in 2.91 seconds using the Slope mapping + preprocessing. This equates to a material waste of 6.71%. Compared to SOKÏO's solution of 46 new sheets and 21.58% wasted material (which took a week of planning). Looking at these results, it is safe to say that the two objectives that were a priority for SOKÏO, reducing the amount of time needed to obtain the solutions and improving their quality based on the number of new sheets used, have been achieved. Figures 8 and 9 in Appendix A show visualizations of the scrap minimization process for instance 2 using the Bounding box and Slope mapping + preprocessings. All instances in Table 3 are feasible.

Although SOKÏO did not yet work on the other instances, we can assume that the time needed to compute their solution, as well as the number of new sheets used, will be much lower in the results provided by the first model than the manual results SOKÏO would have obtained if they had done so. The models can also work with scraps. This allows SOKÏO to recycle them, which they could not even consider before due to their small sizes in the solution they had. Table 3 shows that while instances that contain scraps take more time to compute to optimality, they offer even better solutions both in terms of the new sheets used and in the amount of waste generated (which is computed on all sheets used, not only new ones). The results also show that the Slope mapping + preprocessing is the best of the four we presented on these metrics. This is easily explained by the fact that the subdivisions generated by Slope mapping + do not contain any wasted area, while those generated by the other three preprocessings on triangular and trapezoid pieces do. As mentioned above, Slope mapping + cuts some pieces into three smaller pieces, and the current context allows it. For a

Table 3 Model 1 - Performance Metrics Across All Instances and Preprocessing Methods

Preprocessing	$|\mathcal{D}|$	New Sheets [0/30/60]			Waste (%) [0/30/60]			Time (s)[1] [0/30/60]		
Instance 1 (15 Pieces)										
Bounding box	15	9 / 7 / 7			20.67 /13.13/10.51			0.51 / 0.60 / 0.74		
Staircase - $sub = 2$	21	9 / 7 / 7			20.67 /13.13/10.51			0.63 / 0.78 / 0.92		
Staircase - $sub = 4$	33	8 / 8 / 7			**10.76**/10.76/10.51			0.83 / 1.27 / 1.59		
Slope mapping	15	9 / 7 / 7			20.67 /13.13/10.51			0.47 / 0.60 / 0.69		
Slope mapping +	21	8 / 6 / 5			**10.76**/**7.51**/**5.01**			0.71 / 1.01 /10.23		
Instance 2 (74 Pieces)										
Bounding box	74	44 / 42 / 39			17.31 /16.42/14.11			2.17 / 4.88 /64.77		
Staircase - $sub = 2$	94	43 / 42 / 38			15.38 /15.14/13.86			6.21 / 8.68 / –		
Staircase - $sub = 4$	134	43 / 42 / 38			15.38 /14.39/12.10			12.96/19.36/ –		
Slope mapping	67	42 / 40 / 36			13.37 /11.73/ 9.29			1.90 / 3.45 /13.80		
Slope mapping +	73	39 / 37 / 35			**6.71** /**5.34**/**4.30**			2.91 / – / –		
Instance 3 (106 Pieces)										
Bounding box	106	65 / 63 / 60			12.26 /11.32/ 9.46			– / – / –		
Staircase - $sub = 2$	114	65 / 62 / 60			12.26 /11.10/ 9.69			– / – / –		
Staircase - $sub = 4$	130	65 / 62 / 59			12.26 /10.58/ 9.05			– / – / –		
Slope mapping	106	65 / 63 / 57			12.26 /11.32/ 9.46			– / – / –		
Slope mapping +	114	63 / 60 / 59			**9.47** / **7.41**/**7.18**			– / – / –		
Instance 4 (188 Pieces)										
Bounding box	188	123/120/118			10.29 / 8.90 / 8.34			– / – / –		
Staircase - $sub = 2$	214	122/119/116			9.55 / 8.71 / 8.03			– / – / –		
Staircase - $sub = 4$	266	122/119/118			9.55 / 8.83 / 8.19			– / – / –		
Slope mapping	188	123/119/117			10.29 / 8.90 / 8.34			– / – / –		
Slope mapping +	214	116/115/112			**4.87** / **4.44**/**4.52**			– / – / –		

Values in columns represent results for instances where $\mathcal{S}$ contains [0 scraps / 30 scraps / 60 scraps].
Lowest waste percentage per scrap group within each instance highlighted in bold.
[1] '–' indicates the 300-second time limit was reached.

problem where this behavior is not allowed, the Slope mapping and Staircase preprocessings also offer great results, especially when the sheet set $\mathcal{S}$ contains reused scraps.

