Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2012, Article ID 254346, 28 pages
doi:10.1155/2012/254346

Research Article

A Variable Neighborhood Search Algorithm for the Leather Nesting Problem

Cláudio Alves, Pedro Brás, José M. Valério de Carvalho, and Telmo Pinto

Centro de Investigação Algoritmi da Universidade do Minho, Escola de Engenharia, Universidade do Minho, 4710-057 Braga, Portugal

Correspondence should be addressed to Cláudio Alves, claudio@dps.uminho.pt

Received 20 May 2011; Accepted 6 October 2011

Academic Editor: J. J. Judice

The leather nesting problem is a cutting and packing optimization problem that consists in finding the best layout for a set of irregular pieces within a natural leather hide with an irregular surface and contour. In this paper, we address a real application of this problem related to the production of car seats in the automotive industry. The high quality requirements imposed on these products combined with the heterogeneity of the leather hides make the problem very complex to solve in practice. Very few results are reported in the literature for the leather nesting problem. Furthermore, the majority of the approaches impose some additional constraints to the layouts related to the particular application that is considered. In this paper, we describe a variable neighborhood search algorithm for the general leather nesting problem. To evaluate the performance of our approaches, we conducted an extensive set of computational experiments on real instances. The results of these experiments are reported at the end of the paper.

1. Introduction

The leather nesting problem (LNP) is a two-dimensional cutting problem whose objective is to find the best layout for a set of irregular pieces within the boundaries of an irregular surface, which is a natural leather hide. The hides not only are irregular in their contour, but alsomay have holes and zones with different quality levels. On the another hand, the pieces are subject to quality requirements that restrict their placement within the hides. They are divided into quality zones that specify the minimum quality of the hide from which these parts of the pieces must be cut. Here, we consider a particular application of the LNP in the automotive industry, and in particular in the production of car seats. This application has influence on, namely, the characteristics of the pieces. For instance, the pieces found in this sector contrast with the pieces of other applications whose complexity tends to be low like in

the furniture industry. In this paper, our conclusions and data are supported by the real case study of a multinational company operating in the automotive sector.

The LNP addressed in this paper shares most of the characteristics and requirements of other nesting problems. Obviously, the pieces must be placed within the boundaries of the hides, and they should not overlap. Except for the quality constraints mentioned above, no other constraints apply to the layouts like, for example, constraints on the directionality of the pieces. Since the hides have a limited area (in opposition to the strips whose area may be considered virtually unlimited), the standard optimization criterion that is considered when building the layouts is the total raw material usage. When multiple hides are involved, we may consider additional criteria like the percentage of remaining space that is left on the last hide [1]. The typology proposed in [2] classifies this LNP as a 2D-RCSP (*two-dimensional residual cutting stock problem*). Because of all these characteristics, the problem is very challenging. At the same time, the value of the hides and the potential for savings make this problem very relevant from an economical point of view.

A first contribution for a similar LNP can be found in [3]. In this paper, Heistermann and Lengauer describe a constructive heuristic that starts by selecting an empty region of the hide using a multicriteria quality function. A subset of pieces is then selected according to their geometry and to the geometry of the selected region. These pieces are placed in this region in the positions that maximizes another multicriteria function used to evaluate how good each piece fits in a given position. The authors report on computational results for real instances that show the competitiveness of their approach compared to the results achieved by human nesters.

More recently, Alves et al. [1] proposed a set of constructive heuristics for the general LNP addressed in this paper. Their approach combines different strategies for grouping the pieces, for selecting the next piece to place on the hide, for selecting an empty region of the hide where the selected piece will be placed, and for evaluating the different placement positions in this region. The authors use no-fit polygons to ensure the feasibility of the layouts, and also to guide the different selection processes of their heuristic. They report on a set of computational experiments on real instances that attest the performance of some of the combinations of strategies described in their paper.

Heistermann and Lengauer [3] and Alves et al. [1] are the only authors to report on results for the same general LNP as the one addressed in this paper. All the other contributions proposed in the literature [4–7] focus on variants of this LNP. In [4], Crispin et al. describe two genetic algorithms for a LNP with directionality constraints. They consider an application on the shoe-making industry where the rotations that can be applied to the pieces are restricted over the hide. In practice, these constraints reduce the complexity of the problem.

In [7], Yuping and Caijun explore a variant of the LNP with no quality constraints. They describe a solution approach based on genetic algorithms and simulated annealing and report on computational results with material usage of 70% achieved in one hour of computing time. Lee et al. [6] addressed a similar LNP with no quality zones and multiple hides. Their algorithm consists in placing the pieces sequentially on the hide, and in adjusting their placement through translations and rotations.

In this paper, we describe a variable neighborhood search (VNS) algorithm for the general LNP described above. Our approach relies on a fast constructive heuristic used to generate the first layout and to complete the partial layouts that might be generated during the local search procedure. It relies also on different neighborhood structures that are explored through the general framework proposed by Mladenović and Hansen in [8].

Our neighborhood structures are based on the sequence by which the pieces are placed on the hide and on the quality of the fitness of these pieces in the current layout. The goal is to improve quickly the incumbent solution by exploring neighboring solutions obtained by removing or replacing the pieces with the worst fitness in a given part of the sequence. We explore different strategies for removing and replacing the pieces with the worst fitness, and to generate the rest of the sequence from this piece forward. These strategies define the moves in the corresponding neighborhoods. The VNS framework is used to systematically explore the various neighborhoods and to escape from the regions that contain a local optimum through randomization. In our algorithm, the local search phase is performed on a single neighborhood space, which is the same as the one used in the shaking phase. As a consequence, our algorithm corresponds to an implementation of the basic or standard VNS. Extensive computational experiments are reported for real instances from our real case study. The tests were performed with two objectives: to tune the algorithm so as to identify the best set of parameters and evaluate the real impact of each neighborhood in the quality of the layouts and to evaluate the performance of the best approach on a large set of real instances.

To the best of our knowledge, no results have been reported in the literature concerning the application of the VNS metaheuristic to the general LNP addressed in this paper. In fact, we are not aware of any algorithm based on VNS for cutting and packing problems involving irregular shapes. Recent applications of VNS to cutting and packing problems can be found in [9–11].

In [9], the authors describe a hybrid method for the strip packing problem combining the greedy randomized adaptive search procedure (GRASP) with VNS. The problem consists in finding the best packing for a set of rectangles within a 2-dimensional strip with (virtually) infinite length. Their objective is to find the pattern with the minimum length. The authors describe an approach that embeds a VNS algorithm in the postprocessing phase of GRASP. The VNS metaheuristic is used to improve the packing of the last rectangles that were placed in the constructive phase. Their neighborhood structures are based on the permutations of these rectangles. They report on a set of computational experiments where their method compares favorably with a simulated annealing algorithm proposed in the literature [12].

In [10], Parreño et al. describe a VNS algorithm for the container-loading problem. The initial solution is generated using a constructive heuristic proposed in [13]. This heuristic builds iteratively a valid layout for the problem by choosing and placing a set of boxes into a so-called maximal space, that is, the space where the largest parallelepiped can be placed. The authors propose five different neighborhood structures for the problem based on five types of movements involving the deletion of layers, the insertion of columns and boxes, and emptying a complete region of the container. In [10], their algorithm is compared with other state-of-the-art approaches proposed in the literature on a set of 1500 benchmark instances. Their approach outperforms these other algorithms for the set of instances that were considered.

In [11], the authors describe a VNS approach for different variants of the 2-dimensional cutting stock problem with guillotine constraints. They propose a set of greedy heuristics for the problem, together with three neighborhood structures and their corresponding local search procedures. These structures rely on the representation of the solutions as a sequence of items, and on movements based on swapping items and reversing subsequences of items. These neighborhood spaces are explored within a variable neighborhood descent procedure. The authors report on computational experiments for real instances of the furniture industry. The number of bins and corresponding waste decreased with their approach when compared with the results achieved by the companies.

The paper is organized as follows. In Section 2, we describe the characteristics of the general LNP addressed in this paper. In Section 3, we introduce the geometrical issues related to the problem and the strategies used to overcome these issues. In Section 4, we describe the constructive heuristic used to generate the initial solution and to complete the partial layouts within the local search phases. In Section 5, we describe the different components of our algorithm including the movements and the corresponding neighborhood structures, and we describe how these components are integrated in our VNS algorithm. In Section 6, we report on an extensive set of computational experiments conducted on real instances from our real case study. Some final conclusions are drawn in Section 7.

2. The Leather Nesting Problem

The LNP is a two-dimensional cutting stock problem that consists in finding the best way to cut a set of irregular pieces from a natural leather hide. The hides and the pieces have both an irregular contour. Furthermore, they may have holes and regions with different quality grades. For the hides, the holes (as other defects), and the quality, zones are a consequence of the nature of the product. On the contrary, a quality zone in a piece defines a minimum quality requirement set by the client. It stipulates that the corresponding part of the piece must be cut from a region of the hide that has at least this level of quality.

Figure 1 shows a leather hide with holes (in white) and different quality zones (in black, red, green, and blue). The regions in grey at the border of the hide are nonusable areas. They are not defects in the strict sense of the word, but because their quality is too low, they are not used to produce any piece. In our specific application, the quality zones are divided into four groups denoted by A, B, C, and D. The zones A correspond to the best-quality regions of the hides. The quality decreases from A to D, with D being the worst quality level. The zones D of the hides are typically used to cut the parts of the pieces that are not visible in the car seats. In the forthcoming figures, we will depict the zones A, B, C, and D of the hides in black, red, green, and blue, respectively. The holes will be represented in white.

