Mixed-Integer Programming models for nesting problems

Matteo Fischetti, Dipartimento di Ingegneria dell'Informazione, Università di Padova [*]
Ivan Luzzi, Dipartimento di Metodi Quantitativi, Università di Brescia [†]

18 September 2006; Revised, 20 September 2007

Abstract

Several industrial problems involve placing objects into a container without overlap, with the goal of minimizing a certain objective function. These problems arise in many industrial fields such as apparel manufacturing, sheet metal layout, shoe manufacturing, VLSI layout, furniture layout, etc., and are known by a variety of names: layout, packing, nesting, loading, placement, marker making, etc. When the 2-dimensional objects to be packed are non-rectangular the problem is known as the nesting problem. The nesting problem is strongly NP-hard. Furthermore, the geometrical aspects of this problem make it really hard to solve in practice.

In this paper we describe a Mixed-Integer Programming (MIP) model for the nesting problem based on an earlier proposal of Daniels, Li and Milenkovic, and analyze it computationally. We also introduce a new MIP model for a subproblem arising in the construction of nesting solutions, called the multiple containment problem, and show its potentials in finding improved solutions.

Keywords: MIP models, nesting, cutting&packing, containment problem, computational geometry.

1 Introduction

Several industrial problems involve placing objects into a container so that no two objects overlap. The general goal is either to minimize the used part of the container, or to

[*]Corresponding author. E-mail: `matteo.fischetti@unipd.it`
[†]E-mail: `iluzzi@eco.unibs.it`

find an optimal collection of objects that can be placed (either in terms of number or total area). These problems arise in many industrial fields such as apparel manufacturing, sheet metal layout, shoe manufacturing, VLSI layout, furniture layout, etc., and are known by a variety of names: layout, packing, nesting, loading, placement, marker making, etc. When the 2-dimensional objects to be packed are non-rectangular the problem is known as the *nesting* problem. The reader is referred to Dowsland and Dowsland [15] for a thoughtful survey paper, and to [6, 7, 18, 20, 33] for very recent papers on the topic.

In this paper we investigate the use of Mixed Integer Programming (MIP) models for solving the nesting problem or, more realistically, a subproblem (called the *multiple containment problem*) arising when some “large” objects have already been placed in the container, and one wants to make use of the unutilized area to place as many “small” objects as possible. Our main goal is to evaluate the viability of using modern MIP techniques to address practically-relevant nesting (sub-)problems.

The paper is organized as follows. In Section 2 we give a detailed description of the nesting problem, we study and review some fundamental computational geometry notions such as Minkowski sums and no-fit polygons. A basic MIP model is presented in Section 3, which is based on an earlier proposal of Daniels, Li and Milenkovic [12, 23]. In the same section we introduce a number of improvements aimed at strengthening the quality of the formulation (lifting) and at guiding the enumerative search for its optimal solution. Computational results are presented on a class of very difficult nesting instances (broken glass), showing that nesting instances of real size and complexity are unlikely to be solvable in practice. In view of this negative result, a more realistic goal is addressed in Section 4, namely to provide a (even approximate) MIP model for the multiple-containment subproblem that can be solved in reasonable time, and that can enhance the quality of the solutions provided by heuristic methods. Some conclusions are finally drawn in Section 5.

The present paper is based on the Ph.D. dissertation of the second author [25].

2 The nesting problem

The nesting problem addressed in the present paper can formally be described as follows. We are given a set $\mathcal{P} = \{1, \cdots, n\}$ of n two-dimensional *pieces* (some of which possibly identical), along with a rectangular placement region (*marker region*) with fixed height $maxY$ and virtually infinite length. The shape of each piece is defined by a simple (hole-less) polygon, described through the list of its vertices. Each piece is associated with a *reference point* whose (unknown) 2D *displacement coordinates* $v_i = (x_i, y_i)$ define its actual placement location in the marker region. Here x_i is viewed as the horizontal coordinate, y_i as vertical one, and $(0, 0)$ corresponds the leftmost bottom point of the marker region. The distances of the reference point of each piece i to the border of its bounding box define the values of top_i, $bottom_i$, $left_i$ and $right_i$; see Figure 1 for an illustration.

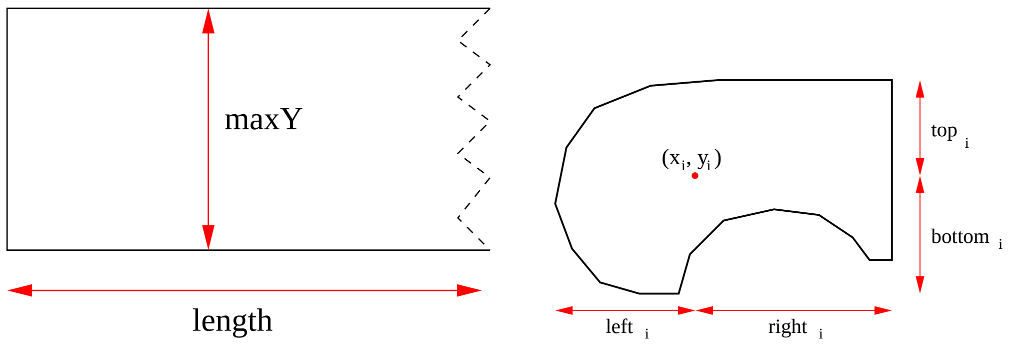


Figure 1: Input data for a nesting problem: pieces (left) and marker region (right).

The *length* of a given placement is defined as the horizontal coordinate x of the right-most used point of the marker region, namely

$$length = \max\{x_i + right_i : i \in \mathcal{P}\}.$$

The *efficiency* of a placement is defined as the ratio between the total area of the placed pieces and the area of the marker region where these pieces lay, i.e.,

$$\text{Efficiency} \;=\; \frac{\sum_{i=1}^{n} area_i}{length * maxY}$$

The nesting problem then consists in placing all the n pieces in the marker region, with no overlap and without exceeding the height $maxY$ of the marker region, so as minimize the associated *lentgh* (or, equivalently, to maximize the corresponding efficiency).

The *Minkowski sum* of two polygons A and B is defined as

$$A \oplus B \;=\; \{a + b : \; a \in A, \; b \in B\} \qquad \text{or equivalently:} \qquad A \oplus B \;=\; \bigcup_{b\,\in B} A^b$$

where notation $A^b \;=\; \{a + b : \; a \in A\}$ refers to the translation of polygon A by a given vector b.

The *Minkowski difference* of polygons A and B is defined as:

$$A \ominus B \;=\; \bigcap_{b\,\in B} A^b$$

The *no-fit polygon* between two polygons A and B is defined as

$$U_{AB} \;=\; A \oplus (-B)$$

where $-B = \{-b : \; b \in B\}$. When the reference point of polygon A is translated in the origin (as shown in Figure 2), the no-fit polygon can be interpreted as the trajectory followed by the reference point of B when it slides (without rotating) around polygon A.

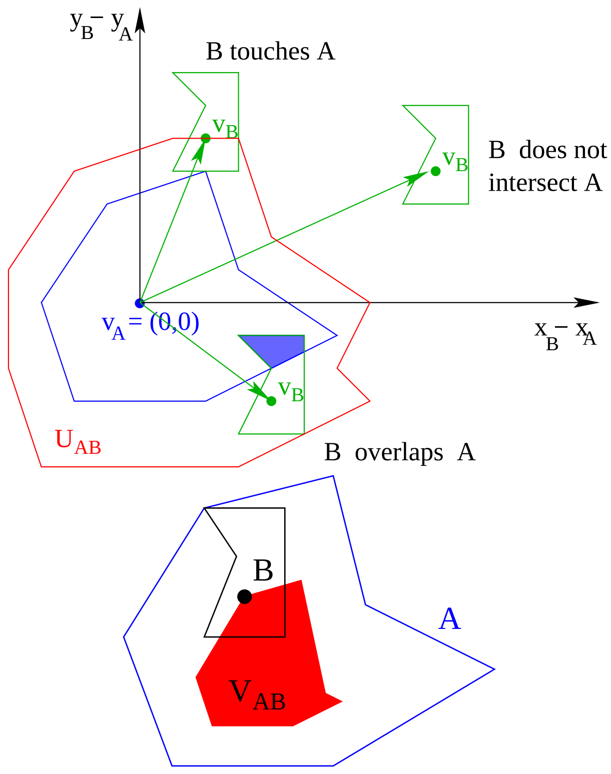


Figure 2: The no-fit polygon U_{AB} and the inner-fit region V_{AB} of two polygons A and B.

No-fit polygons play a cental role in modelling (and solving) nesting problems. Indeed, regardless of the actual position of two given polygons A and B in the plane, we can verify whether they intersect by simply checking if the difference of their displacement vectors, computed as $(x_B - x_A, y_B - y_A)$, belongs to the no-fit polygon U_{AB}.

Minkowski difference also offers an useful method to determine whether a polygon can be fully contained in another one without overlapping its border. This is important when we want to check whether a given piece lays entirely within the marker region, or to determine which is the free region of movement of a piece within a given hole. Let

A and B be two polygons and p be a point in the plane. Then $B^p \subseteq A$ if and only if $p \in A \ominus (-B)$. The *inner-fit region* of polygon B within polygon A is thus defined as

$$V_{AB} \;=\; A \ominus (-B)$$

As shown in Figure 2, piece B is entirely contained within piece A as long as the reference point of B belongs to the inner-fit region V_{AB} (shaded region in the figure).

