ELSEVIER

European Journal of Operational Research 91 (1996) 553-564

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

Theory and Methodology

A recursive exact algorithm for weighted two-dimensional cutting

M. Hifi [a], V. Zissimopoulos [b,*]

[a] *CERMSEM, Université de Paris 1 - Panthéon-Sorbonne, 90 rue de Tolbiac, 75634 Paris Cedex 13, France*

[b] *L.R.I., URA 410 CNRS, Université de Paris Sud, Centre d'Orsay, 91405 Orsay, France*

Abstract

Gilmore and Gomory's algorithm is one of the better actually known exact algorithms for solving unconstrained guillotine two-dimensional cutting problems. Herz's algorithm is more effective, but only for the unweighted case. We propose a new exact algorithm adequate for both weighted and unweighted cases, which is more powerful than both algorithms. The algorithm uses dynamic programming procedures and one-dimensional knapsack problem to obtain efficient lower and upper bounds and important optimality criteria which permit a significant branching cut in a recursive tree-search procedure. Recursivity, computational power, adequateness to parallel implementations, and generalization for solving constrained two-dimensional cutting problems, are some important features of the new algorithm.

Keywords: Knapsack; Two-dimensional cutting; Dynamic programming; Exact algorithms; Heuristics; Recursivity

1. Introduction

The two-dimensional cutting (TDC) problem is a generalization of the well-known one-dimensional knapsack and it turns out to be surprisingly common in applications [4,6,12,13]. The problem consists of cutting a given finite set of rectangular pieces into a rectangle of fixed dimensions with minimum wastage or maximum profit. Among obvious examples of applications are the production of glass, metal sheets, leather or multiprogrammed computer systems. In such kinds of applications, the objective is either to minimize the total amount of waste or to maximize the total profit of the used available material in stock by realizing required quantities of rectangular pieces. The unconstrained two-dimensional cutting problem then appears, either directly, when simple heuristic methods are developed, or as an auxiliary problem for generating columns when generalized linear programming is used. In the latter case, an instance of the problem is described by a set of weights (shadow prices) associated to each rectangular piece and the objective is to find a feasible cutting pattern which maximizes the total weight. The same weighted version of the problem appears also when clever heuristics are used for real applications. For example, in a production line, the weights could define priorities for some types of pieces or even to impose that some pieces, already present in the current cutting pattern, should appear also in the next cutting pattern. In other cases without weights on the pieces, the optimizing criterion is simply minimizing waste. But, it can also be seen as a weighted case by considering a weight equal to the surface of each rectangular piece, and the objec-

* Corresponding author.

SSDI 0377-2217(95)00343-2

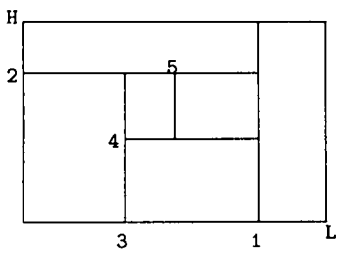

Fig. 1. A guillotine cutting pattern.

tive becomes equivalent to maximizing the total occupied area in the initial rectangle (unweighted version). Frequently, in order to reduce the number of possible cutting patterns we consider two restrictions, i.e. the cutting is guillotine type (horizontal and vertical cuts producing two sub-rectangles, see Fig. 1) and the pieces are of fixed orientation (a piece of length l and height h is different from a piece of length h and height l, $l \neq h$). These restrictions, however, do not considerably limit the scope of the applications since a large number of real-word problems naturally appear with such restrictions.

In this paper, we discuss the better known exact algorithms for the weighted and unweighted versions of the problem. We present a recursive exact algorithm which can be seen as a generalization of Herz's algorithm [8] and we develop efficient lower and upper bounds which, with some established optimality criteria, considerably limit branching in the developed tree. Next, we present an extensive experimental study with a large number of randomly generated instances and the computational power of the algorithm is compared to Gilmore and Gomory's algorithm (GG) [7], as well as to a modified version of this algorithm proposed by Beasley (MGG) [1].

2. Exact algorithms

An instance of TDC is described by a triplet $(\mathcal{R}, \mathcal{S}, c)$. $\mathcal{R}$ is a stock rectangle or initial rectangle with dimensions L (length) and H (height). $\mathcal{S}$ is a finite set of rectangular pieces with smaller dimensions (l_i, h_i) and $c = (c_i)$, $i = 1, \ldots, n$ a weight vector associated to each piece to cut. The parameters L, H, l_i, h_i and c_i are integer values.

A feasible cutting pattern is a vector $a = (a_1, a_2, \ldots, a_n)$ where a_i is the number of pieces of type i cut in $\mathcal{R}$ with respect to the piece's orientation and guillotine cuts.

A solution of the problem instance is a pair (a, F) with a a feasible cutting pattern and $F = a^t c$ is the value of the cutting pattern.

An optimal solution of the TDC problem is a pair (a^*, F^*) such that the cutting pattern a^* gives the maximum value F^*, over all possible feasible cutting patterns.

In [8], a powerful recursive tree search algorithm was given for solving unweighted unconstrained TDC problems, i.e. without weights associated to each piece and without the assumption that the number of appearances of a type of piece in a cutting pattern is bounded by a number strictly inferior to the obvious bound $\lfloor L/l_i \rfloor \lfloor H/h_i \rfloor$. The algorithm takes advantages of some easily calculated lower and upper bounds related to the area occupied by the pieces and the area of the initial rectangle or sub-rectangle. A significant branching cut in the developed tree is obtained by these bounds which make the algorithm particularly efficient for solving small and medium size problem instances. Unfortunately, the bounds are not valid anymore when weights others than the piece's area are associated to the pieces. This inconvenience reduces dramatically the power of the algorithm and limits considerably its scope of applications. For example, in generalized linear programming approaches of the general cutting stock problem or in scheduling procedures, the weights do not have anything in common with the piece's area. Morabito et al. [10] have developed a heuristic based upon this search procedure including a depth bound and a hill climbing strategy.