6.1.2 Scrap reusability maximization model

The benchmark for the second model consists of 12 instances obtained by testing each of the 4 base instances from model 1 with the 3 different numbers of scrap sheets, $|\mathcal{R}| \in \{0, 30, 60\}$. These new instances are based on the results provided by the best preprocessing in Table 3. In all cases, the best preprocessing (based on the waste ratio metric) was Slope mapping +, so all 12 runs were performed using the results of this preprocessing. Table 4 shows the results obtained for these 12 experiments after a 30-minute timeout (which was reached in all instances). A higher touching perimeter percentage (TPP) means a higher density of the pieces in each sheet, which translates into better packing and more reusable scraps. A TPP of 100% could only be obtained with a waste ratio of 0%. Bigger instances generally provide a higher TPP, because the larger number of possible combinations yields a higher probability

Table 4 Model 2 - Touching Perimeter Ratio (Solutions from model 1 using Slope mapping +).

Instance	Touching Perimeter (%)[1]		
	0 Scraps	30 Scraps	60 Scraps
Instance 1 (15 Pieces)	81.71	77.68	77.88
Instance 2 (74 Pieces)	78.82	77.43	77.54
Instance 3 (106 Pieces)	83.68	–	–
Instance 4 (188 Pieces)	73.00	–	–

[1] '–' indicates no result was obtained before reaching the 30-minute time limit.

that efficient packing can be achieved. Figure 10 in Appendix A shows visualizations of the scrap reusability maximization process for instance 2, using the result from the Slope mapping + preprocessing in Figure 9. All instances in Table 4 are feasible.

Table 4 shows that the model generates initial solutions quickly (and keeps increasing their quality) for small instances that reuse no scraps, but takes a longer time for larger instances that try to reuse many scraps. Of course, the objective of the second model is way more complex than the one of the first model, and large instances suffer from that.

When time is not an issue, waiting for the solver to output solutions for the second model is clearly beneficial. After 16 hours, the solver achieved a TPP of 88.04% on the 60-scrap version of instance 4, which is by far the largest and most complex. Although 16 hours may seem long, it is still a short amount of time compared to the manual results of SOKÏO. Where SOKÏO took a week to compute their results for instance 2 and did not generate any reusable scraps, we are able to guarantee that our solution for instance 4 (which is more than twice the size of instance 2) obtained in 16 hours will result in scraps with great reusability.

6.2   A word on the dataset

All the different instances of the problem considered in this paper were created using the product configurator mentioned in Section 3. This software was developed using the Unity® game engine, and allows the user to edit the model of a structure to his/her liking before ordering it. This offers a load of possibilities in regards to industry 4.0 parametric product configuration, at the cost of needing to optimize each newly created building instance and its related parts, such as insulation cutting and nesting, as discussed.

7   Conclusion

We introduced four different preprocessing methods that work with trapezoidal shapes in the bin packing problem, using solvers that typically only work with rectangular shapes. We compared the nesting produced by our algorithms with the one manually obtained by SOKÏO. Each preprocessing achieved better results than those provided by SOKÏO. We introduced a new objective function that optimizes scrap reusability. This objective refines the results returned by the sheet minimization model and proposes new solutions that make better use of the space given by the optimal sheets that were found, allowing SOKÏO to create more reusable scraps. Directions for future work include the implementation of a constraint described by Beldiceanu et al. [5] that directly supports trapezoidal shapes. We also aim to further explore the implementation of the touching perimeter objective function to speed up the convergence of the second model. Finally, we want to model scrap sheets of non-rectangular shapes by fixing virtual pieces at locations where the scrap cannot be used.

References

1 Ranga Prasad Abeysooriya. *Cutting patterns for efficient production of irregular-shaped pieces.* PhD thesis, University of Southampton, 2017. URL: `https://eprints.soton.ac.uk/414693/1/23._Final_submission_of_thesis.pdf`.

