Computers & Operations Research 39 (2012) 48–53

Contents lists available at ScienceDirect

Computers & Operations Research

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

Exact algorithms for the two-dimensional guillotine knapsack

Mohammad Dolatabadi [a], Andrea Lodi [b,*], Michele Monaci [c]

[a] Faculty of Mathematics and Statistics, University of Ferdowsi, Mashhad, Iran
[b] DEIS, Università di Bologna, Viale Risorgimento 2, 40136 Bologna, Italy
[c] DEI, Università di Padova, Via Gradenigo 6/A, 35131 Padova, Italy

ARTICLE INFO

Available online 14 January 2011

Keywords:
Two-dimensional knapsack
Guillotine packing
Exact algorithm

ABSTRACT

The two-dimensional knapsack problem requires to pack a maximum profit subset of "small" rectangular items into a unique "large" rectangular sheet. Packing must be *orthogonal without rotation*, i.e., all the rectangle heights must be parallel in the packing, and parallel to the height of the sheet. In addition, we require that each item can be unloaded from the sheet in stages, i.e., by unloading simultaneously all items packed at the same either y or x coordinate. This corresponds to use guillotine cuts in the associated cutting problem.

In this paper we present a recursive exact procedure that, given a set of items and a unique sheet, constructs the set of associated guillotine packings. Such a procedure is then embedded into two exact algorithms for solving the guillotine two-dimensional knapsack problem. The algorithms are computationally evaluated on well-known benchmark instances from the literature.

The C++ source code of the recursive procedure is available upon request from the authors.

1. Introduction

In the *(orthogonal) two-dimensional knapsack problem* (2KP), one is given a rectangular sheet (*knapsack*) of width W and height H, and a set $N=\{1,\ldots,n\}$ of types of rectangles (called *items*). The j-th type of items contains a_j items, each having a *width* w_j, a *height* h_j and a *profit* p_j. The objective is to select and pack a maximum profit subset of items into the knapsack, with the constraint that items do not overlap. In addition, items must be orthogonally packed without rotation, i.e., each item of type j, if selected, must have its edge of height h_j parallel to the edge of the knapsack of height H. This is the most natural two-dimensional generalization of the well-known *one-dimensional knapsack problem* (KP), arising as a special case if $h_j=H$ for all $j \in N$. While KP is solvable in pseudopolynomial time by dynamic programming, it is known that 2KP is strongly NP-hard, since it generalizes the One-Dimensional Bin Packing (1BP) problem as well.

Real-world applications of 2KP arise in loading, transportation, resource allocation, just to mention a few. In addition, 2KP arises as subproblem in many complex problems; for instance, it turns out to be the *pricing* problem when column generation techniques are used to solve the Two-Dimensional Bin Packing problem (2BP). Since the seminal work of Gilmore and Gomory [17], a large amount of literature has been proposed for 2KP, both from practical and theoretical viewpoints. Boschetti et al. [5] presented Integer Linear Programming (ILP) formulations for 2KP, used to derive upper bounds through Lagrangian or surrogate relaxations and embedded into a branch-and-bound algorithm. Fekete et al. [15] proposed bounding procedures based on *dual feasible functions* and exploited these bounds within an exact algorithm for 2KP. Caprara and Monaci [6] considered the relaxation given by the KP instance with item weights coincident with the rectangle areas, and proved that this bound has an absolute wort-case performance ratio equal to 3 (the performance being the same even if rotation of items is allowed). In addition, different branch-and-bound algorithms based on the KP relaxation were proposed. From a computational viewpoint, the exact approaches in [6] and [15] have similar performance. On the approximation side, Jansen and Zhang [22] showed that 2KP is strongly NP-hard even for packing squares with identical profits, and presented a $(2+\varepsilon)$-approximation algorithm.

In recent years, most of the literature on two-dimensional packing referred to d-staged packing, i.e., the constrained version of the problem in which one is required to produce so-called guillotine patterns (see, e.g., Fig. 1) where the maximum number of cuts allowed to unload each item is fixed to a given threshold d. This constraint can make the problem easier than pure 2KP from a computational viewpoint, but it increases the amount of wasted space in the solution. However, if automatic machines are used for unloading/cutting items and the sheet is not significantly expensive, it is preferable to look for solutions of this type.

* Corresponding author. Tel.: +39 051 2093029; fax: +39 051 2093073.
E-mail addresses: mo_do43@yahoo.com (M. Dolatabadi), andrea.lodi@unibo.it (A. Lodi), monaci@dei.unipd.it (M. Monaci).