For the weighted unconstrained TDC problem, the Gilmore and Gomory [7] algorithm based on dynamic programming is one of the most powerful algorithms. Obviously, the algorithm is applicable also to unweighted unconstrained TDC problems by assigning weights $c_i = l_i h_i$ to each piece, but in this case the algorithm of Herz is more efficient.

Gilmore and Gomory's algorithm is based on the following recursive functional equation:

$$F(x,y) = \max_{\substack{x \geqslant x_1 + x_2,\ 0 < x_1 \leqslant x_2 \\ y \geqslant y_1 + y_2,\ 0 < y_1 \leqslant y_2}} \begin{cases} \max\{\{0, c_j\} : l_j \leqslant x, \\ \quad h_j \leqslant y, \forall (l_j, h_j) \in \mathcal{S}\} \\ F(x_1, y) + F(x_2, y) \\ F(x, y_1) + F(x, y_2) \end{cases}$$

Box 1. Gilmore and Gomory's algorithm:

Input: an instance of unconstrained two-dimensional cutting problem.
Output: the optimal solution denoted F^* and its structure L^s and H^s.

Initialization
set $F^*(x,y) = F_0(x,y)$;
$L^s(x,y) = x$; $H^s(x,y) = y$; for x and y such that $0 \leqslant x \leqslant L$ and $0 \leqslant y \leqslant H$;
set $x_2 = 1$ and $y_2 = 1$;

Step 1
set $x_1 = 1$;
repeat
$V = F^*(x_1, y_2) + F^*(x_2, y_2)$;
if $V > F^*(x_1 + x_2, y_2)$ *then*
set $F^*(x_1 + x_2, y_2) = V$; $L^s(x_1 + x_2, y_2) = x_1$; $H^s(x_1 + x_2, y_2) = y_2$;
else
if $V = F^*(x_1 + x_2, y_2)$ *then* $L^s(x_1 + x_2, y_2) = x_1$;
$x_1 = x_1 + 1$;
until $(x_1 > x_2)$ *or* $(x_1 + x_2 > L)$;

Step 2
set $y_1 = 1$;
repeat
$V = F^*(x_2, y_1) + F^*(x_2, y_2)$;
if $V > F^*(x_2, y_1 + y_2)$ *then*
set $F^*(x_2, y_1 + y_2) = V$; $L^s(x_2, y_1 + y_2) = x_2$; $H^s(x_2, y_1 + y_2) = y_1$;
else
if $V = F^*(x_2, y_1 + y_2)$ *then* $L^s(x_2, y_1 + y_2) = y_1$;
$y_1 = y_1 + 1$;
until $(y_1 > y_2)$ *or* $(y_1 + y_2 > H)$;

Step 3
if $x_2 < L$ *then* $x_2 = x_2 + 1$ and go to step 1;
if $y_2 < H$ *then* $y_2 = y_2 + 1$, set $x_2 = 1$ and go to step 1;

Step 4
Exit with the optimal solution $F^*(L, H)$ and its structure.

where x, x_1, x_2, and y, y_1, y_2 have all discrete values.

The basic idea of the method is to consider the set of sub-rectangles $(1,y),(2,y),\ldots,(x_2,y)$, $(x_2 + 1, y)\ldots,(L,y)$ for $y = 1,\ldots,H$, and then, for each such sub-rectangle to find the best solution by using the already known best solution of its sub-rectangles. The main steps of the algorithm are given in Box 1. Beasley [1] has shown how discretization procedure described by Herz [8] and Christofides and Whitlock [3] can be used to improve the performance of the GG algorithm and use it to provide a heuristic for large problems.

In [5], an approximation algorithm was presented for solving unweighted unconstrained TDC problems. The algorithm is based on the one-dimensional knapsack problem and it turns out to be very effi-

cient for large size instances. When dynamic programming methods are used for solving the involved one-dimensional knapsack problems within pseudo-polynomial time complexity, the algorithm remains quite efficient and provides near-optimal cutting patterns within short computational times. In [9], we have used this algorithm to obtain efficient lower and upper bounds in the recursive tree search procedure of Herz for solving exactly unweighted unconstrained TDC problems. It has been shown that the resulting algorithm improved considerably the efficiency of Herz's algorithm. An extensive experimental study on large size instances has shown that the improved algorithm provided an important average gain (25%) in the required computational time.

In this paper, we use the main features of the algorithm presented in [5] which permit to develop an exact recursive algorithm for the weighted unconstrained TDC problem. The algorithm constitutes a generalization of the algorithm described in [9] dealing with both the weighted and unweighted cases. For the unweighted case, it behaves as the improved algorithm of Herz [8,9], whereas for the weighted case, where Herz's algorithm breaks down, it behaves better than the GG algorithm and MGG, the modified version of this algorithm introduced by Beasley.

3. The algorithm

The algorithm is based on the recursive property characterizing a guillotine unconstrained TDC problem. That is, the optimal solution, for a given rectangle $\mathcal{R}$ and a set of pieces $\mathcal{S}$, is deduced as the sum of the optimal solutions of the two sub-rectangles produced by a guillotine cut. This observation permits us to develop a recursive procedure as follows: consider all possible vertical and horizontal dissections of the initial rectangle. For each dissection, consider the two sub-rectangles produced. For each sub-rectangle, consider also all possible dissections and so on. A sub-rectangle is not considered for further dissections if it produces only one piece belonging in $\mathcal{S}$. In this way, the optimal solution for each sub-rectangle is found and thus the optimal one for the initial rectangle.

