PERGAMON

Intl. Trans. in Op. Res. 7 (2000) 185–210

INTERNATIONAL TRANSACTIONS IN OPERATIONAL RESEARCH

www.elsevier.com/locate/orms

Constrained two-dimensional cutting stock problems a best-first branch-and-bound algorithm

Van-Dat Cung[a], Mhand Hifi[a,b,*], Bertrand Le Cun[a]

[a]*Laboratoire PRiSM-CNRS UMR 8636, Université de Versailles-Saint Quentin en Yvelines, 45 avenue des États-Unis, 78035 Versailles Cedex, France*

[b]*CERMSEM, Maison des Sciences Economiques, Université de Paris 1, 106-112 Bd de L'Hopital, 75013 Paris, France*

Received 1 February 1999; received in revised form 15 September 1999; accepted 16 September 1999

Abstract

In this paper, we develop a new version of the algorithm proposed in Hifi (Computers and Operations Research 24/8 (1997) 727–736) for solving exactly some variants of (un)weighted constrained two-dimensional cutting stock problems. Performance of branch-and-bound procedure depends highly on particular implementation of that algorithm. Programs of this kind are often accelerated drastically by employing sophisticated techniques. In the new version of the algorithm, we start by enhancing the *initial lower bound* to limit initially the space search. This initial lower bound has already been used in Fayard et al. 1998 (Journal of the Operational Research Society, 49, 1270–1277), as a heuristic for solving the constrained and unconstrained cutting stock problems. Also, we try to improve the *upper bound* at each internal node of the developed tree, by applying some simple and efficient combinations. Finally, we introduce some new *symmetric-strategies* used for neglecting some unnecessary *duplicate patterns*. The performance of our algorithm is evaluated on some problem instances of the literature and other hard-randomly generated problem instances. © 2000 IFORS. Published by Elsevier Science Ltd. All rights reserved.

Keywords: Combinatorial optimization; Cutting stock; Dynamic programming; Heuristics; Knapsacks; Optimization

* Corresponding author.
E-mail address: hifi@univ-paris1.fr (M. Hifi).

1. Introduction

Cutting and Packing problems belong to an old and very well-known family, called CP in Dyckhoff (1990) and Sweeney and Paternoster (1992). This is a family of natural combinatorial optimization problems, admitted in numerous real-world applications from computer science, industrial engineering, logistics, manufacturing, management, production process, etc. Long and intensive research on the problems of this family has led (and still leads) to the development of models and mathematical tools, so interesting by themselves that their application domains surpass the framework of CP problems.

Concerning the study and the development of tools for solving cutting problems (started 60 years ago), we quote here the fascinating typology performed in Dyckhoff (1990) and Dyckhoff and Finke (1992) as well as the interesting categorized bibliography of Sweeney and Paternoster (1992).

We study in this paper one of the most interesting problems of CP, the *Constrained Two-Dimensional Cutting stock problem* (shortly CTDC). In CTDC, we are given a large stock rectangle S of given dimensions $L \times W$ and n types of smaller rectangles (pieces) where the i-th type has dimensions $l_i \times w_i$. Furthermore, each type i, $i = 1, \ldots, n$, is associated with a profit c_i and an upper bound b_i. The problem is now to cut off from the large rectangle a set of small rectangles such that: (i) all pieces have *fixed orientation*, i.e., a piece of length l and width w is different from a piece of length w and width l (when $l \neq w$); (ii) all applied cuts are of *guillotine* type, i.e., cuts that start from one edge and run parallel to the other two edges; (iii) there are at *most* b_i rectangles of type i (the demand value of the i-th piece) in the (feasible) cutting pattern; (iv) the overall profit obtained by

$$\sum_{i=1}^{n} c_i x_i,$$

where x_i denotes the number of rectangles of type i in the cutting pattern, is *maximized.*

This problem has a wide range of industrial and commercial applications. This version of the problem appears in the problem of cutting steel or glass plates (Hahn, 1967) into required stock sizes to minimize the waste. The problem also appears in a production line, where the profits could define priorities for some pieces or even to impose that some pieces, already present in the current cutting pattern, should appear also in the next cutting pattern (see Dyson and Gregory, 1974; Morabito and Garcia, 1998).

For the cutting stock problem, depending on the framework of the application giving rise in the particular problem and, also the computational resources of the solvers, a large variety of solution-methods, exact and approximate, have been devised.

The unconstrained two-dimensional cutting has been solved optimally by dynamic programming (Beasley, 1985; Gilmore and Gomory, 1966) and by the use of recursive (tree search) procedures (Herz, 1972; Hifi, 1994; Hifi and Zissimopoulos, 1996). The CTDC version, has been solved optimally by applying a general tree search based upon a depth-first search method (Christofides and Whitlock, 1977; Hifi, 1994; Hifi and Zissimopoulos, 1997) and also by the use of a branch-and-bound procedure based upon a best-first search method (Hifi, 1997; Viswanathan and Bagchi, 1993). Previous works have also developed heuristic approaches to the unconstrained and constrained ones (see Fayard and Zissimopoulos, 1995; Fayard et al., 1998; Hifi and Ouafi, 1997; Morabito and Arenales, 1996; Oliveira and Ferreira, 1990; Vasko, 1989; Wang, 1983; Zissimopoulos, 1985).

This paper is organized as follows. First (Section 2), we describe the main idea of two different exact approaches: the *top-down* approach (Christofides and Whitlock, 1977) and the *bottom-up* one (Viswanathan and Bagchi, 1993) (called VB). The modified VB algorithm (called MVB in Hifi (1997)) is summarized by considering: (i) the complementary upper bound used at each internal node (using a bounded one-dimensional knapsack problem) and improved upper bounds and, (ii) the initial lower bound produced by using a series of (*bounded*) *one-dimensional knapsacks* and two *strip packing problems*. Second (Section 3), we develop a new version of the MVB algorithm by enhancing (i) the lower and upper bounds (Sections 3.1 and 3.2), (ii) a good representation of each internal node (Sections 3.3 and 3.4) and (iii) three new symmetric-strategies used to reduce drastically the number of generated nodes (Section 3.5). Finally, in Section 4 the performance of different versions of the new algorithm is evaluated on some instances of the literature and on different randomly generated instances with different sizes and, benchmark results of these are given.

2. Exact algorithms

Up to now, people working on guillotine CTDC have developed very interesting exact and approximate algorithms. To our knowledge, two exact approaches have been developed for the CTDC stock problem. The first one, called the *top-down approach*, has been originally proposed by Christofides and Whitlock in 1977 and it is a depth-first search method. Using the same strategy, in Hifi (1994) and Hifi and Zissimopoulos (1997) we have developed an improved version of the last algorithm, called MCW algorithm. The second one, called the *bottom-up approach*, has been originally proposed by Wang in 1983 and generalized by Viswanathan and Bagchi in 1993. This method is based upon best-first search strategy. A modified version of this algorithm, using Wang's principle, has been proposed by Hifi (1997), called MVB algorithm. This version of the algorithm is considered as a modified version, because the algorithm can easily run without using the best-first search strategy.

In what follows, we focus on the bottom-up approach since the first one, i.e. the top-down approach, can generally be applied especially for solving the CTDC problem with constraints (on the number of pieces) of types $\leq b_i$, $i = 1, \ldots, n$. The bottom-up approach is flexible enough to be easily tailored to meet almost any type of constraints on the patterns generated in the CTDC stock problem. For example, the following constraints can be handled easily:

- constraints on the total number of pieces in each pattern (see Dyson and Gregory (1974));
- constraints on the number of different types of pieces in each pattern (see Morabito and Garcia (1998));
- constraints on the total number of pieces belonging to specified subsets of types of demand pieces (see Haessler (1971, 1975));
- all pieces and patterns can be turned 90° (see Wang (1983)).

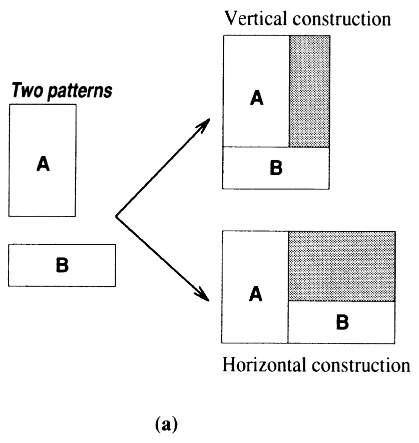
Furthermore, constraints of the types $\leq$, $=$, $\geq (b_i, i = 1, \dots n)$ can be mixed in each instance of the CTDC problem, and so, the flexibility of the bottom-up approach increases even more.

2.1. The bottom-up approach

The bottom-up approach (Viswanathan and Bagchi, 1993) is based mainly upon best-first search strategy. This approach is based on the observation that all guillotine cutting patterns on the initial plate can be obtained by means of horizontal and vertical builds. These builds are based on Wang (1983) constructions which can also be seen as a particular case of the method used by Albano (1977) and by Albano and Sapuppo (1980) for solving especially the irregular-shape cutting stock problem (improved versions of Wang's approach were presented separately by Oliveira and Ferreira (1990) and Vasko (1989).

The main idea to get an optimal cutting pattern for the initial stock rectangle S is to combine successively pieces to larger subrectangles using horizontal and vertical builds. A horizontal build of two subrectangles A and B of dimensions $x_A \times y_A$ and $x_B \times y_B$ is a subrectangle of dimensions $(x_A + x_B) \times \max\{y_A, y_B\}$. A vertical build of the same subrectangles is a subrectangle of dimensions $\max\{x_A, x_B\} \times (y_A + y_B)$ (see Fig. 1(a)). If the demand constraints, for all pieces, are not violated and the dimensions of the resulting subrectangle (called the *internal rectangle*), denoted by R, do not exceed the dimensions of the initial stock rectangle, then the algorithm uses R by placing it at the bottom-left corner of the initial stock rectangle (Fig. 1(b)). Then it exploits the unoccupied area in the initial rectangle (the region P of Fig. 1(b)), to obtain good upper bounds.

