Contents lists available at GrowingScience

International Journal of Industrial Engineering Computations

homepage: www.GrowingScience.com/ijiec

Hybrid heuristic for the one-dimensional cutting stock problem with usable leftovers and additional operating constraints

Massimo Bertolini[a*], Davide Mezzogori[a], and Francesco Zammori[b]

[a]*University of Modena and Reggio Emilia, Italy*
[b]*University of Parma, Italy*

C H R O N I C L E

Article history:
Received March 15 2023
Received in Revised Format July 6 2023
Accepted October 25 2023
Available online October 25 2023

Keywords:
Cutting Stock Problem
Simulated Annealing
Multiple Stock Lengths
Production Scheduling
Metal Bars

A B S T R A C T

The One-Dimensional Cutting Stock Problem consists in cutting long bars into smaller ones, to satisfy customers' demand, minimizing waste and cost. In this paper the standard problem is extended with the inclusion of additional constraints that are generally neglected in scientific literature, although relevant in many industrial applications. We also modified the standard objective function, by assuming that bars may have a different economical value and a different processing or shipping priority. Moreover, in line with business requirements, among solutions that generate the same cutting waste, we prefer the ones that generate a low number of leftovers, especially if leftovers are long, so that the likelihood of their reuse is high. To solve the problem, we propose a Simulated Annealing based heuristic, which exploits a specific neighbor search. The heuristic is implemented in a parametric way that allows the user to set the priorities of the bars and to choose the specific sub-set of constraints he or she wants to consider. The heuristic is finally tested on many problem instances, and it is compared to three benchmarks and to one commercial software. The outcomes of this comparative analysis demonstrate both its quality and effectiveness.

1. Introduction

Waste reduction is a key issue for many companies, both in terms of costs and of environmental sustainability. Materials' waste, in fact, must be either recycled or disposed of, generating costs with a significant impact on the income (Bertolini et al., 2021). The cutting process is undoubtedly one of the most wasteful industrial processes (Cheng et al., 1994) and, because of this, the minimization of waste from cutting activities is an issue of paramount importance, which has been studied since the early thirties (Kantorovich, 1960). How to optimally cut an object into smaller parts of specified size and shape is indeed a well-known operation research problem, generally referred to as the Cutting Stock Problem (CSP). In general, letting a 'cutting pattern' be a feasible combination of cuts, the optimal solution is the set of cutting patterns that generate all the required cuts, and that minimizes an objective function, which is usually the minimization of the cut material that cannot be used and that is wasted. Bar cutting is a typical case to which this problem applies, but similar considerations also apply to a variety of other industrial cases, such as cutting of trunks and wooden planks (Kokten & Sel, 2020), two-dimensional cutting of metal sheets (Cintra et al., 2008), or fabric cutting in the textile industry (Javanshir et al., 2010).

In this paper we focus on a common version of the cutting problem, namely the One-Dimensional Cutting Stock Problem (1DCSP), which concerns the cutting of standard-sized items into smaller pieces of specified size, as requested by the end customer. Standard-size items are generally rotation solids with axial symmetry such as bars, pipes, trunks, rolls, coils, and ropes. So, neither the shape nor the section of the standard-sized item is altered by the cuts, as they are always executed perpendicularly to the axis of symmetry. For the sake of clarity, a feasible solution of the 1DCSP is provided in Fig. 1, where

* Corresponding author
E-mail: massimo.bertolini@unimore.it (M. Bertolini)
ISSN 1923-2934 (Online) - ISSN 1923-2926 (Print)
2024 Growing Science Ltd.
doi: 10.5267/j.ijiec.2023.10.006

three bars of equal length are cut into six usable parts coloured in white. In the same figure, the generated waste is coloured in grey; these are residual bars that cannot be put back in stock, because they are too short to be reused.

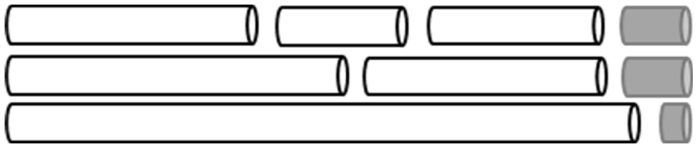

Fig. 1. An example of feasible cutting patterns generated by a perpendicular cut.

The 1DCSP is a generalization of the bin packing problem and, as such, it is known to be a NP-hard combinatorial problem (Martello et al., 1990a). Therefore, an exhaustive search of all the feasible cutting patterns would make the problem untreatable, even for a small number of orders and stocked bars (Abuhassan & Nasereddin, 2011). Nonetheless, a quasi-optimal solution can be obtained in a polynomial time using the famous 'columns generation scheme' first proposed in the seminal works by Gilmore & Gomory (1961). This method solves the 1DCSP starting from an initial set of cutting patterns, say , generated by a constructive heuristic. A linear programming model is then used to select the subset that satisfies the demand with the minimum waste. Next, an auxiliary knapsack problem, which exploits dual variables information, is solved and the best cutting pattern not yet included in is found. If the current best subset of cutting patterns cannot be improved by adding to the original set , the process stops and is declared optimal. Otherwise, it is extended with the inclusion of and the linear programming problem is solved again. This procedure is iteratively repeated until the stopping criteria is finally met, a condition that certainly occurs before all possible cutting patterns have been enumerated. The columns generation scheme is an extremely elegant and high performing approach that is used in many operational research problems, as in the works by Wilhelm (1999), Wilhelm (2001), Lübbecke and Desrosiers (2005) and Erhald (2021). However, this approach becomes untreatable as soon as additional operating constraints are included in the 1DCSP. For instance, the generation of non-used parts that may be returned to stock for future use, namely leftovers, is a typical feature that is difficult to deal with using the columns generation scheme, and which requires the use of fast performing heuristics (Cherry et al., 2009). In addition to leftovers, we also consider constraints related to the available inventory, and to the technical details needed to carry out the cut correctly. We also account for the fact that stocked bars may have different values and priorities and that, among solutions that are similar in terms of waste, the one that generates the minimum number of leftovers is generally preferable, especially if the generated leftovers are long, so that the likelihood of their future reuse is high. All these features are commonly required in industrial practice and cannot be neglected to obtain a qualitatively acceptable cut. Nonetheless, they are seldom, or never, considered in scientific literature. Therefore, their consideration can be seen as the main contribution of the paper.

Furthermore, we also provide an efficient heuristic to obtain, in an acceptable time, a feasible and very good solution. Specifically, the heuristic can be classified as a hybrid Simulated Annealing (SA), as the cutting patterns are generated using different approaches, but most of the time they are created using a SA that exploits a specific and effective neighbour search.

The remainder of the paper is structured as follows. At first, a brief overview on the 1DCSP is presented in Section 2. Then, an accurate description of the problem and of the additional constraints herein considered is provided in Section 3. The proposed heuristic is described in Section 4, and it is validated in Section 5. Specifically, it is compared with a commercial software and with three well-known algorithms (Next Fit Decreasing, First Fit Decreasing and Best Fit Decreasing) that are typically used to solve the bin packing problem. Conclusions and future research perspectives are finally drawn in Section 6.

2. Literature review

The 1DCSP can be formulated as follows: given a set of standard-sized bars (or solids with axial symmetry) available in stock and a set of sub-bars to be cut, the objective is to select an optimal set of cutting patterns that satisfies demands while minimizing waste (Haessler & Sweeney, 1991). Note that with the term 'cutting pattern' we refer to a specific way of cutting a bar of length L into sub-bars of lengths $l_i < L$. Also, we denote a cutting pattern with the mapping $c = \{l_1: x_1, l_2: x_2, \dots, l_i: x_i, \dots\}$, which specifies the number x_i of bars of length l_i that are cut from the original bar of length L. The first known formulation of the 1DCSP dates to 1931 and it can be ascribed to the Russian economist Kantorovich (1960). Since then, the 1DCSP has become one of the most studied operational research problems, due to its complexity and mathematical attractiveness and, above all, to its practical relevance. Literature in the subject matter is extensive, as demonstrated by the review by Sweeney & Paternoster (1992), which catalogues about 500 works (books, articles, and dissertations) concerning the 1DCSP. Over years, many linear programming models have been proposed to solve the 1DCSP. Among them, the most famous one is undoubtedly the 'delayed column generation scheme' proposed by Gilmore & Gomory (1961, 1963), which is based on an integer linear programming model, enhanced by an auxiliary knapsack problem that generates new solutions by exploiting dual variables information. However, the problem is known to be NP-hard, and therefore exact linear programming models are limited to a few peculiar and/or oversimplistic situations and they become inapplicable as soon as additional features are considered. Because of this, the problem is generally solved using approximate algorithms, which make use of upper and lower bounds to guide the search through the exploration of the solution space (Martello & Toth, 1990 b; Crainic

et al., 2007; Balogh et al., 2015). As a viable alternative, heuristics or metaheuristics have been increasingly used in recent years. According to Delorme et al. (2016), the first two heuristics were proposed by Roodman (1986) and by Vahrenkamp (1996). In the first study, an initial greedy solution is progressively improved with a local search; in the second one, new cutting patterns are generated using a random search based on the constructive heuristic proposed by Haessler (1975). After these publications, the interest in heuristics has exploded, and many extensions and readaptation of classic approximation algorithms have been proposed (Osogami & Okano, 2003; Bhatia et al., 2009; Fleszar & Charalambous, 2011). Beside these, also standard and/or population-based metaheuristics have been frequently used. Typical examples include Simulated Annealing (Kämpke, 1988; Loh et al., 2008), Tabu Search (Alvim et al., 2004; Jahromi et al., 2012), Ant Colony Optimization (Levine & Ducatelle, 2004) and Genetic Algorithm (Ülker et al., 2008; Quiroz-Castellanos et al., 2015). Some evolutionary approaches specifically designed for the 1DCSP have also been proposed, as in the work by Liang et al. (2002) and Sim & Hart (2013). Other interesting works focus on boundary conditions and/or additional constraints that frequently occur in real industrial cases. These contributions have great managerial implications, as they provide efficient and ready to use solutions. For instance, Cherry et al., (2009) and Cui & Yang (2010), introduced the concept of 'reusable leftovers', while Benjaoran and Bhokha (2014) considered the need to prioritise leftovers, anytime a new cut is made. This is a very common requirement, as many companies prefer using leftovers rather that cutting virgin items, if not necessary. A last notable work is the one by Garraffa et al. (2016), which shows that cutting waste can be reduced by optimizing the sequence in which cuts are made. This also streamlines production planning and shipping orders.

To the authors' view, the present work belongs to the latter stream of research. Our heuristic, in fact, is operation-oriented, and allows one to include many operating constraints identified in close collaboration with a famous Italian firm, that operates in the field of metal bars cutting.

3. Problem description and preliminary notations

In this section we clearly describe the additional constraints we included in the 1DCSP, and their industrial implications. Since the study of the 1DCSP arose from a research collaboration done with a metallurgical company, we will refer to the stocked items as 'bars'; nonetheless, the problem and the proposed solution can be generalized and extended to other cases with no need for further modifications. We also point out that, to the authors best knowledge, some of the considered aspects can be found in other scientific contributions, but they have never been treated together. For sake of clarity, all the symbols and notations used in this section (and in the rest of the paper) are accurately described in Appendix A. We just point out that the subscript i is used to indicate a bar ordered by a customer and the subscript j is used to denote a specific type of bar available in stock.

3.1. Additional constraints

The additional constraints included in the 1DCSP are listed below.

