CADERNOS DO LOGIS

Volume 2017, Number 1

Improved Space-State Relaxation for Constrained Two-Dimensional Guillotine Cutting Problems

André Soares Velasco, Eduardo Uchoa

July, 2017

LOGIS
Núcleo de Logística Integrada e Sistemas

Universidade Federal Fluminense

Improved Space-State Relaxation for Constrained Two-Dimensional Guillotine Cutting Problems

André Soares Velasco[a], Eduardo Uchoa[b,*]

[a] *Instituto Federal Fluminense – IFF/Universidade Federal Fluminense – UFF*
Av. Souza Mota, 350. Parque Fundão – Campos dos Goytacazes – RJ. CEP: 28060-010.
[b] *Departamento de Engenharia de Produção – Universidade Federal Fluminense – UFF*
Rua Passo da Pátria 156, Bloco D. São Domingos – Niterói–RJ. CEP:24210-240.

Abstract

Christofides and Hadjiconstantinou (1995) introduced a dynamic programming state space relaxation for obtaining upper bounds for the Constrained Two-dimensional Guillotine Cutting Problem. The quality of those bounds depend on the chosen item weights, they are adjusted using a subgradient-like algorithm. This paper proposes Algorithm X, a new weight adjusting algorithm based on integer programming that provably obtains the optimal weights. In order to obtain even better upper bounds, that algorithm is generalized into Algorithm X2 for obtaining optimal two-dimensional item weights. We also present a full hybrid method, called Algorithm X2D, that computes those strong upper bounds but also provides feasible solutions obtained by: (1) exploring the suboptimal solutions hidden in the dynamic programming matrices; (2) performing a number of iterations of a GRASP based primal heuristic; and (3) executing X2H, an adaptation of Algorithm X2 to transform it into a primal heuristic. Extensive experiments with instances from the literature, for both variants with and without item rotation, show that X2D can consistently deliver high-quality solutions and sharp upper bounds. In many cases the provided solutions are certificated to be optimal.

Keywords: Cutting, Dynamic Programming, Integer Programming

1. Introduction

The Two-dimensional Guillotine Cutting Problem (TGCP) consists in determining the most valuable way of cutting a rectangular object with length L and width W, using only orthogonal guillotine cuts, in order to produce smaller rectangular pieces, that are copies of m distinct items with predefined dimensions and value. For $1 \leq i \leq m$, l_i denotes the length of i, w_i its width and v_i its value. Some authors also refer to that problem as the Guillotine Two-dimensional Knapsack Problem. The Constrained TGCP (CTGCP) is the generalization where each item i also has a

*Corresponding author
Email addresses: asvelasco@iff.edu.br (André Soares Velasco), uchoa@producao.uff.br (Eduardo Uchoa)

given demand D_i, the maximum number of copies of an item in the cutting pattern. Orthogonal Rotations of items in the cutting patterns may be permitted or not. This work addresses both variants: CTGCP with rotation and without rotation. We assume that there are no restrictions on the number of stages of a cutting pattern. Sometimes the value of an item is defined by its area, the so called unweighted case. We also consider the case known as weighted, where item values are arbitrary.

The CTGCP is a classical problem with many industrial applications. For examples, the objects to be cut may be glass or wood panels, metal sheets, marble or granite slabs, etc. While TGCP can be solved in pseudo-polynomial time by Dynamic Programming (DP), CTGCP is known to be strongly NP-hard (Hifi, 2004) and can be much harder in practice. The proposed exact methods for CTGCP include Christofides and Whitlock (1977); Christofides and Hadjiconstantinou (1995); Cung et al. (2000); Chen (2008); Dolatabadi et al. (2012); Furini et al. (2016). Recent heuristics for CTGCP can be found in Alvarez-Valdés et al. (2002); Hifi (2004); Morabito and Pureza (2010). A class of heuristics of particular interest is the primal-dual heuristic, where a dual method (able to find upper bounds on the optimal solution value) is adapted for also finding primal feasible solutions (that yield lower bounds on the optimal solution value). As shown in Morabito and Pureza (2010), primal-dual heuristics can find some solutions that are difficult to be found using pure primal heuristics, like metaheuristics. By their own nature, primal-dual heuristics provide solutions with a guarantee of quality. Sometimes the upper bound matches the lower bound, certificating that the best solution found is indeed optimal.

The best known upper bounds for the CTGCP that can be obtained in pseudo-polynomial time are usually those by the DP State Space Relaxation (DPSSR), introduced in Christofides and Hadjiconstantinou (1995). It is based in the following idea:

- The DP for the TGCP cannot be turned into an efficient exact algorithm for CTGCP, since that would require adding up to m dimensions to its recursion, leading to an exponential explosion in the number of states. Instead, they propose a DP recursion with a single additional dimension that can be viewed as a relaxation of the exact recursion: a non-negative integer weight q_i is associated to each item i and it is imposed that the sum of the weights of the items in a solution should not exceed $Q = \sum_{i=1}^{m}(D_i q_i)$.