Obviously, in order to deal with a finite number of dissections we discretize the length and the height of each rectangle or sub-rectangle, without however losing the optimal solution. Such a discretization was used in Refs. [3,8]. The adequate points for making vertical dissections are the points which are linear combinations of lengths of the pieces entering a given (sub-)rectangle, whereas for horizontal dissections the adequate points are linear combinations of heights of the same pieces. However, the efficiency of this recursive procedure remains considerably limited. In order to speed up the process, we take into account the implied symmetry and thus, the discretization points are limited to half of the length or the height of the (sub-)rectangle that we cut at each node of the developed tree.

Next, we develop some optimality criteria, lower and upper bounds, and branching strategies which increase drastically the computational power of this procedure.

3.1. Lower bounds

A lower bound for the initial rectangle can be obtained by applying the 0-cut phase of the "Best Strips Cutting Algorithm" proposed in [5]. This algorithm uses the one-dimensional knapsack for creating a set of horizontal and vertical strips and then combines them for constructing a feasible cutting pattern. By using dynamic programming procedures for solving the one-dimensional knapsack problem, it was shown that all horizontal and vertical strips and even more, the optimal ones with respect to a given height or length respectively, can be created by solving only two one-dimensional knapsack problems. A feasible cutting pattern, then, is realized by solving two other one-dimensional knapsack problems.

Consequently, an initial solution for the unconstrained weighted or unweighted TDC problem is obtained by solving four one-dimensional knapsack problems. Notice that the one-dimensional knapsack problems which are involved are of small sizes even for large size instances of the TDC problem. Therefore, dynamic programming methods for solving these one-dimensional knapsacks turn out to be quite efficient. Moreover, as will be discussed later, they supply optimal solutions for some sub-rectangles that appear in some inner nodes of the developed tree. Next, in order to make the paper self-contained and for increasing clarity of the proposed algorithm, we

recall briefly the procedure which creates horizontal strips.

We reorder the elements of the set S in increasing order such that $h_1 \leqslant h_2 \leqslant \ldots \leqslant h_n$. Let r denote the number of different heights of the set S and (α, β) denote a (sub-)rectangle. We define the one-dimensional knapsack problem $(K^i_{\alpha\beta})$, for $i = 1, \ldots, r$, as follows:

$$(K^i_{\alpha\beta}) \begin{cases} f_i(\alpha) = \max \sum_{j \in R_{\alpha\beta}} c_j x_j \\ \text{such that } \sum_{j \in R_{\alpha\beta}} l_j x_j \leqslant \alpha, \ x_j \in \mathbb{N}, \ j \in R_{\alpha\beta} \end{cases}$$

where $R_{\alpha\beta}$ is the set of rectangular pieces entering in the (sub-)rectangle (α, β), x_j denotes the number of times the piece j appears in the ith strip, c_j the weight associated to piece j of the set $R_{\alpha\beta}$, and $f_i(\alpha)$ the solution value for ith strip.

These r one-dimensional knapsacks generate r optimal strips with respect to the length α and the height h_i if they are solved exactly. Each strip $i = 1, \ldots, r$ is characterized by its solution value $f_i(\alpha)$ and its height β_i equal to some h_j, $j = 1, \ldots, n$. By selecting the best of these strips we construct a feasible cutting pattern. This can be realized, by solving the one-dimensional knapsack problem $(K_{\alpha\beta})$ defined as follows:

$$(K_{\alpha\beta}) \begin{cases} \mathcal{L}(\alpha, \beta) = \max \sum_{i=1}^{r} f_i(\alpha) y_i \\ \text{such that } \sum_{i=1}^{r} h_i y_i \leqslant \beta, \ y_i \in \mathbb{N} \end{cases}$$

where β is the height of (sub-)rectangle to cut and y_i, $i = 1, \ldots, r$ is the number of occurrences of ith strip. $\mathcal{L}(\alpha, \beta)$ is a lower bound for (sub-)rectangle (α, β). This procedure, when (α, β) is the initial rectangle (L, H), is basically the two-phase procedure of Gilmore and Gomory [6] for the unconstrained two-dimensional guillotine cutting problem. Here, we extend it for vertical strips and we exploit intermediate optimal solutions for easily obtaining lower bounds for smaller rectangles (α, β), $\alpha \leqslant L$, $\beta \leqslant H$. The vertical strips are created by replacing in the previous procedure: h_i by l_i, l_i by h_i, α by β and β by α. In this way a second feasible cutting pattern (see also [5]) is obtained.

The best of these two cutting patterns is retained as a lower bound for the initial rectangle. For the subsequent sub-rectangles, in order to obtain quickly lower bounds, we limit ourselves to lower bounds corresponding to horizontal strips. We have already mentioned that four one-dimensional knapsacks provide a lower bound for the initial rectangle. In fact, if we solve the problem $(K^i_{\alpha\beta})$ for $\alpha = L$, $\beta = H$ and $i = r$, by using dynamic programming techniques, the optimal solutions of the $r - 1$ one-dimensional knapsacks are known and the $r - 1$ optimal strips are already created. Consequently, the parameters used in the $(K_{\alpha\beta})$ problem are perfectly known. Thus, the first lower bound $\mathcal{L}(\alpha, \beta)$ produced by horizontal strips at the zero level is obtained by two one-dimensional knapsacks. The same observations are made for the vertical strips which provide the second lower bound at the zero level, also by two knapsacks. As concerns the inner nodes of the tree where we deal with a sub-rectangle (α, β), the lower bounds should be obtained by solving the r' problems, $(K^i_{\alpha\beta})$, $i = 1, \ldots, r'$, $\alpha < L$, $\beta < H$, $r' < r$ (since some pieces are not entering in (α, β)). Fortunately, the optimal values of these problems are also available by the solution of the (K^r_{LH}) problem at the beginning of the process. Therefore, each lower bound for each sub-rectangle (α, β) in the inner nodes is obtained by solving only the $(K_{\alpha\beta})$ problem.

