Journal of Heuristics (2020) 26:585–618
https://doi.org/10.1007/s10732-020-09443-z

Meta-heuristics for the one-dimensional cutting stock problem with usable leftover

Santiago V. Ravelo[1] · Cláudio N. Meneses[2] · Maristela O. Santos[3]

Received: 7 May 2019 / Revised: 28 January 2020 / Accepted: 21 February 2020 / Published online: 3 March 2020

Abstract
This work considers the one-dimensional cutting stock problem in which the non-used material in the cutting patterns may be used in the future, if large enough. We show that a multiobjective criteria to classify the solutions could be more accurate than previous classifications attempts, also we give a heuristic algorithm and two meta-heuristic approaches to the problem and we use them to solve practical and randomly generated instances from the literature. The results obtained by the computational experiments are quite good for all the tested instances.

Keywords Cutting problem · Combinatorial optimization · Multiobjective optimization · Meta-heuristics

1 Introduction

Cutting stock problems consist of cutting a set of items from a stock of larger objects with the objective of producing cut patterns in order to satisfy the items demands, while minimizing the loss material, the cost of the objects to be cut or maximizing the profit of the produced items. These problems arise from many applications in industrial processes such as paper tubes, plastic films, metal and construction industries. For example, in the wooden working industry the application of solutions that minimized

✉ Santiago V. Ravelo
santiago.ravelo@inf.ufrgs.br

Cláudio N. Meneses
claudio.meneses@ufabc.edu.br

Maristela O. Santos
mari@icmc.usp.br

1 Institute of Informatics, Federal University of Rio Grande do Sul, Porto Alegre, Brazil

2 Center of Mathematics, Computation and Cognition, Federal University of ABC, Sao Paulo, Brazil

3 Institute of Mathematical Sciences and Computation, University of Sao Paulo, Sao Carlos, Brazil

the wasted material in the cutting process represented for some companies 30% of cost savings (Ogunranti and Oluleye 2016), such solutions not only save companies costs but they also reduce greenhouse gas emission being an approach toward green manufacturing (Wattanasiriseth and Krairit 2019). Another industry that may reduce the costs and the negative environmental impact by applying solutions to minimize the waste is the textile, where about 20% of fabric waste is produced by the leftovers after the patterns for the clothes have been cut and such leftovers unusually are recycled (Šajn 2019).

These problems are not only interesting by their practical applications, but also by the theoretical challenge they represent since, generally, they are computationally difficult to solve and belong to the NP-hard class. Cutting stock problems can be classified in several ways (Wäscher et al. 2007) and, in this paper, we consider one of those classifications: the one-dimensional cutting stock problem with usable leftover. In this extension of the problem, if the non-used material of the cutting patterns is large enough, then it may be used in the future. Such feature introduces more difficulties to the problem and has received attention in the literature, creating mathematical models and several algorithms to get *good* solutions. The difficulties introduced by this feature also affect the definition of what makes a *good* solution and, sometimes, the proposed classifications lack in accuracy, for that reason the next section gives a detailed explanation on these criteria.

To the best of our knowledge, the first formulation for the cutting stock problem considering the existence of usable leftovers (retails) was given in Gradišar et al. (1997). The objectives of that model were to minimize the total unsatisfied demands and also, to minimize the total loss of material generated by the cutting patterns, where if the leftover of a pattern was large enough then it was considered a retail and not a loss of material. Besides the model, the authors developed a greedy heuristic to solve the problem. The same model was considered in Gradišar et al. (1999); Gradišar and Trkman (2005), where new restrictions on the maximum length of the retails were added and heuristics approaches for the problems were given.

In Abuabara and Morabito (2009) the authors studied applications in the agricultural light aircraft industry and proposed two new models by modifying the model of Gradišar et al. (1997), where restrictions on the number of generated retails were added and the objective was to minimize the total loss of material. In Cherri et al. (2009) classifications on the quality of the solutions were given and several heuristics were developed and tested, some of them based on greedy criteria while others were residual heuristics. A comparative study of the previous formulations was presented in Ravelo et al. (2010) and also new models were proposed trying to avoid the existence of symmetrical solutions. The objectives of those models were to minimize the loss of material and to minimize the number of generated retails, being one of those new models very suitable for column generation methods.

Some of the results from Cherri et al. (2009) were improved by a new heuristic proposal given in Cui and Yang (2010). In that article the authors created a two stages heuristic, where the first part fulfills the majority of the items demands using linear programming techniques while the second part fulfills the remaining demands by a sequential heuristic. In Gradišar et al. (2011) a study of the problem and existing solutions was done, also two algorithms were proposed for instances where the items were much smaller than the objects (at most $\frac{1}{3}$ of the object size). In Cherri et al. (2014) a survey was presented where the main contributions to the problem were discussed. In Arenales et al. (2015) one of the models from Ravelo et al. (2010) was extended and a new model was given, where linear relaxation and column generation techniques were applied in order to solve it.

Since the majority of the algorithmic approaches found in the literature for this problem are greedy heuristics or heuristics based on mathematical formulations, we explore in this article the applicability of some meta-heuristics ideas to solve the problem. We give a constructive heuristic which uses some of the previous ideas to generate initial solutions that will be used by our main proposal: a GRASP based meta-heuristic and an evolutive based algorithm.

This paper is organized as follows. Section 2 formally presents the problem and discuss on the criteria of what would be a good solution classification. Section 3 describes our algorithmic approaches to solve the problem: a constructive heuristic, a GRASP meta-heuristic and an evolutive algorithm. Section 4 presents computational tests of our algorithms over several instances from the literature, also compares our solutions with the best previous results we found. Finally, Sect. 5 gives the conclusions and future directions.

2 Problem definition

The one-dimensional cutting stock problem with usable leftover (1D- CSPUL) was previously defined in Cherri et al. (2009) as:

Definition 1 On the 1D- CSPUL a set of pieces (items) must be produced by cutting large units (objects) of standard sizes (objects bought from suppliers) or non-standard (objects that are leftover of previous cuts). The demand of the items and the availability of the objects are given. Demands must be met by cutting the available objects such that the leftovers are "small" (denoted by scrap) or "sufficiently large" (denoted by retails) to return to stock, but in a reduced number.

The definition we use is very similar, but we change the objective of returning a lower number of retails to stock by minimizing the final number of retails in stock. So we work with the following definition:

Definition 2 On the 1D- CSPUL a set of items must be produced by cutting objects bought from suppliers or objects that are retails of previous cuts. The demand of the items and the availability of the objects are given. Demands must be met by cutting the available objects such that the leftovers are "small" (denoted by scrap) or "sufficiently large" (denoted by retails) to return to stock, in order to reduce the final number of retails in stock.

Those definitions of the problem use the terms *small* and *sufficiently large*, which need to be formally defined. In order to do that, one can rely on experience and previous knowledge of the problem or we may use the parameters β, θ and δ (defined in Cherri et al. 2009):

- $\beta \in (0, 1)$: is used to know when the leftover material of a standard object is considered little scrap, for that the size of the leftover must be less than or equal to βL, where L is the size of the object.
- $\theta \in (0, 1)$: is used to know when the leftover of a non-standard object is considered little scrap, for that the size of the leftover must be less than or equal to θL, where L is the size of the object.
- $\delta \in \mathbb{R}_+^*$: is used to know when the leftover material of any object is considered retail, for that the size of the leftover must be greater than or equal to δ.
- If the leftover material is not a little scrap and neither a retail, then it is considered a scrap of intermediate size.

Observe that, the 1D- CSPUL must minimize the loss of material and also has to minimize the final number of retails in stock. Then, the problem has more than one objective implying that it requires some classification of the solutions by their quality. The following definition, proposed in Cherri et al. (2009), attempts to classify the solutions:

Definition 3 The solutions of a 1D- CSPUL are defined as:

- *Ideal solution* when a *small* number of objects have little scraps and none of the objects have scraps of intermediate size. In case there are retails, they must be concentrated in a *very small* number of cut objects.
- *Acceptable solution* when a *small* number of objects present scraps of intermediate size and a *small* number of objects present retails.
- *Undesirable solution* when *several* cut objects present scraps of intermediate size or present *several* retails.

Definition 3 also uses imprecise terms as *very small*, *small* and *several*. Those terms were formalized in Cherri et al. (2009) using two parameters: ϵ_1 and ϵ_2, where $0 < \epsilon_1 < \epsilon_2 < 1$, so considering η represents the number of object cuts in the solution:

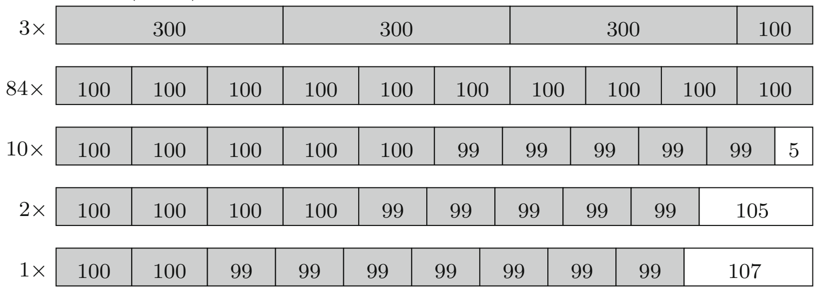
- *very small* number of object cuts: up to $\lceil \epsilon_1 \eta \rceil$;
- *small* number of object cuts: up to $\lceil \epsilon_2 \eta \rceil$;
- *several* object cuts: above $\lceil \epsilon_2 \eta \rceil$.

By using that definition, the aim is to generate ideal solutions or at least acceptable ones. Although Definition 3 gives an interesting classification of the solutions, it is not suitable for general purpose. As we will show, there are instances with an *acceptable* solution better than an *ideal* solution (even we can find instances with *undesirable* solutions as goods as *ideal* solutions). In order to prove that, consider the following instance:

- objects number: 200, all objects are standard and have same length: 1000,
- items sizes: 99, 100 and 300, with demands: 67, 903 and 9, respectively,
- $\beta = 0.005$, $\delta = 99$, $\epsilon_1 = 0.03$ and $\epsilon_2 = 0.1$.

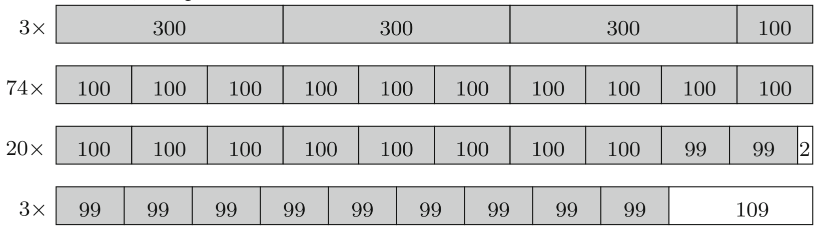
Now we analyze three solutions: an *ideal*, an *acceptable* and an *undesirable* one.