1. The number and type of the bars available in stock is limited.
2. The bars ordered by the end customer can be of different lengths l_j and they may also differ in terms of material m_j (e.g., steel, casted iron, copper, brass, etc.) and cross-sectional area a_j (e.g., circular, squared, hexagonal, etc.).
3. Materials and cross-sectional areas define the type of bars available in stock.
4. Leftovers are admitted. After cutting, the residual material can be returned to stock for future use, provided it is sufficiently long. The minimum acceptable length $\widetilde{w}$ is a fixed parameter greater than or equal to the minimum length that a customer can order.
5. Bars in stock may be either virgin bars or leftovers. Virgin bas are bars of standard lengths, while the length of the leftovers depends on the precedent cuts. Not to confuse the lengths of the bars ordered by the customer with the lengths of the bars available in stock, the first ones are denoted with the letter l_i, whereas the latter ones with the letter L_j.
6. The thickness of the cutting blade, say ε, is not negligible. This is a very important practical aspect, which is rarely considered in the literature. When cutting big and hard objects, a thick and resistant blade is needed. Hence, at each cut, a significant amount of material is inevitably lost, as Figure 2 shows. Note that the residual materials w (fielded in grey) can be either waste or a leftover, depending on its length.

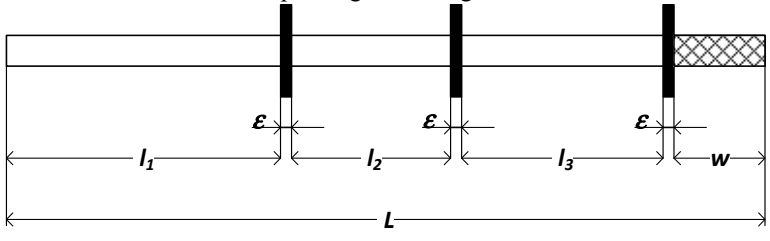

Fig. 2. An example of three perpendicular cuts, with blade width ε and residual material w.

7. Virgin bars must be trimmed on both sides before use. Specifically, two 'head-cuts' h (with $h > \varepsilon$) are needed to remove the extreme parts of the bar, which could be uneven, partially deformed, or damaged. As a convention, the quantity h (that is lost to trim the bar on a single side) includes the waste ε of the cutting blade.
8. To ensure stability, cantilever cutting is not allowed. Hence a gripping area is needed, as clearly shown in Figure 3, where F is the clamping force applied to the bar, and g is the length of the gripping area. According to this scheme, if head-cuts are needed, the following conditions must hold: $h \geq (g + \varepsilon)$.

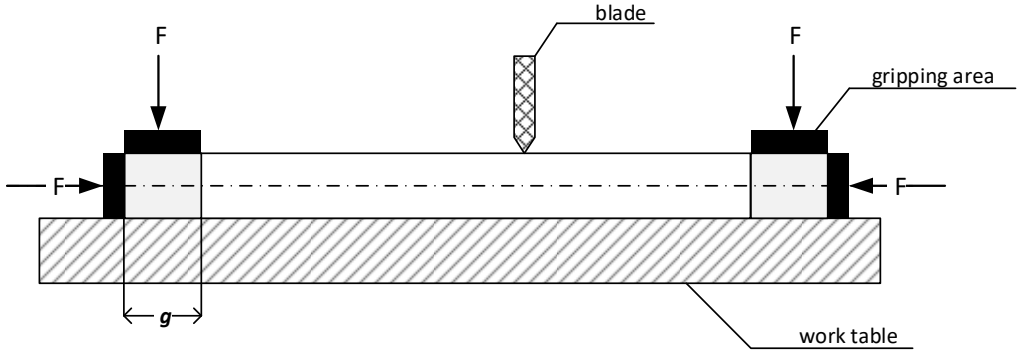

Fig. 3. Schematic representation of a cutting station.

Neither the head-cuts nor the gripping area have ever been considered before in the literature; yet they are very common technical requirements that must be fulfilled to provide a ready to use solution. Indeed, they entail strong limitations in the generation of feasible cutting patterns. To clarify this concept, suppose that a customer ordered a bar of l_i millimetres. To meet the request, if a leftover is used, the length L_j must be either equal to l_i or greater than ($l_i + g + \varepsilon$). In the first case there is a perfect match between the length of the ordered bar and that of the leftover; in the second case the leftover is long enough to be cut to size. Similarly, if a virgin bar is used, the following condition must hold $L_j \geq (l_i + 2h)$.

3.2. Optimization goal

The main objective is the minimization of the cutting waste generated during the cut. This issue will be better explained in Section 4; here we just want to introduce two concepts that will be extensively used in the rest of the paper. When a leftover is used, a cutting pattern is said to be 'perfect' if it generates only the unavoidable waste due to the thickness of the blade. This condition occurs when x sub-bars are generated with $(x - 1)$ cuts, and the total waste is only $w = \varepsilon \cdot (x - 1)$. Similarly, a cutting pattern is said to be 'almost perfect' if, in addition to the unavoidable waste, also the material underneath the gripping area g is lost. In this case, to generate x sub-bars, exactly x cuts are needed, and the total waste becomes $w = (\varepsilon \cdot x + g)$. This is graphically shown by Fig. 4(a) and Fig. 4(b).

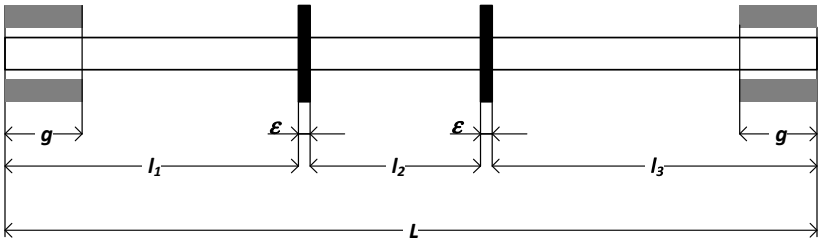

Fig. 4(a). An example of a perfect cutting pattern, obtained from a leftover with $x = 3$.

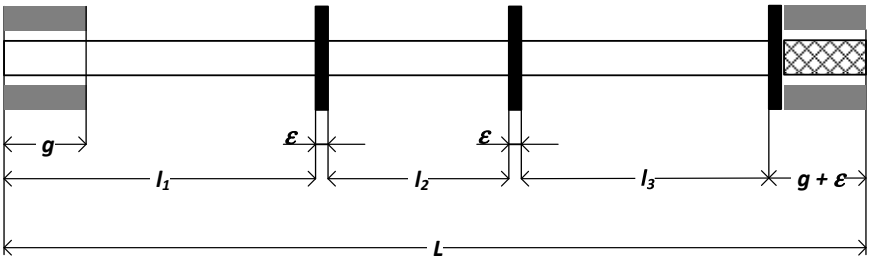

Fig. 4(b). An example of an almost perfect cutting pattern, obtained from a leftover with $x = 3$.

In case of virgin bars there is no distinction between perfect and almost perfect cuts. In fact, as Fig. 4(c) shows, since $g < h$, the part of the bar underneath the gripping area is always an unavoidable lost. Therefore, to obtain x sub-bars a minimum of

$((x-1)+2)$ cuts are required, where the last two cuts are needed to get rid of the bar's heads. Hence the total unavoidable waste becomes $w = \varepsilon \cdot (x-1) + 2h$.

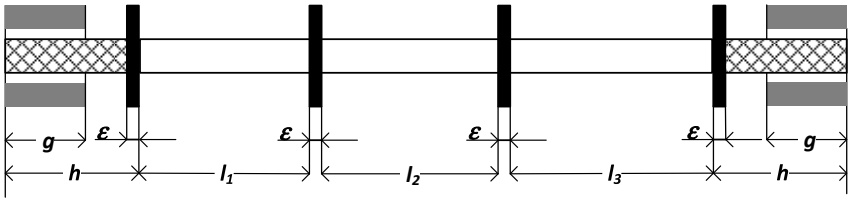

Fig. 4(c). An example of a perfect cutting pattern obtained from a virgin bar, with $x = 3$.

In all the other cases (i.e., cutting patterns that are neither perfect nor almost perfect), either a leftover or a waste w bigger than the unavoidable quantity is always created.

To conclude, we anticipate that, as a second optimization criterion, we will try to minimize the number of leftovers, especially if they are short bars of length close to the minimum acceptable one . Leftovers, in fact, need to be returned to stock, where they occupy almost the same space of virgin bars, thus reducing the effective storage capacity of the warehouse. This issue is particularly critical when an Automated Storage & Retrieval System (ASRS) is used. In this case, bars of equal length are grouped in bundles stored inside cases of standard size, which can be easily moved by a bridge crane equipped with extendable forks. Since each case can store one bundle at most, a bundle made of short bars occupies the same volume of a bundle made of long bars (Zammori et al., 2021).

4. Problem solution

In this section, the proposed algorithm is introduced starting from a high-level description. Afterwards, a more accurate and detailed discussion is provided.

4.1. Initialization and codification

All virgin bars (or leftovers) that are of the same length, material and cross-sectional area are indistinguishable and perfectly equivalent. Therefore, they can be considered as a single group, and the triplet (L_j, m_j, a_j) can be used to identify a specific type of bar. The available stock can therefore be coded by the mapping $B = \{(L_1, m_1, a_1): (n_1, v_1, p_1), \dots, (L_j, m_j, a_j): (n_j, v_j, p_j), \dots, (L_N, m_N, a_N): (n_N, v_N, p_N)\}$, which links the triplet (L_j, m_j, a_j) that codifies a bar of type j, to the triplet (n_j, v_j, p_j) that defines:

- the number of bars n_j available in stock,
- the nature of the bar, which can be either a virgin bar or a leftover. In the first case the Boolean value v_j is true, otherwise it is false.
- the priority p_j of the bar.

We note that priorities will be used to differentiate cutting patterns that are similar in terms of waste. Basically, if two cutting patterns produce the same amount of waste, the one obtained cutting the bars with the highest priority will be selected. Apart from this technicality (explained in Section 4.9), what is important to note is the fact that priorities must be assigned by the user, depending on the operational performances he or she wants to improve. Two alternative approaches are generally recommended:

- If the goal is to free up storage space, a higher priority should be assigned to leftovers and to short virgin bars, as these bars reduce the storage capacity of the warehouse (Benjaoran & Bhokha, 2014).
- If the goal is to increase productivity, higher priority should be assigned to the longest bars (Garraffa et al., 2016). In fact, most of the time needed to cut a bar is due: *(i)* to changeover activities, incurred every time a bar of different shape or length is cut, and *(ii)* to the adjustments needed to place and fix a bar on the cutting machine. The use of long bars is therefore beneficial, as it reduces the total number of bars needed to fulfil an order, thus decreasing the time needed both for changeovers and adjustments.

Following a similar approach, customers' requests are coded by the mapping $R = \{(l_1, m_1, a_1): n_1, \dots, (l_i, m_i, a_i): n_i, \dots, (l_M, m_M, a_M): n_M\}$, which associates to each type of ordered bar (l_i, m_i, a_i) the ordered quantity n_i.

Lastly, a cutting pattern (defined on a bar of type j) is codified by the mapping $c_j = \{l_1: x_1, \dots, l_i: x_i, \dots, l_M: x_M\}$, where each pair of values $(l_i,\ x_i)$ indicates how many sub-bars x_i of length l_i are cut from a bar of type j.

