Ann Oper Res (2010) 179: 297–315
DOI 10.1007/s10479-008-0457-4

A heuristic approach based on dynamic programming and and/or-graph search for the constrained two-dimensional guillotine cutting problem

Reinaldo Morabito · Vitória Pureza

Published online: 13 November 2008

Abstract In this paper we present a heuristic method to generate constrained two-dimensional guillotine cutting patterns. This problem appears in different industrial processes of cutting rectangular plates to produce ordered items, such as in the glass, furniture and circuit board business. The method uses a state space relaxation of a dynamic programming formulation of the problem and a state space ascent procedure of subgradient optimization type. We propose the combination of this existing approach with an and/or-graph search and an inner heuristic that turns infeasible solutions provided in each step of the ascent procedure into feasible solutions. Results for benchmark and randomly generated instances indicate that the method's performance is competitive compared to other methods proposed in the literature. One of its advantages is that it often produces a relatively tight upper bound to the optimal value. Moreover, in most cases for which an optimal solution is obtained, it also provides a certificate of optimality.

Keywords Cutting and packing problems · Constrained two-dimensional guillotine cutting patterns · Dynamic programming · And/or-graph search · Heuristics

1 Introduction

Cutting and packing problems appear in a wide range of classes and practical situations and generally they are of difficult exact solutions (e.g. Dyckhoff and Finke 1992; Lodi et al. 2002; Waescher et al. 2007). Many studies dealing with these problems can be found in the literature, as shown in different surveys and special issues such as in Dyckhoff and Waescher (1990), Dowsland and Dowsland (1992), Sweeney and Paternoster (1992), Martello (1994a, 1994b), Bischoff and Waescher (1995), Mukhacheva (1997), Dyckhoff

R. Morabito (✉) · V. Pureza
Production Engineering Department, Universidade Federal de São Carlos, São Carlos SP, Brazil
e-mail: morabito@ufscar.br

V. Pureza
e-mail: vpureza@dep.ufscar.br

et al. (1997), Arenales et al. (1999), Wang and Waescher (2002), Hifi (2002), Oliveira and Waescher (2007). Additional references can be found in ESICUP (2008).

In this paper, we study the generation of constrained two-dimensional guillotine cutting patterns. Besides the inherent complexity of the problem (NP-hard; see e.g. Fayard et al. 1998; Hifi 2004), we are motivated by its practical relevance in many industrial settings, such as in the cutting of flat sheets in glass cutting processes, the cutting of wood boards in furniture factories, the cutting of fiberglass plates in the circuit board manufacturing, among others. The cutting pattern is called two-dimensional as it involves two relevant dimensions (the lengths and widths of the items). We assume that all cuts must be orthogonal, i.e., parallel to one side of the rectangle. The cutting pattern is called guillotine because, due to constraints imposed by the cutting equipment, the cuts must be of a guillotine type (i.e. they must run from end to end on the rectangle being cut), and it is called constrained because, besides the non overlapping restraints, there are imposed limitations on the maximum number of items that can be produced in a pattern.

Only a few studies proposed exact methods to generate constrained two-dimensional guillotine cutting patterns in the literature. For example, Christofides and Whitlock (1977) presented a tree search algorithm with dynamic programming and a transportation routine, Viswanathan and Bagchi (1993) proposed a best-first search method based on Wang (1983)'s bottom-up approach, Hifi (1997a) presented an improvement of Viswanathan and Bagchi (1993)'s algorithm, and Cung et al. (2000) presented a new version of Hifi (1997a)'s procedure. In particular, Christofides and Hadjiconstantinou (1995) developed a tree search algorithm using bounds derived from a state space relaxation of a dynamic programming formulation and a state space ascent procedure, which is an improvement of Christofides and Whitlock (1977)'s algorithm. The application of the method was illustrated by solving only problem instances of moderate sizes. Due to the complexity and combinatorial explosion of the exact methods, various studies proposed heuristic methods to generate constrained guillotine patterns, as in the papers by Wang (1983), Vasko (1989), Oliveira and Ferreira (1990), Parada et al. (1995), Morabito and Arenales (1996), Mornar and Khoshnevis (1997), Fayard et al. (1998), Parada et al. (1998, 2000), Alvarez-Valdes et al. (2002), Hifi (2004) and Cui (2008); see also Burke et al. (2004) and Dowsland et al. (2006).

In the present study, we combine a variant of Christofides and Hadjiconstantinou (1995)'s method with an and/or-graph search approach. Instead of using a tree search algorithm and performing a complete enumeration, we apply an inner heuristic (called feasibility heuristic) that turn infeasible solutions provided in each step of the subgradient optimization procedure into feasible solutions as an attempt to improve the lower bound. Moreover, we apply an and/or-graph approach in order to start the subgradient optimization procedure with a tight lower bound to the optimal solution value. Unlike Christofides and Hadjiconstantinou's method, our approach does not have a guarantee of optimality; however, it is faster and able to provide high-quality solutions (if not optimal) within reasonable computer runtimes for larger problem instances, as shown in our numerical experiments.

One advantage of the present approach compared to other heuristic methods is that in the cases for which an optimal solution is found, the subgradient procedure may provide a certificate of optimality. Moreover, when the best solution is suboptimal or its optimality cannot be proved, the approach often produces a relatively tight upper bound to the optimal value, which allows a good estimate of the optimality gap. It is worth mentioning that the method ensures optimality when solving the unconstrained two-dimensional guillotine cutting case (the complexity of this case is pseudo-polynomial; Beasley 1985). To evaluate the performance of the method, we solve benchmark and randomly generated examples with varying degrees of difficulty and compare the results to competitive methods of the literature.

The paper is organized as follows. Sections 2 and 3 briefly review the dynamic programming formulation of the problem, the state space relaxation of the formulation and the ascent procedure of a subgradient optimization type. In Sects. 4 and 5, we describe the and/or-graph approach and we present the feasibility heuristic used in the subgradient optimization procedure. In Sect. 6, the computational performance of the method is analyzed followed by concluding remarks and perspectives for future research in Sect. 7.

2 Dynamic programming formulation

Consider a rectangular plate of length and width $L \times W$, and a set of n smaller rectangles (sizes of ordered items or pieces) of lengths and widths $l_1 \times w_1, l_2 \times w_2, \ldots, l_n \times w_n$, demands $b_1, b_2, \ldots, b_n$, and values $v_1, v_2, \ldots, v_n$. We assume that the pieces have a fixed orientation (i.e., sizes $l_i \times w_i$ and $w_i \times l_i$ refer to different pieces) and the lengths and widths of the plate and pieces are integer numbers. The constrained cutting problem consists of generating the most valuable guillotine cutting pattern using no more than b_i pieces of type $i = 1, 2, \ldots, n$ (should the pattern also use at least a_i pieces of type $i = 1, 2, \ldots, n$, the problem would be referred to as double constrained by a_i and b_i). Moreover, the problem is called unweighted if all piece values v_i are equal to the corresponding piece areas $l_i w_i$ (the objective consists of determining the minimum trim loss pattern); otherwise it is weighted.

Christofides and Hadjiconstantinou (1995) proposed a dynamic programming formulation for the constrained guillotine problem assuming that the cuts on the plate are produced in consecutive stages. Figure 1 illustrates a 3-stage cutting pattern to a rectangle (x, y). In the first stage, a horizontal cut divides (x, y) into two rectangles (top and bottom); in the second stage a vertical cut divides the top rectangle into two rectangles (left top and right top) and two vertical cuts divide the bottom rectangle into three rectangles (left bottom, middle bottom and right bottom). Finally, in the third stage a horizontal cut divides the left top rectangle into two rectangles and two horizontal cuts divide the right bottom rectangle into three rectangles.