- **Solution 1** (*Ideal*):



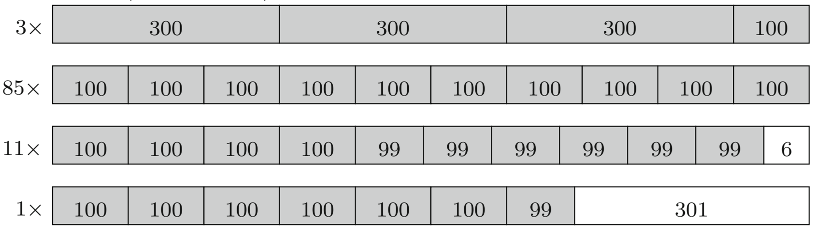
  - three cuts, each one with three items of size 300 and one item of size 100, without loss or retails (3 × (3 × 300 + 100) = 3 × 1000)
  - 84 cuts of ten items of size 100 each cut, without loss or retails (84 × (10 × 100) = 84 × 1000)
  - ten cuts with five items of size 100 and five items of size 99 each cut, it generates ten little scraps of size 5 and a total loss of length 50 (10 × (5 × 100 + 5 × 99) = 10 × (1000 − 5))
  - two cuts with four items of size 100 and five items of size 99, it generates two retails of size 105 (2 × (4 × 100 + 5 × 99) = 2 × (1000 − 105))
  - one cut with two items of size 100 and seven items of size 99, it generates a retail of size 107 (2 × 100 + 7 × 99 = 1000 − 107)

- **Solution 2** (*Acceptable*):



  - three cuts, each one with three items of size 300 and one item of size 100, without loss or retails (3 × (3 × 300 + 100) = 3 × 1000)
  - 74 cuts of ten items of size 100 each cut, without loss or retails (74 × (10 × 100) = 74 × 1000)
  - 20 cuts with eigth items of size 100 and two items of size 99 each cut, it generates 20 little scraps of size 2 and a total loss of length 40 (20 × (8 × 100 + 2 × 99) = 20 × (1000 − 2))
  - three cuts, each one with nine items of size 99, it generates three retails of size 109 (3 × (9 × 99) = 3 × (1000 − 109))

- **Solution 3** (*Undesirable*):



  - three cuts, each one with three items of size 300 and one item of size 100, without loss or retails ($3 \times (3 \times 300 + 100) = 3 \times 1000$)
  - 85 cuts of ten items of size 100 each cut, without loss or retails ($85 \times (10 \times 100) = 85 \times 1000$)
  - 11 cuts with four items of size 100 and six items of size 99 each cut, it generates 11 scraps of intermediate size of 6 and a total loss of length 66 ($11 \times (4 \times 100 + 6 \times 99) = 11 \times (1000 - 6)$)
  - one cut with six items of size 100 and one item of size 99, it generates one retail of size 301 ($6 \times 100 + 99 = 1000 - 301$)

Each one of the above solutions has 100 cuts and according to Definition 3 the first solution is *ideal*, the second one is *acceptable* and the third one is *undesirable*. Observe that the total loss of material of the second solution (40) is lower than the total loss of the first one (50) having, both solutions, the same number of retails (3). So, the second solution (an *acceptable*) is actually better than the first solution (an *ideal*), while according to Definition 3 it is not better, but it is worse.

Also, the third solution, an *undesirable* one according to Definition 3, has only one retail, less than the three retails of the first and second solutions. If we consider the total length of the cuts made (100000), then the total loss difference (16) between the third solution and the first one could be despicable. So, that *undesirable* solution is at least as good as the *ideal* and the *acceptable* solutions of the example. Figure 1 shows the Pareto-dominance relationship between those solutions.

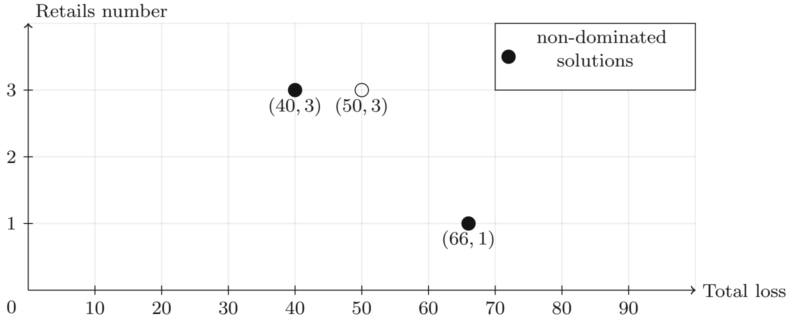


Fig. 1 In the graphic the *ideal* solution is (50, 3), the *acceptable* one is (40, 3) and the *undesirable* is (66, 1). Observe that the *ideal* solution is a dominated solution while the *acceptable* and the *undesirable* are non-dominated

Besides the above discussion, there are instances with no ideal nor acceptable solutions (instances of Sect. 4.2 are two examples). Furthermore, the classification given by Definition 3 is not suitable when we consider demands arrive in different time periods, since a "good" solution may generate more retails at some period that will be consumed in the following periods guaranteeing a small number of retails at the final stock (multi-period instances of Sects. 4.4 and 4.5 are examples). For those reasons, we decided to avoid the use of Definition 3 in the solutions classification during the process and find a more adequate criteria.

Some other approaches may consider classifications criteria that represent their particular scenarios:

- **Limited stocking area** This scenario appears in applications where the number of produced retails is limited by a stocking space (e.g., in Abuabara and Morabito 2009 was studied the problem applied to the agricultural light aircraft industry, where the retails number could not exceed a given threshold). In this case, the objective is to minimize the loss of material while guaranteeing the number of produced retails do not exceed a given upper bound.
- **Bounded scrap disposer** This scenario appears in applications where the capability of discarding the loss material of each cutting plan has limitations (e.g. environmental regulations may require factories to not produce a large amount of pollution resulting from the disposal process). In this case, the objective is to minimize the number of produced retails while guaranteeing the length of the loss material do not exceed a given upper bound.
- **Minimum production cost** This scenario appears in applications where there are no bounds for the number of retails or loss of material, but there exist costs to stock each produced retail and also to discard the scraps. In this case, the objective is to minimize the overall production cost. This is one of the most common scenarios since many industries try to minimize the production costs although, the material waste and the retails stocking costs vary from one application to another, implying that strategies to solve the problem may be adapted to attend the specific instance settings.

Those scenarios are not equivalent, meaning that feasible or "good" solutions for one scenario may not be neither feasible nor "good" for another. However, in all cases, the final number of retails in stock and the total material loss are requested to be upper-bounded or minimized. Notice that, one may reduce the material loss by avoiding to cut many items from an object (producing more retails), or to minimize the number of retails by considering more leftovers as material loss (producing more loss). Thus, to minimize the total loss of material and to minimize the final number of retails at the stock are conflicting objective functions implying that problem can be considered as a multi-objective problem (Eschenauer et al. 1990) and, instead of searching for one solution, the goal is to approximate the Pareto frontier: a maximal set containing non-dominated solutions of an instance (i.e., solutions that are no worse in all the objectives than any other solution).

If we consider the problem as a multi-objective one, where the objectives are to minimize the loss of material and to minimize the final number of retails in stock, then, by adopting the dominance criterion, we obtain a classification rule suitable for the diverse group of scenarios mentioned before. In order to prove that, consider the maximal set S of non-dominated solutions for a given instance (without considering the particular settings of the scenarios) and X^* an optimal solution of the instance for one of the scenarios. Since S is maximal, it must contain X^* or a solution that dominates X^* (i.e., a solution that is not worse in both objectives and strictly better in at least one of the objectives). Therefore, S contains an optimal solution for the instance in the scenario where X^* is optimal. Consequently, methods that attempt to approximate the Pareto frontier can be used for different scenarios of the problem without any adaptation, picking at the end of the process, the cutting plan that fits better to the problem specifications.

In the following section we propose one heuristic and two meta-heuristics to solve the 1D- CSPUL, considering the multi-objective approach of the problem.

3 Algorithms

In this section we propose and describe heuristics for the 1D- CSPUL. First we present a constructive heuristic based on the First Fit algorithm (FF). Our constructive heuristic is used to generate the initial solutions for a GRASP based algorithm and an evolutive based meta-heuristic, those algorithms will be introduced in the following subsections.

3.1 Constructive heuristic

The FF procedure creates a pattern cut for an object at a time, adding to the pattern the first item while it is possible (i.e. the item demand is not satisfied and the object size is greater or equal to the sum of the items sizes in the pattern). After that, the procedure takes the second item, and so on, until the last item is reached. Since the FF solution will depend on the items order, by applying simple permutations of the items, we may produce different solutions. Such strategy will guarantee diversity for initial solutions that will be used by our meta-heuristics. Figure 2 shows how a same instance can generate diverse patterns by just considering different sorting of the items.

If the only algorithm to be used is the constructive heuristic, then it could be better to previously sort the items in decreasing order by their lengths, so the greater items will be selected first for the cut patterns since they are more difficult to combine. Algorithm 1 describes our constructive heuristic.

Algorithm 1 initially creates an empty solution, at line 2, and adds patterns to the solution in the main loop (line 3), until all item demands are satisfied or there are no more objects in stock objects to be cut. At the beginning of each iteration, between lines 4 and 10, the algorithm creates a pattern for each remaining type of object in the stock and, if the created pattern has no leftover, then it is added to the solution removing one object of the current type from stock and updating the unsatisfied demands. If, at the current iteration, no pattern was added between lines 4 and 10, then begins a process to create new patterns between lines 12 and 24. The idea is to remove an item of the first type from the pattern and try to obtain a new pattern by completing the previous one with the FF procedure without considering items of the removed type.

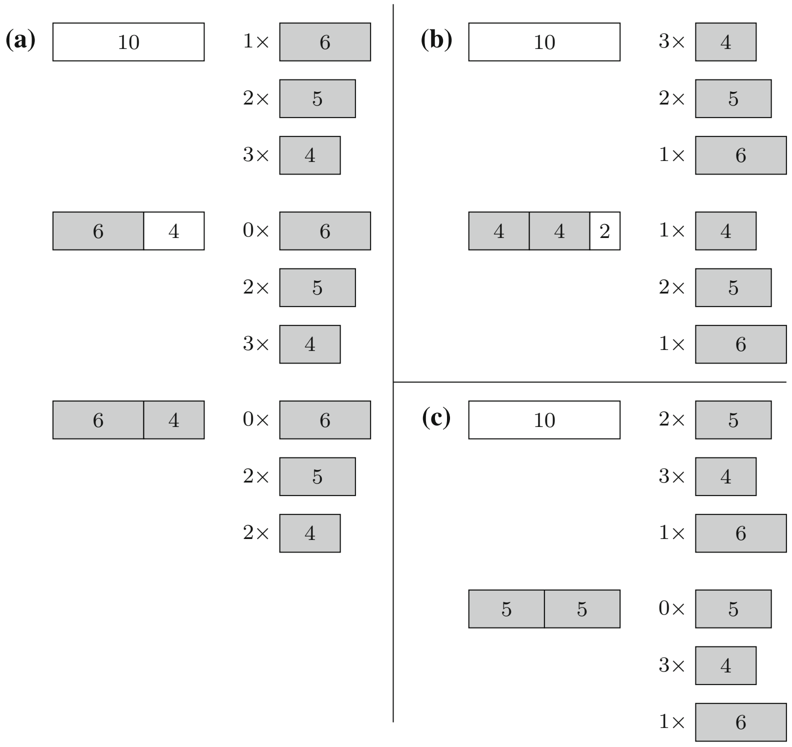


Fig. 2 Example of diverse cut patterns obtained by the FF procedure over different sorting of the items. Given an object of size 10 cm, 3 items of size 4 cm, 2 of size 5 cm and 1 of size 6 cm: In case **a**, the solution considering the items order is 6 cm first, 5 cm second and 4 cm third, producing a cut pattern with 1 item of size 6 cm, 1 of size 4 cm and no leftover. In case **b**, the solution considering the items order is 4 cm first, 5 cm second and 6 cm third, producing a cut pattern with 2 items of size 4 cm and 2 cm of leftover. In case **c**, the solution considering the items order is 5 cm first, 4 cm second and 6 cm third, producing a cut pattern with 2 items of size 5 cm and no leftover

If the resulting pattern is feasible and has no leftover then it is added to the solution updating the remaining objects and demands, otherwise the process is repeated, but removing an item of the second type from the pattern, and so on until all the items originally in the pattern are removed.

