Noname manuscript No.
(will be inserted by the editor)

Heuristic Methods for Minimizing Cut Bars and Using Leftovers from the One-dimensional Cutting Process

Glaucia Maria Bressan · Matheus H. Pimenta-Zanon · Fabio Sakuray

Received: date / Accepted: date

Abstract The cutting problems consist in cutting a set of objects available in stock in order to produce the desired items in specified quantities and sizes. The one-dimensional cutting stock problem involves only one of the relevant dimensions in the cutting process, as in cutting bars, rolls and tubes. The cutting process can generate leftover (which can be reused in a new demand) or losses (which are discarded). This paper presents two heuristic methods for minimizing the number of cut bars in the one-dimensional cutting process, satisfying the items demand in an unlimited bars quantity of just one type. The results of simulations are compared with methods from literature and with the limiting values for this considered type of problem. The results show that proposed heuristics reduce processing time and the number of bars needed in cutting process, while it provides a greater leftover (by grouping losses) for the one-dimensional cutting stock problem. The heuristics contribute to reduction of raw materials or manufacturing costs in industrial processes, such as the automotive industry, construction, bicycle manufacturing and other purposes.

Keywords Heuristic Procedures · Usable Leftovers · Cutting Problems · Optimization

Mathematics Subject Classification (2020) MSC 90-08 · MSC 90C59

G.M. Bressan
Universidade Tecnológica Federal do Paraná (UTFPR). Mathematics Department
Alberto Carazzai, 1640, 86300-000, Cornélio Procópio, PR, Brazil
E-mail: glauciabressan@utfpr.edu.br

M.H. Pimenta-Zanon
Universidade Tecnológica Federal do Paraná (UTFPR). Computer Science Department
Alberto Carazzai, 1640, 86300-000, Cornélio Procópio, PR, Brazil
E-mail: matheus.pimenta@outlook.com

F. Sakuray
State University of Londrina (UEL). Computer Science Department
Rodovia Celso Garcia Cid, Pr 445 Km 380 C.P. 10.011, 86057-970, Londrina, PR, Brazil
E-mail: sakuray@uel.br

1 Introduction

The optimization of cutting stock problems aim to determine a set of items to be produced, in order to reduce the number of raw material in cutting process for manufacturing items. This problem belongs to an important area of Operational Research to assist in decision making process. For this reason, the study of cutting stock problems has aroused the interest of researchers [1, 3, 20, 12, 4].

There are different methods to mathematically model the cutting stock problems. One of the possible models is the Linear Programming, which proposes to solve linear optimization problems, describing real problems in order to maximize or minimize a given objective function, subject to a set of constraints, whose variables are linearly related [22]. By using Linear Programming models, the cutting stock problem consists of cutting a set of objects available in stock produce the desired items in specified quantities and sizes, in order to optimize an objective function.

The one-dimensional cutting stock problem involves only one of the relevant dimensions in the cutting process, as it occurs with cutting steel bars, paper rolls and tubes [8, 9].

Historically, Gilmore and Gomory [13], one of the first to address the cutting stock problem, defines the Cutting Stock Problem as filling an order at minimum cost for specified numbers of lengths of material to be cut from given stock lengths of given cost. Authors described a technique for overcoming the difficulty in the linear programming formulation of the problem, which involves a large number of variables. Later, these authors extended and adapted the methods for cutting stock outlined to the specific full-scale paper trim problem [14]. And, in [15], higher dimensional cutting stock problems are discussed as linear programming problems.

Latter on, authors in [16] developed a sequential heuristic to roll cutting optimization for clothing industry. The issue of roll cutting is defined as a bi-criterial multidimensional knapsack problem. After that, these authors applied the sequential heuristic for optimizing one-dimensional cutting stock when all stock lengths are different [17]. A combined method, which uses a sequential heuristic procedure and the branch-and-bound, is proposed in [18] for the solution to the general one-dimensional cutting stock problem.

One branch of cutting stock problem is called *the usable leftover cutting stock problems* [8], whose purpose is to determine the cutting patterns and to analyze the leftovers generated by the cutting process. This problem consists of one-dimensional cutting stock problem in which the non-used material in the cutting process may be used in the future, if large enough [7].

A particular one-dimensional cutting stock problem with both cutting and reuse decision variables was presented in [3]. It was formulated as an integer linear programming and it was efficiently solved by applying standard packages within a column generation technique.

In [7], classical heuristic methods are modified, so that cutting patterns with undesirable leftover, which are not large enough to be used and nor too small to be acceptable waste, are redesigned. A review of published studies that consider the solution of the one-dimensional cutting stock problem with the possibility of using leftovers to meet future demands, is presented by [9].

Cui and Yang [11] propose a heuristic algorithm for the one-dimensional cutting stock problem with usable leftover, consisting of two procedures: a linear programming that fulfills the major portion of the item demand and a sequential heuristic that fulfills the remaining portion of the item demand. Besides, in [8], it is assumed that leftovers should not remain in stock for a long time. Then, leftovers have priority-in-use compared to standard objects (objects bought by the industry) in stock and a heuristic procedure is proposed for this problem.

The paper [12] presents a review of the most important mathematical models and algorithms developed for the exact solution of the one-dimensional bin packing and cutting stock problems. In addition, the paper experimentally evaluates the performance of the main available software tools. Despite addressing the cutting stock problem, the authors do not address the use of leftovers.

Recently, Cui et al. [10] present an integer programming model for the one-dimensional cutting stock problem with limited leftover types and describes a heuristic algorithm based on a column-generation procedure to solve it. Authors introduced a heuristic procedure in [5], called the Residual Recombination Heuristic. The column generation technique is associated with a set of residual cutting patterns, recombining these residual cutting patterns in different ways, generating new integer feasible cutting patterns. In [6], the only objective to be achieved is to minimize the quantity of cut objects. Authors propose a modification in the Constructive Greedy Heuristic, building a cutting pattern by sorting in descending order the items of pair or odd length.

Finally, a genetic algorithm approach is presented in [2]. A genetic-based decision support system is applied to validate the solution feasibility of the problem and the objective is to minimize the leftovers in pipes cutting.

In the context of cutting stock problems and usable leftovers, the main goal of this paper is to propose two heuristic methods, in order to minimize the number of bars needed to cut known items demand, in the one-dimensional cutting process with no stock. The problem presented in this paper considers an unlimited quantity of one type of bar.

The proposed methods are measured by accuracy and runtime. The two proposed heuristics methods are described in the next section, whose performances are analyzed and compared with methods from literature. The accuracy is rated by: (i) total number of used bars, (ii) the loss and leftover produced in process. The runtime shows that the algorithms present better computational complexity. In this paper, loss and leftover are classified by size of bar not used, i.e, a bar smaller than a smallest item demanded is considered loss, otherwise leftover. The proposed heuristics aim to reduce losses, by grouping multiple losses in the same bar (whenever possible), so that the losses (unusable) become leftovers (usable).

The main contribution is to obtain optimized solutions, which are competitive with those found in the literature for the one-dimensional cutting stock problem, with relevant gains in computational complexity (or runtime) and convert loss to leftover whenever possible, so they can be reused. The heuristics can be used in commercial software solutions, which provide reduction of the raw materials used (or manufacturing costs) and environment preservation.

The remainder of the paper is organized as follows. Section 2 presents the definition of the problem considered in this paper and a numerical example. In Section 3, a numerical example defines the minimum number of bars needed in cutting process. Section 4 initially describes the Selection of Ordered Items (SOI) procedure, used in the first step of both proposed heuristics. After that, the innovations of this paper are presented: the heuristics methods *OptmizationDistBSP* and *OptimizationTREE*. The data set for simulation are provided in Section 5, followed by analysis of simulation's results (Section 6). Finally, in Section 7, the conclusion of the paper and the discussion about some aspects of future work are addressed.

2 Problem Definition

The proposed heuristic methods aim to present solutions for the general problem defined as follows.