3 A MIP model for the nesting problem

We first review a basic MIP model, due to Daniels, Li and Milenkovic [12, 23]. For each pair of pieces $i, j \in \mathcal{P}$, $i < j$, we compute the no-fit polygon U_{ij}. In order for pieces i and j not to overlap, we need to enforce that the difference of their displacement vectors, $v_j - v_i$, is not contained in the no-fit polygon U_{ij}, that is

$$v_j - v_i = \begin{pmatrix} x_j \\ y_j \end{pmatrix} - \begin{pmatrix} x_i \\ y_i \end{pmatrix} \notin U_{ij} \qquad \Longleftrightarrow \qquad v_j - v_i \;\in \overline{U}_{ij}, \qquad \forall\, i,j \in \mathcal{P} :\; i < j \quad (1)$$

where $\overline{U}_{ij}$ is the complement of U_{ij}, hence it is highly non convex. As we need to deal with polyhedral (convex) sets in order to express conditions (1) through linear constraints, we subdivide $\overline{U}_{ij}$ into a collection of convex polyhedral components, assigning a component $\overline{U}^k_{ij}$ for each convex[1] edge, and a component $\overline{U}^k_{ij}$ for each set of consecutive concave[2] edges, as illustrated in Figure 3.

In this way we define a partition of $\overline{U}_{ij}$ into a set of m_{ij} (say) disjoint[3] polyhedra $\overline{U}^k_{ij}$, that we call *slices*, which satisfy:

$$\overline{U}^h_{ij} \cap \overline{U}^k_{ij} \;=\; \emptyset \quad \forall\, h \neq k, \qquad\qquad \bigcup_{k=1}^{m_{ij}} \overline{U}^k_{ij} = \overline{U}_{ij}$$

Figure 3 shows a possible partition of $\overline{U}_{ij}$ into slices. In this example, all components but two are built around a single convex edge, while $\overline{U}^3_{ij}$ and $\overline{U}^4_{ij}$ are built around a pair of concave edges each. Every slice $\overline{U}^k_{ij}$ is a (2-dimensional) polyhedron, hence it can be defined by linear inequalities of the form:

$$\alpha(x_j - x_i) + \beta(y_j - y_i) \leq \gamma$$

[1]An edge is called *convex* if its supporting line does not intersect the interior of the polygon.
[2]An edge is called *concave* if its supporting line intersects the interior of the polygon.
[3]We allow disjoint sets to overlap on a 1-dimensional line

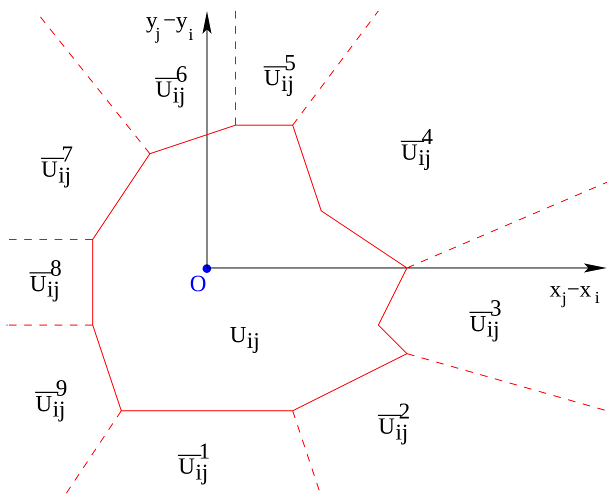


Figure 3: Partition of $\overline{U}_{ij}$ into polyhedral "slices".

where α, β and γ are the coefficients of the line defining a facet of the slice. The whole slice can then be represented in a compact matrix form as

$$\overline{U}_{ij}^{k} = \{u \in \mathbb{R}^2 : \; A_{ij}^k \cdot u \leq b_{ij}^k\} \tag{2}$$

where matrices (A_{ij}^k, b_{ij}^k) are easily computed once the slices $\overline{U}_{ij}^{k}$ have been defined.

Let us introduce the following decision variables:

- $v_i = (x_i, y_i)$: the displacement coordinates of the reference point of each piece $i \in \mathcal{P}$;
- *length*: the rightmost used point of the marker region;
- z_{ij}^k: a binary variable defining the slice k of $\overline{U}_{ij}$ that is active, i.e.,

$$z_{ij}^k = \begin{cases} 1 & \text{if } \; v_j - v_i \in \overline{U}_{ij}^{k} \\ 0 & \text{otherwise} \end{cases} \qquad \forall\, i,j \in \mathcal{P} : \; i < j, \; k = 1 \ldots m_{ij}$$

Let ε and M be a very small and a very large positive value, respectively. Also, let $e = (1, \cdots, 1)$ be the vector with all entries equal to 1. A first formulation for the nesting problem reads:

$$
\begin{aligned}
\min \quad & length + \varepsilon \sum_{i \in \mathcal{P}} (x_i + y_i) && && (3)\\
\text{s. t.} \quad & x_i + right_i \le length && \forall\, i \in \mathcal{P} && (4)\\
& left_i \le x_i && \forall\, i \in \mathcal{P} && (5)\\
& bottom_i \le y_i \le maxY - top_i && \forall\, i \in \mathcal{P} && (6)\\
& A^k_{ij}(v_j - v_i) \le b^k_{ij} + M(1 - z^k_{ij})e && && (7)\\
& && \forall\, i,j \in \mathcal{P}:\ i<j,\ k = 1 \dots m_{ij} && \\
& \sum_{k=1}^{m_{ij}} z^k_{ij} = 1 && \forall\, i,j \in \mathcal{P}:\ i<j && (8)\\
& z^k_{ij} \in \{0,1\} && \forall\, i,j \in \mathcal{P}:\ i<j,\ k = 1 \dots m_{ij} && (9)
\end{aligned}
$$

The main goal of the model is to maximize the efficiency, hence to minimize the total length. In the objective function (3) a second goal is also considered: keeping all pieces together as much as possible, by compacting them toward the origin of the marker region. The second term of the objective function has exactly this meaning, i.e., minimizing the coordinate of the reference point of each figure without affecting the main objective (length minimization). This term has been added for “aesthetic" reasons, in order to avoid having pieces spread around the region, hence parameter ϵ is chosen small enough so that the second term does not affect the real efficiency related to total length. The objective function does not take into account more sophisticated goals such as wasting, dimension of created holes and their usability, etc. In fact, calculating this type of information is quite complex from a geometrical point of view and cannot be included easily into a linear objective function.

Constraints (4) define the value of variable *length*, while constraints (5) and (6) are simple bounds on the feasible values for variables x_i and y_i. Constraints (7) use the compact slice description (2): depending on the active slice of each pair of pieces (identified by the z-variable set to 1), constraints (7) force the pieces not to overlap. Indeed, these constraints enforce $v_j - v_i \in \overline{U}^k_{ij}$ whenever $z^k_{ij} = 1$, otherwise the big-M term deactivates the whole set of constraints. Finally, constraints (8) assure that only one slice can be active for each pair of pieces, whereas constraints (9) force variables z^k_{ij} to assume only binary values.

It is worth noting that the number of constraints (7) grows rapidly with the number (and shape complexity) of the input pieces. Indeed, there are at least 3 facets for each slice, and the number of slices can grow linearly with the number of edges of the corresponding no-fit polygon. The relation between the number of edges of the no-fit polygon and the number of edges of the relative pieces depends on the type of polygons and can be as large as $O(r^2s^2)$, when both polygons are non-convex (where r and s are the number of edges of the two polygons respectively). Since each piece can interact with any other piece, the number of such no-fit polygons is quadratic in the number of pieces.

We observe here that the difficulty of the above MIP model essentially depends on the disjunctive constraints (7), imposing the choice of a suitable slice for each pair of potentially overlapping pieces: once this choice has been done (in a consistent and feasible way), and the corresponding z_{ij}^k have been set accordingly, the problem becomes an easily-solvable LP. Disjunctive constraints arise in several very hard optimization problems, including scheduling [32] and rectangular packing problems—which are notoriously very hard to solve through MIP techniques. In fact, the main constraint in scheduling applications (avoid time overlap between different jobs on the same machine) is just a 1-dimensional version of the non-overlapping condition in nesting problems. Similarly, MIP models for rectangular 2-dimensional packing problems (or even for the 3-dimensional ones, as in Padberg [31]) are just a simplified version of our nesting model, where the no-fit polygons associated with the input rectangular pieces are just rectangles, so 4 rectangular slices always suffice to partition their complement.

These considerations lead to the conclusion that MIP model (3)-(9) inherits the difficulty of known MIP models for scheduling and rectangular packing problems, hence it is likely to be extremely hard to solve—as confirmed by our computational analysis. Nevertheless, one can try to improve the MIP formulation in the attempt of pushing further its range of applicability, as is done in the following subsections.

3.1 Lifting the constraint coefficients

In the basic MIP model, the binary variable z_{ij}^k is used to activate exactly one of the slices for each $\overline{U}_{ij}$. When we try to solve the model using a standard enumerative technique, the integer (binary in this case) requirement is relaxed and the corresponding LP relaxation is solved. In this case, for each pair i, j we are likely to have more than one z_{ij}^k variable with positive values (while their sum is still equal to 1), thus admitting fractional solutions which are not feasible in the original (integer) formulation. A geometrical interpretation of this fact is that, allowing for fractional values of z_{ij}^k, the relaxed model considers to be feasible for $v_j - v_i$ a very large region covering in part (or even at all) the no-fit polygon U_{ij}. This means that the pieces can be placed one on top of the other in the LP relaxation, with essentially no limit on the degree of their overlap.

This behavior is largely due to the presence of the big-M terms in constraints (7), that tend to become inactive for any (even very small) positive value of the corresponding z_{ij}^k variable. For example, the choice $z_{ij}^1 = z_{ij}^2 = \frac{1}{2}$ in (8), makes almost all $v_j - v_i$ feasible, including the total overlap position $v_j = v_i$, since $M >> \max_{k=1\ldots m_{ij}}\{|b_{ij}^k|\}$.