In order to guarantee the algorithm halts, one or more patterns must be added to the solution at each iteration. So, if at the current iteration no pattern was added to the solution, then at line 27 the *best* pattern is selected to the solution (among all the previously constructed patterns). In this context, the *best* pattern is the one with least loss among the patterns whose leftover is a little scrap, if no pattern had little scrap then the *best* pattern is the one with smallest retail and if there are no pattern with retails, then the *best* pattern is the one with least leftover.

Notice that our constructive heuristic follows a greedy criteria to select patterns into the solution. It prioritizes patterns with no leftover and only in cases that it is not capable of constructing such patterns, selects patterns with leftovers avoiding at most medium scraps with the hope of obtaining an ideal solution.

```
Input  : Stock and demands
Output: $Sol$: set of patterns to be produced in order to satisfy all demands
begin
    $Sol \leftarrow \{\}$;
    while stock is not empty and all demands are not satisfied do
        for each object type $k$ in stock do
            $p_k \leftarrow$ FF pattern for $k$;
            if $p_k$ has not leftover then
                $Sol \leftarrow Sol \cup \{p_k\}$;
                Update remaining objects in stock and unsatisfied demands;
            end
        end
        if no pattern were added to $Sol$ then
            for each pattern $p_k$ do
                $(\hat{p}_k, \bar{p}_k) \leftarrow (p_k, p_k)$;
                $v \leftarrow$ first item type in $p_k$;
                while $\hat{p}_k$ has items and $p_k$ has leftover do
                    $\hat{p}_k \leftarrow \hat{p}_k - \{$one item of type $v\}$;
                    $\bar{p}_k \leftarrow \hat{p}_k$ completed with FF without items of type $v$;
                    if $p_k$ loss is intermediate scrap or $\hat{p}_k$ loss is not intermediate scrap then $p_k \leftarrow$ pattern with
                    least loss between $p_k$ and $\hat{p}_k$ if $p_k$ loss is intermediary scrap and $\hat{p}_k$ loss is a retail then
                    $p_k \leftarrow \bar{p}_k$ if $v$ is the last item type in $\hat{p}_k$ then $v \leftarrow$ first item type in $\hat{p}_k$ else $v \leftarrow$ next item
                    type in $\hat{p}_k$ if $p_k$ is feasible and has not leftover then
                        $Sol \leftarrow Sol \cup \{p_k\}$;
                        Update remaining objects in stock and unsatisfied demands;
                        break;
                    end
                end
            end
            if no pattern were added to $Sol$ then
                $p_k \leftarrow$ feasible pattern with least loss;
                if $p_k$ loss is not a little scrap and there are feasible patterns with retails then $p_k \leftarrow$ feasible
                pattern with least retail $Sol \leftarrow Sol \cup \{p_k\}$;
                Update remaining objects in stock and unsatisfied demands;
            end
        end
    end
    return $Sol$;
end
```

Algorithm 1: Constructive Heuristic

The following subsections give two meta-heuristics that use our constructive heuristic to generate initial solutions.

3.2 GRASP based algorithm

First we give a meta-heuristic based on the idea of GRASP algorithms. The general form of a GRASP algorithm is described by Algorithm 2.

Since the 1D- CSPUL is a multiobjective problem, we adapted the general form of the GRASP algorithm to consider not a solution but a collection of solutions to represent the Pareto frontier of our problem. Algorithm 3 describes our adaptation.

Notice that Algorithm 3 begins with an empty pool of solutions at line 2 and new solutions are included to the pool between lines 3 and 14, where the main loop of the algorithm is executed. The main loop executes while some *stop conditions are unsatisfied*. Commonly, for these algorithms, stop conditions depend on the maximum number of iterations and, in our case, we applied two stop conditions: maximum

```
1  begin
2      while $stop\ conditions\ are\ unsatisfied$ do
3          $Sol$ ← solution randomly calculated by a greedy criteria;
4          $Sol$ ← local search on $Sol$;
5          if $Sol^*$ was not defined or $Sol$ is better than $Sol^*$ then
6              $Sol^* \leftarrow Sol$
7          end
8      end
9      return $Sol^*$
10 end
```

Algorithm 2: General form of a GRASP algorithm

```
   Input  : Stock and demands.
   Output: $Pool$: set of non-dominated solutions to approximate the Pareto frontier
1  begin
2      $Pool \leftarrow \{\}$;
3      while $stop\ conditions\ are\ unsatisfied$ do
4          $Sol$ ← constructive heuristic over a random permutation of items;
5          while $internal\ stop\ conditions\ are\ unsatisfied$ do
6              if $Pool$ is not full or $Sol$ is better than any solution of $Pool$ then
7                  if $Pool$ is full then
8                      $Pool \leftarrow Pool - \{$worst solution of $Pool\}$;
9                  end
10                 $Pool \leftarrow Pool \cup \{Sol\}$;
11             end
12             $Sol$ ← next neighbor of $Sol$;
13         end
14     end
15     return $Pool - \{$all dominated solutions of $Pool\}$;
16 end
```

Algorithm 3: GRASP based algorithm

number of iterations and maximum number of iterations without updating the pool, so whenever one of them is satisfied the algorithm halts.

Each iteration of the main loop begins at line 4 with a call to the constructive heuristic (Algorithm 1) over a random permutation of the items. As said before, a random permutation of the items will guarantee a different solution of the constructive heuristic each time, which may allow a better exploration of the solution space by our algorithm. Then, the algorithm enters in a local search loop, between lines 5 and 13. The local search loop has also stop conditions and we used similar conditions as the considered for the main loop: maximum number of iterations and maximum number of iterations without updating the pool.

At each iteration of the local search loop, between lines 6 and 11, we analyze if the current solution may be added to the pool, in order to do that we check if the pool has space or is full. If the pool has space, then we just add the solution. Otherwise, if the current solution is *better* than some solution of the pool, then we remove the *worst* solution from the pool before adding the current one. Observe that we need a definition to determine when a solution is better than another.

Definition 4 Given an instance of 1D- CSPUL, a pool of feasible solutions for that instance and two feasible solutions sol_i and sol_j (not necessarily in the pool) for the instance, we say that sol_i **is better** than sol_j if and only if:

- sol_i dominates[1] sol_j, or
- sol_j does not dominates sol_i and:
  - the number of solutions in the pool dominating sol_i is lower than the number of solutions dominating sol_j, or
  - there are the same number of solutions in the pool dominating sol_i and sol_j, and:
    - sol_i is better than sol_j according to definition 3, or
    - sol_j is not better than sol_i according to definition 3 and sol_i dominates more solutions of the pool than sol_j.

Definition 4 depends on the dominance criterion and only when that comparison is not enough to determine if a solution is better than another one, we use the classification given by definition 3. So, in Algorithm 3, we use definition 4 between lines 6 and 11 to decide if the pool must be updated with the current solution and also to find the worst solution of the pool. After that analysis we update the current solution to its next neighbour at line 12. We defined four different movements from a solution to reach a new one and the decision of what movement to make is taken dynamically avoiding to repeat the same movement consecutively. Algorithm 4 shows this movement selection.

```
   Input  : current solution Sol and remaining items in stock.
   Output: Neighbor of solution Sol.
1  begin
2      if Sol has patterns with intermediate scraps and the last selected movement was not move to create better
       patterns then
3          return Move to create better patterns from Sol;
4      end
5      if The number of retails of Sol is greater than desirable value and the last selected movement was not move to
       eliminate retails then
6          return Move to eliminate retails from Sol;
7      end
8      if The number of little scraps of Sol is greater than desirable value and the last selected movement was not
       move to eliminate little scraps then
9          return Move to eliminate little scraps from Sol;
10     end
11     return Move to diversify from Sol;
12 end
```

Algorithm 4: Next neighbor movement selection

Now we explain each one of the four possible movements:

- **Move to create better patterns**: Algorithm 4 selects this movement between lines 2 and 4 only if it was not the last selected movement and if the current solution has patterns with intermediate scraps. The idea is to avoid patterns producing large loss of material, that are not retails. Algorithm 5 gives us this movement.

  Initially, at lines 2 and 3 in the Algorithm 5, we eliminate all the patterns with intermediate scrap from the solution, returning the objects to stock and leaving

[1] We use the standard domination criterion, in which a solution dominates another one if it is not worse in any of the objective functions and also it is better in at least one of the objectives.

```
   Input  : current solution $Sol$ and remaining items in stock.
   Output: $Sol_n$: neighbor of solution $Sol$.
1  begin
2      $Sol_n \leftarrow Sol - \{$all patterns of $Sol$ with intermediate scrap$\}$;
3      Update the unsatisfied demands and return the removed objects to stock ;
4      do
5          $p \leftarrow$ pattern with all the unsatisfied demands;
6          $v \leftarrow$ first item type of $p$;
7          while $p$ is unfeasible for all objects in stock do
8              $p \leftarrow p - \{$one item of type $v\}$;
9              if $v$ is the last item type in $p$ then $v \leftarrow$ first item type in $p$ else $v \leftarrow$ next item type in $p$
10         end
11         if $p$ leftover is intermediate scrap for all objects in stock and $p$ has more than one item then
12             $p \leftarrow$ remove smallest item from $p$;
13         end
14         $k \leftarrow$ smallest object in stock that makes $p$ feasible and, if possible, guarantees $p$ has not intermediate
           scrap;
15         $Sol_n \leftarrow Sol_n \cup \{p_k\}$;
16         Update remaining objects in stock and unsatisfied demands;
17     while stock is not empty and all demands are not satisfied;
18     return $Sol_n$;
19 end
```

Algorithm 5: Move to create better patterns

some items with unsatisfied demands. Then, at line 5 we create an unique pattern with all the items of the removed patterns. Observe that such unique pattern may be unfeasible since it could be larger than any object from stock.
If the pattern is unfeasible (larger than all objects from stock) we drop an item of the first type and if it is still unfeasible, we drop an item of the second type and so on until the pattern becomes feasible or we reach the last type. If we dropped an item of the last type and the pattern remains unfeasible, then we repeat the process from the first type of item. Notice that, eventually the pattern will became feasible, since at each step we drop an item and the items are smaller than the objects from stock. This process takes place between lines 6 and 10.
When we obtain a feasible pattern, if the leftover is intermediate scrap for any object in stock and it has at least two items, then we drop the smallest item from the pattern. Finally, we select the smallest object from stock larger than the pattern (guaranteeing its feasibility) and, if possible, that also gives us a leftover that is not an intermediate scrap. Then, we add the pattern and object to the solution by updating the remaining objects and unsatisfied demands. The hole process is repeated between lines 4 and 17 until all demands are satisfied or there are no objects in stock.

– **Move to eliminate retails** If Algorithm 4 does not select the movement to create better patterns, then it may decide to apply this movement between lines 5 and 7. This movement is applied if it was not the last selected movement and if the number of retails produced by the current solution is greater than *a desirable value*, which is a fraction of the number of cut patterns producing little scraps. The idea is to avoid patterns producing retails. Algorithm 6 shows this movement.