The upper bound actually provided by DPSSR depends on the chosen weights. Christofides and Hadjiconstantinou (1995) proposed an iterative procedure where all weights start with value zero and are adjusted by a subgradient-like formula. Morabito and Pureza (2010) used DPSSR as the basis of their primal-dual heuristic DP_AOG and proposed an improved formula for weight adjusting. *The main contribution of this paper is Algorithm X, an alternative algorithm for weight adjusting in DPSSR. Algorithm X is based on an integer programming model and is proved to be optimal, in the sense that it finds the weights that yield the best possible upper bound*

obtainable by DPSSR. Other important contributions are:

- A generalized variant of the DPSSR that uses two-dimensional item weights for obtaining even stronger upper bounds. Algorithm X2, also based on integer programming, for the optimal adjustment of those weights is proposed.
- A full primal-dual heuristic, called X2D. It executes Algorithms X and X2, but also uses a number of additional methods for obtaining good feasible solutions:
  - The suboptimal solutions hidden in the dynamic programming matrices are explored. While the optimal DPSSR solution can only be feasible if it is also the optimal CTGCP solution, suboptimal DPSSR solutions can be good feasible CTGCP solutions. Moreover, "near-feasible" solutions obtained from those matrices can often be corrected into good feasible solutions by performing local substitutions.
  - On instances where the gaps between upper and lower bounds are still large ($> 0.3\%$), a number of iterations of a GRASP based pure primal heuristic (Velasco and Uchoa, 2014) are performed.
  - Finally, Algorithm X2H, an adaptation of Algorithm X2 to transform it into a primal heuristic, may be executed.

We report extensive computational experiments on 1,000 instances. On instances without rotation, X2D is compared with the best heuristic (Morabito and Pureza, 2010) and the best exact algorithm (Dolatabadi et al., 2012) available in the literature. We also report results for the CTGCP with rotation. In that case, there are no recent algorithms in the literature for comparisons. Anyway, for both with or without rotation variants, we show that X2D can consistently deliver high-quality solutions and sharp upper bounds in reasonable times. The provided solutions are often certificated to be optimal.

The article is organized as follows. Section 2 describes the existing DPSSR. Section 3 and 4 presents Algorithms X and X2, respectively. Section 5 describes the primal components used in primal-dual heuristic X2D. Section 6 presents computational results. The last section presents final remarks. For simplicity, all the proposed algorithms will assume the variant without rotation. However, in Section 6, we indicate how they can be easily adapted for the variant with rotation.

2. Dynamic Programming State Space Relaxation for the CTGCP

An optimal solution for the Unconstrained TGCP can be assembled from the optimal solutions of the two subproblems defined by each possible horizontal or vertical guillotine cut. This fact allows its solution in pseudo-polynomial time by Dynamic Programming (DP), the complexity depends on the values of L and W. The original recursion proposed by Gilmore and Gomory (1965) limits the maximum number of cutting stages. Of course, it can be used to obtain the

optimal solution without restriction on the number of stages by setting a sufficiently large stage limit. Beasley (1985) gives a simpler recursion for the case without any stage limit. That recursion, together with the concept of Discretization Points from (Herz, 1972), was used by Cintra et al. (2008) on developing an exact DP algorithm for TGCP that is very effective when the values of L and W are not too large. Instances with values of L and W around 100 are solved in milliseconds; instances with L and W around 1000 can be solved in a few seconds in a modern computer.

On the other hand, solving the CTGCP by DP is a much more demanding task. This is related to the need of controlling how many copies of each item appear in the solutions of each subproblem. Let $\mathbf{D} = [D_1, \ldots, D_m]$ and $\mathbf{C} = [C_1, \ldots, C_m]$ be integer vectors indicating the original demand and the maximum number of copies of each item allowed in the solution of a subproblem, respectively. Define

$$v(l, w, \mathbf{C}) = \max(\{v_i | 1 \leq i \leq m : l_i \leq l, w_i \leq w, C_i \geq 1\} \cup \{0\}) \tag{1}$$

as the maximum value that can be obtained by cutting, without rotation, a single copy of an item i with positive C_i from a rectangle with dimensions (l, w). Considering that an optimal solution for a subproblem either has a single piece or is obtained after applying a vertical guillotine cut at position l' or an horizontal guillotine cut at position w', the value of the best solution for a rectangle (l, w) respecting the limits indicated by $\mathbf{C}$, is obtained by:

$$V(l, w, \mathbf{C}) = \max \begin{cases} \{v(l, w, \mathbf{C})\} \cup \\ \{V(l', w, \mathbf{C}') + V(l - l', w, \mathbf{C} - \mathbf{C}') \mid l' \in P_1, l' \leq l/2, \mathbf{0} \leq \mathbf{C}' \leq \mathbf{C}\} \cup \\ \{V(l, w', \mathbf{C}') + V(l, w - w', \mathbf{C} - \mathbf{C}') \mid w' \in P_2, w' \leq w/2, \mathbf{0} \leq \mathbf{C}' \leq \mathbf{C}\} \end{cases} \tag{2}$$

The value of the optimal CTGCP solution is given by $V(L, W, \mathbf{D})$. The sets P_1 and P_2 are the discretization points for the vertical and horizontal cuts, respectively. Assuming the no rotation case, P_1 is calculated from the property that states that, without losing optimality, it can be assumed that vertical guillotine cuts may only be performed in points that correspond to an integer conic combination of item lengths; an analogous property defines P_2 (Herz, 1972). More precisely:

$$P_1 = \{x \mid x = \sum_{i=1}^{m} \alpha_i l_i, \alpha \in \mathbb{Z}_+^m; x \leq L/2\} \tag{3}$$

$$P_2 = \{x \mid x = \sum_{i=1}^{m} \alpha_i w_i, \alpha \in \mathbb{Z}_+^m; x \leq W/2\} \tag{4}$$

The resulting DP can be used only for very small instances. Since it is necessary to consider

all possible ways of splitting a vector $\mathbf{C}$ into integer vectors $\mathbf{C}'$ and $\mathbf{C}-\mathbf{C}'$, the number of states grows by a factor of $\prod_{i=1}^{m}(D_i+1)$ with respect to the number of states of a DP for the corresponding TGCP. It is possible to do better: instead of adding all the m dimensions to the DP, add then incrementally. In other words, start with the unconstrained DP, add only the dimensions corresponding to the items in excess in that solution, solve the new DP, and so on. However, the resulting algorithm would still be unpractical on most instances, since even 5 or 6 added dimensions would already result in a huge number of DP states.

In order to avoid that exponential explosion in the number of states, Christofides and Hadjiconstantinou (1995) proposed a DP State Space Relaxation (DPSSR) for producing upper bounds for CTGCP. The idea is associating a non-negative integer weight q_i for each item i and impose that the sum of the weights of the items in a solution should not exceed the scalar value Q defined as $Q=\sum_{i=1}^{m}(D_i q_i)$. The recursion for determining the value $V(l,w,d)$ of the best solution for a rectangle (l,w) with total weight not larger than the integer scalar d is defined by the following recursion:

$$v(l,w,d)=\max(\{v_i|1\leq i\leq m: l_i\leq l, w_i\leq w, q_i\leq d\}\cup\{0\}) \tag{5}$$

$$V(l,w,d)=\max\begin{cases}\{v(l,w,d)\}\cup\\ \{V(l',w,d')+V(l-l',w,d-d')\mid l'\in P_1, l'\leq l/2, 0\leq d'\leq d\}\cup\\ \{V(l,w',d')+V(l,w-w',d-d')\mid w'\in P_2, w'\leq w/2, 0\leq d'\leq d\}\end{cases} \tag{6}$$

The upper bound for CTGCP is given by $Z=V(L,W,Q)$. The DP for that relaxation is likely to be better solved because the number of states is only multiplied by the factor $(Q+1)$ with respect to the number of states in the corresponding TGCP. In fact, attributing zero weights for most of the items, it is possible to keep the value Q relatively low.

The value of the upper bound obtained by that state space relaxation depends a lot on the chosen weight vector $\mathbf{q}=[q_1,\ldots,q_m]$. In order to obtain good vectors $\mathbf{q}$, those that yield high quality upper bounds, Christofides and Hadjiconstantinou (1995) proposed the following weight adjustment algorithm, inspired by the subgradient method. Let $\mathbf{b}=[b_1,\ldots,b_m]$ be the vector corresponding to the number of copies of each item in the optimal solution of the DP for a certain $\mathbf{q}$, with value Z. If $b_i\leq D_i$ for each item i, then it corresponds to a feasible and optimal CTGCP solution. Otherwise, $\mathbf{b}$ corresponds to an unfeasible CTGCP solution and Z is only a valid upper bound. The algorithm starts with a zero vector $\mathbf{q}$, therefore Q is also zero. The DP is executed and returns a solution corresponding to vector $\mathbf{b}$. The next iteration will use an adjusted $\mathbf{q}$ obtained by the following expression:

$$q_i=\begin{cases}q_i+\lfloor t\sqrt{b_i-D_i}\rfloor & \text{if } b_i>D_i \qquad (7a)\\ \max(0,q_i-\lfloor t\sqrt{D_i-b_i}\rfloor) & \text{if } b_i\leq D_i, \qquad (7b)\end{cases}$$

where t is a positive scalar representing the step size. The idea consists in increasing the weights of the items in excess and reducing the weights of the items with positive slack demand. The step size is defined as:

$$t = \frac{1}{2}\sqrt{\pi(Z_{UB} - Z_{LB})/\sum_{i=1}^{m}(d_i - b_i)^2}, \tag{8}$$

where π is initialized with 1.0 and halved at each 3 iterations; Z_{UB} and Z_{LB} are the current upper and lower bounds on the optimal solution value. The lower bound comes from external heuristics, it may be improved by the method itself only if the current solution $\mathbf{b}$ is feasible (in that case it should be also optimal); the upper bound is the smaller Z value found until the current iteration. The algorithm stops if $Z_{UB} = Z_{LB}$ or if π becomes smaller than a parameter ε or if $Q = \sum_{i=1}^{m}(D_i q_i)$ exceeds a parameter $MaxQ$. Morabito and Pureza (2010) proposed changing the calculation of the step size to

$$t = \max\left\{1, \sqrt{\pi(Z_{UB} - Z_{LB})/\sum_{i=1}^{m}(d_i - b_i)^2}\right\}, \tag{9}$$

where π is initialized with 2.0.

3. Algorithm X

We propose a new algorithm for obtaining upper bounds using the DPSSR of Christofides and Hadjiconstantinou (1995). The weight adjustment of the new algorithm not only considers the solution $\mathbf{b}$ of the current iteration, it considers the solutions obtained in all the n iterations already performed. Let $\mathbf{b}^j = [b_1^j, \ldots, b_m^j]$ be the vector corresponding to the solution of value Z^j obtained in iteration j, b_i^j is the number of times item i appears in that solution. It is assumed that all those n solutions are unfeasible, otherwise the optimal CTGCP solution would already had been found. Define the following Integer Program, named IPX(n):

$$\min Q = \sum_{i=1}^{m} D_i q_i \tag{10}$$

$$\text{s.t.} \quad \sum_{i=1}^{m}(b_i^j - D_i)q_i \geq 1, \qquad \forall j = 1, \ldots, n \tag{11}$$

$$\mathbf{q} \geq 0 \text{ and integer} \tag{12}$$

The new weights for the next iteration are given be the solution of IPX(n). Inequalities (11) are based in the fact that a new vector $\mathbf{q}$ can only improve Z_{UB} if it eliminates all the n previous solutions from the relaxed DP. In the algorithm described below, $MaxIter$ and $MaxQ$ are parameters limiting the maximum number of iterations and maximum value of Q allowed, the

output $CertOpt$ notifies whether a certificate of optimality was obtained.

Algorithm X($MaxIter, MaxQ$)

1: $n = 1, \mathbf{q} = \mathbf{0}, Z_{UB} = \infty$
2: Solve the relaxed DP with vector $\mathbf{q}$, obtaining a solution $\mathbf{b}^n$ with value Z^n;
3: **if** $Z^n < Z_{UB}$ **then** $Z_{UB} = Z^n$;
4: **if** ($\mathbf{b}^n$ is feasible) **then return** $(Z_{UB}, CertOpt = True)$;
5: Solve IPX(n);
6: **if** (IPX(n) is infeasible) **then return** $(Z_{UB}, CertOpt = False)$;
7: Update $\mathbf{q}$ and Q with the optimal solution of IPX(n);
8: $n = n + 1$
9: **if** ($n > MaxIter$ **or** $Q > MaxQ$) **then return** $(Z_{UB}, CertOpt = False)$;
10: **Goto** 2;

Lemma 1. *Even if $MaxIter = \infty$ and $MaxQ = \infty$, algorithm X terminates in a finite number of iterations.*

Proof. The definition of IPX(n) ensures that vector $\mathbf{b}^{n+1}$ will be distinct from vectors $\mathbf{b}^j, 1 \leq j \leq n$. As there is a finite number of possible vectors, the algorithm must stop. □

Theorem 1. *If $MaxIter = \infty$ and $MaxQ = \infty$, Algorithm X always returns $Z_{UB} = Z1^*$, the best upper that can be found by the state space relaxed DP with any vector $\mathbf{q}$. Moreover, bound $Z1^*$ is first found in an iteration that uses the least expensive DP (i.e., it uses the smallest value of Q) that can obtain that upper bound.*