Problem: to minimize the number of bars needed to cut known items demand, in the one-dimensional cutting process, such as the demand of items has to be met, by cutting bars of the same size. The demand of items has to be met:

$$\sum_{p=1}^{P} x_{ip} bar_p \geq d_i, \forall i \in 1, ..., n, bar_p \geq 0 \text{ and integers,} \tag{1}$$

where the variables and parameters are described below.

- bar_p: number of cut bars in the cutting patterns $p = 1, ..., P$ (variable)
- d_i: demand of item i
- x_{ip}: the number of items $i = 1, ..., n$ cut in the cutting pattern p. In the one-dimensional case, any cutting pattern must satisfy [7]

  $$\sum_{i=1}^{n} l_i x_{ip} \leq L, \forall p \in 1, ..., P; 0 \leq x_{ip} \leq d_i, \tag{2}$$

  in which L is the size of the bar, l_i is the length of item i and this expression guarantees that the number of produced items is not bigger than the size of the bar.

The cutting process can generate leftover (which can be reused in a new demand) or losses (which are discarded). The goal is to reduce losses, by grouping them in the same bar, when possible. Thus, the losses become leftovers.

The following numerical example illustrates a cutting process with a known items demands and how the losses and leftovers are identified. In the example illustrated in Figure 1, a set of different sizes and quantities of items must be produced (2 items of 60cm, 3 items of 55cm and 5 items of 30cm) from one fixed-size bar (180cm).

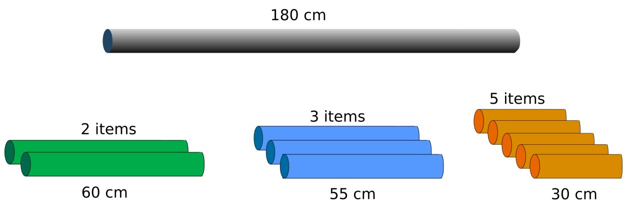


Fig. 1: Available bar and demands.

In order to meet the demand, cutting patterns identified by CP1, CP2 and CP3, represented in Figures 2, 3 e 4, respectively, can be used. However, the selected cutting pattern may not use the whole bar. The numerical values of the possible cutting patterns are shown in Table 1.

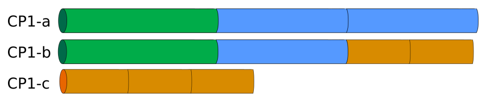


Fig. 2: Cutting Pattern 1.

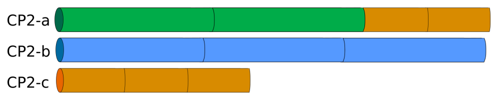


Fig. 3: Cutting Pattern 2.

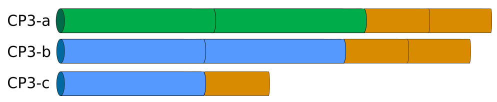


Fig. 4: Cutting Pattern 3.

The three possible cutting patterns are using 3 bars to produce the demanded items. The total amount of unused material is 105cm. The main difference between the cutting patterns is the distribution of the unused material

Table 1: Possible cutting patterns.

CP	**Bar**	**Not used**	**Classification**
1	a	10 cm	loss
	b	5 cm	loss
	c	90 cm	leftover
2	a	0 cm	fully used
	b	15 cm	loss
	c	90 cm	leftover
3	a	0 cm	fully used
	b	10 cm	loss
	c	95 cm	leftover

in bars. CP1 distributes the unused material on the three bars, while CP2 and CP3 concentrate it on two bars. Then, unused material is classified as loss (H_{loss}) or leftover (H_{left}), according to its size, the H represents the heuristic to be used. Considering as loss the unused smaller bar than the smallest demanded item, in this case, the loss will be discarded because it can not be used to produce a new item. Otherwise, the unused material is considered as leftover and can be used for future demand. Equation 3 presents the not used part of the bar by heuristic H .

$$Unused\ bar = L - \sum_{1}^{n} l_i x_i \tag{3}$$

In this equation, l_i and x_i are the size and the number of items i used in the cutting pattern, respectively. The classification unused bar:

– Loss (H_{loss}): if unused bar is smaller than the smallest demanded item;
– Leftover(H_{left}): if unused bar is greater or equal than the smallest demanded item.

Table 1 shows the cutting patterns, in which all CPs have the same size of unused bars. Thus, given a set of demanded items, the cutting patterns should use the smallest number of bars possible, minimizing losses and grouping them in order to make them leftovers.

3 Minimum Number of Bars

The hypothetical cutting pattern illustrated in Figure 5 presents a distribution of items into the bars, in which the minimum number of bars required to produce the demanded items of the numerical example above, is determined. This is a hypothetical solution since all items must be produced in a continuous piece. Note that an item of 30cm was divided in two parts (the red parts of bars "b" and "c"), resulting in the use of full bars "a" and "b".

The minimum number of bars needed to produce the demanded items is defined in Equation 4. The *lower bound of bars* (B_{bar}) is used to evaluate the performance of proposed heuristics, i.e., better solutions use bars close to B_{bar}.

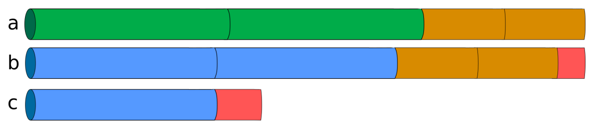


Fig. 5: Hypothetical Cutting Pattern.

If any of the heuristics provide as solution a minimum number of cutting bars smaller than B_{bar}, it means that the algorithm has an error and the solution is unreal.

$$B_{bar} = \left\lceil \frac{\sum_{i=1}^{n} l_i d_i}{L} \right\rceil \tag{4}$$

Considering the items of the numerical example, B_{bar} can be obtained as:

$$B_{bar} = \left\lceil \frac{(60 \times 2) + (55 \times 3) + (30 \times 5)}{180} \right\rceil = \lceil 2.416 \rceil = 3. \tag{5}$$

Therefore, the minimum number of bars is 3, the same number of bars used in cutting patterns presented in Table 1. When this theoretical limit is achieved, the unused material remaining is defined in Equation 6.

$$Minimum\ Theoretical\ limit\ of\ Unused\ Material = \left(B_{bar} - \frac{\sum_{i=1}^{n} l_i d_i}{L} \right) L \tag{6}$$

The minimum theoretical limit of unused material (Equation 6) can be classified as follows:

- Loss (B_{loss}): if unused material is smaller than the smallest demanded item;
- Leftover(B_{left}): if unused material is greater or equal than the smallest demanded item.

Considering the numerical example, when the lower bound of bars is reached, the size of unused material is classified as leftover:

$$(3 - 2.416)\ 180 = 105cm \Rightarrow B_{left} \tag{7}$$

This result indicates that:

- The proposed cutting patterns (CP1, CP2 and CP3) are using the minimum number of bars;
- The cutting pattern CP3 with continuous leftover nearest to B_{left} is the best solution, comparing to cutting patterns CP1 and CP2.

4 The Proposed Heuristics

In this section, two proposed heuristics methods to solve the problem described in section 2 are presented. These heuristics are evaluated by total number of bars used in whole cutting process, the loss and leftover provided and also computational complexity of solution method.

4.1 Selection of Ordered Items (SOI)

First of all, a Selection of Ordered Items (SOI) procedure, described in Procedure 1, must be presented, since this ordering procedure, which is based on the First fit algorithm [19], is used in the first step of both proposed heuristics.

This procedure uses as input the set of items demanded ($\mathbf{l}$ and $\mathbf{d}$), sorted in descending order of size. Then, the items are selected to be cut, starting with the biggest one; if not possible, the next item is tested and so on. Other inputs are: the size of the bar (L) and the total number of items (n). The outputs are: the number of each items i in the cutting pattern or bar ($\mathbf{x}$), the size of bar not used ($\hat{L}$) and the updated demand of items ($\mathbf{d}$).