So, given an instance of the CTDC problem and an internal rectangle R, the objective is to seek an *upper bound* for the value of the best guillotine cutting pattern on S, that is constrained



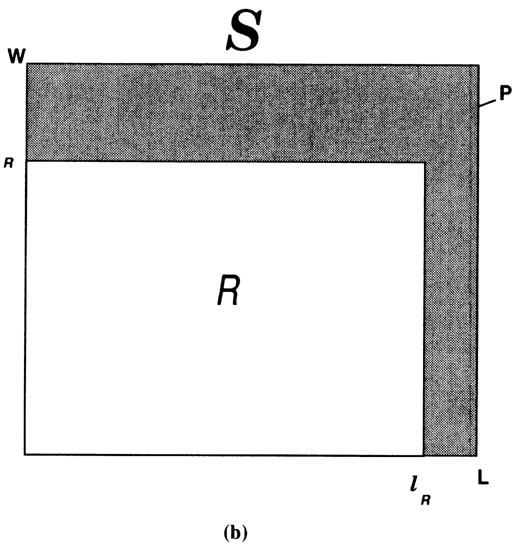


Fig. 1. (a) Vertical and horizontal constructions, and (b) the stock rectangle S, the guillotine rectangle R and its dimensions (l_R, w_R).

to include R. Let $g(R)$ be the *internal value* of R that is the sum of the profits of the pieces contained in R. Let $h(R)$ be the maximum value which could be obtained, without violating the demand constraints, from the region P of Fig. 1(b). Then $f(R) = g(R) + h(R)$ represents the maximum value of a feasible cutting pattern, that is constrained to include R. The goal now is to determine an estimation $h'(R)$ of the upper bound of $h(R)$.

In Hifi (1997), *a modified version* of VB approach (called MVB) has been proposed. These modifications are relied (i) to initialize the VB algorithm using a lower bound at the root node (see Hifi, 1994, 1997; Hifi and Zissimopoulos, 1997); (ii) to introduce a complementary upper bound using a bounded one-dimensional knapsack problem and, (iii) to implement a second upper bound obtained by a judicious manipulation of the results given by Gilmore and Gomory's (1966) procedure and the internal solution value at each node (internal rectangle).

2.2. Estimation of $h'(R)$ at nodes

Viswanathan and Bagchi (1993) proposed two ways to compute the upper bound $h'(R)$. One way to perform such a bound ($h'(R)$) denoted $U_1(R)$ is shown in Gilmore and Gomory (1966), which is mainly based upon dynamic programming procedures. The second way which is an improved version of the previous one (denoted $U_2(R)$) can be found in Hifi (1997) and Viswanathan and Bagchi (1993).

In Hifi (1997), we have proposed two complementary upper bounds. The first one was obtained by applying dynamic programming techniques to solve a *bounded one-dimensional knapsack problem* (denoted $U_2'(R)$). The second one was obtained by combining Gilmore and Gomory's algorithm and the local lower bound of the node with configuration R (denoted $U_1'(R)$). Hence, the new global upper bound, at each (internal) node containing R, is the bound which achieves the *minimum value* between the different upper bounds (denoted $U_2''(R)$).

Once $g(R)$ is known and $h'(R)$ has been estimated, we can compute $f'(R) = g(R) + h'(R)$, which is an *upper bound* of the guillotine cutting pattern on S containing R.

2.3. The lower bound at the root node

In Hifi (1994, 1997) and Hifi and Zissimopoulos (1997) an approximate algorithm has been used in order to construct an initial feasible solution at the root node. The algorithm is based on the following *observations*.

Observation 1: an optimal cutting pattern is often composed by some vertical and/or some horizontal strips in the initial stock rectangle.
Observation 2: an optimal cutting pattern is frequently evaluated to another cutting pattern composed by some strips.
Observation 3: generally, an optimal cutting pattern is composed by few subrectangles produced by using some cuts on the initial stock rectangle, and where each subrectangle has a strip structure.

Box 1. *The MVB algorithm*

```
Open: the set of subproblems
Clist: the list of stored best subproblems
$R$ and $q$: are the guillotine rectangles
$g(R)$, $h'(R)$, $f'(R)$: the internal value, the upper bound of the region $P$ and the upper bound
        of the subproblem containing the configuration $R$.
best, Opt: the best current feasible solution and its solution value.
---
Input: an instance of the constrained CTDC problem.
Output: an optimal solution best of value Opt.
---
Open:= $\{R_1, R_2, ..., R_n\}$; Clist:=empty set; finished:=false;
Compute $U_2''(x_R, y_R)$ for each $(x_R, y_R) = (1, \ldots, L, 1, \ldots, W)$ (Section 2.3)
$Opt = g(best)$ where best represents the solution obtained by using the initial lower bound
on $S$ (Section 2.4).

repeat
  choose a rectangle $R$ from Open having the highest $f'$ value;
  if $(h'(R) = 0)$ or $(f'(R) - Opt(best) \leq 0)$ then finished:=true
  else
   begin
    1. transfer $R$ from Open to Clist;
    2. construct all guillotine rectangles $Q$ such that:
       2.1. $Q$ is a horizontal or vertical build of $R$ with some
            rectangle $R'$ (including $R$) of Clist;
       2.2. dimensions of $Q \leq (L, W)$;
       2.3. $\forall q \in Q$, $d_k^q \leq b_k$, $k = 1, ..., n$;
    3. for each element $q \in Q$, do:
      if $g(q) > Opt(best)$ then set $best = q$ and $Opt(best) = g(q)$;
      if $f'(q) > Opt(best)$ then set Open=Open $\cup\{q\}$;
    4. remove all elements $R$ from Open having $f'(R) \leq Opt(best)$;
    5. if Open={} then finished:=true;
   end;
until finished;
```

Basic ideas of this algorithm are mainly based upon Gilmore and Gomory's procedure (1966): (i) to create *optimal horizontal strips* with respect to the length of the (sub)rectangle to cut, the *demand* constraints and a given width and then to select the *best* of them with respect to the width of the rectangle for *filling the rectangle*; (ii) to create *optimal vertical strips* with respect to the width of the (sub)rectangle to cut, the *demand* constraints and a given length and then to select the *best* of them with respect to the length of the rectangle for *filling the rectangle*; (iii) to select the best feasible solution among both solutions and considered it as a lower bound to the CTDC problem.

The creation of the optimal horizontal strips can be obtained by solving only a bounded one-dimensional knapsack. It is possible because *dynamic programming procedures* are applied to the *largest one-dimensional knapsack* problem to construct the *highest strip* and so, all (sub)strips with a width small than the highest one, are available. Similarly, the optimal vertical strips can be obtained by solving another bounded one-dimensional knapsack problem. Finally, the best feasible solution (considered as an initial lower bound) is obtained by applying two other *unbounded knapsacks* and/or two *strip packing problems* (for more details about the resolution of these problems, the reader can be referred to Hifi (1994, 1997); Hifi and Zissimopoulos (1997) and Zissimopoulos (1985).

In The MVB algorithm, we describe the formal MVB algorithm by introducing the initial lower bound and the refinement upper bounds. The MVB algorithm starts by evaluating the *initial lower bound* (see Hifi, 1994, 1997; Hifi and Zissimopoulos, 1997), the upper bounds (see Gilmore and Gomory, 1966; Hifi, 1997; Viswanathan and Bagchi, 1993) and, both *complementary* and *improved* upper bounds (see Hifi, 1997).

The algorithm uses two lists, Open and Clist. The Open list initially contains n elements such that each element $R_i(\in \text{Open})$, $i = 1, \ldots, n$, is composed by the dimensions of the i-th piece $(l_{R_i}, w_{R_i}) = (l_i, w_i)$, the internal value $g(R_i) = c_i$, the estimation value $h'(R_i)$, the upper bound $f'(R_i)$ and a vector d^{R_i} of dimension n, where $d_k^{R_i}$, $k = 1, \ldots, n$, is the number of occurrences of the k-th piece in the guillotine rectangle R_i. The list Clist is initialized to empty set. At each iteration, a rectangle R with highest upper bound $f'(R)$ is selected from Open and moved to Clist. A set Q of new rectangles is created by combining R with elements of Clist (including R) using horizontal and vertical builds. The Q set contains only the elements R' which satisfy $(l_{R'}, w_{R'}) \leq (L, W)$, and $d_k^{R'} \leq b_k$, $k = 1, \ldots, n$ (demand constraints). The algorithm stops when the selected rectangle R from Open has either the value $h'(R) = 0$ or the value $f'(R)$ small than the best solution value $Opt(best)$ found up to now. Then the best current solution *best* represents the optimal solution of the problem with value $Opt(best)$, since $f'(R)$ represents the highest (the best upper bound) value over all values of the Open list and necessary it is equal to $Opt(best)$.

3. A new version of the MVB algorithm

First, we describe the improvement of the *initial lower* bound (feasible solution) in order to reduce initially the search space. Second, we present the complementary upper bounds for the internal nodes and a new development strategy. Moreover, we summarize the BOB library applied to our algorithm and we introduce an improvement that is focused on managing the Clist list (see Box 1). It consists in finding a better representation of all elements of this list in order to speed up the search process. Finally, in the last part we describe the symmetric-strategies applied in our algorithm to reduce some useless branches.

3.1. A better initial lower bound

In this paper, we just describe the main idea of the heuristic which generates a better initial solution and for more details the reader can be referred to Fayard et al. (1998).

As shown in Hifi (1997) and Hifi and Zissimopoulos (1997), the initial lower bound is obtained by solving four bounded one-dimensional knapsack problems and by solving two strip packing problems. In Fayard et al. (1998), another idea has been used. This approach can be considered as an improved and a faster version of the heuristic considered in Zissimopoulos (1985). It consists in dealing with a *finite number of subrectangles* which also shall be filled by *optimal horizontal and/or vertical (sub)strips*, i.e. construct a set of couples (R_1, R_2), where $R_1 = (x, W)$ and $R_2 = (L - x, W)$ when the cut is vertical, and $R_1 = (L, y)$ and $R_2 = (L, W - y)$ if the cut is horizontal. Hence, we deal with solving a class of (small) one-dimensional knapsack problems. The heuristic developed in Fayard et al. (1998) is called the *General Best Strip Cutting* (GBSC) algorithm.