In order to mitigate the negative effect of the big-M terms, we applied a *lifting* technique [28] on the constraint coefficients. This methodology consists of replacing the big-M terms by a series of smaller values that guarantee that the constraints remain valid for the solutions of the integer models (but hopefully not for several solutions of the relaxation).

Let us concentrate on a single slice $\overline{U}_{ij}^k$. We have t_{ij}^k facets f defining this region, each defined by a constraint of the form

$$\alpha_{ij}^{kf}(x_j - x_i) + \beta_{ij}^{kf}(y_j - y_i) \leq \gamma_{ij}^{kf} + M(1 - z_{ij}^k) \qquad \forall\, f = 1 \ldots t_{ij}^k$$

We first replace the big-M term with a new set of terms, one for each binary variable related to the pair of pieces i and j, and write

$$\alpha_{ij}^{kf}(x_j - x_i) + \beta_{ij}^{kf}(y_j - y_i) \leq \gamma_{ij}^{kf} + \sum_{h=1}^{m_{ij}} \theta_{ij}^{kfh}\, z_{ij}^h$$

By exploiting the fact that $\sum_{h=1}^{m_{ij}} z_{ij}^h = 1$, we replace the constant term γ_{ij}^{kf} by $\gamma_{ij}^{kf}\sum_{h=1}^{m_{ij}} z_{ij}^h$ and obtain the equivalent form

$$\alpha_{ij}^{kf}(x_j - x_i) + \beta_{ij}^{kf}(y_j - y_i) \leq \sum_{h=1}^{m_{ij}} \delta_{ij}^{kfh}\, z_{ij}^h$$

where $\delta_{ij}^{kfh} = \gamma_{ij}^{kf} + \theta_{ij}^{kfh}$. As already said, when slice $\overline{U}_{ij}^k$ is not active its binary variable z_{ij}^k is set to 0, and some other variable z_{ij}^h will be set to 1. Then the computation of each lifting coefficient δ_{ij}^{kfh} amounts to finding the maximum value of the left-hand side when only one variable z_{ij}^h has value 1. In other words, the lifting coefficients δ_{ij}^{kfh} can be computed as

$$\delta_{ij}^{kfh} := \max_{(v_j - v_i)\,\in\,\overline{U}_{ij}^h \cap B} \alpha_{ij}^{kf}(x_j - x_i) + \beta_{ij}^{kf}(y_j - y_i)$$

where B is a sufficiently large bounding box for the feasible displacement vector $v_j - v_i$, e.g., a rectangle with height $2 \cdot maxY$ and length $2 \cdot maxX$ (where $maxX$ is an upper limit on the marker length).

An important observation is that the above maximization problem can be solved by simply evaluating the function to be maximized on the *vertices* of the closed region

$\overline{U}_{ij}^{\,h} \cap B$. Figure 4 shows the vertices of slice $\overline{U}_{ij}^{\,h}$ that have to be considered when evaluating the lifting coefficient δ_{ij}^{kfh} of variable z_{ij}^{h} with respect to a generic facet f of $\overline{U}_{ij}^{\,k}$ (this facet is represented in dashed bold line in the figure).

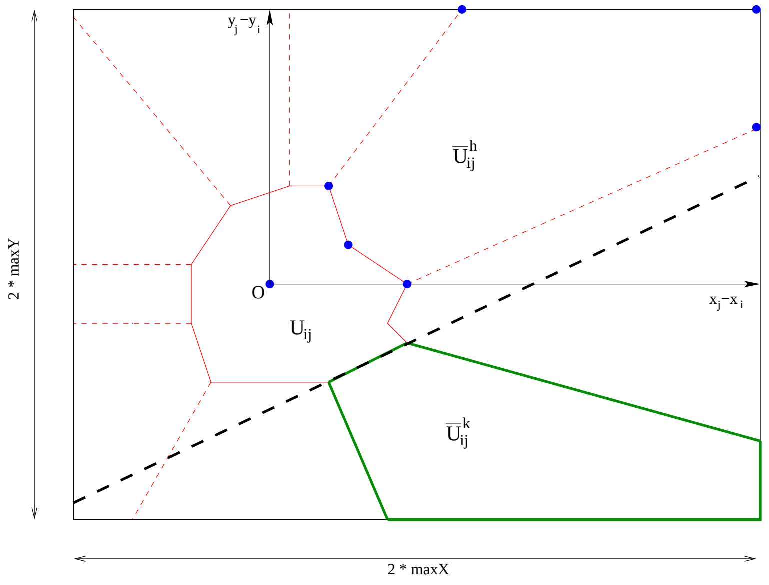


Figure 4: Computation of the lifting coefficients.

The enhanced model can now be rewritten as follows.

$$\min \quad length + \varepsilon \sum_{i\in\mathcal{P}} (x_i + y_i) \tag{10}$$

$$\text{s. t.} \quad x_i + right_i \le length \qquad \forall\, i \in \mathcal{P} \tag{11}$$

$$left_i \le x_i \qquad \forall\, i \in \mathcal{P} \tag{12}$$

$$bottom_i \le y_i \le maxY - top_i \qquad \forall\, i \in \mathcal{P} \tag{13}$$

$$\alpha_{ij}^{kf}(x_j - x_i) + \beta_{ij}^{kf}(y_j - y_i) \le \sum_{h=1}^{m_{ij}} \delta_{ij}^{kfh}\, z_{ij}^{h} \tag{14}$$

$$\forall\, i,j \in \mathcal{P}:\; i<j,\; k = 1\ldots m_{ij}\; f = 1\ldots t_{ij}^{k}$$

$$\sum_{k=1}^{m_{ij}} z_{ij}^{k} = 1 \qquad \forall\, i,j \in \mathcal{P}:\; i<j \tag{15}$$

$$z_{ij}^{k} \in \{0,1\} \qquad \forall\, i,j \in \mathcal{P}:\; i<j,\; k = 1\ldots m_{ij} \tag{16}$$

3.2 Guiding the search tree

During the execution of an enumerative technique for solving a MIP model, some binary z_{ij}^k are forced to 1 by the branching mechanism. The meaning of setting $z_{ij}^k = 1$ at a certain node of the branch-decision tree is to impose that pieces i and j assume a certain non-overlapping relative position, hence these two pieces will never overlap in the solutions of the LP relaxations obtained in the descendent nodes. However, fixing in a similar way the relative position of 3 (or more) pieces can easily lead to inconsistent subproblems.

Let us make an illustrative example where 3 out of the $n \geq 4$ pieces are the rectangles A, B and C shown in Figure 5.

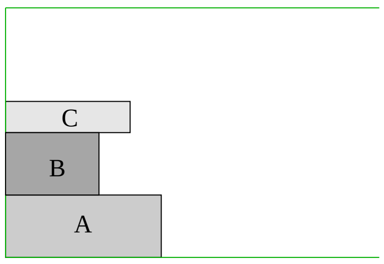


Figure 5: The relative position of three rectangular pieces A, B, and C.

For simplicity of notation, let us say that z_{XY}^{up} represents the binary variable related to the feasible region having piece Y above (*up*) X. Suppose we run an enumerative algorithm to get to a branching node corresponding to the situation depicted in Figure 5: we first fix (by means of branching) $z_{AB}^{up} = 1$, meaning that we want piece B above A, and then $z_{BC}^{up} = 1$, meaning that we want piece C above B. At this point, the relation between pieces A and C is apparently set, as the only logically-consistent choice is to fix $z_{AC}^{up} = 1$. However, for the relaxed LP model the choice $z_{AC}^{up} < 1$ is also feasible, meaning that the solution of the LP relaxation after fixing $z_{AB}^{up} = z_{BC}^{up} = 1$ can still have an overlap between A and C. Therefore, before fixing other variables it is worth fixing explicitly $z_{AC}^{up} = 1$ in order to be sure that the 3 pieces A, B and C do not overlap.

As the previous example suggests, a clever branching strategy should try to fix *all* the relative positions between a (possibly small) subset of pieces, rather than fixing few relative positions for a larger set of pieces. In other words, the branching strategy should favor the definition of "cliques" of consistent fixings, thus ensuring that a certain subset of pieces do not overlap each other. Following this idea, we implemented a branching scheme that guarantees that any new piece involved in a branching choice is positioned in a way to ensure feasibility with respect to the other pieces already subject to branching. To be specific, we tend to branch so as to completely determine first the relative positions of 2 pieces (say A and B), then that of 3 pieces (A, B and, say, C), of 4 pieces (A, B, C and, say, D), and so on. To this end, we exploit the so-called "branching priority" mechanism available in most commercial MIP solvers: the user can specify a branching priority for each variable, variables with higher priority being branched first. In our implementation, branching priorities are assigned to variables by using the following simple procedure.

```
$\overline{\mathcal{P}} = \mathcal{P}$ (the whole set of pieces);
$\psi$ = number of $z$ variables in the MIP model;
$\mathcal{S} = \emptyset$ ;
while $\overline{\mathcal{P}} \neq \emptyset$ do
    select a piece $p \in \overline{\mathcal{P}}$ ;
    for all $q \in \mathcal{S}$ do
        for $k = 1, \cdots, m_{pq}$ do
            assign priority $\psi$ to variable $z^k_{pq}$ ;
            $\psi = \psi - 1$
        end do
    end do
    $\mathcal{S} = \mathcal{S} \cup \{p\}$ ;
    $\overline{\mathcal{P}} = \overline{\mathcal{P}} \setminus \{p\}$
end do
```

The algorithm starts with a "yet unsettled" set of pieces $\overline{\mathcal{P}}$, an empty clique set $\mathcal{S}$ and the highest value of a priority variable ψ. It then takes into account a new piece p to be added to the clique set, and for each other piece q already in the clique it assigns a decreasing branching priority ψ to every relative position variable z^k_{pq}. Once the piece is added to the clique set, it is also removed from the original piece set $\overline{\mathcal{P}}$.