Procedure 1: Selection of Ordered Items(SOI)

```
input  : $\mathbf{l}$, $\mathbf{d}$, $L$, $n$
output: $\mathbf{x}$, $\hat{L}$, $\mathbf{d}$
1 $\hat{L} = L$
2 for $i = 1$ to $n$ do
3     $x_i = \lfloor \frac{\hat{L}}{l_i} \rfloor$
4     if $x_i > d_i$ then
5         $x_i = d_i$
6     end
7     $\hat{L} = \hat{L} - (x_i * l_i)$
8     $d_i = d_i - x_i$
9 end
```

4.2 OptimizationDistBSP Heuristic

The proposed OptimizationDistBSP heuristic is described in Algorithm 1, in which the set of demanded items is sorted in descending order of size ($\mathbf{l}$). Then, the Selection of Ordered Items (SOI) procedure is used to generate the first cutting pattern (1^{st} phase). After this first selection, if the bar is not fully used, an optimization process is started (2^{nd} phase). In the second phase, the Algorithm 1 try to change one item selected to be cut (in 1^{st} phase) for another two smaller items, using ones that generate a smaller leftover bar. This process is repeated to all items selected in the 1st phase. After 2nd phase, the bar not used ($\hat{L}$) is classified as leftover or loss.

The inputs of Algorithm 1 are:

– the items size $\mathbf{l} = \{l_1, ..., l_n\}$;
– the items demand $\mathbf{d} = \{d_1, ..., d_n\}$;
– the size of bar L;
– the number of items to be cut n.

Algorithm 1: OptimizationDistBSP Heuristic

```
input  : l, d, $L$, $n$
output: $leftover$, $loss$, $bar$
1  bar=1
2  Sort items in descending order of size (l)
3  smallest-item=$l_n$                                  // smallest item size
4  while $\sum_{i=1}^{n} d_i > 0$ do
5      x,$\hat{L}$,d = SOI(l,d,L,n)        // starting $1^{st}$ phase -- call Procedure 1)
6      if $\hat{L} > 0$ then // (starting $2^{nd}$ phase)
7          for $i = 1$ to $n$ do
8              if $x_i > 0$ then
9                  $\bar{L} = \hat{L} + l_i$   // if item $i$ is selected to be cut, then remove
                   it
10                 for $j = i + 1$ to $n$ do
11                     $selec = l_j$                // try to insert items $j$ and $k$
12                     for $k = n - 1$ to $j + 1$ do
13                         if $d_k > x_k$ then
14                             if $\bar{L} - (selec + l_k) \geq 0$ and $\bar{L} - (selec + l_k) < \hat{L}$ then
                                       // items $j$ and $k$ provide a better bar
                                   utilization
15                                 $\hat{L} = \hat{L} - (selec + l_k)$
16                             end
17                         end
18                     end
19                 end
20             end
21         end
22         if $\hat{L} <$ $smallest\text{-}item$ then // classifying the not used segment
23             loss = loss + $\hat{L}$                                  // loss
24         end
25         else
26             leftover = leftover + $\hat{L}$                          // leftover
27         end
                   /* update the number of items remain to be cut ($d_i$) */
28     end
29     bar=bar+1
30 end
```

The outputs of Algorithm 1 are:

- the $leftover$ of all cuting process;
- the $loss$, the sum of loss in each bar used;
- the bar with sum of used bars.

The variables used in the Algorithm 1 are described as follows:

- $\hat{L}$: size of bar not used during the execution of the algorithm
- x_i: the number of items i to be cut in the bar, for $i = 1, ..., n$

4.3 Tree-based Heuristic (*OptimizationTREE*)

A Tree-based Heuristic (*OptimizationTREE*) is described in Algorithm 2, which has two phases: (i) selection of the items to be cut and (ii) loss reduction. First, the set of demanded items is sorted in descending order of size (1). In the 1st phase, a set of items is selected according to Procedure 1. This first set is then used in the 2nd phase to reduce losses as follows: for each selected item, reduce the number of items to be cut by one unit, trying to fill the rest of the bar with larger elements and then with smaller elements. After the next smaller item to be cut is selected, the algorithm verifies if a larger item can be used, if not, the next smaller item is processed. The inputs, outputs and main variables of Algorithm 2 are the same of Algorithm 1.

In this proposed heuristic, the losses of the cutting process are concentrated on the smallest number of bars possible, using a tree structure [21], in order to convert losses (unusable) into leftovers (usable) and reduce the number of bars needed in process.

5 Data Set for Simulations

The numerical values used in simulations of proposed heuristics are based on [6], with the aim of minimizing the quantity of cut objects, considering the one-dimensional cutting stock problem. The database contains 18 categories of problems, each one with 100 instances. The categories are divided according to the quantity of items m, the average of the items demands $\bar{d}$, and different combinations of values v_1 and v_2 for determining the length of the items that are generated in the interval $[v_1 L, v_2 L]$, with L the size of the bar, as shown on Table 2 [6].

Python language is used for all the heuristics implementation. In *OptimizationDistBSP* and *OptimizationTREE* heuristics, the sorting method used is Quick Sort. Indeed, in order to maintain the same comparison environment, all the methods use Quick Sort. The hardware used for the simulations consists of an Intel processor ® *Inside*^{TM} Core i7 920 @2.67Ghz with 16Gb of RAM. For the heuristic simulations, the Linux operating system Ubuntu 18.04 x64 is used. The runtime of the presented instances is also considered in this paper, showing that the results are acceptable in practice.

Algorithm 2: Tree-based Heuristic (*OptimizationTREE*)

```
input  : l, d, L, n
output: bar, leftover, loss
bar=1
Sort items in descending order of width ($l$)
smallest-item=$l_n$                                         // smallest item size
while $\sum_{i=1}^{n} d_i > 0$ do
    x,$\hat{L}$,d = SOI(l,d,L,n)          // start $1^{st}$ phase -- call Procedure 1)
    $i = 1$
    while (($\hat{L} \neq 0$) and ($i \neq n$)) do // start $2^{nd}$ phase (new tree branch)
        $x_i = x_i - 1$                              // remove one unit of item $i$
        $\bar{L} = \hat{L} - (l_i * x_i)$
        for $k = i + 1$ to $n$ do
            $x_k = \lfloor \frac{\bar{L}}{l_k} \rfloor$                              // try to use item $k$
            $\bar{L} = \bar{L} - (x_k * l_k)$
            for $m = 1$ to $i - 1$ do // seeking for an item to complete the bar
                if $\bar{L} == l_m$ then
                    $x_m = 1$
                    $\bar{L} = 0$
                    Break
                end
            end
            m=i+1
            while (($\bar{L} > 0$) and ($m \leq n$)) do // bar not completely used,
             seeking for an item to reduce the not used part of the bar
                if (($\bar{L} - l_m$) $\geq 0$) and ($\bar{L} - l_m < \hat{L}$) then
                    $\bar{L} = \bar{L} - l_m$
                    $x_m = 1$
                end
                m=m+1
            end
        end
        if $\bar{L} < \hat{L}$ then
            $\hat{L} = \bar{L}$
        end
        $i = i + 1$
    end
    if $\hat{L}$ < smallest-item then // classifying the not used segment
        loss = loss + $\hat{L}$                                            // loss
    end
    else
        leftover = leftover + $\hat{L}$                                  // leftover
    end
    /* update the number of items remain to be cut ($d_i$)                  */
    bar=bar+1
end
```

Table 2: Data for numeric simulation.

category	m	v_1	v_2	$\bar{d}$
1	10	0.01	0.2	10
2	10	0.01	0.2	100
3	20	0.01	0.2	10
4	20	0.01	0.2	100
5	40	0.01	0.2	10
6	40	0.01	0.2	100
7	10	0.01	0.8	10
8	10	0.01	0.8	100
9	20	0.01	0.8	10
10	20	0.01	0.8	100
11	40	0.01	0.8	10
12	40	0.01	0.8	100
13	10	0.2	0.8	10
14	10	0.2	0.8	100
15	20	0.2	0.8	10
16	20	0.2	0.8	100
17	40	0.2	0.8	10
18	40	0.2	0.8	100

6 Results and Discussion

The execution of the instances on each category provides results that allow to analyze in detail the performance of the proposed heuristics in comparison with methods from literature. Results were compared with First Fit Decreasing (FFD) and Greedy constructive heuristics, whose algorithms are found in [6]. As recommended in [6], the method used in this paper for solving the knapsack problem was that implicit enumeration suggested by [14], based on depth search first, enabling through the use of bounding that the worst solutions be discarded without losing the optimal solution. There are, in the literature, other techniques for solving the knapsack problem, such as in [24] and [23]. The constructive heuristics FFD and Greedy are the most famous and most studied methods for an approximative solution of the bin-packing problem. The heuristic proposed by [6], called "Modified Greedy Heuristic for the one-dimensional cutting stock problem" was not used for comparison, since the code is not available.