3.2. Upper bounds

An optimal cutting pattern for a (sub-)rectangle (α, β), since it should not violate the area in which it is performed, is obviously a feasible solution of the following one-dimensional knapsack, with integers bounded variables x_j:

$$(K_u) \begin{cases} \mathcal{U}(\alpha, \beta) = \max \sum_{j \in R_{\alpha\beta}} c_j x_j \\ \text{such that } \sum_{j \in R_{\alpha\beta}} (l_j h_j) x_j \leqslant (\alpha\beta) \\ x_j \leqslant \lfloor \frac{\alpha}{l_j} \rfloor \times \lfloor \frac{\beta}{h_j} \rfloor \\ x_j \geqslant 0, \ j \in R_{\alpha\beta} \end{cases}$$

where $R_{\alpha\beta}$ is the set of rectangular pieces entering a (sub-)rectangle (α, β), x_j is the number of appearances of jth piece into (sub-)rectangle (α, β). However, in order to avoid the long computational time required by a such large-dimension knapsacks, we re-

lax the integer requirement on the variables and we quickly obtain a lower quality upper bound equal to $\lfloor \mathcal{U}(\alpha,\beta) \rfloor$ by solving the (K_u) problem. This upper bound has already been used in similar tree-search procedures [14,2,10].

3.3. Effects of the bounds

Let now see how we can use the previously described lower and upper bounds in a tree-search procedure. We consider a sub-rectangle (α,β) and a vertical cut at point x. Let (x,β) and $(\alpha-x,\beta)$ denote the produced two sub-rectangles. The horizontal cuts are treated similarly. We denote V the best current value.

(i) Consider an upper bound $\mathcal{U}(x,y)$ and the optimal value $Opt(x,y)$ for a sub-rectangle (x,y). Set $Opt(x,y)=\infty$, if the optimal value is not yet known. Set

$$v_0 = V - \min\{\mathcal{U}(x,\beta), Opt(x,\beta)\}.$$

Clearly, if $v_0 \geqslant \mathcal{U}(\alpha-x,\beta)$ then it is not necessary to investigate the dissection for the sub-rectangle $(\alpha-x,\beta)$. The value v_0 can not be attained by cutting $(\alpha-x,\beta)$. Similarly, if the sub-rectangle $(\alpha-x,\beta)$ is chosen and $v_0 \geqslant \mathcal{U}(x,\beta)$, then further dissection of (x,β) is avoided.

(ii) If $\min\{\mathcal{U}(x,\beta), Opt(x,\beta)\} + \min\{\mathcal{U}(\alpha-x,\beta), Opt(\alpha-x,\beta)\} \leqslant \mathcal{L}(\alpha,\beta)$, then a cut at x can be skipped without loss of optimality, since the best patterns for the sub-rectangles (x,β) and $(\alpha-x,\beta)$ cannot improve the already known pattern $\mathcal{L}(\alpha,\beta)$ for the (sub)rectangle (α,β).

(iii) A lower bound $\mathcal{L}(\alpha,\beta)$ for a (sub-)rectangle (α,β) is always updated when the next produced two sub-rectangles, provide a better lower bound. That is, if $\mathcal{L}(x,\beta)+\mathcal{L}(\alpha-x,\beta) > \mathcal{L}(\alpha,\beta)$, then $\mathcal{L}(\alpha,\beta)$ is replaced by $\mathcal{L}(x,\beta)+\mathcal{L}(\alpha-x,\beta)$.

Consequently, if both optimal values for the next two sub-rectangles are already found, then we stop the branching and the current lower bound $\mathcal{L}(\alpha,\beta)$ is updated by the new value $\max\{\mathcal{L}(\alpha,\beta), Opt(x,\beta) + Opt(\alpha-x,\beta)\}$.

(iv) If the sum of the two lower bounds for the two sub-rectangles produced by a cutting at a point x exceeds the known upper bound $\mathcal{U}(\alpha,\beta)$ for the current (sub-)rectangle (α,β) i.e. $\mathcal{L}(x,\beta)+\mathcal{L}(\alpha-x,\beta) \geqslant \mathcal{U}(\alpha,\beta)$ then the branching on the resulting sub-rectangles is stopped.

3.4. Branching strategy

The procedure which creates lower bounds, as was confirmed in [5] for the unweighted TDC problem and by the experimental justification reported here for the weighted TDC problem, is particularly efficient for large size instances. Therefore, the cuts made on a (sub-)rectangle (α,β) are examined from the middle to the left of the length for the vertical cuts and from the middle to the bottom for the horizontal ones. This branching strategy in the tree-search procedure permits us to deal firstly with large sub-rectangles, in order to take profit of the high quality lower bounds. For the same reasons, after a cut has been made, the largest produced sub-rectangle is systematically chosen in the algorithm for further dissections.