Let $s_1, s_2, \ldots, s_n$ be the amount of pieces of types $1, 2, \ldots, n$ available to produce a feasible cutting pattern to rectangle (x, y). Representing these pieces by the ordered sequence $S_{xy} = \{s_1, s_2, \ldots, s_n\}$, let $F_k(x, y, S_{xy})$ be the value of the optimal guillotine cutting pattern produced with at most k cutting stages to rectangle (x, y), using a feasible combination of some or all of the rectangles available in set S_{xy}, with the cuts of the first stage parallel to axis y. Similarly, let $G_k(x, y, S_{xy})$ be the value of the optimal guillotine cutting pattern with at most k cutting stages to rectangle (x, y), using a feasible combination of some or all of the rectangles available in set S_{xy}, with the cuts of the first stage parallel to axis x. Defining the sets $X = \{1, 2, \ldots, L\}$ and $Y = \{1, 2, \ldots, W\}$, the dynamic programming recursive functions $F_k(x, y, S_{xy})$ and $G_k(x, y, S_{xy})$, for all $k \geq 1, x \in X, y \in Y$ and

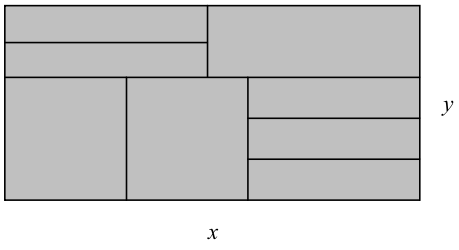

Fig. 1 Example of a 3-stage cutting pattern to a rectangle (x, y)

$S_{xy} \subseteq S_{LW} = \{b_1, b_2, \ldots, b_n\}$, are given by:

$$F_k(x, y, S_{xy}) = \max\Big[G_{k-1}(x, y, S_{xy}), \max_{x'<x, x' \in X, S' \subset S_{xy}} \big[F_k(x', y, S') + G_{k-1}(x - x', y, S_{xy} - S')\big]\Big], \quad (1)$$

$$G_k(x, y, S_{xy}) = \max\Big[F_{k-1}(x, y, S_{xy}), \max_{y'<y, y' \in Y, S' \subset S_{xy}} \big[G_k(x, y', S') + F_{k-1}(x, y - y', S_{xy} - S')\big]\Big]. \quad (2)$$

The first term inside the brackets of the recursive formula (1) corresponds to the case in which, in the first stage of the optimal pattern with at most k stages of (x, y), there are no cuts on (x, y) parallel to axis y. This implies that the pattern can be considered as produced with at most $k - 1$ stages, with the cuts of the first stage parallel to axis x. The second term inside the brackets of formula (1) corresponds to the case in which, in the first stage of the optimal pattern with at most k stages of (x, y), there is at least one cut on (x, y) parallel to axis y, say a vertical cut $x' \in X, x' < x$, producing two rectangles (x', y) and $(x - x', y)$. In this case, we have two cutting patterns associated to these rectangles. In the first, the pieces belong to set S', the cuts of the first stage of the optimal pattern of (x', y) are parallel to axis y, and there are k stages in the total (given that it may have another vertical cut $x'' < x'$ on (x', y), which does not imply an additional stage on (x, y)). In the second case, the pieces belong to the complementary set $S_{xy} - S'$, the cuts of the first stage of the optimal pattern of $(x - x', y)$ are parallel to axis x, and there are only $k - 1$ stages in the total (given that horizontal cuts on $(x - x', y)$ imply in an additional stage on (x, y)). These considerations are applicable in a similar way to the recursive formula (2).

The value $k = 0$ in $F_k(x, y, S_{xy})$ or $G_k(x, y, S_{xy})$ (formulas (1) and (2)) corresponds to the allocation of the most valuable piece in set S_{xy} to rectangle (x, y). Vertical and/or horizontal trimming cuts necessary to produce this piece on (x, y) are not considered as additional cutting stages. Thus, the initial conditions for formulas (1) and (2), for all (x, y) and S_{xy}, are:

$$F_0(x, y, S_{xy}) = \max\Big[0, \max_{i \in S_{xy}}[v_i \mid l_i \le x, w_i \le y]\Big], \quad (3)$$

$$G_0(x, y, S_{xy}) = F_0(x, y, S_{xy}). \quad (4)$$

Note that the maximum between $F_k(x, y, S_{xy})$ and $G_k(x, y, S_{xy})$ in (1)–(4) corresponds to an optimal constrained guillotine pattern with at most k cutting stages to rectangle (x, y), generated using the pieces in set S_{xy}, when the direction of the cuts of the first stage is not specified. In particular, $\max[F_k(L, W, S_{LW}), G_k(L, W, S_{LW})]$ corresponds to an optimal constrained k-stage guillotine pattern to plate (L, W).

In the present paper, we assume that a maximum number of stages is not specified; therefore, we should consider the value of k as the one for which: $F_k(L, W, S_{LW}) = F_{k+1}(L, W, S_{LW})$ and $G_k(L, W, S_{LW}) = G_{k+1}(L, W, S_{LW})$. Without loss of generality, the cardinalities of sets X and Y in formulas (1)–(4) can be substantially reduced using the restraint of canonical or normal patterns (Christofides and Whitlock 1977; Beasley 1985). Taking this into account, these sets can be redefined as:

$$X = \left\{x \,\middle|\, x = \sum_{i=1}^{n} \theta_i l_i,\ 1 \le x \le L, 0 \le \theta_i \le b_i \text{ and } \theta_i \text{ integer},\ i = 1, \ldots, n\right\},$$

$$Y = \left\{y \,\middle|\, y = \sum_{i=1}^{n} \tau_i w_i,\ 1 \le y \le W,\ 0 \le \tau_i \le b_i \text{ and } \tau_i \text{ integer},\ i = 1, \ldots, n\right\}.$$

Note in formulation (1)–(4), in order to generate an optimal unconstrained two-dimensional guillotine cutting pattern it is enough to relax the state variable S_{LW} so that the formulas are reduced to the ones presented in Beasley (1985) and Gilmore and Gomory (1966). In this (unconstrained) case the complexity is pseudo-polynomial, i.e. $O(nLW(L+W))$ (note that by redefining X and Y as the normal pattern sets above, it is $O(n|X||Y|(|X|+|Y|))$ as pointed out in Beasley 1985). In fact, the main difficulty of formulation (1)–(4) is the size of the state space associated to S_{LW}.

3 Solution method based on a state space relaxation

In this section we review a relaxation of the state space of (1)–(4) combined with a subgradient optimization procedure, as explored in Christofides and Hadjiconstantinou (1995). This approach often produces tight bounds to the original problem and it was also applied to travelling salesman, set covering and network flow problems (Christofides et al. 1981; Hadjiconstantinou and Christofides 1995; Fontes et al. 2006). In the sequel, we develop a feasibility heuristic that in each iteration of the subgradient optimization procedure, modifies the relaxed solution as an attempt to improve the incumbent feasible solution to the problem. It should be mentioned that Christofides and Hadjiconstantinou (1995) did not explore feasibility heuristics in the subgradient optimization, but applied the resulting bounds to a complete enumeration procedure (tree search algorithm) in order to ensure that optimal solutions to the problem could be found.

3.1 Review of the state space relaxation

Consider the formulation (1)–(4) and let $g(.)$ be an integer scalar mapping function from a state domain (x, y, S_{xy}) to another domain $(x, y, g(S_{xy}))$ of smaller cardinality. For instance, defining $g(S_{xy}) = \sum_{s_i \in S_{xy}} s_i$ as the total quantity of available pieces in S_{xy}, then all subsets $S' \subset S_{xy}$ that result in the same value $g(S')$ are represented by only one point in this new domain. Therefore, this is a relaxation of the original state space. Rewriting the recursive formula (1) to the relaxed space, for all $k \geq 1$, $x \in X$, $y \in Y$ and $g(S_{xy})$, we obtain (similarly to the recursive formula (2)):

$$F_k(x, y, g(S_{xy})) = \max \left[\begin{array}{l} G_{k-1}(x, y, g(S_{xy})), \\ \max\limits_{x' < x, x' \in X, g(S') < g(S_{xy})} [F_k(x', y, g(S')) + G_{k-1}(x - x', y, g(S_{xy} - S'))] \end{array} \right]. \quad (5)$$