0305-0548/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2010.12.018

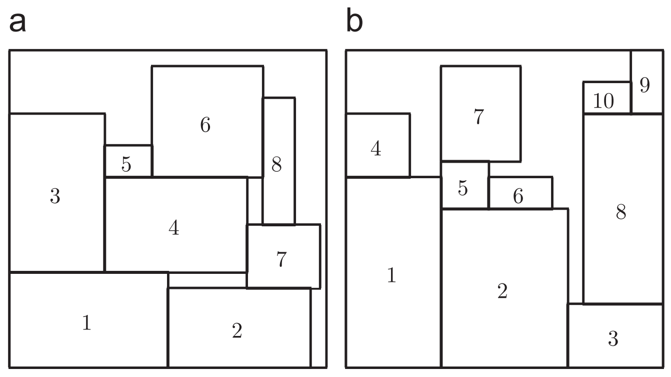

Fig. 1. Examples of non-guillotine cutting (a), and guillotine cutting (b) patterns.

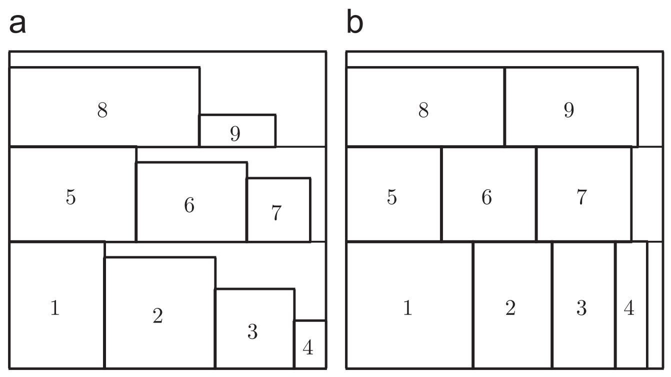

Fig. 2. Examples of 2-staged patterns: non-exact (a) and exact (b) cases.

Many papers in the literature consider the case in which $d=2$, i.e., such that the sheet is cut into levels and each level is further cut to obtain the selected items, one for each of them. Note that, we call *non-exact* 2-staged packing, or packing *with trimming* the case in which a third stage of cutting is allowed *only* to separate an item from a waste area, see Fig. 2(a). Otherwise, we have the *exact* packing, or packing *without trimming* as depicted in Fig. 2(b).

Lodi and Monaci [23] presented two ILP models involving a polynomial number of variables and constraints for the 2-staged 2KP and proposed *ad hoc* techniques aimed at removing symmetries in the models. Recently, arc flow based ILP models for 2-staged 2KP and 2-staged 2BP were proposed by Macedo et al. [24]. A different exact method for 2-staged 2 KP is the branch-and-bound algorithm by Hifi and M'Hallah [21], in which strategies to fathom nodes of the branching tree associated with symmetric solutions are given. Few exact algorithms have been proposed so far for the case in which $d=3$: Puchinger and Raidl [27] extended ILP models in [23] to the 3-staged 2BP. Hifi [19] proposed a depth-first exact algorithm for the unconstrained 3-staged problem, while Vanderbeck [28] solved the constrained version of the problem by using nested decomposition within a classical column generation formulation.

In this paper we go one step further in d-staged packing by considering general guillotine patterns, i.e., patterns where d is not fixed. We denote by G2KP this variant of 2KP. Following the typology by Wäscher et al. [30], the problem could be indicated as guillotine two-dimensional single large object placement problem[1] (2D-SLOPP).

Exact approaches for G2KP have been proposed by Christofides and Whitlock [9], Christofides and Hadjiconstantinou [8], Cung et al. [13], and Pisinger and Sigurd [26].[2] In addition, two upper bounding procedures for G2KP were proposed by Hifi [18]. Among heuristics, the most effective algorithms for G2KP are currently the hybrid approach by Hifi [20], and the recent recursive algorithm by Chen [7]. Moreover, we mention that a similar recursive approach was already used by Cintra and Wakabayashi [10] to design an exact algorithm for G2KP in case no upper bound on the number of items of each type exists (i.e., $a_j=\infty \forall j$).

Recently, the problem of determining if a given set of rectangular items can be inserted into a sheet by means of guillotine cuts has been modeled through oriented graphs by Clautiaux et al. [11]. Finally, Bansal et al. [2] showed that Guillotine 2BP admits an asymptotic polynomial time approximation scheme and showed how general guillotine packings can be approximated by simpler packings.

The paper is organized as follows. In Section 2 we present a recursive exact procedure that, given a set of items and a unique sheet, constructs the set of associated guillotine packings. Such a procedure can be embedded into different types of (exact) algorithms for solving guillotine packing problems. In particular, we present two exact algorithms that make use of this procedure for solving G2KP: the first one, described in Section 3.1, is based on the iterative execution of the recursive procedure with different input parameters, so as to determine the optimal solution value. The second scheme, presented in Section 3.2, is based on an ILP model of the problem; a branch-and-cut algorithm is obtained by relaxing some constraints, and using the recursive procedure as a black box to either prove the optimality of the current solution, or to find violated constraints. Both algorithms are computationally evaluated in Section 4 on some well-known benchmark instances from the literature. Finally, in Section 5 we draw some conclusions.