Let us see now how we can construct the finite number of subrectangles. When we deal with a rectangle (α, β), Christofides and Whitlock (1977) have proved that each optimal cutting pattern on this rectangle has an equivalent *normal cutting pattern*. In a normal cutting pattern each guillotine cut is made on some points which are linear combinations of the lengths (resp. widths) of pieces entering in (α, β). Such normal patterns allow to restrict the points where cuts are made. These two sets of linear combinations can be represented by

$$P_\alpha = \{x \mid x = \sum_{i=1}^{n} l_i z_i \le \alpha,\ z_i \le b_i,\ z_i \in N\} \text{ and } P_\beta = \{x \mid x = \sum_{i=1}^{n} w_i z_i \le \beta,\ z_i \le b_i,\ z_i \in N\}$$

In Fayard et al. (1998), we have shown that the two sets can be limited only to the subrectangles of types (x, W) and $(L-x, W)$ if the cut is applied vertically and to (L, y) and $(L, W-y)$ if the cut is made horizontally. Now, the main idea of GBSC algorithm can be summarized by the following points (we just consider the set of points P_L):

1. Let $F_{best}(=0)$ be the value of the best feasible solution "*best*".
2. Consider a vertical x-cut on (L, W) such that $x \in P_\alpha$ and denote the resulting rectangles (x, W) and $(L-x, W)$, respectively.
3. Construct a feasible solution on (x, W) by applying the heuristic approach used for starting the MVB algorithm (see Hifi, 1997). Let F be its value and b^* be its composition.
4. Construct a feasible solution on $(L-x, W)$ by applying the same heuristic approach as for the previous point, but by considering the new demand vector $b_j - b_j^*$, $j = 1, \ldots, n$. Let F' be the value of the obtained solution.
5. If $F_{best} < F + F'$ then update the new solution and its value.
6. Repeat steps 2–5 until all x-cuts are considered.

As we have shown in Fayard et al. (1998), sometimes P_L and P_W become large and the computational effort becomes also important. Our computational results showed that the initial phase produces *good initial lower bounds* and *without too much computational effort*, when these sets were limited only to *few linear combinations*. Our computational results are evaluated by considering $P_L = P_{L/2} \cup \{l_i \mid l_i \ge \lfloor L/2 \rfloor + 1,\ i = 1, \ldots, n\}$ and $P_W = P_{W/2} \cup \{w_i \mid w_i \ge \lfloor W/2 \rfloor + 1,\ i = 1, \ldots, n\}$. Consequently, this strategy is applied in order to obtain better initial solution values (for reducing initially the search space) without too much computational effort (for increasing the efficiency of the new version of the algorithm).

3.2. Upper bounds and development strategy

3.2.1. The upper bound at internal nodes

The MVB uses an upper bound obtained by combining three complementary upper bounds. The two first upper bounds are obtained by considering the *unconstrained cutting problem*, i.e. without considering the demand values b_i, $i = 1, \ldots, n$. Furthermore, the third bound is obtained by considering these demand constraints.

The first upper bound is obtained by using Gilmore and Gomory's (1966) function. The second one was given by Viswanathan and Bagchi (1993) which represents an improvement to the first upper bound. The third one was given by Hifi (1997) which combines two different upper bounds: (i) the first one was obtained by using the result produced by Gilmore and Gomory's function and the value of each internal rectangle $g(R)$ (by considering the bounds b_i, $i = 1, \ldots, n$) and (ii) the second one was computed by applying dynamic programming procedures for solving the following bounded knapsack ($K_{\alpha\beta}$), for a given rectangle (α, β).

$$(K_{\alpha\beta})\begin{cases} V(\alpha\beta) = \max & \sum_{i\in S_{\alpha\beta}} c_i x_i \\ \text{subject to} & \sum_{i\in S_{\alpha\beta}} (l_i w_i) x_i \leq (\alpha\beta) \\ & x_i \leq \min\{b_i, \lfloor \alpha/l_i \rfloor \lfloor \beta/w_i \rfloor\} \\ & x_i \in IN,\ i \in S_{\alpha\beta} \end{cases}$$

where $S_{\alpha\beta}$ is the set of the rectangular pieces that can be cut from a (sub)rectangle (α,β), $\alpha \leq L$, $\beta \leq W$, x_i denotes the number of occurrences of the i-th piece in (α,β) and $V(\alpha\beta)$ the solution value of the problem $K_{\alpha\beta}$. The upper bound on the region P (its area) is computed easily at any node of the tree. Indeed, by solving the largest knapsack K_{LW} at the beginning of the algorithm by dynamic programming procedures, solutions for all $K_{\alpha\beta}$ are known and therefore also upper bounds for all (sub)rectangles (α,β).

We propose in this section some refinements to this upper bound. These refinements are simple but efficient, because we try to exploit, at each node, the remaining demand values of each considered pieces.

Let R be an internal rectangle and P be the unoccupied area in the initial rectangle. Clearly, the value

$$U_3(P) = V(LW) - g(R)$$

represents also an upper bound which can sometimes become better to $V(P)$.

Let b'_i be the number of occurrences of the i-th piece in R. Then the value

$$U_4 = \sum_{i|(l_i,\ w_i)\in \bar{S}} c_i(b_i - b'_i)$$

represents also an upper bound of $h(R)$, where $(l_i, w_i) \in \bar{S}$ if and only if $l_i \leq L - l_R$ or $w_i \leq W - w_R$ and $b_i - b'_i > 0$. Hence, the upper bound of the region P is obtained by combining the best upper bound given in Hifi (1997) and the two bounds U_3 and U_4.

3.2.2. Development strategy

The main difficulty encountered when we apply a best-first search strategy is the choice of the best candidate R achieving the best upper bound $f'(R)$. Generally, the number of generated nodes depends directly on the considered strategy. Return now to Box 1 (Line 1, the repeat loop) and remark that the strategy used is to select R from Open which achieves the greatest value, i.e.

$$R = \mathrm{Argmax}_{R_i \in Open}\{f'(R_1), f'(R_2), \ldots, f'(R_{|Open|})\}.$$

Let $\mathscr{L} = \{R_1, \ldots, R_t\}$ ($t \leq |Open|$) be the set of the candidate elements for which the value $f'(R)$ is reached.

Generally, we can distinguish four different strategies: (i) the *F-element* strategy which selects the first element R of the list $\mathscr{L}$, (ii) the *L-element* strategy which takes the latest element R_t as the candidate, (iii) the *R-element* strategy which chooses randomly an element from the global list $\mathscr{L}$ and (iv) the *G-element* strategy in which the candidate $R \in \mathscr{L}$ achieves the best feasible solution with value $g(R)$.

In the previous work (Hifi, 1997), the first strategy has been used and here we use the *G-element* strategy because it produces generally a good behavior to the new version of the algorithm.

3.3. *The* BOB *library*

The BOB library (Cung et al., 1997; Le Cun, Roucairol and TNN Team, 1995) is generally applied to facilitate the development of the branch-and-bound applications (min/max problems). This library has the double goal of allowing on the one hand the operational research and artificial intelligence communities (i) to implement their applications without worrying about the architecture of available machines and (ii) benefiting the advantages provided by the sequential and parallel approaches.

The BOB library is composed by four main user functions: `Bob_GenChild()`, `Bob_Init()`, `Bob_PrintSolution()` and `Bob_End()`. We have easily adapted these functions to the CTDC problem, principally, by reporting the existent MVB algorithm over the standard BOB library. Though an additional Clist list (that represents the closed lists of the algorithm), which does not belong to the classical branch-and-bound paradigm, is needed to keep in memory all the generated solutions, this has been done for sequential runs by simply adding a global data structure (two arrays of linked lists) as described in Section 3.3. With this coding, it is possible to run the program with all the search strategies and the data structures offered by the BOB library. The choice is as usual done at compilation time. For our test runs, we used the default options of BOB: *best-first* search and a *skew-heap* for the *global priority queue* (the Open list).

3.4. *A good representation of the list Clist*

In order to accelerate the algorithm we propose a new representation of one of the main lists, say Clist, used by the method.

We have already mentioned that Clist stores the best subproblems already found. At each step of the algorithm an element R from Open is chosen, transferred to Clist and combined with *some elements*, say $\mathscr{Q}$, of Clist (including R). In our study, we try to locate the subset $\mathscr{Q}$ of Clist without too much computational effort.