Proof. By Lemma 1, Algorithm X stops. If X stops with a certificate of optimality, certainly $Z_{UB} = Z1^*$. Otherwise, X stopped in iteration n because IPX(n) is infeasible. Suppose that the returned Z_{UB} is not optimal, i.e., $Z_{UB} > Z1^*$. In that case, there exists some vector $\mathbf{q}^*$ that produces $Z1^*$. It is not possible that $\sum_{i=1}^{m}(b_i^j - D_i)q_i^* < 1$ for some $j, 1 \leq j \leq n$. Otherwise, solution $\mathbf{b}^j$ with value $Z^j \geq Z_{UB}$ would be a solution of the relaxed DP with vector $\mathbf{q}^*$. Therefore, $\mathbf{q}^*$ is feasible for IPX(n). Contradiction.

Let n' be the first iteration that produced a solution with value $Z^{n'} = Z1^*$. If $n' = 1$, then the value of Q was 0 in that iteration. If $n' > 1$, the vector $\mathbf{q}$ used in iteration n' was obtained from the optimal solution of IPX($n' - 1$). For every j, $1 \leq j \leq n' - 1, Z^j > Z1^*$. So, any vector $\mathbf{q}$ that produces a solution with value $Z1^*$ must satisfy $\sum_{i=1}^{m}(b_i^j - D_i)q_i \geq 1$, for all $j = 1, \ldots, n' - 1$. Therefore, $Q = \sum_{i=1}^{m} D_i q_i$ can not be smaller than the value of Q obtained from the solution of IPX($n' - 1$). □

In the majority of the tests with instances from the literature, Algorithm X with $MaxIter = \infty$ and $MaxQ = \infty$ terminates in less than 10 iterations and with values of Q up to 30. However, in a few instances the number of iterations and the values of Q can grow a lot, so the algorithm gets slow. Moreover, in those cases the final optimal bound $Z1^*$ obtained is not significantly better than the bounds obtained in the first iterations, when the values of Q were reasonable small.