As was already mentioned, in order to deal with a finite number of dissections without loss of optimality, the cuts are made at points in the horizontal axis which are linear combinations of lengths of pieces entering a sub-rectangle to cut. Similarly, the cuts on the vertical axis are linear combinations of the heights. In this way, the patterns that are produced have a structure with pieces always at the left and the bottom of the rectangle. These patterns are the so-called normalized cutting patterns used also by Herz [8] and Christofides and Whitlock [3]. A function which generates these points can be found in [3]. In the sequel, the set of points on the horizontal axis, for a (sub-)rectangle (α,β) is denoted by

$$P_{\alpha\beta} = \{x \mid x = \sum_{i=1}^{n} l_i z_i \leqslant \alpha,\ z_i \in \mathbb{N}, h_i \leqslant \beta\},$$

and the set of points on the vertical axis is denoted by

$$Q_{\alpha\beta} = \{y \mid y = \sum_{i=1}^{n} h_i z_i \leqslant \beta,\ z_i \in \mathbb{N}, l_i \leqslant \alpha\}.$$

3.5. Optimality criteria

During the search, some branches are discarded by virtue of optimality for some sub-rectangles. In fact, if we consider the sets $P_{\alpha\beta} = \{x_1, x_2, \ldots, x_k\}$ and $Q_{\alpha\beta} = \{y_1, y_2, \ldots, y_{k'}\}$, where k and k' are the cardinalities of the sets $P_{\alpha\beta}$ and $Q_{\alpha\beta}$ respectively, then the optimal solutions are already known or are obtained when we

Box 2. Recursive algorithm

Input: an instance of unconstrained two-dimensional cutting.
Output: The optimal solution value denoted by Opt and its structure
1. Construct the sets P_{LH} and Q_{LH}
2. Compute lower bound $\mathcal{L}(L,H)$ at zero level; set $v_0 = \mathcal{L}(L,H)$
3. Opt=max$\{v_0, \mathcal{F}(L,H,v_0)\}$
4. Function $\mathcal{F}(\alpha,\beta,v_0)$: *integer*;
   Compute upper bound $f = \mathcal{U}(\alpha,\beta)$
   if $v_0 \geqslant f$ then exit with $\mathcal{F} = 0$ {effect 1}
   else let $\alpha_0 = \sup\{x \mid x \leqslant \alpha,\ x \in P_{\alpha\beta}\}$ and $\beta_0 = \sup\{y \mid y \leqslant \beta,\ y \in Q_{\alpha\beta}\}$
     if the optimal solution for (α_0,β_0) is already known then exit with this value {OPT 3}
     else
       if $\min_{h_j/l_j \leqslant \alpha_0} \{h_j\} > \beta_0/2$ then exit with *the value of the optimal strip* {OPT 1}
       else find a lower bound $\mathcal{L}(\alpha_0,\beta_0)$ {if it is not the zero level}
         if $\mathcal{L}(\alpha_0,\beta_0) \geqslant \mathcal{U}(\alpha_0,\beta_0)$ then exit with $\mathcal{F} = \mathcal{L}(\alpha_0,\beta_0)$
         else
           if $\min_{l_j/h_j \leqslant \beta_0} \{l_j\} > \alpha_0/2$ then exit with *the optimal solution* {OPT 2}
           else $V = \mathcal{L}(\alpha_0,\beta_0)$
             for all $x \in P_{\alpha,\frac{\beta}{2}}$
               compute $Opt(x,\beta) = \mathcal{F}(x,\beta,\max(V,v_0) - \mathcal{U}(\alpha - x,\beta))$
               and record solution's structure
               if $\min\{\mathcal{U}(x,\beta), Opt(x,\beta)\} + \min\{\mathcal{U}(\alpha - x,\beta), Opt(\alpha - x,\beta)\} \leqslant \mathcal{L}(\alpha,\beta)$ {effect 2}
                 then exit with best known value V
               else
               set $V_1 = Opt(x,\beta) + \mathcal{F}(\alpha - x,\beta,\max(V,v_0) - Opt(x,\beta))$
               and record solution's structure
             if $V_1 \geqslant \mathcal{U}(\alpha,\beta)$ then exit with $\mathcal{F} = V_1$ {effect 4}
             else $V_1 = \max(V,V_1)$ and record solution's structure {effect 3}
             repeat for all $y \in Q_{\alpha\frac{\beta}{2}}$
   exit with *the best solution.*

solve the $K_{\alpha\beta}$ problem for many sub-rectangles. For example, the following sub-rectangles:

$$\{(L,y_1),(x_k,y_1),(x_{k-1},y_1),\ldots,(x_1,y_1)\},$$
$$\{(L,y_2),(x_k,y_2),(x_{k-1},y_2),\ldots,(x_1,y_2)\},$$
$$\vdots$$
$$\{(L,y_\ell),(x_k,y_\ell),(x_{k-1},y_\ell),\ldots,(x_1,y_\ell)\}$$

with $y_i < 2h_s$, $i = 1,\ldots,\ell$ and h_s, $s \leqslant n$, the *minimum* height of the pieces entering each of these sub-rectangles are known to be solved to the optimum at the beginning of the algorithm, after we have solved the (K'_{LH}) problem. This is because the optimal solution of these sub-rectangles are composed by only one strip and the strips considered here are optimal (OPT 1).

Also, the following sub-rectangles are solved to the optimum after we have applied the $K_{\alpha\beta}$ problem:

$$\{(x_1,H),(x_1,y_{k'}),(x_1,y_{k'-1}),\ldots,(x_1,y_1)\},$$
$$\{(x_2,H),(x_2,y_{k'}),(x_2,y_{k'-1}),\ldots,(x_2,y_1)\},$$
$$\vdots$$
$$\{(x_{\ell'},H),(x_{\ell'},y_{k'}),(x_{\ell'},y_{k'-1}),\ldots,(x_{\ell'},y_1)\}$$

with $x_i < 2l_s$, $i = 1,\ldots,\ell'$ and l_s the *minimum* length of the pieces entering each of these sub-rectangles.

This is, because their optimal solution can not contain more than one pieces horizontally and therefore its structure is a vertical strip (i.e. trivial horizontal strips) obtained by the $(K_{\alpha\beta})$ problem (OPT 2).