Indeed, we introduce a new *data structure* of the Clist list. It contains all patterns ordered on both non-decreasing order of lengths and widths (see Fig. 2). Of course, the interval of lengths (resp. widths) is limited by the dimensions (L, W) of the stock rectangle. So, by following the

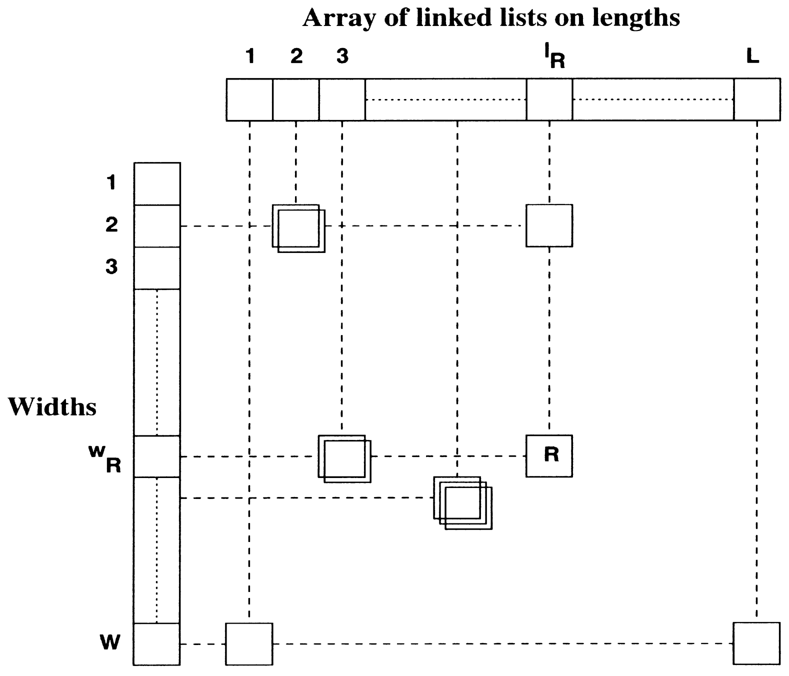


Fig. 2. The data structure of the extra list Clist.

schema of Fig. 2, we can distinguish all elements of $\mathcal{Q}$ as follows: let R be the candidate element chosen from Open and transferred to Clist; let $\mathcal{Q}_{l_R}$ and $\mathcal{Q}_{w_R}$ be the sets of the candidate elements of Clist for combinations with R, given as follows:

$$\mathcal{Q}_{l_R} = \{q \mid l_q = l_R + l_p \leq L,\ p \in \text{Clist}\} \text{ and } \mathcal{Q}_{w_R} = \{q \mid w_q = w_R + w_p \leq W,\ p \in \text{Clist}\}.$$

Then, $\mathcal{Q} = \mathcal{Q}_{l_R} \cup \mathcal{Q}_{w_R}$ is the subset of the valid candidates (authorized) of Clist.

In a previous work of Tschöke and Holthöfer (1995), the authors have proposed a manner for limiting some duplicates patterns. Their idea can be summarized as follows: let T be a new generated feasible pattern with dimensions (l_T, w_T). Consider that T_{b_i}, $i = 1, \ldots, n$, is the number of times that the i-th piece appears in T. Now, the procedure consists in examining all patterns of the list Clist and compared them to T. Finally, the pattern T is rejected if and only if there exist a pattern T' for which (i) $(l_T, w_T) \geq (l_{T'}, w_{T'})$ and (ii) $T_{b_i} \leq T'_{b_i}$, $i = 1, \ldots, n$.

Remark that for each created element T, the list Clist is already scanned and so, such a duplicate allows to reject some elements but it uses a lot of computational effort. We present in our study, three efficient duplicates pruning which needs small computational effort compared to the previous one. Furthermore, the detection of these duplicates is independent to the size of the main list Clist (which stores all best subproblems already found). This remark is not true when Tschöke and Holthöfer's (1995) duplicates are considered.

3.5. Detection of some duplicate patterns: symmetric-strategies

The goal of this section is to avoid some duplicate (symmetric) patterns produced by the construction process of the algorithm. We remind that in the previous work of Tschöke and Holthöfer (1995), the authors have suggested that for each pattern, a procedure was applied in order to detect an equivalent pattern throughout the Clist list. Their strategy may detect many duplicates but it uses a lot of computational effort. Indeed, if the current number of nodes (patterns) stored in the Clist list is $O(\rho)$, then the complexity of each iteration used by the procedure needs $O(\rho)$ operations.

In our study, we present different ways to detect duplicate patterns. For each node (pattern), we store information that represents its code. We define some rules to create the code of a pattern R obtained from the codes of two patterns, say A and B, which contribute to construct R. With these rules, some tests are defined in order to avoid a combination between two selected patterns which represent a duplicate (symmetric) pattern, i.e. we can neglect a construction between two patterns when we take into account the code of the patterns.

We start the following section by describing the main principle of the code of each pattern and we present, later, the different duplicate patterns used by our algorithm.

3.5.1. The pattern's code

Let A be a pattern obtained by combining vertically (resp. horizontally) a set of subpatterns A_1^v, A_2^v, ..., A_r^v (resp. A_1^h, A_2^h, ..., A_s^h), where A_i^v (resp. A_j^h) denotes a NV-pattern (resp. NH-pattern), i.e. there does not exist a horizontal (resp. vertical) cut on A_i^v (resp. A_j^h). Let l_A (resp. w_A) be the length (resp. width) of the pattern A.

On one hand, we assume that each pattern A has two identifiers denoted by I_A^v and I_A^h, where I_A^v (resp. I_A^h) represents the order of insertion of the NV-pattern (resp. NH-pattern) A in the Clist list. On the other hand, we assume that the identifiers of each pattern of the Open list are undefined.

Now, suppose that A is not a NV-pattern (resp. NH-pattern), then I_A^v (resp. I_A^h) is represented by the set of identifiers of the NV-pattern (resp. NH-pattern) which contribute to construct A, i.e. the set of the identifiers represented by $I_{A_1}^v$, $I_{A_2}^v$, ..., $I_{A_r}^v$ (resp. $I_{A_1}^h$, $I_{A_2}^h$, ..., $I_{A_s}^h$). We denote the number of distinct NV-patterns which contribute to produce A (which have different identifiers) by D_A^v (note that D_A^v represents also the cardinality of I_A^v) and we denote N_A^v the number of the same NV-patterns (with the same identifier) which contribute to construct A (clearly, in this case D_A^v is equal to 1).

Let R be the one obtained by combining horizontally two patterns A and B. Then, the code of R is obtained as follows (of course, the vertical case could be made symmetrically):

vertical: $I_R^v \leftarrow$ *undefined*; $D_R^v \leftarrow 1$ and $N_R^v \leftarrow 1$.
horizontal: $I_R^h \leftarrow I_A^v \cup I_B^v$; $D_R^v \leftarrow |I_R^h|$ the cardinality of D_R^v and if $D_R^v = 1$, then $N_R^v \leftarrow N_A^v + N_B^v$.

Example. Fig. 3 illustrates some generated patterns which are described in the following. We assume that all patterns of Fig. 3 have been inserted in the Clist list. In this case, the code corresponding to each produced pattern is computed as follows:

A initial piece (NH-pattern and NV-pattern): $D^v_A = D^h_A = 1$, $N^v_A = N^h_A = 1$, $I^v_A = I^h_A = 3$
B (NV-pattern but not a NH-pattern) vertical combination of two instances of the pattern A: $D^v_B = 1$, $N^v_B = 1$, $I^v_B = 5$, $D^h_B = 1$, $N^h_B = 2$, $I^h_B = 3$
C initial piece (NH-pattern and NV-pattern): $D^v_C = D^h_C = 1$, $N^v_C = N^h_C = 1$, $I^v_C = I^h_C = 4$
E (NH-pattern but not a NV-pattern) horizontal combination of A and C: $D^v_E = 2$, $N^v_E =$ *undef*, $I^v_E = 3, 4$, $D^h_E = 1$, $N^h_E = 1$, $I^h_E = 6$
F (NH-pattern but not a NV-pattern) horizontal combination of two instances of the pattern C: $D^v_F = 1$, $N^v_F = 2$, $I^v_F = 4$, $D^h_F = 1$, $N^h_F =$ *undef*, $I^h_F = 7$
G (NV-pattern but not a NH-pattern) vertical combination of two instances of pattern A: $D^v_G = 1$, $N^v_G =$ *undef*, $I^v_G = 8$, $D^h_G = 1$, $N^h_G = 2$, $I^h_G = 4$
H (NV-pattern but not a NH-pattern) horizontal combination of the patterns A and G: $D^v_H = 2$, $N^v_H =$ *undef*, $I^v_H = 3, 8$, $D^h_H = 1$, $N^h_H = 1$, $I^h_H = 9$
J (NV-pattern but not a NH-pattern) horizontal combination of the patterns A and F: $D^v_J = 2$, $N^v_J =$ *undef*, $I^v_J = 3, 7$, $D^h_J = 1$, $N^h_J = 1$, $I^h_J = 10$
K (NV-pattern but not a NH-pattern) vertical combination of two instances of the patterns E: $D^v_K = 1$, $N^v_K = 1$, $I^v_K = 11$, $D^h_K = 1$, $N^h_K = 2$, $I^h_K = 6$
M (NV-pattern but not a NH-pattern) horizontal combination of the patterns G and B: $D^v_M = 2$, $N^v_M =$ *undef*, $I^v_M = 5, 8$, $D^h_M = 1$, $N^h_M = 1$, $I^h_M = 12$

3.5.2. The dominated patterns: Type D1

Generally, a pattern can be produced by successive horizontal and vertical builds. Sometimes the dimensions of two different patterns can coincide and by using some translations on each of them, we can produce a pattern containing less number of pieces than the other one. In this case, it is necessary to locate if one of them is "dominated" by the other one. For example, in Fig. 3, the pattern H is dominated by the patterns K and M. Indeed, the following proposition shows that some patterns can be neglected if some conditions are satisfied.

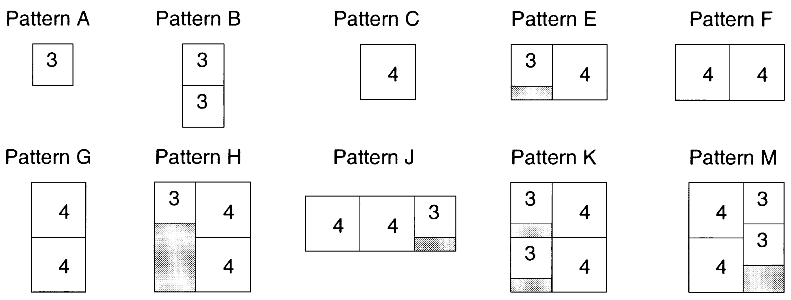


Fig. 3. Some constructed patterns with their codes.