Since this is a relaxation of the recursive formulation (1)–(4), $F_k(x, y, g(S_{xy})) \geq F_k(x, y, S_{xy})$ and $G_k(x, y, g(S_{xy})) \geq G_k(x, y, S_{xy})$. Hence, $\max[F_k(x, y, g(S_{xy})), G_k(x, y, g(S_{xy}))]$ is an upper bound to the optimal solution of (1)–(4) with at most k cutting stages (proof of this statement can be found in Christofides et al. 1981). Christofides and Hadjiconstantinou (1995) associate a non-negative integer weight (multiplier) q_i to each piece type i and define:

$$g(S_{xy}) = q \equiv \sum_{s_i \in S_{xy}} s_i q_i, \quad (6)$$

$$g(S') = q' \equiv \sum_{s_i \in S'} s_i q_i. \quad (7)$$

Then, $g(S_{xy} - S')$ is easily computable by $q - q'$ using (6) and (7). Based on (5)–(7), formulas (1)–(4) can be rewritten as:

$$F_k(x, y, q) = \max\Big[G_{k-1}(x, y, q), \max_{x' < x, x' \in X, q' = 0, \ldots, q} [F_k(x', y, q') + G_{k-1}(x - x', y, q - q')]\Big], \quad (8)$$

$$G_k(x, y, q) = \max\Big[F_{k-1}(x, y, q), \max_{y' < y, y' \in Y, q' = 0, \ldots, q} [G_k(x, y', q') + F_{k-1}(x, y - y', q - q')]\Big], \quad (9)$$

$$F_0(x, y, q) = \max\Big[0, \max_{i=1,\ldots,n} [v_i \mid l_i \le x, w_i \le y, q_i \le q]\Big], \quad (10)$$

$$G_0(x, y, q) = F_0(x, y, q). \quad (11)$$

The relaxed formulation (8)–(11) is applied to all $k \ge 1$, $x \in X$, $y \in Y$ and $q = 0, \ldots, Q$, where $Q = \sum_{b_i \in S_{LW}} b_i q_i$ for a given set of weights $q_1, q_2, \ldots, q_n$. This calculation involves $O(n|X||Y|Q(|X|Q + |Y|Q))$ operations (note that Q varies with the problem instance and the set of weights). It can be used to obtain an upper bound Z_{UB} to the optimal solution of formulation (1)–(4) with at most k stages, for a given set of weights $q_1, q_2, \ldots, q_n$. This upper bound is defined by: $Z_{UB} = \max[F_k(L, W, Q), G_k(L, W, Q)]$. Note that for $q_1 = 0$, $q_2 = 0, \ldots, q_n = 0$ and k sufficiently large, the bound coincides with the value of the optimal unconstrained two-dimensional guillotine cutting pattern.

Let γ_i be the number of times that piece i appears in the solution (cutting pattern) generated by relaxed formulation (8)–(11). If $\gamma_i \le b_i$ for all $i = 1, \ldots, n$, then this relaxed solution is also feasible to formulation (1)–(4) and, therefore, it is optimal. Otherwise, a state space ascent procedure of subgradient optimization type can be used to minimize the upper bound Z_{UB}, by varying the weights $q_1, q_2, \ldots, q_n$. The objective is to penalize the infeasibilities so that the resulting solution of the relaxed problem is close to a feasible solution. Let $\boldsymbol{q}$ be the weight vector $(q_1, q_2, \ldots, q_n)$, $f(\boldsymbol{q})$ be the upper bound Z_{UB} using vector $\boldsymbol{q}$, and D be the maximum size of the relaxed state space (i.e., the maximum value of Q). The following optimization problem has to be solved:

$$f(q^*) = \min_{q \ge 0}[f(q)],$$

$$\text{s.t.:} \quad \sum_{b_i \in S_{LW}} b_i q_i \le D.$$

This problem is generally difficult to solve because $f(\boldsymbol{q})$ is a discontinuous function in $\boldsymbol{q}$. However, a subgradient optimization procedure can be used to modify the weights q_i and ascend in the state space. The procedure is based on the following principle: if $\gamma_i \le b_i$ for all $i = 1, \ldots, n$, then formulation (1)–(4) was optimally solved; otherwise, try to reduce the difference $\gamma_i - b_i$ in order to move the resulting solutions into the feasible space. A simple way to do this is to increment the weight q_i of the pieces for which $\gamma_i > b_i$ and, at the same time, decrement the weight q_i of the pieces for which $\gamma_i \le b_i$. In the present work, we use the following formula to update the values of the weights, as proposed in Christofides and Hadjiconstantinou (1995):

$$q_i = \begin{cases} q_i + \lfloor t\sqrt{(\gamma_i - b_i)} \rfloor & \text{if } \gamma_i > b_i, \\ \max[0, q_i - \lfloor t\sqrt{(b_i - \gamma_i)} \rfloor] & \text{if } \gamma_i \le b_i, \end{cases} \quad (12)$$

where $\lfloor z \rfloor$ denotes the largest integer less than or equal to z, and t is the size of the positive scalar step, given by:

$$t = \max\left[1, \sqrt{\pi(Z_{UB} - Z_{LB}) / \sum_{i=1}^{m} (b_i - \gamma_i)^2}\right]. \tag{13}$$

Expression (13) is a slight modification of the one used in Christofides and Hadjiconstantinou (1995). Our motivation for the lower limit $t = 1$ is to allow the change of q_i in (12) when $\gamma_i \neq b_i$. The parameter value π is initially fixed and then it is halved along the iterations until it becomes less than an exogenous parameter ε.

3.2 Proposed approach

Let Z_{LB} be a lower bound to the optimal solution of formulation (1)–(4). It corresponds to a feasible solution to (1)–(4) and it is initialized using the and/or-graph approach described in Sect. 4. Z_{UB} is the value of the current upper bound to (1)–(4). Note that if $Z_{UB} = Z_{LB}$, then the subgradient optimization procedure can be halted, returning Z_{LB} as the optimal solution value of (1)–(4). The steps of the subgradient optimization algorithm are described as follows.

Subgradient optimization algorithm

1. Compute the normal sets X and Y (Sect. 2) and initialize the lower bound Z_{LB} using the and/or-graph approach (described in Sect. 4). Make the weights $q_i = 0$, $i = 1, \ldots, n$.
2. Solve the relaxed formulation (8)–(11) with a sufficiently large k to obtain the optimal value Z_{UB} relative to the current vector $\boldsymbol{q}$.
3. Verify if the cutting pattern associated to Z_{UB} corresponds to a feasible solution to formulation (1)–(4). In this case, the solution is optimal and the procedure is halted. Otherwise, if $Z_{UB} < Z_{min}$ (i.e. the minimum upper bound obtained so far), then update Z_{min} with $Z_{min} = Z_{UB}$.
4. Apply the feasibility heuristic (described in Sect. 5) to find a feasible solution to (1)–(4). If the value of this solution is greater than Z_{LB} (i.e. the value of the best feasible solution obtained so far), then update Z_{LB}. If $Z_{LB} = Z_{min}$, then this solution is optimal and the procedure is terminated.
5. Modify vector $\boldsymbol{q}$ using expressions (12)–(13) and update Q and π.
6. Go to step 2 with the modified vector $\boldsymbol{q}$, unless the maximum number of iterations is reached or $\pi \leq \varepsilon$.

At the end of the algorithm, an optimal solution to formulation (1)–(4) may be found (steps 3 or 4). Otherwise, the returned solution is either the solution obtained in step 1 with the and/or-graph approach or the best solution obtained in step 4 with the feasibility heuristic, which does not ensure optimality. The main differences between this algorithm and the one used in Christofides and Hadjiconstantinou (1995) are in steps 1 and 4.

Lucena (2004) proposed some modifications of relax and cut type in subgradient optimization procedures. In particular, it was suggested that a good practice for the convergence of the algorithm is to adjust the subgradients before the computation of step size t in (13), using $s_i = 0$ if $s_i \geq 0$ and $q_i = 0$, where $s_i = b_i - \gamma_i$ is the subgradient associated to the restraint of availability of piece i. In other words, if constraint i is not violated (i.e., $s_i \geq 0$) in

current iteration, and it has a null multiplier associated (i.e., $q_i = 0$), then its subgradient is ignored, fixing it to $s_i = 0$. Taking this into account, it does not contribute to the calculus of the step size t in (13). This strategy of adjusting multipliers was also applied to the present study.