First, at line 2 in the Algorithm 6, we remove from the solution all patterns producing leftover (little or intermediate scraps, or even retails). That operation will

```
Input  : current solution $Sol$ and remaining items in stock.
Output: $Sol_n$: neighbor of solution $Sol$.
1  begin
2      $Sol_n \leftarrow Sol - \{$all patterns of $Sol$ with leftover greater than zero$\}$;
3      Update the unsatisfied demands and return the removed objects to stock ;
4      do
5          for each object type $j$ in stock do
6              $p_j \leftarrow$ solution of knapsack problem over the unsatisfied demands, item lengths and object type $j$;
7          end
8          $p_k \leftarrow$ pattern with least leftover over all the obtained patterns $p_j$;
9          $Sol_n \leftarrow Sol_n \cup \{p_k\}$;
10         Update remaining objects in stock and unsatisfied demands;
11     while stock is not empty and all demands are not satisfied;
12     return $Sol_n$;
13 end
```

Algorithm 6: Move to eliminate retails

return some objects to stock and also some items will have unsatisfied demands. Then, for each object type j in stock, we solve the following knapsack problem between lines 5 and 7:

$$\begin{aligned} \max \quad & \sum_{i=1}^{n} s_i x_i \\ s.t \quad & \sum_{i=1}^{n} s_i x_i \leq d_j \\ & x_i \leq \hat{b}_i, x_i \geq 0, \ x_i \in \mathbb{Z}, \ (1 \leq i \leq n) \end{aligned}$$

where: d_j is the length of the objects of type j in stock, n is the number of items, s_i is the length of item i, and $\hat{b}_i$ is the unsatisfied demand of item i.
Then, at line 8, we select among all the objects the one whose knapsack solution gives us the pattern with least leftover. We add that pattern to the solution at line 9, updating the unsatisfied demands and objects in stock at line 10. This hole process is repeated from line 4 to 11, until the demands are satisfied or the stock gets empty.

– **Move to eliminate little scraps** This movement is selected by Algorithm 4 between lines 8 and 10, if it was not the last selected movement, also if the algorithm did not select one of the previous movements and if the number of cut patterns with little scrap of the current solution is greater than *a desirable value* (a proportion of the number of retails).
The idea is to produce a solution with less loss of material and probably with more retails. Algorithm 7 shows us this movement. Notice that this algorithm is very similar to Algorithm 6 (move to eliminate retails), the only difference is between lines 9 and 11 in which if the selected pattern produces loss of material and has at least two items, then we remove the first item of the pattern in order to produce a retail.

– **Move to diversify** In case none of the above movements were selected by Algorithm 4, then we apply a movement to diversify. This movement tries to generate

```
Input  : current solution $Sol$ and remaining items in stock.
Output: $Sol_n$: neighbor of solution $Sol$.
1  begin
2      $Sol_n \leftarrow Sol - \{$all patterns of $Sol$ with leftover greater than zero$\}$;
3      Update the unsatisfied demands and return the removed objects to stock;
4      do
5          for each object type $j$ in stock do
6              $p_j \leftarrow$ solution of knapsack problem over the unsatisfied demands, item lengths and object type $i$;
7          end
8          $p_k \leftarrow$ pattern with least leftover over all the obtained patterns $p_j$;
9          if no pattern without loss or with retail was added to $Sol_n$ and $p_k$ has at least two items and a leftover
           that is not a retail then
10             $p_k \leftarrow p_k - \{$one item of first type in $p_k\}$
11         end
12         $Sol_n \leftarrow Sol_n \cup \{p_k\}$;
13         Update remaining objects in stock and unsatisfied demands;
14     while stock is not empty and all demands are not satisfied;
15     return $Sol_n$;
16 end
```

Algorithm 7: Move to eliminate little scraps

a new solution by removing all the patterns with leftovers from the current one and solving with the constructive heuristic the generated sub-problem with the non-used objects and the unsatisfied demands. Algorithm 8 give us this idea.

```
Input  : current solution $Sol$ and remaining items in stock.
Output: neighbor of solution $Sol$.
1  begin
2      $Sol_n \leftarrow Sol - \{$all patterns of $Sol$ with leftover greater than zero$\}$;
3      Update the unsatisfied demands and return the removed objects to stock;
4      $Sol_s \leftarrow$ constructive heuristic solution for sub-problem with objects in stock and unsatisfied demands;
5      return $Sol_n \cup Sol_s$;
6  end
```

Algorithm 8: Move to diversify

Observe that our GRASP based algorithm (Algorithm 3) maintains a set with the best solutions found, instead of only one best solution. At the end of the algorithm, at line 15, we remove from that set all the dominated solutions, returning a collection of non-dominated solutions that approximates the Pareto frontier. Since, in multiobjective problems an optimal solution may not exist, it could be interesting to provide some good solutions and from them to decide which is the best strategy to follow.

Next subsection will present our second meta-heuristic, which is based on evolutive algorithms.

3.3 Evolutive based algorithm

As the GRASP based algorithm, our evolutive based algorithm solves the multiobjective problem given as answer a set of non-dominated solutions. This algorithm uses the idea of the general evolutive algorithms, described by Algorithm 9.

```
1  begin
2      $Pool \leftarrow$ randomly generated initial population of solutions;
3      Calculate fitness value of each solution $s \in Pool$;
4      while stop conditions are unsatisfied do
5          $Parents \leftarrow$ subset of $Pool$ with the best fitness values;
6          $New\_Pool \leftarrow$ new solutions obtained through crossover and mutation operations over the elements of
           $Parents$;
7          Calculate fitness value of each solution $s \in New\_Pool$;
8          $Pool \leftarrow$ best $|Pool|$ solutions from $Pool \cup New\_Pool$;
9      end
10     return $Pool$;
11 end
```

Algorithm 9: General form of an evolutive algorithm

Notice that the general evolutive algorithms associate a fitness function to each solution in order to select the parents for the new solutions (at lines 3 and 7 of Algorithm 9). In our case, we do not have a fitness function, instead of that we keep a small population and we apply the crossover operator to all pairs of solutions in the population. After that, we apply a mutation operator to the new solutions and we select the best solutions to the new population. If no new solution is considered to be part of the new population, then we apply a second mutation operator and select one more time the best solutions to create the new population. Algorithm 10 shows this idea.

```
   Input   : Stock and demands.
   Output: Set of non-dominated solutions to approximate the Pareto frontier
1  begin
2      $Pool \leftarrow$ set of solutions generated with the constructive heuristic over random permutations of the items;
3      while stop conditions are unsatisfied do
4          $New\_Sols \leftarrow \{\}$;
5          for each pair of solutions $s_1, s_2 \in Pool$ do
6              $s_{12} \leftarrow$ crossover (combination) of solutions $s_1$ and $s_2$;
7              $s'_{12} \leftarrow$ mutation of $s_{12}$;
8              $New\_Sols \leftarrow New\_Sols \cup \{s'_{12}\}$;
9          end
10         $Pool \leftarrow$ best $|Pool|$ solutions from $Pool \cup New\_Sols$;
11         if no solution from $New\_Sols$ was added in $Pool$ at this iteration then
12             for each solution $s \in Pool$ do
13                 $s' \leftarrow$ second mutation of $s$;
14                 $New\_Sols \leftarrow New\_Sols \cup \{s'\}$;
15             end
16             $Pool \leftarrow$ best $|Pool|$ solutions from $Pool \cup New\_Sols$;
17         end
18     end
19     return $Pool - \{$dominated solutions of $Pool\}$;
20 end
```

Algorithm 10: Evolutive based algorithm

At the beginning of Algorithm 10 we populate our pool of solutions using the constructive heuristic over different random permutations of the items (line 2). After that the algorithm enters into its main loop between lines 3 and 18, until *stop conditions are unsatisfied*. The stop conditions we used were maximum number of iterations or not updating the pool at current iteration.

As said before, we maintain a small population and apply the crossover over all the pairs of solutions in the population. That operation occurs between lines 4 and 9. The crossover between two solutions is given by Algorithm 11. Observe that it begins at line 2 with an empty new solution. Then, between lines 5 and 8, the algorithm takes, simultaneously, patterns from each solution to the new one, with one condition: the pattern to be taken must be a possible pattern (cannot make unfeasible the constructed solution) and also must not have leftover. The constructed solution is feasible if and only if it does not use more objects of any type than the available in stock and it does not cut more items of any type than its required demand. The algorithm stops when all the patterns of the parent solutions are analyzed, returning the new solution.

```
Input : parent solutions $s_1$ and $s_2$, stock and demands.
Output: $s_{12}$: partial solution constructed from $s_1$ and $s_2$
1  begin
2      $s_{12} \leftarrow \{\}$;
3      $p_1 \leftarrow$ first pattern of $s_1$;
4      $p_2 \leftarrow$ first pattern of $s_2$;
5      do
6          if $p_1$ is not empty and $p_1$ has no leftover and $(s_{12} \cup \{p_1\})$ is feasible then $s_{12} \leftarrow s_{12} \cup \{p_1\}$ if $p_2$ is not
           empty and $p_2$ has no leftover and $(s_{12} \cup \{p_2\})$ is feasible then $s_{12} \leftarrow s_{12} \cup \{p_2\}$ $p_1 \leftarrow$ next pattern of $s_1$
           or empty if $s_1$ has no more patterns;
7          $p_2 \leftarrow$ next pattern of $s_2$ or empty if $s_2$ has no more patterns;
8      while $p_1$ or $p_2$ are not empty;
9      return $s_{12}$;
10 end
```

Algorithm 11: Crossover operator

Observe that the new solution may be partial (incomplete), since there could be some items with unsatisfied demands. So, at line 7 of Algorithm 10, we apply to the solution a mutation operator. That operator, given by Algorithm 12, solves the subproblem with the unsatisfied item demands and the unused objects applying the same knapsack problem used by the GRASP based algorithm. Then, from the knapsack solution, we select the patterns that minimize the leftover, trying to avoid those with leftover of intermediate size.

After obtaining all the new solutions, Algorithm 10 selects at line 10 the best solutions to the next population. If there are no new solutions added to the next population, then between lines 11 and 17, it is applied a second mutation operator to all the solutions in the population trying to obtain new solutions. The best solutions from the actual population and the mutated solutions are selected to the next population.

The second mutation operator follows the idea of assigning to each pattern a rank value, based on the number of large items they produce (where the minimum length of a large item is calculated as a fraction of the length of the smallest standard object, used 0.25). It creates a mutated partial solution, only with the patterns that have no leftover and positive rank. The objective is to have a greater number of free small items, that are easier to combine. Then, the first mutation operator is applied to the new partial solution. Algorithm 13 shows this idea.

```
Input  : partial solution $Sol$, unsatisfied demands and remaining items in stock.
Output: $Sol$: complete solution.
begin
    do
        for each object type $j$ in stock do
            $p_j \leftarrow$ solution of knapsack problem over the unsatisfied demands, item lengths and object type $j$;
        end
        $p_k \leftarrow$ pattern with least leftover over all the obtained patterns $p_j$;
        if $p_k$ has at least two items and a leftover that is intermediate scrap then
            $p_k \leftarrow p_k -$ {one item of first type in $p_k$}
        end
        $Sol \leftarrow Sol \cup \{p_k\}$;
        Update remaining objects in stock and unsatisfied demands;
    while stock is not empty and all demands are not satisfied;
    return $Sol$;
end
```

Algorithm 12: First mutation operator

```
Input  : current $Sol$ and remaining items in stock.
Output: mutated solution.
begin
    $Sol_m \leftarrow Sol$;
    for each pattern $p \in Sol$ do
        $rank_p \leftarrow$ number of large items produced by $p$;
        if $p$ has leftover or $rank_p = 0$ then
            $Sol_m \leftarrow Sol_m - \{p\}$;
            Update unsatisfied demands and objects in stock;
        end
    end
    $Sol_m \leftarrow$ first mutation operator over $Sol_m$, the unsatisfied demands and the remaining objects in stock;
    return $Sol_m$;
end
```

Algorithm 13: Second mutation operator

4 Computational experiments

In this section we present tests with instances used by Cherri et al. (2009, 2013); Cui and Yang (2010); Tomat and Gradišar (2017). Some of the instances used in Cherri et al. (2009); Cui and Yang (2010) were taken from Gradišar and Trkman (2005) (the numerical instances), other were taken from Abuabara and Morabito (2009) (the practical instances), and the rest were randomly generated by Cherri et al. (2009). The instances from Cherri et al. (2013); Tomat and Gradišar (2017) where also randomly generated but they consider demands in different time periods of a time horizon (the multi-period instances). We compare our solutions with the solutions from Cherri et al. (2009), following the criterion of a good solution given by definition 3, being, the values used in Cherri et al. (2009): $\epsilon_1 = 0.03$, $\epsilon_2 = 0.1$, $\beta = 0.005$, $\theta = 0.05$ and $\delta = \min s_i$, where s_i is the length of the item i. For both of our meta-heuristics, the seeds used were the integers from 1 to 10, the maximum number of iterations was set to 100 and the maximum number of iterations without updating the pool was set to 10, being the pool maximum size also 10.