2. A *Recursive* procedure for guillotine packing

In this section we describe our recursive procedure for enumerating guillotine two-dimensional packings. In the procedure, referred to as *Recursive*, we denote each feasible assignment of a subset of items to the sheet as a *feasible packing*. Each feasible packing can be represented as a non-negative integer vector $f=[f_1, \ldots, f_n]$, where each entry $f_i \le a_i$ $(i=1,\ldots,n)$ represents the number of items of type i in the packing. We denote as $p(f)=\sum_{i=1}^{n} p_i f_i$ the profit of packing f. We say that a feasible packing f is *maximal* if no further items can be packed in the sheet, i.e., if packing $f'=[f_1,\ldots,f_k+1,\ldots,f_n]$ turns out to be infeasible for all type k of items such that $f_k<a_k$. Given two feasible packing $f=[f_1, \ldots, f_n]$ and $\bar{f}=[\bar{f}_1,\ldots,\bar{f}_n]$, we define a new packing $\hat{f}=f+\bar{f}$ as follows: $\hat{f}_i := \min\{f+\bar{f}_i, a_i\}$ $(i=1,\ldots,n)$.

Procedure *Recursive* implicitly enumerates all feasible packings by recursively dividing the sheet into two parts by means of a (either horizontal or vertical) guillotine cut. The procedure receives in input a parameter z_0 which is a lower bound on the profit of any feasible (guillotine) packing to be returned.

As observed by Christofides and Whitlock [9], for any two-dimensional packing problem an optimal solution corresponding to a *normal pattern* exists, i.e., a solution in which any item is packed with its left edge adjacent either to the right edge of another item or to the left of the sheet. This means that we can consider only vertical cuts occurring at x coordinates that can be

[1] We added the term "guillotine" in front because no information about guillotine cuts is present in this classification.

[2] More precisely, Pisinger and Sigurd [26] solve G2KP as pricing problem in a column generation algorithm for Guillotine 2BP. However, for the sake of comparison, in the current paper we will discuss [26] only in terms of their solution approach to G2KP.

obtained as a combination of the widths of the items, i.e., that belong to set

$$\mathcal{W} = \left\{ x : x = \sum_{i=1}^{n} w_i \alpha_i,\ 1 \le x \le W,\ 0 \le \alpha_i \le a_i, i = 1, \dots, n \right\}.$$

In a similar way, we consider only horizontal cuts that occur at y coordinates belonging to the following set

$$\mathcal{H} = \left\{ y : y = \sum_{i=1}^{n} h_i \alpha_i,\ 1 \le y \le H,\ 0 \le \alpha_i \le a_i, i = 1, \dots, n \right\}.$$

We assume that the elements of both $\mathcal{W}$ and $\mathcal{H}$ are sorted according to increasing values, and let $t = |\mathcal{W}|$ and $s = |\mathcal{H}|$.

Given $x \in \mathcal{W}$ and $y \in \mathcal{H}$, and a threshold solution value z_0, let $F(x, y, z_0)$ be the set of all feasible packing of the given items into a sheet of size $x \times y$ that can produce (together with the residual items and sheet) a profit larger than or equal to z_0. Given two sets of feasible packing $F^1 = \{f^1, \dots, f^k\}$ and $F^2 = \{\overline{f^1}, \dots, \overline{f^m}\}$, we denote by $F^1 \oplus F^2$ the pairwise sum of the packings in sets F^1 and F^2, formally:

$$F^1 \oplus F^2 := \{f^i + \overline{f^j} : i = 1, \dots, k, j = 1, \dots, m\}.$$

Intuitively, $F^1 \oplus F^2$ is the set of packings that can be obtained by combining any packing $f^i \in F^1$ with any packing $\overline{f^j} \in F^2$, no matter the sizes of sets F^1 and F^2. It is clear that once set $F(x_1, y_1, z_0)$ has been defined, one can compute sets $F(x_1, y_2, z_0)$, $F(x_1, y_3, z_0)$, …, $F(x_1, y_s, z_0)$, provided all the x_j (resp. y_i) elements belong to ordered set $\mathcal{W}$ (resp. $\mathcal{H}$). In a similar way, the knowledge of set $F(x_1, y_1, z_0)$ allows us to determine sets $F(x_2, y_1, z_0)$, $F(x_3, y_1, z_0)$, …, $F(x_t, y_1, z_0)$. Indeed, it is enough to note that each packing $f \in F(x_j, y_i, z_0)$ that can produce a profit at least equal to z into a $x_j \times y_j$ rectangle can be obtained as the sum of two feasible packings defined for smaller sizes of the bin. Formally, $f = \overline{f} + \hat{f}$ where either $\overline{f} \in F(x_q, y_i, z_0)$ and $\hat{f} \in F(x_j - x_q, y_i, z_0)$ for some $x_q > 0$, or $\overline{f} \in F(x_j, y_q, z_0)$ and $\hat{f} \in F(x_j, y_i - y_q, z_0)$ for some $y_q > 0$. Thus, knowing $F(x_i, H, z_0)$ and $F(W, y_j, z_0)$ for every $i = 1, \dots, t-1$ and $j = 1, \dots, s-1$ we can easily generate $F(H, W, z_0)$ in a recursive fashion.