Tables 3, 4, 5 and 6 present the results of *OptimizationDistBSP*, *OptimizationTREE*, *FFD* and *Greedy* heuristics, respectively. The mean of variable *bar* ($\overline{bar}$), the mean of $leftover$ ($\overline{left}$) and the mean of *loss* ($\overline{loss}$) are presented by category, followed by its respective standard deviation: $\sigma(bar)$, $\sigma(left)$ and $\sigma(loss)$. The penultimate column of these tables shows the runtime for each category, in seconds; and the last column shows the standard deviation of the runtime.

Table 3: Results of *OptimizationDistBSP* heuristic

categories	**OptimizationDistBSP**							
	$\overline{bar}$	$\sigma(bar)$	$\overline{left}$	$\sigma(left)$	$\overline{loss}$	$\sigma(loss)$	$\overline{time}$	$\sigma(time)$
1	11.6	2.11	534.08	278.71	86.62	78.70	0.0002	2.9057
2	111.62	21.00	816.5	760.29	1071.51	906.46	0.0014	0.0002
3	22.19	3.14	541.34	281.60	70.35	52.84	0.0007	0.0001
4	217.47	31.83	1049.6	1338.75	1019.94	696.04	0.0043	0.0009
5	42.99	4.08	448.7	284.10	46.74	41.64	0.0018	0.0002
6	425.92	40.29	818.43	634.31	889.29	497.56	0.0136	0.0023
7	50.33	13.82	6602.73	6231.46	1254.22	2297.58	0.0005	7.7266
8	501.05	138.75	63755.77	63195.05	12704.98	23235.25	0.0052	0.0008
9	94.04	21.19	9652.77	9748.58	1017.12	955.38	0.0016	0.0001
10	936.46	211.81	93802.93	97924.63	10290.55	9717.24	0.0154	0.0020
11	177.78	24.97	12803.81	10130.19	976.28	887.37	0.0053	0.0008
12	1772.73	250.02	125661.81	102668.77	9702.56	8783.28	0.0506	0.0083
13	63.59	13.12	7133.78	7842.93	4950.66	4670.89	0.0006	9.1420
14	633.69	131.97	69056.7	79194.77	49761.5	46650.37	0.0060	0.0008
15	119.9	209.41	12483.4	12573.31	5614.1	3641.52	0.0018	0.0001
16	1195.05	25.07	121626.76	126491.44	56469.34	36465.79	0.0169	0.0018
17	225.29	25.07	16621.37	14857.54	7123.91	3807.03	0.0054	0.0003
18	2247.46	250.25	162619.9	149012.36	71656.02	37905.31	0.0513	0.0036

Table 4: Results of *OptimizationTREE* heuristic

categories	**OptimizationTREE**							
	$\overline{bar}$	$\sigma(bar)$	$\overline{left}$	$\sigma(left)$	$\overline{loss}$	$\sigma(loss)$	$\overline{time}$	$\sigma(time)$
1	11.64	2.08	599.37	292.01	52.8	53.08	0.0006	0.0001
2	112.18	21.28	1850.18	1664.15	581.81	620.20	0.0045	0.0010
3	22.25	3.14	582.7	280.40	51.83	37.70	0.0025	0.0005
4	218.12	31.77	2061.05	1606.07	580.93	410.84	0.0222	0.0045
5	43.31	4.15	721.69	405.89	48.84	29.37	0.0109	0.0019
6	427.99	40.78	3102.85	2545.51	573.65	340.45	0.1118	0.0224
7	53.13	13.63	9837.47	6182.57	803.64	2244.90	0.0016	0.0004
8	530.19	135.67	97567.42	61837.92	8018.69	22688.81	0.0161	0.0039
9	100.63	20.73	16628.51	9949.95	592.53	580.64	0.0082	0.0017
10	1002.41	207.30	164014.63	100256.90	6015.21	5902.85	0.0812	0.0161
11	195.03	26.15	30456.63	12559.39	534.05	508.14	0.0451	0.0074
12	1949.67	257.76	307174.28	121235.33	5081.1	4297.14	0.4579	0.0787
13	65.39	12.77	9598.81	8017.47	4285.63	4852.56	0.0020	0.0004
14	652.0	127.34	94291.14	79295.18	42837.06	48460.37	0.0206	0.0043
15	123.82	20.40	17488.31	12278.71	4526.54	3701.95	0.0110	0.0015
16	1234.39	204.82	171675.8	124876.38	45755.04	37166.98	0.1095	0.0152
17	236.96	25.63	30361.63	16500.12	5053.65	3918.16	0.0628	0.0072
18	2364.25	254.60	300548.74	165015.22	50509.13	38917.36	0.6339	0.0767

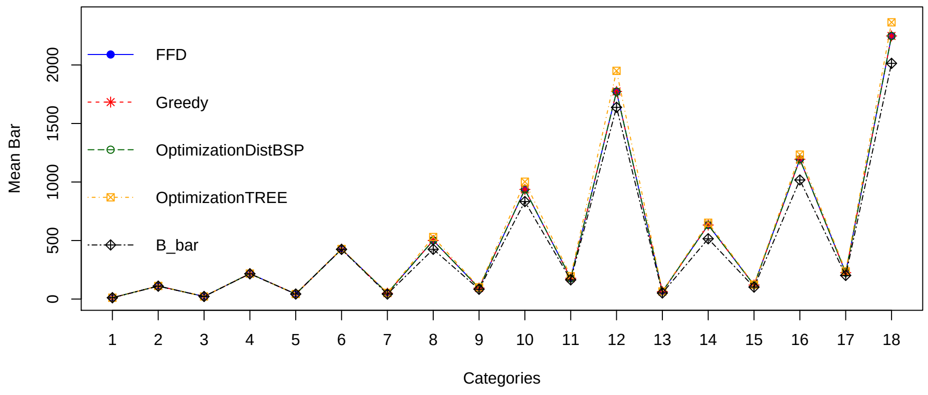
Table 5: Results of FFD heuristic

categories	FFD							
	$\overline{bar}$	$\sigma(bar)$	$\overline{left}$	$\sigma(left)$	$\overline{loss}$	$\sigma(loss)$	$\overline{time}$	$\sigma(time)$
1	11.61	2.11	518.18	276.53	114.94	90.35	0.0001	2.4100
2	111.68	21.04	759.54	745.23	1188.47	935.97	0.0012	0.0002
3	22.22	3.15	512.64	288.03	134.99	71.05	0.0005	6.3114
4	217.7	31.70	936.01	1253.91	1364.39	786.32	0.0043	0.0006
5	43.13	4.09	484.77	307.68	159.69	54.94	0.0018	0.0001
6	426.53	40.22	669.49	546.09	1648.8	579.07	0.0163	0.0018
7	50.35	13.79	6603.87	6229.74	1273.08	2295.53	0.0005	0.0001
8	501.15	138.62	63655.83	63266.46	12904.92	23212.73	0.0050	0.0012
9	94.06	21.13	9567.81	9805.52	1122.08	1056.37	0.0017	0.0002
10	936.3	211.49	92674.47	98630.13	11259.01	10576.71	0.0165	0.0029
11	177.56	25.02	12350.72	10435.04	1210.29	1099.85	0.0061	0.0006
12	1770.26	250.82	120758.59	105763.48	12136.68	10936.84	0.0588	0.0059
13	63.59	13.12	7131.53	7844.86	4952.91	4669.46	0.0007	0.0001
14	633.7	131.95	69061.7	79190.41	49766.5	46648.98	0.0067	0.0013
15	119.93	20.86	12478.18	12578.21	5649.32	3634.25	0.0022	0.0003
16	1195.29	209.07	121645.11	126473.80	56690.99	36368.65	0.0222	0.0036
17	225.3	25.07	16574.29	14900.02	7180.99	3804.80	0.0080	0.0007
18	2247.42	250.36	162240.85	149340.92	71995.07	37771.34	0.0782	0.0078