The quality zones of the pieces are defined using a similar scheme with four different quality grades. In our case, the pieces are the parts of a car seat. The quality zone of a piece can only be cut from a region of the hide whose quality grade is similar or higher. For example, a zone C of a piece will be cut only from a region A, B, or C of a hide. Note that in the real application considered in this paper the pieces may be highly irregular. They may have different concavities, and their area may differ significantly.

A feasible layout is an arrangement of the pieces on the hide such that the pieces are all placed within the boundaries of the hide (its usable part) without overlapping with each other, and such that the quality requirements of the pieces are fulfilled. Note that while the defects on the hides do not increase the complexity of the problem compared to other nesting problems, the existence of quality zones really do. A defect on the hide (including the holes) can be treated as a piece already placed on this region of the hide. However, the same does not apply to the quality zones. While a quality zone of a hide cannot overlap with some parts of the pieces, it may overlap with others as long as its corresponding quality level is the same or better than the quality requirement of these parts of the pieces.

The most common criterion used to evaluate the quality of a layout is the total material usage. In the automotive industry, other criteria are used to improve the efficiency of the cutting processes [1]. For instance, when a production order requires more than a single hide, an important issue is to ensure that the total usable area that remains on the last hide after the

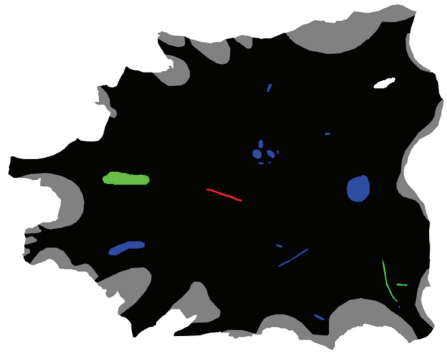


Figure 1: Leather hide.

final cutting operation is maximized. Indeed, the parts of the last hide that are too small are discarded, while the others are used as an input for subsequent orders. Hence, the criterion that applies when generating the layout for the last hide is no more the material usage, but instead the total usable area of this hide. In this paper, we will focus on the generation of efficient layouts on a single hide, and hence, we will use the maximization of the material usage as the optimization criterion.

3. Geometric Issues

The hides, the pieces, and their quality zones, holes, and defects are all represented using polygons. These polygons are generated after a scanning process for the hides, and using a CAD system for the pieces of the car seats. To improve the efficiency of our approach, we simplified first the representation of the hides and pieces by generating approximations of the corresponding shapes using a procedure described in [1]. The outer border of the hides is represented using an inner approximation obtained by removing sequentially the vertices whose distance from its immediate predecessor is smaller than a given (small) parameter. A vertex is removed only if the resulting polygon is in fact an inner approximation of the original polygon. On the contrary, the holes and the quality zones of the hides are replaced by an outer approximation using a similar scheme. All the polygons representing the pieces are replaced by outer approximations using again a similar scheme. This procedure ensures that every layout that is feasible for the approximated hide will remain feasible for the original hide. It is important to note that the number of vertices of the resulting polygons is still much larger than the number of vertices of the polygons used in [3]. The number of vertices used to represent the hides and the pieces decreased, respectively, to 300 and 80 for the instances used in our computational experiments. In [3], the number of vertices per hide and per piece was, respectively, equal to 100 and 70 before their simplifications.

In our approach, we resorted to no-fit polygons (NFPs) to compute the relative position between two pieces and between the pieces and the quality zones of the hide. To ensure that a piece is placed within the boundaries of the hide, we used inner-fit polygons (IFPs). Both the no-fit and the inner-fit polygons are computed using the Minkowski sums [14, 15]. In the particular case of the LNP, testing whether a piece is placed correctly on the hide is more complex than in other nesting problems because of the quality zones. The quality zones of a hide cannot be treated as pieces already placed on its surface (just as its holes

and defects). Indeed, while some parts of the pieces will not be allowed to overlap with the zones of the hide whose quality is lower, these parts will be allowed to overlap with other zones that meet the corresponding quality requirements. Hence, any standard method used to avoid that two pieces overlap must be adapted in the case of the LNP.

The no-fit and inner-fit polygons of two polygons A and B will be denoted by $\mathrm{NFP}_{A,B}$ and $\mathrm{IFP}_{A,B}$, respectively. The outer border of a no-fit polygon $\mathrm{NFP}_{A,B}$ is the path followed by a reference point of B when B slides around A without ever overlapping with A but such that B is always in contact with A. The inner-fit polygon $\mathrm{IFP}_{A,B}$ is defined in a similar way except that the polygon B must always remain inside A. The NFPs are used to know if two polygons overlap or not, while the IFPs are used to know if a polygon is completely contained within another one. Indeed, if the reference point of B is inside $\mathrm{NFP}_{A,B}$, then A and B overlap. If it lies at the boundary of A, then B touches A without overlapping with it. Finally, if the reference point of B is outside $\mathrm{NFP}_{A,B}$, then A and B overlap. Similarly, if the reference point of B is inside $\mathrm{IFP}_{A,B}$, then B is completely inside A. If it is at the border of $\mathrm{IFP}_{A,B}$, then B remains inside A and it touches its border.

The NFPs between all the pairs of pieces have to be computed, together will all the IFPs between each piece and the usable area of the hide, and all the NFPs between each quality zone of the pieces and the regions of the hide with a lower quality grade. The IFPs of a given piece and the hide and the NFPs of this piece and the quality zones of the hide describe the set of feasible placement positions of the piece inside the hide. Note that there can be more than one (unconnected) IFP describing the feasible placement positions inside a hide. The constructive heuristic used in our algorithm takes advantage of the information provided by these polygons.

4. Generating Feasible Layouts: A Constructive Heuristic

To generate feasible layouts, we use a constructive heuristic that follows the same steps as those described in [1]. The approach can be divided as follows:

(a) definition of groups of pieces,

(b) selection of the next piece to place on the hide from a given group of pieces,

(c) selection of a region of the hide where to place the piece,

(d) evaluation of the placement positions in this region and selection of one of these positions.

The steps (b) to (d) are repeated until no more pieces can be placed on the hide, or until there are no more pieces available. The heuristic is used many times in the course of the algorithm, and hence it should remain fast however, without neglecting the quality of the layouts. The strategies used in the steps (b) to (d) were chosen with this objective in mind.

Let m be the total number of different pieces, and let b_i, with $i = 1, \ldots, m$, denote the demand associated to the piece i. The step (a) consists in grouping the pieces according to a given criterion. Let n be the number of groups, and let G_j denote the set of pieces that belong to the group j. For ease of presentation, we will assume that a reindexing of the pieces is applied after the pieces have been assigned to a group. Hence, denoting by m_j the total number of pieces in the group j, we will have $G_j = \{1, 2, \ldots, m_j\}$. The demand of the piece i in group j will be denoted by b_{ji}.

The pieces are assigned to the groups according to the value of their areas. A piece i is assigned to a group j if its area A_i is such that

$$A_i \in \left[\min + \frac{\max - \min}{n} \times (j-1); \min + \frac{\max - \min}{n} \times j\right), \tag{4.1}$$

with min and max representing, respectively, the area of the smallest piece and the area of the largest piece and n being the total number of groups that are allowed. Note that for $j = n$, the previous interval is closed on the right. Group G_1 corresponds to the pieces with the smallest areas. The areas of the pieces increase with the index of the groups, and hence, the pieces with the largest values belong to the nth group G_n.

In the step (b) of the heuristic, we start by choosing the group of pieces from which the next piece will be selected. The groups are chosen by decreasing order of their index. We select the first group that has pieces to be placed and such that at least one of these pieces still fits on the hide. The selection of the next piece and its rotation is based on the index of the chosen group and on the characteristics of the IFPs of the pieces and the hide. Let j be the index of the chosen group. If $j < n/2$, we choose the piece with the smallest IFP, while if $j \geq n/2$, the piece that is selected is the one that has the largest IFP. The reference point of the selected piece is then placed within the IFP that determined the selection of the piece.

The principle is to use the information provided by the IFPs as an indicator of the expectable quality of the fitness of the pieces in the corresponding regions of the hide. We recall that there can be more than one IFP representing the feasible placement positions of a given piece inside the hide. If one of the IFPs of a small piece is a polygon with a small area, then one may expect that the fitness of this piece in the corresponding region of the hide will be good. For the largest pieces, the objective is to place them in regions where the impact on the placement of the next pieces is lower.

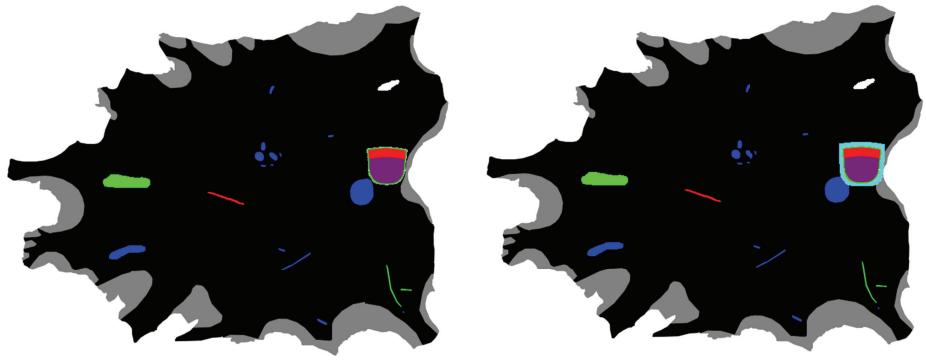
Once a piece and a placement region have been selected, we choose the final placement position of the piece within this region. We use a simple criterion to evaluate each placement position. The position that is selected is the point in the border of the IFP that is nearer from the border of the hide. Here, the goal is to fill the hide preferentially from the border to the center.