The basic algorithm can be improved as follows. For each packing f associated with a sheet of width x and height y, an upper bound, say $U(f)$, on the maximum profit that can be obtained with the residual area is computed, to possibly stop the search for packings including f. To this end, consider a knapsack instance with capacity $WH - xy$, n types of items, the j-th available in $a_j - f_j$ copies, each with profit p_j and weight $w_j\, h_j$. The optimal solution of this instance, or any upper bound on this value, gives an upper bound on the maximum profit that can be obtained by packing the remaining items into the residual sheet area. It is clear that all elements $f \in F(x, y, z_0)$ such that $\sum_{i=1}^{n} p_i f_i + U(f) < z_0$ can be deleted from set $F(x, y, z_0)$, since they cannot lead to a feasible solution having profit larger than z_0. In our implementation, we compute the value of upper bound U_2 on the optimal solution of the (one dimensional) knapsack instance (see [25]), the values of upper bounds U_1 and U_2' by Hifi [18] and that[3] proposed for the unconstrained G2KP by Young-Gun and Kang [16], and we use the minimum of such values as upper bound.

In addition, note that the width and height of the sheet can be reduced to x_t and y_s, respectively, which leads to a smaller knapsack capacity when solving the knapsack relaxation, hence to tighter upper bounds. Finally, note that only maximal feasible packing have to be stored for each set $F(x, y, z_0)$, and that any non-maximal element can be disregarded. This reduces the number of elements in $F(x, y, z_0)$, hence the memory requirement and the computing time of the algorithm.

[3] The bound in [16] is based on the dynamic programming solution of KPs for every possible (integer) height and width values. The only slight modification to such computation is an improved discretization in the dynamic programming computation.

```
Procedure Recursive($z_0$)
1. compute $\mathcal{W} = \{x_1, x_2, \dots, x_t\}$
   compute $\mathcal{H} = \{y_1, y_2, \dots, y_s\}$
   $F(x_0, y_i, z_0) := \emptyset,\ i = 1, \dots, s$
2. for $i = 1$ to $s$ do
       for $j = 1$ to $t$ do
           $S := F(x_{j-1}, y_i, z_0)$
           $q = 1$
           while $x_q \le \frac{x_j}{2}$ do
               if there is a $\bar{q}$ such that $x_q + x_{\bar{q}} = x_j$ then
                   $S := S \cup (F(x_q, y_i, z_0) \oplus F(x_{\bar{q}}, y_i, z_0))$
               $q := q + 1$
           $S := S \cup F(x_j, y_{i-1}, z_0)$
           $q = 1$
           while $y_q \le \frac{y_i}{2}$ do
               if there is a $\bar{q}$ such that $y_q + y_{\bar{q}} = y_i$ then
                   $S := S \cup (F(x_j, y_q, z_0) \oplus F(x_j, y_{\bar{q}}, z_0))$
               $q := q + 1$
           if there is an item with dimension $y_i \times x_j$ add it to $S$
           $F(x_j, y_i, z_0) := \{f \in S : p(f) + U(f) \ge z_0,\ f \text{ maximal}\}$
       endfor
   endfor
3. $S := F(x_t, y_s, z_0)$
```

Fig. 3. Procedure *Recursive*.

The overall procedure *Recursive* is depicted in Fig. 3. The C++ source code of the recursive procedure is available upon request from the authors.

3. Two exact algorithms for G2KP

In this section we briefly present two exact algorithms for the G2KP that internally use procedure *Recursive*. Both algorithms take as input the value of an upper bound on the optimal solution value. In addition, one of them (the first) also requires as input the value of a heuristic solution (lower bound) which in our implementation is computed by executing a simple heuristic algorithm described in Section 4.

3.1. Algorithm A1

The first algorithm, denoted as A1, is based on the iterated execution of procedure *Recursive* with different threshold values. The first execution of procedure *Recursive* corresponds to the choice $z_0 = U$, where U is an upper bound on the optimal solution value. In our implementation, we define the upper bound on the optimal solution value

$$U := \min\{U_{kp}, U_{unc}\}, \tag{1}$$

where U_{kp} denotes the optimal solution of the associated non-guillotine two-dimensional instance, and U_{unc} is the optimal solution value of the associated unconstrained two-dimensional instance. In case U_{kp} is unknown, we replaced it with the (weaker) upper bound obtained by the optimal solution of the associated one-dimensional knapsack problem with item weights equal to the rectangle areas (in which case, we used the exact algorithm by Martello and Toth [25]). The value of U_{unc} is computed by applying the exact algorithm by Cintra and Wakabayashi [10].