When a sub-rectangle of dimensions α and β has been solved to the optimum, we record its optimal value and its dimensions in order to avoid to compute it again at other nodes. Remark, however, that each sub-rectangle (α, β) has the same optimal solution as the sub-rectangle (α_0, β_0) where $\alpha_0 = \sup\{x/x \leqslant \alpha,\ x \in P_{\alpha\beta}\}$ and $\beta_0 = \sup\{y/y \leqslant \alpha, y \in Q_{\alpha\beta}\}$. Therefore, in order to better exploit the optimality obtained for the sub-rectangle (α, β) we record the dimensions α_0 and β_0 instead of α and β. Hence, sub-rectangles larger or smaller than (α, β) but with the same α_0 and β_0 are not treated (OPT 3).

We remark that the effectiveness of optimality criteria will be significant at lower levels of the developed tree. Moreover, the effectiveness of lower and upper bounds will be more significant at the upper levels.

In summary, the algorithm we propose is a recursive tree-search procedure, similar to the procedure of Herz. Its particularities are that at the beginning it applies a powerful heuristic to obtain an initial solution, by solving four one-dimensional knapsack problems. For each inner node it solves a one-dimensional knapsack for obtaining a lower bound and also a relaxed bounded one-dimensional knapsack for obtaining an upper bound. Box 2 gives a detailed description of the algorithm. $P_{\frac{\alpha}{2}\beta}$ and $Q_{\alpha\frac{\beta}{2}}$ in the algorithm denote subsets of $P_{\alpha\beta}$ and $Q_{\alpha\beta}$ respectively, including points less or equal to half of the length or the height of the sub-rectangle (α, β), in order to avoid effects of symmetry.

4. Computational results

This section presents empirical evidence for the performance of the new algorithm by comparing it to GG and MGG algorithms. As is already mentioned, the power of the algorithm is arising from the high quality lower bounds and the way they are calculated (solving only one one-dimensional knapsack at each node of the tree). Before studying the performance of the algorithm let us see the effectiveness of these lower bounds.

Table 1
Lower bounds quality

Instances size	Group 1	Group 2	Total
Av. time (s) for GG	4.03	12.07	8.05
Av. time (s) for MGG	3.69	10.71	7.20
Av. time (s) for lower bounds	0.43	0.79	0.61
Percentage time	11.65	7.38	8.47
Av. approximation ratio	0.993	0.997	0.995
(%) Lower bound = Optimal solution	76	72	74

We consider two groups of randomly generated instances. The first group includes 150 instances, with sizes L and H taken in the interval $[30, 80]$ and the number of pieces to cut are taken in the interval $[10, 40]$. The second group includes also 150 instances. The parameters L and H range in the interval $[80, 150]$, whereas the number of pieces to cut ranges in the interval $[20, 100]$. The first group includes small or medium instances, whereas the second group includes rather large instances for exact algorithms. The dimensions L and H of the initial rectangle and the number of pieces to cut are taken uniformly on the fixed interval, while the dimensions of the pieces to cut are picked up uniformly in the interval $]0, L]$ and $]0, H]$, respectively. The weight associated to a piece i is computed by $c_i = \lceil \gamma \pi_i \rceil$, where π_i, $i = 1, \ldots, n$, is the piece's area and γ is a number uniformly distributed in the real interval $[1, 5]$.

In Table 1 we observe that MGG performs better than the standard version GG. Also, lower quality bounds at the zero level as well as the required computational times are given. For all treated instances, excellent quality lower bounds are obtained within average time representing 8.47% of the average time required by MGG algorithm. Moreover, 74% of the lower bounds are equal to the optimal values.

In particular, for group 1, the lower bounds represent feasible solutions very close to the optimal ones. The average approximation ratio (*lower bound value/optimal solution value*) is 0.993. The required average computational time represents 11.65% of the time required by MGG algorithm. The percentage of the bounds equal to the optimal values is 76%. In the case of group 2 the efficiency of the lower bounds is still increasing: the average approximation ratio is equal to 0.997, whereas the average required

Table 2
Computational time performance of the new algorithm compared to GG and MGG algorithms on some particular instances. The number n of pieces to cut is 5, 7, 10, 20, 10 and 10, respectively.

Instances size	(127,98)	(15,10)	(40,70)	(40,70)	(70,40)	(70,40)
Value of optimal solution	12348	249	3076	2240	2758	2776
GG (Av. time (s))	3.19	0.051	0.44	1.17	0.51	0.48
MGG (Av. time (s))	2.70	0.053	0.32	0.98	0.53	0.47
New algorithm (Av. time (s))	1.42	0.047	0.21	0.69	0.43	0.42
(%) time gain versus MGG	47.41	11.32	34.37	29.59	18.87	10.64

computational time represents 7.38% of the MGG algorithm. In the last case, the percentage of bounds which are optimal solutions is 72%. 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 becomes more important.

So, by including these good quality bounds in a tree-search procedure, one expects also a good behavior of the resulting algorithm. Notice also that the lower bounds in the inner nodes of the tree require less computational time, since only one one-dimensional knapsack of small size is solved at each node. We return now to examine the performance of the algorithm.

We consider (see Table 2) some instances taken in the literature and we compare computational time required by MGG (which in general behaves better than GG) algorithm to the time of new algorithm. The first instance is taken from [8]. The weight associated to each piece is exactly the area of the piece, i.e. $c_i = l_i h_i$ for $i = 1, \ldots, n$. The three following instances are taken from [3], where we have relaxed the upper bounds on the number of repetitions of each type of piece. The weights associated to each piece are assigned independently of the piece's area. The two other instances are taken from [11]. For all these instances the new algorithm is faster than GG and MGG algorithms. The computational time gain is at least 11.32% (second instance), and for some instances it is very important, i.e. 47.41% (first instance) and 34.37% (third instance).