4 And/or-graph approach

In step 1 of the subgradient optimization algorithm of Sect. 3 we use the and/or-graph approach presented in Morabito and Arenales (1994, 1996) to produce an initial feasible solution of high quality (i.e. a tight lower bound). Other related methods to the and/or-graph approach are found in Parada et al. (1995, 2000) and Hifi (1997b, 2004). This approach (denoted by AOG) is an implicit enumeration and represents each possible constrained two-dimensional guillotine cutting pattern as a complete path in an and/or-graph such that: (i) the nodes of the graph correspond to either the initial rectangle (plate), the intermediary rectangles obtained during the cutting process, the ordered rectangles (pieces) or the waste (trim losses) of the pattern, and (ii) the and-arcs of the graph correspond to the cuts on the rectangles, linking a node to other nodes. Figure 2 illustrates a sequence of five guillotine cuts C_1, C_2, ..., C_5, producing a cutting pattern containing the rectangles (pieces) denominated R_8, R_9, R_{10}, R_{13}, R_{16} and R_{17}. Figure 3 presents the correspondent complete path of the and/or-graph with 17 nodes (rectangles $R_1, R_2, \ldots, R_{17}$) and 5 and-arcs (cuts $C_1, C_2, \ldots, C_5$). Note that the horizontal cut C_1 on rectangle R_1 (initial node) produces the intermediary rectangles R_2 and R_3, the vertical cut C_2 on rectangle R_2 produces rectangles R_4 and R_5 (which cannot be cut further and results in pieces R_8 and R_9 after a 0-cut, i.e., a cut that represents the option of stop cutting the rectangle, linking it to a final node), the vertical cut C_3 on rectangle R_3 produces the intermediary rectangles R_6 and R_7, and so on.

To reduce the graph search, the AOG uses the normal sets X and Y defined in Sect. 2 and applies rules of symmetry, exclusion and cut ordering. Moreover, the AOG discards non-promising paths using hybrid search strategies (combinations of *backtracking* and *hill-climbing*) and applying some heuristics (heuristics H1–H4). For instance, heuristic H1 discards a path from a node if the upper bound of this path value is not $\lambda_1\%$ greater than the value of the incumbent solution of this node. Heuristic H2 discards a path from a node if the lower bound of this path value is not at least $\lambda_2\%$ of the value of the incumbent solution of this node. Heuristic H3 imposes a symmetry rule with loss of generality to reduce the graph search. As the graph search substantially increases with the cardinalities of sets X and Y, heuristic H4 imposes a limit M on the maximum number of elements of X and Y. The quality of the solutions generated by the AOG is highly dependent on these heuristics' parameters. Further details can be found in Morabito et al. (1992) and Morabito and Arenales

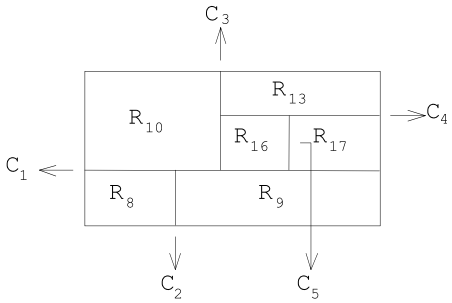

Fig. 2 Sequence of guillotine cuts producing a cutting pattern

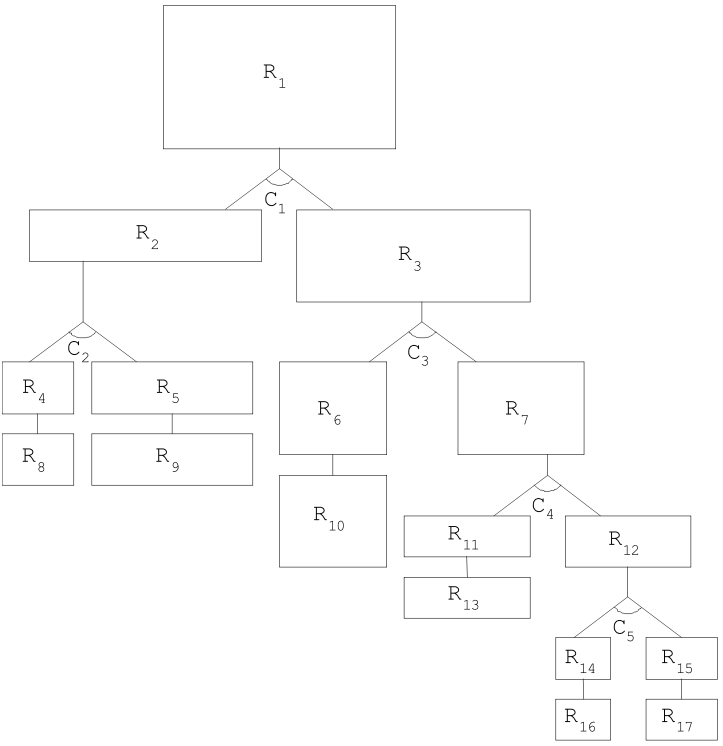

Fig. 3 Complete path of the and/or-graph representing the cutting pattern of Fig. 2

(1994, 1996). The performance of the and/or-graph search could be also reduced by means of more effective lower bounds (feasible solutions) in the graph nodes, as the one used in Hifi (2004) based on the computation of a series of single bounded knapsack problems.

5 A feasibility heuristic

As mentioned in Sect. 3, step 4 of the subgradient optimization algorithm comprises an inner heuristic that turn infeasible solutions provided in a given step of the algorithm into feasible solutions. The motivation for using this heuristic is that it may improve the instance's current lower bound. In this case, if a feasible solution with value Z_{min} is found during the procedure, then this solution must be optimal. Otherwise, the best solution produced by the feasibility heuristic is returned along with the optimal solution upper bound Z_{min}.

The feasibility heuristic consists of replacing *blocks* of type i pieces for which the demand has been exceeded by other piece types j that still have a demand to be fulfilled. Block B_i is a rectangular structure in the current pattern comprising one or more adjacent i pieces. Its exclusion results in a rectangular empty area S_i which may also include unused spaces around B_i's outer perimeter. If there is more than one unit of piece i in excess, B_i comprises a number of adjacent pieces as close as possible to the number in excess. In the case of exactly one unit in excess, B_i corresponds to this single piece or, if the piece area does not allow the inclusion of any eligible piece type, the procedure searches for a block containing

two adjacent units of i. Note that by removing two or more adjacent pieces at once, instead of a piece at a time, a more efficient use of the empty area is achieved.

Once B_i is removed, the resulting area S_i can be filled with eligible pieces j. For each type i piece in excess, the heuristic selects the combination (i, S_i) that provides the most valuable homogeneous (arrangement of pieces with the same type) solution. If the arrangement does not completely fill S_i, a horizontal and/or a vertical guillotine cut are applied to obtain up to two rectangles corresponding to the unused areas, which are also subject to the arrangement of pieces, taking into account the pieces used in the previous level. We disregard them if these cuts add further stages to the cutting pattern, since we are assuming that a maximum number of stages is not specified. However, only arrangements that maintain a guillotine cutting pattern of the plate are considered. This defines a recursive scheme applied as long as there is enough area to be filled with eligible pieces j.

Therefore, rather than a single arrangement for each combination (i, S_i), the heuristic stores a sequence of arrangement decisions A_i for each combination (i, S_i). The sequence of decisions A_i^* that provides the most valuable cutting pattern is implemented followed by the update of the demand for used pieces. The procedure is repeated until a feasible solution is obtained. The steps of the feasibility heuristic are described as follows.

Feasibility heuristic

1. Let $I =$ set of types with pieces in excess, and $J =$ set of types for which the demand is not fulfilled in the current pattern.
2. For each type $i \in I$ with e_i pieces in excess do:
   2.1 Select a block B_i in the current pattern such that: (i) the number of pieces is close as possible to e_i, and (ii) its exclusion produces the largest empty area S_i. If $e_i = 1$ and it is not possible to fill the area with another piece type $j \in J$, and there is a block comprising at least two pieces of type i, make the number of pieces to be removed equal to 2 and define B_i and S_i accordingly. Remove B_i from the current pattern.
3. For each area S_i resulting from step 2:
   3.1 Let A_i (set of selected homogeneous arrangements) $= \emptyset$ and $S_{current} = S_i$.
   3.2 Determine the most valuable homogeneous solution H_i^* among pieces $j \in J$ that can be placed in space $S_{current}$, without exceeding the demand of j and that maintains the guillotine pattern of the plate. Make $A_i = A_i \cup H_i^*$.
   3.3 If H_i^* does completely fill $S_{current}$, obtain up to two rectangles corresponding to the unused areas by means of a horizontal and a vertical guillotine cut. For each rectangle resulting from each type of cut, verify if the area is large enough to be filled with pieces $j \in J$ considering the arrangement decisions in A_i. In this case, make the area equal to $S_{current}$ and go to step 3.2, storing the arrangement in the sequence of the associated decision tree.
   3.4 Select A_i^* as the sequence of arrangement decisions that results in the most valuable cutting pattern for S_i.
4. Fill S_i (with A_i^*) that results in the most valuable cutting pattern. Update the demand.
5. Return to step 1 while there are pieces in excess in the current pattern.