The above scheme is intended to be just a hint to the solver to guide the exploration of the search tree, in the attempt of avoiding the visit of subtrees that are infeasible because of inconsistent variable fixings that could have been detected at higher levels in the tree.

3.3 Computational experiments

We carried some experiments in order to evaluate the effect of our improvements (in particular, the priority-guided branching) to the basic MIP model. To this end we generated some *broken glasses* instances, where a square region is broken at random into n polygonal pieces; see Figure 6 for an illustration. This kind of instances is important because one knows in advance the value of the optimal solution, which is the length of the entire glass. In addition, the resulting MIPs turn out to be very hard to solve even by using state-of-the-art general-purpose MIP solvers.

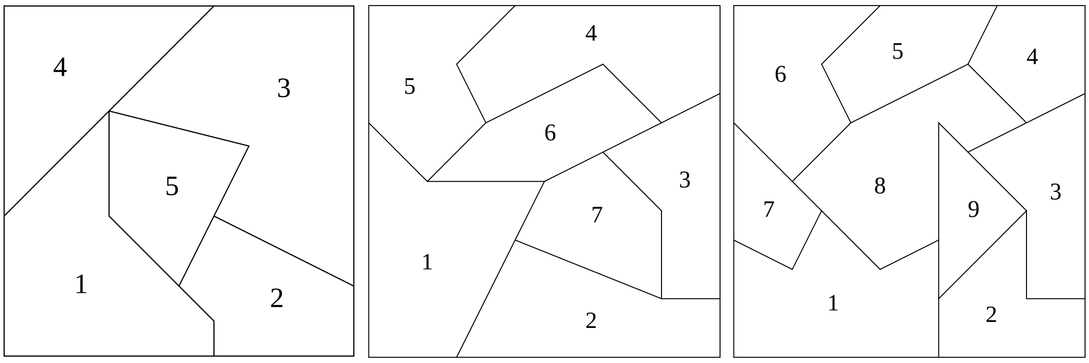


Figure 6: Three broken glass instances.

The outcome of our computational experience was quite discouraging: even by using the improved models, only very small instances with less than 10 pieces could be solved to proven optimality within reasonable computing time, while for bigger instances the MIP solver had to face a large gap between the heuristic solution and the computed lower bounds. The model improvement related to coefficient lifting had a limited practical impact on the overall performance. As to the use of branching priorities to guide the construction of the search tree, it helped the solver significantly in that some small instances could be solved to optimality only by using this strategy. It appears clear, however, that the MIP approach (as described) is unlikely to be useful for solving to proven optimality nesting instances of practical size.

As an example, we report in Table 1 the outcome of our experiments on the three small instances depicted in Figure 6. Computational results refer to runs on a PC AMD Athlon 1.2 GHz with 512 Mbyte RAM, using the commercial MIP solver ILOG-Cplex 7.0. For every instance, the table gives: the name of the instance; the number of pieces involved; the number of integer (binary) variables; the flag for activating the priority branching rule (coefficient lifting being always active); the total number of nodes in the branch and bound tree; the time to solve the model (in CPU seconds); and the gap between the best solution found and the best lower bound in case the solver reached the maximum number of branching nodes allowed (100,000).

INSTANCE	PIECES	INT. VAR.	PRIORITY	NODES	TIME	GAP
Glass1	5	73	no	470	0.26”	0%
			yes	111	0.11”	0%
Glass2	7	173	no	100,000	97.40”	32.08%
			yes	11,414	13.29”	0%
Glass3	9	302	no	100,000	157.76”	59.82%
			yes	100,000	203.48”	58.70%

Table 1: Computational results for the broken glass instances of Figure 6.

4 The Multiple Containment Problem

The bad computational performance described in the previous section gave us motivation to use MIP techniques to address not the nesting problem itself, but a simplified yet relevant subproblem arising in the design of clever nesting heuristics.

The usual way to build a layout is to place the “big pieces” [4] first and then to insert the remaining ones in the holes left by the big pieces. This is a typical strategy followed by human experts, and is also the approach used by many automatic algorithms. In this section we deal with the second problem, called *multiple containment problem*. More formally, we are given a set $\mathcal{P}$ of n “small” pieces called *trims*, and a set $\mathcal{H}$ of m irregular polygons called *holes*, and we want to find the best assignment of pieces into holes along with their displacement vectors, such that the maximum number (or maximum total area) of pieces is placed and the maximum hole area is used.[5] This problem is NP-hard, as it coincides with the nesting problem in case we have just one (rectangular) hole. Note however that in this problem we are not requested to place all trims, nor to use all holes, but to choose the holes and pieces that maximize the objective function. Similar containment problems have been recently addressed in [9, 21].

A first heuristic approach to the multiple containment problem that we use for benchmark purposes is to use the following greedy strategy. Take a trim at a time and scan all the available holes starting from the smallest one; once a hole that fits the trim is found, place the trim using a given placement policy (we used a bottom-left strategy). Once a new piece is inserted, calculate the newly created holes, insert them in an ordered list of holes, and select the next trim to be inserted. The greedy strategy has two evident drawbacks: the first is that any placement policy (lower left, upper left, etc...) can

[4]The threshold to differentiate between small and big pieces can be defined by using a coefficient α of the pieces average area.

[5]The *used hole area* is the region of a hole covered by trims.

compromise the free space for other trims in that hole; the second one is that assigning a trim to a certain hole in a greedy fashion, often leads to suboptimal allocations. For example, suppose we only consider the area of trims and holes, without any geometrical consideration. We are given two holes, h_1 and h_2, the first with area 90 and the second with area 95, and a set of 4 trims p_1, p_2, p_3, p_4 with area 85, 50, 40 and 10, respectively. A greedy strategy would place the biggest trim p_1 in the smallest hole h_1, then trims p_2 and p_3 in hole h_2; at this point trim p_4 cannot be placed anymore and we have a wasted area of 5 in hole h_1 and 5 in hole h_2. Conversely, a more clever assignment would place p_2 and p_3 in hole h_1, and then p_1 and p_4 in hole h_2; this would allow placing all trims with no wasted area at all.

The simplified problem in the above example is of course just a *knapsack problem*, hence it can be solved in a satisfactory way in practice (though the problem is NP-hard). On the other hand, if all the geometrical constraints were taken into account in a detailed way when modelling the multiple containment problem, we would face again a hard nesting problem. Therefore it is quite natural to investigate a simplified (yet useful) model, that exploits the fact that trims are often easy to place—provided that some rough geometrical considerations (e.g., on the trim and hole area) are taken into account.

4.1 Geometrical characterization of trims and holes

We observed that trims can very often be approximated in a sufficient way by their *bounding box*, defined as the smallest rectangle (with edges parallel to the x- and y-axis) containing a given piece. Replacing (in first approximation) the trims by their bounding boxes allows one to only check vertical and horizontal interaction among pieces, thus avoiding the nasty problems deriving from the non-convexity of the trims.

Figure 7 shows the set of pieces that composes a shirt: on the left we have the big pieces, on the right the small ones (trims) and their corresponding bounding boxes.

Another important consideration is that the set of holes comes from the geometrical difference of the marker region and the union of already placed "big pieces". Depending on the exact position of these pieces, this difference may consist of one or more polygonal components connected together by narrow strips (the light grey region in Figure 8). However, when the holes are too small or the strips are too narrow, they are not useful, since no trim can fit into them. Actually, what we are really interested in, is not the original shapes of the holes but their *usable* part, i.e., the area of the holes that can be occupied by the given trims.

Trims may have different sizes, therefore a hole that can contain a certain trim, may

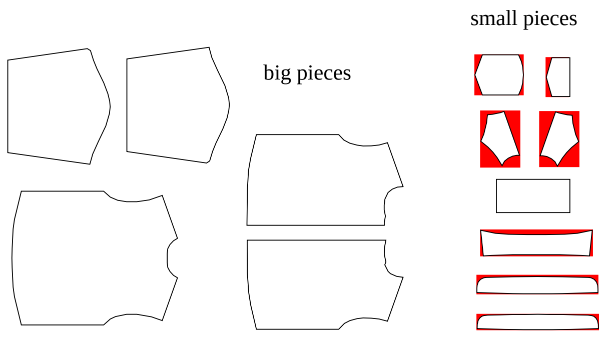


Figure 7: Big and small pieces of a shirt.

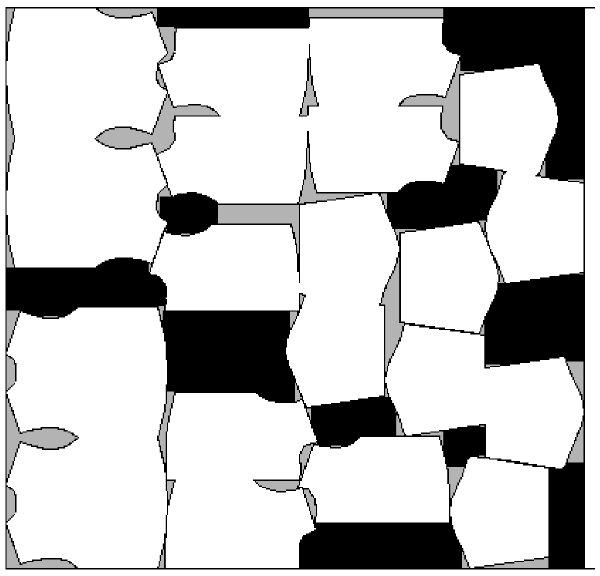


Figure 8: Original holes and their usable region.

not be useful for another one. In our simplification process we build a rectangle with dimensions equal to the minimum length and minimum height of all trim pieces, and call it the *minBox*.[6] Using the Minkowski difference operation illustrated in Section 2, we compute the "free movement" region of our minBox within the original holes (inner-fit region), thus defining a set of so-called *usable holes*. Figure 8 shows in light grey the original holes and, in black, the usable holes.

