UNIVERSITÁ DEGLI STUDI DI PADOVA

Dipartimento di Matematica Pura ed Applicata

Dottorato di Ricerca in Matematica Computazionale XV Ciclo

Exact and Heuristic Methods for Nesting Problems

Ivan Luzzi

Coordinatore: Ch.mo Prof. Renato Zanovello

Supervisore: Ch.mo Prof. Matteo Fischetti

AA. AA. 1999-2002

Padova, dicembre 2002

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Cutting and Packing problems

Many industrial problems involve placing objects into containers so that no two objects overlap each other. The general goal is to either minimize the size of the container or to find an optimal collection of objects that can be placed (either in terms of number or total area).

These problems are known by a variety of names, such as layout, packing, nesting, loading, placement, marker making, etc... A number of industries apply layout techniques when cutting parts from stock material. In apparel manufacturing, for instance, pattern pieces are arranged on cloth, and the goal is to find a non-overlapping arrangement which uses the least amount of cloth. In sheet metal layout, objects are cut from metal stock sheets; the goal here is typically to cut as many objects as possible from a given sheet. In shoe manufacturing instead, each hide has a different shape, and the goal is to cut as many objects from each hide as possible. In VLSI layout, rectangular modules are arranged on a chip in such a way as to best meet the two competing goals of minimizing the chip area and minimizing the length of interconnections between modules on the chip. Finally, in furniture layout an arrangement of furniture usually must even satisfy a set of aesthetic criteria.

When the 2-dimensional objects to be packed are non-rectangular the problem is called *irregular nesting* and the output produced by a packing algorithm is called a *layout*.
The material utilization, or *efficiency*[1], of a layout is the ratio of the area occupied by the pieces to the rectangular area of the used stock sheet. The *waste* of material is the unused area of the stock sheet within the used length, and is therefore equal to 1 minus the efficiency (depending on the stock material, on its shape and on the shape and dimension of the pieces to be cut, the waste of material in particular contexts ranges

[1] In the literature the efficiency is also referred as *yield*.

from 10-15% up to 40% or even 50%).

From now on we will focus our attention on the irregular nesting problem in the textile industry, also known as the *marker making problem*.

Currently, marker making is done by experienced human marker makers with the assistance of interactive CAD systems. A human marker maker needs about a year of training before he/she can generate markers of acceptable efficiency, and it takes between 30 to 45 minutes to generate a marker of good efficiency (estimated within 1-2 percent of the optimal).

Current automated marker making systems fall short of human performance by up to 5 percent in marker efficiency. Although the prospective of developing an automated system may not be very attractive from the pure efficiency point of view, it is convenient from other aspects: the time to find an acceptable solution can be much shorter if done by a specialized algorithm; computers run 24 hours a day 7 days a week and never get sick; there is no training time; furthermore this system could be used to find a solution that gives a good estimate of the real efficiency, for example to evaluate different garment combinations in a marker.

1.2 Problem Representation

In marker making, the garment pieces are represented by polygons[2]. The vertices of a polygon are the extreme points on the boundary of a pattern, and each pair of vertices is connected by an edge (a straight line segment). There are more vertices and shorter edges in regions of higher curvature, and fewer vertices and longer edges in regions of lower curvature.

As shown in figure 1.1, there is a local coordinate system attached to each piece. The origin of the local coordinate system is usually fixed at the center of the *bounding box* of the piece, defined as the smallest rectangle containing the piece with sides parallel to the coordinate axes. The coordinates of the vertices are specified in the local coordinate system. The position of piece in the marker is given with respect to a global coordinate system. The origin of the global coordinate system corresponds to the lower left corner of the rectangular sheet on which the pieces should be laid.

The placement of the pieces is restricted to a horizontal rectangular strip in the first orthant of the global coordinate system, and is called the *marker region*. The marker region is bounded from the left by the y axis and is open on the right. After the placement of a few pieces, the uncovered regions remaining in the marker are broken into a set of connected components called *holes*, and can possibly be used to fit some of the remaining pieces.

The rules of marker making sometimes permit a piece to change its orientation by

[2] In the sequel we will use the terms piece, figure and polygon interchangeably.

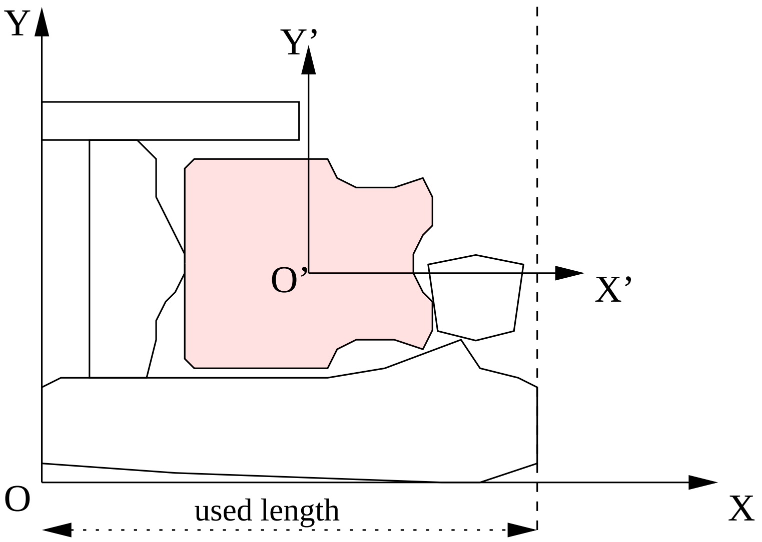


Figure 1.1: Global and local coordinate system in the layout.

rotating and/or flipping in the local coordinate system. A basic *rotation step* is specified for each piece. The amount is usually 45, 90 or 180 degrees. A piece can be rotated by a multiple of its rotation step about its local origin. In addition to multiples of its basic rotation amount, a piece can, in some special cases, also rotate by some small amount called *tilt*, which is usually less than 3 degrees but can be as large as 7 or 8 degrees. This is done to better fit pieces together and to get a tighter solution, but it can be forbidden in some applications due to grain or fabric material constraints.
In the sequel we only consider the basic case of 180 degrees rotation and no flip and tilt allowed, which is by far the most common rotation allowed in practical applications.

The irregular nesting problem is strongly NP-hard [GJ79], even in its simplified version where all the polygons are rectangles and can only translate. This can be proved by reduction from the NP-hard PARTITION problem, defined as follows: given a set S of integers, decide if S can be partitioned into two subsets $S1$ and $S2$ such that the sum of the elements of $S1$ equals the sum of the elements of $S2$.

We reduce PARTITION to NESTING by the following construction. For any instance $\{a_1, a_2, \ldots, a_n\}$ of PARTITION, we build rectangles of height 1 and length $a_i, (i = 1, .., n)$. We put the rectangles into a container of height 2. PARTITION has a solution if and only if the blocks can be compacted to the length $\sum a_i/2$.

1.3 Outline of the thesis

In chapter 2 the irregular nesting problem is introduced and analyzed in all its variants; a quick survey of the literature is given together with the main bibliographic references.

In chapter 3 a first approach to the problem is analyzed, where pieces are represented by bitmaps where a bit is 1 if it corresponds to a point in the interior of the figure, 0 otherwise. The overlap check is done via straightforward bitwise comparison. Different strategies are used to find an initial solution, and a tabu search technique has been used to explore neighbor solutions. This approach was easy to implement and produced good quality solutions. A main drawback is that the size of the bitmaps grows proportionally with the precision required in the piece representation, and the execution time increases steeply. To overcome this problem a different approach has been analyzed, where all figures are represented by polygons defined by the corresponding vertices. Chapter 4 deals with the computational geometry aspects involved in this approach: the concept of no-fit polygon is defined and its use to detect overlap among pieces is described.

In chapter 5 a MIP model for the irregular nesting problem is defined, in two different versions, and some computational results are presented. A basic version of this model was first introduced by Daniels et al. [DLM94]; we extended it by lifting some of the coefficients in the constraints, so as to improve the formulation.

The usual way to build a solution is to place the "big pieces" first and then insert the remaining ones in the holes left by the big pieces. However, choosing the pieces to be put in the holes is not an easy task, and most of the times a greedy strategy proves not adequate. In chapter 6 a new bi-dimensional knapsack problem is presented to model this task in a simplified way, and some interesting computational results are presented.

Finally, in appendix A we introduce a new standard specification to represent instances of the nesting problem. We converted all the instances found in the literature to this standard and provide a web site (NESTLIB) where this data can be downloaded by the scientific community to test different algorithms.

1.4 Main contributions

In this thesis we design, implement and test a new algorithm for the automatic nesting of irregular pieces using the tabu search methodology and a bitmap representation of the pieces. We introduce a new alternative bitmap representation and an efficient technique to check for overlap which allows to reduce the execution time by a factor 3. We investigate the behaviour of the algorithm with different tabu parameter, and apply the concepts of "intensification" and "diversification" of the search to this particular problem.

We define a MIP model for the irregular nesting problem based on the one proposed by Daniels et al. We enhance this model by applying a lifting technique to the constraint coefficient. Furthermore we implement a branching technique based on priorities to guide the visit of the search tree, and show that this methodology effectively improves the solution time.

Finally we define a new knapsack type model for the multiple containment problem, based on geometrical considerations. It is used to place small pieces in the gaps left by the previous placement of big pieces. We implement this model and test it on real world instances, showing the efficiency of its results.

Chapter 2

The irregular nesting problem

In this chapter we will make a quick survey on the irregular nesting problem: the field of application where it arises and the different approaches that were used to deal with it.

2.1 Application of the irregular nesting problem

One of the first industrial fields interested in the irregular cutting has been the ship-building industry. The problem involves determining a cutting pattern for a given set of parts, each of which consisting in a number of different pieces, from a series of metal stock sheets; the pieces from a single part must be kept together. Since more than one sheet is usually required, the way in which the different parts are combined together has a significant effect on the overall cutting problem; this problem is known as the *sequencing problem*. Arbel [Arb93] suggests a two phase method: first an integer programming model based on estimated packing efficiencies assigns pieces to single sheets, then the actual nesting of the pieces is performed.

A problem involved in the metal cutting process is related to the metal distortion due to the torch heat. For this reasons the same cut cannot be used for two adjacent pieces, but "bridges" or gaps are left between them in order to avoid movement of the pieces during the cutting process. Another aspect to be taken into account when dealing with metal, is the cutting time: depending on the type of torch (oxygen or laser), and on the type of metal and its width, cutting speed ranges from 100 mm per minute up to 7 m per minute. For this reason the objective function is a tradeoff between waste and cutting time. Usually more copies of the same piece have to be cut, and therefore patterns are clustered in parallel blocks in order to use multiple torches working in parallel and following the same cutting sequence. Tàvora [Tàv89] suggests an algorithm based on a clustered traveling salesman model to minimize the movement of the cutter between cutting operations. The choice between parallelizing the process or reducing the waste of material becomes crucial in some situations. For example, when many identical pieces with complex shape have to be cut, one can choose between parallel placement to improve the cutting time, or alternate placement to improve efficiency (see figure 2.1).

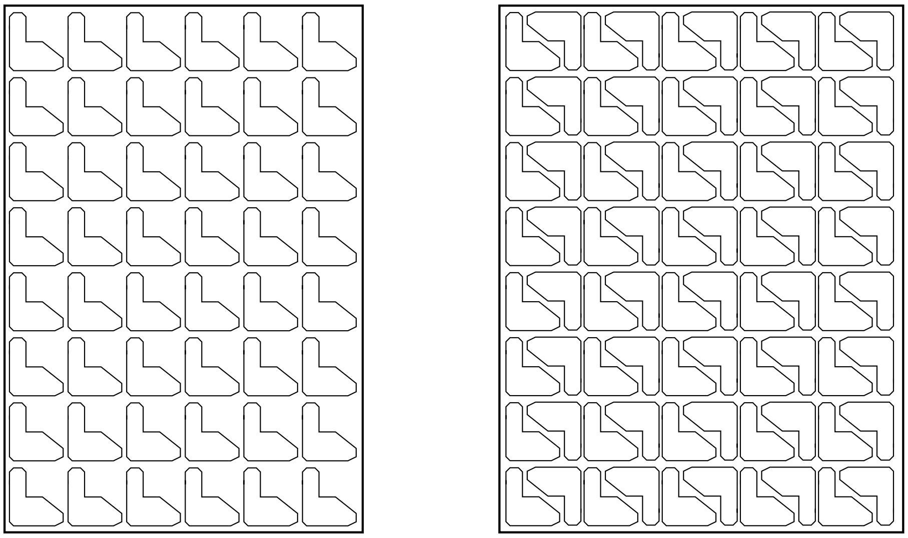


Figure 2.1: Comparison between parallel and alternate placement.

In the textile industry the marker layout problem is very different: here there is no need for bridges between pieces, but the grain and pattern of the fabric means that the orientation is usually fixed or at most a 180 degree rotation is allowed. Furthermore, if the fabric has a pattern, the placement of certain pieces have to respect special rules and must fit into a lattice. A typical marker consists of several copies of the same garment (different sizes can be included to improve efficiency) consisting of a few large pieces called *panels* (e.g. trouser legs), and some smaller ones called *trims* (e.g. pockets and waist bands).

Also in the furniture industry and the wood industry in general, orientation is a main concern, due to the grain of wood; furthermore, the presence of defects of any sort (knots, creeks, etc..) imposes to waste part of the material. Usually in this field, an human experts draws the patterns directly on the wood, avoiding the different defects, then a cutting machine connected to a digital camera recognizes the tracks and follows them on the cutting process.

The leather industry faces all the problems above, but also considers the quality of the different parts of a hide. Here each hide has its own shape, tone, and possibly defects. Certain parts of the final product have to be cut from high quality sections of the hide, while others don't have any special requirement. Furthermore, if the leather is not to be dyed, it's important to cut pieces which will be together in the finished product, from adjacent parts of the hide, in order to guarantee the same color.

In spite of these differences all these problems have the common requirement of finding a feasible layout of a given set of pieces on a stock sheet region, with the goal of minimizing the total waste of material.

There are three different types of approach to this problem:

- pieces are first grouped in smaller subsets and inserted, either singularly or in group, into a super set of ideal polygons with fixed shape and size; then the latter are nested on the stock sheet;

- pieces are considered one at the time and directly placed on the marker region;

- an initial placement is found, which could also allow some overlap among pieces, then iterative methods are used to improve the solution.

2.2 Polygonal approximation and tessellation methods

One of the first approaches to the irregular nesting problem, especially in the past, has been that of including one or more irregular pieces into a more regular shape, usually a rectangle, and then to use one of the bin packing algorithm available to place these simpler figures on the marker region. In the case of free rotation, one can include the figure in a rectangle with minimum area; Freeman and Shapira [FS75] first include the piece in a convex polygon, then examine all rectangles with base corresponding to each of the polygon edges until they find the rectangle with minimum area.

In practice, solutions of this type offer acceptable results only in the cases where the pieces are themselves almost rectangular, so that the waste within the enclosing rectangle is low. Adamowicz and Albano [AA76] impose a threshold on the amount of waste they are willing to accept in any enclosure: if a single piece enclosed in the rectangle exceeds this limit, it is combined with a copy of itself rotated by 180°. If neither of these two options stay within the waste threshold, an attempt is made to fill the remaining space with some of the other pieces.

An alternative to using rectangles, is to nest all pieces into identical polygons which can be used to tile the plane (*tessellation*). The most used plane-tiling polygons are: triangles, quadrilaterals, pentagons and hexagons. Dori and Ben-Bassat [DBB84] base their packing algorithm on tessellation. They first nest the required shapes into polygons with minimum wasted area, then enclose them into hexagons which are suitable for tiling the plane. They only considered convex pieces, although they state that non-convex shapes could be nested in a similar way; Karoupi and Loftus [KL91] have extended this method to deal with curved and non-convex polygons (see figure 2.2).

This type of approach gives reasonable quality solutions only if the original shapes match well with the hexagonal contour, and if the problem involves a large number of

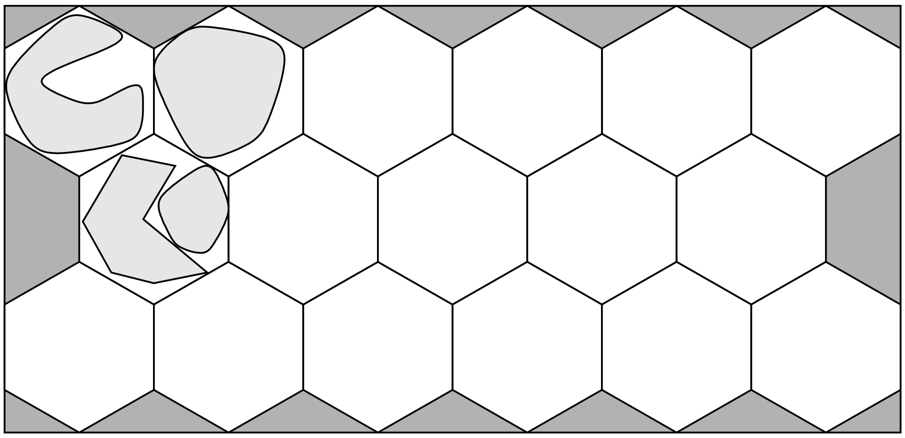


Figure 2.2: Tessellation of the region with hexagons.

relatively small pieces; hexagons indeed, form a perfect[1] tessellation only of an infinite two-dimensional region, but will give rise to some waste around the edges when fitted into a finite rectangular region, as shown in figure 2.2.

2.3 Direct placement methods

Straightforward single pass packing strategies involve taking the pieces in a given order and placing them on the stock-sheet according to a given placement policy. This may be repeated several times for different orderings or different placement strategies, and the best solution is chosen.

The simplest such approach is the Monte Carlo method, which forms the basis of a nesting package described by Böme and Graham [BG79]. The pieces are "thrown" onto the stock sheet at random; it is suggested that at least 2000 of these random trials are done in order to obtain acceptable solutions. The best solution found is then further improved by small random perturbations.

Qu and Sanders [QS87] propose two interesting extensions to the standard *strip packing* methods which proved to be quite effective. In the first, the pieces are sorted by decreasing length and then placed along two adjacent edges of the stock sheet. The resulting L-shaped region is then considered packed and effectively removed from consideration; the process is then repeated with the rectangular remaining area. The second method uses the same placement policy, but orders the pieces according to their heights. In the simple strip packing method the pieces are sorted into decreasing height order and laid in strips across the bottom of the stock sheet. The authors suggest a modification to this method to ensure that full advantage is taken of the possible interlock between pairs of non-rectangular pieces. Each strip is started with the tallest

[1] A *perfect* tessellation covers completely a given region with no waste.

remaining piece, but then the other pieces are dynamically sorted in order to chose the *best fitting* piece for each placement. Each remaining piece is tried to be placed close to the current profile and the ratio between the area of the piece and the free space remaining from the current packed profile to the right border of the piece, is calculated: the piece maximizing this ratio is chosen for placement.