5. A Variable Neighborhood Search Algorithm

5.1. Overview

The VNS framework was introduced by Mladenović and Hansen [8] and applied successfully to many optimization problems (see [16] for a recent survey). This framework leads to a metaheuristic that drives the search into different neighborhoods in a systematic way so as to improve the incumbent solution. In this paper, we use this framework to explore alternative neighborhoods for the general LNP. Our approach relies on the constructive heuristic introduced above and on a set of neighborhoods that are briefly described below. The details of the neighborhood structures will be given in the next section.

The initial layout is obtained by applying the constructive heuristic defined above. This heuristic determines the sequence by which the pieces are placed and the exact position of each piece on the hide. A layout is defined by the set of pieces that are placed on the hide together with their rotation and position. As an alternative, we can represent a layout as



(a) (b)

Figure 2: Evaluating the quality of the fitness of a piece at a given placement position.

a sequence of pieces combined with the iterative application of the steps (c) and (d) of our constructive heuristic. By iterative, we mean that the pieces are chosen and placed on the hide one after the other following the order defined by the sequence and that the process repeats until there are no more pieces in the sequence. We will use this latter representation to describe our neighborhood structures and associated movements.

We propose four types of distinct movements defined from a sequence of pieces: exchanging a piece p by another that is not in the sequence, removing all the pieces from this piece forward and filling the hide using the constructive heuristic; exchanging a piece p by another that is not in the sequence, keeping the rest of the sequence unchanged, placing the corresponding pieces using iteratively the steps (c) and (d) of the constructive heuristic and filling the rest of the hide by applying the complete heuristic; swapping two pieces p and p' in the sequence if p' can be placed on the hide with a better fitness than p following the steps (c) and (d) of the heuristic placing all the pieces of the sequence using the steps (c) and (d) of the heuristic and filling the rest of the hide with the complete heuristic; removing a piece p of the sequence, placing all the pieces of the resulting sequence by applying iteratively the steps (c) and (d) of the heuristic, and filling the rest of the hide with the complete heuristic.

The piece p that is exchanged, swapped with another piece, or removed from the sequence is chosen from a set of candidate pieces with the worst fitness. The quality of the fitness of a piece is evaluated by computing the areas of the polygons resulting from the intersection of an offset of the piece with the outer part of the hide and with the current layout and by dividing this total area by the area of the offset. The largest will be this value, the better will be the fitness. Ideally, when this value is equal to 1, the whole border of the piece touches the border of the hide or the border of the current layout. Figure 2 illustrates the process.

For ease of presentation, we will always denote in this paper the piece that is to be exchanged, swapped with another piece, or removed from the sequence by p, and the set of candidate pieces from which p is selected by P. Similarly, the set of pieces that are candidate to substitute p in the sequence will be denoted by P', and the piece that is selected from P' will be denoted by p'. Furthermore, we will denote by S a sequence of pieces, and we will define it as a vector of tuples as follows:

$$S = (s_1, s_2, \ldots, s_{|S|}), \tag{5.1}$$

with $s_k = (j, i)$ denoting the group (G_j) and the piece $(i \in G_j)$ that is in the kth position in the sequence. The vector containing the values of the corresponding fitness of these pieces will be denoted by F and defined as follows:

$$F = (f_1, f_2, \ldots, f_{|S|}), \tag{5.2}$$

with $0 \leq f_k \leq 1$, $k = 1, \ldots, |S|$.

A key characteristic of our approach is that the neighbors of a solution L (except eventually the one defined through the fourth type of movement described above) are layouts in which the fitness of the piece p' is always better than the fitness of the piece p in L. Furthermore, the pieces p will not be chosen from any part of the sequence, but from a restricted subpart of the sequence. These subparts will be defined based on the position of the pieces in the sequence. However, instead of using the indexes k of the tuples s_k in S to denote these positions, we use the percentage of material usage achieved right after a piece is placed on the hide. For example, we will choose the pieces that are in a window of 50% to 70% of material usage meaning that only the pieces that lead to a material usage in this interval right after being placed on the hide will be chosen. We will denote by U the vector of values of the material usage achieved after placing each piece of the sequence S. We will have

$$U = (u_1, u_2, \ldots, u_{|S|}), \tag{5.3}$$

with $u_k < u_{k+1}$, $k = 1, \ldots, |S| - 1$. The vector U will represent the evolution of the material usage as the pieces are placed on the hide. To identify a subpart of the sequence, we will use a lower and an upper bound for the value of the material usage. We will denote these values by $u_{\min}$ and $u_{\max}$, respectively. For the example given above, we have $u_{\min} = 50\%$ and $u_{\max} = 70\%$.

Specific neighborhoods are defined from the general movements referred to above. The neighborhoods further depend on the following set of parameters:

(i) the number of pieces that are candidate to be exchanged, swapped with another piece, or removed, that is, $|P|$ (we will denote this parameter by q);

(ii) the number of pieces that are candidate to substitute the piece p, that is, $|P'|$ (we will denote this parameter by r);

(iii) the subpart of the sequence from which the pieces of P are chosen, that is, $(s_i, \ldots, s_j)$ with

$$\begin{aligned} i &= \operatorname{argmin}_{l \in \{1,\ldots,|S|\}} \{u_l : u_l \geq u_{\min}, u_l \in U\}, \\ j &= \operatorname{argmax}_{l \in \{1,\ldots,|S|\}} \{u_l : u_l \leq u_{\max}, u_l \in U\}. \end{aligned} \tag{5.4}$$

Note that when $q = 1$, the piece that is exchanged, swapped, or removed is the piece with the worst fitness in the given subsequence. Similarly, when $r = 1$, the piece p is replaced in the sequence by the piece that leads to the best fitness, either from the sequence S or not. It is possible to define various neighborhoods by setting these parameters to different values. The details of these neighborhoods will be given in the next section.

To explore these neighborhoods, we use the standard VNS framework. At each iteration, the procedure tries to improve the incumbent by looking to local optima within each neighborhood. Driving the search to a different neighborhood may improve the solutions obtained through the exploration of the previous neighborhoods since the local optima in two different neighborhoods are not necessarily the same. In this paper, we propose an approach based on VNS that explores systematically the neighborhoods obtained through the general movements described above and by setting the above parameters to some specific values.

5.2. Movements and Neighborhood Structures

We begin by describing in detail the procedures that define each one of the four movements referred to above. These movements will be denoted by M_1, M_2, M_3, and M_4, respectively.

(A) Movement M_1

(1) Select the piece p:

Let $k_1 = \text{argmin}_{l\in\{1,\ldots,|S|\}}\{u_l : u_l \geq u_{\min}, u_l \in U\}$ and $k_2 = \text{argmax}_{l\in\{1,\ldots,|S|\}}\{u_l : u_l \leq u_{\max}, u_l \in U\}$;
Let P be the set of the q pieces from the subsequence $(s_{k_1}, \ldots, s_{k_2})$ with the smallest fitness in F;
Choose a piece p from P;

(2) Select the piece p':

Let C be the set of pieces that are not in the sequence S;
Let P' be the set of the r pieces of C with the best fitness when placed on the hide right after s_{k_1-1} following the steps (c) and (d) of the heuristic;
Choose a piece p' from P';

(3) Update the sequence S:

Let G_j be the group of the piece p and i the index of p in G_j, and let k_3 be the index of (j, i) in S corresponding to the chosen piece p;
Let $G_{j'}$ be the group of the piece p' and i' the index of p' in $G_{j'}$;
Remove the subsequence $(s_{k_3}, \ldots, s_{|S|})$ from S;

(4) Place the pieces of the sequence $(s_1, \ldots, s_{k_3-1}, (j', i'))$ on the hide following the steps (c) and (d) of the constructive heuristic (if a piece of the sequence cannot be placed on the hide, update S by removing this piece from the sequence);

(5) Fill the hide (and complete the sequence S) using the constructive heuristic.

Note that if $k_3 = 1$, then the subsequence $(s_1, \ldots, s_{k_3-1})$ is an empty vector.The movement M_2 differs from M_1 only on the steps (3) and (4). As happens in M_1, the piece p is replaced in the current sequence S by another (p') with a better fitness and that is not in S, while the rest of the sequence from p forward remains unchanged. The pieces of the new sequence are placed on the hide using the steps (c) and (d) of the constructive

heuristic. When all these pieces have been placed (or discarded eventually if they do not fit on the hide), the complete heuristic is used (eventually) to fill the empty spaces that may remain.

(B) Movement M_2

(1) Select the piece p: similar to step (1) of M_1;

(2) Select the piece p': similar to step (2) of M_1;

(3) Update the sequence S:

Let G_j be the group of the piece p and i the index of p in G_j, and let k_3 be the index of (j,i) in S corresponding to the chosen piece p;
Let $G_{j'}$ be the group of the piece p' and i' the index of p' in $G_{j'}$;
Replace (j,i) in the position k_3 of the sequence by (j',i'), that is,

$$S = \left(s_1, \ldots, s_{k_3-1}, \left(j', i'\right), s_{k_3+1}, \ldots, s_{|S|}\right); \tag{5.5}$$

(4) Place the pieces of the new sequence S on the hide following the steps (c) and (d) of the constructive heuristic (if a piece of the sequence cannot be placed on the hide, update S by removing this piece from the sequence);

(5) Fill the hide (and complete the sequence S) using the constructive heuristic.

In the movement M_3, the piece p' is selected from the subsequence of S starting from p and up to the end of the sequence. The pieces p and p' are swapped in the sequence S, and the pieces of the resulting sequence are placed using the steps (c) and (d) of the heuristic. Again, the constructive heuristic is used at the end of the process to fill the empty spaces.