4.2 The placement grid inside usable holes

Once we have defined the usable holes, we divide them into *cells* by drawing an underlying rectangular grid in the following way. First we compute the average length and height of the trims (*aveLength* and *aveHeight*). Then, for each hole[7] we take the associated bounding box and map it into a rectangular grid where each cell has dimensions $cellLength_h \times cellHeigth_h$. Average trim length and height are just used to define the number of cells of a certain hole, but then for each hole, the specific $cellLength_h$ and $cellHeigth_h$ are calculated.

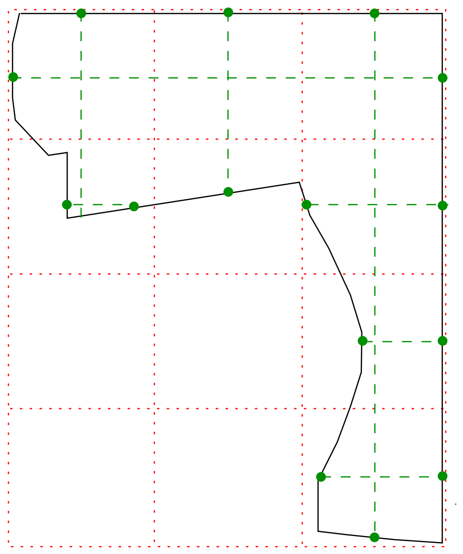


Figure 9: The grid on a specific hole and the corresponding rows/columns.

For every cell of the grid we draw a horizontal line (called *row*) through its center point and store the start and end points where the line intersects the figure; if the shape

[6]Actually, since such dimensions only consider the bounding box of each polygon, but not their real shape, we further reduce the minBox by a certain factor fixed heuristically to 0.8.

[7]From now on, by the term hole we mean a usable hole.

is non convex and the line intersects many times the hole border, we store only the first and last intersection points. Furthermore, we store the actual length of each row as the sum of lengths of all its internal segments. In a similar way, we draw vertical lines (called *columns*) through the cell centers and obtain the start and end points of the columns together with their actual length. It is assumed that each cell can accommodate, at most, one trim.

Figure 9 shows an example of a specific hole and its subdivision into cells: small circles indicate the starting and ending points of the internal segments defining the rows/columns; in this example, the third row (from the bottom) is the only one made by two distinct segments.

4.3 The simplified model

We now have all the information needed to start building our simplified model. Given the set $\mathcal{P}$ with the n trims and the set $\mathcal{H}$ with the m usable holes, we compute for each trim $p \in \mathcal{P}$

- $pieceArea_h$: the trim area;
- $pieceBoundingBoxArea_p$: the area of the trim bounding box

and for each hole $h \in \mathcal{H}$

- $holeArea_h$: the area of the hole;
- $origX_h, origY_h$: the coordinate of the reference point (lower left point) of the hole bounding box;
- $holeLength_h, holeHeight_h$: the hole bounding box dimensions;
- $cellLength_h, cellHeight_h$: the average length and height of each cell in the hole (different from hole to hole);
- R_h, C_h: the number of rows and columns in the hole, respectively.

Note that a cell in a hole is identified by the row and column coordinates of its center, say (r, c), where $1 \le r \le R_h$ and $1 \le c \le C_h$. Furthermore, for each row and column of every hole we have the following input data, defining the row/column starting point, ending point and length/height, respectively:

$$\left.\begin{array}{l} rowStart_r^h \\ rowEnd_r^h \\ rowLength_r^h \end{array}\right\} \quad \forall\, h \in \mathcal{H},\ r = 1 \dots R_h \qquad \left.\begin{array}{l} colStart_c^h \\ colEnd_c^h \\ colHeight_c^h \end{array}\right\} \quad \forall\, h \in \mathcal{H},\ c = 1 \dots C_h$$

To define our model we need the following variables:

- $U^h \in \{0,1\} \qquad \forall\, h \in \mathcal{H}$:
  a binary variable of value 1 if hole h is active (i.,e., it is used to place some trims), =0 otherwise;
- $Z_{rc}^{hp} \in \{0,1\} \qquad \forall\, h \in \mathcal{H},\ p \in \mathcal{P},\ r = 1 \dots R_h,\ c = 1 \dots C_h$:
  a binary variable of value 1 if trim p is placed in cell (r,c) of hole h; =0 otherwise.
- $X_{rc}^h,\ Y_{rc}^h \geq 0 \qquad \forall\, h \in \mathcal{H},\ r = 1 \dots R_h,\ c = 1 \dots C_h$:
  real coordinates defining the actual position of the lower-left point of the bounding box of the trim placed in cell (r,c) of hole h (if any);

We can now formulate our simplified model for the multiple containment problem as follows:

$$\max \quad \sum_{h\in\mathcal{H}}\sum_{p\in\mathcal{P}}\sum_{r=1}^{R_h}\sum_{c=1}^{C_h}(pieceBoundingBoxArea_p \cdot Z_{rc}^{hp}) - \varepsilon \sum_{h\in\mathcal{H}}\sum_{r=1}^{R_h}\sum_{c=1}^{C_h}(X_{rc}^h + Y_{rc}^h) \tag{17}$$

$$\text{s. t.} \quad \sum_{p\in\mathcal{P}} Z_{rc}^{hp} \le U^h \qquad \forall\, h\in\mathcal{H},\ r=1\dots R_h,\ c=1\dots C_h \tag{18}$$

$$\sum_{p\in\mathcal{P}} pieceArea_p \sum_{r=1}^{R_h}\sum_{c=1}^{C_h} Z_{rc}^{hp} \le holeArea_h \qquad \forall\, h\in\mathcal{H} \tag{19}$$

$$X_{rc}^h + \sum_{p\in\mathcal{P}} length_p\, Z_{rc}^{hp} \le X_{r,\,c+1} \qquad \forall\, h\in\mathcal{H},\ r=1\dots R_h,\ c=1\dots C_h-1 \tag{20}$$

$$X_{rc}^h + \sum_{p\in\mathcal{P}} length_p\, Z_{rc}^{hp} \le rowEnd_r^h \qquad \forall\, h\in\mathcal{H},\ r=1\dots R_h,\ c=C_h \tag{21}$$

$$\sum_{p\in\mathcal{P}} length_p \sum_{c=1}^{C_h} Z_{rc}^{hp} \le rowLength_r^h \qquad \forall\, h\in\mathcal{H},\ r=1\dots R_h \tag{22}$$

$$Y_{rc}^h + \sum_{p\in\mathcal{P}} Height_p\, Z_{rc}^{hp} \le Y_{r+1,\,c} \qquad \forall\, h\in\mathcal{H},\ r=1\dots R_h-1,\ c=1\dots C_h \tag{23}$$

$$Y_{rc}^h + \sum_{p\in\mathcal{P}} Height_p\, Z_{rc}^{hp} \le colEnd_c^h \qquad \forall\, h\in\mathcal{H},\ r=R_h,\ c=1\dots C_h \tag{24}$$

$$\sum_{p\in\mathcal{P}} Height_p \sum_{r=1}^{R_h} Z_{rc}^{hp} \le colHeight_c^h \qquad \forall\, h\in\mathcal{H},\ c=1\dots C_h \tag{25}$$

$$\max(rowStart_r^h,\ origX_h + (c-1)\cdot cellLength_h) \le X_{rc}^h \le \min(rowEnd_r^h,\ origX_h + c\cdot cellLength_h) \qquad \forall\, h\in\mathcal{H},\ r=1\dots R_h,\ c=1\dots C_h \tag{26}$$

$$\max(colStart_c^h,\ origY_h + (r-1)\cdot cellHeight_h) \le Y_{rc}^h \le \min(colEnd_c^h,\ origY_h + r\cdot cellHeight_h) \qquad \forall\, h\in\mathcal{H},\ r=1\dots R_h,\ c=1\dots C_h \tag{27}$$

$$U^h \in \{0,1\} \qquad \forall\, h\in\mathcal{H} \tag{28}$$

$$Z_{rc}^{hp} \in \{0,1\} \qquad \forall\, h\in\mathcal{H},\ p\in\mathcal{P},\ r=1\dots R_h,\ c=1\dots C_h \tag{29}$$

The objective function (17) is aimed at maximizing the total area of the bounding box of the allocated trims, while its the second term (the one multiplied by the small positive value ϵ) helps positioning the trims with a bottom-left policy within the cells.

Constraints (18) activate or deactivate all the selection variables of cell (r, c) in hole h, depending on the value of the hole activation variable U^h. Furthermore, since U^h is a binary variable, when active (equal to 1) this constraint imposes that only one trim can be assigned to a single cell.

Constraints (19) are typical *capacity knapsack constraints*, and impose that the total area of all the trims placed in a certain hole cannot exceed the area of the hole itself.

Constraints (20) impose that a trim in cell $(r, c + 1)$ must have x-coordinate greater or equal to the rightmost point of the trim positioned in cell (r, c), while constraints (21) impose a bound on the rightmost point of the trim positioned in the last column of each cell. These coupling constraints also ensure a consistent definition of the X and Z variables.

Constraints (22) are again knapsack constraints, but refer to the total length of each row: the sum of lengths of all the trims placed in all cells of a certain row cannot exceed the length of that row.

The next three sets of constraints, (23), (24) and (25), impose similar conditions on columns (instead of rows), and limit the y-coordinate of the trims in a column as well as the total height of each column.

Constraints (26) and (27) impose lower and upper bounds to the x- and y-coordinate of the trims assigned to each cell, respectively. Finally, constraints (28) and (29) impose integrality conditions.