Proposition 1. *Let A and B be two patterns. Suppose that R is a feasible pattern obtained using a horizontal build between A and B. Let b'_k, $k \in S = \{1, \ldots, n\}$, be the number of times the k-th piece is used in R. Then R is a dominated (duplicate) pattern if and only if*

$$\exists k \in S{:}b_k - b'_k > 0 \text{ and } (l_R, w_R) \geq (l_k, w_k)$$

where $w_R = w_A - w_B$ and $l_R = l_B$ if $w_A > w_B$, $w_R = w_B - w_A$ and $l_R = l_A$ otherwise.

Proof. Suppose that $w_R = w_A - w_B$ and $l_R = l_B$ and, k is the index of the candidate piece for which $(l_R, w_R) \leq (l_k, w_k)$. Recall that $g(A)$ and $g(B)$ denote respectively the *internal values* of A and B.

Now a vertical build between B and the k-th piece produces a pattern, say B', with internal value equal to $g(B') = g(B) + c_k$, where c_k is the profit of the candidate piece k. Another horizontal build between B' and A gives obviously a horizontal pattern, say R', with internal value equal to $g(R') = g(A) + g(B) + c_k$. Clearly, the two (different) patterns R and R' have the same dimensions, i.e. $(l_R, w_R) = (l'_R, w'_R)$. Furthermore, $g(R) \leq g(R')$, since

$$g(R) = g(A) + g(B) \leq g(A) + g(B') = g(A) + g(B) + c_k,$$

where $c_k > 0$. Hence, R is dominated by the pattern R' and so, R is a duplicate pattern.

Note that Proposition 1 treats only horizontal builds between two patterns A and B. Moreover, when we deal with vertical builds, we can also use the principle of Proposition 1 in order to eliminate other dominated patterns.

3.5.3. Symmetries (duplicates) on opposite directions: Type D2

The patterns K and M of Fig. 3 are symmetric, because they have the same size and they use exactly the same set of patterns (or initial pieces) and so, their costs are equal. The following proposition is a generalization to avoid the construction of this kind of symmetries.

Let A denote a pattern obtained by combining horizontally the NH-patterns $A_1, A_2, \ldots, A_r$ and B is a pattern obtained by combining horizontally the NH-patterns $B_1, B_2, \ldots, B_s$, with $d = l_A - l_B$ (we consider that $d \geq 0$).

Proposition 2. *A vertical build between two patterns A and B has a symmetric (duplicate) pattern if there exist two vectors $x = (x_i)_{i=1, \ldots, r}$ and $y = (y_j)_{j=1, \ldots, s}$ satisfying the following inequalities:*

$$\begin{cases} \sum_{i=1}^{r} l_{A_i} x_i - \sum_{j=1}^{s} l_{B_j} y_j \leq d \\ \sum_{i=1}^{r} x_i < r \text{ and } \sum_{j=1}^{s} y_j \leq s \\ x_i \in \{0, 1\} \text{ and } y_j \in \{0, 1\} \end{cases}$$

Proof. Let A' (resp. B') be a pattern obtained by combining the elements of type A_i for which $x_i = 1$ (resp. B_j such that $y_j = 1$). Let A'' (resp. B'') denote the pattern constructed by combining the remaining patterns of type A_i for which $x_i = 0$ (resp. B_j such that $y_j = 0$). Of course, if a pattern of type A_k, $k \in \{1, \ldots, r\}$ contributes to construct A', then A'' does not contain this pattern. Let $A|B$ (resp. A/B) be the pattern obtained by using a horizontal (resp. vertical) build between A and B.

Then, we can state that $R_1 = (A' \mid A'')/(B' \mid B'')$ (a vertical build) and $R_2 = (A' \mid A'')|(B' \mid B'')$ (a horizontal build) are symmetric, i.e. if we keep R_2, the second construction R_1 represents a duplicate pattern.

Now, it is sufficient to prove that R_1 and R_2 **(I)** contain the same (sub)patterns and **(II)** the dimensions of R_2 is smaller or equal to the dimensions of R_1, i.e. $(l_{R_1}, w_{R_1}) \geq (l_{R_2}, w_{R_2})$.

(I) — it is clear that R_1 and R_2 have the same (sub)patterns, since initially we only use the same set of patterns of types A_i, $i = 1, \ldots, r$, and B_j, $j = 1, \ldots, s$.

(II) — in order to prove that there exists a pattern R_1 which can represent a duplicate pattern it suffices to prove that its length is greater than or equal to the length of R_2 and its width is greater than or equal to the width of R_2.

Length — it is clear that $l_{R_1} = \max\{l_A, l_B\} = l_A$, since we suppose that $0 \leq l_A - l_B = d$. We remark that l_{R_2} is equal to $\max\{l_{A'}, l_{B'}\} + \max\{l_{A''}, l_{B''}\}$. We recall that we can always construct two subpatterns A' and B' which can verify the following inequality: $0 \leq l_{A'} - l_{B'} \leq d$. On one hand, since $(l_{A'} - l_{B'}) \leq d$, then $(l_{A'} - l_{B'}) = d - \epsilon$, where $\epsilon \geq 0$. On the other hand, we have

$$l_A - l_B = d \Leftrightarrow (l_{A'} + l_{A''}) - (l_{B'} + l_{B''}) = d \Leftrightarrow (l_{A'} - l_{B'}) + (l_{A''} - l_{B''}) = d,$$

so,

$$(l_{A''} - l_{B''}) = d - (d - \epsilon) \Leftrightarrow (l_{A''} - l_{B''}) = \epsilon \geq 0.$$

We deduce that $l_{A''} \geq l_{B''}$. In this case, we obtain

$$l_{R_2} = l_{A'} + \max\{l_{A''}, l_{B''}\} = l_{A'} + l_{B''} \leq l_{A'} + l_{A''} = l_{R_1}.$$

Hence, $l_{R_2} \leq l_{R_1}$.

Width — we have already mentioned that each pattern contains some information concerning the vertical and horizontal builds used on it, up to now.

In particular, for the pattern A, we have $w_A = \max\{w_{A_1}, w_{A_2}, \ldots, w_{A_r}\}$ and $w_B = \max\{w_{B_1}, w_{B_2}, \ldots, w_{B_2}\}$. Also, $w_{R_1} = w_A + w_B$ which means that $w_{R_1} = \max\{w_{A_1}, w_{A_2}, \ldots, w_{A_r}\} + \max(w_{B_1}, w_{B_2}, \ldots, w_{B_s}\}$.

On the other hand, for the pattern R_2, we have $w_{R_2} = \max\{w_{A'/B'}, w_{A''/B''}\} \Longrightarrow w_{R_2} = \max\{w_{A'} + w_{B'}, w_{A''} + w_{B''}\}$, since $w_{A'/B'} = w_{A'} + w_{B'}$ and $w_{A''/B''} = w_{A''} + w_{B''}$.

It is clear that $w_{A'} \leq w_A$, $w_{A''} \leq w_A$, $w_{B'} \leq w_B$ and $w_{B''} \leq w_B$ and so, $w_{R_2} \leq w_A + w_B$. Hence $w_{R_2} \leq w_{R_1}$.

Remark 4. We recall that A is composed by r patterns. Let $\mathscr{T}$ be the set of linear combinations between each $r - 1$ patterns of A, i.e. each linear combination of $\mathscr{T}$ is obtained by combining horizontally $r-1$ patterns of A throughout the r patterns. Let $T_i \in \mathscr{T}$ be a linear combination with the length l_{T_i}. Now, if we find all duplicate (symmetries) patterns, it is sufficient to prove that for each construction between A and B there does not exist a duplicate pattern. This means that there does not exist a vector y which satisfies $l_{T_i} - d \leq \sum_{j=1}^{s} l_{B_j} y_j \leq l_{T_i}$ (obtained by replacing the length of the pattern T_i in the first equation given by proposition 2). This is equivalent to enumerate all possibilities of combinations between the elements of $\mathscr{T}$ and all components of B, which represents an exponential enumeration.

To avoid this exponential growth, in our study we use a more simpler procedure in order to eliminate only some constructions. We have adopted the following principle: let A and B be two patterns and suppose that we have the following inequality $0 \leq A_i - B_j \leq d$, where A_i and B_j represent respectively the subpatterns of A and B, stored in the Clist list according to their lengths. In this case, we forbid the vertical build between the previously patterns A and B.

3.5.4. Symmetries on the same direction: Type D3

The third test is applied in order to discard other forms of duplicate patterns. It consists in rejecting some constructions obtained by combining vertically (resp. horizontally) two patterns, say A and B. In this case, we suppose that each pattern A and B are composed by (different) subpatterns combined vertically (resp. horizontally). For example, the pattern J of Fig. 3 could be constructed by combining horizontally the pattern H and the pattern A, but also by combining horizontally the pattern E and the pattern C.

The following proposition treats only the horizontal case and the vertical case could be made in the same way.

Proposition 3. *The horizontal build between two patterns A (taken from Open) and B (taken from Clist) is discarded if one of the three following cases are verified:*

1. $D_A^h = 1$ *and* $D_B^h = 1$ *and* $I_A^h = I_B^h$ *and* ($N_A^h - N_B^h < -1$ *or* $N_A^h - N_B^h > 1$).
2. $D_A^h = 1$ *and* $D_B^h \neq 1$ *and* $I_A^h \subset I_B^h$.
3. $D_A^v \neq 1$.

Proof. We consider two classes of patterns which represent all generated patterns. For each class, we prove that there exists two patterns A and B which can produce a new pattern R (using a horizontal build) which is not discarded by the previous tests of Proposition 3.

1. The case $D_R^h = 1$: the obtained pattern R is produced by combining several subpatterns composed by only one rectangle (pattern). In this case, we distinguish the two following cases:
   1.1. $N_R^h = 1$: this case has been already proved.
   1.2. $N_R^h \neq 1$: in this case, there exists two patterns A and B for which we have $N_R^h = N_A^h + N_B^h$, with $-1 \leq N_A^h - N_B^h \leq 1$. Such a pattern R is constructed because it is easy to see that A and B can be constructed by recurrence.