If no solution exists having profit at least z_0, the procedure returns $S = \emptyset$ in a negligible computing time and a new execution of procedure *Recursive* is performed with a smaller value of z_0. In particular, at each iteration the current value of z_0 is decreased by

$(U-z_H)/10$, where z_H is the value of the initial solution produced by the heuristic algorithm of Section 4. Preliminary computational experiments showed that this way of updating z_0 is more effective than using a standard binary search for our instances. Indeed, in many cases, the upper bound U gives a tight approximation of the optimal solution value, and can be obtained in short computing times. This makes the performances of our approach independent on the initial solution value, thus no computational effort is spent for heuristics. The algorithm ends as soon as an iteration is executed in which the procedure returns $S \neq \emptyset$; in this case the most profitable packing corresponds to the optimal G2KP solution.

3.2. *Algorithm A2*

The second exact approach, denoted as A2, is a branch-and-cut algorithm based on an ILP model of the problem. For the sake of simplicity, in this formulation we consider each item to be distinct, i.e., for each type j of rectangles ($j \in N$), we define a_j identical items having width w_j, height h_j and profit p_j. Let $\overline{n} = \sum_{j=1}^{n} a_j$ be the overall number of items. For each item k we introduce a binary variable x_k taking value 1 if and only if item k is included in the optimal solution. A straightforward ILP model for G2KP is the following:

$$\max \sum_{k=1}^{\overline{n}} p_k x_k \tag{2}$$

$$\sum_{k=1}^{\overline{n}} (w_k h_k) x_k \le WH \tag{3}$$

$$\sum_{k=1}^{\overline{n}} p_k x_k \le U \tag{4}$$

$$\sum_{k \in S_i} x_k \le |S_i|-1, \quad i \in \mathcal{C} \tag{5}$$

$$x_k \in \{0,1\}, \quad k = 1,\ldots,\overline{n}, \tag{6}$$

where U is any upper bound on the optimal solution value and $\mathcal{C}$ denotes the set of all subsets S_i of items that cannot be packed into the sheet with a guillotine pattern. For the threshold value U we used U_{kp}, i.e., the optimal solution value of the associated 2KP instance, corresponding to the relaxation in which guillotine constraints are omitted. Note that constraints (3) and (4) are redundant, but are added to the formulation to strengthen it. Our algorithm solves the relaxed problem in which constraints (5) are eliminated and checks if the current solution x^* is feasible or not by solving the following separation problem: Do all items in $S^* = \{k : x_k^* = 1\}$ fit into a sheet with a guillotine pattern? In case the answer is positive, an optimal solution to G2KP is found. Otherwise, a new violated constraint is found and the process is iterated.

This approach is similar to that proposed by Caprara and Monaci [6] for the exact solution of 2KP and by Pisinger and Sigurd [26] for solving G2KP itself. In particular, algorithm A2 can be seen as a modified (somehow simplified) version of the one in [26]. Precisely, model (2)–(6) is solved by a specialized branch-and-bound in which items are sorted according to non-increasing efficiencies $p_i/(w_i\ h_i)$, and branching is iteratively performed on the most efficient unassigned variable x_i. Upper bounds are derived from the LP relaxation of (2)–(3) by using the upper bound U_2 by Martello and Toth [25]. Backtracking occurs whenever the upper bound does not exceed the current incumbent solution, or when some of the constraints (3)–(5) are violated. Several sophisticated ingredients are used in [26] and are not in A2 and, in addition, a simplification is obtained here by considering constraint (4) and specifically the (strong) bound computed as optimal solution of the 2KP in [6]. Finally, one major difference between A2 and that in [26] is that the separation of inequalities (5) is performed in [26] by solving a *Constraint Satisfaction Problem* (CSP) while, as anticipated, algorithm A2 invokes procedure *Recursive* by assigning each item a profit equal to 1 and a threshold value equal to $|S^*|$.

4. Computational experiments

In this section we computationally analyze the two exact approaches based on the procedure *Recursive* on a classical set of instances from the literature. Algorithm A1 requires on input a heuristic solution value which is computed by the following simple and fast procedure.

Computing a heuristic solution: The procedure defines a random order $\ell_1, \ell_2, \ldots$ of the items and defines a set of items, according to the given order, that allows to improve the incumbent solution value, say z^I. In particular, we determine the first item, say k, such that $\sum_{j=1}^{k} p_{\ell_j} > z^I$, and possibly try to pack all items $\ell_1, \ldots, \ell_k$ into the bin according to a First Fit Decreasing strategy (see [12]). This attempt is not performed if the sum of the areas of these items is larger than the area of the bin, in which case a further iteration is performed. The resulting algorithm, which turns out to be extremely fast in practice, is executed several times, until a given stopping condition is met, returning the value of the best solution found.

Testbed: We considered two classical sets of two-dimensional packing instances that are public available on the web. The first set is taken from the ORLIB library (see [4], web site http://www.ms.ic.ac.uk/info.html), and includes `gcut1-gcut13` (see [3]), `cgcut1-cgcut3` (see [9]), `wang20` (see [29]), and `okp1-okp5` (see [14]). As to the second set of instances, we considered some test problems that were originally proposed by Alvarez-Valdes et al. [1] and later used by Hifi [20] and by Chen [7]. In particular, we considered both the unweighted[4] and weighted large problems, namely problems `APT30` to `APT39` and problems `APT40` to `APT49`, respectively. All these instances are public available at http://www.laria.u-picardie.fr/hifi/OR-Benchmark.