4.2. Preliminary operations

Neither the material m nor the cross-sectional area a affects the way in which cutting patterns are generated. Hence, in a first preliminary step, orders are partitioned into non-overlapping subsets, say $R_{m,a}$, that are homogeneous in terms of both m and a, i.e., $R = \bigcup_{m,a}(R_{m,a})$ and $\bigcap_{m,a} R_{m,a} = \emptyset$. A similar partitioning is made also on the stocked bars B. By doing so a set of independent subproblems $(R_{m,a}, B_{m,a})$ is defined, and each subproblem can be individually solved. The overall solution can be finally obtained bringing together the partial solutions. Owing to this fact, from here on we will implicitly refer to the solution of a generic subproblem and we will no longer make use of the subscripts a and m in the notation.

A second preliminary step is also needed to find the so-called 'perfect matches', that is orders of length l_i that can be fulfilled with leftovers of length $L_j = l_i$, without having to make any cuts. All the orders that can be fulfilled with a perfect match are removed from R, and the used leftovers are removed from B.

4.3. The main procedure

Basically, the optimization algorithm consists in an iterative procedure that ends when all the customers' orders have been fulfilled, or when the stock ends. In the first case, the list of the optimal cutting patterns is returned, whereas in the second case the problem is declared unfeasible, and the partial list of the optimal cutting patterns is returned, together with the backlog list of the unfulfilled orders.

4.3.1. Optimization tasks performed at each iteration

At each iteration t, and for each bar of type $j \in B_t$, an integer optimization problem is solved (as described in Section 4.4), and the best cutting pattern $c^*_{j,t}$ that can be obtained on a bar of length L_j using the ordered lengths l_i in R_t is obtained. Note that we included the subscript t in the notation, to keep track of the current iteration. So B_t denotes the available stock at t, R_t is the set of the unfulfilled orders at t, and $c^*_{j,t}$ is the cutting pattern with the minimum waste obtained from a bar of type j at iteration t. A simple example should clarify this concept. Let j be a 1,050 mm leftover and $R_t = \{500 : 4,\ 300: 3,\ \ 1000: 1\}$ be the set of the customers' requests. Therefore, 4 sub bars of 500 mm, 3 of 300 mm, and 1 of 1,000 mm were ordered. Assuming a blade $\varepsilon = 50$ [mm] and a gripping area $g = 100$ [mm], and recalling that leftovers do not need head-cuts, 5 alternative cutting patterns can be made on j, as shown below.

- $c^1_{j,t} = \{500 : 1,\ 300 : 0,\ 1{,}000 : 0\}$,
- $c^2_{j,t} = \{500 : 2,\ 300 : 0,\ 1{,}000 : 0\}$,
- $c^3_{j,t} = \{500 : 0,\ 300 : 1,\ 1{,}000 : 0\}$,
- $c^4_{j,t} = \{500 : 0,\ 300 : 2,\ 1{,}000 : 0\}$,
- $c^5_{j,t} = \{500 : 1,\ 300 : 1,\ 1{,}000 : 0\}$.

It is easy to see that the best one is $c^*_{j,t} = c^2_{j,t} = \{500 : 2,\ 300 : 0,\ 1{,}000 : 0\}$. This cutting pattern, in fact, is optimal as it generates only the unavoidable waste. Repeating the optimization procedure for each bar of type j in the set B_t, a list $C^*_t = [c^*_{1,t}, c^*_{2,t}, \dots, c^*_{j,t}, \dots, c^*_{N,t}]$ of N optimal cutting patterns is finally obtained. Next, this list is sorted, and the top element, say $c^*_{top,t}$, is selected and used (or repeated) as many times as possible. Please note that, by the expression *"repeating a cutting pattern c_j for n times"* we mean that n bars of type j are cut as defined by the cutting pattern c_j. The way in which C^*_t is sorted is fully detailed in Section 4.9, but basically, the generated waste is used as the main sorting criterion. If two or more cutting patterns generate the same waste, the priority p_j (of the bar on which they were cut) is used as the secondary sorting criterion. Concerning the number of repetitions, the top-ranked cutting pattern c^*_{top} is repeated until either the required bars terminate, or the number of generated sub-bars of length l_i exceeds the ordered quantity $n_{i,t}$. More formally, if $c^*_{top,t} \equiv c^*_{j,t}$ the number of repetitions r is computed as in (1):

$$r = min\left(n_{j,t}; \min_i\left(\left\lfloor \frac{n_{i,t}}{x_i} \right\rfloor\right)\right) \tag{1}$$

where $n_{j,t}$ is the number of bars of type j available in stock at iteration t, $n_{i,t}$ is the number of sub-bars of length l_i ordered by the customers that at iteration t has not been fulfilled yet, and x_i is the number of cuts of length l_i in $c^*_{j,t}$. For instance, referring to the example above, the cutting pattern $c^*_{j,t} = \{500 : 2,\ 300 : 0,\ 1{,}000 : 0\}$ could be repeated at most $r = \min\left(n_{j,t}, \min\left(\left\lfloor \frac{4}{2} \right\rfloor\right)\right) = 2$ times, as each cutting pattern generates only 2 sub-bars of length $l_i = 500$ $(x_i = 2)$ and 4 cuts of that length are needed $(n_{i,t} = 4)$. Lastly, R_t and B_t are updated, by removing the fulfilled requests $n_{i,t} = \left(n_{i,(t-1)} - r \cdot x_i\right)$ and the used bars $n_{j,t} = \left(n_{j,(t-1)} - r\right)$, respectively. Similarly, the optimal cutting pattern $c^*_{j,t}$ and the number of repetitions r are saved in the set of the optimal cutting patterns C_{Tot}, After updating these quantities, the iterative procedure

restarts and proceeds until all orders have been fulfilled. At this point the algorithm ends and the set C_{Tot} of the optimal cutting patterns is returned.

4.3.2. Some additional insights

It is interesting to note that, starting from the second iteration, there is no need to solve an integer optimization problem for each bar of type j in B. Let $c_j^* = \{l_1: x_1, \dots, l_i: x_i, \dots, l_M: x_M\}$, be the optimal cutting pattern generated for bar j at the previous iteration. Provided that the condition $x_i \leq n_i$ is satisfied for each required length l_i, the cutting pattern c_j^* is still optimal and should not be regenerated. Conversely, if the condition is not met, c_j^* is no longer useful, as it would generate more cuts of length l_i than needed. Only in this case a new optimal cutting pattern must be regenerated for a bar of type j. We also note that, as the algorithm gets close to the end, generating perfect or almost perfect cutting patterns becomes increasingly difficult. This is because most of the sub-bars of length l_j will have been cut and, similarly, the number of different lengths L_j in stock will have shrunk. With few lengths L_j to cut and even fewer length l_j to be cut, the generation of perfect cuts is hard. Therefore, even the top-ranked cutting pattern $c_{top,t}^*$ in C_t^* could generate either some waste or a leftover. In the latter case, r leftovers (one for each repetition of $c_{top,t}^*$) will be re-stocked. However, none of these r leftovers will be used to update the available stock B, and none of them will be considered in next generation of the optimal cutting patterns. This choice can be motivated as follows:

- First, the algorithm returns the set of the optimal cutting patterns, but it does not specify in which orders they should be processed (because this choice typically depends on exogenous logistic constraints, such as shipping times or the likes). So, the cutting sequence is unknown and there is no way to know, in advance, when a leftover will be returned to stock and will be available for future use.
- Secondly, leftovers are generally placed in a separate storage area, close to the cutting machine, where they wait to be bundled before being brought back to the warehouse. Their immediate reuse is therefore problematic, from a practical point of view.

For the sake of clarity, a block diagram presenting the main procedure is shown in Fig. 5.

The generation of the optimal cutting pattern (for a specific type of bar j) is a non-linear extension of the Bounded Knapsack Problem (BKP). Specifically, if the considered bar is a leftover, the problem can be formulated as follows.

Objective Function

$$max \left(\sum_{i=1}^{M} x_i l_i \right) \cdot \alpha + (1 - \alpha) \cdot w \tag{2.1}$$

Constraints

$$\left(L - \sum_{i=1}^{M} (l_i + \varepsilon) \cdot x_i \ - \varepsilon \right) \cdot z = 0 \tag{2.2}$$

$$\left(\sum_{i=1}^{M} (l_i + \varepsilon) \cdot x_i + g \right) \cdot (1 - z) \leq L \tag{2.3}$$

$$w = \min \left\{ \widetilde{w}, \left(L - \sum_{i=1}^{M} (l_i + \varepsilon) \cdot x_i \right) \cdot (1 - z) \right\} \tag{2.4}$$

$$x_i \leq n_i \quad \forall\, i \tag{2.5}$$

$$x_i \in \mathbb{N}_0 \quad \forall\, i \tag{2.6}$$

$$z \in \{0; 1\} \tag{2.7}$$

Where, in addition to the already introduced notation:

- x_i is the decision variable, that is the number of cuts of length l_i that are made.
- L is the length of the bar,
- M is the number of different ordered lengths l_i,
- w is the length of an eventual leftover, and $\widetilde{w}$ is the minimum admissible length for a leftover,
- z is a Boolean variable that is one for a perfect cutting pattern and it is zero otherwise,
- $\alpha \in [0;\ 1]$ is a weight factor.

Note that x_i and z are the decision variables, whereas w is a dependent variable, whose value directly depends on the one taken by x_i and z. All the others are known parameters.