Our implementations were executed with Cygwin 2.10.0-1 on a processor Intel(R) Core(TM) i5-7400 CPU @ 3.00GHz and 8GB of RAM, under Windows 10 Home Single Language operating system.

Table 1 Items' lengths and demands of instance 1

Item	1	2	3	4	5
Length (cm)	235	200	347	471	274
Demand	4	51	42	16	37

The following subsections discuss the instances and show the obtained results. The results are given in tables in which are used `O.Cut` to denote the number of cut objects, `T.Len` to denote the total length of the cut objects, `T.Loss` to denote the total length of the leftovers, `T.Ret` to denote the total length of the retails generated by the solution, `LScrap` to denote the number of cuts with little scrap, `MScrap` to denote the number of cuts with leftovers of intermediate size, `NRet` to denote the number of cuts with retails, and `Sol` to denote the classification of the solution according to definition 3, where `ID` is *ideal*, `AC` is *acceptable*, and `UND` is *undesirable*. To refer our algorithms we use the terms $C.H.$ to the constructive heuristic, $GRASP.B$ to the GRASP based algorithm, and $Evol.B$ to the evolutive based algorithm.

4.1 Numerical instances

These instances were originally given by Gradišar and Trkman (2005). Since the best known results for them were given by Cherri et al. (2009), we compare our approaches with their best algorithms, which are: RGR_L1, RGR_L2 and RGR_L3. In the comparison, we selected some of the non-dominated solutions found by our meta-heuristics over each seed.

Instance 1 has 20 types of different objects with lengths between 2200 cm and 6000 cm. The stock contains exactly one object of each type. Table 1 gives us the items' lengths and demands.

Table 2 shows a resume of the results obtained by our approaches and those from Cherri et al. (2009) over the first numerical instance. From that table, one can see that our GRASP and Evolutive based algorithms give solutions with more cuts than the obtained by RGR_L1, RGR_L2 and RGR_L3, being shorter the total length of the cuts given by our algorithms. This means that our algorithms preferred to cut shorter objects first, saving the larger objects (easier to use in cut patterns) for future cuts. Also, our GRASP algorithm was capable of producing two Pareto optimal solutions, being one of them very different from the solutions given by the other algorithms since it does not produce any retail to return to stock, but a very small loss of material (12 cm of loss compared to the 43400 cm of cuts).

Instance 2 has 20 types of different objects with lengths between 2100 cm and 5000 cm. The stock contains exactly one object of each type. Table 3 gives us the items' lengths and demands.

Table 4 shows a resume of the results obtained over the second numerical instance. Observe that our GRASP and Evolutive based algorithms give, each one, a set of different solutions in which we can find none or just one retail to stock, or with no loss of material or very small loss. For this instance, some of the solutions found by

Table 2 Solutions for instance 1

	$RGR_L(1,2)$	RGR_L3	C.H.	GRASP.B		Evol.B
O.Cut	**10**	**10**	13	11	11	11
T.Len	45245	46507	45800	**43400**	44000	44800
T.Loss	**0**	**0**	34	12	**0**	**0**
T.Ret	1857	3119	2378	**0**	612	1412
LScrap	**0**	**0**	6	1	**0**	**0**
MScrap	**0**	**0**	**0**	**0**	**0**	**0**
NRet	1	2	2	**0**	1	1
Sol	**ID***	AC	AC	**ID***	**ID***	**ID***

The * indicates that the solution is a Pareto-optimal
The best values are highlighted in bold

Table 3 Items' lengths and demands of instance 2

Item	1	2	3	4	5
Length (cm)	549	433	207	308	583
Demand	39	27	43	30	2

Table 4 Solutions for instance 2

	RGR_L1	RGR_L2	RGR_L3	C.H.	GRASP.B		Evol.B		
O.Cut	15	16	**14**	16	15	16	16	15	15
T.Len	57554	58159	56395	55944	**55212**	58832	56856	57036	58100
T.Loss	6	3	7	93	31	13	18	7	**0**
T.Ret	2367	2972	1207	670	**0**	3638	1657	1848	2919
LScrap	4	4	5	12	7	5	6	5	**0**
MScrap	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
NRet	2	3	2	2	**0**	2	1	2	7
Sol	**AC**	UND	**AC**	**AC**	**AC**	**AC**	**AC**	**AC**	UND

The best values are highlighted in bold

our approaches were dominated by the ones of RGR_L1, RGR_L2 and RGR_L3, even so, both of our meta-heuristics were capable of finding new non-dominated solutions.

Instance 3 has 90 types of different objects with lengths between 3000 cm and 9000 cm. The stock contains exactly one object of each type. Table 5 gives us the items' lengths and demands, being a total of 15 different items.

Table 6 shows a resume of the results obtained over the third numerical instance. Notice that for this instance two of the algorithms from Cherri et al. (2009) reached the same solution, while our approaches were capable of finding similar solutions to the ones given by Cherri et al. (2009) and also another good non-dominated solutions. Analyzing the number of cuts, we observe that, once more, our algorithms had more cuts but less length of the cut material than RGR_L1, RGR_L2 and RGR_L3. That show us the preference our approaches give to cut first little objects saving the larger objects to future cut patterns, which could be a nice strategy since, generally, larger objects are easier to use in the cuts.

Table 5 Items' lengths and demands of instance 3

Item	1	2	3	4	5	6	7	8
Length (cm)	569	718	520	540	492	547	632	430
Demand	34	26	25	12	30	2	6	36
Item	9	10	11	12	13	14	15	
Length (cm)	750	387	804	389	835	684	687	
Demand	7	20	3	32	18	39	10	

Table 6 Solution for instance 3

	$RGR_L(1,2)$	RGR_L3	C.H.	GRASP.B		Evol.B	
O.Cut	**22**	**22**	27	28	26	39	27
T.Len	172114	170989	169060	169544	**169052**	169700	169060
T.Loss	**0**	**0**	14	**0**	6	**0**	14
T.Ret	3098	1943	**0**	498	**0**	654	**0**
LScrap	**0**	**0**	2	**0**	2	**0**	2
MScrap	**0**	**0**	**0**	**0**	**0**	**0**	**0**
NRet	1	1	**0**	1	**0**	1	**0**
Sol	**ID**	**ID**	**ID**	**ID**	**ID**	**ID**	**ID**

The best values are highlighted in bold

Table 7 Items' lengths and demands of instance 4

Item	1	2	3	4	5
Length (cm)	250	273	285	525	1380
Demand	2	2	4	4	4

4.2 Practical instances

These instances were originally given by Abuabara and Morabito (2009) and the only heuristic results for them were given by Cherri et al. (2009), so we compare our approaches with their algorithms. Since, for these instances the number of objects to be cut is very small, the values of ϵ_1 and ϵ_2 are not appropriate, that is the reason we will not classify them by definition 3.

Instance 4 has only one type of object with length 3000 cm and availability of 10 in stock. Table 7 gives us the items' lengths and demands.

Table 8 shows a resume of the results obtained over the first practical instance (instance 4). Observe that two of the algorithms proposed by Cherri et al. (2009) found the same solution and none of the solutions from Cherri et al. (2009) are Pareto optimal, while both of our meta-heuristics found Pareto optimal solutions, even more, for that instance our evolutive based algorithm was able to find the complete Pareto frontier.

Instance 5 has only one type of object with length 6000 cm and availability of 10 in stock. Table 9 gives us the items lengths and demands.

Table 8 Solution for instance 4

	$RGR_L(1,2)$	RGR_L3	C.H.	GRASP.B			Evol.B	
O.Cut	5	5	5	5	**4**	**4**	**4**	**4**
T.Len	15000	15000	15000	15000	**12000**	**12000**	**12000**	**12000**
T.Loss	**0**	4	45	**0**	4	240	**0**	240
T.Ret	5194	5190	5149	5194	2190	**1954**	2194	**1954**
LScrap	**0**	1	**0**	**0**	1	**0**	**0**	**0**
MScrap	**0**	**0**	1	**0**	**0**	1	**0**	1
NRet	3	3	4	5	2	**1**	2	**1**
Sol	–	–	–	–	–	*	*	*

The * indicates that the solution is a Pareto-optimal
The best values are highlighted in bold

Table 9 Items' lengths and demands of instance 5

Item	1	2	3	4
Length (cm)	370	905	910	930
Demand	5	5	5	5

Table 10 Solution for instance 5. The * indicates that the solution is a Pareto-optimal

	$RGR_L(1,2,3)$	C.H.	GRASP.B		Evol.B	
O.Cut	**3**	**3**	**3**	**3**	**3**	**3**
T.Len	**18000**	**18000**	**18000**	**18000**	**18000**	**18000**
T.Loss	150	**0**	**0**	250	**0**	250
T.Ret	2275	2425	2425	**2175**	2425	**2175**
LScrap	**0**	**0**	**0**	**0**	**0**	**0**
MScrap	1	**0**	**0**	2	**0**	2
NRet	2	3	3	**1**	3	**1**
Sol	–	*	*	*	*	*

The best values are highlighted in bold

Table 10 shows a resume of the results obtained over the second practical instance (instance 5). For this instance the three algorithms from Cherri et al. (2009) gave the same answer which was not a Pareto optimal solution, while all of our algorithms found Pareto optimal solutions and our both meta-heuristics gave two different optimal solutions to describe the Pareto frontier.

4.3 Randomly generated instances

These instances were first given by Cherri et al. (2009) and tested with their heuristics approaches. Also, in Cui and Yang (2010) these instances were tested and the results from Cherri et al. (2009) improved. We compare our results for these instances with both approaches, the algorithms from Cherri et al. (2009) (RGR_L1, RGR_L2 and RGR_L3) and the heuristic from Cui and Yang (2010) ($RSHP$).

Table 11 Random instances description

Class	Obj. type	Itm. type	Items	Class	Obj. type	Itm. type	Items
1	5	10	Small	9	7	20	Small
2	5	10	Average	10	7	20	Average
3	5	20	Small	11	7	40	Small
4	5	20	Average	12	7	40	Average
5	5	40	Small	13	9	10	Small
6	5	40	Average	14	9	10	Average
7	7	10	Small	15	9	20	Small
8	7	10	Average	16	9	20	Average

These instances were divided in 16 classes with 20 instances each class. The instances were generated by Cherri et al. (2009) considering some parameters for each class. One of those parameters was the number of different types of object in stock, which can be 5, 7 or 9, where they always had 2 types of standard objects of sizes 1000 and 1200, being the rest of the types non-standard with random lengths lower than 1000. Another parameter was the number of different types of items to produce, where there could be 10, 20 or 40 different types of items. Finally, a parameter was considered to bound the sizes of the different items, where they could be of *small* size (up to 0.2 times the average of standard object sizes: 1100) or they could be *average* size (up to 0.8 times the average of standard object sizes: 1100). Table 11 describes, for each class, how these parameters were given.

