Computers & Operations Research 48 (2014) 69–80

Contents lists available at ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/caor

Fast pattern-based algorithms for cutting stock

Filipe Brandão *, João Pedro Pedroso

INESC TEC and Faculdade de Ciências, Universidade do Porto, Portugal

ARTICLE INFO

Available online 13 March 2014

Keywords:
Cutting stock
First fit decreasing
Best fit decreasing

ABSTRACT

The conventional assignment-based first/best fit decreasing algorithms (FFD/BFD) are not polynomial in the one-dimensional cutting stock input size in its most common format. Therefore, even for small instances with large demands, it is difficult to compute FFD/BFD solutions. We present pattern-based methods that overcome the main problems of conventional heuristics in cutting stock problems by representing the solution in a much more compact format. Using our pattern-based heuristics, FFD/BFD solutions for extremely large cutting stock instances, with billions of items, can be found in a very short amount of time.

1. Introduction

The one-dimensional cutting stock problem (1DCSP) is a combinatorial NP-hard problem (see, e.g., [7]) in which pieces of different lengths must be cut from rolls, each with length L, in such a way that the amount of unused space is minimized. In this problem, we are given the length L of the rolls, the number of different piece lengths m, and for each index $k \in \{1, \ldots, m\}$ the corresponding piece length l_k and demand b_{l_k}. Since all the rolls have the same length and demands must be met, the objective is equivalent to minimizing the number of rolls that are used.

The bin packing problem (BPP) can be seen as a special case of the cutting stock problem (1DCSP). In BPP, n objects of m different weights must be packed into a finite number of bins, each with capacity W, in a way that minimizes the number of bins used. However, in the 1DCSP, items of equal size (which are usually ordered in large quantities) are grouped into orders with a required level of demand; in BPP, the demand for a given size is usually close to one. According to Wäscher et al. [10], cutting stock problems are characterized by a weakly heterogeneous assortment of small items, in contrast with bin packing problems, which are characterized by a strongly heterogeneous assortment of small items.

Since the BPP is hard to solve exactly, many different heuristic solution methods have been proposed. In our paper, we will focus on two heuristics: first fit decreasing (FFD) and best fit decreasing (BFD). Coffman et al. [3] and [4] analyze the average and worst case behavior of these two heuristics and provide an excellent survey on on-line and off-line heuristics for BPP.

The best known implementations of FFD and BFD heuristics run in $\mathcal{O}(n \log n)$ and $\mathcal{O}(n \log(\min(n, W)))$, respectively. Neither of these run times is polynomial in the 1DCSP input size. Therefore, for large 1DCSP instances, computing a FFD/BFD solution may take a very long time using any conventional algorithm. These algorithms are usually fast enough for BPP, but for large 1DCSP instances better alternatives are necessary. Johnson [8] proves that for any implementation of the FFD/BFD heuristics, there exists a list of items of length n for which it will take at least $\Omega(n \log n)$ time to compute a solution. However, his proof requires lists of n items of different weights, which are not usual in cutting stock problems.

Pattern-based algorithms overcome the main problems of conventional approaches in cutting stock problems by representing the solution in a much more compact format. The final solution is represented by a set of patterns with associated multiplicities that indicate the number of times the pattern is used. Moreover, each pattern is represented by the piece lengths it contains and the number of times each of them appears. This representation of the output allows us to avoid the $\Omega(n)$ limit introduced by the assignment piece-by-piece. Besides that, this output format is much more practical in some situations where we are just interested in knowing how to cut the rolls.

The main contributions of this paper are pattern-based algorithms for FFD/BFD heuristics. Our best pattern-based FFD and BFD implementations run in $\mathcal{O}(\min(m^2, n) \log m)$ and $\mathcal{O}(\min(m^2, n) \log(\min(n, W)))$, respectively. Both run times are polynomial in the 1DCSP input size. These algorithms allow us to find solutions for large 1DCSP instances quickly; for example, solutions for instances with one thousand million pieces are found in much less than one second of CPU time on current computers.