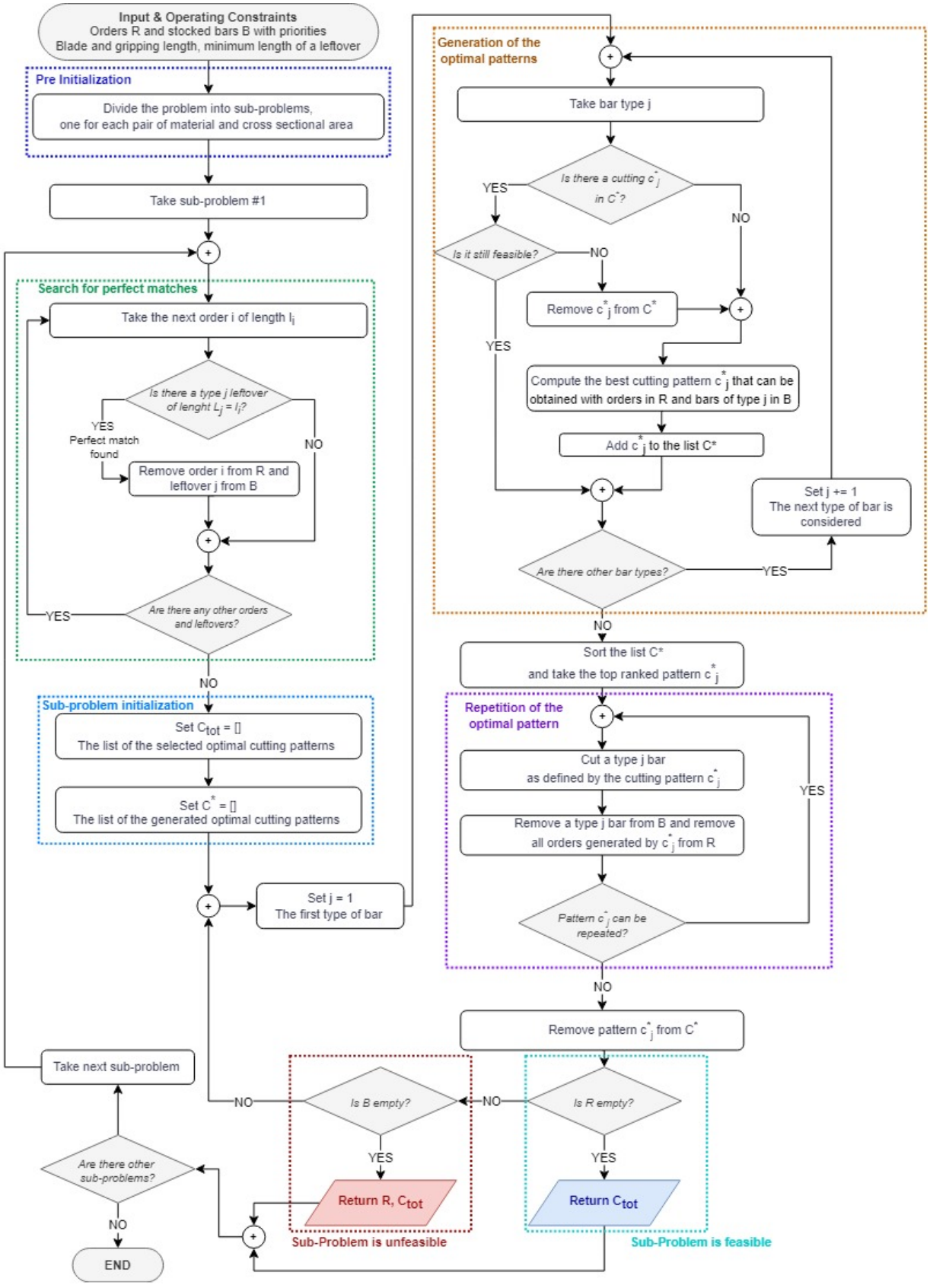

Fig. 5. Flow diagram of the algorithm

4.4. Generation of the optimal cutting patterns

As Eq. (2.1) states, the objective is to maximize the useful cut length, defined as a weighted sums of the cut lengths l_i and of the eventual leftover w. Clearly, by maximizing the useful length, the cutting waste is automatically minimized. Constraints (2.2) and (2.3) assure the feasibility of the generated cutting pattern. Specifically, constraint (2.2) and (2.3) apply to the perfect and non-perfect patterns, respectively. In the first case, in fact, the sum of the cut lengths l_i and of the cutting scrapes ε must be exactly equal to the length of the bar L. Also note that in a perfect pattern, the number of cuts is one less than the number of generated sub-bars. That is why the value of ε is finally subtracted in eq. (2.2). In the second case, instead, the sum of the cut lengths l_i, cutting scrapes ε, and gripping area g, must be less than or equal to the total length of the bar L.
The last three constrains assure that x_i is an integer, less or equal than the ordered quantity n_i and that z is Boolean.

As mentioned, this formulation is valid for a leftover; if the used bar is virgin, head cuts are also needed and constraints (2.2) and (2.3) must be changed as shown below.

$$\left(L - 2h - \sum_{i=1}^{M} (l_i + \varepsilon) \cdot x_i \; - \varepsilon\right) \cdot z = 0 \tag{2.8}$$

$$\left(\sum_{i=1}^{M} (l_i + \varepsilon) \cdot x_i\right) \cdot (1 - z) \leq L - 2h \tag{2.9}$$

We conclude this section with a last operating consideration. If, as is often the case, head cuts are made as soon as a virgin bar is received (that is before the virgin bar is taken in the warehouse for the first time), all bars can be considered as leftovers, provided that the original length L is rectified, by subtracting the unusable quantity $2h$. If so, the first formulation of the integer optimization problem can be applied to all the bars available in stock.

4.5 Solution approaches

To solve the integer optimization problem for the generation of the optimal cutting patterns, three different procedures are used, as shown below.

1. An exhaustive search.
2. A dynamic programming approach.
3. A metaheuristic based on Simulated Annealing (SA).

Whenever possible, cutting patterns are generated using either the exhaustive or the dynamic programming approach, as both procedures converge to the global optimum. Should there be multiple optimal solutions, the first one is returned. Conversely, the metaheuristic returns a feasible cutting pattern with a very good objective function, but optimality cannot be granted. Hence, from here on, this solution will be referred to as the 'pseudo-optimum'. To decide whether to use the exhaustive or the dynamic approach the total number of iterations, say I, needed to reach convergence is computed (using eq. (3) and eq. (4) provided in the next subsections). The approach with the smallest value of I is selected. Since the considered problem is NP hard, these two procedures can be used only for problem instances of medium-low size; a condition that frequently occurs when short bars are cut and/or when, after a certain number of iterations, only a limited number of lengths l_i remain to be cut. Anyhow, to prevent the computational time from growing too much, a threshold limit I_{max} is defined on the number of acceptable iterations. Anytime I exceeds I_{max}, the metaheuristic is used, and the search of the global optimum is abandoned, in favour of a pseudo-optimum generated in a reasonable amount of time. The choice of I_{max} is subjective, but it could be guided by the following approach.

- Z complexity levels $I = \{I_1, I_2, \dots, I_k, \dots, I_Z\}$ are selected, with $I_1 < I_2 < \cdots < I_k < \cdots < I_Z$.
- W problem instances are generated, for each complexity level I_k (for a total of $W \cdot Z$ problems).
- Each set of problems at level I_k is optimally solved with the dynamic programming approach, and the average computational time is determined.
- The biggest value of I, for which the average computational time is deemed acceptable, sets the threshold level I_{max}.

4.6. Exhaustive search

The exhaustive search evaluates all possible cutting patterns that can be generated on a bar of length L, when there are M ordered lengths l_i, each of which has to be cut n_i times, i.e., $R = \{l_1{:}\, n_1, \dots, l_i{:}\, n_i, \dots, l_M{:}\, n_M\}$. Recalling that a cutting pattern is defined as $c = \{l_1{:}\, x_1,\ l_2{:}\, x_2, \dots, l_i{:}\, x_i, \dots, l_M{:}\, x_M\}$, with $x_i \leq n_i$, all the cutting patterns can be easily generated, by considering all the values of $x_i \in \{0, 1, 2, \dots, n_i\}$ for each length l_i. Hence, the total number of iterations can be computed as in eq. (3).

$$\mathcal{N}_{ex} = \prod_{i=1}^{M}(n_i + 1) \tag{3}$$

All the $\mathcal{N}_{ex}$ patterns are generated one by one, discarding those that are not feasible. For the feasible ones the objective function is computed as in (2.1), and the best one is returned. To reduce the computational time, if a cutting pattern turns out to be perfect (as defined in Section 3.2), the procedure is immediately interrupted, and the cutting pattern is returned. A perfect cutting pattern, in fact, is certainly optimal.

4.7. Dynamic programming

We propose an extension of the well-known dynamic programming procedure used to solve the BKP, which is clearly described by Martello and Toth (1990 a). Dynamic programming is a technique that can be efficiently used if the main problem can be subdivided into overlapping sub-problems with an optimal substructure property. For the problem at a hand, given a bar of length L and M lengths to be cut $\{l_1, l_2, \dots, l_i, .., l_M\}$, a sub problem is defined as the search of the best cutting pattern $c^*_{i,\lambda}$ that can be made on a sub-bar of length λ, with $\lambda \in \{0, 1, 2, 3, \dots, L\}$, using cuts of length l less than or equal to l_i (i.e., the possible lengths that can be cut are $l_1, l_2, \dots, l_i$).

If, for convenience, we also consider the null cutting length l_0= 0, all the best cutting patterns $c^*_{i,\lambda}$ can be organized in a $((M + 1) \times (L + 1))$ matrix D, whose generic element (or problem state) $D[i, \lambda]$ is the best cutting pattern $c^*_{i,\lambda}$. In other words, the column index defines the length of the bar to be cut, whereas the row index defines which cutting lengths can be used (i.e., at row i only cuts of length l_0 to l_i) can be made. Therefore, all the elements in the first row, say D_0, are null (i.e., $D[0, \lambda] = 0 \; \forall \; \lambda$). In this row, in fact, only the null cutting length l_0 can be used. Conversely, the elements of the second row D_1 are obtained trying to add one or more cuts of length l_1 to the elements of D_0 and, similarly, the elements of the third row D_2 are obtained trying to add one or more cuts of length l_2 to the elements of D_1, and so on for all the subsequent rows.
More formally, the generic state $D[i, \lambda]$ is computed as follows:

- The maximum number of possible cuts of length l_i is computed as $n_{i,\lambda} = \lfloor \lambda/(l_i + \varepsilon) \rfloor$
- If $n_{i,\lambda} = 0$, a cut of length l_i cannot be made and so the value of $D[i, \lambda]$ is set to $D[(i - 1), \lambda]$. This value, in fact, is the best cutting pattern that can be generated on a sub bar of length λ using cuts of length $\{l_0, l_1, l_2, \dots, l_{i-1}\}$.
- Otherwise, $n_{i,\lambda}$ new cutting patterns $c^{\nu}_{i,\lambda}$ are generated adding ν cuts of length l_i to the optimal cutting patterns found at state $D_{i-1}[(i - 1), \; \lambda - (l_i + \varepsilon) \cdot \nu + g]$, with $\nu = 1, 2, \dots, n$. According to constraint (2.3), in fact, the cutting pattern in $D_{i-1}[(i - 1), \; \lambda - (l_i + \varepsilon) \cdot \nu + g]$ is the best one (made with cuts shorter than l_i) that can accommodate ν new cuts of length l_i.
- The objective function of all the new cutting patterns $c^{\nu}_{i,\lambda}$ is computed, and it is compared with that of the cutting patterns stored at state $D[i, \lambda - 1]$ and at state $D[(i - 1), \lambda]$ (i.e., the best solutions found at the previous step, and the best solutions found for a bar of the same length λ, but considering only cutting length shorter that l_i).
- Let $c^*_{i,\lambda}$ be the best objective function of the above mentioned $(2 + \nu)$ patterns. State $D[i, \lambda]$ is finally updated with this value (i.e., $D[i, \lambda] = c^*_{i,\lambda}$).

Note that if $\lambda = L$, constraint (2.3) is substituted by constraint (2.2), because in this case only perfect cuts can be considered. So, cutting patterns $c^{\nu}_{i,\lambda}$ are generated starting from the optimal cutting patterns found at state $D_{i-1}[(i - 1), \; \lambda - (l_i + \varepsilon) \cdot (\nu - 1)]$. Also note that, if a virgin bar is used, constraint (2.2) and (2.3) are substituted by constraint (2.8) and (2.9), respectively. Once the matrix D has been completed, the optimal solution can be read in the final state $D[(M + 1), (L + 1)]$ and, starting from this cell, the best cutting pattern can be reconstructing backtracking through the table. For the sake of clarity, a simple example is provided in Appendix B.

For what we said, it should be clear that the number of required iterations is proportional to the product of the number of cutting length and the length of the used bar, which is $O(M \cdot L)$. Also, since at each state at most $n_{i,\lambda}$ new cutting patterns are generated, in the worst-case scenario, the total number of cutting patterns to be considered can be obtained a in Eq. (4)

$$\mathcal{N}_{dyn} = \sum_{\lambda \in \Lambda} \sum_{i \in I} n_{i,\lambda} \tag{4}$$

with $\Lambda = \{0, 1, 2, \dots, L\}$ and $I = \{l_0, l_1, \dots, l_M\}$

Notably, if the Greatest Common Divisor (GCD) of L, ε, g and of all the cutting length l_i is greater than 1, the number of iterations $\mathcal{N}_{dyn}$ can be reduced. Indeed, rather than using a unitary step from one state to the next one in D_i, steps of size $\text{GCD}(L, \varepsilon, g, l_1, l_2, \dots, l_M)$ can be used, since GCD is the minimum increment that allows the introduction of a new cut. If so, the total number of iterations drops to $\mathcal{N}_{dyn}/\text{GCD}$.