Since the comparison with other approaches is made over average values and our meta-heuristics give a set of undominated solutions, our average value considers only one of the solutions of each set. Notice that, in general, an average value over set of Pareto optimal will give a *very bad* solution value for each objective function, for that reason we need a criterion to select one of the non-dominated solutions we found for the average comparison. Then, for each instance and each seed, we only considered to the average computation the solution of the pool with less loss of material. Another possible criteria to consider for the average value could be: the solution with least retails or the solution with best classification by definition 3, or even the solution that found more dominated solutions during the algorithm execution.

Table 12 compares our solutions with the ones obtained by Cherri et al. (2009). As seen on the previous instances our approaches cut more objects but with less total length, implying the preference of our algorithms for cutting smaller objects, saving the larger for future cuts. Such preference allowed all of our algorithms to cut much more non-standard objects (retails from previous cuts) than RGR_L1, RGR_L2 and RGR_L3 (see `NonStand` row of Table 12). So, even when the algorithms from Cherri et al. (2009) produce less retails, at the end of the process the total number of retails in stock given by our approaches is lower than the achieved by Cherri et al. (2009). Then, since our evolutive based algorithm produces less loss of material than RGR_L1, RGR_L2 and RGR_L3, in general their solutions are dominated by ours, while neither our constructive heuristic or our GRASP based algorithm are dominated by any other

Table 12 Comparison with the results from Cherri et al. (2009)

	RGR_L1	RGR_L2	RGR_L3	C.H.	GRASP.B	Evol.B
O.Cut	**106.9**	**106.9**	**106.9**	127.7	125	119.7
Stand	102.6	102.5	102.6	**100.8**	103.7	107.3
NonStand	4.3	4.4	4.5	**26.9**	21.3	12.4
T.Len	111957.6	**111923.5**	111979.4	112342.7	112979.7	113319
T.Loss	11.5	11.8	12.5	168.6	15.5	**0**
T.Ret	587.4	**552.8**	601.9	815.3	1605	1960.2
LScrap	3.2	3.2	3.6	32	2.8	**0**
MScrap	0.1	0.1	0.1	5	0.6	**0**
NRet	**2.6**	**2.6**	2.8	2.9	6.2	8
Final Ret	26.4	26.3	26.4	**4.1**	13	23.7
Sol	**ID**	**ID**	**ID**	AC	AC	AC
Time (s)	22.55	22.74	81.85	$<$ **0.001**	0.21	1.14

The best values are highlighted in bold

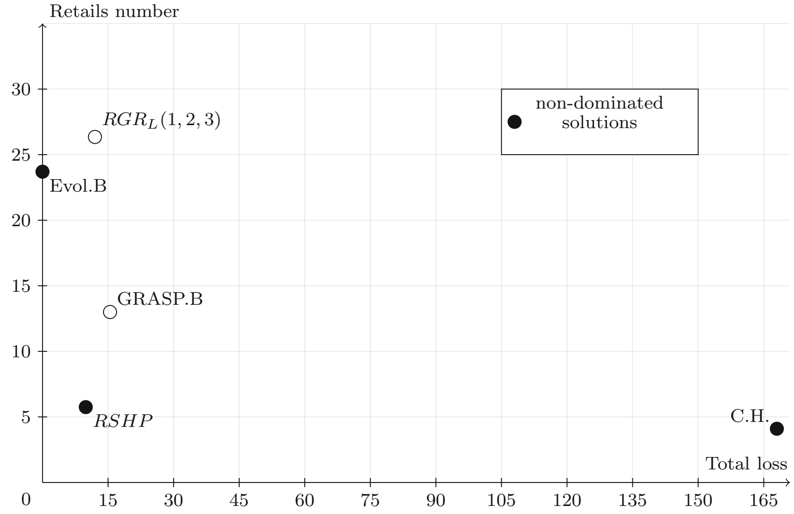
Table 13 Comparison with the results from Cherri et al. (2009) and Cui and Yang (2010)

	RGR_L1	RGR_L2	RGR_L3	$RSHP$	C.H.	GRASP.B	Evol.B
Stand	102.6	102.5	102.6	101.5	**100.8**	103.7	107.3
NonStand	4.3	4.4	4.5	23.6	**26.9**	21.3	12.4
T.Len	111957.6	111923.5	111979.4	**111672**	112342.7	112979.7	113319
T.Loss	11.5	11.8	12.5	9.8	168.6	15.5	**0**
T.Ret	587.4	552.8	601.9	**303.2**	815.3	1605	1960.2
NRet	2.6	2.6	2.8	**1.2**	2.9	6.2	8
Av.Ret.Len	225.9	212.6	215	252.7	**281.1**	258.9	245
Final Ret	26.4	26.3	26.4	5.7	**4.1**	13	23.7
Time (s)	22.55	22.74	81.85	1.56	$<$ **0.001**	0.21	1.14

The best values are highlighted in bold

solution of Table 12. This proves once more that solutions classified as ideals by definition 3 are not necessarily better, but could be worse, than solutions classified as acceptables. Beside that, our algorithms seem to be very efficient, as it can be seen at row `Time (s)` of Table 12.

Table 13 compares our solutions with the ones obtained by Cherri et al. (2009) and Cui and Yang (2010). Analyzing the objective values one can see that the lower loss of material is given by our evolutive based algorithm with an average value of 0 (no loss of material) and even without loosing material this meta-heuristic was able to reduce the number of retails in stock. The next approach with less material loss is the $RSHP$ given by Cui and Yang (2010), which is also the second one with less retails in stock at the end of the process. The RGR_L1, RGR_L2, RGR_L3 and GRASP based algorithm had slightly greater loss of material than $RHSP$, while the loss of material from the constructive heuristic was significantly greater. On the other hand, the constructive



RGR_L1 (11.5, 26.4)	C.H. (168.6, 4.1)	$RSHP$ (9.2, 5.7)
RGR_L2 (11.8, 26.3)	GRASP.B (15.5, 13)	
RGR_L3 (12.5, 26.4)	Evol.B (0, 23.7)	

Fig. 3 The values associated to RGR_L1, RGR_L2 and RGR_L3 are dominated by the ones associated to Evol.B and $RSHP$, the value of GRASP.B is only dominated by the one of $RSHP$, while the values of C.H., $RSHP$ and Evol. BA are non-dominated

heuristic had the smallest number of retails at the end in the stock, while RGR_L1, RGR_L2 and RGR_L3 were the worst algorithms to achieve this objective, however these algorithms were designed to reduce the number of produced retails instead of the retails at the final stock (see definition 1) and, for that objective, they achieved good results loosing only for $RSHP$. We also analyze the average size of the produced retails and RGR_L1, RGR_L2 and RGR_L3 produced the shorter retails, which could be bad for future cuts, while the constructive heuristic gave us largest average sized retails.

In order to visualize the Pareto dominance of the different heuristics over the random generated instances, one can see the Fig. 3. Despite the solutions of our proposals do not dominate the solutions of $RHSP$, given by Cui and Yang (2010), we observe that the average loss of material of our evolutive based algorithm was 0, which is more desirable than the loss of material produced by $RHSP$ and since our evolutive based algorithm also reduces the number of retails in stock, one can suppose that over a time horizon with different demands, this meta-heuristic will maintain a small number of retails in stock. So, the solutions achieved by our evolutive based algorithm could be more interesting for applications with expensive materials such as photographic paper, where the cost of the waste may be more significant than the cost of the stock. Another applications where our evolutive based algorithm could be better than the $RHSP$, because it is more important to reduce the waste than the stock, are those with high difficulty on discarding the losses, since the non-used material may cause environmental pollution or may be hard to recycle, an example occurs at the process of cutting construction materials like stone labs Tam et al. (2006).

4.4 Multi-period instances from Cherri et al. (2013)

The instances given by Cherri et al. (2013) were divided in two sets. Each set contains twenty 12-period sequences (i.e. 20 simulations, each one with 12 different orders). For each sequence, the first period starts with the stock containing only standard objects (without retails) and, at further periods, the non-standard objects are the retails generated in previous periods which were not used in the solutions. The standard objects for all the tests are of two lengths 1000 and 1100 with unlimited availability in the stock in all the periods. The demands consist on generate items from 50 different lengths in the interval [10.5, 262.5] (*small* items) for the set 1 and in the interval [10.5, 420] (*mixed* items) for the set 2. For each set, the first ten items from the 50 types are considered the regular items, whose average value determine the minimum retail length. At each period, the orders are compound by selecting regular items demands in the interval [200, 500] and also by selecting between 10 and 30 of the other items with demands in the interval [1, 10].

Since in a multi-period simulation the number of non-standard objects at some period are the non used retails of previous periods, we must select one of the non-dominated solutions given by our meta-heuristics at each period, to uniquely determine the stock of the next period. The authors of Cherri et al. (2013) gave more relevance to reduce the number of retails in the stock than to minimize the loss of material, for that reason, at each period, each meta-heuristic selected the solution with less retail at the stock, then updated the stock and tested the next period. Figures 4 and 5 show for sets 1 and 2, respectively, the average value for the cumulative loss of material produced by our algorithms along the periods and the average number of retails in stock per period. From those figures we observe that the lowest loss of material for both sets per period was given by the $GRASP.B$, followed by the $Evol.B$. Also, the $GRASP.B$ algorithm managed to not generate retails for instances of set 1 and the generated retails per period for instances of set 2 was almost zero. The $Evol.B$ algorithm was the second with lowest retails generation, whose number was almost constant for both sets at each period. On the other hand, the $C.H.$ presented the worst solutions, generating more loss of material and more retails at each period.

In Cherri et al. (2013) the authors proposed the RGR_L^p heuristic which modifies their previous heuristic RGR_L (RGR_L2 in Cherri et al. (2009)) and compared the results obtained with the heuristic RGR of Poldi and Arenales (2009). From their analysis there was not clear which heuristic produced the best results, so we compare the three heuristics with our own solutions.

Table 14 compares the average results for each set. The parameters we considered were the total length of the loss material (`T.Loss`) and its percentage over the total length of the object cuts (`T.Loss %`), the number of retails at the stock after the final period (`Final Ret`) and a parameter to indicate if the solution is dominated or not (`Dominated`). For the set 1 the undominated solutions were given by $GRASP.B$ and RGR_L^p, were RGR_L^p only dominated the solutions from RGR_L and $C.H.$ while

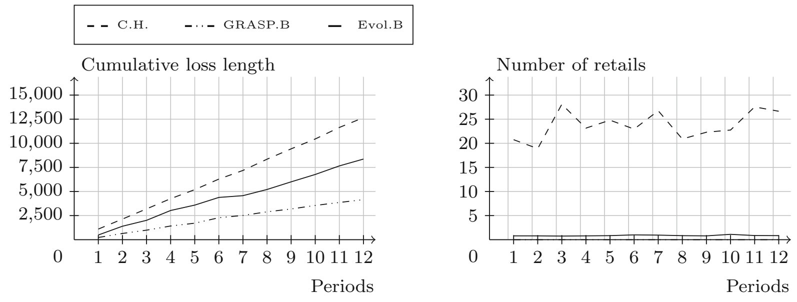


Fig. 4 Average values for cumulative loss of material and total number of retails per period for set 1 from Cherri et al. (2013)

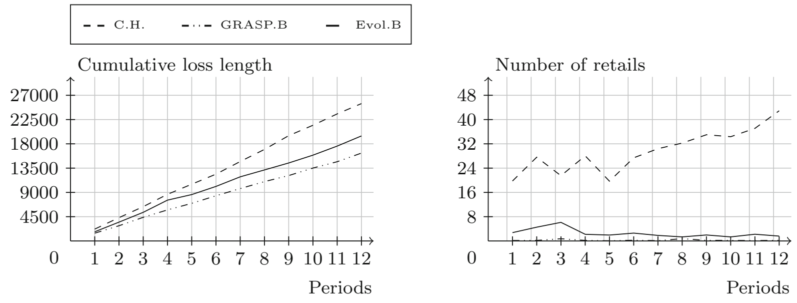


Fig. 5 Average values for cumulative loss of material and total number of retails per period for set 2 from Cherri et al. (2013)

$GRASP.B$ dominated the solutions from RGR, $C.H.$ and $Evol.B$. Even more, the percentage of loss material produced by $GRASP.B$ and RGR_L^p were very close, while our $GRASP.B$ managed to produce zero retails. Analyzing the results for the set 2, we observe that the solutions given by the $GRASP.B$ were much better in both objectives than the solutions produced by the rest of the algorithms. In general we may conclude that, for these instances, our $GRASP.B$ algorithm obtained the best solutions, reducing significantly the number of retails at the final stock.