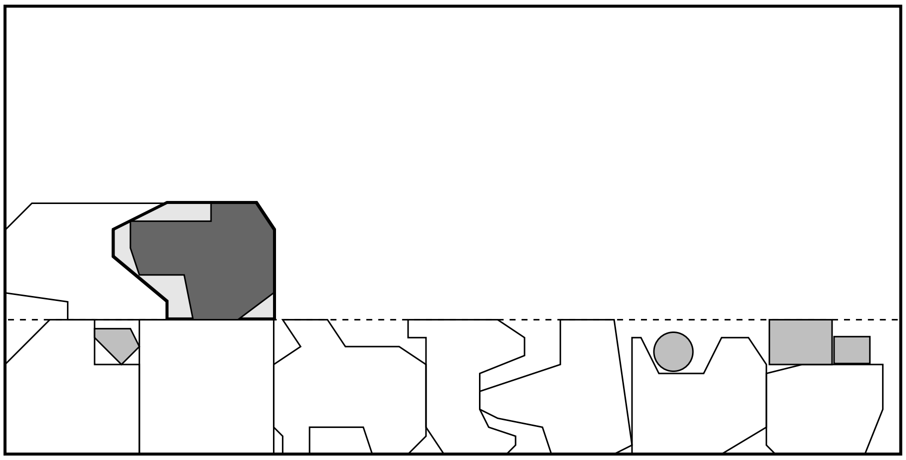


Figure 2.3: Example of a strip packing variant.

Once the whole strip has been packed as far as possible along the stock sheet border, the smaller pieces are considered for filling the spaces above the current placement, as long as they do not extend above the current height of the strip. The same process is then repeated for a new strip as shown in figure 2.3.

Other common methods use a *leftmost placement* rule: each piece is taken one at the time from a given list, sorted on a specific criterion (e.g. decreasing area), and placed on the stock sheet region as left as possible on the current packing profile. In one of the earliest paper on marker layout, Art [Art66] uses a leftmost placement policy, but suggests that a weighted combination of different factors be used to determine the exact location: these include minimizing the leftmost point of the piece, its rightmost point, and the wasted area. He also suggests that one ore more pieces could be manually combined together to form "meta-pieces" that could be more easily placed on the region, but could be also broken up during the algorithm if this appeared to be beneficial.

Dowsland e Dowsland [DDB98] also use a leftmost placement policy together with a random ordering of the pieces, but unlike many other, they don't limit the placement to the current profile and allow new pieces to effectively "jump" over pieces already placed to fill holes which have been left between larger pieces. This first solution is then improved by a "jostle" procedure that will be described in the next section.

Amaral et al. [ABJ90] use a more sophisticated method that doesn't allow hole filling but attempts to ensure that such holes will not be generated by selecting the next piece dynamically. Their method is based on the so called *sliding process* in order to find for each piece a suitable position. Pieces are sorted in decreasing order of their area and two different placement policies are used for large and small pieces, since small pieces can be used to fill holes left between the larger pieces in the layout. This reflect the strategy of a human expert working on the layout problem manually. A profile with the "visible" edges of already placed pieces is created; in order to simplify the calculation, the profile is approximated by the vertical and horizontal segments of the bounding box of the packed pieces. A set of placement zones is calculated, which is rectangular regions laying between adjacent steps of the current profile (see figure 2.4).

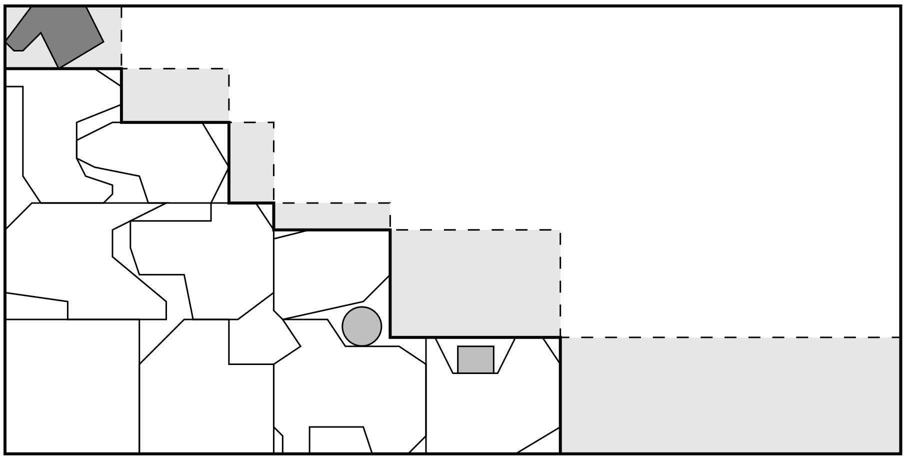


Figure 2.4: Dynamic selection of the piece that best fit the candidate zone.

The first piece which will fit into the leftmost placement zone is then selected and placed using a sliding vector technique until it touches one of the placed pieces. If the piece to be placed is classified as small, it must be inserted below the current profile, thus filling a potential hole that would be generated if a big piece were to be placed above there. In this phase the actual pieces dimensions, rather than its bounding box ones, are used to position the new piece accurately. Although the solutions provided by this algorithm were unable to match those of human experts, computational times were much faster.

Albano and Sapuppo [AS80] are concerned with the more complex problem in which the pieces may be placed in a number of different orientations. They also use leftward placement but attempt to improve a single pass algorithm by allowing some backtracking. They assume that the pieces are convex polygons which can assume any orientation; thus the full state space of solutions consists of the permutations of all pieces, each one assigned to all feasible positions and in all its possible rotations. Clearly it is not possible to scan the whole search space in a reasonable time even for limited size instances, therefore the search is guided by two bounds. The first one represents the quality of the partial solution obtained so far, and is a measure of the wasted are behind the current packing profile. The second is a rough estimate on the waste which will be generated by the remaining pieces, and is approximated by a fixed proportion of their area. The branch which minimizes the sum of these two bounds is chosen next. To further reduce the search space, the number of branches at each node is limited in two ways: only one orientation is allowed for each piece, namely the one which results in the smallest extension to the right. Thus each remaining piece generates only one branch at each node, but not all of them are visited: only a fixed number of nodes with the smallest right projection will be considered. Finally, when backtracking occurs, the process is forced to backtrack of a certain number of levels, in order to avoid visiting nodes corresponding to solutions too close to the one just discarded.

A major research project on marker making for the textile industry has been recently developed at Harvard University by Victor Milenkovic and its research group. One of their first papers on this subject [MDL92], describes an algorithm which seeks to approximate the approach of an experienced worker for the marker layout problem of trousers. They first try to place the larger pieces in columns, then they insert the small pieces into the gaps left by the big ones eventually moving some of them to easily accommodate the trims. In order to respect the grain constraint, pieces are only allowed to rotate by 180° and to flip vertically, for a total of 4 orientations per piece. For every column, all possible combinations of big pieces are tried, and the one with maximum height is chosen. The precise y-coordinate of all pieces are calculated once all the stacks have been decided, but the feasibility of each column is verified by means of geometrical constraints among adjacent pieces. Furthermore, a heuristic algorithm checks that there exists a feasible placement of the remaining pieces (only considering the x-coordinate), with a quasi-vertical right profile of the last column. Figure 2.5 shows an example of a marker generated with this method on a trouser instance.

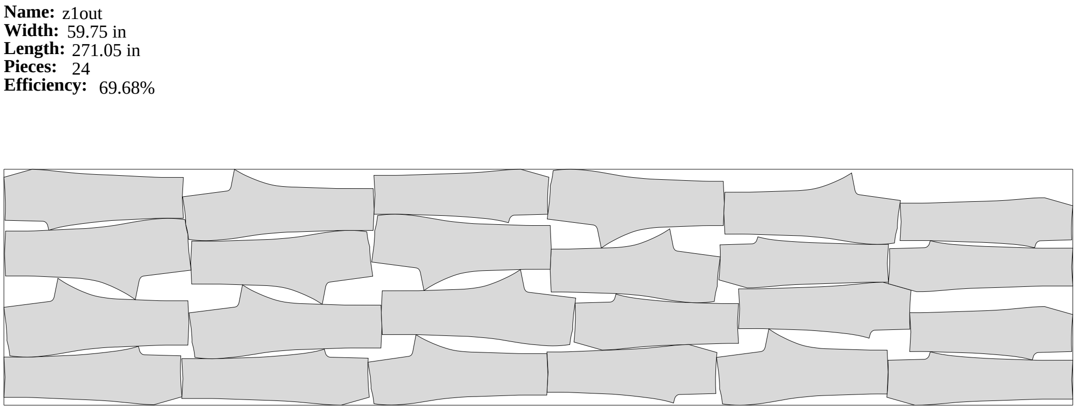


Figure 2.5: Automatic placement of the larger pieces using the heuristic algorithm by Milenkovic.

Finally when all big pieces have been placed, the algorithm proceeds with the insertion of the small ones in the gaps left by the big pieces. If some of the trims cannot fit, a special procedure is called to enlarge some gaps to accommodate the remaining pieces without increasing the total length.

Haistermann and Lengauer [HL93] faced the more complex problem of determining cutting patterns for leather hides. Since no two hides are identical, a new pattern must be generated for each one, so the objective becomes finding a fast method to generate solutions of reasonable efficiency. Both the hides and the demanded pieces are very irregular, so the first step of the algorithm is to use a smoothing routine to approximate the original shapes by polygons with fewer points, and hence fewer "deep" concavities. Pieces are then coded with a set of parameters which gives a rough estimation of the profile of the piece. This, together with the size, is then used to evaluate the suitability of each piece for placing against the current packing profile.

2.4 Improvement methods

A different approach, which has become increasingly popular in recent years, is to produce an initial layout which may be feasible but suboptimal, or slightly infeasible, and then to use small changes in order to improve it. Such an approach may either continuously seek for improvement, or may incorporate a meta-heuristic technique such as *simulated annealing* or *tabu search* in order to allow non-improving steps.

Jain et al. [JFR92] use simulated annealing to solve a nesting problem with many pieces and any orientation. For each figure they store a set of parameters: coordinate of the reference point, angular rotation and displacement vector with successive copies of the same piece. A solution is then defined by associating a set of values with these parameters. The quality of the nesting is measured as a linear combination of area utilization and penalty for overlap. A neighbor solution is defined by changing one of the parameters and therefore corresponds to changing the orientation of one piece, or its position, or eventually the displacement between two pieces. In order to avoid large perturbations of the solution, which are unlikely to be accepted at low temperature, the magnitude of each one of the changes is bounded by a factor which decreases with the temperature. Computational experiments using simulated annealing suggest that this technique can give good results, but has the drawback of excessive computing time: to guarantee good solution indeed, the temperature has to decrease very slowly, thus augmenting the computational time.

Also Lutfiyya et al. [LMPD92] tackle the same problem but allow only a fixed set of orientations. Infeasible solutions in which some pieces overlap are included in the solution space and are penalized in the cost function by a penalty factor multiplied by an approximation of the overlap area. The second term of the objective function tries to ensure that pieces are nested well, by rewarding edges at similar angles which are coincident or close to one another, while the third term forces the pieces to lay as close as possible to the origin. A neighborhood move consists of displacing a piece, changing its orientation or swapping two pieces.

Oliveira end Ferreira [OF93] use a formulation which core closely meets the actual requirements of a typical practical application. They fix the width of the stock sheet and use simulated annealing to minimize its length. Overlap solutions are allowed and penalized in the objective function together with the total length required. Neighbor solutions are obtained by moving a single piece to a new position.

Dowsland et al. [DDB98] introduce a new improvement technique called *jostling*, which outperforms their own implementation of the typical simulated annealing algorithm. The "jostle" procedure is inspired by the observation that a jagged right hand end of a marker is less efficient than a flatter one and that when granular products are stored in a container, any unevenness in the surface can be removed by shaking the container back and forth. According to their algorithm, pieces are first sorted by any criteria (either arbitrary or following some heuristic rules) and then placed using a standard leftmost placement policy; then they are selected in decreasing order of the x-coordinate of their rightmost point, and packed according to a rightmost placement policy. Once this packing is complete, the pieces are reordered in increasing order of their leftmost points and packed again using a leftmost placement policy. This process is repeated for a fixed number of iterations, and can be applied to any initial ordering and placement policy. Computational experiments suggest that around 20 "jostles" can give results which significantly improve upon a one-pass layout.

A research project with a direct application in the industry field has been carried out by Milenkovic, Daniels and Li. In two important papers [MDL92] and [LM95] they suggest an improvement method for practical problem in the garment industry. They observe that many instances consist of similar mixes of similar shapes, and therefore suggest that solutions to new problem be initialized from a database of human expert solutions to similar problems. Depending on the exact dimensions of the pieces, such a solution may contain small amounts of overlap or small gaps; thus it may need an expansion process to remove the overlap or a compaction process to fill the gaps, or a combination of the two. The basic idea of compaction is to simulate forces on the pieces which move them in the required way. Thus if the objective is to minimize the length required by a given layout, a "gravitational" force squeezing everything to the left will have the desired effect. Conversely, if the objective is to open up a gap between two pieces in order to fit a smaller piece, then a repulsion force between the two is required as shown in figure 2.6.

Such process can be modeled either using simulation techniques developed for computer simulations of rigid objects acting under different forces, or by solving a series of

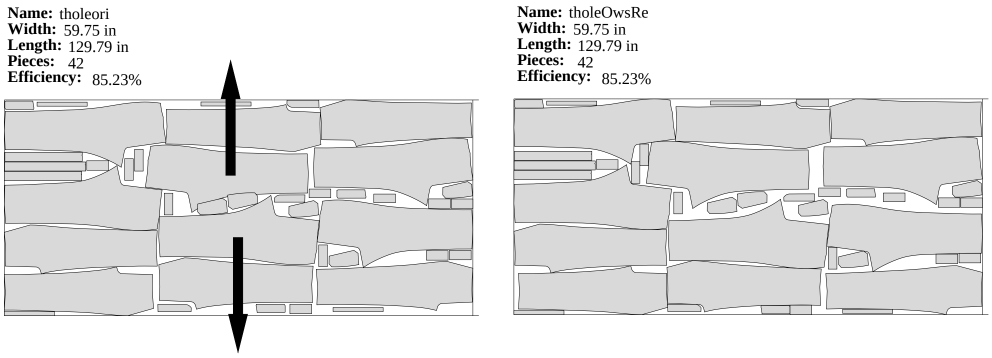


Figure 2.6: Opening of a gap to better accommodate small pieces.

differential equations. However both these approaches prove to be too slow from a practical point of view. The solution finally adopted involves the use of linear programming. The displacement vector of the pieces are calculated, and for each pair of pieces, the no overlap constraint defines a feasible region for the difference between their vectors $u - v$ (in chapter 4 we give a detailed explanation on how to build and use such constraints). The objective is to move the pieces in such a way as to minimize the potential energy caused by the imaginary forces $E = -\sum_i f_i \cdot v_i$, subject to the vectors $u - v$ remaining within their feasible regions. In general these regions will not be convex an the resulting minimization problem is NP-hard. However it is possible to select convex subregions which allow some movement in the required direction. The problem can then be solved using linear programming, and the precess repeated until no further improvements are possible. A similar model can be used to move pieces apart to eliminate overlap. This model will be analyzed in detail in chapter 5.

Chapter 3

A tabu search heuristic for the Irregular Nesting Problem

3.1 Representation

Our first approach to the nesting problem has been using a simple representation and a tabu search technique.

We decided to use matrices to represent the figures to be placed and the holes where to place them. Bitmaps are the simplest way to store this information, thus we used different size matrices where each pixel is set to 1 if it is part of the figure, 0 otherwise. This structure is quite straightforward to implement and offers an easy way to detect overlap among figures: all you need is just to check if the bits representing the same point on the solution are both set to 1. But this approach also has the disadvantage of requiring a lot of memory to store the whole matrix, and a lot of time to analyze all its points. Furthermore, the size of the matrix depends on the resolution adopted, and is independent from the figure it stores (see fig. 3.1). So, depending on the resolution to be used, the number of points to represent the same figure grows very quickly and the time to deal with these matrices may become excessive.

We adopted the matrix representation but implemented some add-on structure to speed up the overlap checking. Together with the bitmap we store the perimetric points plus some internal points selected in a particular way. This double structure together with some other tricks explained later, allows our algorithm to run with reasonable times even with medium resolution.

3.2 Initial placement

We start with all pieces unplaced stored in a proper *piece-list* and the stock sheet empty; the *hole-list* initially contains the whole stock sheet, considered as a hole of fixed

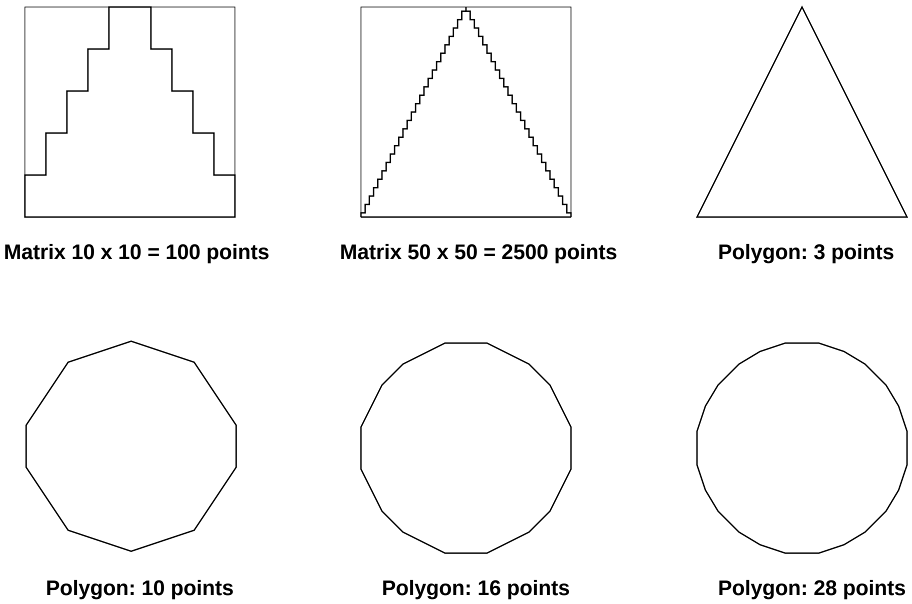


Figure 3.1: Matrix and polygon representation of different figures.

width and "virtually infinite" length.

As a first step we sort the *piece-list* by some criterion, usually by area, and start placing one piece at a time until the list is emptied. For each piece we consider all the holes big enough to include it, starting from the smallest one: we try all the possible rotations of the piece in order to find the best one. The so called *best fit criterion* allows us to use as much as we can all the holes created from the placement of the preceding pieces, always choosing the smallest hole where the piece can fit. We are sure that every piece is placed since there always exists a "virtually infinite" hole, namely the stock sheet extending to the right of all placed pieces. After a new piece is placed we update the *hole-list* by removing the old hole and inserting the just created ones: in the example of fig. 3.2 hole a is replaced by new holes b, c and d.