2. The case $D_R^h \neq 1$: it is sufficient to see that there always exist two patterns A and B such that $I_R^h = I_A^h \cup I_B^h$ with $D_A^h = 1$ and $I_A^h \not\subset {}_B^h$.

Let us notice that Proposition 3 allows to eliminate lots of useless branches of the developed tree (cf Section 4.1), but we think that an interesting problem is to add some other efficient and competitive strategies in order to reject some more complex combinations. Of course, this problem remains open.

4. Computational results and future works

All approaches were coded in `c` and tested on a `UltraSparc1` (145 Mhz, 64 Mbytes of RAM, under Solaris 2.5 with CPU time limited to 2 hours and virtual memory limited to 300 Mbytes) or a DEC Alpha (processor 21164, 500 Mhz and 1 Gbytes of RAM). This section is divided into two parts. First, we present empirical evidence of the performance of the new version. In the second part we discuss some future works of the proposed approaches.

4.1. Computational results

We compare in this section the performance of the new version of the algorithm, denote NMVB-`BOB` to the last published one executed by applying the characteristics of the `BOB` library (denoted MVB-`BOB`). Our computational study was conducted on 36 problem instances with different sizes and densities. The density of each instance depends on the number of linear combinations of the sets P_L and P_W. It means that the complexity increases if the dimensions of some pieces are very small compared to the dimensions of the initial stock rectangle. All test problem instances are publicly available from `ftp://www.panoramix.paris1.fr/pub/CERMSEM/hifi/2Dcutting`.

The performance of the new version of the algorithm is evaluated on both *weighted* and *unweighted* CTDC stock problems. We have considered some problem instances of the literature and other randomly generated ones. For the random instances, the dimensions l_i and w_i of pieces to cut are taken uniformly from the intervals [0.1L, 0.75L] and [0.1W, 0.75W] respectively. The weight associated to a piece i is computed by $c_i = \lceil \rho l_i w_i \rceil$, where $\rho = 1$ for the unweighted case and $\rho \in [0.25, 0.75]$ for the weighted case. The constraints b_i, for $i = 1, \ldots, n$, have been chosen such that $b_i = \min\{\rho_1, \rho_2\}$, where $\rho_1 = \lfloor L/l_i \rfloor \lfloor W/w_i \rfloor$ and ρ_2 is a number randomly generated in the interval [1, 10].

On one hand, we have considered 27 problem instances of medium sizes and nine large-scale problem instances, one the other hand. For the *weighted version* (see Table 1), we have considered ten problem instances. Four instances (2–3 and A1–A2) are taken from Hifi (1997). Two other instances `STS2–STS4` are taken from Tschöke and Holfhöfer (1995) (representing the second and the last instances of table 6 of the appendix). We have also considered four randomly generated problem instances of `CHL1–CHL4`.

For the *unweighted version* (see Table 1), we have considered 17 problem instances. The instances `2 s–3 s`, `A1 s–A2 s`, `STS2 s–STS4 s` and `CHL1 s–CHL4 s` represent exactly the instances `2–3`, `A1–A2`, `STS2–STS4` and `CHL1–CHL4`, respectively for which the profit of each piece is represented by its area. Four other instances `A3–A5` and `HH` are taken from Hifi (1997) and finally, we have also considered three randomly generated problem instances `CHL5–CHL7`.

In Table 1, for each problem instance we report:

- its dimensions $L \times W$ and n the number of types of pieces to cut;
- the Upper Bound (UB) obtained at the root node, the optimal solution value (Opt), the Lower Bound (case: LB) obtained when all linear combinations (of the two sets P_L and P_W) are considered and the Halved-Lower Bound (case: HLB) when the two sets P_L and P_W are halved;

Table 1
Representation of the lower bound quality (a) when all linear combinations are used (denoted LB) and (b) when the two sets of points are halved (denoted HLB)[a]

#Instance	$L \times W$	n	UB	Opt	Initial lower bound			Initial heuristic lower bound		
					LB	EAR	T (s)	HLB	EAR	T(s)
2	40 × 70	10	2919	2892	2731	0.94	0.85	2614	0.90	0.60
3	40 × 70	20	2020	1860	1740	0.93	2.19	1740	0.93	1.46
A1	50 × 60	20	2140	2020	1860	0.92	2.21	1860	0.92	1.58
A2	60 × 60	20	2705	2505	2395	0.96	2.40	2395	0.96	1.81
STS2	55 × 85	30	4790	4620	4620	1.00	8.15	4620	1.00	5.81
STS4	99 × 99	20	9954	9700	9505	0.98	7.85	9505	0.98	5.78
CHL1	132 × 100	30	9009	8671	8222	0.94	20.30	8222	0.94	15.35
CHL2	62 × 55	10	2432	2326	2240	0.96	0.85	2240	0.96	0.61
CHL3	157 × 121	15	5283	5283	5283	1.00	11.18	5283	1.00	6.75
CHL4	207 × 231	15	8998	8998	8998	1.00	25.18	8998	1.00	15.68
2 s	40 × 70	10	2800	2778	2626	0.94	0.85	2626	0.94	0.59
3 s	40 × 70	20	2743	2721	2623	0.96	2.28	2623	0.96	1.52
A1 s	50 × 60	20	2950	2950	2950	1.00	2.25	2950	1.00	1.57
A2 s	60 × 60	20	3576	3535	3445	0.97	2.54	3445	0.97	1.92
STS2 s	55 × 85	30	4675	4653	4625	0.99	8.47	4625	0.99	5.99
STS4 s	99 × 99	20	9785	9770	9502	0.97	7.83	9481	0.97	5.77
CHL1 s	132 × 100	30	13,200	13,099	12,830	0.98	21.21	12,830	0.98	15.33
CHL2 s	62 × 55	10	3410	3279	3244	0.99	0.86	3244	0.99	0.61
CHL3 s	157 × 121	15	7402	7402	7402	1.00	11.19	7402	1.00	6.71
CHL4 s	207 × 231	15	13,932	13,932	13,932	1.00	24.94	13,932	1.00	15.46
A3	70 × 80	20	5562	5451	5348	0.98	4.21	5348	0.98	3.21
A4	90 × 70	20	6300	6179	5951	0.96	5.04	5951	0.96	3.63
A5	132 × 100	20	13,174	12,985	12,710	0.98	9.75	12,710	0.98	6.91
HH	127 × 98	5	12,348	11,586	11,391	0.98	1.04	11,391	0.98	0.84
CHL5	20 × 20	10	400	390	361	0.93	0.20	361	0.93	0.12
CHL6	130 × 130	30	16,900	16,869	16,492	0.98	25.99	16,492	0.98	19.54
CHL7	130 × 130	35	16,900	16,881	16,527	0.98	36.08	16,527	0.98	24.65

[a] UB represents the upper bound obtained at the root node.

- the experimental approximation ratio (EAR = (LB (or HLB))/*Opt*) and the required computational time (*T* (s)) of the lower bound.

For all treated instances, excellent quality lower bounds are obtained. We remark that lower bounds HLB (when we use some linear combinations) are generally close to the values given by LB (which uses all linear combinations), but the average required time by the HLB case is less important than the one given by the LB case. Note that, for the HLB case, the average approximation ratio is 0.97 and the average computational time is 6.29 s. Remark also that the approximation ratio varies between 0.90 (instance 2) and 1 (instances STS2, CHL3, CHL4, A1 s, CHL3 s and CHL4 s). Therefore, by including these lower bounds in the general tree search procedure (with good quality), one expects also a good behavior of the resulting

Table 2
Performance of the new version of the algorithm compared to the standard MVB-BOB version[a]

#Instance	Opt.	MVB-BOB		MVB-BOB^{Clist}		NMVB-BOB^{0cut}		NMVB-BOB*	
		T (s)	GT (s)	*T* (s)	GT (s)	*T* (s)	GT (s)	*T* (s)	GT (s)
2	2892	0.17	1.10	0.16	0.24	0.07	0.17	0.06	0.66
3	1860	3.68	3.81	0.42	0.55	0.15	0.28	0.15	1.61
A1	2020	2.01	2.17	0.51	0.67	0.17	0.33	0.15	1.73
A2	2505	65.41	65.55	2.29	2.45	0.81	0.97	0.78	2.59
STS2	4620	20.68	20.97	10.21	10.49	3.04	3.33	3.10	8.91
STS4	9700	37.50	37.97	23.70	24.17	3.30	3.77	2.72	8.50
CHL1	8671	4318.38	4319.23	3193.08	3193.94	90.50	91.36	24.31	39.66
CHL2	2326	0.02	0.12	0.02	0.21	0.02	0.13	0.02	0.63
CHL3	5283	N/A	N/A	N/A	N/A	N/A	N/A	< 0.01	6.76
CHL4	8998	N/A	N/A	N/A	N/A	N/A	N/A	< 0.01	15.69
2 s	2778	0.21	0.30	0.19	0.30	0.05	0.15	0.05	0.64
3 s	2721	< 0.01	0.13	< 0.01	0.13	< 0.01	0.13	< 0.01	1.53
A1 s	2950	< 0.01	0.14	< 0.01	0.14	< 0.01	0.14	< 0.01	1.58
A2 s	3535	< 0.01	0.18	< 0.01	0.16	< 0.01	0.16	< 0.01	1.93
STS2 s	4653	0.06	0.35	0.05	0.34	0.09	0.37	0.03	6.02
STS4 s	9770	0.16	0.67	0.10	0.58	0.09	0.60	0.06	5.83
CHL1 s	13,099	16.09	17.00	7.79	8.68	1.83	2.72	1.17	16.50
CHL2 s	3279	0.08	0.18	0.05	0.16	0.02	0.12	0.02	0.63
CHL3 s	7402	N/A	N/A	N/A	N/A	N/A	N/A	< 0.01	6.72
CHL4 s	13,932	N/A	N/A	N/A	N/A	N/A	N/A	< 0.01	15.47
A3	5451	0.66	0.91	0.42	0.67	0.22	0.47	0.15	3.36
A4	6179	2.72	3.01	2.26	2.55	0.65	0.95	0.57	4.20
A5	12,985	25.14	25.79	7.88	8.54	0.80	1.45	0.70	7.61
HH	11,586	5.79	6.13	1.94	2.28	0.16	0.50	0.17	1.01
CHL5	390	0.50	0.54	0.15	0.19	0.04	0.07	0.04	0.16
CHL6	16,869	0.65	1.79	0.77	1.92	0.31	1.46	0.19	19.73
CHL7	16,881	1.60	2.90	1.22	2.51	0.78	2.08	0.49	25.04

[a] N/A means that the solution is Not Available, i.e. this version is not able to produce the optimal solution after 2 h.

algorithm. We recall that the initial lower and (internal) upper bounds are combined together in order to reduce the search (sub)space at each node of the developed tree.