4.8 Simulated Annealing (SA)

The adopted SA is very similar to the standard framework by Kirkpatrick et al. (1983), and its structure is represented in the pseudo code of Figure 6, written in a Python 3.7 style. Briefly, the algorithm is based on two solutions, namely the current best c^* and the current solution c, which may or may not coincide. In each epoch k, a neighbor search is performed on c, and the best neighbor c_{new} is selected. If c_{new} is either a perfect or an almost perfect cutting pattern, the SA ends and c_{new} is returned as the optimal solution. In both cases, in fact, it is almost impossible to further improve the objective function. Otherwise, three alternative conditions can occur, as defined next.

- if c_{new} is better than c^*, c_{new} replaces both c and c^*,
- if c_{new} is better than c but not than c^*, c_{new} replaces c,
- if c_{new} is feasible, but it is worse than c, the Boltzmann's rule (Eq. (5) provided in subsection 4.8.1) is used to decide whether it should still be accepted as the new current solution.

After this choice the current epoch ends and the temperature T is linearly decreased, according to eq. (6) provided in subsection 4.8.1. If T falls below zero, the SA ends and the current best c^* is returned as the pseudo-optimal solution. Lastly, unless c^* is a perfect or an almost perfect cutting pattern, a last neighbor search is made aiming to further improve the current best. To this aim the Advanced First Fit Decreasing method, described in Section 4.8.3, is used.

```
# INITIALIZATION
Generate an initial solution c_∅ # The initial solution can be an 'empty' cutting pattern.
Set Perfect ← False; c ← c_∅; c* ← c_∅; T ← T_0; # The initial cutting pattern and starting temperature.
While T ≥ 0:
    # GENERATION OF NEIGHBOR SOLUTIONS
    Set Available_Lghs ← shuffle([l_1, l_2, ..., l_M]) # The shuffled list of the lengths that must be cut.
    For l in Available_Lghs:
        Set OFs ← [ ] # An empty list; it will be filled with the value of the objective function of the neighbors.
        Set Nghs ← neighbor_search(l, c) # A function returning a list of neighbors, as detailed in (4.8.2).
        ''' The value of the objective function of(n)of each neighbor is stored in the list OFs Note that if a neighbor
        is unfeasible, its objective function is negative. '''
        For ngh in Nghs: OFs.append(of(ngh))
        Set m ← max(OFs)
        If m > 0: # A feasible solution has been found; the for loop ends and the best neighbor is returned.
            Set c_new ← Nghs[OFs.index(m)] # The best neighbor is found.
            Break # Exit the for loop.
    Else: # This part is executed if none of the length in Available_Lghs has generated a feasible solution.
        Break # The while loop ends.
    #ACCEPTANCE STEP
    If m > of(c*):
        Set c* ← c_new; c ← c_new;
        If perfect_cut(c*) Or almost_perfect_cut(c*):
            Set Perfect ← True
            Break # The while loop ends, c* cannot be improved any more.
    Else:
        If accepted(c_new, c): # The Boltzmann's acceptance rule is applied, as in eq. (5)
            Set c ← c_new;
    Set T = (T − ΔT) # The temperature is decreased, and the while loop proceeds.
# LAST IMPROVEMENT
If not Perfect:
    For r in range(1 to R): # R is a user defined parameter, less than the number of cutting lengths M.
        Set c_new ← Affd(c*, r) # The advanced first fit decreasing procedure is applied on c* (see section 4.8.3).
        If of(c*) < of(c_new) Set c* ← c_new
Return c* # The optimization procedure ends, and c* is returned.
```

Fig. 6. Pseudo-code of the proposed SA.

4.8.1 Basic features of the SA

In each epoch k of the SA we made use of:

- the Boltzmann's acceptance rule, with exponentially decreasing acceptance probability P, as in Eq. (5),
- a linear decreasing temperature, as in Eq. (7).

$$P = exp\left(\frac{-\Delta E_k}{\beta T_k}\right) \tag{5}$$

$$\Delta E_k = \frac{OF(c_k^*) - OF(c_k)}{OF(c_k^*)} \tag{6}$$

$$T_k = T_{k-1} - \Delta T \tag{7}$$

where:

- ΔT is the constant cooling factor,
- β is a non-negative parameter that plays the role of the Boltzmann's constant in the annealing process,
- ΔE_k is the variation in energy level, expressed as the percentage worsening of the objective function.

Since c, $c^*$and T dynamically changes as the SA progresses, in the notation we made use of the subscript k to indicate the value taken by these variables at the current epoch k.

4.8.2 Initialization and neighbor search

To initiate the SA, an initial cutting pattern must be defined. This initial solution could be generated in any way but, in virtue of the peculiar neighbor search described next, even an empty cutting pattern, say $c_\emptyset = \{l_1: 0,\ l_2: 0, \dots, l_M: 0\}$ can be used, as suggested in the pseudo-code of Fig. 6. Concerning the neighbor search, first we try a basic strategy and next, if this strategy is ineffective, a more complex one is performed. Specifically, to generate a neighbor solution, say $c_{k,i}^+$, a cut of length l is randomly selected from the set $\{l_1,\ l_2, \dots,\ l_i, \dots, l_M\}$. Let this length be l_i, the new cutting pattern is obtained by incrementing of one the number x_i of cuts of length l_i in c_k, that is $c_{k,i}^+ = \{l_1: x_1,\ l_2: x_2, \dots,\ l_i: (x_i + 1), \dots, l_M: x_M\}$. The notation $c_{k,i}^+$ is used, expressively, to indicate that the neighbor is obtained adding one cut of length l_i to c_k. This basic approach works well during the early epochs of the SA, when new cuts are added to an initially empty patter $c_\emptyset$. Conversely, when the sum of the lengths of the cuts (performed on c_k) approaches the length of the bar L, it becomes difficult to find enough space to add and additional cut of length l_i. Therefore, quite often the neighbor $c_{k,i}^+$ results unfeasible. If so, an opposite generation strategy is used and, rather than increasing the number x_i of the randomly selected cut l_i, a new solution, say $c_{k,i}^-$, is obtained by removing one cut of length l_i from c_k, that is $c_{k,i}^- = \{l_1: x_1,\ l_2: x, \dots,\ l_i: (x_i - 1), \dots, l_M: x_m\}$. Next, three alternative neighbor solutions are generated from $c_{k,i}^-$ using the following approaches:

1. Inc_1. For each value $v \in \{1, 2, \dots, M\}$, with $v \neq i$, a neighbor solution is generated increasing by one the number of cuts x_v of length l_v in $c_{k,i}^-$.
2. Inc_2. The neighbor is generated with the same strategy used by Inc_1, but this time the number of cuts x_v of length l_v is increased by two.
3. Inc_1_1. For each pair of values (v, z) with $v \neq i, z \neq i, v \neq z$, a new solution is generated increasing by one the number of cuts x_v and of x_z in $c_{k,i}^-$.

Hence, at total of $(0.5M^2 + 0.5M - 1)$ neighbors are generated; specifically, $(2M - 2)$ are generated by Inc_1 and Inc_2, and the remaining $[0.5 \cdot (M - 1) \cdot (M - 2)\ ]$ are generated by Inc_1_1. The best neighbor is selected and used as candidate solution for the acceptance step. If none of the generated neighbors is feasible, another length l_i is randomly selected and the neighbor search is repeated. So, at most $(0.5M^3 + 0.5M^2 - M\ )$ neighbors are generated during each epoch.

4.8.3 Deep neighbor search

In the event that the value c^* returned by the SA is neither an optimal nor an almost optimal cutting pattern, an additional deep neighbor search is executed on c^*, using an approach readapted from the Advanced First Fit Decreasing (AFFD) procedure by Cherri et al. (2009). Let q be a natural number in the range $[1, Q]$, with Q a user defined parameter. First, the q longest cuts are removed from c^*, to generate a sub cutting pattern c_-^*. Next, the removed cuts are virtually joined with the residual material (either waste or leftover) of the original cutting pattern c^* to create a new dummy bar d. A simple example should clarify this concept. Let us consider the cutting pattern $c^* = \{500 :\ 1,\ 300 : 2,\ 200 : 4\}$, and let $q\ =\ 2$. Therefore, the two longest cuts to be removed are the 500 and the 300 mm ones, and the sub cutting pattern becomes $c_-^* = \{500 : 0,\ 300 : 1,\ 200 : 4\}$. Also, joining the removed cuts with the remaining material a dummy bar d of length $L = (500 + 300 + w + 2\varepsilon)$ is generated, as clearly shown in Fig. 7. The best cutting pattern that can be generated on the dummy bar d, say c_d^*, is found using one of the three optimization approaches described above. In doing so, all the lengths l_i in R can be used, except the ones that have already been used in the sub cutting pattern c_-^*. Lastly, c_d^* and c_-^* are rejoined to form a real bar and, in case of improvement, the overall best c^* is updated accordingly. After the deep search has been repeated for all values of $q \in [1, Q]$, the optimization process ends, and the current best c^* is returned as the pseudo-optimal solution. To conclude we note that, since the dummy bar d is short, c_d^* is almost always generated using the exhaustive or the dynamic approach; hence the AFFD procedure has a very fast execution time.

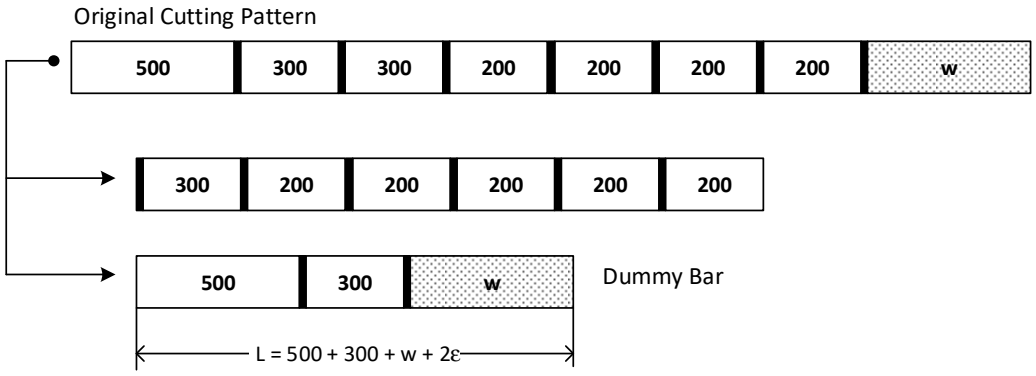

Fig. 7. Generation of the sub cutting pattern and dummy bar.

4.9 Selecting the best cutting pattern to replicate

After executing the optimization procedure on all bars in B, a set of N optimal or nearly optimal cutting patterns $C^* = \{c_1^*, c_2^*, \dots, c_N^*\}$ is obtained. Therefore, a criterion is needed to select the overall best patter in C^* that will be replicated as many times as possible. There is no doubt that preference should be given to the perfect and, next, to the almost-perfect cutting patterns. Hence, if in C^* there is only a perfect cutting patter, it is immediately selected as the overall best. Conversely, if there are two or more perfect cutting patterns, they are sorted in descending order of the priorities p_i of the bars from which they were cut, and the first one is selected. If there are no perfect cutting patterns, the same approach is repeated on the almost-perfect ones.