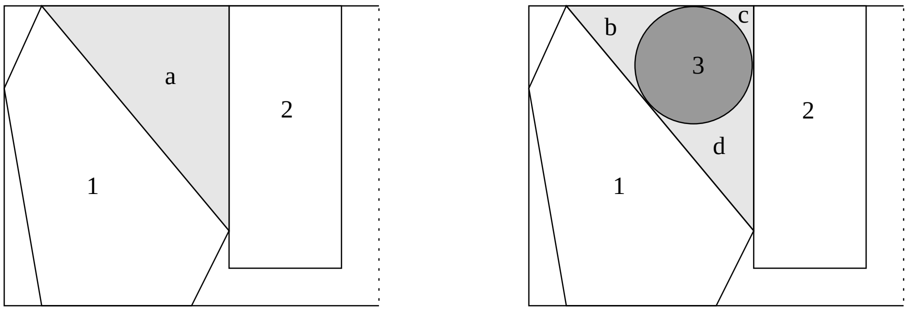


Figure 3.2: Situation before and after the placement of piece 3.

The main goal is to minimize the length of the right used margin, which is to compact all pieces as much as possible toward the left side of the stock sheet.

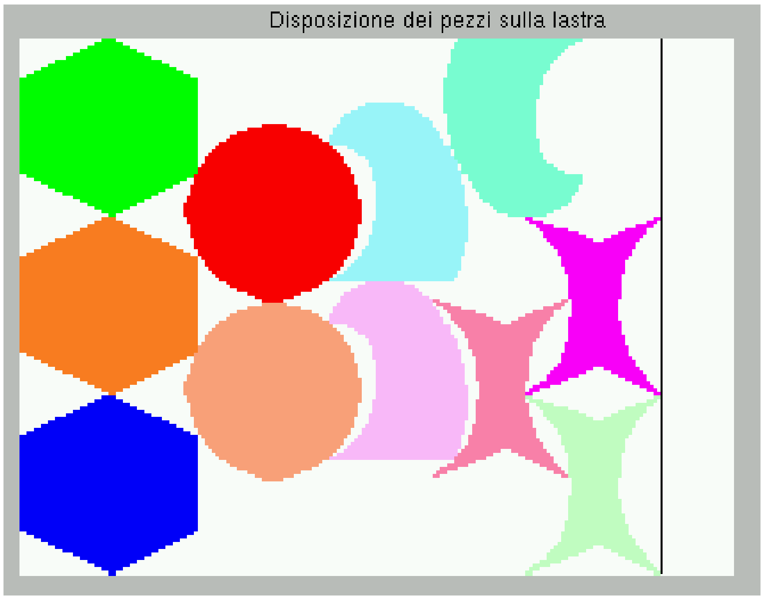


Figure 3.3: Initial placement of the pieces.

As one can see from the example of fig. 3.3, our algorithm places both circles as much as possible to the left, nesting them among the hexagons, but wasting a lot of area above and below them. This is the disadvantage of using a greedy strategy: optimizing the placement of a single figure doesn't guarantee the optimality of the complete solution. Placing the circles toward the borders requires a bigger length but it would allow a third circle to fit between them.

This drawback is particularly evident in the instances where all pieces are more or less of the same size. If there are smaller pieces, instead, these will probably fit in the holes, thus reducing the waste generated by the big pieces; this fact can be observed in figure 3.4 which shows the initial solution obtained for test 9 and 10 derived from Blazewicz et al. [BHW93].

To overcome the problem above one could sort the initial *piece-list* by different criteria: area of enclosing rectangle, length of enclosing rectangle, width of enclosing rectangle, perimeter of enclosing rectangle, area of polygon, perimeter of polygon, utilization ratio of enclosing rectangle, are the most cited in the literature. We made several simulations and found out that it is very difficult to find a good criterion for all instances, because some of them work well for a certain set of instances, but others work better on some other instances. We also investigated the behavior of a randomized

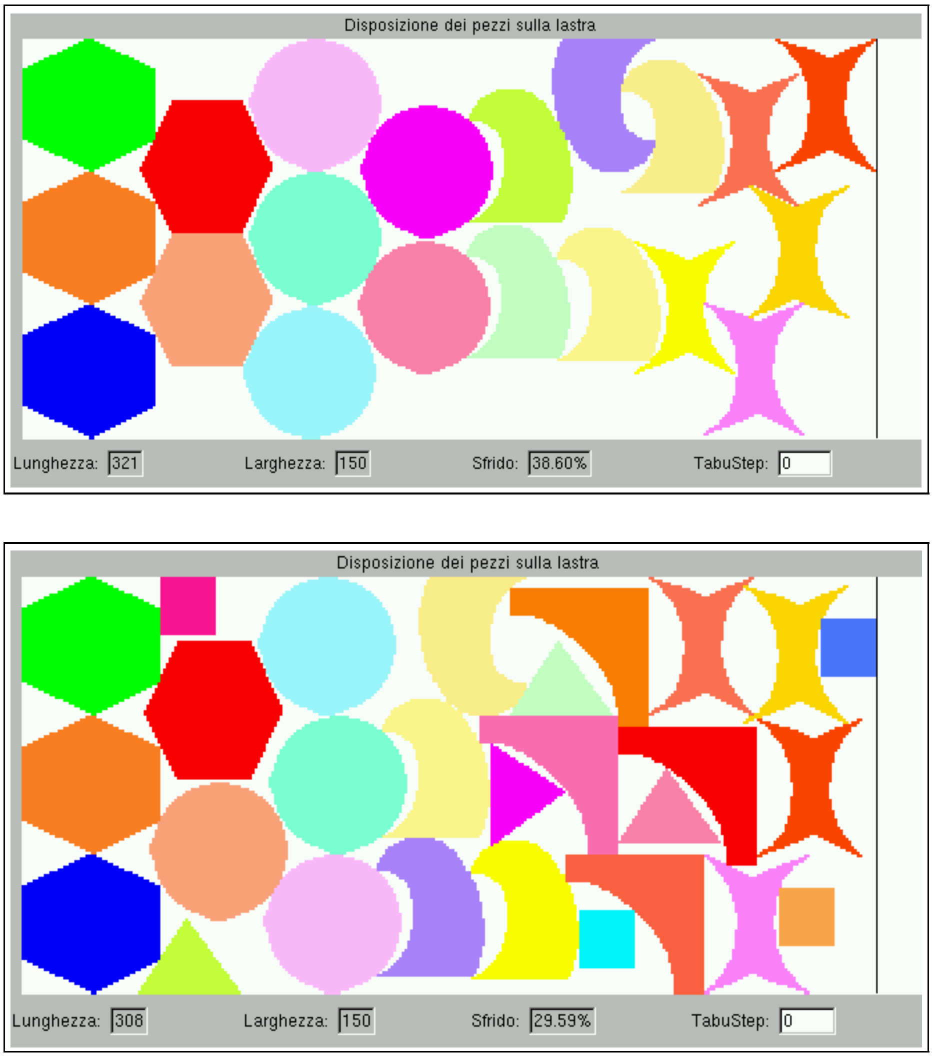


Figure 3.4: Initial placement for test 9 and test 10.

choice of the position, where for every piece the algorithm decides at random (with a certain probability) if placing the piece toward the left side or toward the upper or lower borders.

Figure 3.5: Randomized initial placement for test 9 and test 10.

The initial solution obtained with the randomized placement are shown in fig. 3.5 and can be compared with those obtained with "normal" placement in fig. 3.4 ; for test 9 there is a considerable reduction of waste, from 38.60% to 32.03%, while for test 10 the reduction of waste from 29.59% to 27.96% is not so impressive. As expected, the improvement is more significant for instances where all pieces have a comparable size.

3.3 Data structure improvement

As mentioned in section 3.1, the main disadvantage of using a matrix representation is the need of checking a large number of points (theoretically all points of a piece) in order to detect a possible overlap with another figure. Instead of checking all points, we only check the perimetric points and eventually some suitably-defined internal points to make sure that one piece does not completely include the other. If no points overlap, then the position is considered feasible.

In order to use this mechanism, for any figure its perimetric points have to be calculated; this operation is done by a recursive procedure which analyze all points close to the figure border. The precomputation phase adds an overhead time when reading the input figures but saves a lot of time in the placement phase, since one of the most time consuming procedures is indeed the check for overlap. The first two columns of table 3.1 gives a rough idea of the time saving obtained by exploiting this technique. As one can see the loading time is increased for the instances with high resolution due to the perimeter calculation time, but the placement time is drastically reduced by a factor 3.

	MATRIX	PERIMETER	DYNAMIC LIST
Test 9 low resolution			
Loading	0.13	0.12	0.20
Placement	0.22	0.16	0.14
Total	0.51	0.38	0.51
Test 10 low resolution			
Loading	0.27	0.27	0.29
Placement	0.25	0.18	0.15
Total	0.71	0.55	0.49
Test 9 high resolution			
Loading	2.38	2.40	2.43
Placement	22.73	7.24	3.15
Total	26.69	11.32	7.24
Test 10 high resolution			
Loading	2.58	3.25	3.21
Placement	23.85	7.10	3.32
Total	27.13	12.21	8.47

Table 3.1: Comparison of loading and placement time with the different data structures.

To reduce the execution time even further, we adopted a dynamic management of

the perimetric point list. When trying to place a certain piece, the algorithm tries many positions for the reference point of the figure with respect to a same hole, namely all the possible ones starting from the left. For a placement to be feasible all its perimetric points have to be checked, but if just one point overlaps we can stop the check and declare this position unacceptable.
If a certain perimetric point determines an overlap with another point in the hole, it is quite probable that the same point will cause an infeasibility also when the reference point is moved to an adjacent position, thus it is convenient to check this point first. Therefore, every time an infeasibility is detected, the perimetric point that caused it is moved to the head of the list so that it will be the first point checked in the next iteration.

The overhead induced by this operation is minimum since it just needs to move some pointers in the list, but again we obtained a considerable reduction of the placement times as one can see from the last column of table3.1.

3.4 Tabu search

Once an initial solution has been found, a tabu search technique is applied to improve the solution. It consists in a sequence of piece movements from their original position into new holes, both internal and external[1].

Tabu search is a heuristic technique to improve a given solution by visiting its neighborhood. It was introduced in the '70's as an alternative apprach to the traditional iterative methods, and successively formalized by Fred Glover [Glo89, Glo90], and Hertz and de Werra [HdW90]. Classical descent iterative methods have the main disadvantage of getting stuck in local optima that rarely coincide with the global one. To overcome this problem one could start from different initial solution, or could allow non-improving steps so that the algorithm can escape from local optima and visit a wider neighborhood (see fig. 3.6).

After a non-improving step it is highly probable that the algorithm will return to the previous local optimum, since this corresponds to the most improving step. To avoid this inconvenient one has to store not only the value of the best solution found so far, but also the path to the current one. These informations are used to avoid considering already visited solutions, that are considered tabu[2].

A tabu search algorithm keeps track of the previous solutions and forbids to visit the same solution a second time. However since the time and memory to store a whole list of solutions can be excessively high, one can only store the "move" to transfer from one solution to the next one, or equivalently its "inverse move". In our case we

[1]The external hole is the unused part of the stock sheet extending to the right of all placed pieces.
[2]From here the name of the algorithm: *tabu search*.

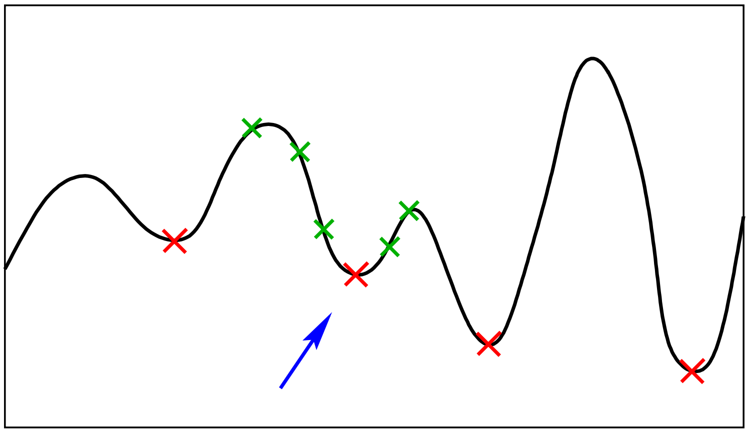


Figure 3.6: Tabu search exploration scheme.

accomplish this task by storing, for each piece, the last iteration where it was moved: a piece is in the tabu list if the difference between the current tabu iteration and the last iteration where it was moved, is smaller than a certain parameter *tabu size*.

The tabu size parameter determines the length of the tabu list and is a fundamental parameter of this methodology. Depending on the tabu size the algorithm can intensify the search in a restricted neighborhood, or it can diversify the search by visiting a wider neighborhood.

3.5 Using *tabu search* to improve the solution

As already said in the previous section, once an initial solution is found the algorithm keeps moving pieces in order to find a better solution. The main objective is to reduce the total length, thus it first try to move an external piece[3] into an internal hole using the best-fit criterion. If this is not possible, the algorithm computes the waste of each internal piece, as the sum of all the holes adjacent to that piece. Then the piece with the higher waste is removed from the solution and placed on the external hole.

Before considering a certain piece the algorithm always checks whether this piece is available, meaning that it does not belong to the tabu list; as explained before, this is accomplished by simply checking the tabu iteration in which it has been moved for the last time. After a piece is moved, the current tabu iteration is marked on the piece, so that it will not be available for the next *tabu size* iterations.

The following scheme represent a main iteration of the tabu search algorithm; where $TABU$ and $HOLES$ represent the list of the tabu pieces and of the available holes, respectively.

[3]We define a piece as *external* if it is adjacent to the external hole, that is if it belongs to the right profile.

try to place an external piece $p \notin TABU$ in an internal hole $h \in HOLES$
by using the *best fit* criterion;
if move is feasible
then
perform the move;
else
chose the internal piece $p \notin TABU$ with the biggest waste;
remove p from the solution and place it in the external hole;
endif
update the tabu list $TABU$;
recalculate the holes and update the hole list $HOLES$;

Figure 3.7: Iteration of the tabu search algorithm.

Figure 3.8 shows some tabu steps on a given instance, with a reduction of the waste from 36.31% to 29.70%.

3.6 Compaction

As pieces are moved around during the tabu search phase, holes are not always completely filled by new pieces and often big wastes of material occur. This happens especially when a piece is removed from a certain hole and replaced by a smaller one: the narrow holes just created are not big enough for another piece, and the total waste grows significantly.
In this case a compaction of the pieces is needed, in order to reduce the waste and compress all pieces toward the origin[4].

Our compaction procedure starts sorting all pieces by their leftmost point in ascending order. Then the first piece is moved to the left of one unit: if the move is feasible it is recursively moved left by twice the previous quantity until an overlap is detected. Now with a bisection method the algorithm looks for the maximum left shift admittable and fixes the new position of the piece.
The procedure is then repeated an all other pieces, until every figure has at least one contact point with every other neighbor. This process doesn't guarantee of course a complete elimination of waste, but it helps reducing it.

To better exploit this idea, we also allow pieces to slid diagonally when they get in contact with another piece, thus improving the power of the compaction procedure.

In our code we run the compaction procedure every 25 tabu iterations; figure 3.9

[4] Lower left angle.

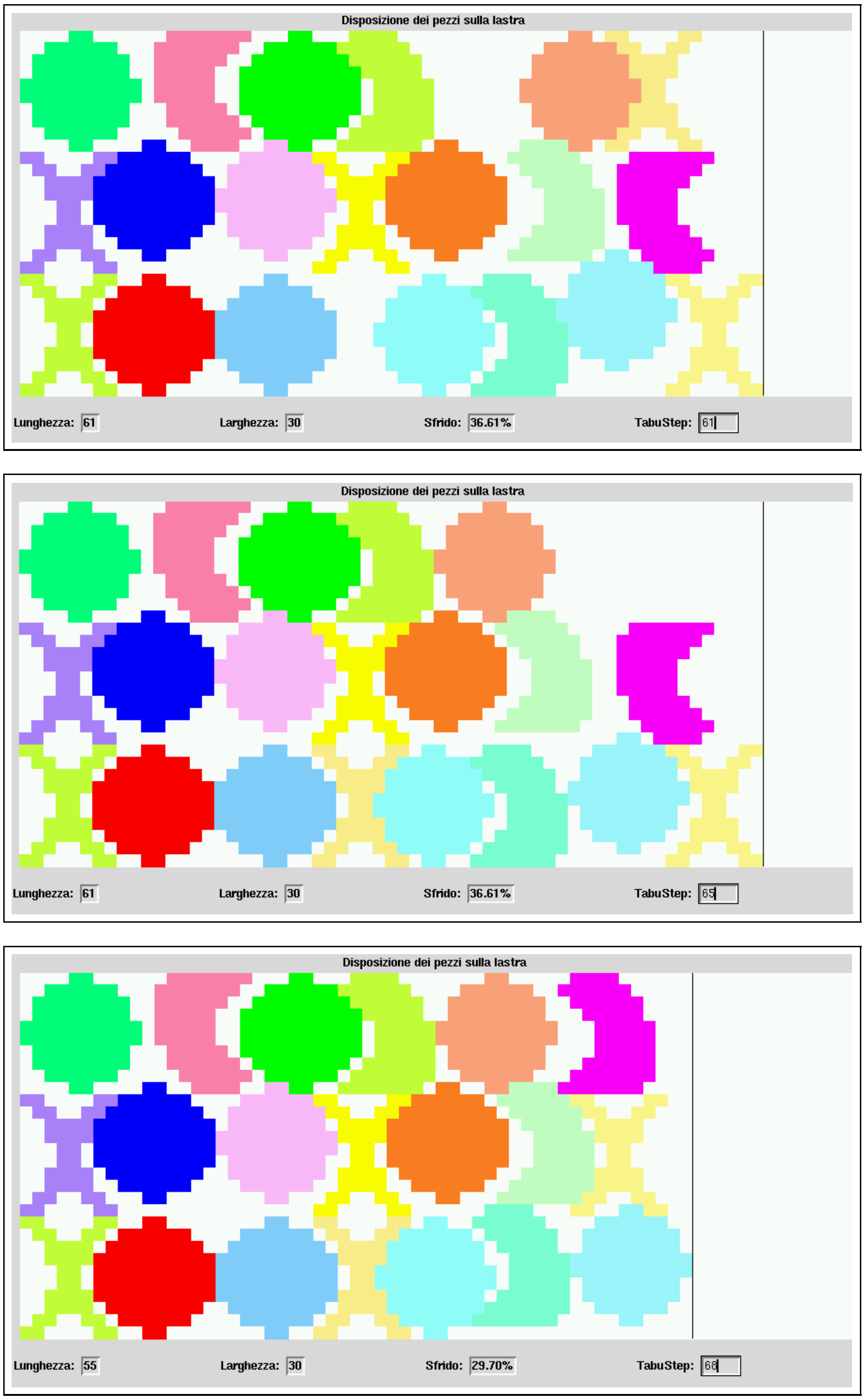


Figure 3.8: Some tabu steps.