2 Abderrahmane Aggoun and Nicolas Beldiceanu. Extending CHIP in order to solve complex scheduling and placement problems. In Jean-Paul Delahaye, Philippe Devienne, Philippe Mathieu, and Pascal Yim, editors, *JFPL'92, 1ères Journées Francophones de Programmation Logique, 25-27 Mai 1992, Lille, France*, volume 17, page 51, 1992. URL: `https://www.sciencedirect.com/science/article/pii/089571779390068A`, `doi:10.1016/0895-7177(93)90068-A`.

3 Brenda S. Baker, Edward G. Coffman Jr., and Ronald L. Rivest. Orthogonal packings in two dimensions. *SIAM J. Comput.*, 9(4):846–855, 11 1980. `doi:10.1137/0209064`.

4 N Beldiceanu and E Contejean. Introducing global constraints in chip. *Mathematical and Computer Modelling*, 20(12):97–123, 1994. URL: `https://www.sciencedirect.com/science/article/pii/0895717794901279`, `doi:10.1016/0895-7177(94)90127-9`.

5 Nicolas Beldiceanu, Qi Guo, and Sven Thiel. Non-overlapping constraints between convex polytopes. In Toby Walsh, editor, *Principles and Practice of Constraint Programming - CP 2001, 7th International Conference, CP 2001, Paphos, Cyprus, November 26 - December 1, 2001, Proceedings*, volume 2239 of *Lecture Notes in Computer Science*, pages 392–407. Springer, Springer, 2001. URL: `https://doi.org/10.1007/3-540-45578-7_27`, `doi:10.1007/3-540-45578-7_27`.

6 Julia A. Bennell and José Fernando Oliveira. A tutorial in irregular shape packing problems. *J. Oper. Res. Soc.*, 60(S1):S93–S105, 2009. `arXiv:https://doi.org/10.1057/jors.2008.169`, `doi:10.1057/jors.2008.169`.

7 David W. Cantrell, Erich Friedman, et al. Packing unit squares in squares: A survey and new results. *The Electronic Journal of Combinatorics*, 5(DS7), 1998. URL: `https://www.combinatorics.org/files/Surveys/ds7/ds7v1-1998.pdf`.

8 CH Cheng, BR Feiring, and TCE Cheng. The cutting stock problem—a survey. *International Journal of Production Economics*, 36(3):291–305, 1994. `doi:10.1016/0925-5273(94)00045-X`.

9 Henrik I. Christensen, Arindam Khan, Sebastian Pokutta, and Prasad Tetali. Approximation and online algorithms for multidimensional bin packing: A survey. *Comput. Sci. Rev.*, 24:63–79, 2017. `doi:10.1016/j.cosrev.2016.12.001`.

10 Geoffrey Chu. *Improving combinatorial optimization.* PhD thesis, University of Melbourne, Australia, 2011. URL: `https://hdl.handle.net/11343/36679`.

11 Emir Demirovic, Geoffrey Chu, and Peter J. Stuckey. Solution-based phase saving for CP: A value-selection heuristic to simulate local search behavior in complete solvers. In John N. Hooker, editor, *Principles and Practice of Constraint Programming - 24th International Conference, CP 2018, Lille, France, August 27-31, 2018, Proceedings*, volume 11008 of *Lecture Notes in Computer Science*, pages 99–108. Springer, Springer, 2018. URL: `https://doi.org/10.1007/978-3-319-98334-9_7`, `doi:10.1007/978-3-319-98334-9_7`.

12 Andreas A. Falkner, Alois Haselböck, Gerfried Krames, Gottfried Schenner, and Richard Taupe. Constraint solver requirements for interactive configuration. In Lothar Hotz, Michel Aldanondo, and Thorsten Krebs, editors, *Proceedings of the 21st Configuration Workshop, Hamburg, Germany, September 19-20, 2019*, volume 2467 of *CEUR Workshop Proceedings*, pages 65–72. CEUR-WS.org, 2019. URL: `https://ceur-ws.org/Vol-2467/paper-12.pdf`.