The choice is less obvious if in C^* there are neither perfect nor almost-perfect cutting patterns. In this case, in fact, a trade off must be solved, since there is the need to compare cutting patterns of different length that generate either waste or a leftover. To this aim we recall that the longer is a leftover the better it is. A short leftover (close in length to the minimum acceptance size $\widetilde{w}$), in fact, has a bad volumetric occupation of the storage space and has little chances to be reused in the next future. Hence, it could remain unused for a long time inside the warehouse. In view of this, and in agreement with the technical staff of the company for which the algorithm was developed, within certain limits, a cutting pattern that generates a small waste (i.e., a solution that approximates an almost perfect cut) is generally preferable than a solution that generates a short leftover. To concretise this idea, we suggest using the utility function U, computed as in eq. (8), to compare alternative and sub-optimal cutting patterns:

$$U = \begin{cases} 1 - \left(\frac{w}{\widetilde{w}}\right), & if\ w < \widetilde{w} \\ \min\left\{s_m \cdot \left(\frac{w}{\widetilde{w}}\right), \quad s_M\right\}, & if\ w \geq \widetilde{w} \end{cases} \tag{8}$$

where w is the length of the unused material (either waste or leftover), $\widetilde{w}$ is the minimum allowed length of a leftover, s_m and s_M are the minimum and maximum utility of a leftover, with $0 \leq s_m < s_M < 1$.

According to Eq. (8):

- The utility of the generated waste linearly decreases (from one to zero) as the length w increases. More precisely, $\lim_{w \to \widetilde{w}^-} U = 0$, which means that the utility of a piece of waste that is just shorter than the minimum acceptable leftover $\widetilde{w}$ is null.
- The utility of a leftover starts at its minimum value s_m (when $w = \widetilde{w}$) and linearly increases, until the maximum s_M is reached for a length $w \geq \widetilde{w} \cdot \left(\frac{s_M}{s_m}\right)$. In this way long leftovers are preferred over the short ones.
- A very little waste, such that $w < \widetilde{w} \cdot (1 - s_M)$, is always preferable to a leftover regardless of its length.

Also in this case, should there be cutting patterns with the same or very similar utility, the sorting is based on the priority p_i of their bars.

5. Computational Experiments

The algorithm was tested on many problem instances, and it was compared to three well-known benchmarks and to a commercial software for bars cutting, that is very common in the steel industry. To this aim, the algorithm and the three

benchmarks were implemented in Visual Basic.Net and they were executed on a personal computer Intel Dual Core i3 CPU at 2.4GHz with 8Gb RAM powered by Windows 7 operative system. All the operational details are provided next.

5.1 Parameters setting

To fine-tune the parameters of the algorithm, we considered a set of 100 cutting problems provided to us by the partner company. The dimension of these problems is shown in Table 1.

Table 1
Dimensions of the problems used to fine tune the operating parameters

Problem's feature	**Min. Value**	**Max. Value**
Number of bars available in stock	200	500
Different bars' lengths L_i	10	20
Different ordered lengths l_i,	15	30
Number of cuts to be made	2,000	4,000

With the objective to find a solution within the maximum time deemed acceptable in industrial applications, we set $I_{max} = 100{,}000$, $Q = 2$, $\beta = 1$, $T_0 = 1$ and $\Delta T = 0.02$. With these parameters, in fact, any problem of the original set can be solved in no more than 1 minute of time, and the average duration of the exhaustive procedure stays in in the range [0.4; 1.2] seconds. We also note that a value of $\beta = 1$ is very frequent because, joined with an initial temperature $T_0 = 1$, it assures an acceptance probability of around 90% for a solution worse than 10% of the current best (Bertolini et al., 2019). We also note that, with the selected parameters, the acceptance probability drops below 1% when the temperature reaches its ending value of $T_{end} = 0.02$.

All the other parameters (more subjective and user-dependent) were set in agreement with the experts of the partner company. Specifically, the weight factor (used in the objective function) was set to $\alpha = 1$, and the minimum and maximum utility of a leftover were set to $s_m = 0.2$ and $s_M = 0.6$, respectively.

5.2 The benchmark algorithms

The proposed algorithm was compared with three classic heuristics, which are typically used for the solution of the bin packing problem (Martello & Toth 1990b). These are: Next Fit Decreasing (NFD), First Fit Decreasing (FFD), and Best Fit Decreasing (BFD). As a fourth and last benchmark, we also considered RealCut1D, a frequently used commercial software, developed, and commercialized by the Optimal Programs company. A brief overview of each of these methods is provided next.

5.2.1 Next Fit Decreasing, First Fit Decreasing, and Best Fit Decreasing: a brief introduction

NFD is a very simple and straightforward approach. Basically, any time an order (i.e., a cut of length l_i) is considered, a check is made to see if it fits in the currently used bar. If so, the order is assigned to the current bar and the next order is considered. Otherwise, the bar is 'closed', and the order is assigned to a new bar. The orders assigned to a bar define a cutting pattern, which is returned (as output) anytime a bar is closed. For the sake of clarity, the pseudo code of NFD is provided in Figure 8, limited to the case of a feasible problem.

Set $R \leftarrow sort([l_1, \dots, l_i, \dots], length)$ *# The list of the orders, sorted in ascending order of their length.*
Set $B \leftarrow sort([b_1, \dots, b_j, \dots], length)$ *# The list of the bars, sorted in ascending order of their length.*
Set $b \leftarrow B.pop()$ *# The last bar in B (i.e., the longest one) is used as the initial 'open' bar* b.
Set $c \leftarrow c_\emptyset$ *# The initially empty cutting pattern.*
Set $C^* = [\,]$ *# An empty list that will be filled with the generated cutting patterns.*

While $len(R) > 0$:
    Set $r \leftarrow R.pop()$ *# The last order in R (i.e., the longest one).*
    If $fit(r, b)$: *# function fit() returns true if there is enough space to accommodate* r *in the current bar* b.
        Set $c \leftarrow add(r, c)$ *# The order is added to the cutting pattern.*
    Else:
        $C^*.append(c)$ *# The cutting pattern is added to the list.*
        Set $b \leftarrow B.pop()$ *# A new bar is opened.*
        Set $c \leftarrow add(r, c_\emptyset)$ # For simplicity we assume that a new order always fits on a new bar.

Return C^*

Fig. 8. Pseudo code of the NFD algorithm.

FFD is very similar to NFD. The only difference is that a bar remains open until its residual length is greater than or equal to the minimum ordered length $min_i(l_i)$ added to the blade's thickness ε. Only in this case, in fact, a bar cannot accommodate any other cut. Therefore, any time a new order is taken, all the bars that are still open are considered, and the order is assigned to the first bar that has enough space to accommodate it.

Concerning BFD, what distinguishes it from FFD is the idea to place new orders in the 'tightest' open bar. Specifically, any time an order is taken, all the 'open' bars are considered, and the order is assigned to the bar that, after the assignment, remains with the shortest residual length.

5.2.2 RealCut1D

RealCut1D is a commercial software known to be well-designed and efficient. Although it is not open-source software, and the operation of its optimization engine is unknown, its functioning is well aligned with our approach, as it considers the thickness of the blade, and it also manages bar priorities. Unfortunately, it does not consider head cuts and so the stocked bars must be pre-trimmed before they can be used as input. The software is widely used by many companies operating in the steel and wood sectors, as well as the partner company that inspired this work. Because of this, we chose RealCut1D as a worthy benchmark.

5.3 Results and discussion

5.3.1 Generated scenarios

To validate our approach, we generated and solved a comprehensive set of cutting stock problems, with size and complexity typical of the steel industry. As Table 2 shows, problems were organized in four macro-scenarios, which differ in the range of the lengths used for the stocked bars L_j and for the ordered bars l_i.

Table 2

The four generated scenarios.

Scenario	Length L_j of stocked bars	Length l_i of customers' orders
1	Short [100 mm - 2,500 mm]	Short [50 mm - 2,300 mm]
2	Short [600 mm - 4,000 mm]	Long [100 mm - 3,500 mm]
3	Long [1,200 mm - 6,500 mm]	Short [50 mm - 2,300 mm]
4	Long [1,200 mm - 6,500 mm]	Long [100 mm - 3,500 mm]

Both in scenario 1 and scenario 2, the length L_j of the stocked bars is labelled as Short, although the used ranges differ. A different range was purposely chosen, to avoid that an unproper overlapping of the range used for L_j and for l_j could cause a frequent generation of unfeasible problems (with ordered lengths longer than the lengths of the bars available in stock).

A brief discussion on the choices we made is provided next.

- Scenario 1. Stocked bars are short, and their length is similar to those ordered by the customers. So, the algorithms are tested in a hard condition because the chance of finding a perfect cutting pattern is low and the probability of generating cutting waste is high.
- Scenario 2. This scenario is like the previous one, but the expected quantity of generated waste is even higher. The likelihood of generating unfeasible customers' orders is also high.
- Scenario 3. Ordered lengths are short, relative to the length of the stocked bars. Hence, the probability to find perfect cutting patterns is rather high. For the same reason (i.e., short, ordered lengths) the generation of leftovers is expected, too.
- Scenario 4. Ordered lengths can be similar but also much shorter than those of the stocked bars. So, the number of perfect cutting patterns and of sub-optimal cutting patterns (including either waste or leftovers) should be similar.

5.3.2 Data generation and parameters' setting

For each scenario and for each problem instance, data were generated using the following procedure.

- To generate the initial stock:
  - 10 lengths $\{L_1, L_2, \dots, L_i, \dots, L_{10}\}$ were defined, by dividing the range of Table 2 into 10 equal parts,

- the number n_i of stocked bars of length L_i (for each i = 1, …, 10) was randomly generated, using a uniform distribution between 10 and 200.

- To generate the customers' orders, 25 lengths $\{l_1, l_2, \dots, l_j, \dots, l_{25}\}$ were defined using the same approach, and the number of orders n_j was randomly generated using a uniform distribution between 5 and 150.

For each scenario, 50 feasible problems were generated, and the following realistic parameters were used: $\varepsilon = 5$, $g = 10$, $\widetilde{w} = 50$ and $h = 0$.

As mentioned above, head-cuts were not considered, because RealCut1D is unable to handle them. Concerning priorities, we stress that in our algorithm they are used just as a second sorting criterion, and they are not accounted for in the objective function. A similar behaviour is also provided by RealCut1D that tends to prioritize longer bars, regardless of the priority assigned to them. In view of this and considering that priorities are not accounted for by the other three benchmarks, we chose to give the same priority to all the stocked bars, so as to assure a fair comparison of the tested algorithms.

5.3.2 Performance measures and results

The comparison was made with respect to three performance measures: *(i)* TL [mm], the total or cumulative length (in millimetres) of the stocked bars used to fulfil customers' orders, *(ii)* TW [mm], the total cutting waste, and *(iii)* RWL, the dimensionless ratio between TW and TL (i.e., the percentage generated waste). Obtained results (averaged over the 50 solved problems) are reported in Table 3, where, for each scenario, the best value of each performance measure is highlighted in bold.