Results: Both algorithms were coded in C++ language and run on an Intel(R) Dual CPU T3400 2.16 GHz with a time limit of 1 h of CPU time. Table 1 gives the outcome of our computational experiments on the first set of instances. For each instance, the table gives:

- The number n of items' types.
- The overall number of items $\overline{n} = \sum_{i=1}^{n} a_i$.
- The size of the sheet, W and H.
- The value z_H of the solution provided by our heuristic with a time limit of 1 min (and 10 min for instance `gcut13`) and the value of the upper bound U on the optimal solution value (see Eq. (1)).
- The computing time in CPU seconds, T_1, and the value of the best solution found, z_1, for algorithm A1. If the time limit has been reached (`T.L.` in column T_1) the reported solution value is a *lower bound*. Otherwise, the entry gives the optimal solution value. The computing time T_1 includes the time required by the heuristic.
- The computing time in CPU seconds, T_2, and the value of the best upper bound found, UB_2, for algorithm A2. If the time limit has been reached (`T.L.` in column T_2) the value reported is an *upper bound*. Otherwise, the entry gives the optimal solution value.

[4] An instance is said to be unweighted if the profit of each item is equal to its area; otherwise the problem is said to be weighted.

Table 1
Results on the instances from the OR-LIBRARY.

Problem					Bounds		A1		A2	
Name	n	$\overline{n}$	W	H	z_H	U	T_1	z_1	T_2	UB_2
gcut1	10	10	250	250	48,368	48,368	2.89	48,368	0.10	48,368
gcut2	20	20	250	250	59,307	59,798	5.56	59,307	7.58	59,307
gcut3	30	30	250	250	59,895	61,275	6.50	60,241	T.L.	61,070
gcut4	50	50	250	250	60,942	61,380	12.95	60,942	T.L.	61,379
gcut5	10	10	500	500	192,907	195,582	4.35	195,582	0.03	195,582
gcut6	20	20	500	500	236,305	236,305	7.13	236,305	0.02	236,305
gcut7	30	30	500	500	238,974	240,143	11.06	238,974	113.00	238,974
gcut8	50	50	500	500	245,758	245,758	18.71	245,758	42.60	245,758
gcut9	10	10	1,000	1,000	919,476	939,600	14.51	919,476	0.28	919,476
gcut10	20	20	1,000	1,000	903,435	937,349	17.85	903,435	6.99	903,435
gcut11	30	30	1,000	1,000	954,366	969,709	29.65	955,389	T.L.	966,150
gcut12	50	50	1,000	1,000	970,744	979,521	46.48	970,744	T.L.	979,426
gcut13	32	32	3,000	3,000	8,532,720	8,736,757	T.L.	8,532,720	T.L.	8,736,757
cgcut1	7	16	10	15	244	244	1.68	244	0.00	244
cgcut2	10	23	70	40	2,681	2,892	0.90	2,892	23.29	2,892
cgcut3	19	62	70	40	1,740	1,860	3.35	1,860	38.57	1,860
wang20	19	42	40	70	2,711	2,726	3.82	2,721	T.L.	2,726
okp1	15	50	100	100	26,441	27,718	2.75	27,589	T.L.	27,718
okp2	30	30	100	100	20,594	22,502	16.43	22,502	0.90	22,502
okp3	30	30	100	100	22,762	24,019	11.14	24,019	0.50	24,019
okp4	33	61	100	100	29,960	32,893	7.69	32,893	58.50	32,893
okp5	29	97	100	100	23,396	27,923	2.37	27,923	T.L.	27,923

Table 2
Results on the instances from http://www.laria.u-picardie.fr/hifi/OR-Benchmark.

Name	n	$\overline{n}$	W	H	Bounds			z_1		T_1
APT30	38	192	152	927			140,904	140,904		2.43
APT31	51	258	964	856	823,976	–	824,931	**823,976**		178.99
APT32	55	249	124	307			38,068	38,068		0.37
APT33	44	224	983	241	236,611	–	236,818	**236,611**		40.62
APT34	27	130	456	795	361,197	–	362,520	**361,398**	*	79.08
APT35	29	153	649	960	621,021	–	622,644	**621,021**		115.36
APT36	28	153	244	537			130,744	130,744		18.06
APT37	43	222	881	440			387,276	387,276		48.03
APT38	40	202	358	731	261,395	–	261,698	**261,395**		44.63
APT39	33	163	501	538			268,750	268,750		33.70
APT40	56	290	138	683	67,154	–	67,654	**67,154**		25.86
APT41	36	177	367	837	206,542	–	215,699	**206,542**		229.30
APT42	59	325	291	167	33,503	–	34,098	33,503	+	T.L.
APT43	49	259	917	362	214,651	–	222,570	214,651	+	T.L.
APT44	39	196	496	223	73,868	–	74,887	**73,868**		19.23
APT45	33	156	578	188	75,808	–	75,888	**74,691**	◦	19.73
APT46	42	197	514	416	149,911	–	151,813	**149,911**		37.00
APT47	43	204	554	393	150,234	–	153,747	**150,234**		51.80
APT48	34	167	254	931	167,660	–	170,914	**167,660**		75.93
APT49	25	119	449	759	218,388	–	226,346	**219,354**	*	2,680.39