We examine now the performance of the new version of the algorithm. In order to evaluate the effect of each strategy used, we have considered different versions of the new algorithm. Five versions of the algorithm are considered:

1. the first version is represented by the MVB algorithm using the characteristics of the BOB library, denoted MVB-BOB (see Tables 2 and 3).

   We should point out that limited experiments have shown that MVB-BOB is actually 1.3 time faster than the MVB in Hifi (1997). This is due to the effectiveness of the BOB library in terms of data structure and implementation;

Table 3
The number of generated nodes produced by the different versions of the algorithm[a]

#Instance	MVB-BOB		MVB-BOB^{Clist}		NMVB-BOB^{0cut}		NMVB-BOB*	
	#Evl. nd.	#Not exp.	#Evl. nd.	#Not exp.	#Evl. nd.	#Not exp.	#Evl. nd.	#Not exp.
2	4843	12,975	4818	6679	729	1313	684	2187
3	2139	614,120	2034	59,150	667	16,185	603	16,395
A1	7378	336,866	7363	50,839	1356	10,456	1041	11,008
A2	11,590	8,090,754	11,758	344,350	3345	87,112	3155	88,629
STS2	3261	4,937,077	3261	1,624,159	1417	461,227	1417	461,227
STS4	94,009	7,690,302	93,953	2,578,235	14,746	350,764	8634	357,424
CHL1	227,725	113,012,273	227,320	18,150,278	29,156	1,064,913	20,955	1,113,988
CHL2	578	5215	577	2007	300	1131	164	1263
CHL3	N/A	N/A	N/A	N/A	N/A	N/A	1	1
CHL4	N/A	N/A	N/A	N/A	N/A	N/A	1	1
2 s	5629	15,082	5325	7691	617	1131	528	2012
3 s	93	543	95	182	76	157	45	155
A1 s	25	39	25	29	25	29	1	1
A2 s	58	339	58	193	54	189	49	190
STS2 s	1403	6550	1445	3729	969	2523	148	2785
STS4 s	2688	28,376	2407	8709	984	3353	634	3134
CHL1 s	81,571	3,275,526	81,327	730,446	13,213	71,895	5236	92,271
CHL2 s	380	36,723	379	10,347	167	2877	132	2893
CHL3 s	N/A	N/A	N/A	N/A	N/A	N/A	1	1
CHL4 s	N/A	N/A	N/A	N/A	N/A	N/A	1	1
A3	4865	138,327	4925	49,060	1718	15,467	863	16,384
A4	22,778	325,174	23,013	148,643	5038	36,486	4066	37,267
A5	24,346	4,698,428	24,472	939,417	3816	84,503	2533	87,745
HH	18,468	946,276	18,405	217,973	1308	12,982	1297	13,456
CHL5	2633	88,949	2919	11,516	391	1312	353	1516
CHL6	11,590	73,172	12,495	44,615	2997	8128	1560	10,179
CHL7	23,460	186,095	22,291	51,741	6943	18,723	3340	28,558

[a] N/A means that the number of nodes is not available.

Table 4
Effect of the duplicate patterns on the new version of the algorithm

#Instance	Type D1 ♯nodes	Type D2 ♯nodes	Type D3 ♯nodes	Global ♯nodes	NMVB-BOB* ♯nodes	NMVB-BOB** ♯nodes	Reduction δ
2	92	499	478	1069	684	3199	−78.62
3	219	73	451	743	603	1837	−61.17
A1	375	175	562	1112	1041	4486	−76.79
A2	871	208	1470	2549	3155	11,064	−71.48
STS2	67	0	1017	1084	1417	3261	−56.55
STS4	1441	96	6278	7815	8634	55,284	−84.38
CHL1	4009	3722	19,679	27,410	20,955	188,269	−88.87
CHL2	10	0	42	52	164	233	−29.61
CHL3	0	0	0	0	1	1	0.00
CHL4	0	0	0	0	1	1	0.00
2 s	72	149	275	496	528	2542	−79.23
3 s	4	0	10	14	45	61	−26.23
A1 s	0	0	0	0	1	1	0.00
A2 s	0	0	3	3	49	52	−5.77
STS2 s	3	0	8	11	148	180	−17.78
STS4 s	52	47	179	278	634	1616	−60.77
CHL1 s	596	0	4248	4844	5236	20,825	−74.86
CHL2 s	12	0	76	88	132	320	−58.75
CHL3 s	0	0	0	0	1	1	0.00
CHL4 s	0	0	0	0	1	1	0.00
A3	126	0	379	505	863	2575	−66.48
A4	630	237	2619	3486	4066	16,629	−75.55
A5	226	3	1991	2220	2533	15,840	−84.00
HH	328	1067	1077	2472	1297	17,393	−92.54
CHL5	106	411	290	807	353	2291	−84.59
CHL6	76	0	454	530	1560	2931	−46.78
CHL7	341	0	1236	1577	3340	5224	−36.06
Average	357.63	247.67	1586.00	2191.30	2209.31	13,696.81	−83.87

2. the second version of the algorithm, denoted MVB-BOB^{Clist}, represents the MVB-BOB with the new data structure of the Clist list. This version has been introduced to compare the effect of the new data structure on the standard MVB-BOB algorithm;
3. the third version, denoted NMVB-BOB^{0cut}, represents an extension of the MVB-BOB^{Clist} algorithm in which the following strategies are considered: (i) the *initial lower bound* developed in Hifi, 1997; Hifi and Zissimopoulos, 1997 as the first duplicate pruning, (ii) the *improved upper bounds* at internal nodes and (iii) the *three other duplicates pruning* (types D1–D3) (see Tables 2 and 3);
4. the fourth version, denoted NMVB-BOB*, represents the NMBV-BOB^{0cut} with the *improved initial lower bound* denoted by HLB (see Tables 1–3);
5. the last version of the algorithm, denoted by NMVB-BOB**, represents the NMVB-BOB*

applied without using the three duplicate patterns (see Table 4). This version is used in order to examine th effect of the symmetric-strategies.

In Table 2 for each version, we report its required search time (T (s)) and the global time (GT (s)) which represents the time consumed by the initial phase and the search process, and in Table 3 we report the number of evaluated nodes (♯Evl. nd.) and the number of nodes not explored by the search process (♯Not. exp.).

In Table 2 we observe that MVB-BOB^{Clist} performs better than the first version MVB-BOB algorithm and "generally" the number of evaluated nodes is equal (see Table 3).

Let us notice that during the runs, the MVB-BOB and the MVB-BOB^{Clist} versions may swap if the amount of memory used is over 64 Mo. This is the case for CHL1 (see Tables 2 and 3). The unusually large amount of time spent by the two algorithms is explained by this memory swap phenomenon. For example, on CHL4, this excessive time consuming prevents the MVB-BOB and MVB-BOB^{Clist} algorithms to find the optimal solution within the given 2 h limit. The swap phenomenon did not occur for the NMVB-BOB* algorithm because the number of evaluated and expanded nodes are significantly less than those of MVB-BOB and MVB-BOB^{Clist}.

We also observe in Table 2 that the third version (NMVB-BOB^{0cut}) outperforms the fourth version (NMVB-BOB*) when we treat some medium problems, because the initial phase of NMVB-BOB^{0cut} is less time consuming. But, the NMVB-BOB* algorithm is largely more efficient when the complexity of the instance increases. For example, if we consider the instances CHL3 and CHL4, then we remark that the NMVB-BOB* algorithm gives the optimal solutions at the root node and the NMVB-BOB^{0cut} algorithm fails to produce the optimal solutions after two hours. Another example is the instance CHL1 for which the NMVB-BOB^{0cut} produces the optimal solution using 91.36 s and 39.66 s when the NMVB-BOB* is applied. Furthermore, for the same instance NMVB-BOB^{0cut} and NMVB-BOB* algorithms generate 29,156 and 20,955 nodes respectively, which represent a percentage reduction equal to 28.13%.

In Table 2, when we look at the columns 7 and 9 (which represent the computational time consumed by the search process for each version of the algorithm), we can remark that the values of column 9 are "generally" less than the values of column 7. This can be explained by the importance of the efficiency of lower bounds at the root node, especially on the medium and large problem instances. Moreover, it is worth noticing the time gain in relation to instances size. For increasing instances size we obtain better lower bounds whereas the computational gain of the search process becomes more important.

Consequently, when we deal with the NMVB-BOB* version, we have two alternatives for pruning branches: (i) the improved lower and upper bounds which are more important on high levels of the developed tree and (ii) the effect produced by using the three propositions (duplicates of type D1–D3) which are important at each level of the tree.

In order to evaluated the effect of each of the algorithmic duplicate patterns in the NMVB-BOB* algorithm, we have compared the number of generated nodes of this version to the same version but without using the three duplicate patterns D1–D3. In Table 4, we have reported:

1. the number of nodes (♯nodes) removed by each type of duplicate patterns (columns 1, 2 and 3);
2. the global nodes rejected when all duplicate patterns are used (column 4);

3. the number of generated nodes when the NMVB-BOB* algorithm is considered (column 5);
4. the number of generated nodes when the NMVB-BOB** algorithm is applied (column 6);
5. the percentage reduction for the number of generated nodes (the last column). This percentage is computed by $\delta = ((\text{NMVB-BOB}^*(I) - \text{NMVB-BOB}^*(I)^*)/\text{NMVBBOB}^{**}(I)) \times 100$, where I denotes an instance of the problem.