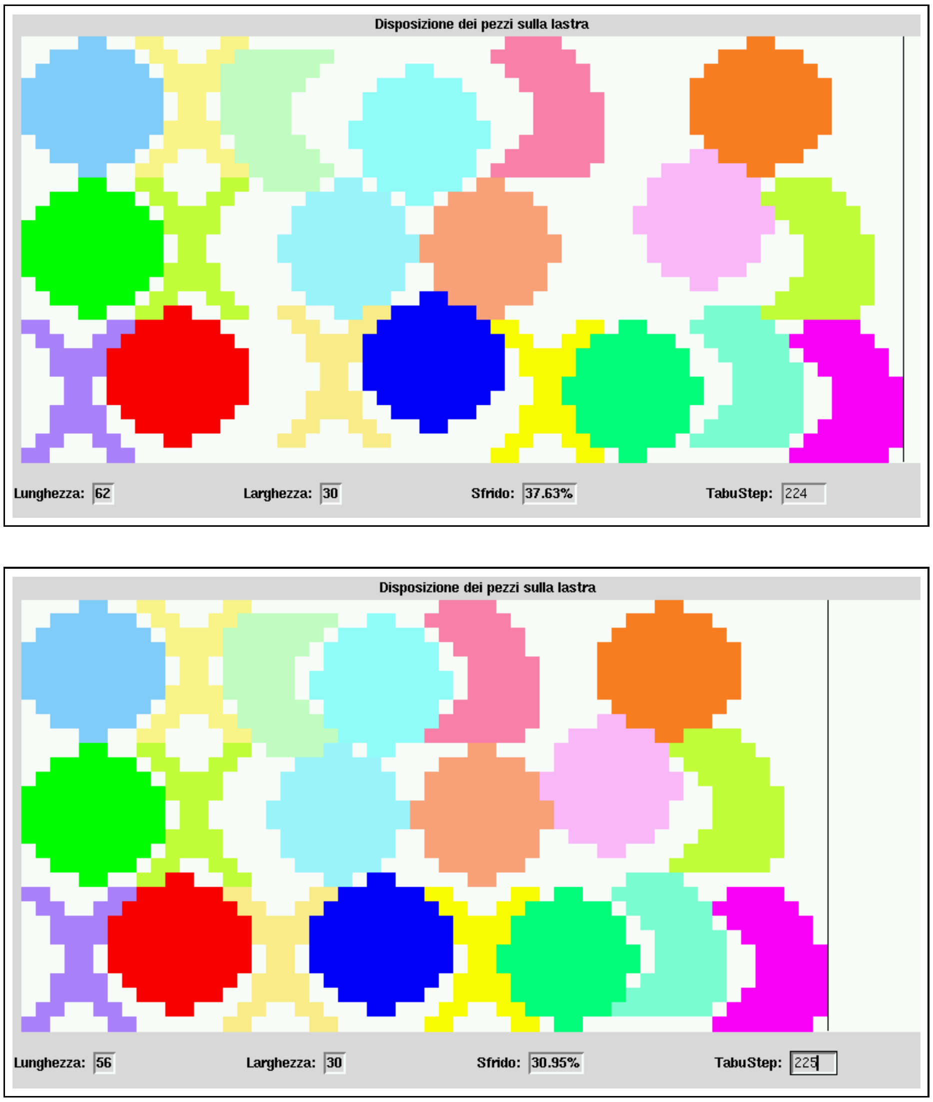


Figure 3.9: A compaction step applied to the current solution.

shows the solution before and after a compaction step.

3.7 Diversification

Another key concept in the tabu search methodology is diversification: it consists of a big step to a completely different solution, i.e. a move toward an unexplored region of the solution space.

We run a diversification step every 100 tabu iterations in the following way. First a compaction step is performed using the procedure described in the previous section; then the waste of each piece is calculated, and the first k pieces with the largest waste are removed from the solution and placed in the external hole.

The resulting solution is now quite bad but after a few tabu steps, it will get more compact and much different than the previous one.

This is not the only way to perform a diversification step. As we explained before, the tabu size is the key parameter to act on the algorithm. The smaller the tabu size, the sooner a piece becomes available again: this directly affects the solution with a continuous movement of the external pieces which translates into an intensification of the search around the current solution. One the other hand, increasing the tabu size keeps the pieces blocked for a longer period, allowing a greater number of pieces (eventually also the internal ones) to be processed and moved: this means a great movement of pieces around the stock sheet, hence a diversification step.

3.8 Computational results

We tested our code on instances found in the literature and described in the paper by Oliveira et al. [OGF00]. On almost all the tests we improved the best solution found in the literature, though the results cannot be compared directly, due to the different piece representation adopted (see table 3.2). These results confirm the validity of the heuristic approach to very difficult problems like the irregular nesting problem, and confirm once more the effectiveness of the tabu search methodology.

	GREEDY BY SIZE	RANDOM GREEDY	TABU SEARCH	BEST FROM LITERATURE
Blazewicz 9: data from Oliveira & Ferreira (width = 150)				
Length	254	246	237	
Waste	33.15%	30.98%	28.35%	
Blazewicz 10: data from Oliveira & Ferreira (width = 150)				
Length	301	277	269	289
Waste	34.01%	29.16%	27.06%	
Oliveira & Ferreira: fixed rotation (width = 40)				
Length	69		66	66.75
Waste	38.48%		35.68%	
Oliveira & Ferreira: rotation 180° (width = 40)				
Length	66	75	61	61
Waste	35.68%	33.40%	30.41%	
Shirts: data from Oliveira & Ferreira (width = 400)				
Length	652		640	664
Waste	16.79%		15.23%	
Trousers: data from Oliveira & Ferreira (width = 79)				
Length	259		256	263.17
Waste	12.41%		11.38%	

Table 3.2: Computational results of the tabu search algorithm on instances from Oliveira & Ferreira.

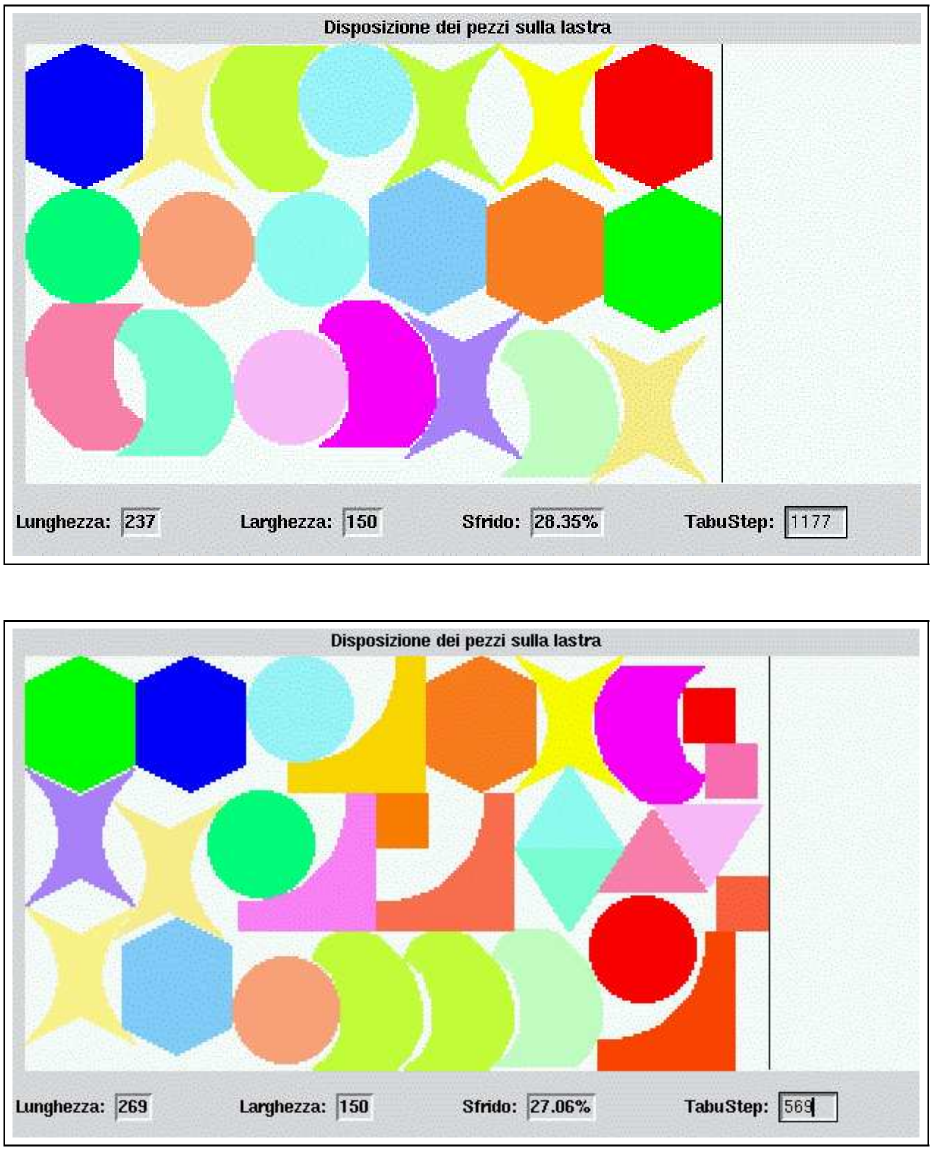
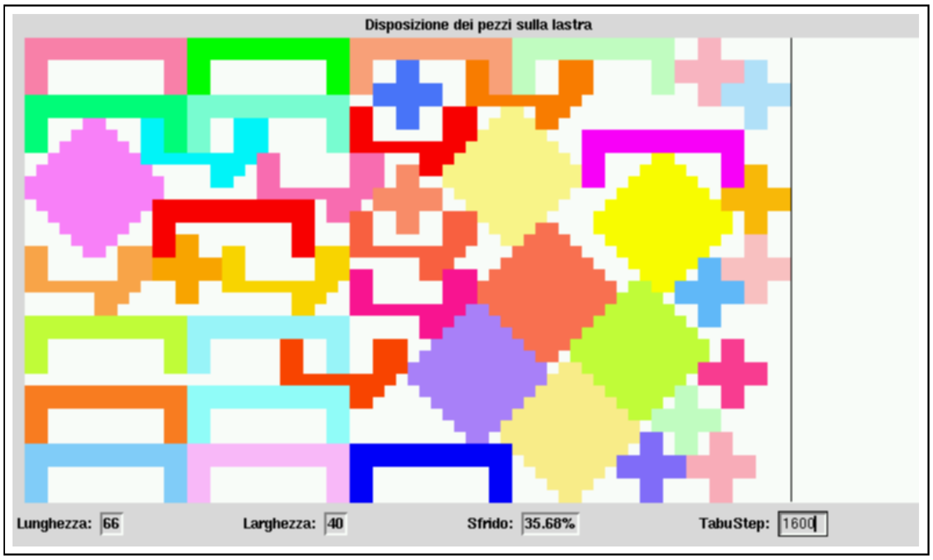


Figure 3.10: Blazewicz 9 and 10: data from Oliveira & Ferreira



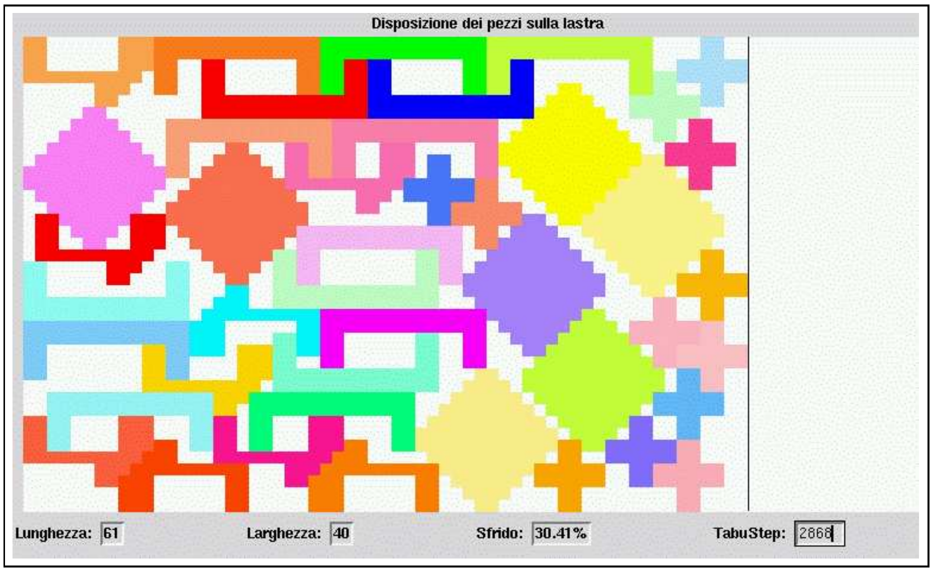


Figure 3.11: Test Oliveira & Ferreira: fixed rotation (top) and 180 ° rotations (bottom)

Figure 3.12: Test Oliveira & Ferreira: shirts (top) and trousers (bottom)

Chapter 4

Computational geometry

When placing pieces on the stock sheet the most frequent operation is the definition of a feasible position, meaning a position where the current piece is completely within the marker region and does not overlap any other already placed piece.

The simplest operation to perform is an overlap checking between two polygons. Preparata and Shamos [PS85] introduced an algorithm to accomplish this task. Amaral et al. [ABJ90] and Oliveira and Ferreira [OF93] used another method called *D functions* to detect intersection of segments. If these methods work quite well in the simple case, things get really complicated when both polygons are non convex. In this case a possible solution is to divide one of the figures in convex components and to solve the simplified problem introducing special constraints to glue these components together.

Another methodology to detect intersection of figures uses the so-called *no-fit polygon*. This concept was first introduced by Adamowicz and Albano [AA76], then Milenkovich et al. [LM95] gave a detailed explanation of its utilization in the nesting process, and it is now probably the most used method to determine the interaction between two polygons. The definition of the no-fit polygon is based on the mathematical concept of Minkowski sum [Min03].

4.1 Minkowski sum and difference

The Minkowski sum and difference are powerful preprocessing tools for polygon intersection and containment problems. Using the Minkowski sum and difference, respectively, we can convert a polygon-polygon intersection (overlap) query and a polygon-polygon containment query into point-in-polygon queries, which allow us to achieve sub-linear query times.

Definition 4.1 ([Min03, GRS83, Ser82]) *The* **Minkowski sum** *of two polygons A and B is defined as:*

$$A \oplus B \;=\; \{a + b :\; a \in A,\; b \in B\}$$

or equivalently:

$$A \oplus B \;=\; \bigcup_{b \in B} A^b$$

where the notation A^b means the translation of polygon A by vector b:

$$A^b \;=\; \{a + b :\; a \in A\}$$

Definition 4.2 ([Min03, GRS83, Ser82]) *The* **Minkowski difference** *of polygons A and B is defined as:*

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

From the definition it follows that:

$$A \oplus B \;=\; B \oplus A$$

but in general

$$A \ominus B \;\neq\; B \ominus A$$

The next lemma establishes the relationship between Minkowski sum and difference.

Lemma 4.1 ([Ser82])

$$A \ominus B \;=\; \overline{\overline{A} \oplus B}$$

where $\overline{A}$ denotes the complement of A:

$$\overline{A} := \{a : a \notin A\}$$

and symmetrically:

Lemma 4.2 ([Ser82])

$$A \oplus B \;=\; \overline{\overline{A} \ominus B}$$

The next lemma shows that the "shape" of the Minkowski sum and Minkowski difference are translation invariant.

Lemma 4.3 *Let A and B be two point sets. Let s and t be two points. Then*

$$A^s \oplus B^t \;=\; (A \oplus B)^{s+t}$$

and

$$A^s \ominus B^t \;=\; (A \ominus B)^{s+t}$$

4.2 Intersection

Definition 4.3 *The* **no-fit polygon** *between two polygons A and B is:*

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

where $-B$ is the reflective image of B w.r.t. the origin:

$$-B = \{-b : b \in B\}$$

The no-fit polygon between A and B represents the region of interaction between the two polygons. Given the definition above, we can state the following:

Theorem 4.1 ([GRS83, Ser82]) *Let A and B be two point sets and x be a point in the plane. Then $A \cap B^x \neq \emptyset$ if and only if $x \in U_{AB}$.*

From lemma 4.3 on the translation we derive the following:

Corollary 4.1 *$A^s \cap B^t \neq \emptyset$ if and only if $t - s \in U_{AB}$*

Corollary 4.1 states that, regardless of the actual position of the polygons in the plane, we can verify their intersection by simply checking if the difference of their displacement vectors belongs to the no-fit polygon U_{AB} of A and B calculated in the origin.

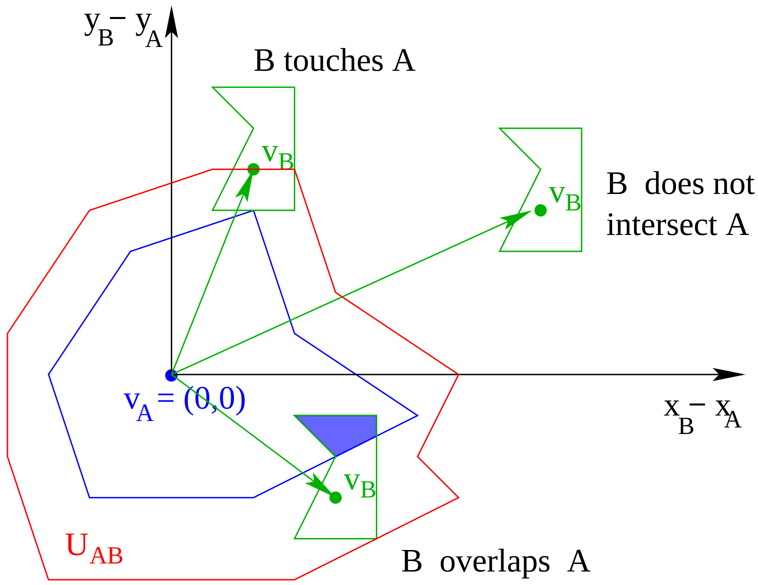


Figure 4.1: Using no-fit polygon U_{AB} to determine intersection between two polygons A and B.

When the reference point of polygon A is in the origin (as shown in figure 4.1), the no-fit polygon can be interpreted as the locus of points of the reference point of B when it slides without rotating around polygon A.

Lemma 4.4 *One of the following situations is verified:*

- *the reference point of B lies inside the no-fit polygon U_{AB}*
  $\Longrightarrow$ *pieces A and B overlap;*

- *the reference point of B lies exactly on the border of the no-fit polygon U_{AB}*
  $\Longrightarrow$ *pieces A and B touch each other without overlapping;*

- *the reference point of B lies outside the no-fit polygon U_{AB}*
  $\Longrightarrow$ *pieces A and B are far apart.*

4.3 Containment

Minkowski difference 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.

Theorem 4.2 ([Ser82]) *Let A and B be two point sets and x be a point in the plane. Then $B^x \subseteq A$ if and only if $x \in A \ominus (-B)$.*

The theorem above can be proved by the following intuition: let $\overline{A}$ be the complement of A. If A is a polygon, then $\overline{A}$ is the exterior of A. For a vector x, the condition that B^x is totally contained in A is equivalent to the condition that the intersection of B^x with $\overline{A}$ is empty. By Theorem 4.1, this is the same as $x \notin \overline{A} \oplus (-B)$, or equivalently, $x \in \overline{\overline{A} \oplus (-B)}$. But lemma 4.1 states that $\overline{\overline{A} \oplus (-B)}$ is exactly the Minkowski difference $A \ominus (-B)$, which proves the claim.