4.5 Multi-period instances from Tomat and Gradišar (2017)

In Tomat and Gradišar (2017), the authors tested four sequences of instances, each sequence consisting of 30-period orders. All the periods began with two types of standard objects: 100 objects of size 1000 and 100 objects of size 1100. The first period of each sequence only had standard objects at the stock and, at each further period, the non-standard objects were the non used retails generated in previous periods. The demands were randomly generated for each period with at most 20 different items. From sequence 1 to 4 the ratio between the average object length and items length was decreasing, implying that sequence 4 had greater items and the solutions should be worst (more retails and loss of material). Table 15 shows the interval in which the

Table 14 Comparison with the solutions of Cherri et al. (2013)

Set 1	RGR	RGR_L	RGR_L^p	C.H.	GRASP.B	Evol.B
T.Loss	5572	428	**378**	12658	4154	8364
T.Loss %	0.09	**0.01**	**0.01**	0.22	0.07	0.13
Final Ret	1.0	2.4	1.1	26.7	**0**	0.8
Dominated	Yes	Yes	**No**	Yes	**No**	Yes
Set 2	RGR	RGR_L	RGR_L^p	C.H.	GRASP.B	Evol.B
T.Loss	43256	30684	30902	25462.8	**16295.1**	19472.5
T.Loss %	0.5	0.36	0.36	0.35	**0.23**	0.27
Final Ret	0.8	6.9	1.2	42.9	**0.2**	1.6
Dominated	Yes	Yes	Yes	Yes	**No**	Yes

The best values are highlighted in bold

Table 15 Parameters for order generation by Tomat and Gradišar (2017)

	Sequence 1	Sequence 2	Sequence 3	Sequence 4
Items length interval	[5, 83]	[6, 146]	[8, 209]	[11, 335]
Number of pieces	125	102	79	34

lengths of the items were generated for the demands of each sequence and also the number of pieces to be cut.

Since at each period the non-standard objects in stock depends on the solution of the previous period, in order to test the simulation we needed to select only one solution of the undominated solutions set given by our meta-heuristics. Then, for each sequence, we solved each period instance with our algorithms and, for the meta-heuristics, we selected the solution with less loss of material among the undominated set of solutions they obtain. Then, we updated the stock with the generated retails of the selected solution and tested the next period.

Table 16 compares the solutions we obtained with the ones given by $COLA$ (the heuristic algorithm used by Tomat and Gradišar (2017) to test the instances). That table considers the total length of the material loss for each sequence (`T.Loss`), the final number of retails at the stock after the last period (`Final Ret`) and a parameter to indicate if the solution is dominated or not (`Dominated`). For all the sequences, the $GRASP.B$ obtained undominated solutions, always dominating the solutions given by $COLA$. Also, along the tests, the $Evol.B$ did not produced loss of material, however in two of the sequences the final number of retails at the final stock was very high.

In order to analyse better the simulations, we provide graphics for each sequence considering the cumulative loss over the periods and also the number of retails at each period. The results for sequence 1 simulation are shown in Fig. 6. For that instance, neither $GRASP.B$ nor $Evol.B$ had loss of material at any period and $C.H.$ produced loss of material only at the initial periods, while $COLA$ had loss of material in almost all the periods producing the most loss of material among the four algorithms. Analyzing the retails number, the four algorithms had only one retail at the final stock, but while the number of retails of $GRASP.B$ remained constant for the complete simula-

Table 16 Comparison with the solutions of COLA

Sequence 1	COLA	C.H.	GRASP.B	Evol.B
T.Loss	90	31	**0**	**0**
Final Ret	**1**	**1**	**1**	**1**
Dominated	Yes	Yes	**No**	**No**
Sequence 2	COLA	C.H.	GRASP.B	Evol.B
T.Loss	1269	382	**0**	**0**
Final Ret	6	2	**1**	11
Dominated	Yes	Yes	**No**	Yes
Sequence 3	COLA	C.H.	GRASP.B	Evol.B
T.Loss	4561	1176	**0**	**0**
Final Ret	145	3	**1**	45
Dominated	Yes	Yes	**No**	Yes
Sequence 4	COLA	C.H.	GRASP.B	Evol.B
T.Loss	7999	2899	38.9	**0**
Final Ret	210	**3**	14.6	280
Dominated	Yes	**No**	**No**	**No**

The best values are highlighted in bold

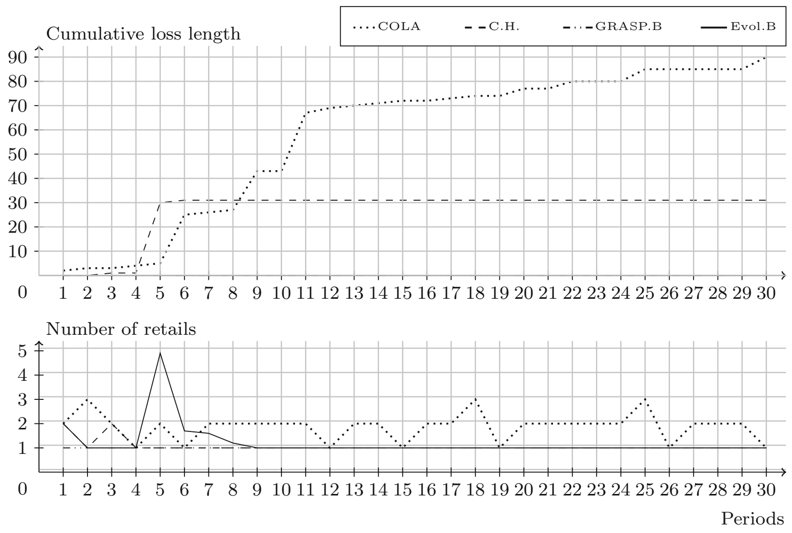


Fig. 6 Cumulative loss of material and total number of retails per period for sequence 1 from Tomat and Gradišar (2017)

tion, we observe that $C.H.$ and $Evol.B$ had some fluctuations only at the beginning of the process, and the number of retails of $COLA$ presented more perturbations, which could be undesirable if one wants to predict the behavior of the algorithm.

Figure 7 shows the results for the simulation of sequence 2. The loss of material analysis for this instance is very similar to the previous one (sequence 1): neither $GRASP.B$ nor $Evol.B$ had loss of material at any period and $C.H.$ produced loss of material only at the first periods, while COLA had loss of material in every period pro-

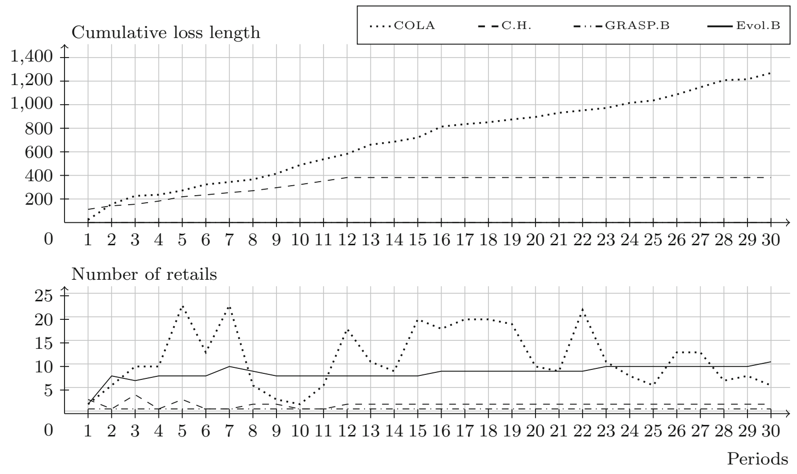


Fig. 7 Cumulative loss of material and total number of retails per period for sequence 2 from Tomat and Gradišar (2017)

ducing the most loss of material among the four algorithms. The retails analysis shows different results: at the final stock $Evol.B$ produced the greater number of retails, followed by $COLA$, while $GRASP.B$ and $C.H.$ produced the less retails. Despite the number of retails at the final stock from $Evol.B$ is greater than the produced by $COLA$, we observe that along the simulation, $Evol.B$ maintained an almost constant number of retails while $COLA$ presented enormous fluctuations on that number. Such fluctuations are not desirable because if the simulation would ended at a previous period, for example the 22^{th}, then $COLA$ would had produced the greatest number of retails and also the most loss of material.

The results of sequence 3 simulation are given by Fig. 8. For that instances, the $Evol.B$ and $GRASP.B$ produced the lowest loss material values, which were 0 at each period. Also, the $C.H.$ produced a small loss compared to the produced by $COLA$. Similarly to the previous simulations, the lowest average retail number per period was given by $GRASP.B$ which was one, consuming and producing one retail at each period and the $C.H.$ maintained for almost all the periods the constant number of one retail at the stock. Despite $Evol.B$ presented an increasing number of retails at the stock, such increment follows almost a linear proportion over the periods and is much smaller than the number of retails produced by $COLA$, which presents big fluctuations along the simulation.