(C) Movement M_3

(1) Select the piece p: similar to step (1) of M_1;

(2) Select the piece p':

Let P' be the set of the r pieces of S from the chosen piece p up to the end of S with the best fitness when placed on the hide right after s_{k_1-1} following the steps(c) and (d) of the heuristic;
Choose a piece p' from P';

(3) Update the sequence S:

Let G_j be the group of the piece p and i the index of p in G_j, and let k_3 be the index of (j,i) in S corresponding to the chosen piece p;
Let $G_{j'}$ be the group of the piece p' and i' the index of p' in $G_{j'}$, and let k_4 be the index of (j',i') in S corresponding to the chosen piece p';
Swap s_{k_3} and s_{k_4} in S, that is,

$$S = \left(s_1, \ldots, s_{k_3-1}, s_{k_4}, s_{k_3+1}, \ldots, s_{k_4-1}, s_{k_3}, s_{k_4+1}, s_{|S|}\right); \tag{5.6}$$

(4) Place the pieces of the new sequence S on the hide following the steps (c) and (d) of the constructive heuristic (if a piece of the sequence cannot be placed on the hide, update S by removing this piece from the sequence);

(5) Fill the hide (and complete the sequence S) using the constructive heuristic.

Here, we assume that $k_3 < |S|$, otherwise it is clear that p' will not exist.The fourth movement consists in removing the piece p from the sequence S and placing the remaining pieces from this piece forward using the constructive heuristic.

(D) Movement M_4

(1) Select the piece p: similar to step (1) of M_1;

(2) Update the sequence S:

Let G_j be the group of the piece p and i the index of p in G_j, and let k_3 be the index of (j, i) in S corresponding to the chosen piece p;

Remove s_{k_3} from S, that is, $S = (s_1, \ldots, s_{k_3-1}, s_{k_3+1}, \ldots, s_{|S|})$;

(3) Place the pieces of the new sequence S on the hide following the steps (c) and (d) of the constructive heuristic (if a piece of the sequence cannot be placed on the hide, update S by removing this piece from the sequence);

(4) Fill the hide (and complete the sequence S) using the constructive heuristic.

The objective of these movements is to explore different neighborhoods of the current solution L that are all composed by solutions obtained from L by removing one of its pieces with a bad fitness, and replacing it (eventually) by another piece with a better fitness. They differ essentially from each other in the way the piece p is treated. Note that while in M_1, M_2, and M_3, the piece p is replaced by another that leads to a better fitness (by definition), there is no guarantee that the piece s_{k_3+1} in M_4 will improve the fitness of the piece p.

The movement M_1 is the most computationally expensive among the four movements, since it forces to repeat the complete heuristic right after p has been exchanged. In the other movements, the step (b) of the heuristic consisting in the selection of the next piece to place is not applied, while the pieces of the new sequence S are being placed. The heuristic is applied in its entirety only after all these pieces have been placed on the hide. Typically, the number of iterations at this stage is small. The movements M_2 and M_3 vary essentially on the source from which p' is selected, either from the sequence or not. The movement M_4 is the simplest movement. The neighborhoods of a solution L defined through M_4 are composed by $|P|$ solutions. This movement is proposed as a fast strategy that aims at removing a piece with a bad fitness, letting the heuristic complete the layout starting from its immediate successor s_{k_3+1} in S.

Each movement will be associated to a specific neighborhood. We will denote the neighborhoods of a solution L obtained through M_i by $N_i(L)$, $i = 1, \ldots, 4$. The neighborhoods $N_1(L)$, $N_2(L)$, and $N_3(L)$ of a solution L consists in the solutions generated by taking every piece $p \in P$, and for each one these pieces by replacing (or swapping) it by every piece $p' \in P'$, and then by applying the steps (3) to (5) of M_1, M_2 and M_3, respectively. The neighborhood $N_4(L)$ of a solution L is composed by all the solutions obtained from L by removing each one of the pieces of P, and by applying the steps (2) to (4) of M_4. These neighborhoods depend on different parameters, namely, $u_{\min}$ and $u_{\max}$ that determine the subpart of the sequence

```
Input:
    t_max neighborhood structures N_t, t = 1, ..., t_max;
    A stopping criterion;
(1) Initialization:
      x :=findInitialSolution ();
(2) Repeat the following steps until the stopping criterion is met:
    (a) t := 1;
    (b) Repeat the following steps until t = t_max:
          x' := shaking (x, t);
          x'' := localSearch (x', t);
          if v(x'') < v(x) then
              x := x'';
              t := 1;
          else
              t := t + 1;
```

Algorithm 1

from which the pieces of P are selected, and the parameters q and r that determine the size of the sets P and P', respectively. Specific neighborhoods can be obtained from these general definitions by setting these parameters to given values. To explore in a systematic way the neighborhoods induced by these general definitions, we developed an algorithm based on the VNS framework described in [8]. The details of this approach are given in the following section.

5.3. Variable Neighborhood Search

The VNS metaheuristic defines a general framework for optimization based on the systematic exploration of different neighborhoods. It is a local search metaheuristic that allows for the escape from local optima by switching among the neighborhoods. It relies on the simple observation that a local optimum in a given neighborhood may not remain optimal in another neighborhood. Different variants are defined from this general framework, such as the variable neighborhood descent algorithm and the variable neighborhood decomposition search procedure. In [16], Hansen et al. reviewed different possible implementations of the VNS metaheuristic. These schemes are general enough to allow for different practical implementations. In this paper, we describe an implementation of the VNS metaheuristic based on the basic approach. To improve the efficiency of our approach, we adapted some components of the basic scheme, essentially in the local search phase. A computational study was performed to tune the resulting algorithm, to identify promising sets of parameters and to evaluate the impact of each neighborhood in the quality of the solutions. The results of this study are provided in Section 6.

Let $v(x)$ denote the value of a solution x. The general steps of the basic VNS are described in Algorithm 1 for a minimization problem.

The output of this algorithm is the incumbent solution x. The procedure `findInitialSolution` generates an initial solution for the problem which becomes also the first incumbent solution. At each iteration, a solution is generated randomly from the tth neighborhood of x through the `shaking` procedure. A local optimum is then sought in this

Input:
For each N_i, $i = 1, \ldots, 4$, t_{N_i} sets of parameters $(u^j_{\min,i}, u^j_{\max,i})$, q^j_i and r^j_i, $j = 1, \ldots, t_{N_i}$, such that $t_{\max} = \sum_{i=1}^{4} t_{N_i}$ (the resulting neighborhood structures will be indexed with the index t, with $t = 1, \ldots, t_{\max}$);
A limit t_{limit} on the total computing time;
(1) Initialization:
L :=`findInitialSolution();`
(2) Repeat the following steps until `cpuTime()` $\geq t_{\text{limit}}$:
(a) $t := 1$;
(b) Repeat the following steps until $t = t_{\max}$:
$L' :=$ `shaking`(L, t);
$L'' :=$ first Improvement (L', t);
if $v(L'') < v(L)$ then
$L := L''$;
$t := 1$;
else
$t := t + 1$;

Algorithm 2

tth neighborhood space through the local search procedure `localSearch`. If the value of this local optimum is better than the value of the incumbent solution, the incumbent is updated and the search is resumed starting from the first neighborhood structure and with this new incumbent. If it is worse or equal than the value of the incumbent solution, the local search procedure is then applied on the neighborhood N_{t+1}. The process repeats until the stopping criterion is finally met. Note that this general definition allows for different implementations. For instance, the strategy used to switch between two neighborhood structures may be different from the sequential scheme followed in the basic VNS described above. Similarly, the criterion used to accept a local optimum and move to the corresponding solution may not be based (only) on the value of this local optimum.

Our implementation of the basic VNS for the general LNP is described next. The neighborhood structures N_i refer to the structures introduced in the previous section. Furthermore, we denote by $(u^j_{\min,i}, u^j_{\max,i})$ the jth pair of lower and upper bounds $(u_{\min}, u_{\max})$ for the value of the material usage associated to the neighborhood N_i, which define the subpart of the sequence from which the pieces of P are selected. The parameters q^j_i and r^j_i will be used to denote the general parameters q and r introduced in Section 5.1 and associated to the jth neighborhood defined from N_i. For each N_i, $i = 1, \ldots, 4$, let t_{N_i} denote the number of parameter sets that are considered, each one consisting on given values for the parameters $(u^j_{\min,i}, u^j_{\max,i})$, q^j_i, and r^j_i, $j = 1, \ldots, t_{N_i}$. Each one of these parameter sets defines a specific neighborhood structure within one of the general neighborhoods N_i, $i = 1, \ldots, 4$ (whose precise definition depends on this set of parameters). In practice, $\sum_{i=1}^{4} t_{N_i}$ neighborhood structures are effectively considered in our algorithm. Let $t_{\max} = \sum_{i=1}^{4} t_{N_i}$. For the sake of clarity, in our description of our VNS algorithm given in Algorithm 2, we will index these neighborhoods sequentially using an index t such that $t = 1, \ldots, t_{\max}$.

The initial layout L is generated using the constructive heuristic described in Section 4. The neighborhood structures used in the shaking phase and in the local search procedure are the same. In our implementation, we used a first improvement local search procedure. To further accelerate this procedure, we restrict the search to a subpart of the neighborhood of

the solution L'. Indeed, whenever q (the size of the set P of pieces to exchange, swap with others, or remove) is greater than 1, we explore only the neighbors obtained by choosing randomly one piece p of P, and by taking all the pieces p' of P' in the case of N_1, N_2, and N_3. Note that to generate a neighboring solution, we have to use the constructive heuristic. The heuristic is computationally expensive, namely, because of the computation of the NFPs and IFPs, even if, in practice, the heuristic is applied only on the subpart of the sequence that was changed, since the head of the sequence that was kept unchanged produces exactly the same partial layout. Finally, the execution of the algorithm is stopped after a given time limit has been reached.