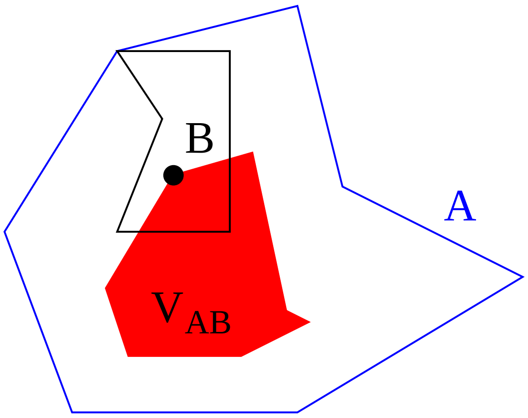


Figure 4.2: Using Minkowski difference to determine the feasible displacement of a polygon inside another one.

As shown in figure 4.2, piece B is entirely contained within polygon A as long as the reference point of B belongs to the shaded region $V_{AB} = A \ominus (-B)$.

4.4 Algorithms for computing the Minkowski sum

The complexity of computing the Minkowski sum of two polygons varies depending on the class of polygons under considerations.

Guibas et al. [GRS83] observed that the Minkowski sum of two convex polygons can be computed in linear time by merging the edge segments of the two polygons.

In general, it is easy to see that an edge segment on the boundary of the Minkowski sum of polygons P and Q is part of an edge segment formed as the sum of a vertex in P and an edge in Q or vice versa. Let us call the edges formed by the sum of a vertex in one polygon and an edge of the other polygon the *candidate edges*. If there are n vertices in P and m vertices in Q, then there are $O(mn)$ candidate edges. A natural idea for generating the Minkowski sum is to calculate the arrangement of the candidate edges in $O(m^2n^2 \log mn)$ time. The algorithms in Kaul et al. [KOS91] and Agarwal et al. [AST92] for calculating the Minkowski sum of two simple polygons follows this idea. Kaul et al. [KOS91] introduced the concept of vertex-edge supporting pairs which reduces the number of candidate edges.

In the worst case, the Minkowski sum of two simple polygons can have $O(m^2n^2)$ edges and the same number of internal holes. The following table summarizes the maximum number of candidate edges for the possible combinations of types of polygons.

Polygon A	**Polygon B**	**Candidate edges**
convex	convex	$O(m+n)$
non convex	convex	$O(mn)$
non convex	non convex	$O(m^2n^2)$

Milenkowic et al. [MDL92] identified a class of polygons called *star-shaped* which are not as restricted as convex polygons but also easier to handle than simple polygons. A polygon P is star-shaped if there exists a point $k \in P$ such that for each other point $p \in P$, the entire segment kp lies inside P. Such a point k is called a *kernel point* of the polygon. Note that a convex polygon is a special case of a star-shaped polygon in which the kernel equals the entire polygon.

They proved that the Minkowski sum of two star-shaped polygons is also a star-shaped polygon; thus star-shaped polygons are "closed" under Minkowski sum operations, a property only valid for the convex polygons but not for the simple ones. This property states that the Minkowski sum of two star-shaped polygons cannot have holes. Thus, the computation of Minkowski sum is reduced to calculating the outer envelope of the arrangement of the $O(mn)$ candidate edges by an angular sweepline algorithm.

With the proper data structure this operation can be done in $O(mn \log mn)$ time.
For the few cases where there are non-star-shaped polygons, a decomposition algorithm is used to decompose the polygon into a small number of star-shaped ones, with the constraint of keeping these components "glued" together.

Chapter 5

A MIP model for the Irregular Nesting Problem

I chapter 3 we presented our first approach to the problem using a bitmap representation of the pieces and the tabu search technique. One of the most evident drawbacks of that type of representation is the lack of precision in the figure shape definition. Beside all the problems in terms of speed of computation, the bitmap representation does not allow a fair comparison of the results with other algorithms.

For these reasons we adopted the polygonal representation and implemented new placement algorithms starting from scratch. In chapter 4 we explained the main concepts of computational geometry that we need to perform all operations related with polygon overlap checking and so on.

In this chapter we present a MIP model for the irregular nesting problem based on the compaction model introduced by Milenkovic et al. [LM95].

5.1 The model

We are given a set $\mathcal{P}$ of n possibly different pieces, and a rectangular marker region with fixed width $maxY$ and virtually infinite $length$. The shape of each piece is defined by a simple polygon, and the coordinates $\boldsymbol{v_i} = (x_i, y_i)$ of its reference point defines its placement location on the marker sheet. 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 5.1 for an illustration.

We can now give an introductory version of our model for the nesting problem, where ε represents a sufficiently small positive coefficient.

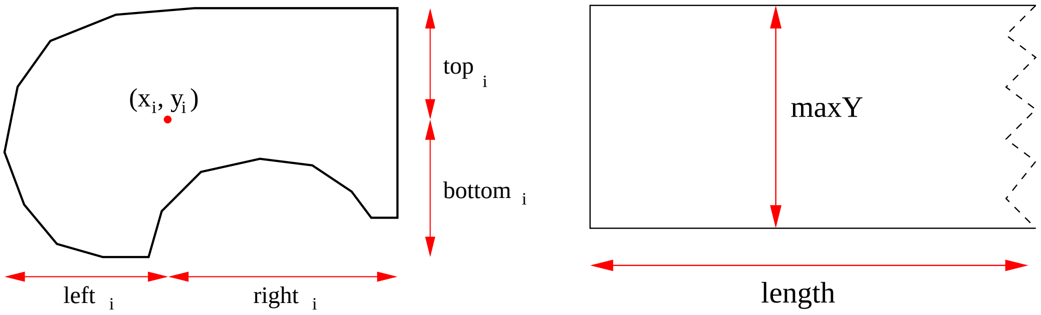


Figure 5.1: Input data for pieces (left) and marker region (right).

$$
\begin{array}{lll r}
\min & length + \varepsilon \sum_{i \in \mathcal{P}} (x_i + y_i) & & (5.1)\\
\text{s. t.} & x_i + right_i \leq length & \forall\, i \in \mathcal{P} & (5.2)\\
& left_i \leq x_i & \forall\, i \in \mathcal{P} & (5.3)\\
& bottom_i \leq y_i \leq maxY - top_i & \forall\, i \in \mathcal{P} & (5.4)\\
& \text{“Pieces } i \text{ and } j \text{ do not overlap”} & \forall\, i, j \in \mathcal{P} : i < j & (5.5)
\end{array}
$$

The main goal is to maximize the efficiency, hence to minimize the total length. In our objective function (5.1) we also consider a second goal: 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).

Constraints (5.2) defines the value of variable *length*, while constraints (5.3) and (5.4) are simple bounds on the feasible values of variables x_i and y_i, defining the position of the the reference point of each piece i.

Finally constraints (5.5), stated in words at this preliminary stage, avoid overlaps among pieces.

5.2 Using no-fit polygons

A possible way to translates constraints (5.5) makes use of the so-called no-fit polygons, as defined in section 4.2. For each pair of figures i and j, $i < j$, we calculate the no-fit polygon U_{ij}. In order for pieces i and j not to overlap, we need to enforce that their displacement vector $\boldsymbol{v_j} - \boldsymbol{v_i}$ is not contained in the no-fit polygon U_{ij}, that is conditions (5.5) translate into:

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

Since $\overline{U}_{ij}$ is the complement of a closed polygon, it is highly non convex. So we need to decompose it into polyhedral components in order to express conditions (5.5) through linear constraints. We divide $\overline{U}_{ij}$ into a set of m_{ij} disjoint 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 concave[2] edges, as illustrated in figure 5.2.

In this way we define a partition of $\overline{U}_{ij}$ into a set of disjoint polyhedra $\overline{U}^k_{ij}$, called *slices*, which satisfy:

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

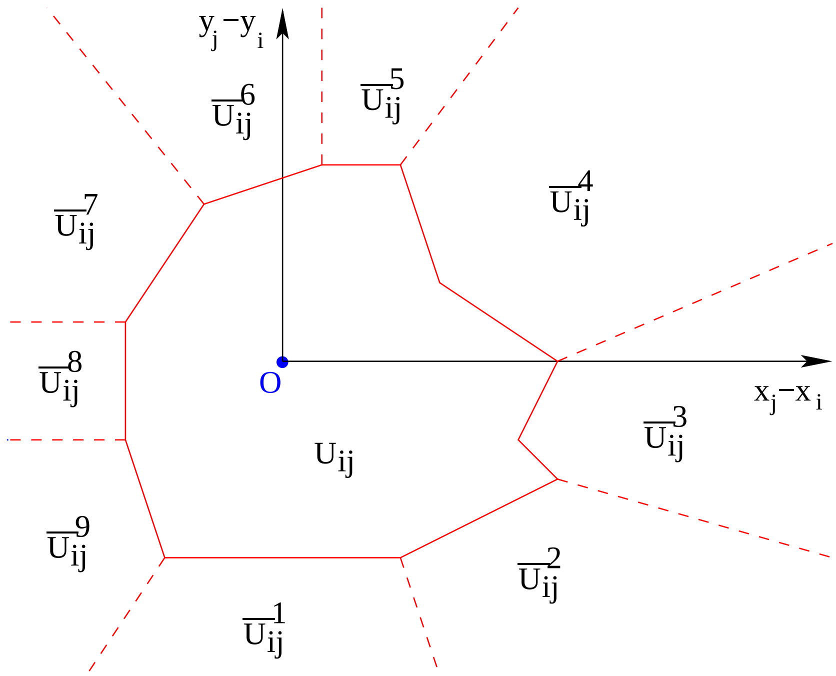


Figure 5.2: Partition of $\overline{U}_{ij}$ into polyhedral "slices".

Figure 5.2 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. This means that having a "simple" convex figure does not simplify the problem, because we may need as many components as the number of its edges.

[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.

Every slice $\overline{U}_{ij}^k$ is a polyhedron defined by linear inequalities of the form:

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

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

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

For every pair of pieces i and j, the displacement vector $\boldsymbol{v_j} - \boldsymbol{v_i}$ has to lay in one of these slices: we then introduce a binary variable for each of them:

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

and impose the constraint that exactly one among $z_{ij}^1, \ldots, z_{ij}^{m_{ij}}$ has to be set to 1 (i.e. active at the same time). Now we can express the non-overlapping condition (5.5) with the following linear constraints:

$$\begin{aligned} \boldsymbol{A_{ij}^k}(\boldsymbol{v_j} - \boldsymbol{v_i}) &\leq \boldsymbol{b_{ij}^k} + M(1 - z_{ij}^k) \cdot \mathbf{1} && \forall\, i,j \in \mathcal{P} :\ i < j,\ k = 1 \ldots m_{ij} && (5.6) \\ \sum_{k=1}^{m_{ij}} z_{ij}^k &= 1 && \forall\, i,j \in \mathcal{P} :\ i < j && (5.7) \\ z_{ij}^k &\in \{0,1\} && \forall\, i,j \in \mathcal{P} :\ i < j,\ k = 1 \ldots m_{ij} && (5.8) \end{aligned}$$

where M is a sufficiently large positive value (big-M value).

As can be easily seen from the formulation, constraints (5.6) are active if and only if the corresponding binary variable has value 1; otherwise the big-M factor deactivates the whole set of constraints.

It is worth noting that the number of these constraints grows rapidly with the input size (number of pieces): there are at least 3 facets for each slice and the number of slices is linear in the number of edges of the no-fit polygon. The relation between edges of the no-fit polygon and edges of the relative pieces depends on the type of polygons, as explained in section 4.4. Since each piece can interact with any other piece, the number of such no-fit polygons is quadratic in the input size.

5.3 Interpretation of the relaxed model

As we have seen in the previous section, 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 the standard branch and bound technique, the integer (binary in this case) requirement is relaxed and the corresponding linear formulation is solved.

In this case, for each pair i, j we are likely to have more than one z_{ij}^k variables with positive values (while their sum is still equal to 1), thus admitting solutions which are not feasible in the original (integer) formulation.

The geometric interpretation of this fact is that with fractional values of z_{ij}^k, the linearized model considers as feasible a wider region, that covers part or even all of the original no-fit polygon U_{ij}.

This behaviour is emphasized by the presence of the big-M factor which can deactivate the whole constraint even for values of the z_{ij}^k close to 1 but not integer.

For example the choice $z_{ij}^1 = z_{ij}^2 = \frac{1}{2}$ in (5.6), makes feasible almost all $\boldsymbol{v_j} - \boldsymbol{v_i}$, including the total overlap position $\boldsymbol{v_j} = \boldsymbol{v_i}$, since $M >> \max_{k=1\ldots m_{ij}}\{|b_{ij}^k|\}$.

5.4 Lifting of constraint coefficients

To reduce the negative effect of the big-M factor on the linear relaxation of the model, we applied a *lifting* technique [NW88] on the constraint coefficients. This methodology consists of eliminating the big-M factor and substituting it with the a series of minimum values that guarantee the constraints to be valid.

Let us concentrate on a single convex slice k for the pair of pieces i and j; we have (say) t_{ij}^k facets defining this region, each of them defined by a constraint of the form:

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

We substitute the big-M factor with a new set of terms, one for each binary variable related to the pair of pieces i and j:

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

By exploiting the fact that $\sum_{h=1}^{m_{ij}} z_{ij}^h = 1$, we can replace the constant term γ_{ij}^{kf} by $\gamma_{ij}^{kf} \sum_{h=1}^{m_{ij}} z_{ij}^h$ thus obtaining the general form:

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

where $\delta_{ij}^{kfh} = \gamma_{ij}^{kf} + \theta_{ij}^{kfh}$.

As already said, when the slice 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 the maximum value of the left-hand side when variable z_{ij}^h has value 1. Formally:

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

that can be computed by inspection on the vertices of the closed region $\overline{U}_{ij}^{h} \cap B$, where B is a sufficiently large bounding box for the feasible displacement vector $\boldsymbol{v_j} - \boldsymbol{v_i}$, e.g. a rectangle with height $2 \cdot maxY$ and length $2 \cdot maxX$ (where $maxX$ is an upper limit on the marker length). Figure 5.3 shows the vertices of slice $\overline{U}_{ij}^{h}$ where to evaluate the lifted coefficient δ_{ij}^{kfh} of variable z_{ij}^{h} with respect to a generic facet f of slice $\overline{U}_{ij}^{k}$, namely $\alpha_{ij}^{kf}(x_j - x_i) + \beta_{ij}^{kf}(y_j - y_i) = \delta_{ij}^{kfh}$ (in dashed bold line in the figure).

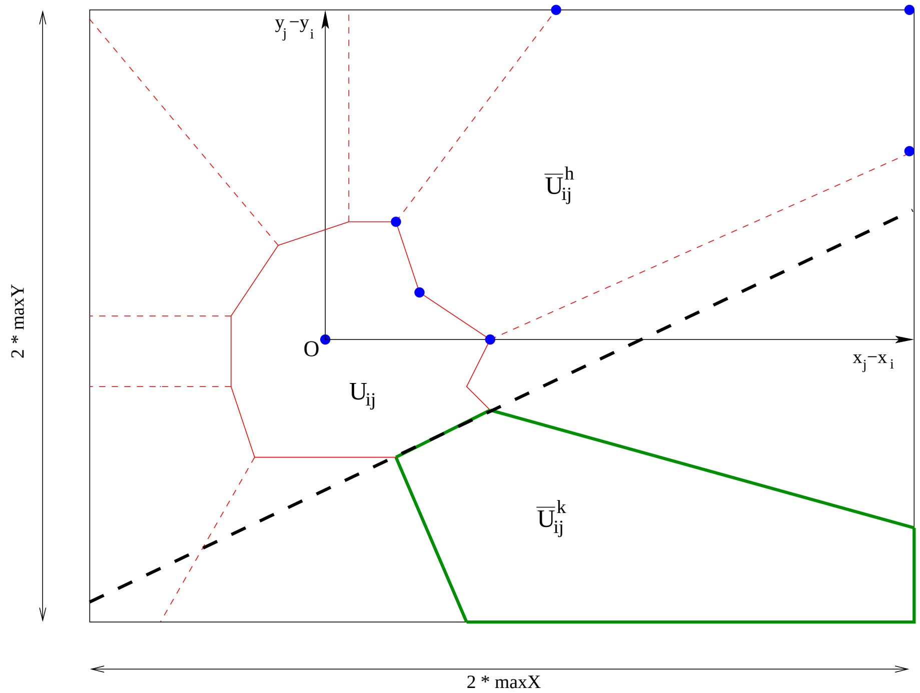


Figure 5.3: Lifting of shaded line coefficients of slice k into vertices of slice h.

The enhanced model can now be rewritten as follows.

$$\min \quad length + \varepsilon \sum_{i \in \mathcal{P}} (x_i + y_i) \tag{5.9}$$

$$\text{s. t.} \quad x_i + right_i \leq length \qquad \forall\, i \in \mathcal{P} \tag{5.10}$$

$$left_i \leq x_i \qquad \forall\, i \in \mathcal{P} \tag{5.11}$$

$$bottom_i \leq y_i \leq maxY - top_i \qquad \forall\, i \in \mathcal{P} \tag{5.12}$$

$$\alpha_{ij}^{kf}(x_j - x_i) + \beta_{ij}^{kf}(y_j - y_i) \leq \sum_{h=1}^{m_{ij}} \delta_{ij}^{kfh}\, z_{ij}^{h} \tag{5.13}$$

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

$$\sum_{k=1}^{m_{ij}} z_{ij}^{k} = 1 \qquad \forall\, i,j \in \mathcal{P}:\ i < j \tag{5.14}$$

$$z_{ij}^{k} \in \{0,1\} \qquad \forall\, i,j \in \mathcal{P}:\ i < j,\ k = 1 \ldots m_{ij} \tag{5.15}$$

5.5 Guiding the search tree

In section 5.3 we explained the weakness of the relaxed formulation and its consequences. One of the main drawbacks is the heavy use of branch and bound to get integer values for the binary variables. The meaning of variable z_{ij}^k set to 1 is that pieces i and j assume a feasible relative position. When dealing with more than two pieces, the relative positions among them can easily become inconsistent. Let us make an example with three pieces A, B and C stacked one above the other in the given order as shown in figure 5.4 (B lies above A, and C lies above both A and B).

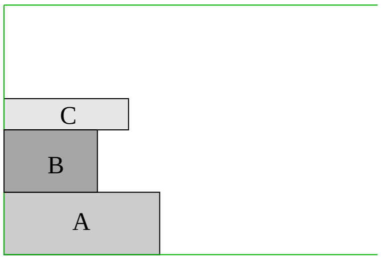


Figure 5.4: Assigned positions of three pieces A, B, and C.

For simplicity of notation, let us say that z_{AB}^{up} represents the variable related to the feasible region having piece B above (up) A. Suppose we run a branch and bound scheme to get to this situation: we first fix the value of variable $z_{AB}^{up} = 1$, meaning that piece B lies above A. Then we set variable $z_{BC}^{up} = 1$, meaning that piece C lies above B. Now we get to the relation between piece A and C. Here the consistent choice would be to fix $z_{AC}^{up} = 1$; any other settings will lead to an infeasible solution. However, for the relaxed model the choice $z_{AC}^{up} < 1$ is also feasible.

Suppose now that the branching strategy only sets the relation between A and B ($z_{AB}^{up} = 1$), and A and C ($z_{AC}^{up} = 1$), but not the relation between pieces B and C. In this case two situations may occur, as shown in figure 5.5.