6 Experiments and computational results

In this section, we present the experiments and computational results in order to illustrate the performance of the method described in Sects 2 to 5, here denoted as DP_AOG. The implementation was coded in Pascal (Delphi 7–Borland) and executed in a microcomputer

Pentium IV (2.99 GHz–2.0 GBytes RAM). After some preliminary experiments, parameters for the subgradient optimization algorithm (Sect. 3) were chosen as $\pi = 2$ (this value was halved at each 3 iterations in step 5) and $\varepsilon = 0.05$ (which implies in a limit of 18 iterations, according to steps 5 and 6). In addition, the strategy of the multiplier adjustment was activated. The AOG approach parameters (Sect. 4) were set as: search depth limit $DB = 6$, $\lambda_1 = 0$ (heuristic H1 turned off) and $\lambda_2 = 0.9$ (heuristic H2 turned on). The symmetry rule was activated (heuristic H3 turned on) and sets X and Y were fully generated for each example (heuristic H4 turned off).

6.1 Benchmark instances

The performance of DP_AOG was initially evaluated by comparing its results to competitive methods reported in the literature. Three sets of benchmark constrained guillotine cutting instances with known optimal solutions were used in this analysis. The first set of instances (Table 1) comprises 8 widely applied weighted/unweighted examples with up to $n = 20$ piece types: *ChW1*, *ChW2* and *ChW3* (Christofides and Whitlock 1977), *WANG1*, *WANG2* and *WANG3* (Wang 1983), and *OF1* and *OF2* (Oliveira and Ferreira 1990). The other two sets correspond to 22 well-known larger instances of 25 to 50 piece types (Table 2),

Table 1 Results—1st set of benchmark instances

Instance	n	Algorithm								
		CW77	W83	OF90	VB93	CH95	H97	APT02	H04	DP_AOG
ChW1	7	**244**** *2.5*	–	–	**244**** *1.6*	**244**** *120*	**244**** *<1*	–	–	**244**** *<0.1 0.1*
ChW2	10	**2892**** *24.1*	–	–	**2892**** *12*	**2892**** *4236*	**2892**** *7.1*	**2892*** –	**2892*** –	**2892**** *0.8 38.4*
ChW3	20	**1860**** *66.1*	–	–	**1860**** *69.8*	**1860**** *3912*	**1860**** *29.8*	**1860*** –	**1860*** –	**1860**** *0.1 19.9*
WANG1	20	–	**2277*** *23*	–	–	–	–	–	–	**2277**** *<0.1 0.1*
WANG2	20	–	**2694*** *34*	–	–	–	–	–	–	**2694**** *<0.1 0.2*
WANG3	20	–	**2721*** *73*	–	**2721**** *180*	–	–	–	–	**2721**** *<0.1 0.6*
OF1	10	–	–	**2737*** –	–	–	–	–	**2737*** –	**2737**** *0.1 11.2*
OF2	10	–	–	**2690*** –	–	–	–	–	**2690*** –	**2690*** *0.1 23.6*

Columns 3–10: best solution value and total runtime

Column 11: best solution value, time to the best solution, and total runtime

*Optimal solution without optimality certificate

**Optimal solution with optimality certificate

– Not available

Table 2 Results—2nd and 3rd set of benchmark instances

Instance	n	Algorithm				Instance	Algorithm			
		FHZ98	APT02	H04	DP_AOG		FHZ98	APT02	H04	DP_AOG
CW1	25	**6402***	**6402***	**6402***	**6402****	*CU1*	12312	**12330***	**12330***	**12330****
		5.3	–	–	*1.1 25.6*		*7.1*	–	–	*0.1 0.5*
CW2	35	**5354***	**5354***	**5354***	**5354***	*CU2*	25806	**26100***	**26100***	**26100****
		6.8	–	–	*0.7 1118.8*		*19.8*	–	–	*0.2 1.4*
CW3	45	5148	**5689***	**5689***	**5689****	*CU3*	16608	16679	**16723***	**16723****
		17.3	–	–	*0.5 26.9*		*30.5*	–	–	*0.3 5.8*
CW4	45	6168	6170	**6175***	**6175****	*CU4*	98190	99366	**99495***	**99945****
		30.8	–	–	*1.3 46.8*		*55.8*	–	–	*3.8 35.0*
CW5	50	11550	11644	**11659***	**11659****	*CU5*	171651	**173364***	**173364***	**173364****
		19.8	–	–	*0.8 1268.5*		*89.6*	–	–	*2.7 873.2*
CW6	45	12403	**12923***	**12923***	**12923****	*CU6*	**158572***	**158572***	**158572***	**158572****
		79.0	–	–	*33.3 890.6*		*29.8*	–	–	*1.3 11.8*
CW7	45	9484	**9898***	**9898***	**9898****	*CU7*	246860	**247150***	**247150***	**247150***
		61.4	–	–	*4.1 10.7*		*26.7*	–	–	*0.1 1800.0*
CW8	35	4504	**4605***	**4605***	**4605****	*CU8*	432198	432714	**433331***	**433331****
		107.4	–	–	*20.8 340.5*		*68.8*	–	–	*0.5 26.8*
CW9	25	**10748***	**10748***	**10748***	**10748****	*CU9*	**657055***	**657055***	**657055***	**657055****
		125.3	–	–	*7.0 121.3*		*47.2*	–	–	*0.1 19.7*
CW10	40	6116	**6515***	**6515***	**6515****	*CU10*	764696	773485	**773772***	**773772***
		281.1	–	–	*2.8 182.2*		*123.9*	–	–	*69.2 1800.0*
CW11	50	6084	**6321***	**6321***	**6321****	*CU11*	913387	922161	**924696***	**924696***
		197.5	–	–	*2.0 1375.0*		*219.6*	–	–	*333.6 1293.1*
Average time					*6.8 491.5*	Average time				*37.4 533.4*

Columns 3–5 and 8–10: best solution value and total runtime

Columns 6 and 11: best solution value, time to the best solution, and total runtime

*Optimal solution without optimality certificate

**Optimal solution with optimality certificate

– Not available

11 weighted examples *CW1–CW11* and 11 unweighted examples *CU1–CU11* (Hifi 1997a; Fayard et al. 1998; Alvarez-Valdés et al. 2002; Hifi 2004).

The solution values obtained by competitive algorithms were included (when provided by the source), as well as the required computational times (if available). The algorithms are referred to as follows:

CW77 (Christofides and Whitlock 1977) Tree search algorithm with dynamic programming and a transportation routine—computer CDC 7600

W83 (Wang 1983) Algorithm 1 with $\beta = 0.01$, 0.02 and 0.03, respectively—computer Univac 1100

OF90 (Oliveira and Ferreira 1990) Variation of Wang (1983)'s algorithm 1—microcomputer 80286/7

VB93 (Viswanathan and Bagchi 1993) Best-first search algorithm—computer VAX 11

CH95 (Christofides and Hadjiconstantinou 1995) Tree search algorithm with state space relaxation of a dynamic programming formulation—microcomputer IBM 486

H97 (Hifi 1997a) Variation of Viswanathan and Bagchi's algorithm—computer Data General AV 8000

FHZ98 (Fayard et al. 1998) Algorithm based on solving a series of knapsack problems using dynamic programming—computer Sparc 20

APT02 (Alvarez-Valdés et al. 2002) Tabu search algorithm—microcomputer Pentium II

H04 (Hifi 2004) Algorithm based on dynamic programming and hill climbing techniques—computer Ultra Sparc 10

It should be emphasized that among these algorithms, CW77, VB93, CH95 and H97 are implementations of exact methods. It is worth noting that, because of the different computational environments, it is difficult to fairly compare the runtimes of the various algorithms.