6. Computational Experiments

Two sets of experiments were conducted: one to compare possible strategies for our VNS algorithm and to tune the parameters of the algorithm and another to evaluate the performance of the best approach that emerged from the tuning experiments on a large set of real instances. In this section, we report on the results of these experiments. The instances (the hides and the pieces) came from the company that is used as a case study in this paper, and in particular from two car models whose seats are produced by this company. We used the data corresponding to one of the seats for each car model. The total number of different pieces of the first car model was equal to 23, while 22 different pieces were used for the second car model. The pieces can be rotated, but since each rotation implies the computation of a new set of NFPs and IFPs, we restricted the rotations to multiples of 45 degrees.

The experiments were conducted on a PC with an Intel Core i3 CPU with 2.27 GHz and 4 GB of RAM. The algorithms were coded in C++, and the computational geometry routines were implemented using CGAL 3.7 (Computational Geometry Algorithms Library).

6.1. Tuning the VNS Algorithm

The tuning experiments focus on the different aspects and parameters of our VNS algorithm. The objectives of these experiments are summarized next:

(a) evaluating the impact of each neighborhood on the quality of the layouts that are generated;

(b) comparing different strategies for setting the intervals of material usage $(u^j_{\min,i}, u^j_{\max,i})$ from which the pieces of P are selected;

(c) evaluating alternative values for the parameters q^j_i, that is, the size of the sets P from which the pieces that are exchanged, swapped with another, or removed are selected;

(d) evaluating alternative values for the parameters r^j_i, that is, the size of the set P' of pieces that will be inserted in the sequence;

(e) comparing the quality of the layout when the neighborhoods are explored by different orders.

For this purpose, we used 24 instances from the two car models referred to above. From the first car model, we generated 4 production orders with different subsets of pieces from the global set of 23 pieces. The number m of different pieces for each order was, respectively,

equal to 8, 12, 15, and 23. The demand for each piece was set equal to 100. For each order, we repeated the algorithm on 3 different leather hides. In practice, that led to 12 different instances of the LNP. From the second car model, we generated 12 different orders in the same way. The number of different pieces varied from 5 to 22 ($m \in \{5, 7, 8, 9, 12, 14, 20, 22\}$). Again, the demand for each piece was equal to 100. We used the same hide for each production order. In all the cases, the production orders were larger than the hide. The pieces of the production orders completely filled the hide, and there were always pieces that remained to be cut. Our goal was to evaluate the capacity of the VNS algorithm to find good quality layouts, that is, with a high material usage on each single hide. The limit on the total computing time was set to 600 s. In the company, the pieces of a production order are placed on the hide by two operators. The average time used by these human nesters is typically around 600 s. In the subsequent tables, the instances related to the first car model are identified by the index of the corresponding production order x and the index y of the hide as follows: $x; y$. The instances associated to the second car model are identified by the index of the corresponding production order. The column *Inst.* in these tables identifies the problem instance.

In Table 1, we give the results obtained when different neighborhood structures are used in the VNS algorithm. The objective is to evaluate whether the best layouts tend to be found by exploring a particular neighborhood, or if all the neighborhoods contribute in the same way to the quality of the final layouts. Nine strategies were explored: using all the neighborhood structures N_i with $t_{N_i} = 2$, $q_i^1 = 1$ and $q_i^2 = 3$, $i = 1, \ldots, 4$; using the neighborhood structure N_1 with $t_{N_1} = 1$ and $q_1^1 = 1$ (that consists in exchanging, swapping, or removing the piece with the worst fitness within a given interval defined by the parameters $u_{\min,1}^1$ and $u_{\max,1}^1$), and similarly using only one of the neighborhoods N_1 with $q_1^1 = 3$, N_2 with $q_2^1 = 1$, N_2 with $q_2^1 = 3$, N_3 with $q_3^1 = 1$ or N_3 with $q_3^1 = 3$, N_4 with $q_4^1 = 1$ or N_4 with $q_4^1 = 3$. For all the cases except the first, we assume that $t_{N_i} = 1$, $i = 1, \ldots, 4$, that is, only one specific neighborhood structure was used for each one of these cases. In all these configurations, the other parameters of the neighborhoods were set as follows: $r_i^j = 3$ and $(u_{\min,i}^j, u_{\max,i}^j) = (10\%, 0.95 \times \text{MU}\%)$, $j = 1, \ldots, t_{N_i}, i = 1, \ldots, 4$, with MU being the percentage of material usage achieved in the initial layout. In the case where all the neighborhoods were used, we explored them by increasing value of their indexes. Note that we considered explicitly the neighborhood structures N_i with $q_i^j = 1$, $i = 1, \ldots, 4$, because we observed that the best results were usually obtained when this neighborhood was included in the search.

The tests were repeated three times for each instance and configuration. A total of 648 runs were performed (4.5 days of total computing time). In Table 1, we indicate the number of times each configuration provided the best layouts for the corresponding instance. These results show that the best layouts are more often generated when all the neighborhoods are used. Indeed, 30 of the 72 best layouts are found with this configuration, while the second-best strategy (N_1 with $q_1^1 = 3$) reaches the best layout only in 9 cases.

The best strategies identified at some step of our tuning experiments are always used in the following tests. For instance, the strategy that consists in using all the neighborhoods were used in the next experiments.

The second set of tuning experiments was conducted to compare some alternative strategies for setting the values of the lower and upper bounds $(u_{\min,i}^j, u_{\max,i}^j)$ that are used on the material usage to define the subsequences from which the pieces of P are selected. We analyzed five strategies. Broadly speaking, we analyzed whether the best results are achieved by using a single or more than one interval for each general neighborhood structure N_i,

Table 1: Evaluating the impact of the neighborhood structures (number of times the best layout is found with the corresponding strategy).

Inst.	All	Neighborhood							
		N_1 $q_1^1 = 1$	N_1 $q_1^1 = 3$	N_2 $q_2^1 = 1$	N_2 $q_2^1 = 3$	N_3 $q_3^1 = 1$	N_3 $q_3^1 = 3$	N_4 $q_4^1 = 1$	N_4 $q_4^1 = 3$
Car model 1									
1; 1	3								
1; 2	2						1		
1; 3			2		1				
2; 1	1	2							
2; 2	1		1			1			
2; 3	1		1	1					
3; 1					1		1		1
3; 2	2			1					
3; 3					1		1		1
4; 1	2								1
4; 2	2			1					
4; 3	2				1				
Total	16	2	4	3	4	1	3	0	3
Car model 2									
1		3							
2								3	
3	2	1							
4	2								1
5	1	1					1		
6			1		2				
7	1						2		
8	1		1			1			
9	1		2						
10	2		1						
11	2				1				
12	2				1				
Total	14	5	5	0	4	1	3	3	1

$i = 1, \ldots, 4$, and we tried to identify where this interval should be located. The parameters that characterize the five approaches are given next:

(1) $t_{N_i} = 2,\ i = 1, \ldots, 4$:

$$q_i^1 = 1, r_i^1 = 3, (u_{\min,i}^1, u_{\max,i}^1) = (50\%, 0.95 \times \mathrm{MU}\%);$$
$$q_i^2 = 3, r_i^2 = 3, (u_{\min,i}^2, u_{\max,i}^2) = (50\%, 0.95 \times \mathrm{MU}\%);$$

(2) $t_{N_i} = 2,\ i = 1, \ldots, 4$:

$$q_i^1 = 1, r_i^1 = 3, (u_{\min,i}^1, u_{\max,i}^1) = (10\%, 50\%);$$
$$q_i^2 = 3, r_i^2 = 3, (u_{\min,i}^2, u_{\max,i}^2) = (10\%, 50\%);$$

(3) $t_{N_i} = 2,\ i = 1, \ldots, 4$:

$$q_i^1 = 1, r_i^1 = 3, (u_{\min,i}^1, u_{\max,i}^1) = (10\%, 0.95 \times \mathrm{MU}\%);$$
$$q_i^2 = 3, r_i^1 = 3, (u_{\min,i}^2, u_{\max,i}^2) = (10\%, 0.95 \times \mathrm{MU}\%);$$

(4) $t_{N_i} = 4,\ i = 1, \ldots, 4$:

$$q_i^1 = 1, r_i^1 = 3, (u_{\min,i}^1, u_{\max,i}^1) = (10\%, 50\%);$$
$$q_i^2 = 1, r_i^2 = 3, (u_{\min,i}^2, u_{\max,i}^2) = (50\%, 0.95 \times \mathrm{MU}\%);$$
$$q_i^3 = 3, r_i^3 = 3, (u_{\min,i}^3, u_{\max,i}^3) = (10\%, 50\%);$$
$$q_i^4 = 3, r_i^4 = 3, (u_{\min,i}^4, u_{\max,i}^4) = (50\%, 0.95 \times \mathrm{MU}\%);$$

(5) $t_{N_i} = 4, i = 1, \ldots, 4$:

$$q_i^1 = 1, r_i^1 = 3, (u_{\min,i}^1, u_{\max,i}^1) = (25\%, 50\%);$$
$$q_i^2 = 1, r_i^2 = 3, (u_{\min,i}^2, u_{\max,i}^2) = (50\%, 0.95 \times \mathrm{MU}\%);$$
$$q_i^3 = 3, r_i^3 = 3, (u_{\min,i}^3, u_{\max,i}^3) = (25\%, 50\%);$$
$$q_i^4 = 3, r_i^4 = 3, (u_{\min,i}^4, u_{\max,i}^4) = (50\%, 0.95 \times \mathrm{MU}\%).$$

The first strategy consists in using a single neighborhood for each general structure N_i, $i = 1, \ldots, 4$, and in selecting the pieces of the set P preferentially from the tail of the sequence, while in the second strategy, the pieces of P are chosen from the other part of the sequence. The third strategy encompasses both the two previous strategies. It consists in selecting the pieces of P from almost all the sequence except its very beginning. In the fourth and fifth strategies, we explore the use of two neighborhoods for each general neighborhood structure N_i, N_i, $i = 1, \ldots, 4$, based on different intervals $(u_{\min,i}^j, u_{\max,i}^j)$. The fourth strategy combines the intervals used in the first and second strategy. In the fifth strategy, we try to explore neighborhoods where the head of the sequences is kept unchanged. Indeed, the pieces that are at the head of the sequence are placed first, and they lead usually to a good fitness when compared to the other pieces of the sequence.