The purpose of the experiment whose results are reported in Table 1 is to show that effective approaches for G2KP can be obtained by embedding procedure *Recursive* in different searching schemes. Indeed, the results in Table 1 show that both algorithms A1 and A2 are able to solve a large number of instances in the testbed. Namely, algorithm A1 solves all instances in the testbed but gcut13 (which is known to be very hard) within 50 CPU seconds. On the other hand, A2 reaches the time limit more often (eight times) but is faster than A1 on several instances.

A natural question concerns the impact of procedure *Recursive* within the Pisinger and Sigurd [26] approach. We did test a variant of algorithm A2, closer to the original version [26], in which the separation of constraints (5) is performed by solving a CSP. It turns out that algorithm A2 using our CSP implementation is around 15% slower than the one using procedure *Recursive*. Note, however, that the former is a further simplification of the original Pisinger and Sigurd [26] algorithm which uses a more sophisticated CSP implementation adopting a (not sufficiently described) *limited-depth forward propagation* to derive strong inequalities (5).

In summary, although a strict comparison between the algorithms is outside the scope of the experiment, also because A2 is only a simplified version of the algorithm by Pisinger and Sigurd [26] for G2KP, overall, algorithm A1 seems to perform better than algorithm A2 and we believe that algorithm A1 is a viable and competitive way for solving G2KP.

Table 2 reports the results of algorithm A1 on the second set of instances. For each instance, the table gives:

- The number n of items' types.
- The overall number of items $\overline{n} = \sum_{i=1}^{n} a_i$.
- The size of the sheet, W and H.
- The values of the best lower and upper bound, derived from papers by Hifi [20] and by Chen [7]. If lower and upper bounds coincide, only one entry is given, i.e., the optimal solution value.
- The value of the best solution returned by algorithm A1 within a time limit of 1 h of CPU time. Algorithm A1 uses as initial lower and upper bounds those in Hifi [20] and Chen [7].
- The computing time, in CPU seconds, required by algorithm A1.

The results reported in Table 2 are in general very satisfactory. On the 20 instances of the second testbed algorithm A1 is able to certify optimality in 18 cases, 10 over 10 for the unweighted instances, within the time limit of 1 CPU hour. Specifically, the computing time is always very small, less than 5 CPU minutes, but for the two instances that reach the time limit, namely APT42 and APT43 (denoted in Table 2 by a "+"), and for instance APT49 which requires more than 40 CPU minutes. The algorithm is able to prove optimality for the first time on 13 cases (entries in boldface) and it improves over the best known (heuristic) solution in 2 cases (denoted in Table 2 by a "*"). Instance APT45 (denoted by a "◦") presents some issues. Indeed, the optimal solution found by algorithm A1 has a value, namely 74,691, smaller than the lower bound reported by Chen [7], namely 75,808. We tried to contact the author of [7] without success, so we both tried to reproduce the result by implementing Chen's heuristic and we tried to convince ourselves that no such solution exists by enumerating all KP solutions with such a value[5] and try to pack them in a two-dimensional bin by also using an exact algorithm for the non-guillotine case, i.e., a different algorithm with respect to our current ones. Our implementation of Chen's heuristic could not find the solution of value 75,808 and we failed in constructing such a solution as well, thus we currently believe the value of 74,691 is the optimal one of instance APT45 and the corresponding solution (as well as the others) are available upon request from the authors.

5. Conclusions

In the context of two-dimensional cutting problems, we have presented a recursive exact procedure that, given a set of items and a unique sheet, constructs the corresponding maximal guillotine packings. Such a procedure can be potentially embedded into exact and heuristic algorithms for solving several types of guillotine two-dimensional problems. Specifically, we have embedded the procedure in two exact methods for the guillotine two-dimensional knapsack and these algorithms have been computationally evaluated on well-known benchmark instances from the literature.

The C++ source code of the recursive procedure is available upon request from the authors.

Acknowledgments

Part of this research was done when the first author was visiting the Department of Electronics, Computer Science and Systems (DEIS) of the University of Bologna under the support of the Ministry of Science, Research and Development of the Islamic Republic of Iran. We are grateful to two anonymous referees for useful comments.