4.4 Converting a solution of the simplified model into a feasible layout

Once the simplified model has been solved, we obtain the list of active holes, and the position of the bounding boxes of the chosen trims within these holes. This solution suffers however from all the geometrical simplifications that we applied in order to make the model solvable. First of all, we considered the bounding box of the trims, instead of their original shape. Furthermore, we only checked the trim intersections along the lines through the cell centers, but we did not consider any other possible intersection.

For all these reasons, when we actually replace the bounding boxes by the original trims, we might get a solution with many overlaps, both among trims and among trims and big pieces. Figure 10 shows the solution provided by our model for the multiple containment instance of Figure 8: usable holes are drawn in light grey with black borders, and their corresponding cell centers are marked by a cross; placed trims are drawn in dark gray. As the figure clearly shows, there are several overlapping trims, and many other trims extend outside the assigned hole.

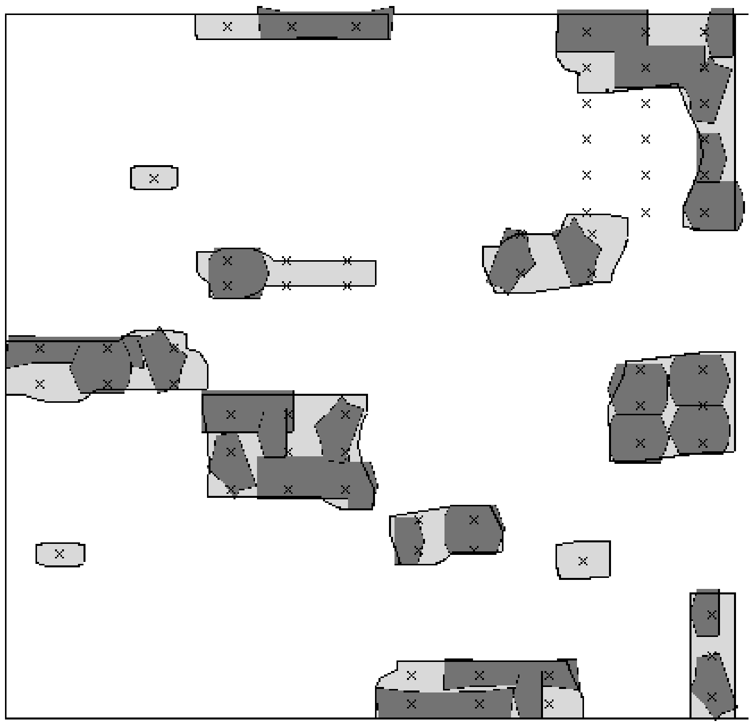


Figure 10: Trim pieces allocated by the simplified model.

The main reason for overlap is that no geometrical relations have been imposed among trims assigned to cells that differ in both row and column indexes. Figure 11 presents an example that illustrates this situation. As before, cell centers are marked with a cross, and are labelled according to their position. Piece A is assigned to cell (1, 1), piece B to cell (1, 2), and piece C to cell (2, 1). Pieces A and B lie on the same row (1) and satisfy all imposed constraints: both starting and ending points are within the hole borders, and the sum of their lengths does not exceed the hole length. Pieces A and C lie on the same column (1) and satisfy all constraints regarding starting and ending points as well as total height. Pieces B and C, instead, have an evident overlap, since they lay on different rows and columns and no constraint in our model prevents this kind of infeasibility.

We therefore need a post-processing procedure to try to clean-up the solution obtained by our model and remove the trim overlaps. This can be done using a *compaction/separation* algorithm [24], based on a linear version of the model that we analyzed

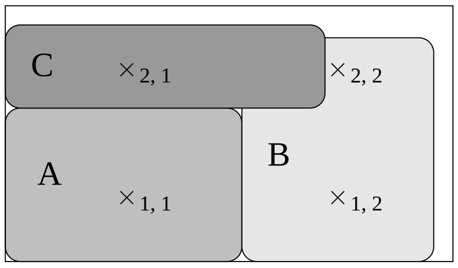
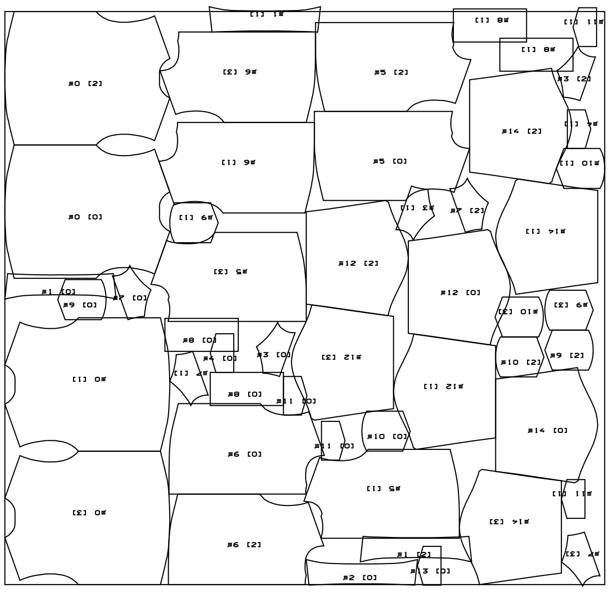


Figure 11: Weakness in the geometrical constraints of the model.

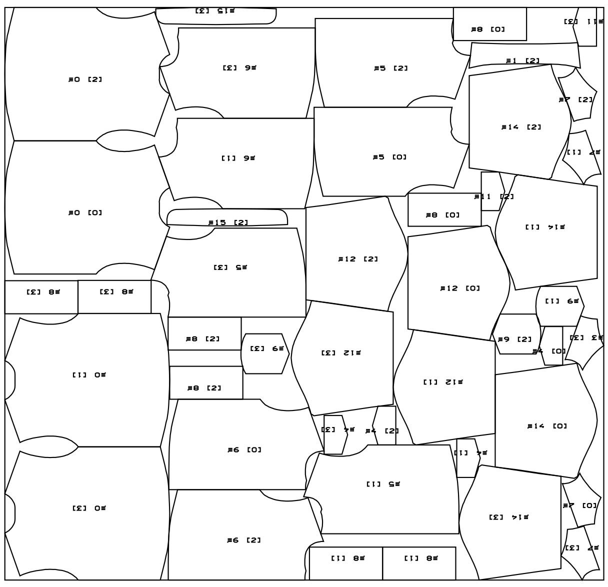
in Section 3. As all (large and small) pieces are already placed when post-processing is invoked, we know the relative positions of each pair of pieces. Thus we can fix the z variables of model (10)-(16) accordingly, by setting $z_{ij}^k = 1$ whenever $(v_j - v_i) \in \overline{U}_{ij}^k$. For overlapping pairs (i, j) of pieces, the variable z_{ij}^k to be set to 1 is instead chosen, heuristically, so as to activate the slice $\overline{U}_{ij}^k$ closest to $(v_j - v_i)$. Only the constraints relative to the z variables fixed to 1 are included in the model. The resulting model is just an LP, and therefore very easy to solve: if a feasible solution exists, then it corresponds to an overall feasible placement of the trims together with the big pieces. If the compaction/separation post-processing fails, however, a corrective action has to take place. A simple repairing heuristic is to find iteratively for the most overlapping trim piece, remove it from the solution, and try to apply again the compaction/separation model.

Figure 12 depicts the solution obtained by our model, and a corresponding solution obtained using a greedy strategy. Although the efficiency of our solution is higher than the greedy one, the figure shows many overlaps that the compaction/separation algorithm could not solve. Therefore some trims should be removed, resulting into a less satisfactory placement. A more clever strategy is to group the trims with similar dimensions into classes, and to solve our model separately for each class, in a hierarchical order. Indeed, we observed that when all trims have more or less the same shape and dimension, the geometrical problems discussed above are not likely to occur (or, if they occur, the degree of overlap is small and is usually repaired easily by our compaction/separation post-processing). If we apply this technique to the instance of the previous figure, we obtain the solution of Figure 13. This solution still presents some overlaps, but this time our compaction/separation procedure has no difficulty in removing it.

The overall solution approach we propose is illustrated in Figures 14 and 15. A first class of (relatively) large trims is first addressed. Figure 14 compares our final feasible solution (after the compaction/separation step) with the solution obtained by



Pieces: 50/76 Length: 1654.02 Eff.: 87.25%



Pieces: 45/76 Length: 1652.52 Eff.: 85.86%

Figure 12: Infeasible solution obtained by our model (top) and a corresponding feasible solution obtained by a greedy strategy.

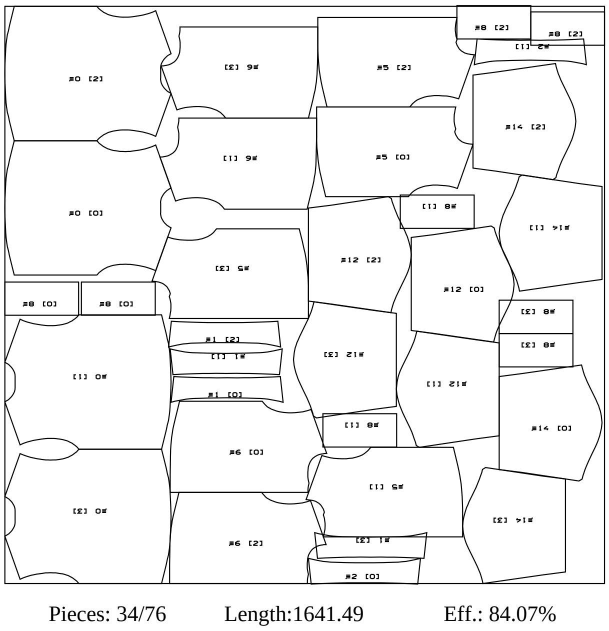


Figure 13: Solution obtained by our model for an homogeneous subclass of trims.