The tests were executed only once for each instance and configuration. A total of 120 runs were performed (0.83 days of total computing time). The results of these experiments are given in Table 2. Again, we report on the number of times that a given strategy led to the best solution for each instance. For the first car model, the results are nearly the same for all the strategies, while for the second car model, the second strategy clearly dominates the other. When the pieces of P are allowed to be selected from the first part of the sequence (as happens in the second strategy), the layouts that are in the neighborhood of the current solution L may be quite different from L, since a large part of the layout may have to be rebuilt using the constructive heuristic if the piece p that is selected is at the very beginning of the sequence. That allows for the exploration of more diverse solutions, and thus it increases the possibilities of finding a layout with an improved material usage.

The third set of tuning experiments focus on the parameters q_i^j of the neighborhood structures. The corresponding results are reported in Table 3. For each general neighborhood structure, we defined two specific structures, that is, $t_{N_i} = 2$, $i = 1, \ldots, 4$. The structures differ in the value of the parameter q_i^j. We used $q_i^1 = 1$ and $q_i^2 = 3$ in one case, and $q_i^1 = 1$ and $q_i^2 = 5$ in the other, for $i = 1, \ldots, 4$. The parameters r_i^1 and r_i^2 were set equal to 3, while the intervals $(u_{\min,i}^j, u_{\max,i}^j)$, $j \in \{1, 2\}$, were selected according to the second strategy described

Table 2: Evaluating the impact of the parameters $(u^j_{\min,i}, u^j_{\max,i})$ (number of times the best layout is found with the corresponding strategy).

Inst.	Definition of $(u^j_{\min,i}, u^j_{\max,i})$				
	1	2	3	4	5
			Car model 1		
1; 1				1	
1; 2			1		
1; 3			1		
2; 1	1				
2; 2					1
2; 3					1
3; 1		1			
3; 2					1
3; 3			1		
4; 1	1				
4; 2				1	
4; 3		1			
Total	2	2	3	2	3
			Car model 2		
1	1				
2			1		
3			1		
4		1			
5		1			
6		1			
7	1				
8		1			
9				1	
10		1			
11		1			
12	1				
Total	3	6	2	1	0

above. In summary, we used two different values for the parameter q_i^2, namely, $q_i^2 = 3$ and $q_i^2 = 5$, $i = 1, \ldots, 4$. The objective was to analyze whether an increase in the value of this parameter could lead to better layouts. The tests were repeated three times for each instance and configuration. A total of 144 runs were performed (one day of computing time). We can observe from Table 3 that the quality of layouts decreases as we increase the value of this parameter for both the car models.

To evaluate the impact of the r_i^j parameter, we conducted different experiments using the neighborhood structures corresponding to the following configurations:

(1) $t_{N_i} = 2,\ i = 1, \ldots, 4$:

$$q_i^1 = 1, r_i^1 = 1, (u^1_{\min,i}, u^1_{\max,i}) = (10\%, 50\%);$$
$$q_i^2 = 3, r_i^2 = 1, (u^2_{\min,i}, u^2_{\max,i}) = (10\%, 50\%);$$

Table 3: Evaluating the impact of the parameters q_i^j (number of times the best layout is found with the corresponding strategy).

Inst.	$q_i^2 = 3$	$q_i^2 = 5$
	Car model 1	
1; 1	1	2
1; 2	1	2
1; 3	2	1
2; 1	0	3
2; 2	2	1
2; 3	3	0
3; 1	3	0
3; 2	1	2
3; 3	3	0
4; 1	1	2
4; 2	3	0
4; 3	2	1
Total	22	14
	Car model 2	
1	2	1
2	1	2
3	1	2
4	2	1
5	2	1
6	1	2
7	3	0
8	2	1
9	2	1
10	1	2
11	2	1
12	2	1
Total	21	15

(2) $t_{N_i} = 2$, $i = 1, \ldots, 4$:

$$q_i^1 = 1,\ r_i^1 = 3,\ (u_{\min,i}^1, u_{\max,i}^1) = (10\%, 50\%);$$
$$q_i^2 = 3,\ r_i^2 = 3,\ (u_{\min,i}^2, u_{\max,i}^2) = (10\%, 50\%);$$

(3) $t_{N_i} = 2$, $i = 1, \ldots, 4$:

$$q_i^1 = 1,\ r_i^1 = 5,\ (u_{\min,i}^1, u_{\max,i}^1) = (10\%, 50\%);$$
$$q_i^2 = 3,\ r_i^2 = 5,\ (u_{\min,i}^2, u_{\max,i}^2) = (10\%, 50\%).$$

The three configurations differ exclusively on the value of the r_i^j parameters. Here, our objective was to analyze the impact on the quality of the layouts when one increases or decreases the value of this parameter. The results are reported in Table 4. A total of 216 runs were performed (1.5 days of computing time). From these experiments, we can conclude that

Table 4: Evaluating the impact of the parameters r_i^j (number of times the best layout is found with the corresponding strategy).

Inst.	$r_i^j = 1$	$r_i^j = 3$	$r_i^j = 5$
	Car model 1		
1; 1		3	
1; 2	1		2
1; 3		3	
2; 1			3
2; 2		2	1
2; 3			3
3; 1		3	
3; 2	1	2	
3; 3	3		
4; 1			3
4; 2	2	1	
4; 3		3	
Total	7	17	12
	Car model 2		
1	1		2
2	2		1
3		2	1
4		2	1
5		2	1
6	1	2	
7		3	
8	1	2	
9		3	
10		2	1
11		3	
12	1	1	1
Total	6	22	8

the best layouts are achieved when $r_i^j = 3$. The quality of the solutions gets worse whether we increase or decrease the value of this parameter for both the car models used in the experiments. Increasing the value of the parameters r_i^j increases the number of neighboring solutions. However, from our experiments this larger variety of solutions does not reflect upon the quality of the final layouts.

Our last set of tuning experiments were conducted to evaluate the impact of the order by which the neighborhoods are explored on the quality of the final layouts. For this purpose, we used the best set of parameters identified in the previous experiments, and we tried 12 different sequences for the search, namely,

(1) N_1, N_2, N_3, and N_4;

(2) N_2, N_1, N_3, and N_4;

(3) N_1, N_3, N_2, and N_4;

Table 5: Evaluating the impact of the order by which the neighborhoods are explored (number of times the best layout is found with the corresponding strategy).

Inst.	Order of the neighborhoods											
	1	2	3	4	5	6	7	8	9	10	11	12
Car model 1												
1; 1		1										
1; 2								1				
1; 3						1						
2; 1			1									
2; 2				1								
2; 3					1							
3; 1					1							
3; 2							1					
3; 3							1					
4; 1	1											
4; 2					1							
4; 3							1					
Total	1	1	1	1	3	1	3	1	0	0	0	0
Car model 2												
1			1									
2	1											
3			1									
4										1		
5	1											
6												1
7				1								
8										1		
9	1											
10				1								
11						1						
12							1					
Total	3	0	2	2	0	1	1	0	0	2	0	1

(4) N_2, N_3, N_1, and N_4;

(5) N_3, N_1, N_2, and N_4;

(6) N_3, N_2, N_1, and N_4;

(7) N_4, N_1, N_2, and N_3;

(8) N_4, N_2, N_1, and N_3;

(9) N_4, N_1, N_3, and N_2;

(10) N_4, N_2, N_3, and N_1;

(11) N_4, N_3, N_1, and N_2;

(12) N_4, N_3, N_2, and N_1.

Table 6: Instances.

Car model	Order	m
1	1,...,5	5
	6,...,10	8
	11,...,15	10
	16,...,18	12
	19,...,21	15
	22	23
2	1,...,10	5
	11,...,20	8
	21,...,30	10
	31,...,35	12
	36,...,40	15
	41	22

The tests were performed only once for each instance and each configuration. A total of 288 runs were performed (2 days of computing time). The results are reported in Table 5. From these results, we can observe that while some sequences never led to the best layouts (configuration 9 and 11), none of the other sequences clearly dominated all the others. The best results are obtained with the first and seventh sequences, which both generate the best layout for four instances.

6.2. Computational Results

In this section, we report on the results obtained with our VNS algorithm on a large set of real instances. For these experiments, we used the best strategies and parameters identified from the previous tuning experiments. These configurations are summarized next:

(i) Use all the neighborhood structures $N_i,\ i = 1,\ldots,4$;

(ii) Explore the neighborhoods in the following order: $N_1,\ N_2,\ N_3,$ and N_4;

(iii) $t_{N_i} = 2,\ i = 1,\ldots,4$:

(a) $q_i^1 = 1,\ r_i^1 = 3,\ (u_{\min,i}^1, u_{\max,i}^1) = (10\%, 50\%)$;

(b) $q_i^2 = 3,\ r_i^2 = 3,\ (u_{\min,i}^2, u_{\max,i}^2) = (10\%, 50\%)$.

We generated 22 production orders using the set of pieces associated to the first car model in the same way as in the tuning experiments. The number m of different pieces varies from 5 to 23. Each production order is cut from three different hides, and hence, we have 66 instances defined from this car model. Forty-one production orders were generated using the pieces of the second car model. Since only one hide is used for each production order, in this case, we have 41 different instances derived from the second car model. In Table 6, we describe the characteristics of the production orders. In summary, our set of instances is composed of 107 different problem instances. As in the tuning experiments, we used a time limit of 600 s. For each instance, the tests were repeated four times. Hence, 428 runs were performed which took almost 3 days of computing time.