13 Thibaut Feydy, Adrian Goldwaser, Andreas Schutt, Peter J Stuckey, and Kenneth D Young. Priority search with minizinc. In *ModRef 2017: The Sixteenth International Workshop on Constraint Modelling and Reformulation*, 2017. URL: `https://ozgurakgun.github.io/ModRef2017/files/ModRef2017_PrioritySearchWithMiniZinc.pdf`.

14 M. R. Garey and David S. Johnson. "strong" np-completeness results: Motivation, examples, and implications. *J. ACM*, 25(3):499–508, jul 1978. `doi:10.1145/322077.322090`.

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

16   Paul C Gilmore and Ralph E Gomory.   A linear programming approach to the cutting-stock problem.   *Operations research*, 9(6):849–859, 1961.   URL: `https://www.researchgate.net/publication/266478800_A_Linear_Programming_Approach_to_the_Cutting_Stock_Problem_I`, `doi:10.1287/opre.9.6.849`.

17   Mikael Z. Lagerkvist, Martin Nordkvist, and Magnus Rattfeldt. Laser cutting path planning using CP. In Christian Schulte, editor, *Principles and Practice of Constraint Programming - 19th International Conference, CP 2013, Uppsala, Sweden, September 16-20, 2013. Proceedings*, volume 8124 of *Lecture Notes in Computer Science*, pages 790–804, Berlin, Heidelberg, 2013. Springer. URL: `https://doi.org/10.1007/978-3-642-40627-0_58`, `doi:10.1007/978-3-642-40627-0_58`.

18   Andrea Lodi, Silvano Martello, and Michele Monaci. Two-dimensional packing problems: A survey. *Eur. J. Oper. Res.*, 141(2):241–252, sep 2002. `doi:10.1016/S0377-2217(02)00123-6`.

19   Andrea Lodi, Silvano Martello, and Daniele Vigo. Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems. *INFORMS J. Comput.*, 11(4):345–357, 1999. `doi:10.1287/ijoc.11.4.345`.

20   MiniZinc Documentation. *Redundant Constraints*. URL: `https://docs.minizinc.dev/en/stable/efficient.html#redundant-constraints`.

21   Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, and Guido Tack. Minizinc: Towards a standard CP modelling language. In Christian Bessiere, editor, *Principles and Practice of Constraint Programming - CP 2007, 13th International Conference, CP 2007, Providence, RI, USA, September 23-27, 2007, Proceedings*, volume 4741 of *Lecture Notes in Computer Science*, pages 529–543. Springer, Springer, 2007. URL: `https://doi.org/10.1007/978-3-540-74970-7_38`, `doi:10.1007/978-3-540-74970-7_38`.

22   Óscar Oliveira, Dorabela Gamboa, and Elsa Silva. An introduction to the two-dimensional rectangular cutting and packing problem. *Int. Trans. Oper. Res.*, 30(6):3238–3266, nov 2023. `doi:10.1111/itor.13236`.

23   Laurent Perron and Vincent Furnon. Or-tools. URL: `https://developers.google.com/optimization/`.

24   Andreas Schutt, Peter J. Stuckey, and Andrew R. Verden. Optimal carpet cutting. In Jimmy Ho-Man Lee, editor, *Principles and Practice of Constraint Programming - CP 2011 - 17th International Conference, CP 2011, Perugia, Italy, September 12-16, 2011. Proceedings*, volume 6876 of *Lecture Notes in Computer Science*, pages 69–84, Berlin, Heidelberg, 2011. Springer. URL: `https://doi.org/10.1007/978-3-642-23786-7_8`, `doi:10.1007/978-3-642-23786-7_8`.

25   Helmut Simonis and Barry O'Sullivan. Search strategies for rectangle packing. In Peter J. Stuckey, editor, *Principles and Practice of Constraint Programming, 14th International Conference, CP 2008, Sydney, Australia, September 14-18, 2008. Proceedings*, volume 5202 of *Lecture Notes in Computer Science*, pages 52–66, Berlin, Heidelberg, 2008. Springer. URL: `https://doi.org/10.1007/978-3-540-85958-1_4`, `doi:10.1007/978-3-540-85958-1_4`.