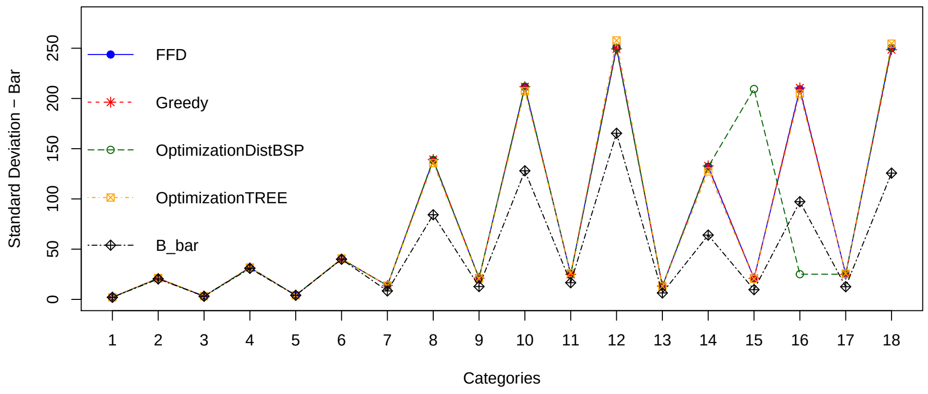
Table 6: Results of Greedy heuristic

categories	Greedy							
	$\overline{bar}$	$\sigma(bar)$	$\overline{left}$	$\sigma(left)$	$\overline{loss}$	$\sigma(loss)$	$\overline{time}$	$\sigma(time)$
1	11.57	2.11	523.9	292.03	69.22	45.64	0.0007	0.0017
2	111.16	20.85	702.97	717.66	725.04	518.02	0.0001	0.0071
3	22.17	3.13	524.36	270.67	73.27	39.46	0.0003	0.0004
4	216.92	31.48	753.15	711.54	767.25	431.48	0.0300	0.0051
5	43.03	4.09	474.27	292.79	70.19	30.52	0.0159	0.0022
6	425.55	40.10	600.32	381.34	737.97	350.76	0.1542	0.0204
7	50.3	13.86	6661.82	6226.74	1165.13	2281.98	0.0029	0.0008
8	500.67	139.27	64405.65	63156.71	11675.1	22967.09	0.0285	0.0083
9	94.04	21.20	9731.04	9712.61	938.85	856.75	0.0114	0.0024
10	936.56	211.68	94745.64	97488.75	9447.84	8668.75	0.1140	0.0250
11	177.88	24.90	12898.9	10049.76	982.11	836.55	0.0538	0.0067
12	1773.42	249.36	126180.46	101760.47	9874.81	8411.86	0.5443	0.0728
13	63.48	13.23	7135.35	7846.00	4839.09	4682.89	0.0037	0.0009
14	632.5	132.92	69003.11	79134.71	48625.09	46741.63	0.0373	0.0097
15	119.77	21.06	12559.73	12502.63	5407.77	3651.62	0.0159	0.0032
16	1194.1	210.50	123091.72	125168.57	54054.38	36642.37	0.1604	0.0331
17	225.32	24.92	16905.58	14606.33	6869.7	3767.97	0.0755	0.0086
18	2247.73	248.54	165927.01	145896.11	68618.91	37504.90	0.7591	0.0933

In order to illustrate the performances of the heuristics, Figure 6 represents: (a) the means of cutting bars resulting in *OptimizationDistBSP*, *Optimization-TREE*, FFD and Greedy heuristics, and it means minimum number of bars ("B_bar"); (b) the standard deviation of cutting bars. Figure 7 represents: (a) the means of leftover, also calculated for the 4 heuristics, and mean of leftover of hypothetical cutting pattern; (b) the standard deviation of leftover. After that, Figure 8 is the same for the losses and Figure 9 presents the runtime for the heuristics. The theoretical value B_bar is also represented in figures.