In Table 3, we can see the computational performance of the new algorithm on the randomly generated instances described above. The average gain on the total number of treated instances is 20.07%.

An important point is that the average time gain is increasing with instances size. This can be explained by the efficiency of lower bounds especially on the large size instances (Table 1), and also by the fact that the GG algorithm performs poorly for large instances. However, for instances of group 1, the GG algorithm is sufficiently efficient. For this reason, we have decided to exploit the efficiency of the GG algorithm on small instances in order to increase even more the efficiency of the new algorithm on large instances. This has been done by developing a hybrid algorithm which still uses the tree-search GG algorithm, but only on a quarter of the initial rectangle. So, at the beginning of the new algorithm, we apply the procedure of GG on the sub-rectangle $(\lfloor L/2 \rfloor, \lfloor H/2 \rfloor)$. Next, at each node of the tree when we deal with a sub-rectangle smaller than a quarter of the initial rectangle we know its optimal solution by virtue of the dynamic programming basis of the GG algorithm.

Table 3
Computational time performance of the new algorithm compared to the MGG algorithm on randomly generated instances

Instances size	Group 1	Group 2	Total
Av. time (s) for MGG	3.69	10.71	7.20
Av. time (s) for new algorithm	3.13	8.38	5.755
Percentage time gain	15.17	21.75	20.07

Table 4
Computational time performance of the new hybrid algorithm versus MGG algorithm on randomly generated instances

Instances size	Group 1	Group 2	Total
Av. time (s) for MGG	3.69	10.71	7.20
Av. time (s) for the hybrid algorithm	2.71	7.32	5.015
Percentage time gain	26.56	31.65	30.35

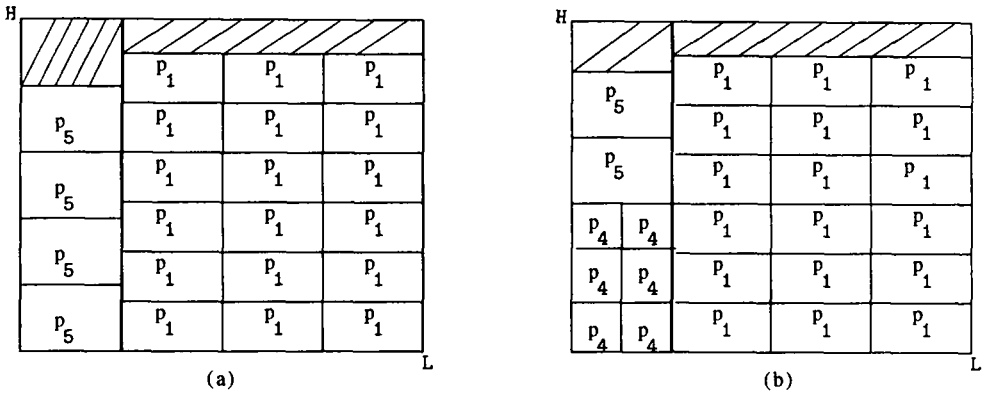

Fig. 2. An instance of the weighted TDC problem with $(L, H) = (99, 80)$ and 5 pieces to cut. (a) the solution structure corresponding to the lower bound at zero level. (b) the structure of the optimal solution provided by the exact algorithms.

Table 5
Lower bounds quality and performance of the new algorithm on 5 instances taken in [10]. "*" denotes the optimal value

Instance	(100,156)	(253,294)	(318,473)	(501,556)	(750,806)
Optimal value	15024	73176	142817	265768	577882
Time (s) Herz's algorithm	3.91	9.92	7.49	6.53	7.47
Time(s) for new algorithm	3.07	6.83	5.81	4.21	3.75
Time for lower bound	0.16	0.43	0.60	0.71	1.22
Value of lower bound	*	72172	141810	*	*

Consequently, we have three alternatives for pruning branches in the hybrid algorithm: the lower bounds, the upper bounds and the optimality criteria provided either by the one-dimensional knapsack or the GG algorithm. We recall here that on high levels of the developed tree one-dimensional knapsacks are more important, since we deal with large subrectangles, whereas on lower levels optimality criteria play a particular role.

Table 4 shows the computational power of the hybrid algorithm. The average computational time gain over the MGG algorithm, for all treated instances, is considerably increased, being now 30.35%. Also, the average time gain continues to increase with instances size.

In Fig. 2 we give the structure of the solutions corresponding to the lower bound at zero level of the tree (2a) and the optimal solution provided by the exact algorithms (2b), for the following instance. The set of the rectangular pieces is $\mathcal{S} = \{p_1 = (21, 13), p_2 = (54, 20), p_3 = (24, 23), p_4 = (18, 35), p_5 = (36, 17)\}$, the corresponding vector of weights is $c = (285, 1083, 556, 273, 729)$ and the dimensions of the initial rectangle L and H are respectively 99 and 80. The lower bound is equal to $\mathcal{L}(L, H) = 7934$ giving an approximation ratio for this instance equal to 0.964. The required calculation time is 28 ms on a PC 386. The value of the optimal solution found by the exact algorithms is 8226. The GG algorithm found it in 445 ms, the MGG algorithm found it in 371 ms. The new algorithm found it in 265 ms, whereas the hybrid algorithm required only 193 ms.