26   The MiniZinc Team. *The MiniZinc Handbook*. URL: `https://docs.minizinc.dev/en/latest/index.html`.

27   Gerhard Wäscher, Heike Haußner, and Holger Schumann. An improved typology of cutting and packing problems. *Eur. J. Oper. Res.*, 183(3):1109–1130, dec 2007. `doi:10.1016/j.ejor.2005.12.047`.

28   Linda L Zhang. Product configuration: a review of the state-of-the-art and future research. *International Journal of Production Research*, 52(21):6381–6398, nov 2014. `doi:10.1080/00207543.2014.942012`.
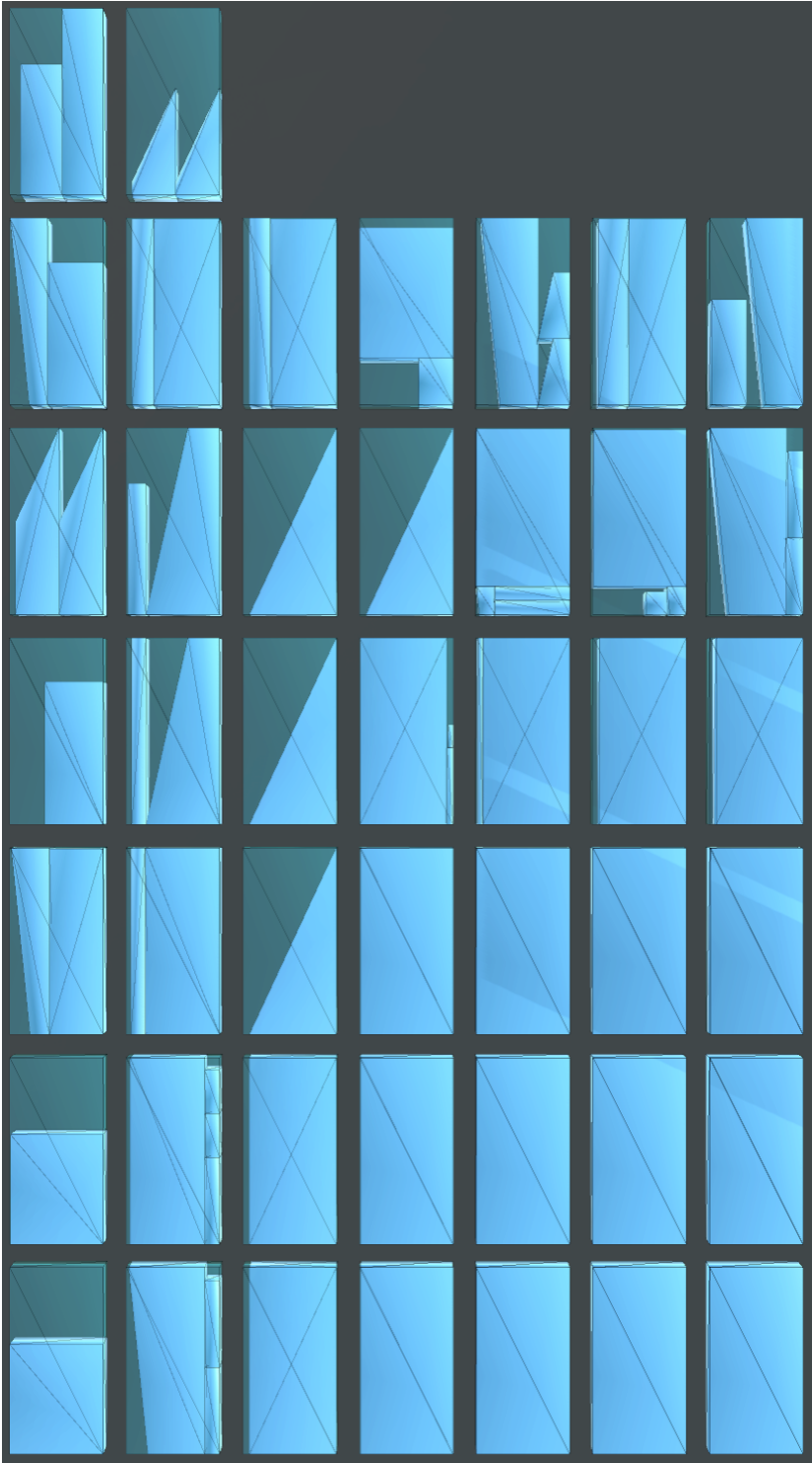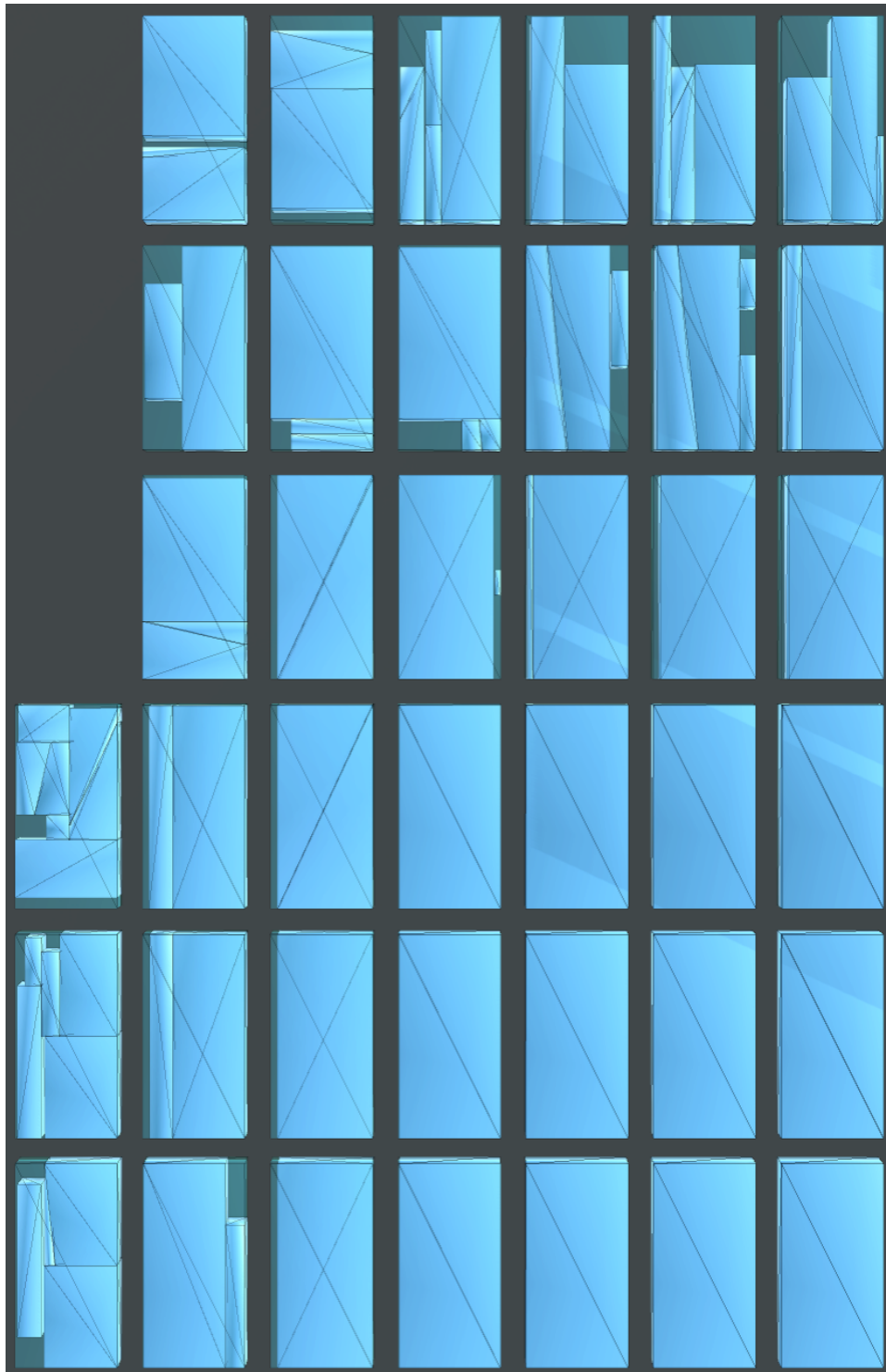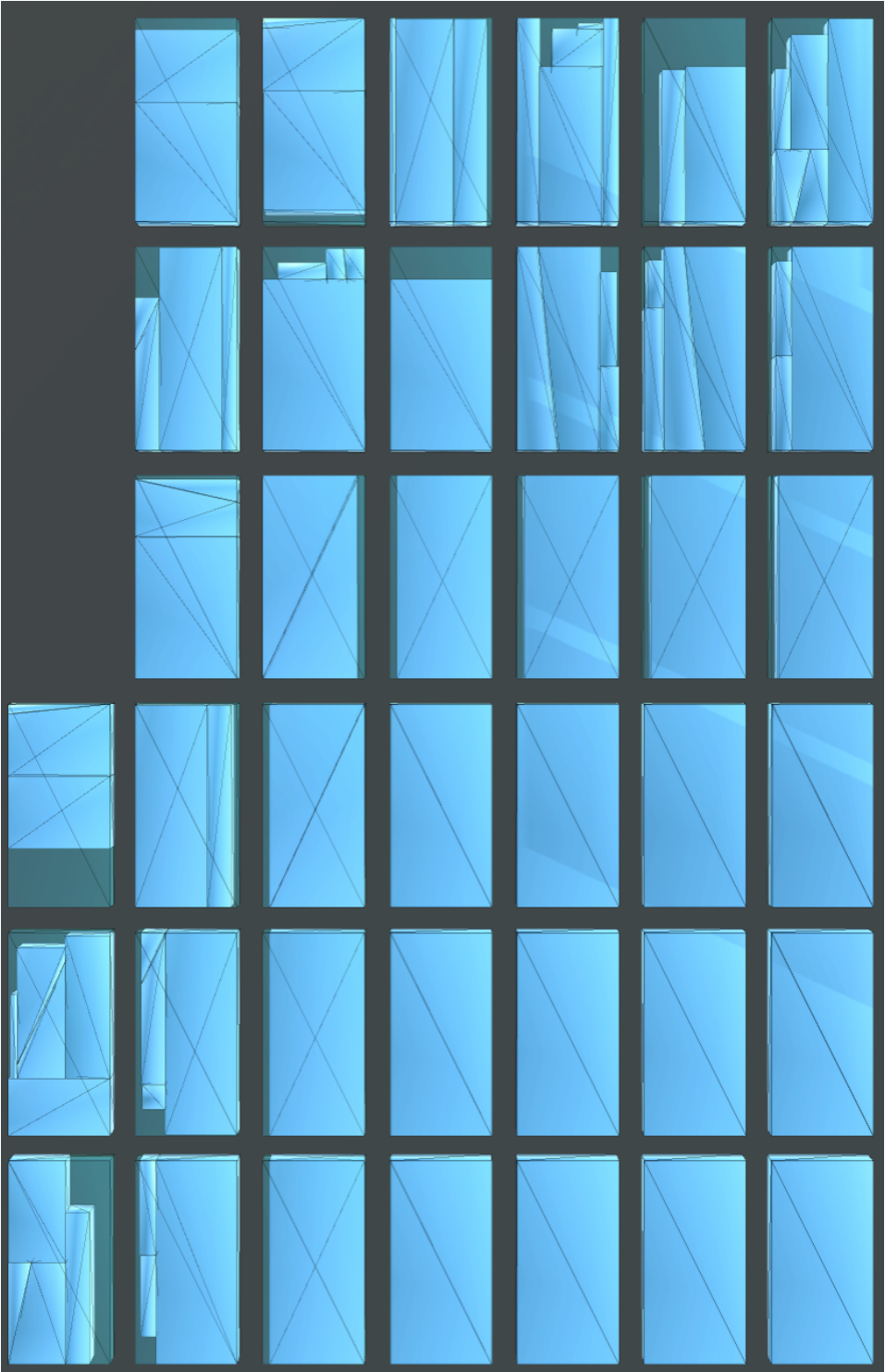
A Additional visualizations

Figure 8 Solution obtained with Bounding box preprocessing on instance 2 - 44 new sheets

654

Figure 9 Solution obtained with Slope mapping + preprocessing on instance 2 - 39 new sheets

Figure 10 Results from Figure 9 after being optimized with the scrap reusability maximization model for one hour, with a final touching perimeter percent of 78.84 %.