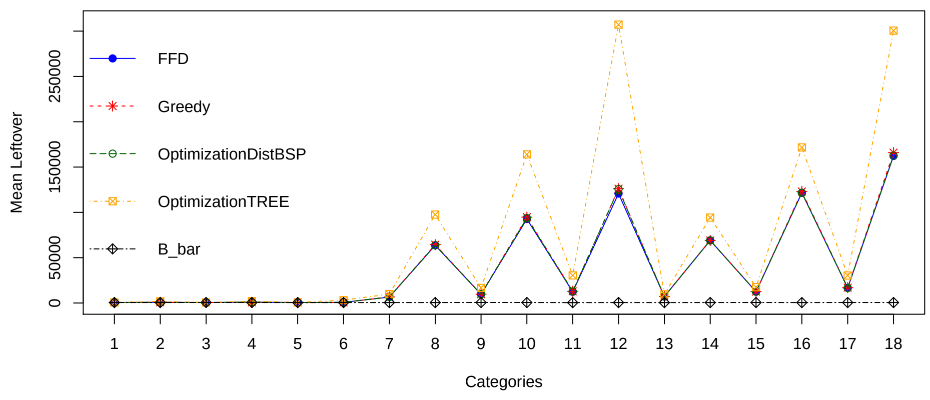


(a) Mean of Cut Bars

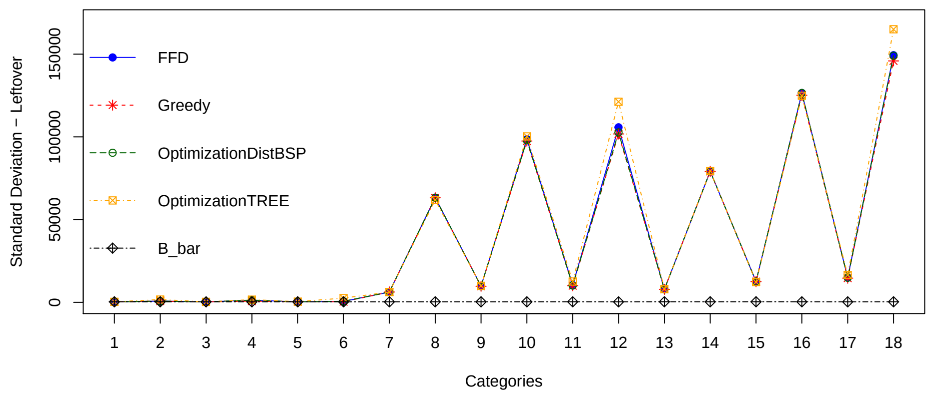


(b) Standard Deviation of Cut Bars

Fig. 6: Statistical Measures for the Number of Cut Bars

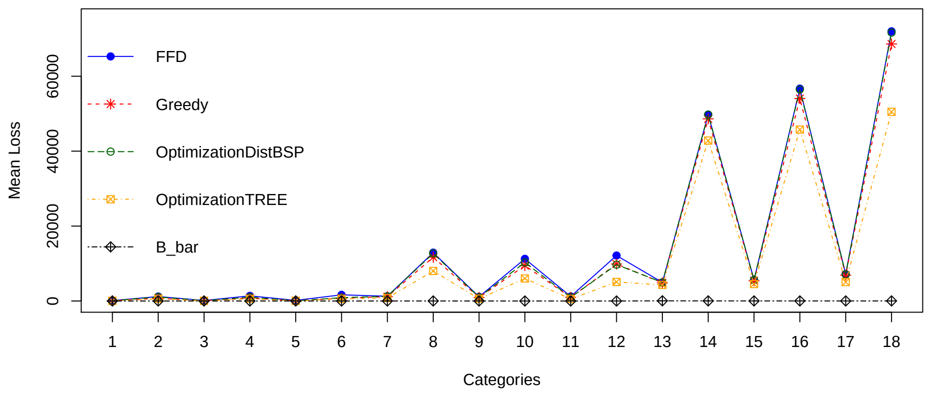


(a) Mean of Total Leftover

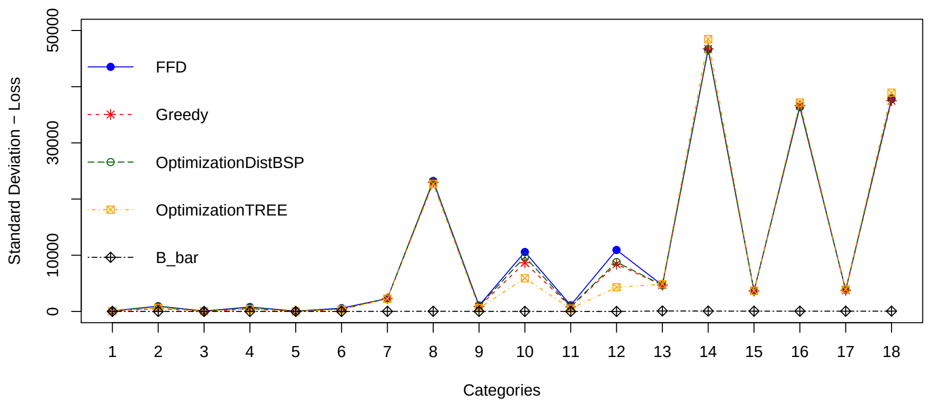


(b) Standard Deviation of Cut Bars

Fig. 7: Statistical Measures for the Leftover

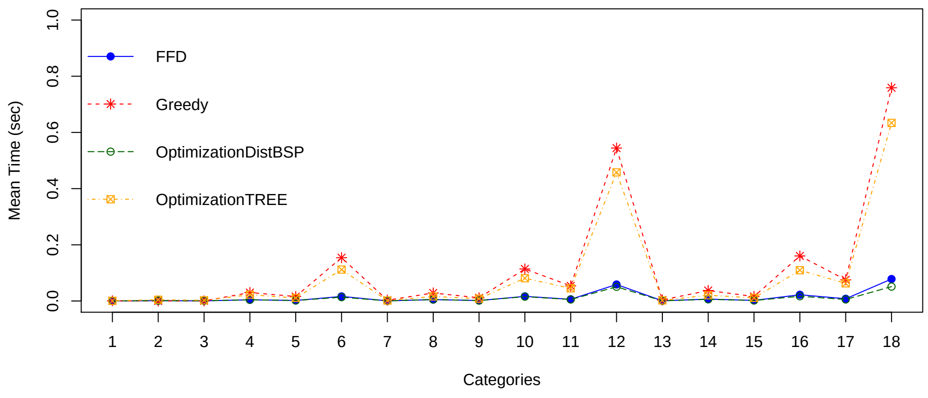


(a) Mean of Total Loss

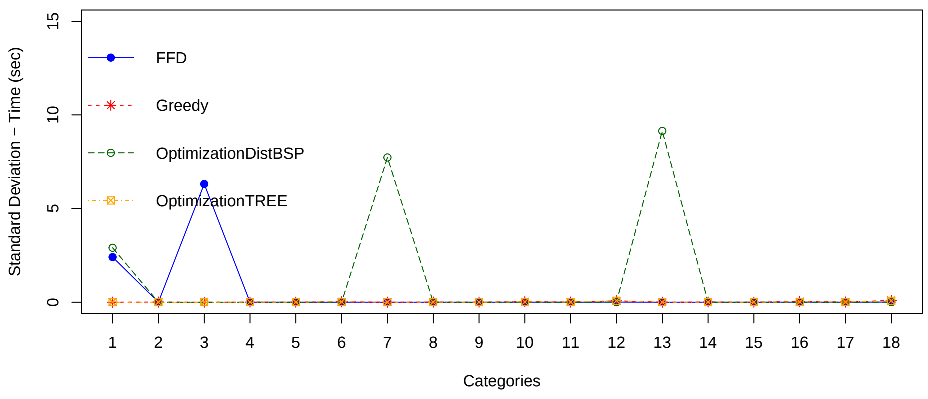


(b) Standard Deviation of Total Loss

Fig. 8: Statistical Measures for the Loss



(a) Mean of Total Runtime



(b) Standard Deviation for the Runtime

Fig. 9: Statistical Measures for the Runtime

The statistical measures presented in Figures 6-9 indicate the performance across categories, considering the number of cutting bars, the leftover, the loss and the runtime of the proposed heuristics (*OptimizationTREE* and *OptimizationDistBSP*) and the FFD and Greedy heuristics, according to algorithms proposed by [6]. Observing the results of Tables 3, 4, 5 and 6 we can notice that:

– The *OptimizationDistBSP* heuristic presents the number of cutting bars similar to FFD heuristic, however, it presents more leftovers and less losses, as desirable.

- The *OptimizationTREE* heuristic uses a little more bars than FFD and Greedy, but presenting more leftovers and the lowest losses in all categories.
- The *OptimizationTREE* and *OptimizationDistBSP* heuristics have more leftovers and less losses when compared to FFD and Greedy heuristics.
- The FFD heuristic provides more losses, that is, small parts spread over the bars used in the cutting process. The *OptimizationTREE* and *OptimizationDistBSP* heuristics provide less losses, which are concentrated in leftover. Thus this leftover can be used to attend the next demand.
- Figure 9 shows that runtime of the Greedy heuristic is higher than the *OptimizationTREE* and *OptimizationDistBSP* heuristics, especially for bigger exemplars.
- Thus, although the 4 heuristics provide a similar number of cutting bars, the FFD and Greedy heuristics generate less leftovers and more losses than the 2 heuristics proposed in this work (*OptimizationTREE* and *OptimizationDistBSP*), performing worse.
- As expected, when observing Figure 8, the *OptimizationTREE* heuristic generates the smallest number of loss, followed by *OptimizationDistBSP* heuristic, showing the more desirable performance.
- Figure 7 shows that the *OptimizationTREE* heuristic generates more leftovers, followed by *OptimizationDistBSP* heuristic. Generating leftovers is desirable when the number of cutting bars is close to B_{bar}, since leftovers are used to meet next demands. In the proposed heuristics, the leftovers are concentrated in the last bars. It is important to note that the peaks presented by *OptimizationTREE* heuristic in Figure 7 are due to the use of the smallest items. In addition, the smallest item for each instance is fixed according to the smallest value of l_i. During the execution of the algorithm, the demand for the items is met and then these items are no more considered for cutting. The remaining items are larger items since the algorithm removes a unit from the largest item in each iteration. Therefore, from a given bar, more leftovers and very few losses are generated, since the higher demand causes these last cutting patterns to be repeated more times, causing the visible peaks.