Let us notice that the value $\delta < 0$ means that the first version NMVB-BOB* rejects $-\delta\%$ of nodes with respect to the nodes generated by the NMVB-BOB** algorithm. In Table 4, it can be seen that each duplicate pattern has a substantial contribution. Let us note that, concerning all duplicate pruning, our strategy was applied as follows: the initial solution operates directly on the list Open and D1–D3 are applied successively when an element is selected from Open and transferred to Clist.

By using this strategy, we can see that the duplicate of type D1 rejects 16.32% on average of the global removed nodes, type D2 deleted 11.30% and type D3 removes 72.38% on average. Remark also when they are applied together, they provide a significant speed improvement (by rejecting 83.87% of the average global generated nodes).

Finally, we can conclude that the NMVB-BOB^{0cut} is very efficient for small and medium problem instances and so, we recommend to apply this version for the class of instances with small and medium sizes. On the other hand, we can remark that the NMVB-BOB^{0cut} becomes less efficient when we consider the large problem instances and so, we recommend to use the NMVB-BOB* algorithm when we try to solve the class of large-scale problem instances.

In order to evaluate the effectiveness of the new version of the algorithm, i.e. the NMVB-BOB* algorithm, we have considered nine large and more complex instances generated as the previous instances. The different versions of the algorithm were not able to solve these instances on UltraSparc1 with CPU time limited to two hours and memory limited to 300 Mbytes). For this reason, we decided to execute the different instances on a DEC workstation with an Alpha 21164 processor, 500 Mhz and memory limited to 1 Gbytes. Table 5 presents the computational performances of the best version of the algorithm. With larger amount of memory and a faster processor, these instances have been solved in reasonable times.

Table 5
Performance of the NMVB-BOB* algorithm on more complex problem instances

#Instance	$L \times W$	n	Opt	HLB	UB	EAR	#Ev. nd.	#Not. exp.	GT (s)
Hchl1	130×130	30	11,303	10,708	11,866	0.95	657,425	547,102,974	32,383.95
Hchl2	130×130	35	9954	9462	10,263	0.95	184,352	83,367,248	1204.61
Hchl3 s	127×98	10	12,215	11,800	12,405	0.97	12,454	1,083,483	16.40
Hchl4 s	127×98	10	11,994	11,532	12,405	0.96	132,818	31,167,891	634.95
Hchl5 s	205×223	25	45,361	44,118	45,666	0.97	272,319	6,148,437	69.03
Hchl6 s	253×244	22	61,040	59,687	61,346	0.98	1744	31,380	8.03
Hchl7 s	263×241	40	63,112	61,837	63,330	0.98	4626	138,700	22.30
Hchl8 s	49×20	10	911	825	974	0.91	73,126	1,982,344	104.97
Hchl9	65×76	35	5240	5030	5370	0.96	223,117	59,387,559	1115.05

4.2. Future works

Actually, the efficient and optimal solutions of some medium (and large) combinatorial optimization problems is highly important for many applications in the field of science and engineering. Using today's sequential algorithmic approaches, sometimes it is not possible to solve some of these instances. Indeed, one solution that seems to be attractive is to design parallel algorithms to solve exactly some large-scale CTDC problems. The use of parallel approaches may increase the problem size which can be solved efficiently.

We think that there is an important direction of research. We have already mentioned that the proposed approach uses a main list, denoted Clist. Generally, it contains the best subproblems constructed by using some vertical and horizontal builds. So, an interesting question is how we can distribute all elements of the Clist list through a set of processors, where, on one hand, for each processor a subset of the best subproblems is assigned and each subset is managed independently on the other hand. In this case, we also think that the effect of the parallelism largely influence the memory space used.

5. Conclusions

We have considered the modified version of Viswanathan and Bagchi's algorithm proposed in Hifi (1997). This algorithm is one of the best exact algorithms known today for the constrained two-dimensional cutting stock problem.

In this paper, we proposed a new version of the algorithm which outperforms the above one. The power of the resulting algorithm is based on the following new proposals we presented:

1. an efficient approximate solution is applied at the root node of the search tree (initial lower bound is obtained by using an extension of the approximate algorithm developed in Hifi, 1997; Hifi and Zissimopoulos, 1997).
2. different refined upper bounds are used at internal nodes;
3. three new symmetric strategies are introduced to reject some nodes and to curtail the search in the developed tree;
4. an implementation using the generic Branch-and-Bound BOB library allows a Best-First technique for selecting the best suspended path;
5. a new representation of the Clist data structure is used in order to speed up the construction of new configurations.

Comparing to the algorithm presented in Computers and Operations Research (Hifi, 1997), this new algorithm achieves 50% in average better and it is able to solve Hard-Problem instances.

References

Albano, A.A., 1977. A method to improve two-dimensional layout. Comp. Aid. Des. 9, 48–52.

Albano, A., Sapuppo, G., 1980. Optimal allocation of two-dimensional irregular shapes using heuristic search methods. IEEE Trans. Sys. Man. Cyb. 10 (5), 242–248.

Beasley, J.E., 1985. Algorithms for unconstrained two-dimensional guillotine cutting. Journal of the Operational Research Society 36, 297–306.

Christofides, N., Whitlock, C., 1977. An algorithm for two-dimensional cutting problems. Operations Research 25, 31–44.

Cung, V.-D., Dowaji, S., Le Cun, B., Mautor, T., Roucairol, C., 1997. Concurrent data structures and load balancing strategies for parallel branch-and-bound/A* algorithms. DIMACS Series in Discrete Mathematics and Theoretical Computer Science 30, 141–161.

Dyckhoff, H., 1990. A typology of cutting and packing problems. European Journal of Operational Research 44, 145–159.

Dyckhoff, H., Finke, U., 1992. Cutting and packing in production and distribution: typology and bibliography. Springer–Verlag Co, Heildelberg.

Dyson, R.G., Gregory, A.S., 1974. The cutting stock problem in the flat glass industry. Opnl. Res. Quart. 25, 41–53.

Fayard, D., Zissimopoulos, V., 1995. An algorithm for solving unconstrained two-dimensional knapsack problems. European Journal of Operational Research 84, 618–632.

Fayard, D., Hifi, M., Zissimopoulos, V., 1998. An efficient approach for large-scale two-dimensional cutting stock problems. Journal of the Operational Research Society 49, 1270–1277.

Gilmore, P., Gomory, R., 1966. The theory and computation of knapsack functions. Operations Research 14, 1045–1074.

Haessler, R.W., 1971. A heuristic programming solution to a nonlinear cutting stock problem. Management Science 17B, 793–802.

Haessler, R.W., 1975. Controlling cutting pattern changes in one-dimensional trim problems. Operations Research 23, 483–493.

Hahn, S., 1967. On the optimal cutting of defective glass sheets. IBM, New York Scientific Center, Report No. 320-2916.

Herz, J.C., 1972. A recursive computing procedure for two-dimensional stock cutting. IBM Journal of Research and Development 16, 462–469.

Hifi, M., 1994. Study of some combinatorial optimization problems: cutting stock, packing and set covering problems. PhD thesis, University of Paris 1 Pantheon-Sorbonne.

Hifi, M., 1997. An improvement of Viswanathan and Bagchi's exact algorithm for cutting stock problems. Computers and Operations Research 24 (8), 727–736.

Hifi, M., Ouafi, R., 1997. Best-first search and dynamic programming methods for cutting problems: the cases of one or more stock plates. Computers and Industrial Engineering 32 (1), 187–205.

Hifi, M., Zissimopoulos, V., 1996. A recursive exact algorithm for weighted two-dimensional cutting. European Journal of Operational Research 91, 553–564.

Hifi, M., Zissimopoulos, V., 1997. Constrained two-dimensional cutting: an improvement of Christofides and Whitlock's exact algorithm. Journal of the Operational Research Society 48, 324–331.

Le Cun, B., Roucairol, C., TNN Team, 1995. BOB: a unified platform for implementing branch-and-bound like algorithms. Technical report ♯95/106, PR*i*SM, Université de Versailles/St-Quentin-en-Yveline, 78035, Versailles Cedex, France.

Morabito, R., Arenales, M., 1996. Staged and constrained two-dimensional guillotine cutting problems: An and/or-graph approach. European Journal of Operational Research 94 (3), 548–560.

Morabito, R., Garcia, V., 1998. The cutting stock problem in hardboard industry: a case study. Computers and Operational Research 25, 469–485.

Oliveira, J.E., Ferreira, J.S., 1990. An improved version of Wang's algorithm for two-dimensional cutting problems. European Journal of Operational Research 44, 256–266.

Sweeney, P., Paternoster, E., 1992. Cutting and packing problems: a categorized application-oriented research bibliography. Journal of the Operational Research Society 43 (7), 691–706.

Tschöke, S., Holthöfer, N. 1995. A new parallel approach to the constrained two-dimensional cutting stock problem. In: Proceedings of the Second International Workshop, Parallel Algorithms for Irregularly Structured Problems, LNCS 980, pp. 285–300.

Vasko, F.J., 1989. A computational improvement to Wang's two-dimensional cutting stock algorithm. Computers and Industrial Engineering 16, 109–115.

Viswanathan, K.V., Bagchi, A., 1993. Best-first search methods for constrained two-dimensional cutting stock problems. Operations Research 41 (4), 768–776.

Wang, P.Y., 1983. Two algorithms for constrained two-dimensional cutting stock problems. Operations Research 31 (3), 573–586.

Zissimopoulos, V., 1985. Heuristic methods for solving (un)constrained two-dimensional cutting stock problems. Methods of Operations Research 49, 345–357.