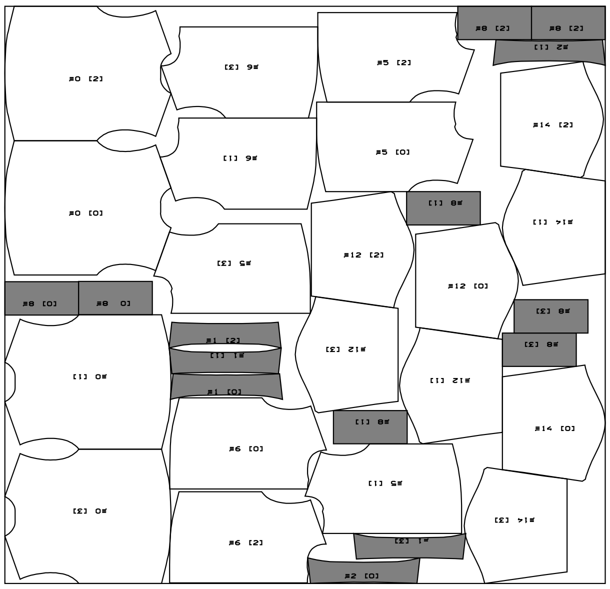
the greedy algorithm. Our solution placed 14 large-trims, compared to the 10 large-trims placed by the greedy algorithm. The efficiency of our solution is 83.97% compared to 81.54% of the greedy solution, for a total improvement of about 2.5%. The same process is then repeated on the remaining class of "smaller" trims. Figure 15 shows the corresponding placements obtained by both algorithms. The solution proposed by our model (and successively corrected by the compaction/separation routine) placed 10 trims and achieved an efficiency of 86.13%, while the greedy algorithm was able to place only 8 trims, for a total efficiency of 85.67%.

4.5 Computational results

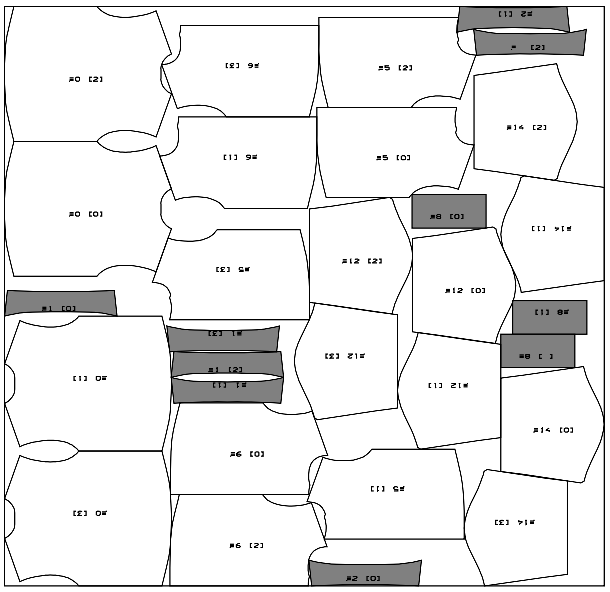
We tested our multiple containment heuristic on a set of instances taken from the garment industry [8] and compared the resulting solutions to those obtained by a greedy algorithm. The corresponding computational results are reported in Table 2, where our method goes under the label *MIP-place*. The running time of our algorithm (not reported in the table) goes from less than 1 second to a few seconds,[9] and is definitively comparable to the time needed by the greedy algorithm.

[8]available on request from the second author

[9]Using CPLEX 7.0 on a AMD Athlon 1.2 GHz with 512 Mbyte RAM



Pieces: 34/76 Length: 1643.53 Eff.: 83.97%



Pieces: 30/76 Length: 1634.55 Eff.: 81.54%

Figure 14: Feasible solution obtained by our model (top), and a corresponding feasible solution obtained by a greedy strategy (bottom), for a homogeneous class of "large" trims.

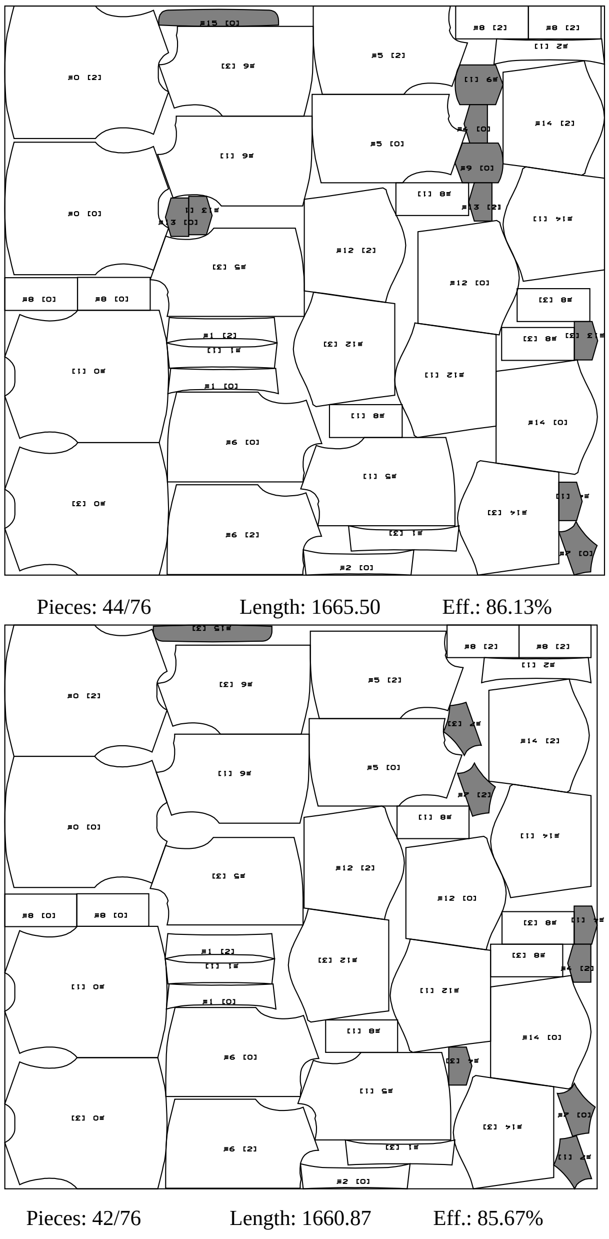


Figure 15: Feasible solution obtained by our model (top), and a corresponding feasible solution obtained by a greedy strategy (bottom), for the remaining class of “small” trims.

INSTANCE	PIECES	TRIMS	LENGTH	EFFICIENCY
82 - group 1				
MIP-place	34/76	14	1643.53	83.97%
greedy	30/76	10	1634.55	81.54%
82 - group 2				
MIP-place	44/76	10	1665.50	86.13%
greedy	42/76	8	1660.87	85.67%
101				
MIP-place	44/50	10	3840.28	82.12%
greedy	42/50	8	3838.27	81.57%
385				
smart	44/54	22	4697.05	83.74%
greedy	39/54	17	4671.81	83.58%

Table 2: Computational results for the multiple containment problem.

As the table shows, the new method compares very favorably with the greedy one, and increased the overall efficiency by 1-2% (an economically very relevant figure for this kind of instances). Some solutions are depicted in Figures 16-17

5 Conclusions

Several industrial problems involve placing objects into a container so that no two objects overlap each other, with the goal of either minimizing the size of the container or to find an optimal collection of objects that can be placed. We have addressed one of the most general (and difficult) problems of this type, namely the nesting problem, where the objects to be placed are represented by 2-dimensional closed polygons of any shape. The nesting problem is strongly NP-hard. Furthermore, its geometrical aspects make it really hard to solve in practice.

We have addressed a Mixed-Integer Programming (MIP) model for the nesting problem, with some improvements with respect to an earlier proposal [12, 23], and have analyzed it computationally. We have also introduced a new MIP model for a sub-problem arising in the construction of heuristic nesting solutions, called the multiple containment problem, and have shown its potentials in finding improved solutions.

The outcome of our research is that MIP techniques, though still not appropriate to solve the complexity of real-world nesting instances, can be very useful to address some