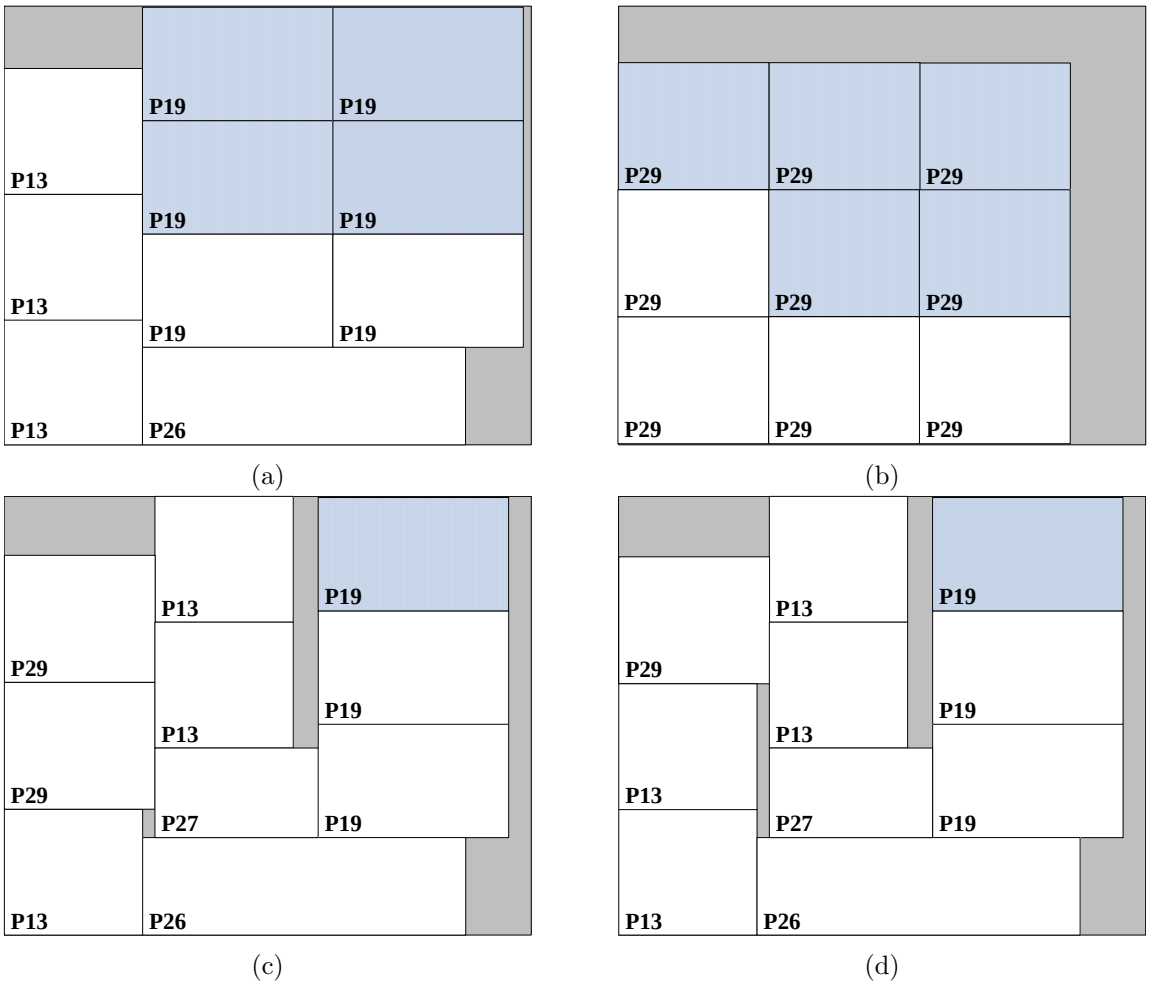

Figure 1: Infeasible solutions found by Algorithm X on CW4.

In order to avoid such waste of time, it is recommended to make use of parameters $MaxIter$ and $MaxQ$. Theorem 2 shows that Algorithm X is still optimal for a given $MaxQ$. Its proof is omitted for being similar to the proof of Theorem 1.

Theorem 2. *Suppose that Algorithm X is executed with $MaxIter = \infty$ and a finite value for $MaxQ$. The returned Z_{UB} will be always equal to $Z1^*(MaxQ)$, the best upper that can be found by the state space relaxed DP with any vector* $\mathbf{q}$ *such that $Q = \sum_{i=1}^{m} D_i q_i \leq MaxQ$.*

In order to illustrate Algorithm X, we present a detailed run over the classical instance CW4, introduced in Christofides and Whitlock (1977). In that weighted instance (item values are not proportional to their areas), $m = 38, L = 465$, and $W = 387$. Algorithm X starts by solving the DP with $\mathbf{q} = \mathbf{0}$. This is equivalent to solving the Unconstrained TGCP. The optimal solution obtained, with value $Z^1 = 6551$, is shown in Figure 1(a). The hatches indicate excess production of some item. The solution vector $\mathbf{b}^1$ has as non-zero components $b^1_{13} = 3(D_{13} = 6), b^1_{19} = 6(D_{19} = 2)$, and $b^1_{26} = 1(D_{26} = 3)$, IPX(1) is shown next. Remark that variables corresponding to items

that do not appear in any solution $\mathbf{b}^j, 1 \leq j \leq n$, can always be omitted from IPX(n) as they have value zero in any optimal solution.

$$\min Q = 6q_{13} + 2q_{19} + 3q_{26}$$
$$\text{s.t.} \quad -3q_{13} + 4q_{19} - 2q_{26} \geq 1$$
$$\mathbf{q} \geq 0 \text{ and integer}$$

The optimal solution of IPX(1) provides a weight vector where the only non-zero component is $q_{19} = 1$, so $Q = 2$. The DP in iteration 2 obtains the solution with value $Z^2 = 6183$ depicted in Figure 1(b). Item 28 is produced 9 times, but its demand is only 4. IPX(2) is shown next and has an optimal solution where the non-zero components are $q_{19} = 2, q_{29} = 1$, so $Q = 8$.

$$\min Q = 6q_{13} + 2q_{19} + 3q_{26} + 4q_{29}$$
$$\text{s.t.} \quad -3q_{13} + 4q_{19} - 2q_{26} - 4q_{29} \geq 1$$
$$-6q_{13} - 2q_{19} - 3q_{26} + 5q_{29} \geq 1$$
$$\mathbf{q} \geq 0 \text{ and integer}$$

Iteration 3 produces the solution shown in Figure 1(c), with $Z^3 = 6238$. As $D_{27} = 7$, IPX(3) shown next, has optimal solution with $q_{19} = 7, q_{29} = 3$ and $Q = 26$.

$$\min Q = 6q_{13} + 2q_{19} + 3q_{26} + 7q_{27} + 4q_{29}$$
$$\text{s.t.} \quad -3q_{13} + 4q_{19} - 2q_{26} - 7q_{27} - 4q_{29} \geq 1$$
$$-6q_{13} - 2q_{19} - 3q_{26} - 7q_{27} + 5q_{29} \geq 1$$
$$-3q_{13} + q_{19} - 2q_{26} - 6q_{27} - 2q_{29} \geq 1$$
$$\mathbf{q} \geq 0 \text{ and integer}$$