Table 1 presents the solution values provided by each algorithm for the first set of instances. For algorithm DP_AOG, the following two figures in each cell correspond to the time for the best found solution and the total runtime (in seconds), respectively. The results indicate that DP_AOG successfully produced optimal solutions to all examples. The required time to find this solution is usually much less than one second (see last column of Table 1)—the remaining execution time is spent trying to prove its optimality, which actually occurred for 7 out of the 8 examples. The algorithms of the literature also provided optimal solutions for all instances to which they were applied; however, the number tackled by each procedure is too limited for a comparative analysis. Figure 4 presents the optimal patterns obtained by DP_AOG for some examples from this set.

Results for the 2nd and 3rd set of instances are presented in Table 2 (some optimal patterns are also shown in Fig. 5). Algorithms FHZ98, APT02 and H04 optimally solved 3, 9 and 11 instances from the 2nd set, respectively, and 2, 6 and 11 examples from the 3rd set. DP_AOG optimally solved all the 11 instances from the 2nd set, having provided a certificate of optimality for the solutions of all examples but *CW2*. Regarding the 3rd set, some examples with a larger number of pieces required a substantially larger computational time in each iteration of the subgradient. For this reason, the execution time of DP_AOG was limited to a maximum of 1800 seconds. In this case, the referred solution is the best feasible solution found during this time, which occurred with examples *CU7* and *CU10*. DP_AOG optimally solved all the 11 instances from the 3rd set and provided optimality certificates for 9 of them (exceptions are obviously *CU7* and *CU10*).

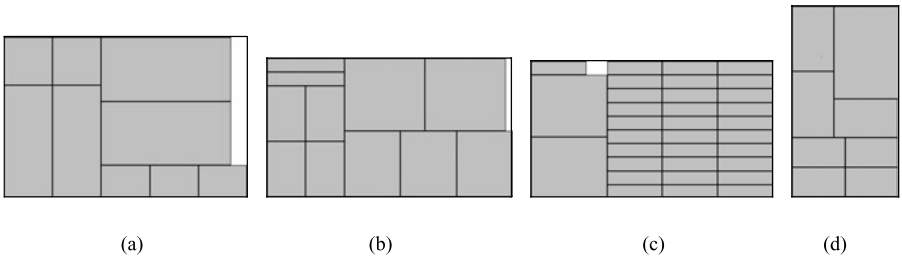

Fig. 4 Optimal patterns for benchmark instances of Table 1—1st set: (**a**) *ChW1*; (**b**) *OF1*; (**c**) *OF2*; (**d**) *WANG2*

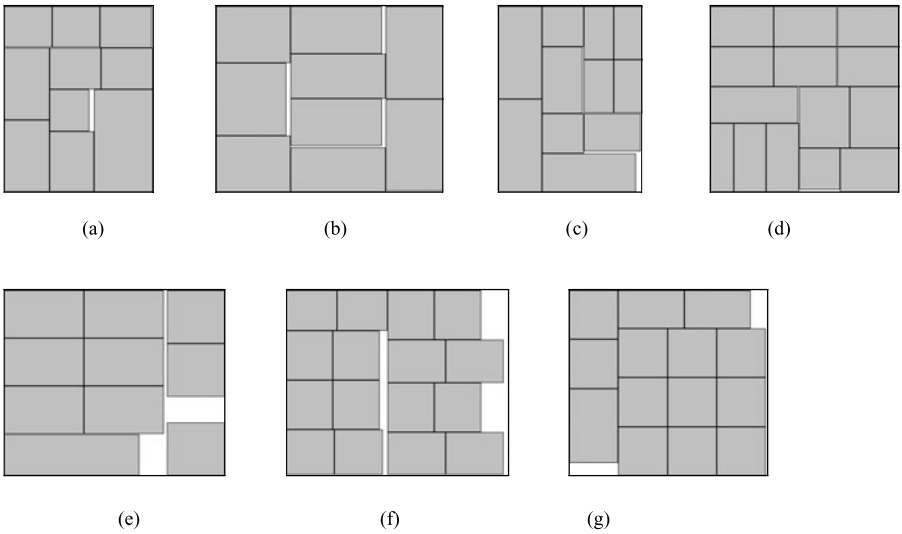

Fig. 5 Optimal patterns for benchmark instances of Table 2—(**a**) *CU4*; (**b**) *CU7*; (**c**) *CU8*; (**d**) *CU11*; (**e**) *CW4*; (**f**) *CW6*; (**g**) *CW8*

Similarly to what was observed from the experiments comprising the 1st set (Table 1), note in Table 2 that the time to find an optimal solution is relatively low (an average of 6.8 and 37.4 seconds for the 2nd and 3rd sets, respectively) if compared to the remaining execution time spent trying to prove its optimality (an average of 491.5 and 533.4 seconds, respectively). The weighted examples *CU1–CU11* are on average more difficult to solve than the unweighted examples *CW1–CW11*. However, for the four examples with non-proven optimal solutions (*CW2*, *CU7*, *CU10* and *CU11*), the gap between the upper bound (Z_{min}) and the optimal value is reasonably small (3.8%, 0.8%, 0.4% and 0.1%, respectively).

6.2 Randomly generated instances

In order to better assess how instances' characteristics affect the DP_AOG's computational performance and its ability to produce optimality certificates, the algorithm was applied to three classes of randomly generated examples. In class 1, each piece demand b_i $(i = 1, \ldots, n)$ was randomly generated within the interval $[1, \lfloor L/l_i \rfloor \lfloor W/w_i \rfloor]$, characterizing moderately constrained instances. In class 2, b_i was sorted from the interval $[1, 4]$ (very constrained instances) while in class 3, b_i was fixed to 1 (extremely constrained instances).

For all classes, we considered the number of item types n as 10, 20, 30, 40 and 50. The item lengths l_i and widths w_i were sorted from two sets of intervals: $\{[0.10L, 0.50L], [0.10W, 0.50W]\}$ (i.e., relatively small items), and $\{[0.25L, 0.75L], [0.25W, 0.75W]\}$ (i.e., relatively large items), where $L = W = 100$. Resulting values of l_i, w_i and b_i were then simply rounded. Note that these intervals have been used by different authors in the literature (see e.g. Beasley 1985; Morabito and Arenales 1996; Hifi 1997a). Since the unweighted problem examples of Table 2 were more difficult to solve (by DP_AOG) than the weighted ones, the examples have v_i equal to the piece area $l_i w_i$.

In the following, R_n_S denotes a set of 15 randomly generated examples with n item types of relatively small sizes (similarly, R_n_L stands for instances with relatively large item sizes). A total of 10 sets with 15 examples each were generated and solved for each of the three classes, resulting in 450 examples—these examples are available at ESICUP

(2008). Note that since the DP_AOG approach is exact for the unconstrained problem, class 1 examples are expected to be easier to solve than the class 2 and 3 examples. Moreover, for each class, examples from set S should be more difficult to solve than the examples from the set L, as their associated normal sets X and Y have on average a larger number of elements.