In Table 5 we consider five instances taken in Ref. [10]. They are described by (L, H) equal to $(100, 156), (253, 294), (318, 473), (501, 556)$ and $(750, 806)$ respectively, and 10 pieces to cut with weights equal to the surface. The execution of GG and MGG on these instances was impossible with the resources at our disposal. Thus, we compare the new algorithm with Herz's algorithm. The average percentage time gain on the five instances is 32.21. It is worthwhile to notice the impressive quality of the

lower bounds. For the problems 1, 4 and 5 the lower bound is equal to the optimal solution. The heuristic of Morabito et al. gives the same solutions as the lower bound except for the third instance where it is superior (142817). The required execution time by the Morabito et al. heuristic, on a IBM PC-AT (see [10]) is 3, 3, 5, 4 and 3 seconds for the 5 problems respectively. In [1], a large instance was given described by $(L, H) = (3000, 3000)$ with 32 pieces to cut and weights associated to each piece equal to its surface. The best value found by the heuristic of Morabito and al. is equal to 8944026 (99.378% of the initial rectangle surface) within 36 s. Our initial lower bound is equal to 8997780 (99.975%) and it is obtained in 18.6 s.

In conclusion we can state that the new algorithm performs very well, especially for large instances size for the weighted version of the problem. As concerns the unweighted cases, the algorithm is easily adapted by assigning weights equal to the piece's area and replacing the upper bound by the area of the (sub)rectangle. Then, the algorithm behaves as the improved algorithm presented in [9]. There it was concluded that the gain over Herz's algorithm is 25%. Notice that for the unweighted cases Herz's algorithm is superior to the GG algorithm by 20% (see [8]).

The gain of the new algorithm over the other algorithms will be more important in parallel implementations of these algorithms. For example, consider a straightforward approach as follows: processors simultaneously deal with each dissection at a (sub)rectangle and use the same current solution. If one of the processors finds a better solution, then the current solution is replaced by the newly found solution and this solution is communicated to all processors. In this way, load distribution to all processors can be achieved while the high quality lower and upper bounds preserve from going deep down in the tree. The parallelization however of the algorithms presented here constitutes a field for future work.

Another important point for further investigation is the generalization of the new algorithm for solving bounded two-dimensional cutting problems. The lower and upper bounds described here can be used in Christofides and Whitlock's exact algorithm. An important gain is expected, since the transportation routine which makes the algorithm heavy will be executed less frequently. Also, the heuristic of Morabito et al. [10] can easily be adapted to the algorithm proposed here, with important benefits.

5. Conclusion

A new recursive algorithm is presented for solving weighted two-dimensional cutting problems. The algorithm uses efficient lower and upper bounds and some important optimality criteria which considerably reduce the searching effort. Empirical evidence for the power of the algorithm is given by comparing it to the best known exact algorithm. The key of the algorithm is the exploitation of dynamic programming procedures for solving a series of one-dimensional knapsack problems. When the algorithm is slightly modified by applying Gilmore and Gomory's algorithm on small sub-rectangles we obtain more efficient a hybrid algorithm especially for large size instances. Another important feature of the algorithm is that it can be applied also to solving unweighted two-dimensional cutting problems. In this case the new algorithm is still better than the best actually known algorithm of Herz. Finally, the new algorithm could be generalized for solving bounded two-dimensional cutting problems where the actually known algorithms suffer by heavy computational time. In particular, the lower and upper bounds introduced here should considerably increase the computational power of Christofides and Whitlock's algorithm.

Acknowledgements

Many thanks to an anonymous referee for helpful comments and suggestions.

References

[1] Beasley, J.E., "Algorithms for unconstrained two-dimensional guillotine cutting", *Journal of the Operational Research Society* 36/4 (1985) 297-306.

[2] Beasley, J.E., "An exact two-dimensional non-guillotine cutting tree search procedure", *Operations Research* 33/1 (1985) 49-64.

[3] Christofides, N., and Whitlock, C., "An algorithm for two-dimensional cutting problems", *Operations Research* 25/1 (1977) 30-44.

[4] Dowsland, K., and Dowsland, W., "Packing problems", *European Journal of Operational Research* 56 (19920 2-14.

[5] Fayard, D., and Zissimopoulos, V., "An approximation algorithm for solving unconstrained two-dimensional knapsack problems", *European Journal of Operational Research* 84 (1995) 618–632.

[6] Gilmore, P., and Gomory, R., "Multistage cutting problems of two and more dimensions", *Operations Research* 13 (1965) 94-119.

[7] Gilmore, P., and Gomory, R., "The theory and computational of knapsack functions", *Operations Research* 14 (1966) 1045-1074.

[8] Herz, J.C., "A recursive computing procedure for two-dimensional stock cutting", *IBM Journal of Research and Development* 16 (1972) 462–469.

[9] Hifi, M., and Zissimopoulos, V., "Une amélioration de l'algorithme récursif de Herz pour la résolution du problème de découpe à deux dimensions", to appear in *RAIRO, Recherche Opérationnelle*.

[10] Morabito, R., Arenales, M., and Arcaro, V., "An and–or graph approach for two-dimensional cutting problems", *European Journal of Operational Research* 58/2 (1992) 263–271.

[11] Oliveira, J.F., and Ferreira, J.S., "An improved version of Wang's algorithm for two-dimensional cutting problems", *European Journal of Operational Research* 44 (1990) 256–266.

[12] M. Syslo, N. Deo and J. Kowalik, Discrete optimization algorithms, *Prentice-Hall, New Jersey*, 1983.

[13] Sweeney, P., and Paternoster, E., "Cutting and packing problems: A categorized application-oriented research bibliography", *Journal of the Operational Research Society* 43/7 (1992) 691–706.

[14] Zissimopoulos, V., "Heuristic methods for solving (un)constrained two-dimensional cuting stock problems", *Methods of Operations Research* 49 (1984) 345–357 (S.O.R. Osnabrück, Germany, August, 1984).