The solution obtained in iteration 4, with $Z^4 = 6233$ is depicted in Figure 1(d) and leads to IPX(4):

$$\min Q = 6q_{13} + 2q_{19} + 3q_{26} + 7q_{27} + 4q_{29}$$
$$\text{s.t.} \quad -3q_{13} + 4q_{19} - 2q_{26} - 7q_{27} - 4q_{29} \geq 1$$
$$-6q_{13} - 2q_{19} - 3q_{26} - 7q_{27} + 5q_{29} \geq 1$$
$$-3q_{13} + q_{19} - 2q_{26} - 6q_{27} - 2q_{29} \geq 1$$
$$-2q_{13} + q_{19} - 2q_{26} - 6q_{27} - 3q_{29} \geq 1$$
$$\mathbf{q} \geq 0 \text{ and integer}$$

As IPX(4) is infeasible, Algorithm X terminates with $Z_{UB} = Z^2 = Z1^* = 6183$ and $CertOpt = false$. The time solve those IPs is negligible with respect to the time taken by the DP.

4. Algorithm X2

In order to obtain even stronger upper bounds using DP, we introduce here a generalization of the state space relaxation proposed in Christofides and Hadjiconstantinou (1995). The idea is that each item i will be associated to an integer non-negative two-dimensional weight $q_i = (q_i^1, q_i^2)$. Define $\mathbf{q} = [q_1 = (q_1^1, q_1^2), \ldots, q_m = (q_m^1, q_m^2)]$, $\mathbf{q}^1 = [q_1^1, \ldots, q_m^1]$ and $\mathbf{q}^2 = [q_1^2, \ldots, q_m^2]$. The new relaxed DP imposes that the sum of the two-dimensional weights of the items in a solution should not exceed the two-dimensional value $Q = (Q^1, Q^2)$, where $Q^1 = \sum_{i=1}^{m}(D_i q_i^1)$ and $Q^2 = \sum_{i=1}^{m}(D_i q_i^2)$. The new recursion is given by:

$$v(l, w, (d^1, d^2)) = \max(\{v_i | 1 \leq i \leq m : l_i \leq l, w_i \leq w, (d^1, d^2) \geq (q_i^1, q_i^2)\} \cup \{0\}) \tag{13}$$

and

$$V(l, w, d) = \max \begin{cases} \{v(l, w, d)\} \cup \\ \{V(l', w, d') + V(l - l', w, d - d') \mid l' \in P_1, l' \leq l/2, 0 \leq d' \leq d\} \cup \\ \{V(l, w', d') + V(l, w - w', d - d') \mid w' \in P_2, w' \leq w/2, 0 \leq d' \leq d\}, \end{cases} \tag{14}$$

where $d = (d^1, d^2)$ are now two-dimensional weight limits. The upper bound for the CTGCP solution is given by $V(L, W, (Q^1, Q^2))$ The number of states in that DP is increased by a factor of $(Q^1 + 1).(Q^2 + 1)$ with respect to the number of states of the DP for the Unconstrained TGCP. The updating of two-dimensional weights can be done by a Quadratic Integer Program that will be called QIPX2(n):

$$\min \quad (Q^1 + 1)(Q^2 + 1) \tag{15}$$

$$\text{s.t.} \quad Q^1 = \sum_{i=1}^{m}(D_i q_i^1) \tag{16}$$

$$Q^2 = \sum_{i=1}^{m}(D_i q_i^2) \tag{17}$$

$$\sum_{i=1}^{m}(b_i^j q_i^1) - Q^1 + M(1 - y_j) \geq 1 \qquad \forall j = 1, \ldots, n \tag{18}$$

$$\sum_{i=1}^{m}(b_i^j q_i^2) - Q^2 + M y_j \geq 1 \qquad \forall j = 1, \ldots, n \tag{19}$$

$$y_1 = 1 \tag{20}$$

$$\mathbf{y} \text{ binary} \tag{21}$$

$$\mathbf{q}^1, \mathbf{q}^2 \geq 0 \text{ and integer} \tag{22}$$

There are $2(m + 1)$ non-negative integer variables and n binary variables in model QIPX2(n). Equalities (16) and (17) define variables Q^1 and Q^2 in terms of the vector of variables $\mathbf{q}^1$ and $\mathbf{q}^2$.

For each i, $1 \leq i \leq m$, individual variables q_i^1 and q_i^2 represent the two dimensions of the weight of item i. For each j, $1 \leq j \leq n$, binary variable y_j indicates whether solution $\mathbf{b}^j$ should be eliminated from the relaxed DP by the first dimension of the weights (if $y_j = 1$) or by the second dimension (if $y_j = 0$). Constraints (18) and (19) implement those requirements. Coefficient M should be big enough to make sure that a solution $\mathbf{b}^j$ never needs to be eliminated by both dimensions. Constraint (20) is not essential, it helps to reduce the symmetry of QIPX2(n) by forcing solution $\mathbf{b}^1$ to be eliminated by the first dimension. The objective function (15) aims at minimizing the number of states in the relaxed DP.