Table 3 presents the average results for sets R_n_S and R_n_L in each class. In addition to upper bounds (Z_{min}) and the best solution values obtained by the approach (DP_AOG),

Table 3 Results—450 randomly generated instances

Set	Class 1				Class 2				Class 3			
	Z_{min}	DP_AOG	GAP (%)	OC (%)	Z_{min}	DP_AOG	GAP (%)	OC (%)	Z_{min}	DP_AOG	GAP (%)	OC (%)
R_10_S	9838	9834 *8.3 46.3*	0.04	93.3	9829	9640 *19.1 942.3*	1.92	6.7	9263	8115 *0.7 752.2*	12.39	0.0
R_20_S	9978	9976 *7.8 79.6*	0.02	86.7	9961	9909 *46.2 929.9*	0.52	6.7	9923	9575 *89.5 532.6*	3.51	0.0
R_30_S	10000	10000 *3.1 7.3*	0.0	100	9998	9958 *85.4 1206.4*	0.40	6.7	9880	9671 *194.7 638.6*	2.12	0.0
R_40_S	10000	10000 *1.4 5.8*	0.0	100	10000	9996 *48.3 350.6*	0.04	66.7	9890	9807 *425.7 670.6*	0.84	0.0
R_50_S	10000	10000 *0.3 4.2*	0.0	100	10000	10000 *26.1 31.6*	0.00	100	1000	9770 *1.73 1442.4*	2.30	0.0
Average	9963	9962 *4.2 28.7*	0.01	96.0	9958	9901 *45.0 692.2*	0.58	37.4	9791	9388 *142.5 807.3*	4.23	0.0
R_10_L	9129	9129 *<0.1 0.8*	0.0	100	9253	9230 *<0.1 8.3*	0.25	86.7	8883	8746 *<0.1 11.9*	1.54	46.7
R_20_L	9598	9595 *1.1 6.7*	0.03	93.3	9660	9635 *<0.1 19.8*	0.26	60	9469	9322 *<0.1 42.4*	1.55	33.3
R_30_L	9838	9835 *<0.1 7.2*	0.03	93.3	9842	9842 *<0.1 7.6*	0.00	93.3	9745	9685 *<0.1 47.0*	0.62	40.0
R_40_L	9880	9861 *<0.1 12.8*	0.19	80.0	9870	9867 *<0.1 12.6*	0.03	86.7	9832	9767 *<0.1 46.2*	0.66	26.7
R_50_L	9898	9892 *<0.1 8.5*	0.06	93.3	9904	9902 *<0.1 9.9*	0.02	93.3	9881	9823 *<0.1 59.2*	0.59	26.7
Average	9669	9662 *0.3 7.2*	0.06	92.0	9706	9695 *<0.1 11.6*	0.11	84.0	9562	9469 *<0.1 41.3*	0.99	34.7

Z_{min}: mean upper bound of each set provided by the problem relaxation

DP_AOG: mean best solution value, mean time to the best solution, and mean total runtime of each set provided by algorithm DP_AOG

GAP: percent deviation between Z_{min} and DP_AOG of each set

OC: percentage of optimality certificates of each set provided by algorithm DP_AOG

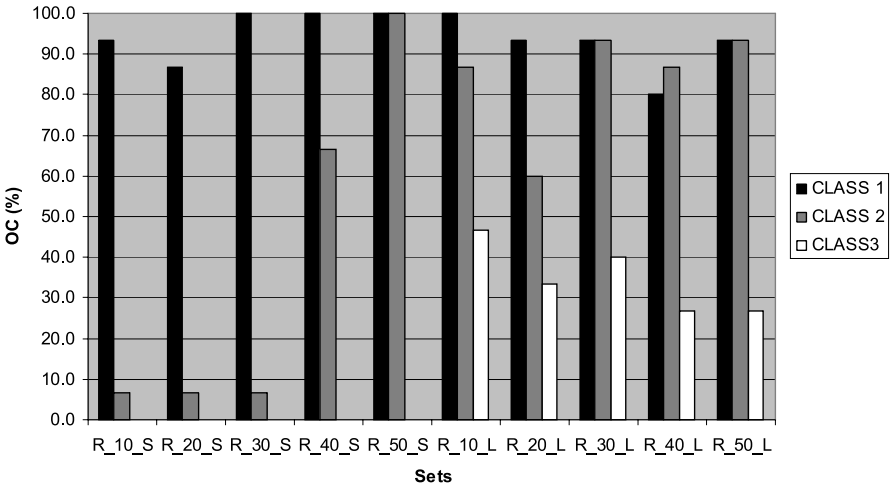

Fig. 6 Percentage of instances with optimality certificates provided by DP_AOG

the average percent deviation between these two values (GAP) as well as the percentage of optimality certificates (OC) obtained in each set are included. For class 1 sets, DP_AOG provided optimality certificates for a number of instances ranging from 80% to 100% of the total. It should be noted that when the piece size is small (sets S), the number of OC tends to improve with the number of piece types (Fig. 6). This behavior is not observed for larger sized instances (sets L). For class 2 examples (more constrained problems), the number of certificates is substantially fewer for examples from sets S, reaching as little as 6.7% of the total number of instances and increasing with the number of piece types. For sets L, the performance regarding OC achieves better levels. As expected, the examples of class 3 are the most difficult for DP_AOG given their heavily constrained nature. Moreover, on average the examples of sets S are more difficult than the ones in sets L.

It should be noted in Table 3 that the mean GAP is relatively small for most of the sets of the three classes. The overall figures for classes 1, 2 and 3 are reasonably small: 0.03%, 0.34% and 2.61%, respectively (as expected, GAP increases from class 1 to class 3). Similarly to what was observed in the previous experiments (Sect. 6.1), the computational time for the best found solution is relatively low if compared to the remaining execution time spent either trying to improve the solution or to prove that it is optimal. Moreover, the runtimes of class 1 sets are also relatively low when compared to the class 2 and 3 sets, and particularly for sets S. As one could expect, total computational times are reasonably well related to the number of proven optimal solutions (see Fig. 7). Instances from classes/sets with few certificates require on average a number of iterations close to the maximum number of the subgradient optimization procedure.

It is worth mentioning that, for most of the problem instances analyzed in this section, the best lower bounds Z_{LB} (feasible solutions) were found by the and/or-graph approach, instead of the inner heuristic. Preliminary experiments without applying the and/or-graph search in step 1 of the algorithm generally provided worse results. Therefore, the use of the and/or-graph approach is relevant for the present performance of the method.

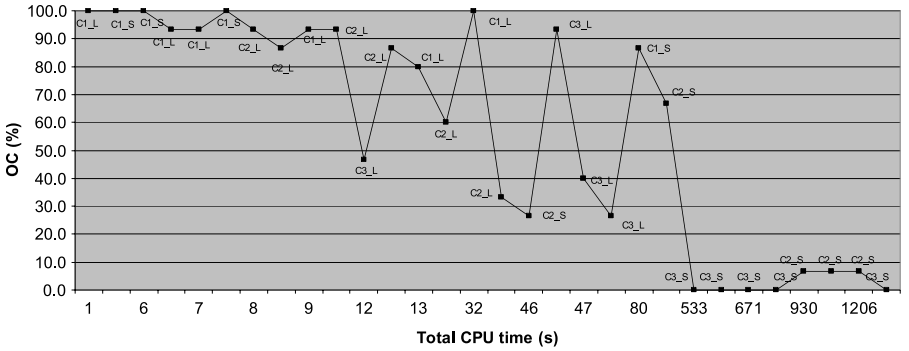

Fig. 7 Percentage of instances with optimality certificate vs. total CPU time (in seconds)

7 Concluding remarks

In this paper we investigate a heuristic approach for generating weighted/unweighted constrained two-dimensional guillotine cutting patterns. This problem appears in different industrial processes of cutting rectangular plates to produce ordered items, such as in the cutting of flat sheets in glass cutting processes, the cutting of wood boards in furniture factories, the cutting of fiberglass plates in the circuit board manufacturing, among others. The approach combines a state space relaxation of a dynamic programming formulation, an and/or-graph search approach, a subgradient optimization procedure and an inner feasibility heuristic. One of the advantages of the method is that by design it provides a certificate of optimality to the solution found or an upper bound to the optimal value.

Experiments with benchmark and randomly generated instances indicate that DP_AOG is competitive compared to other methods proposed in the literature. When applied to benchmark instances, it was able to find an optimal solution for most problems followed by a certificate of optimality (e.g., for more than 90% of the moderately constrained instances). In the cases for which DP_AOG was unable to ensure the optimality of the solution, it has produced a relatively tight upper bound to the optimal value (less than 1% on average) both for moderate and very constrained instances. On the other hand, the method's performance deteriorates for extremely constrained problems (i.e. problems with $b_i = 1$ for all i). Although upper bounds can still be considered within acceptable limits in most cases, a large reduction of optimal proven solutions were observed.