Another analysis can be done between the hypothetical cutting pattern that presents the minimum number of bars, the leftover and the loss resulting of this hypothetical pattern. The distance of these values and the results of heuristics are presented in the following equations. The $\overline{GAP}_{bar}$ defined in Equation 8 presents the mean value of the distance (absolute error) between lower bound of bars (B_{bar_p}) and the heuristic result (H_{bar_p}), for each category. The "H" in the equation represents the chosen heuristic.

$$\overline{GAP}_{bar} = \frac{\sum_{p=1}^{P} \left| B_{bar_p} - H_{bar_p} \right|}{P} \tag{8}$$

Similarly, the $\overline{GAP}_{left}$ (Equation 9) and the $\overline{GAP}_{loss}$ (Equation 10) show the main value of the distance (absolute error) between lower bound of leftovers

(B_{left_p}) and of loss (B_{loss_p}), and the heuristic result of leftover (H_{left_p}) and of loss (H_{loss_p}), respectively, for each category.

$$\overline{GAP}_{left} = \frac{\sum_{p=1}^{P} \left| B_{left_p} - H_{left_p} \right|}{P} \tag{9}$$

$$\overline{GAP}_{loss} = \frac{\sum_{p=1}^{P} \left| B_{loss_p} - H_{loss_p} \right|}{P} \tag{10}$$

Tables 7, 8, 9 and 10 show the values of the GAP obtained by *OptimizationDistBSP*, *OptimizationTREE*, FFD and Greedy heuristics, respectively, considering the number of cutting bars, the leftover and the loss, for each category. The columns present the numeric value provided by Equations 8, 9 and 10, respectively, followed by its respective standard deviations.

Table 7: GAP values obtained by *OptimizationDistBSP* heuristic.

categories	*OptimizationDistBSP*					
	$\overline{GAP}_{bar}$	$\sigma(GAP_{bar})$	$\overline{GAP}_{left}$	$\sigma(GAP_{left})$	$\overline{GAP}_{loss}$	$\sigma(GAP_{loss})$
1	0.12	0.32	185.48	276.82	86.18	78.01
2	1.38	1.07	646.31	658.62	1070.59	905.66
3	0.06	0.23	125.89	206.16	70.35	52.84
4	1.54	1.55	783.27	1287.93	1019.69	695.98
5	0.04	0.19	87.94	169.85	53.04	33.24
6	1.24	0.86	563.21	556.42	889.16	497.45
7	7.37	6.15	6170.92	6158.17	1249.7	2287.80
8	75.98	62.29	63367.08	63135.49	12697.48	23226.08
9	10.12	9.66	9147.54	9724.21	1014.72	952.99
10	103.57	96.93	93284.53	97952.33	10289.07	9716.17
11	13.36	10.09	12387.69	10203.07	975.83	887.44
12	134.89	101.67	125186.83	102751.18	9702.27	8783.22
13	11.59	7.53	6768.71	7781.44	4907.77	4647.68
14	118.3	75.76	68736.51	79102.11	49732.41	46638.59
15	17.56	11.93	12078.8	12469.25	5593.8	3644.60
16	177.61	119.91	121247.96	126409.70	56448.3	36460.36
17	23.21	13.73	16158.74	14817.42	7111.66	3808.66
18	233.76	137.24	162182.21	148963.63	71634.27	37909.70

Table 8: GAP values obtained by *OptimizationTREE* heuristic.

categories	*OptimizationTREE*					
	$\overline{GAP}_{bar}$	$\sigma(GAP_{bar})$	$\overline{GAP}_{left}$	$\sigma(GAP_{left})$	$\overline{GAP}_{loss}$	$\sigma(GAP_{loss})$
1	0.16	0.36	186.81	315.96	51.98	52.94
2	1.94	1.63	1495.33	1556.33	580.89	619.39
3	0.12	0.32	144.65	243.08	51.83	37.70
4	2.19	1.64	1644.6	1487.25	580.68	410.92
5	0.36	0.50	343.83	407.17	48.84	29.37
6	3.31	2.56	2652.79	2526.04	573.52	340.25
7	10.17	6.08	9386.42	6115.37	799.12	2234.95
8	105.12	60.70	97123.55	61807.79	8011.19	22680.46
9	16.71	9.80	16081.02	9967.12	590.13	577.85
10	169.52	98.89	163492.63	100302.53	6013.73	5901.85
11	30.61	12.41	30036.07	12604.16	533.6	508.34
12	311.83	120.10	306699.3	121315.01	5080.81	4297.07
13	13.39	7.37	9171.74	7998.62	4242.74	4827.73
14	136.61	72.96	93856.57	79271.29	42807.97	48446.13
15	21.48	11.62	16974.65	12302.03	4506.24	3705.93
16	216.95	117.68	171244.54	124835.80	45734.0	37160.81
17	34.88	14.94	29841.62	16499.66	5041.4	3917.19
18	350.55	149.63	300056.67	164996.14	50487.38	38921.53

Table 9: GAP values obtained by FFD heuristic.

categories	**FFD**					
	$\overline{GAP}_{bar}$	$\sigma(GAP_{bar})$	$\overline{GAP}_{left}$	$\sigma(GAP_{left})$	$\overline{GAP}_{loss}$	$\sigma(GAP_{loss})$
1	0.13	0.33	208.36	268.28	114.12	90.15
2	1.44	1.13	539.07	674.75	1187.55	934.93
3	0.09	0.28	194.77	212.40	134.99	71.05
4	1.77	1.44	644.8	1237.63	1364.14	786.31
5	0.18	0.38	285.49	269.84	159.69	54.94
6	1.85	0.80	460.55	480.81	1648.67	579.03
7	7.39	6.13	6181.48	6147.36	1268.56	2285.67
8	76.08	62.19	63273.76	63199.99	12897.42	23203.40
9	10.14	9.64	9094.62	9749.99	1119.68	1054.04
10	103.41	96.96	92166.23	98645.99	11257.53	10575.1
11	13.14	10.25	11955.94	10478.30	1209.84	1099.95
12	132.42	103.40	120318.91	105809.26	12136.39	10936.85
13	11.59	7.53	6768.18	7781.89	4910.02	4646.26
14	118.31	75.75	68726.91	79110.39	49737.41	46637.20
15	17.59	11.90	12069.46	12477.88	5629.02	3637.54
16	177.85	119.64	121226.17	126430.49	56669.95	36363.24
17	23.22	13.73	16118.94	14852.41	7168.74	3806.61
18	233.72	137.31	161814.88	149279.17	71973.32	37775.83

Table 10: GAP values obtained by Greedy heuristic.

categories	**Greedy**					
	$\overline{GAP}_{bar}$	$\sigma(GAP_{bar})$	$\overline{GAP}_{left}$	$\sigma(GAP_{left})$	$\overline{GAP}_{loss}$	$\sigma(GAP_{loss})$
1	0.09	0.28	144.86	249.41	68.4	45.33
2	0.92	0.94	554.58	651.20	724.12	517.50
3	0.04	0.19	105.17	166.62	73.27	39.46
4	0.99	0.97	534.06	624.99	767.0	431.54
5	0.08	0.27	134.91	228.86	70.19	30.52
6	0.87	0.59	401.98	282.47	737.84	350.64
7	7.34	6.18	6224.05	6157.10	1160.61	2272.34
8	75.6	62.68	63983.04	63128.63	11667.6	22958.05
9	10.12	9.67	9228.19	9687.68	936.45	854.92
10	103.67	96.84	94224.46	97514.91	9446.36	8667.24
11	13.46	10.03	12486.42	10114.38	981.66	836.60
12	135.58	100.91	125706.92	101844.12	9874.52	8411.79
13	11.48	7.62	6772.36	7781.90	4796.2	4658.77
14	117.11	76.58	68637.42	79082.10	48596.0	46729.27
15	17.43	12.05	12116.85	12432.50	5387.47	3653.88
16	176.66	120.76	122673.78	125122.68	54033.34	36636.11
17	23.24	13.69	16454.53	14553.84	6857.45	3769.11
18	234.03	136.63	165455.16	145882.12	68597.16	37510.68

Analyzing Tables 7, 8, 9 and 10, the *OptimizationTREE* heuristic presents the smallest GAP in relation to the number of losses (B_{bar}). The *OptimizationDistBSP* heuristic presents, in average, the same gap than FFD and Greedy heuristics, but providing more leftovers. *OptimizationTREE* is closer to B_{bar} and generates even bigger leftovers.