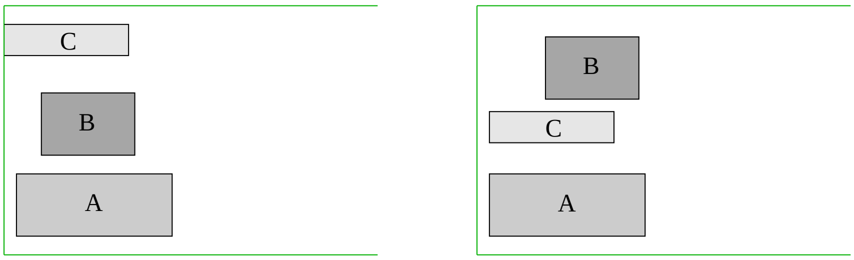


Figure 5.5: Possible relative positions of three pieces A, B, and C, when only two relative positions are specified.

If in the branching scheme we proceeded directly with branching on other pieces

(D, E, ...), without fixing $z^{up}_{AC} = 1$, we might get to infeasible solutions further down in the search tree that could have been detected before.

For this reason, 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. In other words, we tend to branch so as to completely determine first the relative positions (slices) 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.

We impose such a scheme through the use of priority of branching among variables: variables with higher priority are branched first.

We assign branching priorities to variables in the following way:

$\psi = 10.000$;
$\mathcal{S} = \emptyset$;
while $\mathcal{P} \neq \emptyset$ **do**
    select a piece $p \in \mathcal{P}$;
    for all $q \in \mathcal{S}$ **do**
        k = 0
        for all slices $\overline{U}^k_{pq}$ **do**
            assign priority ψ to variable z^k_{pq} ;
            $\psi = \psi - 1$;
            $k = k + 1$;
        end do
    end do
    $\mathcal{S} = \mathcal{S} + p$;
    $\mathcal{P} = \mathcal{P} - p$;
end do

Figure 5.6: Algorithm for the branching priority assignment.

5.6 Computational results

As already explained, the size of the model grows very rapidly with the input size. Thus this model can only be used for instances of limited size.

We created some *broken glasses* instances, where a square region is broken at random into n polygonal pieces. 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. Figures 5.7 shows some of these instances.

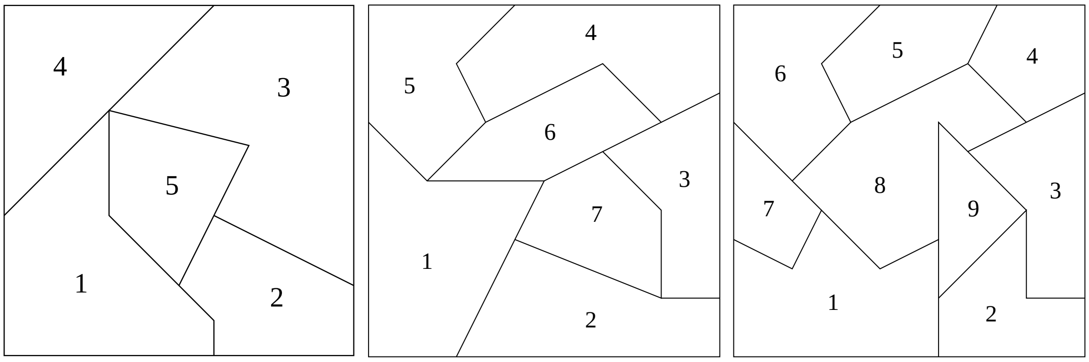


Figure 5.7: Broken glass instances.

The computational results[3] on the instances above are reported in table 5.1.

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 5.1: Computational results of the MIP model for the irregular nesting problem.

For every instance, we report:

- the name of the instance;
- the number of pieces involved;
- the number of integer (binary) variables;
- the flag for activating the priority branching rule;
- the total number of nodes in the branch and bound tree;
- the time to solve the model (in CPU seconds);
- the gap between the best solution found and the best lower bound in case the solver reached the maximum number of nodes allowed (100,000).

As one can see, only the smaller instances could be solved to optimality within the given node limit, while for the bigger ones, we could not get the optimum. Furthermore,

[3] Solved with CPLEX 7.0 on a AMD Athlon 1.2 GHz with 512 Mbyte RAM

the use of the branching priorities to guide the construction of the search tree, helps the solver considerably; indeed, the second instance (Glass 2) could only be solved to optimality using this strategy.

Chapter 6

A MIP model for the Multiple Containment Problem

6.1 The Multiple Containment Problem

As explained in chapter 2, the usual way to build a layout is to place the "big pieces" first and then to insert the remaining ones in the holes left by the big pieces. In this chapter we deal with this second problem, called *multiple containment problem*: given a set $\mathcal{P}$ of n possibly different "small" pieces called *trims*, and a set $\mathcal{H}$ of m irregular polygons called *holes*, find the best assignment of pieces into holes and their relative displacement vectors, such that the maximum number (or total area) of pieces are placed and the maximum hole area[1] is used. Note that in this problem we are not requested to place all pieces, nor to use all holes, but to chose the holes and pieces that maximize our objective function. To clarify this concept, let us make an example. If we have many trims and a few holes we have to decide which pieces to place and in which holes, and it may happen that not all pieces are placed. Conversely, if we have a few pieces and many holes, we want to fill the holes as much as possible, so we have to decide which holes to use and which pieces to place in the used holes.

A first approach to this problem is to use a greedy strategy: take a piece at a time and scan all the available holes starting from the smallest one; once you find a hole that can fit the piece, place it using the given placement policy (e.g. bottom-left). This strategy has two main drawbacks: the first is that any placement policy can compromise the free space for the other pieces, as already discussed in chapter 3; the second one is that assigning a piece to a certain hole in a greedy fashion, can compromise the global objective function.

For example, suppose we only want to consider the area of pieces and holes without any geometrical consideration. We are given two holes (h_1, h_2), the first with area 90 and

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

the second with area 95, and a set of 4 pieces (p_1, p_2, p_3, p_4) with area 85, 50, 40 and 10, respectively. A greedy strategy would place the biggest piece p_1 in the smallest hole h_1, then pieces p_2 and p_3 in hole h_2; at this point piece 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 smart 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 pieces with no wasted area at all.

This simplified problem is well known in the Operation Research community and is called the *knapsack problem*. However, when the geometrical considerations are taken into account, the problem becomes very difficult and it is not easy to write an effective mathematical model to represent it.

6.2 Geometrical considerations

We observed that trim pieces can very often be well approximated by their bounding box (although in some cases, the real piece covers less than 50% of its bounding box area). In this way we only have to deal with the interaction among rectangles, and thus only have to check for vertical and horizontal interaction, thus avoiding the nasty problem of inter-penetration due to non convexity of the pieces. Figure 6.1 shows the set of pieces that composes a shirt: on the left the big pieces, on the right the small ones and their corresponding bounding box.

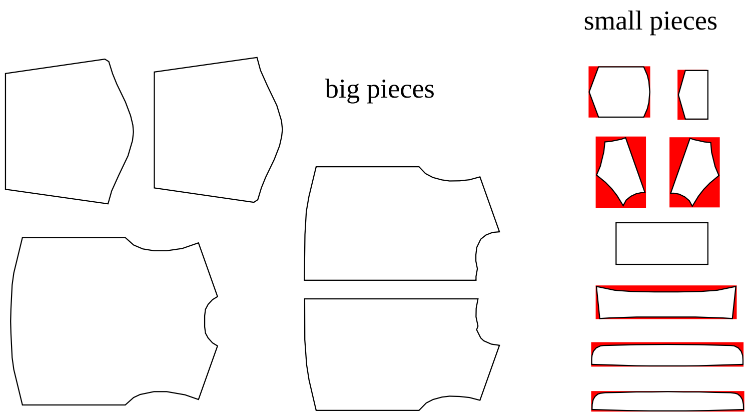


Figure 6.1: Big and small pieces of a shirt.

Another important consideration is that the set of holes comes from the geometrical difference of the stock sheet and the union of already placed “big” pieces. Depending on the exact position of these pieces, this difference may consist of distinct disjoint polygons, or one or more polygonal components connected together by narrow strips (see the dark grey region of figure 6.2). Unfortunately, when the holes are too small or the strips are too narrow, they are not useful, since no pieces can fit into them. Actually, what we are really interesting in, is not the original shapes of the geometrical difference, but their usable part, e.g., the area of the holes that can be occupied by the given trims.

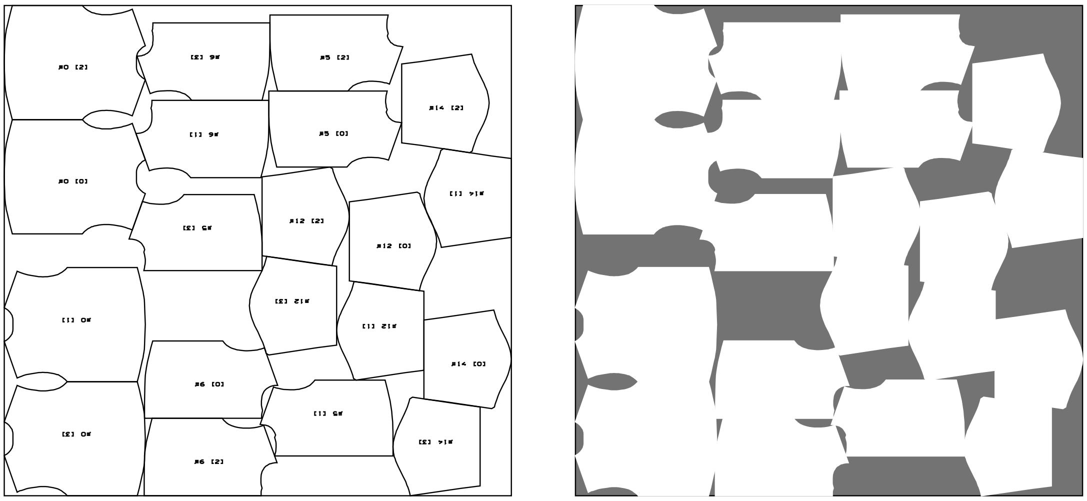


Figure 6.2: Placement of big pieces, and relative holes.

Each piece has its own dimensions, therefore a hole that can contain a certain piece, may not be useful for another one. In our simplification process we build a rectangle with dimension equal to the minimum length and height of all trim pieces, and call it the *minBox*. (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 that we fixed heuristically to 0.8). Now using the Minkowski difference procedure illustrated in chapter 4 we calculate the free movement region of our minBox within the original holes, thus defining a set of so-called *usable holes*. Figure 6.3 shows in light grey the

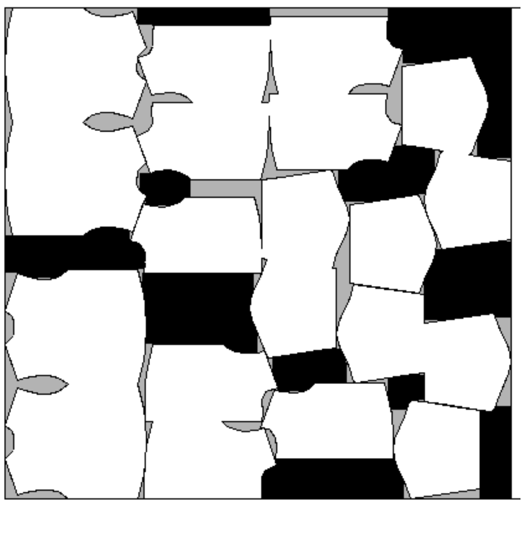


Figure 6.3: Original holes and their usable region.

original holes and, in black, the usable holes.

6.3 The grid

Once we have derived the usable holes, we can start building their relative grid in the following way. First we calculate the average length and height of the trims (*aveLength* and *aveHeight*); then for each hole[2], we take its bounding box and map it into a grid where each cell has dimensions aveLength and aveHeigth, respectively.

For every row we draw a horizontal line through its center point and store the start and end points where the line intersects the figure; if the shape 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 such internal segments. In a similar way, we draw vertical lines through the column centers and obtain the start and end points of the columns together with their actual length.

Figure 6.4 shows an example of a specific hole and its relative measurement points: the dots indicate the starting and ending points of the internal segments; in this example, row 3 is the only one made up of two distinct segments.

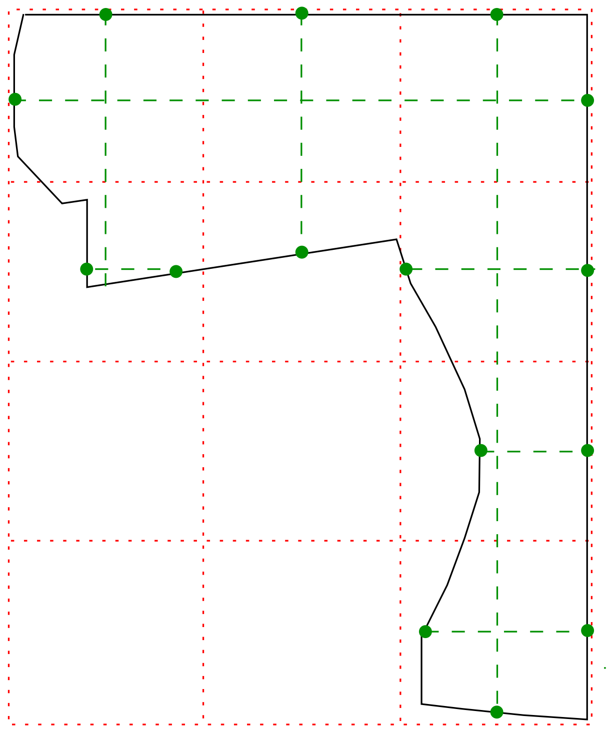


Figure 6.4: Grid of a specific hole and relative measurement points.

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

6.4 The model

Now we have all the information we need to start building the model. We are given a set $\mathcal{P}$ of n possibly different trims, and a set $\mathcal{H}$ of m (usable) holes. For each hole $h \in \mathcal{H}$ we compute:

- $holeArea$: the usable area of the hole;
- $origX_h, origY_h$: the coordinate of the reference point (lower left point) of the hole bounding box;
- $holeLength_h, holeWidth_h$: the hole bounding box dimensions;
- $cellLength_h, cellWidth_h$: the average length and height of each cell in the hole (it may be different from hole to hole because of a rounding factor);
- R_h, C_h: the number of rows and columns in the hole.

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, respectively:

$$\left.\begin{array}{l} rowStart^h_r \\ rowEnd^h_r \\ rowLength^h_r \end{array}\right\} \;\forall\, h \in \mathcal{H},\; r = 1 \dots R_h \qquad \left.\begin{array}{l} colStart^h_c \\ colEnd^h_c \\ colWidth^h_c \end{array}\right\} \;\forall\, h \in \mathcal{H},\; c = 1 \dots C_h$$

To define our model we need the following variables:

- $U^h \in \{0,1\} \qquad \forall\, h \in \mathcal{H}$
  a binary variable for each hole (= 1 whenever the hole is active, e.g. it is used to place some pieces);
- $X^h_{rc},\; Y^h_{rc} \qquad \forall\, h \in \mathcal{H},\; r = 1 \dots R_h,\; c = 1 \dots C_h$
  real positive variables defining the exact coordinate position of the lower left point of the bounding box of each piece inside cell h;
- $Z^{hp}_{rc} \in \{0,1\} \qquad \forall\, h \in \mathcal{H},\; p \in \mathcal{P},\; r = 1 \dots R_h,\; c = 1 \dots C_h$
  a binary variable to decide if piece p is placed in cell h.

We can now formulate the whole model as:

$$
\begin{aligned}
\min \quad & \sum_{h\in\mathcal{H}} \Big(holeArea_h \cdot U^h - \sum_{r=1}^{R_h}\sum_{c=1}^{C_h}\sum_{p\in\mathcal{P}} pieceArea_p \cdot Z_{rc}^{hp}\Big) \\
& \quad + \ \varepsilon \sum_{r=1}^{R_h}\sum_{c=1}^{C_h}\sum_{h\in\mathcal{H}} (X_{rc}^h + Y_{rc}^h) && (6.1)\\
\text{s. t.} \quad & \sum_{p\in\mathcal{P}} Z_{rc}^{hp} \ \le \ U^h \qquad \forall\, h\in\mathcal{H},\ r=1\ldots R_h,\ c=1\ldots C_h && (6.2)\\
& \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} && (6.3)\\
& 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\ldots R_h,\ c=1\ldots C_h-1 && (6.4)\\
& 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\ldots R_h,\ c=C_h && (6.5)\\
& \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\ldots R_h && (6.6)\\
& Y_{rc}^h + \sum_{p\in\mathcal{P}} width_p\ Z_{rc}^{hp} \ \le \ Y_{r+1,\,c} \\
& \qquad \forall\, h\in\mathcal{H},\ r=1\ldots R_h-1,\ c=1\ldots C_h && (6.7)\\
& Y_{rc}^h + \sum_{p\in\mathcal{P}} width_p\ Z_{rc}^{hp} \ \le \ colEnd_c^h \\
& \qquad \forall\, h\in\mathcal{H},\ r=R_h,\ c=1\ldots C_h && (6.8)\\
& \sum_{p\in\mathcal{P}} width_p \sum_{r=1}^{R_h} Z_{rc}^{hp} \ \le \ colWidth_c^h \\
& \qquad \forall\, h\in\mathcal{H},\ c=1\ldots C_h && (6.9)\\
& \max(rowStart_r^h,\ \ origX_h + (c-1)\cdot cellLength_h) \ \le \ X_{rc}^h \\
& \quad \le \ \min(rowEnd_r^h,\ \ origX_h + (c)\cdot cellLength_h) \\
& \qquad \forall\, h\in\mathcal{H},\ \ r=1\ldots R_h,\ c=1\ldots C_h && (6.10)\\
& \max(colStart_c^h,\ \ origY_h + (r-1)\cdot cellWidth_h) \ \le \ Y_{rc}^h \\
& \quad \le \ \min(colEnd_c^h,\ \ origY_h + r\cdot cellWidth_h) \\
& \qquad \forall\, h\in\mathcal{H},\ \ r=1\ldots R_h,\ c=1\ldots C_h && (6.11)\\
& U^h \in \{0,1\} \qquad \forall\, h\in\mathcal{H} && (6.12)\\
& Z_{rc}^{hp} \in \{0,1\} \qquad \forall\, h\in\mathcal{H},\ p\in\mathcal{P},\ r=1\ldots R_h,\ c=1\ldots C_h && (6.13)
\end{aligned}
$$

As already mentioned, our main goal is to minimize the unused area in each active hole; furthermore, the second term of the objective function (6.1) minimizes the placement position of the trims within the cell.

Constraints (6.2) activate or deactivate all the selection variables of a single cell, depending on the value of the relative hole activation variable U^h. Furthermore, since U^h is a binary variable, when active (equal to 1) this constraint imposes that only one piece can be assigned to a single cell.

Constraints (6.3) are typical *capacity knapsack constraints*: in this case they refer to the area and impose that the total area of all the pieces placed in a certain hole cannot exceed the area of the hole itself.