References

[1] Alvarez-Valdes R, Parajon A, Tamarit JM. A tabu search algorithm for large-scale guillotine (un)constrained two-dimensional cutting problems. Computers and Operations Research 2002;29:925–47.
[2] Bansal N, Lodi A, Sviridenko M. A tale of two dimensional bin packing. In: Proceedings of the 46th annual IEEE symposium on foundations of computer science (FOCS 2005). IEEE Computer Society Press; 2005. p. 657–66.
[3] Beasley JE. Algorithms for unconstrained two-dimensional guillotine cutting. Journal of Operational Research Society 1990;36:297–306.
[4] Beasley JE. Or-library: distributing test problems by electronic mail. Journal of Operational Research Society 1990;41:1069–72.
[5] Boschetti MA, Hadjiconstantinou E, Mingozzi A. New upper bounds for the two-dimensional orthogonal non guillotine cutting stock problem. IMA Journal of Management Mathematics 2002;13:95–119.
[6] Caprara A, Monaci M. On the two-dimensional knapsack problem. Operations Research Letters 2004;32:5–14.
[7] Chen Y. A recursive algorithm for constrained two-dimensional cutting problems. Computational Optimization and Applications 2008;41:337–47.
[8] Christofides N, Hadjiconstantinou E. An exact algorithm for orthogonal 2-D cutting problems using guillotine cuts. European Journal of Operational Research 1995;83:21–38.
[9] Christofides N, Whitlock C. An algorithm for two-dimensional cutting problems. Operations Research 1977;25:30–44.
[10] Cintra G, Wakabayashi Y. Dynamic programming and column generation based approaches for two dimensional guillotine cutting problems. In: III Worskshop on efficient and experimental algorithms (WEA 2004) lecture notes in computer science, vol. 3059. Springer; 2004. p. 175–90.
[11] Clautiaux F, Jouglet A, Moukrim A. A new graph-theoretical model for k-dimensional guillotine-cutting problems. In: 7th International workshop on experimental algorithms (WEA 2008) lecture notes in computer science, vol. 5038. Springer; 2008. p. 43–54.
[12] Coffman EG, Garey MR, Johnson DS, Tarjan RE. Performance bounds for level-oriented two-dimensional packing algorithms. SIAM Journal on Computing 1980;9:801–26.
[13] Cung VD, Hifi M, Cun B. Constrained two-dimensional cutting stock problems: a best-first branch-and-bound algorithm. International Transactions in Operational Research 2000;7:185–210.
[14] Fekete SP, Schepers J. A new exact algorithm for general orthogonal d-dimensional knapsack problems. In: Algorithms (ESA 97) lecture notes in computer science, vol. 1284. Springer; 1997. p. 144–56.
[15] Fekete SP, Schepers J, van der Veen J. An exact algorithm for higher-dimensional orthogonal packing. Operations Research 2007;55:569–87.
[16] Young-Gun G, Kang MK. A new upper bound for unconstrained two-dimensional cutting and packing. Journal of the Operational Research Society 2002;53:587–91.
[17] Gilmore PC, Gomory RE. A linear programming approach to the cutting stock problem. Operations Research 1961;9:849–59.
[18] Hifi M. An improvement of Viswanathan and Bagchi's exact algorithm for constrained two-dimensional cutting stock. Computers and Operations Research 1997;24:727–36.
[19] Hifi M. Exact algorithms for large-scale unconstrained two and three staged cutting problems. Computational Optimization and Applications 2001;18:63–88.
[20] Hifi M. Dynamic programming and hill-climbing techniques for constrained two-dimensional cutting stock problems. Journal of Combinatorial Optimization 2004;8:65–84.
[21] Hifi M, M'Hallah R. An exact algorithm for constrained two-dimensional two-staged cutting problems. Operations Research 2005;53:140–50.
[22] Jansen K, Zhang G. On rectangle packing: maximizing benefits. Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms, 2004. p. 204–13.
[23] Lodi A, Monaci M. Integer linear programming models for 2-staged two-dimensional knapsack problems. Mathematical Programming 2003;94: 257–78.
[24] Macedo R, Alves C, de Carvalho V. Arc-flow model for the two-dimensional guillotine cutting stock problem. Computers and Operations Research 2010;37:991–1001.
[25] Martello S, Toth P. Knapsack problems: algorithms and computer implementations. Chichester: John Wiley & Sons; 1990.
[26] Pisinger D, Sigurd M. Using decomposition techniques and constraint programming for solving the two-dimensional bin-packing problem. INFORMS Journal on Computing 2007;19:36–51.
[27] Puchinger J, Raidl JR. Models and algorithms for three-stage two-dimensional bin packing. European Journal of Operational Research 2007;183:1304–27.
[28] Vanderbeck F. A nested decomposition approach to a 3-stage 2-dimensional cutting stock problem. Management Science 2001;47:864–79.
[29] Wang PJ. Two algorithms for constrained two-dimensional cutting stock problems. Operations Research 1983;31:573–86.
[30] Wäscher G, Haussner H, Schumann H. An improved typology of cutting and packing problems. European Journal of Operational Research 2007;183: 1109–30.

[5] As usual, the weight of an item in the one-dimensional knapsack is given by its area.