Regarding computational times, DP_AOG usually requires a short number of iterations to provide the best solution. If the solution cannot be proved as optimal, the total execution time depends on what it is spent in each iteration of the subgradient procedure. These times are usually short for instances with larger sized items but substantially increase when the pieces are small, especially for very and extremely constrained instances. The cardinality of the normal sets X and Y represents an aspect that strongly limits not only the performance, but also the applicability of DP_AOG, particularly when the number of elements is larger than 100. These difficulties may be overcome by altering the parameters of the and/or-graph search (e.g., adopting DB less than 6, λ_1 greater than 0, λ_2 greater than 0.9 and M less than 100) and by decreasing the number of iterations in the subgradient procedure (using ε greater than 0.05). However, it should be noted that this practice would imply in the degradation of the approach's performance in terms of solution and upper bound qualities and optimality certificates.

Based on the above discussion, we conclude that the method is particularly suitable to solve moderately constrained instances and very constrained instances with large sized items. Concerning the cases for which the method showed a limited performance (extremely constrained instances and very constrained instances with small sized items), further research would comprise the investigation of alternatives to improve the convergence of the subgradient optimization. The success in addressing these issues will have a direct impact on the quality of solutions and upper bounds, the number of optimality certificates that can be obtained and the running times.

Acknowledgements The authors would like to thank the three anonymous referees for their valuable comments and suggestions. This research was partially supported by FAPESP (grant 95/9522-0) and CNPq (grants 522973/95-7 and 307665/2003-8).

References

Alvarez-Valdés, R., Parajón, A., & Tamarit, J. (2002). A tabu search algorithm for large-scale guillotine (un)constrained two-dimensional cutting problems. *Computers and Operations Research*, *29*, 925–947.

Arenales, M., Morabito, R., & Yanasse, H. (Eds.) (1999). Cutting and packing problems. *Pesquisa Operacional*, *19*(2), 107–299.

Beasley, J. E. (1985). Algorithms for unconstrained two-dimensional guilhotine cutting. *Journal of the Operational Research Society*, *36*(4), 297–306.

Bischoff, E., & Waescher, G. (Eds.) (1995). Cutting and packing. *European Journal of Operational Research*, *84*, 3.

Burke, E., Kendall, G., & Whitwell, G. (2004). A new placement heuristic for the orthogonal stock-cutting problem. *Operations Research*, *52*(4), 655–671.

Christofides, N., & Hadjiconstantinou, E. (1995). An exact algorithm for orthogonal 2-d cutting problems using guillotine cuts. *European Journal of Operational Research*, *83*, 21–38.

Christofides, N., Mingozzi, A., & Toth, P. (1981). State-space relaxation procedure for the computation of bounds to routing problems. *Networks*, *11*, 145–164.

Christofides, N., & Whitlock, C. (1977). An algorithm for two-dimensional cutting problems. *Operations Research*, *25*(1), 30–44.

Cui, Y. (2008). Heuristic and exact algorithms for generating homogeneous constrained three-staged cutting patterns. *Computers and Operations Research*, *35*, 212–225.

Cung, V., Hifi, M., & Le Cun, B. (2000). Constrained two-dimensional guillotine cutting stock problems: A best-first branch-and-bound algorithm. *International Transactions in Operational Research*, *7*, 185–201.

Dowsland, K., & Dowsland, W. (1992). Packing problems. *European Journal of Operational Research*, *56*, 2–14.

Dowsland, K., Herbert, E., Kendall, G., & Burke, E. (2006). Using tree search bounds to enhance a genetic algorithm approach to two rectangle packing problems. *European Journal of Operational Research*, *168*, 390–402.

Dyckhoff, H., & Finke, U. (1992). *Cutting and packing in production and distribution: typology and bibliography*. Heidelberg: Springer.

Dyckhoff, H., Scheithauer, G., & Terno, J. (1997). Cutting and packing. In M. Amico, F. Maffioli, & S. Martello (Eds.), *Annoted bibliographies in combinatorial optimization* (pp. 393–414). New York: Wiley.

Dyckhoff, H., & Waescher, G. (Eds.) (1990). Cutting and packing. *European Journal of Operational Research*, *44*, 2.

ESICUP—Euro special interest group on cutting and packing. Available in: http://www.apdio.pt/esicup/ (accessed in 2008).

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

Fontes, D., Hadjiconstantinou, E., & Christofides, N. (2006). Lower bounds from state space relaxations for concave cost network flow problems. *Journal of Global Optimization*, *34*, 97–125.

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

Hadjiconstantinou, E., & Christofides, N. (1995). A new exact algorithm for the vehicle routing problem based on q-paths and k-shortest paths relaxations. *Annals of Operations Research*, *61*, 21–43.
Hifi, M. (1997a). An improvement of Viswanathan and Bagchi's exact algorithm for constrained two-dimensional cutting stock. *Computers and Operations Research*, *24*(8), 727–736.
Hifi, M. (1997b). The DH/KD algorithm: a hybrid approach for unconstrained cutting problems. *European Journal of Operational Research*, *97*, 41–52.
Hifi, M. (Ed.) (2002). Special issue: Cutting and packing problems. *Studia Informatica Universalis*, *2*(1), 1–161.
Hifi, M. (2004). Dynamic programming and hill-climbing techniques for constrained two-dimensional cutting stock problems. *Journal of Combinatorial Optimization*, *8*, 65–84.
Lodi, A., Martello, S., & Monaci, M. (2002). Two-dimensional packing problems: a survey. *European Journal of Operational Research*, *141*, 241–252.
Lucena, A. (2004). *Non delayed relax-and-cut algorithms* (Working Paper). Universidade Federal do Rio de Janeiro, Brazil.
Martello, S. (Ed.) (1994a) Special issue: Knapsack, packing and cutting, Part I: One dimensional knapsack problems. *INFOR*, *32*, 3.
Martello, S. (Ed.) (1994b) Special issue: Knapsack, packing and cutting, Part II: Multidimensional knapsack and cutting stock problems. *INFOR*, *32*, 4.
Mukhacheva, E. A. (Ed.). (1997). *Decision making under conditions of uncertainty: cutting–packing problems*. The International Scientific Collection, Ufa, Russia.
Morabito, R., & Arenales, M. (1994). An and/or-graph approach to the container loading problem. *International Transactions in Operational Research*, *1*(1), 59–73.
Morabito, R., & Arenales, M. (1996). Staged and constrained two-dimensional guilhotine cutting problems: An and/or-graph approach. *European Journal of Operational Research*, *94*, 548–560.
Morabito, R., Arenales, M., & Arcaro, V. (1992). An and/or-graph approach for two-dimensional cutting problems. *European Journal of Operational Research*, *58*(2), 263–271.
Mornar, V., & Khoshnevis, B. (1997). A cutting stock procedure for printed circuit board production. *Computers and Industrial Engineering*, *32*(1), 57–66.
Oliveira, J. F., & Ferreira, J. S. (1990). An improved version of Wang's algorithm for two-dimensional cutting problems. *European Journal of Operational Research*, *44*, 256–266.
Oliveira, J. F., & Waescher, G. (Eds.) (2007). Special issue on cutting and packing. *European Journal of Operational Research, 183*.
Parada, V., Alvarenga, A. G., & Diego, J. (1995). Exact solutions for constrained two-dimensional cutting problems. *European Journal of Operational Research*, *84*, 633–644.
Parada, V., Sepulveda, M., Solar, M., & Gomes, A. (1998). Solution for the constrained guillotine cutting problem by simulated annealing. *Computers and Operations Research*, *25*, 37–47.
Parada, V., Palma, R., Sales, D., & Gomes, A. (2000). A comparative numerical analysis for the guillotine two-dimensional cutting problem. *Annals of Operations Research*, *96*, 245–254.
Sweeney, P., & Paternoster, E. (1992). Cutting and packing problems: a categorized, application-oriented research bibliography. *Journal of the Operational Research Society*, *43*, 691–706.
Vasko, F. J. (1989). A computational improvement to Wang's two-dimensional cutting stock algorithm. *Computers and Industrial Engineering*, *16*(1), 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.
Waescher, G., Haussner, H., & Schumann, H. (2007). An improved typology of cutting and packing problems. *European Journal of Operational Research*, *183*, 1109–1130.
Wang, P. Y. (1983). Two algorithms for constrained two-dimensional cutting stock problems. *Operations Research*, *31*, 573–586.
Wang, P. Y., & Waescher, G. (Eds.) (2002). Cutting and packing. *European Journal of Operational Research*, *141*(2), 239–469.