Constraints (6.4) impose that any piece in cell $(r, c+1)$ must have x-coordinate greater or equal to the rightmost point of the piece positioned in cell (r, c), while constraints (6.5) impose a bound on the rightmost point of the piece positioned in the last column of each cell. The next set of constraints (6.6) are again a sort of capacity knapsack constraints, but refer to the total length of each row: the sum of lengths of all the pieces placed in all cells of a certain row cannot exceed the length of that row.

The next three sets of constraints (6.7), (6.8) and (6.9), are the equivalents on columns of the last three sets of constraints on rows; the limit the relative y-coordinate of each cell in a column and limit the total width of each column.

Finally constraints (6.10) and (6.11) impose lower and upper bounds respectively to the x-coordinate and y-coordinate of each cell.

6.5 Converting a solution of the model to a feasible layout

Once the model has been solved we obtain the list of active holes, and the position of the chosen pieces within those holes. Now we can replace the bounding boxes that we have used with their original shape. Since the model deals with rectangles, it does not take into account the rotation of the pieces, but only their position. As a result, when we replace the original shapes, we must assign one of the possible rotations to each piece. For practical reasons, when possible, we assign alternative rotations to consecutive pieces with the same shape, e.g., if we place two consecutive copies of the same piece we assign rotation 0 to the first and rotation 180° to the second, and so on.

The solution of our model suffers from all the geometrical simplifications that we applied in order to make it solvable. First of all, we consider the bounding box of the trims, instead of their original shape, thus increasing the effective area; then we use a reduction coefficient on the dimension to compensate somehow this choice. Furthermore we only check the piece interaction on the lines through the cell centers, but we do not consider any other possible intersection.

For all these reasons, when we actually replace the bounding boxes by the original shapes, we might get a solution with many overlapping pieces, both among trims and among trims and big pieces. Figure 6.8 shows the solution of our model to the instance of figure 6.2: usable holes are drawn in light grey with black borders and their corresponding cell centers are marked by a cross; placed trim pieces are drawn in dark gray. As the figure clearly shows, there are many overlapping pieces, and many others extend outside the assigned hole, although the model considers this a feasible solution.

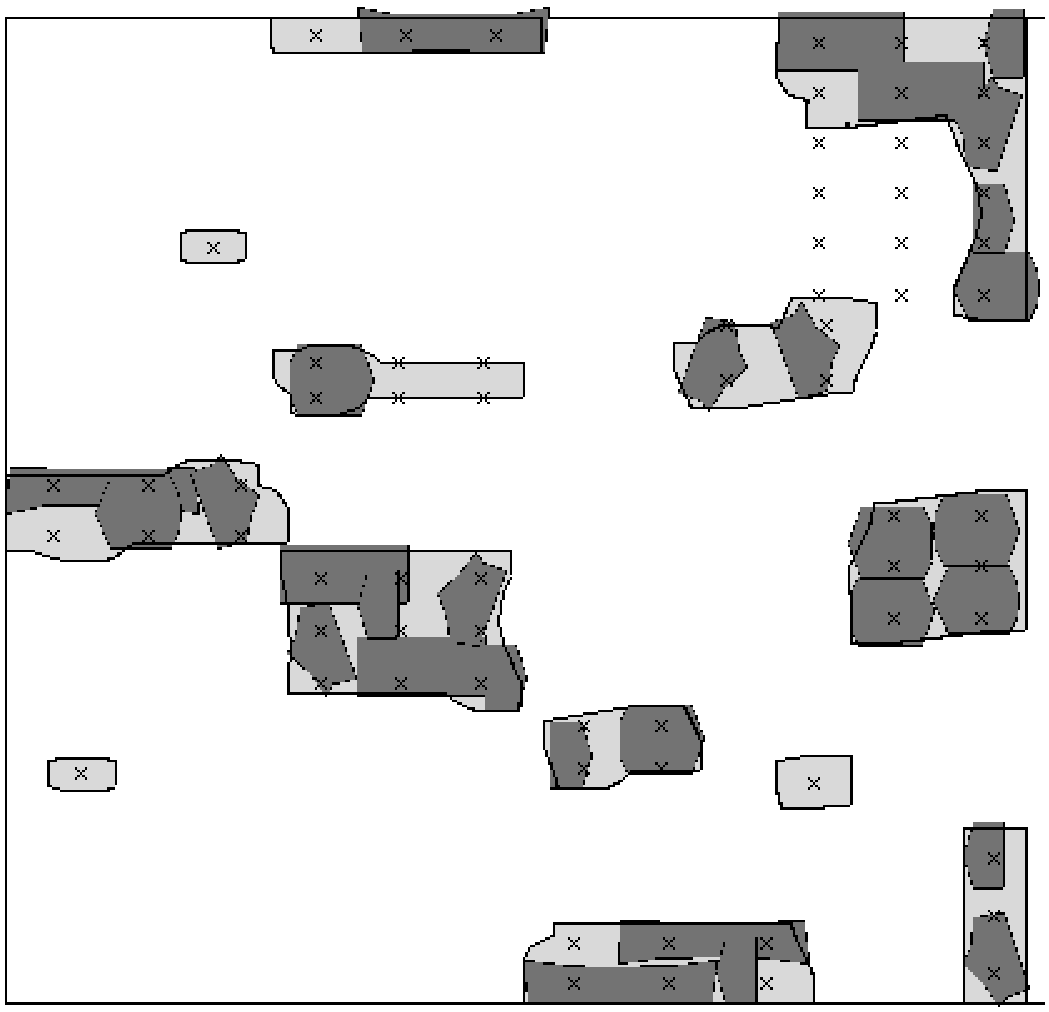


Figure 6.5: Trim pieces allocated by the model.

The main reason of overlap is that there are no geometrical relations imposed among pieces assigned to cells of different rows/columns. Figure 6.6 represents an example that shows the weakness of the model; as before, cell centers are marked with a cross, and are labeled 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 length is less than 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 width. Piece B and piece C occupy the second column and second row, respectively. They both satisfy all model constraints, since they lay in different rows and columns, but they clearly intersect.

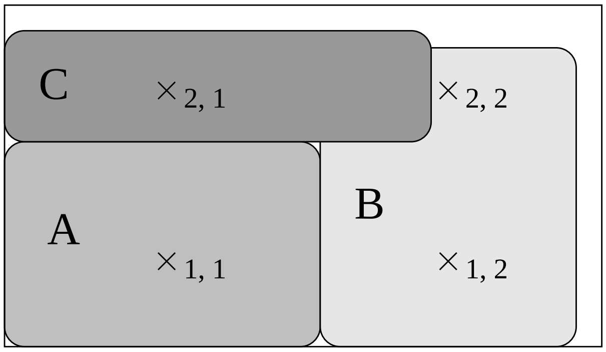
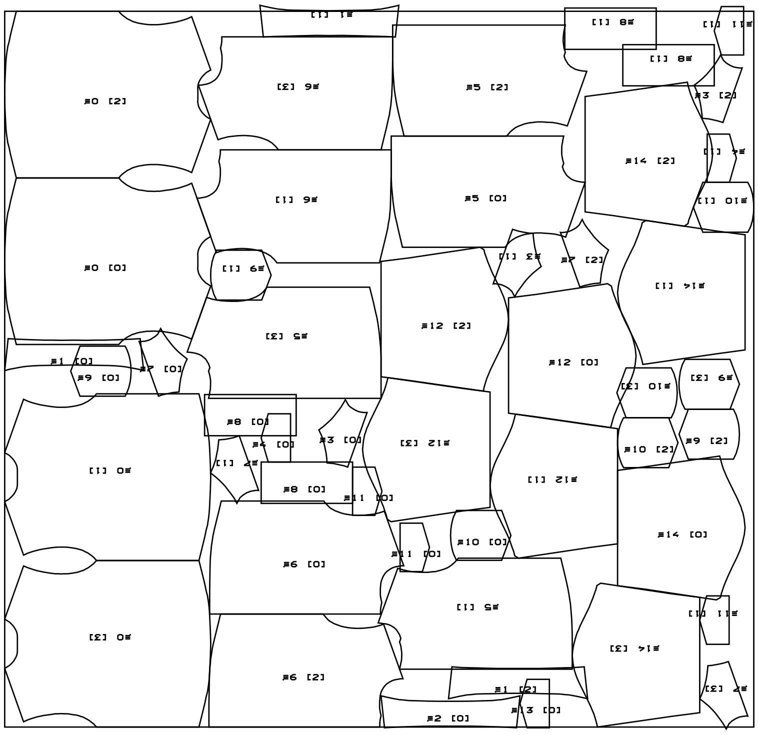


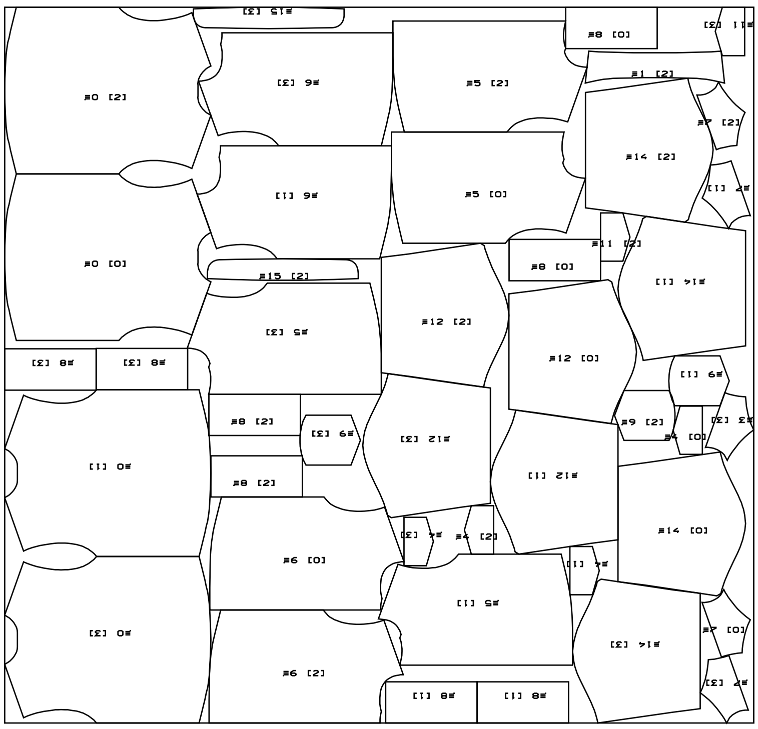
Figure 6.6: Weakness in the geometrical constraints of the model.

It is not possible to take effectively into account in our model all geometrical interactions related to the original shape and dimension of the pieces. What we can do, instead, is to try to correct the solution obtained by our model by eliminating the overlap among figures. This process can be done using an algorithm introduced by Li and Milenkovic [LM95], based on a linear version of the model that we analyzed in chapter 5. In this case, all pieces are already placed, and we know the relative positions of each pair of pieces; thus we can fix the variable which selects a slice in the complement of the corresponding no-fit polygon (or, equivalently, include in the model only the constraints relative to that particular slice). The resulting model is linear, and therefore very easy to solve (if a feasible solution exists). We use this model to separate overlapping pieces and obtain (if possible) the feasibility of the solution. Unfortunately, the separation routine cannot always guarantee to find a feasible solution, because pieces overlap too much, or because they cannot move beyond the stock sheet borders. Figure 6.7 shows the solution obtained by our model, and the corresponding solution obtained using a greedy strategy. Although the efficiency of the model solution is higher than the greedy one, the figure shows many overlapping pieces, and, for this instance, the separation algorithm fails to find a feasible solution. In these cases, a possible way to correct the solution is to remove, one by one, the pieces that present the largest overlap, until the solution becomes feasible.

Another possibility is to divide the trims into groups with similar dimensions, and to run the model separately for each group. Indeed, we observed that when the pieces involved in the model have more or less the same shape and dimension, the geometrical problems discussed above are not likely to occur, or, if they occur, the overlap induced



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



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

Figure 6.7: Infeasible solution obtained by our model (top), and a corresponding feasible solution obtained by a greedy strategy (bottom).

is small and is usually repared by our separation algorithm. If we apply this technique to our instance, we obtain the solution of figure 6.8; this solution also presents some overlap, but this time the separation algorithm is able to remove them.

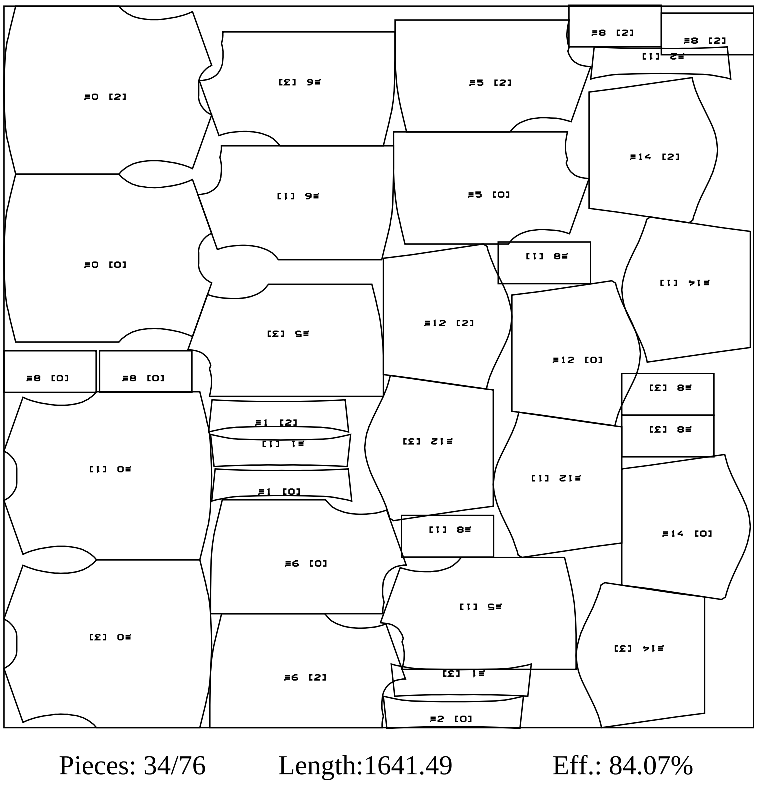
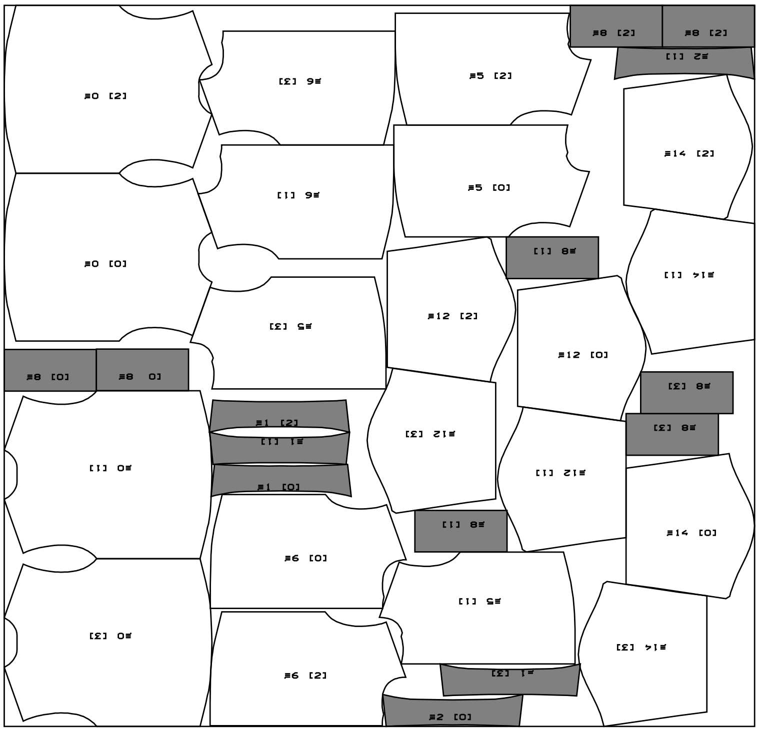


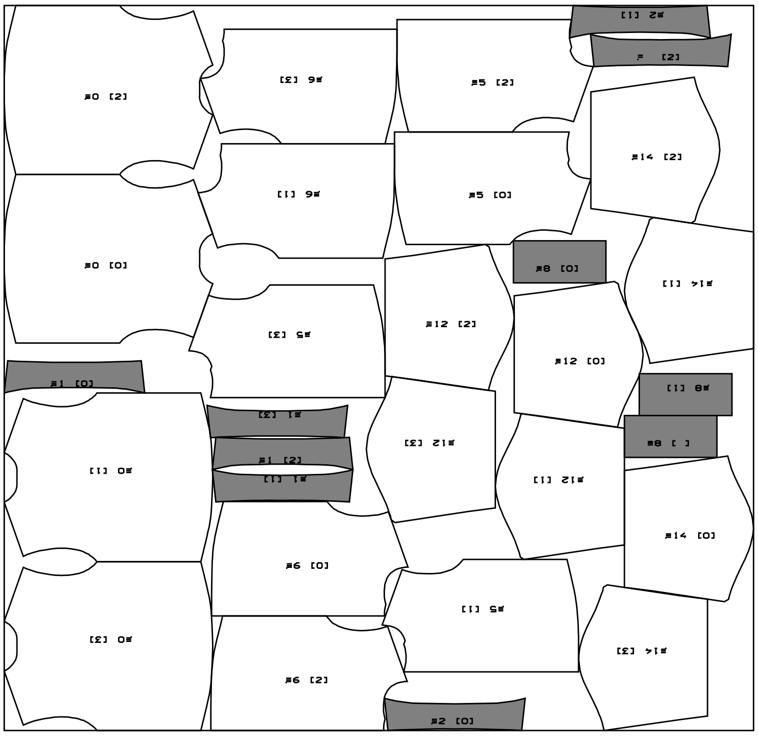
Figure 6.8: Solution obtained by our model for a group of "special" trims.

Figure 6.9 shows the final feasible solution obtained by our model after applying the separation routine, and the solution obtained by the greedy algorithm. The model placed 14 pieces compared to the 10 pieces placed by the greedy algorithm; furthermore, 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 groups of trims, until no more pieces can be fit. In figure 6.10 a second group of "small" pieces are placed with both algorithms. The solution proposed by the model and successively corrected by the 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%.



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



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

Figure 6.9: Feasible solution obtained by our model (top), and a corresponding feasible solution obtained by a greedy strategy (bottom), for a group of “special” trims.