The computational complexity of the proposed algorithms are shown in Table 11, considering the worst case of the two heuristics proposed. The second column presents the number of sorts performed by each algorithm during its execution and the last column shows the complexity order of the algorithms, considering the worst case.

Table 11: Computational Complexity

Algorithm	**Sorts**	**Order**
OptimizationDistBSP	1	$\mathcal{O}(n^2)$
OptimizationTREE	1	$\mathcal{O}(n^2)$
FFD	1	$\mathcal{O}(n^2)$
Greedy	2	limited by the knapsack problem solver

Table 9 shows that the *OptimizationDistBSP* algorithm presents, in the worst case, a quadratic order computational complexity ($\mathcal{O}(n^2)$). According to results, this heuristic uses the same number of cutting bars than FFD and Greedy heuristics (Figure 6), whose computational complexity is limited by the knapsack problem solver. The *OptimizationTREE* algorithm also presents quadratic order $\mathcal{O}(n^2)$, but providing more leftovers than the others to be reused, which is desirable, since *OptimizationTREE* aims to convert losses into leftovers, providing reuse. In addition, *OptimizationDistBSP* runs all categories using less time. The main runtime for all the 18 categories (in seconds) are: *OptimizationDistBSP*=0.0101, *OptimizationTREE*=0.0890, FFD = 0.01282 and Greedy = 0.1116. Then, for the case that considers unlimited quantity of one type of bar, *OptimizationDistBSP* and *OptimizationTREE* are more appropriated than FFD and Greedy heuristics.

7 Conclusion

In this paper, one-dimensional cutting stock problems are considered, which minimizes the number of cutting bars in cutting process, satisfying the demand, considering cutting rolls with known demand, and using the leftovers provided from previous rolls cutting process. In order to accomplish this goal, two heuristic procedures are proposed: *OptimizationDistBSP* and *OptimizationTREE* heuristics. The performances of the proposed heuristics are compared with the FFD and Greedy algorithms, according to [6], which presents the same objective: to minimize the number of cut bars.

It is important to note that when the number of bars is minimized, less losses are produced. With a better positioning of items in bars, the losses are converted in leftovers. In this paper, a loss is an unused bar smaller than the size of the smallest item demanded. However, the loss can be defined by the producer, for example, as a fixed value or a minimum acceptable percentage.

Using the heuristics proposed in this paper, the leftovers from previous cutting process can be used in next item's demands. This fact, therefore, offers a possible use of leftovers, instead of their disposal. Therefore, the results of this paper contribute to the reduction of the raw materials use, to the environment preservation and to the reduction of waste production, providing commercially viable solutions.

Finally, the proposed heuristics were implemented in open source (Python language) and the codes are freely available in the GitHub package resource [1]. All the exemplars used in this work are available on both links.

As prospects for continuing this research, we work in inclusion of a 3^{rd} phase in algorithms, i.e. after defining the all cutting patterns, changing the items distribution to aim to convert losses in leftover. This new phase must be carefully designed since it can increase the processing time.

Conflict of interest

The authors declare that they have no conflict of interest.

[1] *OptimizationTREE*: github.com/omatheuspimenta/heuristictree
OptimizationDistBSP: github.com/omatheuspimenta/heuristiciohbsp

Data Availibility Statement

The data and the codes presented in this study are openly available in GitHub package resource:
github.com/omatheuspimenta/heuristictree and
github.com/omatheuspimenta/heuristiciohbsp.

References

1. Abuabara, A., Morabito, R.: Cutting optimization of structural tubes to build agricultural light aircrafts. Annals of Operations Research **169**(1), 149 (2009)
2. Ali, R., Muhammad, S., Takahashi, R.H.: Decision making viva genetic algorithm for the utilization of leftovers. International Journal of Intelligent Systems **36**(4), 1746–1769 (2021)
3. Arbib, C., Marinelli, F., Rossi, F., Di Iorio, F.: Cutting and reuse: an application from automobile component manufacturing. Operations Research **50**(6), 923–934 (2002)
4. Arenales, M.N., Cherri, A.C., Nascimento, D.N.d., Vianna, A.: A new mathematical model for the cutting stock/leftover problem. Pesquisa Operacional **35**(3), 509–522 (2015)
5. Campello, B., Ghidini, C., Ayres, A., Oliveira, W.: A residual recombination heuristic for one-dimensional cutting stock problems. TOP pp. 1–27 (2021)
6. Cerqueira, G.R., Aguiar, S.S., Marques, M.: Modified greedy heuristic for the one-dimensional cutting stock problem. Journal of Combinatorial Optimization pp. 1–18 (2021)
7. Cherri, A.C., Arenales, M.N., Yanasse, H.H.: The one-dimensional cutting stock problem with usable leftover–a heuristic approach. European Journal of Operational Research **196**(3), 897–908 (2009)
8. Cherri, A.C., Arenales, M.N., Yanasse, H.H.: The usable leftover one-dimensional cutting stock problem—a priority-in-use heuristic. International Transactions in Operational Research **20**(2), 189–199 (2013)
9. Cherri, A.C., Arenales, M.N., Yanasse, H.H., Poldi, K.C., Vianna, A.C.G.: The one-dimensional cutting stock problem with usable leftovers–a survey. European Journal of Operational Research **236**(2), 395–402 (2014)
10. Cui, Y., Song, X., Chen, Y., Cui, Y.P.: New model and heuristic solution approach for one-dimensional cutting stock problem with usable leftovers. Journal of the Operational Research Society **68**(3), 269–280 (2017)
11. Cui, Y., Yang, Y.: A heuristic for the one-dimensional cutting stock problem with usable leftover. European Journal of Operational Research **204**(2), 245–250 (2010)
12. Delorme, M., Iori, M., Martello, S.: Bin packing and cutting stock problems: Mathematical models and exact algorithms. European Journal of Operational Research **255**(1), 1–20 (2016)

13. Gilmore, P.C., Gomory, R.E.: A linear programming approach to the cutting-stock problem. Operations research **9**(6), 849–859 (1961)
14. Gilmore, P.C., Gomory, R.E.: A linear programming approach to the cutting stock problem—part ii. Operations research **11**(6), 863–888 (1963)
15. Gilmore, P.C., Gomory, R.E.: Multistage cutting stock problems of two and more dimensions. Operations research **13**(1), 94–120 (1965)
16. Gradišar, M., Jesenko, J., Resinovič, G.: Optimization of roll cutting in clothing industry. Computers & Operations Research **24**(10), 945–953 (1997)
17. Gradišar, M., Kljajić, M., Resinovič, G., Jesenko, J.: A sequential heuristic procedure for one-dimensional cutting. European Journal of Operational Research **114**(3), 557–568 (1999)
18. Gradisar, M., Trkman, P.: A combined approach to the solution to the general one-dimensional cutting stock problem. Computers & Operations Research **32**(7), 1793–1807 (2005)
19. Johnson, D.S.: Near-optimal bin packing algorithms [ph. d. thesis]. Boston: Massachusetts Institute of Technology (1973)
20. Koch, S., König, S., Wäscher, G.: Integer linear programming for a cutting problem in the wood-processing industry: a case study. International Transactions in Operational Research **16**(6), 715–726 (2009)
21. Leighton, F.: Introduction to Parallel Algorithms and Architectures: arrays, trees, hypercubes. Morgan Kaufmann (1992)
22. Luenberger, D., Ye, Y.: Linear and Nonlinear Programming, Third Version. Springer Science and Business Media LLC (2008)
23. Martello, S., Toth, P.: Knapsack problems: algorithms and computer implementations. Wiley-Interscience series in discrete mathematics and optimiza tion (1990)
24. Pisinger, D.: Algorithms for knapsack problems. Thesis-DIKU, University of Copenhagen, Copenhagen (1995)