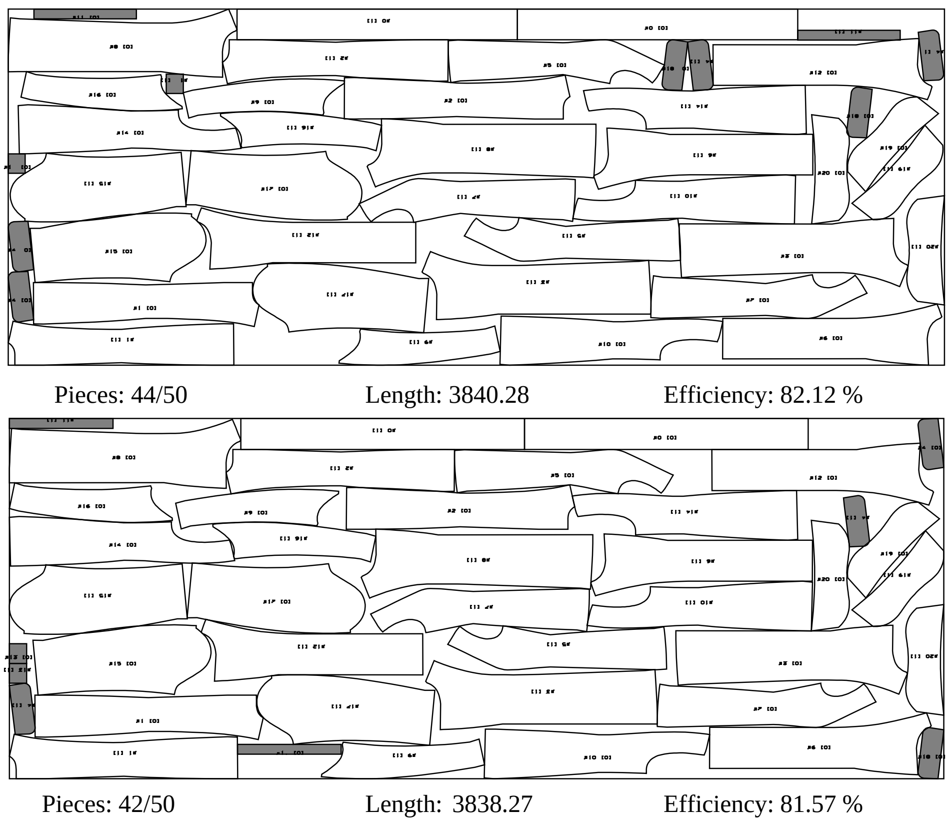
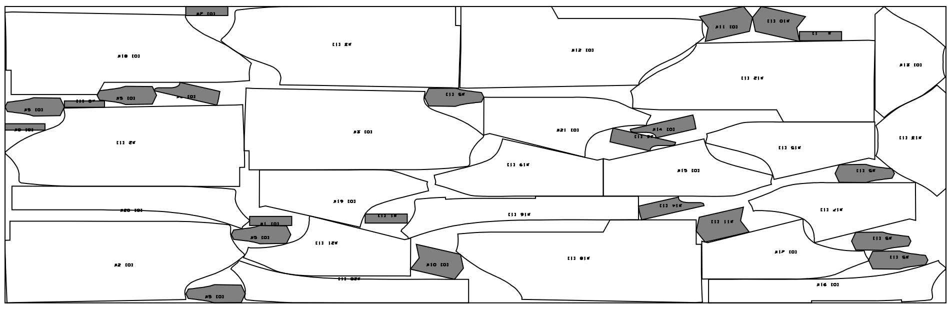


Figure 16: Feasible solution obtained by our model (top), and a corresponding feasible solution obtained by a greedy strategy (bottom) for problem instance 101.


Pieces: 44/54 Length: 4697.05 Efficiency: 83.74 %

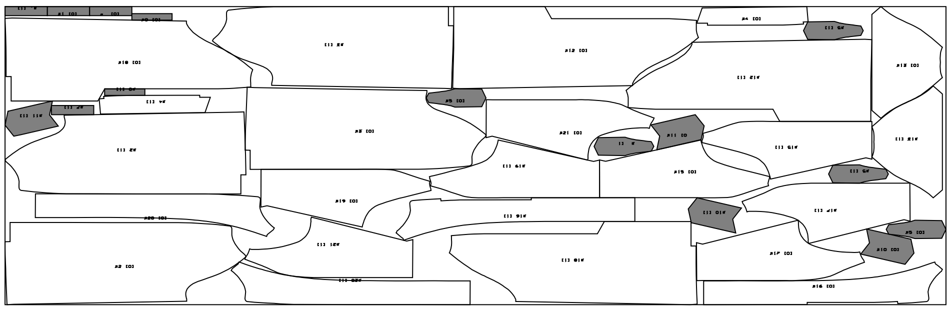

Pieces: 39/54 Length: 4671.81 Efficiency: 83.58 %

Figure 17: Feasible solution obtained by our model (top), and a corresponding feasible solution obtained by a greedy strategy (bottom) for problem instance 385.

simplified (but still NP-hard) subproblems arising in heuristic solution methods. We believe that this line of research deserves more attention in the future, and will hopefully lead to improved heuristic methods for hard nesting problems.

Acknowledgements

Work partially supported by M.I.U.R. and by C.N.R., Italy. Thanks are due to two anonymous referees for their helpful comments.

References

[1] M. Adamowicz, A. Albano, A two-stage solution of the cutting-stock problem, Information Processing 71 (1972) 1086-1091.

[2] A. Albano, G. Sapuppo, Optimal allocation of two dimensional irregular shapes using heuristic search methods, IEEE Transactions on Systems, Man and Cybernetics SMC-10 (5) (1980) 242-248.

[3] R.C. Art Jr., An approach to the two-dimensional irregular cutting stock problem, Technical Report 36.Y08, IBM Cambridge Centre (1966).

[4] J.A. Bennel, K.A. Dowsland, Hybridising tabu search with optimisation techniques for irregular stock-cutting, in: Extended Abstracts of MIC'99, Angra Dos Reis, Rio de Janeiro, Brazil, 1999.

[5] J. Blazewicz, P. Hawryluk, R. Walkowiak, Using tabu search approach for solving the two-dimensional irregular cutting problem in tabu search, in: F. Glover, M. Laguna, E. Taillard, D. de Werra (Eds.), Tabu Search - volume 41 of Annals of Operations Research, J.C. Baltzer AG. (1993).

[6] J. Blazewicz, A. Moret-Salvador, R. Walkowiak, Parallel tabu search approaches for two-dimensional cutting, Parallel Processing Letters 14/1 (2004), 23-32.

[7] E.K. Burke, R.S.R. Hellier, G. Kendall, G. Whitwell, A new bottom-left-fill heuristic algorithm for the two-dimensional irregular packing problem, Operations Research 54/3 (2006), 587-601.

[8] J. Chung, D. Scott, D.J. Hillman, An intelligent nesting system on 2-d highly irregular resources, Applications of Artificial Intelligence VIII (SPIE), 1293/1 (1990), 472-483.

[9] K.M. Daniels, Containment algorithms for nonconvex polygons with applications to layout, Ph.D. Thesis, Center for Research in Computing Technology, Harvard University, Cambridge, MA (1995).

[10] K.M. Daniels, Z. Li. and V.J. V. Milenkovic, Placement and compaction of nonconvex polygons for clothing manufacture, Proceedings of the 4th Canadian Conference on Computational Geometry (1992) 236-243.

[11] K.M. Daniels, V.J. Milenkovic, Limited gaps, Proceedings of the 6th Canadian Conference on Computational Geometry (1994) 225-230.

[12] K.K. Daniels, V.J. Milenkovic, Z. Li, Multiple containment methods, Technical Report 12-94, Center for Research in Computing Technology, Harvard University, Cambridge, MA (1994).

[13] K.M. Daniels, V.J. Milenkovic, Multiple translational containment: approximate and exact algorithms, Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms (1995) 205-214.

[14] K.A. Dowsland, W.B. Dowsland, Packing problems, European Journal of Operational Research 56 (1992) 2-14.

[15] K.A. Dowsland, W.B. Dowsland, Solution approaches to irregular nesting problems, European Journal of Operational Research 84 (1995) 506-521.

[16] K.A. Dowsland, W.B. Dowsland, J.A. Bennell, Jostling for position: local improvement for irregular cutting patterns, Journal of the Operational Research Society 49 (1998) 647-658.

[17] K.A. Dowsland, S. Vaid, W.B. Dowsland, An algorithm for polygon placement using a bottom-left strategy, European Journal of Operational Research 141 (2002) 371-381.

[18] J. Egeblad, B.K. Nielsen and A. Odgaard, Fast neighborhood search for two- and three-dimensional nesting problems, to appear in European Journal of Operational Research (2006).

[19] A.M. Gomes, J.F. Oliveira, A 2-exchange heuristic for nesting problems, European Journal of Operational Research 141 (2002) 359-370.

[20] A.M. Gomes, J.F. Oliveira, Solving Irregular Strip Packing problems by hybridising simulated annealing and linear programming, European Journal of Operational Research 171(3) (2006), 811-829.

[21] R.B. Grinde, K.M. Daniels, Exact and heuristic approaches for assignment in multiple-container packing, Boston College Technical Report BCCS-97-02 (1997).

[22] R. Heckmann, T. Lengauer, Computing closely matching lower and upper bounds on textile nesting problems, European Journal of Operational Research 108 (1998) 473-489.

[23] Z. Li, Compaction algorithms for non-convex polygons and their applications, PhD dissertation, Harvard University, Cambridge (1994).

[24] Z. Li, V.J. Milenkovic, Compaction and separation algorithms for non-convex polygons and their applications, European Journal of Operational Research 84 (1995) 539-561.

[25] I. Luzzi, Exact and heuristic methods for nesting problems, PhD dissertation, Dipartimento di Metodi Quantitativi, University of Padova (2003); available at `http://www.dei.unipd.it/~fisch/ricop/tesi/tesi_dottorato_Luzzi_2002.ps`

[26] A. Mahadevan, Optimization in computer-aided pattern packing, PhD Thesis, North Carolina State University, 1984.

[27] V.J. Milenkovic, K.M. Daniels, Translational polygon containment and minimal enclosure using geometric algorithms and mathematical programming, Technical Report 25-95, Center for Research in Computing Technology, Harvard University, Cambridge (1995).

[28] G.L. Nemhauser, L.A. Wolsey, *Integer and combinatorial optimization*, Wiley, Interscience (1988)

[29] J.F. Oliveira, J.S. Ferreira, Algorithms for nesting problems, in: R.V.V. Vidal (Ed.), Applied Simulated Annealing, Springer, Berlin, 1993, pp. 255-273.

[30] J.F. Oliveira, A.M. Gomes, J.S. Ferreira, TOPOS - A new constructive algorithm for nesting problems, OR Spektrum 22 (2000) 263-284.

[31] M. Padberg, Packing small boxes into a big box, Mathematical Methods of Operations Research 52 (2000), 1-21.

[32] M. Queyranne, A.S. Schulz, Polyhedral approaches to machine scheduling, Technical Report 408/1994, Technical University of Berlin, 1994.

[33] S. Umetani, M. Yagiura, T. Imamichi, S. Imahori, K. Nonobe, T. Ibaraki, A guided local search algorithm based on a fast neighborhood search for the irregular strip packing problem, Proceedings of International Symposium on Scheduling (ISS2006), 126-131, Tokyo, July 18-20, 2006.