Table 7: Evaluating the performance of the VNS algorithm: car model 1 (part I).

Inst.	Initial solution			VNS					
				After 200 s		After 400 s		After 600 s	
	Pieces	Usage	t	Pieces	Usage	Pieces	Usage	Pieces	Usage
1; 1	22	59,53	12,86	22,50	59,61	25,00	60,01	27,50	61,09
1; 2	23	57,93	15,04	27,50	59,31	29,75	59,93	31,00	60,14
1; 3	25	61,57	19,12	25,00	62,34	25,75	62,95	26,75	63,12
2; 1	16	56,11	13,01	16,50	57,63	18,00	57,90	20,75	58,97
2; 2	17	54,43	14,43	19,00	55,67	24,25	58,27	25,25	58,95
2; 3	20	55,77	18,25	20,50	57,18	23,00	59,04	25,00	60,24
3; 1	9	46,02	9,00	14,75	54,60	15,50	55,26	17,25	55,19
3; 2	14	51,92	11,98	15,75	52,84	18,00	54,78	18,00	54,91
3; 3	18	54,77	14,89	18,50	56,04	22,00	57,41	23,50	59,76
4; 1	12	51,68	11,01	15,75	55,31	16,00	56,10	16,00	56,10
4; 2	13	50,13	12,45	16,00	52,58	18,25	54,68	19,00	55,51
4; 3	15	51,87	16,31	15,75	52,82	18,75	56,96	21,00	58,81
5; 1	13	51,51	9,06	21,00	60,67	25,25	60,81	24,50	61,77
5; 2	18	55,38	11,87	21,25	58,00	24,25	59,24	27,25	60,30
5; 3	26	60,99	16,86	26,00	60,99	34,50	61,53	34,75	61,56
avg.	17,40	54,64	13,74	19,72	57,04	22,55	58,32	23,83	59,10
6; 1	20	59,12	16,49	20,50	59,67	23,50	61,28	25,75	62,27
6; 2	24	61,56	18,72	24,75	61,82	25,00	62,26	25,00	62,26
6; 3	23	60,36	22,39	24,50	60,99	25,50	61,75	25,75	62,53
7; 1	17	57,50	18,43	18,25	59,07	19,25	61,77	19,75	62,22
7; 2	19	57,58	20,85	20,00	59,10	22,75	59,34	24,75	60,77
7; 3	22	58,61	26,46	24,00	61,27	24,25	61,55	24,25	61,55
8; 1	20	58,92	15,53	21,00	59,90	22,75	61,66	24,25	62,46
8; 2	22	60,06	19,53	22,75	60,32	23,00	61,03	25,50	61,57
8; 3	24	60,51	23,06	24,00	60,51	25,25	60,93	27,75	63,10
9; 1	12	50,78	16,58	13,75	54,04	14,75	54,47	15,25	54,90
9; 2	13	50,13	19,33	13,00	50,13	17,25	57,01	18,50	58,05
9; 3	14	50,40	23,81	14,50	51,13	18,00	54,88	21,00	58,78
10; 1	20	58,92	15,26	21,00	60,55	22,50	61,90	27,75	62,52
10; 2	23	60,65	19,40	24,50	63,06	25,25	64,01	25,25	64,01
10; 3	25	61,37	23,48	27,25	62,41	28,75	63,47	29,00	63,69
avg.	19,87	57,76	19,95	20,92	58,93	22,52	60,49	23,97	61,38

The results for the first car model are reported in Tables 7 and 8. Those for the second car model are given in Table 9. The column *Inst.* in these tables identifies the problem instance. For the first car model, we use the same notation to identify the instances as in the tables for the tuning experiments, namely, $x; y$, where x is the index of the production order and y the index of the hide. The tables present the results obtained with the constructive heuristic used to generate the first layout, and the results achieved by the VNS algorithm after 200 s, 400 s, and 600 s of computing time. The column *Pieces* indicates

Table 8: Evaluating the performance of the VNS algorithm: car model 1 (part II).

Inst.	Initial solution			VNS					
				After 200 s		After 400 s		After 600 s	
	Pieces	Usage	t	Pieces	Usage	Pieces	Usage	Pieces	Usage
11; 1	20	58,51	19,50	20,00	59,99	23,00	61,06	29,00	61,27
11; 2	24	60,48	23,76	24,00	60,48	24,00	60,48	24,00	60,48
11; 3	22	59,45	28,67	22,00	59,45	25,00	61,03	27,50	62,62
12; 1	19	60,18	21,63	20,25	61,61	22,00	61,79	28,50	62,54
12; 2	22	60,52	24,69	22,00	60,94	24,00	62,24	25,75	62,53
12; 3	25	61,38	31,02	25,00	61,38	25,00	61,38	30,75	64,33
13; 1	19	58,19	18,28	20,50	60,28	21,25	61,98	26,00	64,42
13; 2	21	58,27	21,64	21,00	58,27	23,75	59,61	26,75	61,56
13; 3	23	59,53	25,86	23,00	59,53	25,00	62,75	28,25	63,73
14; 1	20	60,19	20,74	20,00	60,19	25,25	63,77	25,25	63,77
14; 2	23	59,94	25,22	23,00	59,94	24,75	60,53	26,50	61,11
14; 3	23	59,62	29,89	23,00	59,62	25,50	61,53	27,75	63,07
15; 1	19	58,12	17,75	20,00	59,17	24,75	62,73	25,50	63,78
15; 2	24	61,55	22,05	24,00	61,55	26,00	62,41	26,00	62,41
15; 3	23	59,81	25,21	23,00	59,81	26,25	63,99	29,25	64,42
avg.	21,80	59,72	23,73	22,05	60,15	24,37	61,82	27,12	62,80
16; 1	19	58,19	24,57	19,75	59,44	20,50	60,71	22,25	62,76
16; 2	21	58,27	29,69	21,00	58,27	24,00	59,82	26,00	60,67
16; 3	23	59,53	37,42	23,00	59,53	24,50	59,75	27,25	63,65
17; 1	19	59,15	23,91	19,25	59,60	23,00	61,81	25,00	61,94
17; 2	21	59,09	28,67	21,00	59,09	23,50	61,76	24,75	62,55
17; 3	23	59,96	35,12	23,00	59,96	25,00	64,19	26,75	64,21
18; 1	19	58,12	22,15	20,00	59,65	21,00	60,40	24,50	62,63
18; 2	24	61,55	27,99	24,00	61,55	25,75	63,23	26,00	63,24
18; 3	23	60,33	33,58	23,00	60,33	24,50	62,97	26,25	62,17
avg.	21,33	59,35	29,23	21,56	59,71	23,53	61,63	25,42	62,65
19; 1	49	69,68	38,48	49,00	69,68	49,25	69,94	50,00	71,44
19; 2	45	67,27	41,82	45,00	67,27	46,50	67,48	46,25	67,54
19; 3	47	66,57	51,30	47,00	66,57	47,00	66,57	45,25	68,71
20; 1	76	73,17	51,42	76,00	73,17	78,00	73,55	80,50	74,21
20; 2	69	72,29	52,96	69,00	72,29	69,00	72,29	66,75	72,34
20; 3	81	72,42	64,93	81,00	72,42	81,00	72,42	95,75	72,66
21; 1	48	69,05	38,04	48,00	69,05	46,25	69,56	44,00	70,84
21; 2	43	67,57	40,68	43,00	67,57	43,00	67,57	43,00	67,57
21; 3	46	66,78	48,15	46,00	66,78	45,75	66,95	50,50	68,05
avg.	56,00	69,42	47,53	56,00	69,42	56,19	69,59	58,00	70,37
22; 1	84	73,88	99,30	84,00	73,88	84,00	73,88	84,67	74,08
22; 2	81	74,51	100,37	81,00	74,51	81,00	74,51	81,00	74,52
22; 3	90	73,78	116,76	90,00	73,78	90,00	73,78	90,00	73,86
avg.	85,00	74,06	105,48	85,00	74,06	85,00	74,06	85,22	74,15

Table 9: Evaluating the performance of the VNS algorithm: car model 2.