Figure 9 gives the simulation results for the sequence 4. Similar to the previous sequences, the $Evol.B$ did not produced any loss of material, being the second best results from $GRASP.B$, whose loss of material is less than 1.4% of the material loss from $C.H.$ and less than 0.5% of the material loss from $COLA$. For this instance the $C.H.$ presented the lowest numbers of retails at the stock, maintaining the constant value of three retails in almost all the periods. The second algorithm with lowest retail number was the $GRASP.B$, whose number of retails was also almost constant (10) at each period. Despite the $Evol.B$ presented the greatest number of retails at the final

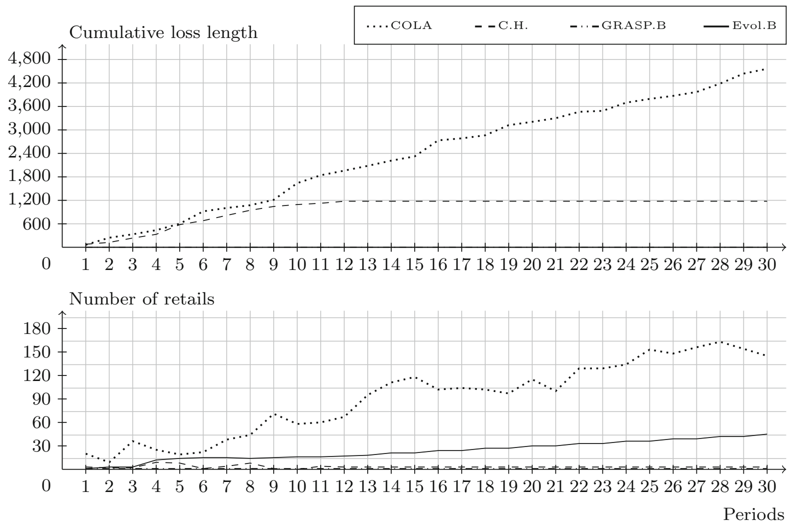


Fig. 8 Cumulative loss of material and total number of retails per period for instance 3 from Tomat and Gradišar (2017)

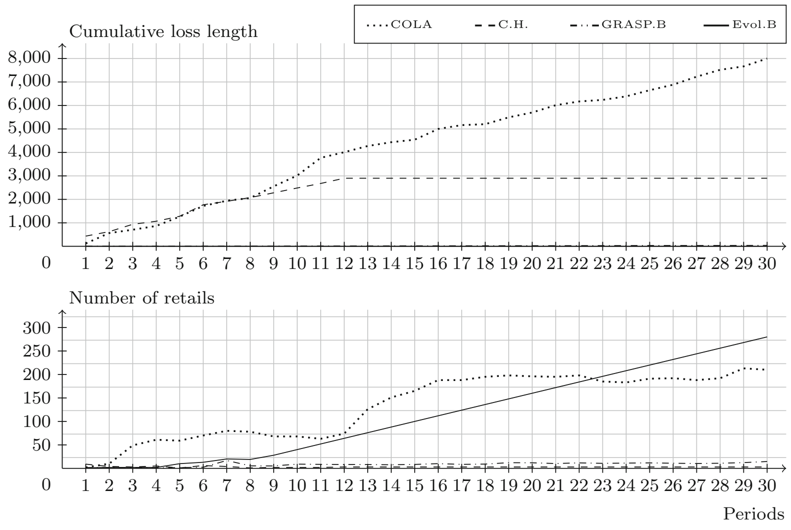


Fig. 9 Cumulative loss of material and total number of retails per period for instance 4 from Tomat and Gradišar (2017)

stock, we observe that over more than half of the periods, the produced number of retails was lower than the produced by $COLA$, so if the simulation would stopped at any period before the 23^{th}, then $COLA$ would had the greatest retail number at the final stock.

The authors from Tomat and Gradišar (2017), also proposed an objective function that considers to minimize the total length of the loss and also the total length of

Table 17 Comparison with the best solutions of the heuristic search process in Tomat and Gradišar (2017)

Sequence 1	H.S.P.	C.H.	GRASP.B	Evol.B
Final Ret	3	**1**	**1**	**1**
Obj	907	436	391	**281**
Sequence 2	H.S.P.	C.H.	GRASP.B	Evol.B
Final Ret	**1**	2	**1**	11
Obj	2461	1456	**486**	6306
Sequence 3	H.S.P.	C.H.	GRASP.B	Evol.B
Final Ret	**1**	**1**	**1**	26
Obj	8764	2406	**422**	9092
Sequence 4	H.S.P.	C.H.	GRASP.B	Evol.B
Final Ret	4	**3**	14.6	280
Obj	16320	7181	**5004.1**	98976

The best values are highlighted in bold

the non used retails along the simulation process. To each length was associated a weight value, the tested weights were 2 for the length of the material loss and 1 for the length of the retails. Then, they tested an heuristic search process ($H.S.P.$) that, after each period, adjusted parameters associated to the size of the retails and the optimal number of retail in the stock. Such process was tested for each sequence with different thresholds for the sizes of the retails and we selected the best results reported by the authors for each sequence, to compare with our results.

Table 17 compares our solutions with the $H.S.P.$, for each sequence we show the number of retails at the stock and the objective function value. For almost all the tests, the best values of the objective function defined by Tomat and Gradišar (2017) was given by the $GRASP.B$, while the $H.S.P$ was always worst than $GRASP.B$, $C.H$ and in some cases than $Evol.B$. The greater values of the objective function for the $Evol.B$ in sequences 2, 3 and 4 are given by the fact that such function depends on the total length of the retails. Notice that, to minimize the total length of the retails generally is a worst strategy than to minimize their number, since generating few greater retails could be better than generating several small ones (ex. there are more possible items and patterns that one can cut from one retail of size 900 than from ten retails of size 80, but if comparing by the total length of the retails it would be preferable the ten retails of size 80). However, to consider the total length of the loss is a good strategy since it directly represents an economy save or a pollution reduction. On that direction, our $Evol.B$ gave the best results since the algorithm did not produce loss of material for any of the simulations.

For all the tested multi-period instances, our $GRAP.B$ algorithm obtained the most balanced solutions according to the length of the loss and the number of retails, achieving in almost all the cases the best solutions in both objectives and improving the best solutions in the literature. Also, our $Evol.B$ was able to produce good solutions in almost all the instances with no loss of material and even our $C.H.$ was able to produce some interesting undominated solutions.

5 Conclusions

We designed and implemented a constructive heuristic, a GRASP meta-heuristic and an evolutive algorithm to solve the 1D- CSPUL. We tested our proposals over several instances from the literature and the computational experiments were quite good to all the tested instances: our algorithms were very fast and arrived on very good solutions, being able to find in many cases Pareto optimal solutions and, in some cases, the complete Pareto frontier.

We compared our approaches with the best ones in the literature and, since our goal was to approximate the Pareto frontier, several times our meta-heuristics were able to reach attractive solutions that the other approaches did not reach. Generally our evolutive algorithm presented the best results, always reaching a non-dominated set of solutions and, among them, finding solutions with the less loss of material compared to the solutions given for all the other algorithms.

During the tests, we observed our algorithms preferred to cut smaller objects first. This strategy could be interesting when future demands are expected, since usually it is easier to create good cut patterns over larger objects. In fact, smaller objects are often retails from past cutting plans, so it is also desirable to prioritize their use.

We also tested our algorithms over multi-period instances, which consider demands arrive in different time periods of a time horizon. The results we obtained were very competitive and for almost all the cases they were a nice improvement over the best solutions in the literature. Different from the one-period instances, for the multi-period cases, the best results were not majorly produced by our evolutive meta-heuristic but by our GRASP, always dominating solutions from previous approaches in the literature.

Some interesting directions for future works are: to also minimize the number of different cutting patterns (see Feifei et al. 2012), or to consider not the one dimensional cutting stock problem but the two and three dimensional cutting stock problem (see Andrade et al. 2016, 2014).

Acknowledgements This work was partially supported by the Brazilian National Council for Scientific and Technological Development (CNPq) under grant 312206/2015-1.

References

Abuabara, A., Morabito, R.: Cutting optimization of structural tubes to build agricultural light aircrafts. Ann. OR **169**, 149–165 (2009). https://doi.org/10.1007/s10479-008-0438-7

Andrade, R., Birgin, E., Morabito, R.: Two-stage two-dimensional guillotine cutting stock problems with usable leftover. Int. Trans. Oper. Res. **23**(1–2), 121–145 (2016). https://doi.org/10.1111/itor.12077

Andrade, R., Birgin, E.G., Morabito, R., Ronconi, D.P.: MIP models for two-dimensional non-guillotine cutting problems with usable leftovers. J. Oper. Res. Soc. **65**(11), 1649–1663 (2014). https://doi.org/10.1057/jors.2013.108

Arenales, M.N., Cherri, A.C., Nascimento, D.N., Vianna, A.: A new mathematical model for the cutting stock/leftover problem. Pesqui. Oper. **35**, 509–522 (2015). https://doi.org/10.1590/0101-7438.2015.035.03.0509

Cherri, A., Arenales, M., Yanasse, H.: The one-dimensional cutting stock problem with usable leftover—a heuristic approach. Eur. J. Oper. Res. **196**, 897–908 (2009). https://doi.org/10.1016/j.ejor.2008.04.039

Cherri, A., Arenales, M., Yanasse, H.: The usable leftover one-dimensional cutting stock problem—a priority-in-use heuristic. Int. Trans. Oper. Res. **20**, 189–199 (2013). https://doi.org/10.1111/j.1475-3995.2012.00868.x

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. Eur. J. Oper. Res. **236**(2), 395–402 (2014). https://doi.org/10.1016/j.ejor.2013.11.026

Cui, Y., Yang, Y.: A heuristic for the one-dimensional cutting stock problem with usable leftover. Eur. J. Oper. Res. **204**(2), 245–250 (2010). https://doi.org/10.1016/j.ejor.2009.10.028

Eschenauer, H., Koski, J., Osyczka, A.: Multicriteria Design Optimization. Springer, Berlin (1990). https://doi.org/10.1007/978-3-642-48697-5

Feifei, G., Lin, L., Jun, P., Xiazi, Z.: Study of one-dimensional cutting stock problem with multi-objective optimization. In: International Conference on Computer Science and Information Processing (CSIP), pp. 571–574 (2012). https://doi.org/10.1109/CSIP.2012.6308918

Gradišar, M., Erjavec, J., Tomat, L.: One-dimensional cutting stock optimization with usable leftover: a case of low stock-to-order ratio. Int. J. Decis. Supp. Syst. Technol. **3**, 54–66 (2011). https://doi.org/10.4018/jdsst.2011010104

Gradišar, M., Jesenko, J., Resinovič, G.: Optimization of roll cutting in clothing industry. Comput. Oper. Res. **24**(10), 945–953 (1997). https://doi.org/10.1016/S0305-0548(97)00005-1

Gradišar, M., Kljajić, M., Resinovič, G., Jesenko, J.: A sequential heuristic procedure for one-dimensional cutting. Eur. J. Oper. Res. **114**(3), 557–568 (1999). https://doi.org/10.1016/S0377-2217(98)00140-4

Gradišar, M., Trkman, P.: A combined approach to the solution to the general one-dimensional cutting stock problem. Comput. Oper. Res. **32**(7), 1793–1807 (2005). https://doi.org/10.1016/j.cor.2003.11.028

Ogunranti, G.A., Oluleye, A.E.: Minimizing waste (off-cuts) using cutting stock model: the case of one dimensional cutting stock problem in wood working industry. J. Ind. Eng. Manag. **9**(3), 834–859 (2016). https://doi.org/10.3926/jiem.1653

Poldi, K.C., Arenales, M.N.: Heuristics for the one-dimensional cutting stock problem with limited multiple stock lengths. Comput. Oper. Res. **36**(6), 2074–2081 (2009). https://doi.org/10.1016/j.cor.2008.07.001

Ravelo, S.V., Meneses, C.N., Santos, M.O.: Mathematical programming models for the one-dimensional cutting stock problem with usable leftover. XLII SBPO (2010)

Tam, V.W.Y., Tam, C.M., Chan, J.K.W., Ng, W.C.Y.: Cutting construction wastes by prefabrication. Int. J. Constr. Manag. **6**(1), 15–25 (2006). https://doi.org/10.1080/15623599.2006.10773079

Tomat, L., Gradišar, M.: One-dimensional stock cutting: optimization of usable leftovers in consecutive orders. CEJOR **25**(2), 473–489 (2017). https://doi.org/10.1007/s10100-017-0466-y

Šajn, N.: Environmental impact of the textile and clothing industry: What consumers need to know. European Parliamentary Research Service (2019)

Wattanasiriseth, P., Krairit, A.: An application of cutting-stock problem in green manufacturing: a case study of wooden pallet industry. IOP Conf. Ser.: Mater. Sci. Eng. **530**, 012005 (2019). https://doi.org/10.1088/1757-899x/530/1/012005

Wäscher, G., Haußner, H., Schumann, H.: An improved typology of cutting and packing problems. Eur. J. Oper. Res. **183**(3), 1109–1130 (2007). https://doi.org/10.1016/j.ejor.2005.12.047