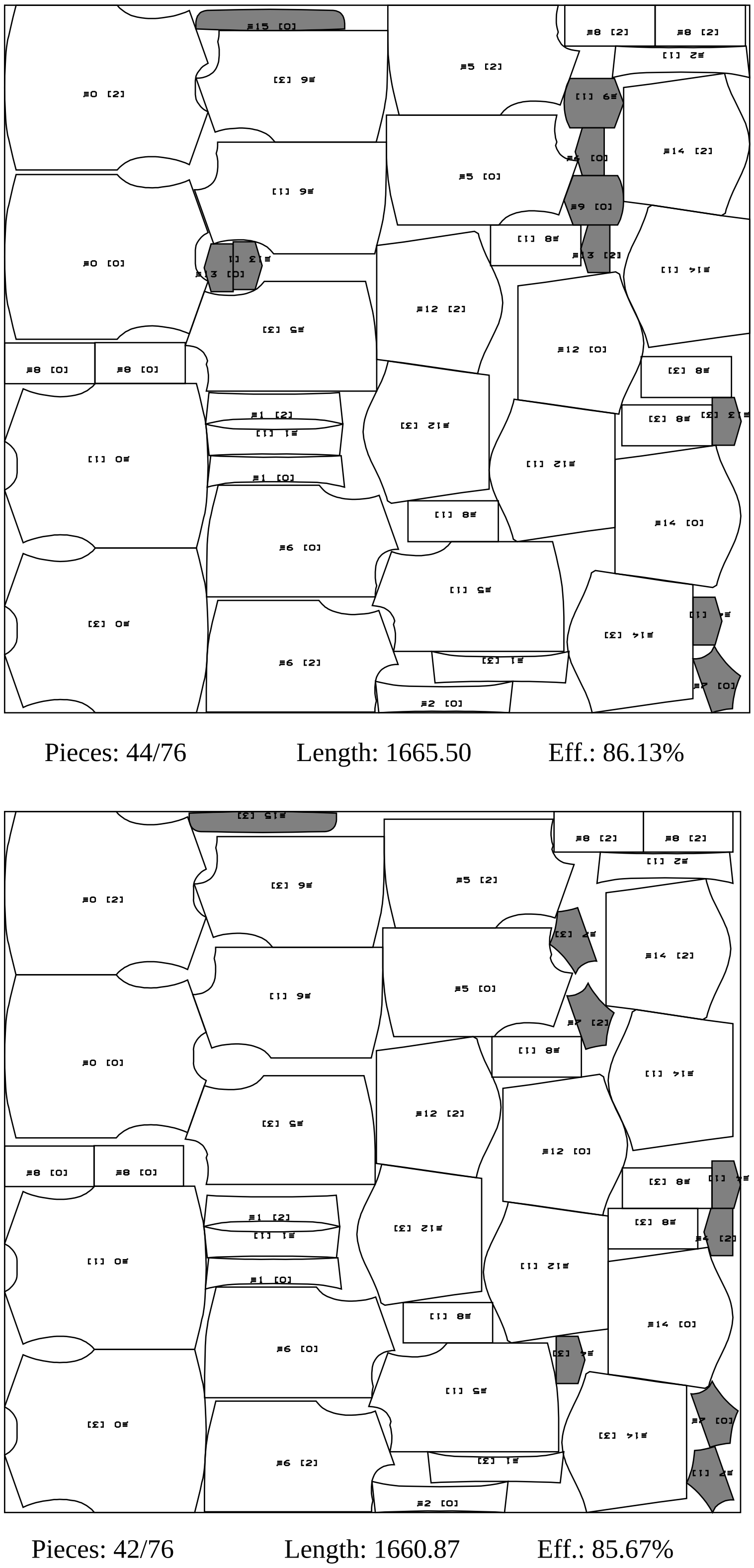


Figure 6.10: Feasible solution obtained by our model (top), and a corresponding feasible solution obtained by a greedy strategy (bottom), for a group of "small" trims.

6.6 Computational results

In our algorithm, we use the solution provided by the model for the multiple containment problem as a first solution for the placement of the trims; we then run the separation routine to remove overlap among pieces: we call this algorithm the *smart placement algorithm*.

We test this algorithm on a set of instances taken from the garment industry, and compare them to those obtained by the greedy algorithm.

The solution obtained are shown in the next figures, and the computational results[3] are reported in table 6.1

The running time of the algorithm for building and solving the model depends on the number of different pieces, and the number and size of the holes. For most of the instances, this time goes from less than 1 second to a few seconds, and is definitively comparable to the time needed by the greedy algorithm.

INSTANCE	PIECES	TRIMS	LENGTH	EFFICIENCY
82 - group 1				
smart	34/76	14	1643.53	83.97%
greedy	30/76	10	1634.55	81.54%
82 - group 2				
smart	44/76	10	1665.50	86.13%
greedy	42/76	8	1660.87	85.67%
101				
smart	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 6.1: Computational results of the algorithm for the multiple containment problem.

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

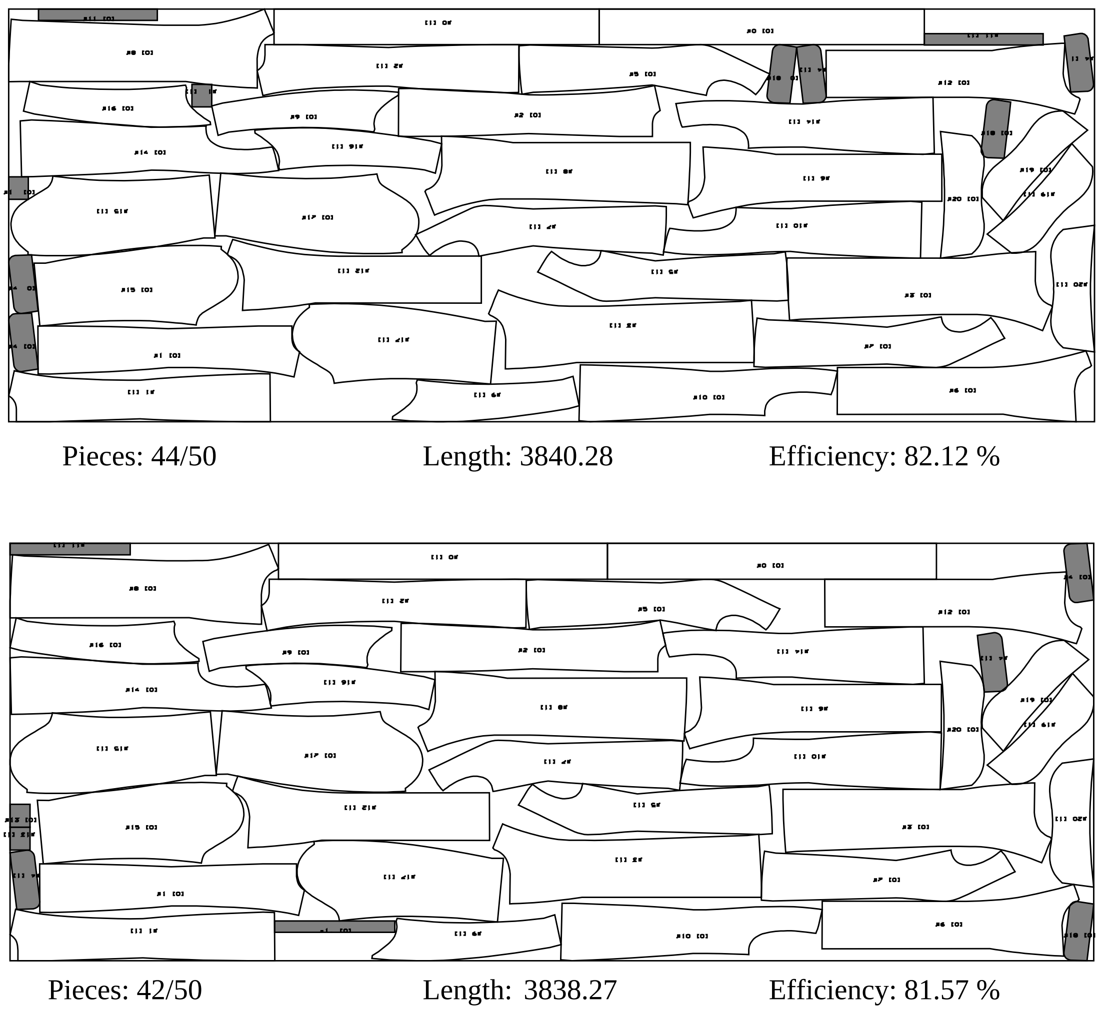


Figure 6.11: Feasible solution obtained by our model (top), and a corresponding feasible solution obtained by a greedy strategy (bottom) for problem instance 101.

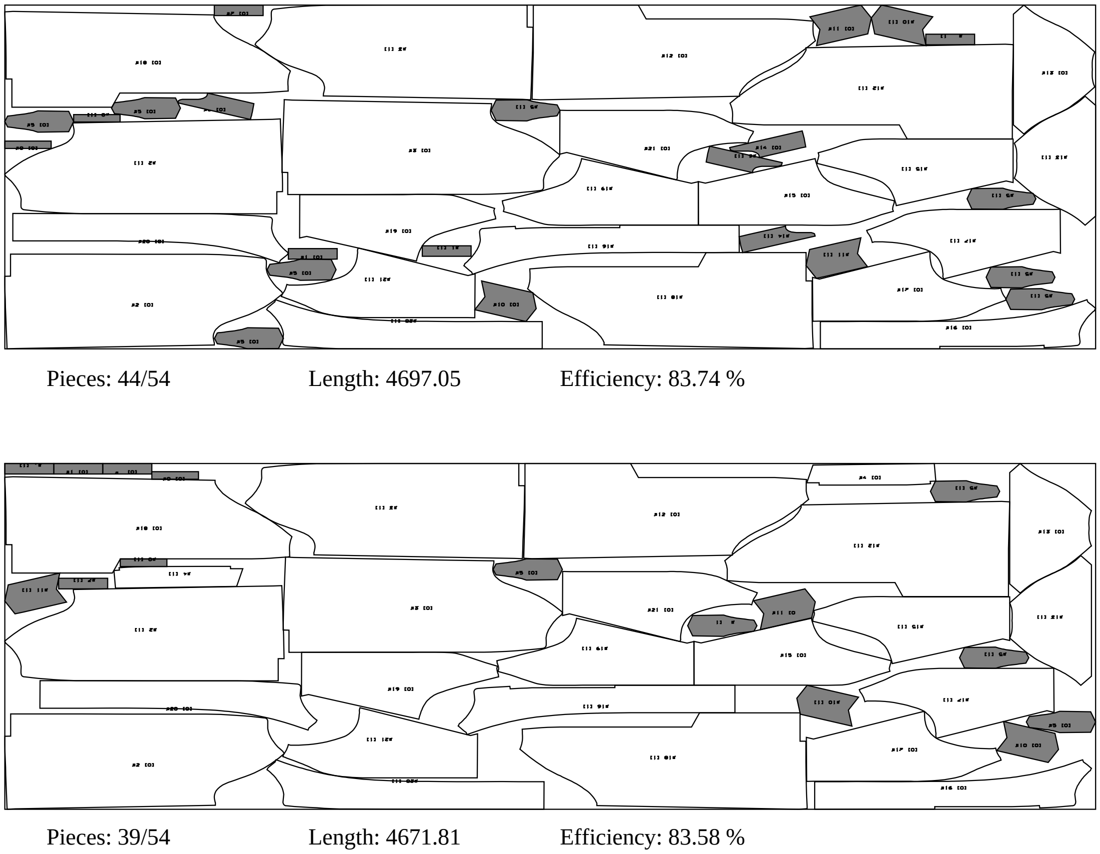


Figure 6.12: Feasible solution obtained by our model (top), and a corresponding feasible solution obtained by a greedy strategy (bottom) for problem instance 385.

Chapter 7

Conclusion

In this thesis we analyzed the *irregular nesting problem*, which belongs to the general class of *cutting & packing* problems. This problem is strongly NP-hard, and the geometrical aspects that characterize it makes it really hard to solve in practice.

We first gave an overview of the different approaches that have been taken in the literature to deal with this problem in the various industrial fields where it arises.

In chapter 3 a first approach to the problem is analyzed: pieces are represented by bitmaps, and the overlap check is done via straightforward bitwise comparison. We designed, implemented and tested a new algorithm for the automatic nesting of irregular pieces: different strategies were used to find an initial solution, and a tabu search technique has been used to explore neighbor solutions. This approach was easy to implement and produced good quality solutions.

Furthermore, we introduced a new alternative bitmap representation and an efficient technique to check for overlap, which allowed to reduce the execution time by a factor 3. We investigated the behaviour of the algorithm with different tabu parameter, and applied the concepts of “intensification” and “diversification” of the search to this particular problem.

We then switched to a different approach, where figures are represented by polygons defined by the corresponding vertices. In chapter 4 we described all the computational geometry aspects involved in this approach, in particular the concept of *no-fit polygon* and its use to detect overlap among pieces.

In chapter 5 we defined a MIP model for the irregular nesting problem based on the one proposed by Daniels et al. We enhanced this model by applying a *lifting* technique to the constraint coefficient, in order to reduce the negative effect of big-M coefficients. Furthermore we implemented a branching technique based on priorities to guide the visit of the search tree, and showed that this methodology effectively improves the solution time.

Another problem related to the irregular nesting is to place the small pieces into some holes. Indeed, the usual way to build a solution is to place the "big pieces" first, and then insert the remaining ones (trims) in the holes left by the big pieces. This problem is known as *multiple containment problem.* Choosing which pieces to put in the holes is not an easy task, and most of the times a greedy strategy proves not adequate.

In chapter 6 we defined a new *knapsack type* model for the multiple containment problem, based on geometrical considerations. We designed, implemented and tested this model on real world instances, and we showed that the results archived by this algorithm are almost always better than the solution provided by a greedy algorithm.

Finally, we introduced a new standard specification to represent instances of the nesting problem. We converted all the instances found in the literature to this standard, and provide a web site (NESTLIB) where this data can be downloaded by the scientific community to test different algorithms.

Appendix A

NESTLIB specification

Bibliography

[AA76] M. Adamowicz and A. Albano, *Nesting two dimensional shapes in rectangular modules*, Computer Aided Design **8/1** (1976), 27–33.

[ABJ90] C. Amaral, J. Bernardo, and J. Jorge, *Marker making using automatic placement of irregural shapes for the garment industry*, Computers and Graphics (1990), no. 14/1, 41–46.

[Arb93] A. Arbel, *Large scale optimization methods applied to the cutting stock problem of irregular shapes*, International Journal of Production Research **31/2** (1993), 483–500.

[Art66] R.C. Art, *An approach to the two-dimensional irregular cutting stock problem*, Tech. Report 36-Y08, IBM Cambridge Scientific Centre Report, 1966.

[AS80] A. Albano and G. Sapuppo, *Optimal allocation of two-dimensional irregular shapes using heuristic search methods*, IEEE Transacions on systems, Man, and Cypernetics (1980), no. 10/5, 242–248.

[AST92] P. K. Agarwal, M. Sharir, and S. Toledo, *Applications of parametric searching in geometric optimization*, Proceedings of the 3rd ACM-SIAM Symposium on Discrete Algorithms, 1992, pp. 72–82.

[BG79] D. Böme and A. Graham, *Practical experiences with semi-automatic and automatic part nesting methods*, Computer Applications in the Automation of Shipyard Operation and Ship Design III (C. Kuo, K. MacCallum, and T. Williams, eds.), North-Holland Publishing Company, 1979, pp. 213–220.

[BHW93] J. Blazewicz, P. Hawryluk, and R. Walkowiak, *Using a tabu search approach for solving the two-dimensional irregular cutting problem*, Annals of Operations Research (1993), no. 41, 313–327.

[DBB84] D. Dori and M. Ben-Bassat, *Efficient nesting of conruent convex figures*, Communications of the ACM **27/33** (1984), 228–235.

[DDB98] K.A. Dowsland, W.B. Dowsland, and J.A. Bennel, *Jostling for position: local improvement for irregular cutting patterns*, Journal of the Operational Research Society (1998), no. 49, 647–658.

[DLM94] K. Daniels, Z. Li, and V. Milenkovic, *Multiple containement methods*, Tech. Report TR-12-94, Harvard University, 1994.

[FS75] H. Freeman and R. Shapira, *Determining the minimum area encasing rectangle for an arbitrary closed curve*, Communications of the ACM **81/7** (1975), 409–413.

[GJ79] M. Garey and D. Johnson, *Computers and intractability: A guide to the theory of np-completeness*, W.H. Freeman and Company, San Francisco, 1979.

[Glo89] F. Glover, *Tabu search - part 1*, ORSA Journal on Computing **1** (1989), no. 3, 190–206.

[Glo90] ______, *Tabu search - part 2*, ORSA Journal on Computing **2** (1990), no. 1, 4–32.

[GRS83] L. Guibas, L. Ramshaw, and J. Stolfi, *A kinetic framework for computational geometry*, Proceedings of the 24th IEEE Symposium on Foundations of Computer Science, 1983, pp. 100–111.

[HdW90] A. Hertz and D. de Werra, *The tabu search metaheurisitc: how we used it*, Annals of Mathematics and Artificial Intelligence (1990), no. 1, 111–121.

[HL93] J. Haistermann and T. Lengauer, *Efficient automatic part nesting on irregular and inhomogeneous surfaces*, Proceedeings of 4th ACM-SIAM Symposium on Discrete Algorithms, SODA93, 1993, pp. 251–259.

[JFR92] P. Jain, P. Fenyes, and R. Richter, *Optimal blank nesting using simulated annealing*, Transactions of the ASME (1992), no. 114, 160–165.

[KL91] F. Karoupi and M. Loftus, *Accomodating diverse shapes within hexagonal pavers*, International Journal of Production Research (1991), no. 29/8, 1507–1519.

[KOS91] A. Kaul, M.A. O'Connor, and V. Srinivasan, *Computing minkowski sums of regular polygons*, Proceedings of the Third Canadian Conference on Computational Geometry (Vancouver, British Columbia), 1991.

[LM95] Z. Li and V. Milenkovic, *Compaction and separation algorithms for nonconvex polygons and their applications*, European Journal of Operational Research (1995), no. 84, 539–561.

[LMPD92] H. Lutfiyya, B. McMillin, D.A.P. Poshyanon, and C. Dagli, *Composite stock cutting through simulated annealing*, Mathematical Computer Modelling (1992), no. 16/1, 57–74.

[MDL92] V. Milenkovic, K. Daniels, and Z. Li, *Placement and compaction of nonconvex polygons for clothing manifacture*, Proccedings of the 4th Canadian Conference on Computational Geometry (St. John's, Newfoundland), August 10-14 1992, pp. 236–243.

[Min03] H. Minkowski, *Volumen und oberfläche*, Mathematische Annalen (1903), no. 57, 447–495.

[NW88] G.L. Nemhauser and L.A. Wolsey, *Integer and combinatorial optimization*, Wiley, New York, 1988.

[OF93] J.F.C. Oliveira and J.A.S. Ferreira, *Algorithms for nesting problems*, Applied Simulated Annealing (R.V.V. Vidal, ed.), Springer-Verlag, Berlin, 1993, Lecture Notes in Economics and Mathematical Systems 396, pp. 255–274.

[OGF00] J.F. Oliveira, A.M. Gomes, and S. Ferreira, *Topos a new constructive algorithm for nesting problems*, OR Spectrum (2000), no. 22, 263284.

[PS85] F.P. Preparata and M.I. Shamos, *Computational geometry*, Springer-Verlag, New york, NY, 1985.

[QS87] W. Qu and J. Sanders, *A nesting algorithm for irregular parts and factors affecting trim losses*, International Journal of Production Research (1987), no. 25/3, 381–397.

[Ser82] J. Serra, *Image analysis and mathematical morphology*, Academic Press, New York, 1982.

[Tàv89] J. Tàvora, *Path planning for a class of cutting operations*, Proceedings of SPIE Meeting, Applications of Artificial Intelligence (M. Trivedi, ed.), vol. VII, SPIE 1095, 1989, pp. 405–415.