The input format of all the algorithms in this paper is the following: m – the number of different lengths; l – the set of lengths; b – the demand for each length; and L – the roll length. The output format depends on the algorithm used. Nevertheless, all the output formats will provide enough information to obtain an assignment piece-by-piece in $\Theta(n)$.

* Corresponding author.

http://dx.doi.org/10.1016/j.cor.2014.03.003
0305-0548/

The remainder of this paper is organized as follows. Section 2 presents a straightforward assignment-based FFD algorithm. Section 3 presents the pattern-based FFD algorithm along with an illustrative example. A complexity analysis is presented in Section 3.5. Section 4 and Section 5 present assignment-based and pattern-based BFD algorithms, respectively. Some computational results are presented in Section 6. Finally, Section 7 presents the conclusions.

2. Conventional first fit decreasing

The straightforward Algorithm 1 starts by sorting the piece lengths in decreasing order, in line 2, and initializes the solution with a single empty roll, in lines 3–5. The current solution is represented by the number of rolls used (R), the list of assignments (Sol; an assignment of a piece of length l_i to a roll k is represented as $l_i \rightarrow$ roll k) and the list of available spaces (Rem). For each length l_i, the b_{l_i} pieces will be inserted one-by-one in the solution. For each piece, we seek for the first roll where the piece fits. If we find a roll where the piece fits (line 11), we add the piece to the roll and update its remaining space. If there is no roll with enough space (line 17), we create a new one. The variable k' is used to improve the running time in 1DCSP instances by starting to seek for the first roll where the last piece of the same length was inserted (since the length is the same, we know that none of the previous rolls had enough space). The running time of this algorithm is $\mathcal{O}(n+mR)$.

Algorithm 1. Straightforward first fit decreasing algorithm.

```
    input: m – number of different lengths; l – set of lengths; b – demand for each length; L - roll length
    output: R – number of rolls needed; Sol – list of assignments
1   function FFD(m, l, b, L):
2       l ← reverse(sort(l));   //sort lengths in decreasing order
3       Sol ← [ ];
4       R ← 1;
5       Rem ← [L];
6       for i ← 1 to m do   //for each length
7           k' ← 1;
8           for j ← 1 to b_{l_i} do   //for each piece of length l_i
9               assigned ← False;
10              for k ← k' to R do   //try each roll
11                  if Rem[k] > l_i then   //if there is enough space
12                      Rem[k] ← Rem[k] − l_i;
13                      Sol.append(l_i → roll k);
14                      assigned ← True;
15                      k' ← k;
16                      break;
17              if not assigned then   //if the piece was not assigned to any roll
18                  R ← R + 1;
19                  k' ← R;
20                  Rem.append(L − l_i);
21                  Sol.append(l_i → roll R);
        return (R, Sol);
```

2.1. Efficient implementation

Johnson [8] shows how to implement the FFD algorithm to run in $\mathcal{O}(n \log n)$ time, where n is the number of objects, using a tree. His tree was generalized for other applications and nowadays it is known as tournament tree, or as winner tree. A winner tree is a complete binary tree in which each node represents the best of its two children. For BPP, we need a tree with at least n leaves corresponding to bins. The value of each leaf is the remaining available space. Fig. 1 shows a winner tree applied to a small BPP instance. To find the leftmost bin

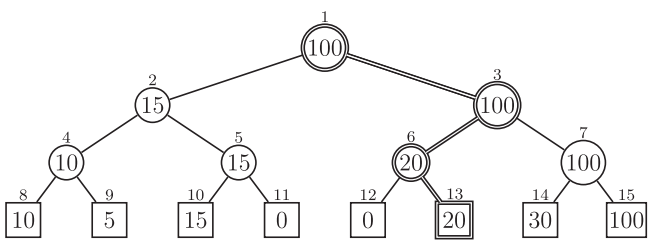

Fig. 1. Winner tree. Winner tree before an assignment of a piece of length 18 when the first 7 rolls, of capacity 100, are filled with 90, 95, 85, 100, 100, 80 and 70, respectively. The highlighted path indicates the search path along the tree that finds the leftmost bin with gap at least 18. The number inside each node is the key value and the number above is the index in the array representation.