Table 3
Results of the numerical experiments, averaged over the 50 solved problems

		SCENARIO			
		1	**2**	**3**	**4**
Proposed approach	**TL [mm]**	969,620	1,222,663	2,063,172	1,883,177
	TW [mm]	12,834	15,491	11,304	11,117
	RWL [-]	1.32%	1.27%	0.55%	0.59%
RealCut1D	**TL [mm]**	853,390	1,124,313	1,942,574	1,702,132
	TW [mm]	16,067	17,997	14,998	12,818
	RWL [-]	1.88%	1.60%	0.77%	0.75%
Next Fit Decreasing	**TL [mm]**	1,013,000	1,306,330	2,136,340	2,091,560
	TW [mm]	14,527	17,032	13,116	13,861
	RWL [-]	1.43%	1.30%	0.61%	0.66%
First Fit Decreasing	**TL [mm]**	796,954	1,034,670	1,801,620	1,677,340
	TW [mm]	17,953	18,832	14,982	14,047
	RWL [-]	2.25%	1.82%	0.83%	0.84%
Best Fit Decreasing	**TL [mm]**	826,404	1,077,638	1,794,700	1,733,814
	TW [mm]	21,832	19,621	18,296	14,496
	RWL [-]	2.64%	1.82%	1.02%	0.84%

First Fit Decreasing is the approach that uses the smallest number of stocked bars. In fact, with respect to TL, it outperforms both the proposed approach and RealCut1D. However, this performance is negatively counterbalanced by two negative issues: *(i)* cutting waste is much higher than that generated by the other approaches, and *(ii)* usable leftovers are hardly ever generated (i.e., cutting patterns are almost always characterized by some waste).

Next Fit Decreasing shows a diametrically opposed behaviour, as it uses the largest number of bars, and it also generates leftovers disproportionately. This is because, as soon as an order cannot be placed on the current bar, the current bar is closed and never considered again to satisfy future orders.

Our approach is the one with the best overall performance. Indeed:

- It obtains the smallest value (i.e., the best) of both TW and RWL, consistently in all the investigated scenarios.
- Although TL is sub-optimal, it does not deviate much from the value obtained by the other methodologies, especially from the value obtained by the commercial software RealCut1D.

Concerning the last point, we also note that the tendency to use more bars is a direct consequence of the preference we gave to the generation of long leftovers instead of short leftovers or waste. As previously explained, this is a reasonable choice that is frequently required in the industry. Also, since the strength of this tendency is defined through the weight factor alpha, if needed, it could be mitigated by decreasing this parameter. For the sake of completeness, these conclusions have been graphically summarized by the box plots of Fig. 9, Fig. 10, and Fig. 11.

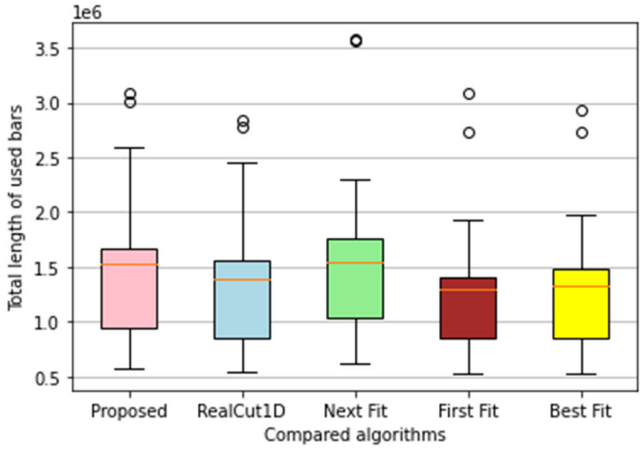

Fig. 9. Box plot of the results, relative to the TL performance measure.

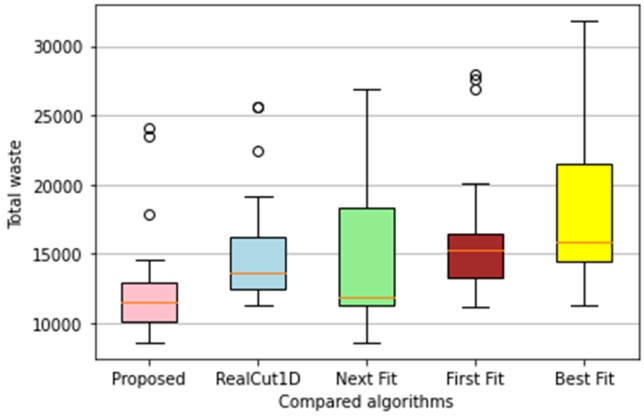

Fig. 10. Box plot of the results, relative to the TW performance measure.

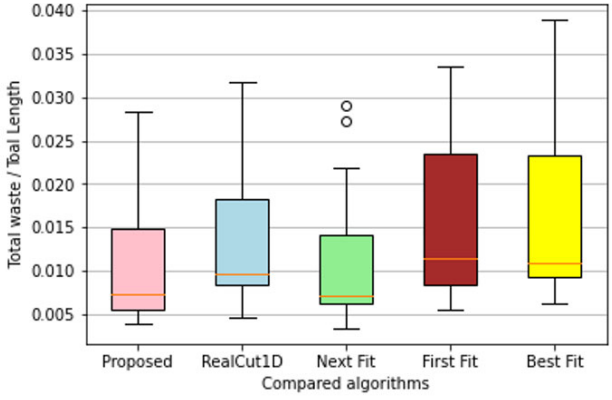

Fig. 11. Box plot of the results, relative to the RWL performance measure.

6. Conclusions and future works

The paper focused on the one-dimensional cutting stock problem and proposed an optimization procedure that minimizes the generated waste and that accounts for leftover bars (that can be returned to stock) as well as for other constraints related to the available inventory and to technical details needed to carry out the cut. It also allows users to assign different priorities to the bars and to define if, and how much, a long leftover is preferable to a small waste. Specifically, our algorithm can be classified as a hybrid Simulated Annealing (SA), because the cutting patterns are generated using different approaches, but most of the time they are created using a SA that exploits a specific and effective neighbor search. The algorithm, compared to three well-known heuristics and to a commercial software, showed favorable results in all tested scenarios. Noticeably, in fact, our algorithm outperforms the other ones in terms of the generated cutting waste.

Upcoming developments could address two shortcomings of the proposed heuristic. The first one is the need to maintain an association among the bars generated by the cutting patterns and the bars ordered by the end customers. In other words, if a cutting pattern generates a bar of length , provided that this length has been order by two or more customers, there is the need to determine to which customer the bar should be given. Although the problem might seem a simple instance of elementary labelling, it could become complex if additional operational constraints are considered, such as the limited spaces available to place the cut bars and/or to the timing with which shipments must be made. If so, in the generation of the cutting patterns one should also consider the associated customer orders and the physical bundling (and the constraints associated with it) for their shipment. A further development could be to expand the concept of prioritization from being subjectively determined according to various business operating policies (dictated by the specific use case), to being determined according to expected cut-off time and/or inventory costs.

Competing Interests Statement

There is no conflict of interest nor other financial or non-financial interests to declare. The paper was funded neither by public nor private companies. None of the authors of the paper is affiliated or involved in any organization or entity with financial or non-financial interest in the subject matter discussed in this manuscript.

Data Availability Statements

The code of the algorithm, as well as all data used to perform the numerical assessment will be shared on request.

Funding

This work was also partially funded under the National Recovery and Resilience Plan (NRRP), Mission 04 Component 2 Investment 1.5 –NextGenerationEU, Call for tender n. 3277 dated 30/12/2021. Award Number: 0001052 dated 23/06/2022.

References

Abuhassan, I.A., & Nasereddin, H.H. (2011) Cutting stock problem: solution behaviors. *International Journal of Research and Reviews in Applied Sciences 6*(4).

Alvim, A. C., Ribeiro, C. C., Glover, F., & Aloise, D. J. (2004). A hybrid improvement heuristic for the one-dimensional bin packing problem. *Journal of Heuristics*, *10*, 205-229.

Balogh, J., Békési, J., Dósa, G., Sgall, J., & Stee, R. V. (2014, December). The optimal absolute ratio for online bin packing. In *Proceedings of the twenty-sixth annual ACM-SIAM symposium on discrete algorithms* (pp. 1425-1438). Society for Industrial and Applied Mathematics.

Benjaoran, V., & Bhokha, S. (2014). Three-step solutions for cutting stock problem of construction steel bars. *KSCE Journal of Civil Engineering*, *18*, 1239-1247.

Bertolini, M., Braglia, M., Marrazzini, L., & Neroni, M. (2022). Project Time Deployment: a new lean tool for losses analysis in Engineer-to-Order production environments. *International Journal of Production Research*, *60*(10), 3129-3146.

Bertolini, M., Mezzogori, D., & Zammori, F. (2019). Comparison of new metaheuristics, for the solution of an integrated jobs-maintenance scheduling problem. *Expert Systems with Applications*, *122*, 118-136.

Bhatia, A. K., Hazra, M., & Basu, S. K. (2009, March). Better-fit heuristic for one-dimensional bin-packing problem. In *2009 IEEE International Advance Computing Conference* (pp. 193-196). IEEE.

Cheng, C. H., Feiring, B. R., & Cheng, T. C. E. (1994). The cutting stock problem—a survey. *International Journal of Production Economics*, *36*(3), 291-305.

Cherri, A. C., Arenales, M. N., & Yanasse, H. H. (2009). The one-dimensional cutting stock problem with usable leftover–A heuristic approach. *European Journal of Operational Research*, *196*(3), 897-908.

Cintra, G. F., Miyazawa, F. K., Wakabayashi, Y., & Xavier, E. C. (2008). Algorithms for two-dimensional cutting stock and strip packing problems using dynamic programming and column generation. *European Journal of Operational Research*, *191*(1), 61-85.

Crainic, T. G., Perboli, G., Pezzuto, M., & Tadei, R. (2007). Computing the asymptotic worst-case of bin packing lower bounds. *European journal of operational research*, *183*(3), 1295-1303.

Cui, Y., & Yang, Y. (2010). A heuristic for the one-dimensional cutting stock problem with usable leftover. *European Journal of Operational Research*, *204*(2), 245-250.

Delorme, M., Iori, M., & Martello, S. (2016). Bin packing and cutting stock problems: Mathematical models and exact algorithms. *European Journal of Operational Research*, *255*(1), 1-20.

Erhard, M. (2021). Flexible staffing of physicians with column generation. *Flexible Services and Manufacturing Journal*, *33*(1), 212-252.

Fleszar, K., & Charalambous, C. (2011). Average-weight-controlled bin-oriented heuristics for the one-dimensional bin-packing problem. *European Journal of Operational Research*, *210*(2), 176-184.

Garraffa, M., Salassa, F., Vancroonenburg, W., Vanden Berghe, G., & Wauters, T. (2016). The one-dimensional cutting stock problem with sequence-dependent cut losses. *International Transactions in Operational Research*, *23*(1-2), 5-24.