Inst.	Initial solution			VNS					
				After 200 s		After 400 s		After 600 s	
	Pieces	Usage	t	Pieces	Usage	Pieces	Usage	Pieces	Usage
1	25	54,81	21,96	26,50	56,26	28,50	57,56	29,75	58,17
2	31	53,18	32,05	33,00	55,69	34,00	57,29	35,00	58,63
3	26	55,20	22,69	27,00	56,07	29,25	57,86	31,50	58,94
4	29	55,72	24,84	30,25	56,53	30,75	57,60	33,75	59,06
5	36	55,84	34,11	35,75	55,94	35,75	55,94	36,50	56,50
6	34	61,82	33,11	37,00	63,86	37,25	64,29	37,25	64,40
7	29	60,80	23,00	29,75	61,41	31,50	62,64	30,25	63,61
8	51	58,22	58,78	51,00	58,22	53,00	60,08	54,00	60,60
9	54	57,07	65,56	54,00	57,07	55,25	59,51	57,25	60,84
10	33	61,56	29,51	34,75	62,58	34,75	63,62	34,50	64,01
avg.	34,80	57,42	34,56	35,90	58,36	37,00	59,64	37,98	60,48
11	22	54,70	24,32	24,50	57,78	24,50	58,36	24,50	58,53
12	39	54,99	52,38	39,00	54,99	40,50	55,89	40,75	57,94
13	41	54,86	52,21	41,00	54,86	41,75	55,37	43,75	57,58
14	39	60,98	53,89	39,00	60,98	41,00	62,11	41,00	62,11
15	39	52,86	58,48	39,00	53,06	39,00	53,67	39,50	53,91
16	33	63,44	34,33	33,00	63,44	33,00	63,93	33,25	63,94
17	35	62,49	33,23	35,00	63,21	35,75	64,29	38,00	65,83
18	52	58,93	56,82	52,00	58,93	54,00	60,31	54,75	60,42
19	50	55,95	76,14	50,00	55,95	50,50	57,28	51,25	58,41
20	29	60,30	31,73	30,50	60,86	32,50	61,84	36,50	63,73
avg.	37,90	57,95	47,35	38,30	58,41	39,25	59,30	40,33	60,24
21	25	57,38	33,19	25,00	57,38	28,00	60,96	27,75	61,42
22	39	54,99	60,14	39,00	54,99	39,25	55,38	41,25	56,86
23	50	59,07	69,43	50,00	59,07	50,00	59,07	50,00	59,17
24	31	64,53	37,23	31,00	64,53	31,50	64,67	31,50	64,67
25	39	52,80	70,57	39,00	52,80	38,50	55,24	39,50	56,01
26	30	62,50	35,50	32,00	63,09	36,25	64,45	37,75	64,75
27	33	64,86	39,78	33,50	65,14	35,75	66,41	36,00	66,54
28	50	56,69	74,65	50,00	56,91	50,00	57,00	51,00	57,71
29	39	61,50	63,95	39,00	61,50	39,75	61,79	41,50	62,84
30	32	62,11	41,71	32,25	62,25	33,00	62,76	39,00	65,52
avg.	36,80	59,64	52,61	37,08	59,77	38,20	60,77	39,53	61,55
31	31	61,41	43,98	31,00	61,41	35,00	63,42	36,75	63,84
32	39	54,35	73,25	39,00	54,35	39,00	54,35	39,50	55,97
33	48	59,16	77,47	48,00	59,16	49,00	59,68	50,00	60,21
34	31	65,21	45,09	31,00	65,21	31,00	65,21	31,75	65,33
35	54	60,74	91,89	54,00	60,74	54,00	60,74	54,00	60,74
avg.	40,60	60,17	66,34	40,60	60,17	41,60	60,68	42,40	61,22
36	31	61,99	53,78	31,00	61,99	35,00	64,07	38,00	64,18
37	48	59,44	91,20	48,00	59,44	48,00	59,44	49,50	60,03
38	47	56,75	105,12	47,00	56,75	48,50	57,40	53,00	58,91
39	32	64,72	51,62	32,00	64,72	32,00	64,72	38,50	67,23
40	54	60,42	106,47	54,00	60,42	54,00	60,42	54,00	60,42
avg.	42,40	60,66	81,64	42,40	60,66	43,50	61,21	46,60	62,15
41	34	64,87	73,84	34,00	64,87	34,00	64,87	39,00	66,89

the number of pieces placed on the hide. Column $Usage$ gives the percentage of material usage achieved by the corresponding layout. Column t is used in the part associated to the constructive heuristic to indicate the time (in seconds) needed by the heuristic to generate the initial solution. The line $avg.$ gives the average values for the group of instances that is above.

On average, the VNS algorithm is always able to improve the results obtained by the constructive heuristic. In Table 7, the best improvement is achieved for the instance 5; 1 with an increase of 10,26% in the total material usage. The VNS algorithm improves the initial solution quickly. Indeed, after 200 s of computation, the algorithm was already able to find a layout that is better than the initial solution. Furthermore, the solution is improved both after 400 s and 600 s of computation. These improvements tend to be less significant for the instances whose results are given in Table 8. On average, the results are always better than the initial solution, but the VNS algorithm takes more time to find a solution that is clearly better than this first solution. In this table, the best improvement is achieved for the instance 13; 1 with an increase of 6,23%. When the variety of pieces is larger, the constructive heuristic is usually able to find layouts that have already a high material usage. In these cases, the space for improvements is smaller, but the VNS algorithm is still able to improve the initial layout. For the hardest instances, those for which the constructive heuristic cannot find a good quality layout, the VNS algorithm appears to work as an effective improvement procedure. Not surprisingly, the time needed by the constructive heuristic to find an initial layout increases with m, the number of different pieces in the instance. This is due in part to the necessity of recomputing the NFPs and IFPs for each piece right after a piece has been placed on the hide. Note that the material usage achieved in the final layouts are comparable to the results obtained by human nesters. Furthermore, these results are achieved within the same time limit as that needed by two human nesters. One operator would need around 1200 s (twice our time limit) to achieve comparable results.

Similar results are obtained for the instances derived from the second car model. The VNS algorithm was always able to improve the initial solution except for two instances (35 and 40). In many cases, a better layout is found right after 200 s of computation. The best improvement is obtained for the instance 2 with an increase of 5,46%. Even in the instance with the largest variety of pieces (instance 41) the VNS algorithm was able to improve by 2,02% the material usage achieved in the first layout.

7. Conclusions

In this paper, we proposed the first local search metaheuristic for the general LNP using the real case study of a company that operates in the automotive sector. We described a constructive heuristic to generate feasible layouts, and we introduced the details of the different neighborhood structures that were explored. These structures rely on the sequence by which the pieces are placed on the hide and on the quality of the fitness of each piece. The fitness of a piece is measured by computing the intersection of an offset of the piece with the border of the hide and the border of the layout. To explore the different neighborhood spaces, we developed a VNS algorithm that searches for local optima in a systematic way on each one of these spaces. Our approaches were tested on a large set of real instances. The experiments show that the VNS algorithm improves the solutions provided by the constructive heuristic. Furthermore, the quality of the final layouts that are generated by this algorithm is competitive with the results achieved by human nesters.

Acknowledgments

This work was partially supported by the Algoritmi Research Center of the University of Minho for C. Alves and J. V. de Carvalho, by the Portuguese Science and Technology Foundation through the Doctoral Grant SFRH/BDE/15650/2007 for P. Brás and through the Research Grant UMINHO/BII/183/2009 for T. Pinto. It was developed in the Systems Engineering, Optimization and Operations Research Group.

References

[1] C. Alves, P. Brás, J. Valério de Carvalho, and T. Pinto, "New constructive algorithms for leather nesting in the automotive industry," *Computers and Operations Research*, vol. 39, no. 7, pp. 1487–1505, 2012.

[2] G. Wäscher, H. Haußner, and H. Schumann, "An improved typology of cutting and packing problems," *European Journal of Operational Research*, vol. 183, no. 3, pp. 1109–1130, 2007.

[3] J. Heistermann and T. Lengauer, "The nesting problem in the leather manufacturing industry," *Annals of Operations Research*, vol. 57, no. 1, pp. 147–173, 1995.

[4] A. Crispin, P. Clay, G. Taylor, T. Bayes, and D. Reedman, "Genetic algorithm coding methods for leather nesting," *Applied Intelligence*, vol. 23, no. 1, pp. 9–20, 2005.

[5] Z. Yuping, J. Shouwei, and Z. Chunli, "A very fast simulated re-annealing algorithm for the leather nesting problem," *International Journal of Advanced Manufacturing Technology*, vol. 25, no. 11-12, pp. 1113–1118, 2005.

[6] W. C. Lee, H. Ma, and B. W. Cheng, "A heuristic for nesting problems of irregular shapes," *Computer Aided Design*, vol. 40, no. 5, pp. 625–633, 2008.

[7] Z. Yuping and Y. Caijun, "A generic approach for leather nesting," in *Proceedings of the 5th International Conference on Natural Computation*, vol. 5, pp. 303–317, August 2009.

[8] N. Mladenović and P. Hansen, "Variable neighborhood search," *Computers and Operations Research*, vol. 24, no. 11, pp. 1097–1100, 1997.

[9] J. Beltran, J. Calderon, R. Cabrera, J. Perez, and J. Moreno-Vega, "GRASP/VNS hybrid for the strip packing problem," in *Proceedings of the 1st International Workshop on Hybrid Metaheuristics*, pp. 79–90, 2004.

[10] F. Parreño, R. Alvarez-Valdes, J. F. Oliveira, and J. M. Tamarit, "Neighborhood structures for the container loading problem: a VNS implementation," *Journal of Heuristics*, vol. 16, no. 1, pp. 1–22, 2010.

[11] F. Alvelos, T. M. Chan, P. Vilaça, T. Gomes, E. Silva, and J. M. Valério De Carvalho, "Sequence based heuristics for two-dimensional bin packing problems," *Engineering Optimization*, vol. 41, no. 8, pp. 773–791, 2009.

[12] E. Hopper and B. C. H. Turton, "Empirical investigation of meta-heuristic and heuristic algorithms for a 2D packing problem," *European Journal of Operational Research*, vol. 128, no. 1, pp. 34–57, 2001.

[13] F. Parreño, R. Alvarez-Valdes, J. M. Tamarit, and J. F. Oliveira, "A maximal-space algorithm for the container loading problem," *INFORMS Journal on Computing*, vol. 20, no. 3, pp. 413–422, 2008.

[14] P. K. Ghosh, "An algebra of polygons through the notion of negative shapes," *CVGIP: Image Understanding*, vol. 54, no. 1, pp. 119–144, 1991.

[15] J. A. Bennell, K. A. Dowsland, and W. B. Dowsland, "The irregular cutting-stock problem—a new procedure for deriving the no-fit polygon," *Computers and Operations Research*, vol. 28, no. 3, pp. 271–287, 2000.

[16] P. Hansen, N. Mladenović, and J. Pérez, "Variable neighbourhood search: methods and applications," *Annals of Operations Research*, vol. 175, no. 1, pp. 367–407, 2010.