Gilmore, P. C., & Gomory, R. E. (1961). A linear programming approach to the cutting-stock problem. *Operations research*,

9(6), 849-859.
Gilmore, P. C., & Gomory, R. E. (1963). A linear programming approach to the cutting stock problem—Part II. *Operations research*, *11*(6), 863-888.
Haessler, R. W. (1975). Controlling cutting pattern changes in one-dimensional trim problems. *Operations Research*, *23*(3), 483-493.
Haessler, R. W., & Sweeney, P. E. (1991). Cutting stock problems and solution procedures. *European Journal of Operational Research*, *54*(2), 141-150.
Jahromi, M. H., Tavakkoli-Moghaddam, R., Makui, A., & Shamsi, A. (2012). Solving an one-dimensional cutting stock problem by simulated annealing and tabu search. *Journal of Industrial Engineering International*, *8*, 1-8.
Javanshir, H., Rezaei, S., Najar, S. S., & Ganji, S. S. (2010). Two dimensional cutting stock management in fabric industries and optimizing the large object's length. *Journal of Research and Reviews in Applied Sciences*, *4*(3), 243-249.
Kämpke, T. (1988). Simulated annealing: Use of a new tool in bin packing. *Annals of Operations Research*, *16*, 327-332.
Kantorovich, L. V. (1960). Mathematical methods of organizing and planning production. *Management science*, *6*(4), 366-422.
Kirkpatrick, S., Gelatt, C.D., & Vecchi, M.P. (1983). Optimization by simulated annealing. *Science, 220*, 671-680.
Kokten, E. S., & Sel, Ç. (2022). A cutting stock problem in the wood products industry: a two-stage solution approach. *International Transactions in Operational Research*, *29*(2), 879-907.
Levine, J., & Ducatelle, F. (2004). Ant colony optimization and local search for bin packing and cutting stock problems. *Journal of the Operational Research society*, *55*(7), 705-716.
Liang, K. H., Yao, X., Newton, C., & Hoffman, D. (2002). A new evolutionary approach to cutting stock problems with and without contiguity. *Computers & Operations Research*, *29*(12), 1641-1659.
Loh, K. H., Golden, B., & Wasil, E. (2008). Solving the one-dimensional bin packing problem with a weight annealing heuristic. *Computers & Operations Research*, *35*(7), 2283-2291.
Lübbecke, M. E., & Desrosiers, J. (2005). Selected topics in column generation. *Operations research*, *53*(6), 1007-1023.
Martello, S., & Toth, P. (1990a). *Knapsack problems: algorithms and computer implementations*. John Wiley & Sons, Inc.
Martello, S., & Toth, P. (1990b). Lower bounds and reduction procedures for the bin packing problem. *Discrete applied mathematics*, *28*(1), 59-70.
Osogami, T., & Okano, H. (2003). Local search algorithms for the bin packing problem and their relationships to various construction heuristics. *Journal of Heuristics*, *9*(1), 29-49.
Quiroz-Castellanos, M., Cruz-Reyes, L., Torres-Jimenez, J., Gómez, C., Huacuja, H. J. F., & Alvim, A. C. (2015). A grouping genetic algorithm with controlled gene transmission for the bin packing problem. *Computers & Operations Research*, *55*, 52-64.
Roodman, G. M. (1986). Near-optimal solutions to one-dimensional cutting stock problems. *Computers & operations research*, *13*(6), 713-719.
Sim, K., & Hart, E. (2013, July). Generating single and multiple cooperative heuristics for the one dimensional bin packing problem using a single node genetic programming island model. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation* (pp. 1549-1556).
Sweeney, P. E., & Paternoster, E. R. (1992). Cutting and packing problems: a categorized, application-orientated research bibliography. *Journal of the Operational Research Society*, *43*(7), 691-706.
Ülker, Ö., Korkmaz, E. E., & Özcan, E. (2008, September). A grouping genetic algorithm using linear linkage encoding for bin packing. In *International Conference on Parallel Problem Solving from Nature* (pp. 1140-1149). Berlin, Heidelberg: Springer Berlin Heidelberg.
Vahrenkamp, R. (1996). Random search in the one-dimensional cutting stock problem. *European Journal of Operational Research*, *95*(1), 191-200.
Wilhelm, W. E. (1999). A column-generation approach for the assembly system design problem with tool changes. *International Journal of Flexible Manufacturing Systems*, *11*, 177-205.
Wilhelm, W. E. (2001). A technical review of column generation in integer programming. *Optimization and Engineering*, *2*, 159-200.
Zammori, F., Neroni, M., & Mezzogori, D. (2021). Cycle time calculation of shuttle-lift-crane automated storage and retrieval system. *IISE Transactions*, *54*(1), 40-59.

Appendix A.

This section summarizes the notation used in the paper. Symbols are grouped by subject area and in order of appearance in the main text.

Table A1
Variables related to the bars.

Symbol	Meaning
L	The length of a bar available in stock.
l	The length of a bar ordered by a customer (i.e., a sub-bar cut from a bar of length L).
m	The material of a bar.
a	The cross-sectional area of a bar.
j	The subscript used to denote a bar type available in stock; the bar type is uniquely defined by the triplet (L_j, m_j, a_j).
i	The subscript used to denote the type of an ordered bar; the bar type is uniquely defined by the triplet (l_i, m_i, a_i).
B	A mapping that codifies all the bars available in stock.
N	The number of different types of bars available in stock.
n_j	The number of j-type bars available in stock.
p_j	The priority of j-type bars.
v_j	A Boolean variable equal to 1 if j-type bars are virgin, and 0 otherwise.
i	The letter i is used to denote a generic bar ordered by a customer.
R	A mapping that codifies all the bars ordered by the end customers.
M	The number of different types of bars ordered by the end customers.
n_i	The number of i-type bars ordered by the end customers.

Table A2
Variables related to the cutting patterns.

Symbol	Meaning
c_j	A cutting pattern made on a bar of type j.
x_{ij}	The number of cuts of length l_i obtained from a cutting pattern c_j. For convenience, when a specific bar type j is considered, the subscript j is dropped from the notation.
C	A finite set of cutting patters that are sufficient to solve an instance of the stock cutting problem.
$\mathcal{C}$	The sub-set of C that solves an instance of the stock cutting problem with the minimum waste.
c_j^*	The best cutting patterns that can be made on a j-type bar.
C^*	A N-dimensional vector (or list) containing all the best cutting patterns c_j^* that can be obtained on each j-type bar (with $j = 1, \dots, N$).
r	The number of repetitions of a certain cutting pattern c.
$OF(c_j)$	the value of the objective function of the cutting pattern c_j.

Table A3
Variables related to the cutting process.

Symbol	Meaning
ε	The thickness of the cutting blade.
h	The width of a head cut
g	The length of the gripping area.

w	The residual material (or sub-bar) remaining after all the cuts of a cutting pattern have been made
$\widetilde{w}$	The minimum length of a leftover. A sub bar of length $w \geq \widetilde{w}$ can be returned to stock for later use. Conversely, a bar shorter than $\widetilde{w}$ cannot be reused and must be disposed off.

Table A4
Variables related to the optimization algorithm.

Symbol	Meaning
α	A factor used to weight the importance of leftover in the objective function.
z	A Boolean variable equal to one for a perfect cutting pattern and zero otherwise.
λ	The length of a sub bar.
D	The dynamic programming matrix (containing the optimal solution for different lengths λ).
I_{max}	A cap on the number of iterations that can be performed to find an optimal or quasi optimal solution.
K	The number of epochs of the simulated annealing.
T_k	The value of the temperature at epoch
ΔT	The temperature drop, at each epoch k.
P	The acceptance probability.
β	The Boltzmann's constant of the Simulated Anneling.
ΔE_k	The variation in energy level at epoch k, computed as the percentage worsening of the objective function.
c	The current solution (cutting pattern), from which new solutions are generated.
c^*	The best solution found so far.
c_{new}	The best neighbor solution generated from c.
$c^+{}_{k,i}$	A cutting pattern generated at epoch k, by adding a cut of length l_i from c.
$c^-{}_{k,i}$	Like before, but a cut of length l_i is removed from c.
$q \leq Q$	The number of cuts to be removed from c^* in the deep neighbour search procedure.
c^*_-	The partial cutting patterns obtained by removing the q longest cuts from c^*.
$U(c_j^*)$	A utility function used to sort the optimal cutting patterns obtained on bars of different type j.
s_m, s_M	The minimum and maximum utility that a leftover can get.

Appendix B.

Let us consider a customer's order $R = \{500{:}\,2,\ 300{:}\,4\}$, a leftover of length $L = 1{,}200$ mm, a cutting blade $\varepsilon = 20$ mm and a gripping area $g = 40$ mm. At first, when only cuts of length $l_1 = 500$ mm can be made, a minimum length $\lambda = 520$ mm is needed to make a cut. Hence, all states from $D[1,0]$ to $D_1[1,519]$ on Row #1 of matrix D (remember that Row #0 has only null values) contain the null cutting pattern $c^*_{i,\lambda} = c^*_{1,0} = \{500 : 0\}$, and only at state $D[1,520]$ the first non-null cutting pattern $c^*_{1,520} = \{500 : 1\}$ can be made. Note that this solution generates a useful length of 500 mm with a cutting scrap of 20 mm and so, according to eq. (2.1), its objective function is 500α. A second and last cut can be made at state $D[1,1040]$, where the optimal solution becomes $c^*_{1,1040} = \{500 : 2\}$ with an objective function of $1{,}000\alpha$. Since no other cut of 500 mm can be added, this solution is shared by all successive states on Row #1 of D. Note, however, that the objective function of each state may change, as the length λ of the sub-bars increases from 1,040 to 1,200 mm. To clarify this concept, let us consider the generic state $D[1,1040+w]$. Relatively to state $D[1,1040]$ a residual bar of length w is cut. If $w \leq \widetilde{w}$ this residual bar is waste, otherwise it is a leftover; in the latter case, according to eq. (2.1), the objective function becomes $1{,}000\alpha + (1-\alpha)\cdot w$.
For the sake of clarity, Row #1 of matrix D is shown in Table B1.

Table B1
Row #1 of matrix D.

$j = 0$	$j = 1$	…	$j = 520$	$j = 521$	…	$j = 1{,}040$	$j = 1{,}041$	…	$j = 1{,}200$
500:0	500:0	…	500:1	500:1	…	500:2	500:2	…	500:2
300:0	300:0	…	300:0	300:0	…	300:0	300:0	…	300:0

OF = 0	OF = 0	…	OF = 500α	OF = 500α	…	OF = 1,000α	OF = 1,000α	…	OF = 1,000α + 360(1−α)

When the third row of D is generated, also cuts of length $l_2 = 300$ mm are admitted. Hence, a first cut can be made at length $\lambda = 320$ mm. This solution, saved at state, $D[2, 320]$ is $c^*_{2,320} = \{500 : 0,\ 300 : 1\}$ and has an objective function of 300α. Also note that this solution is copied in all subsequent states till $D[2, 519]$. Conversely, at state $D[2, 520]$ the solutions stored at $D[1, 520]$ is preferred, because it has an objective function equal to 500α, higher than 300α. Following the same reasoning, Row #2 is generated as shown in Table B2.

Table B2
Row #2 of matrix D.

$j = 0$	…	$j = 320$	…	$j = 520$	…	$j = 640$	…	$j = 840$
500:0	…	500:0	…	500:1	…	500:0	…	500:1
300:0	…	300:1	…	300:0	…	300:2	…	300:1
OF = 0	…	OF = 300α	…	OF = 500α	…	OF = 600α	…	OF = 800α
…	**…**	$j = 960$	**…**	$j = 1,040$	**…**	$j = 1,180$	**…**	$j = 1,200$
…	…	500:0	…	500:2	…	500:1	…	500:1
…	…	300:3	…	300:0	…	300:2	…	300:2
…	…	OF = 900α	…	OF = 1,000α	…	OF = 1,100α	…	OF = 1,100α

The cutting pattern $c^*_{2,1200} = \{500 : 1,\ 300 : 2\}$ found at state $D[2, 1200]$ is indeed optimal. In fact, it uses three cuts to generate a useful length of $(300 + 300 + 500) = 1{,}100$ mm. Therefore, the total waste is $(3 \times \varepsilon) + g = 60 + 40 = 100$ mm and, as there is no waste in addition to the inevitable one, $c^*_{2,1200}$ is an almost perfect cutting pattern.