Define Algorithm X2 as being the direct generalization of algorithm X for two-dimensional weights, that are now updated by the solution of QIPX2(n).

Algorithm X2($MaxIter, MaxQ$)

1: $n = 1, \mathbf{q} = (\mathbf{0}, \mathbf{0}), Z_{UB} = \infty$
2: Solve the relaxed DP with bidimensional vector $\mathbf{q}$, obtaining a solution $\mathbf{b}^n$ with value Z^n;
3: **if** $Z^n < Z_{UB}$ **then** $Z_{UB} = Z^n$;
4: **if** ($\mathbf{b}^n$ is feasible) **then return** $(Z_{UB}, CertOpt = True)$;
5: Solve QIPX2(n);
6: **if** (QIPX2(n) is infeasible) **then return** $(Z_{UB}, CertOpt = False)$;
7: Update $\mathbf{q} = (\mathbf{q}^1, \mathbf{q}^2)$ and $Q = (Q^1, Q^2)$ with the optimal solution of QIPX2(n);
8: $n = n + 1$
9: **if** ($n > MaxIter$ **or** $Q^1 > MaxQ$ **or** $Q^2 > MaxQ$) **then return** $(Z_{UB}, CertOpt = False)$;
10: **Goto** 2;

The proof of the following theorem is similar to the proof of Theorem 1.

Theorem 3. *If $MaxIter = \infty$ and $MaxQ = \infty$, Algorithm X2 always returns $Z_{UB} = Z2^*$, the best upper that can be found by the state space relaxed DP with any bidimensional vector $\mathbf{q}$. Moreover, bound $Z2^*$ is found in an iteration that uses the least expensive DP (i.e., with the smallest value of $(Q^1 + 1)(Q^2 + 1)$) that can obtain that upper bound.*

Theorem 4. *$Z2^* \leq Z1^*$ and there are instances where the inequality is strict.*

Proof. In order to obtain upper bound $Z1^*$ using a DP with bidimensional weights, it is enough to use $\mathbf{q} = (\mathbf{q}^*, \mathbf{0})$, where $\mathbf{q}^*$ is the optimal unidimensional weight vector. In order to show that the inequality can be strict, we refer to the example over instance CW4 shown in end of this section. □

There is software for solving integer programs with quadratic objective function. However, by simplicity and also for avoiding the risk of that solution taking too much time, in our experiments we decided to simplify the objective function to:

$$\min \quad Q^1 + Q^2 \tag{23}$$

We refer to (23) subject to constraints (16)-(22) as IPX2(n). If $MaxIter = \infty$ and $MaxQ = \infty$, Algorithm X2 using IPX2(n) (instead of QIPX2(n)) still finds the best possible upper bound $Z2^*$. However, that bound possibly will be obtained with a vector of bidimensional weights that do not minimize the number of states in the DP.

Bidimensional weights may obtain stronger upper bounds but also may lead to DPs that are more expensive to be solved than those from unidimensional weights. For that reason, we opted in our experiments to first apply Algorithm X. When it stops without a certificate of optimality, Algorithm X2 is called. However, X2 is hot started with the unfeasible solutions $\mathbf{b}^j, 1 \leq j \leq n$, found by Algorithm X. This means X2 will start to be executed in Step 5, solving IPX2(n). This makes sense because X2 can only improve upon the upper bound from X using weights that eliminate from the space state relaxation all the infeasible solutions with value equal or larger than that bound. We illustrate this on instance CW4. Algorithm X2 is initialized with the 4 unfeasible solutions found by Algorithm X, so the first IP solved is IPX2(4):

$$\min Q^1 + Q^2$$

$$\begin{aligned}
\text{s.t.} \quad Q^1 &= 6q^1_{13} + 2q^1_{19} + 3q^1_{26} + 7q^1_{27} + 4q^1_{29} \\
Q^2 &= 6q^2_{13} + 2q^2_{19} + 3q^2_{26} + 7q^2_{27} + 4q^2_{29} \\
3q^1_{13} + 6q^1_{19} + q^1_{26} - Q^1 + M(1 - y_1) &\geq 1 \\
3q^2_{13} + 6q^2_{19} + q^2_{26} - Q^2 + My_1 &\geq 1 \\
9q^1_{29} - Q^1 + M(1 - y_2) &\geq 1 \\
9q^2_{29} - Q^2 + My_2 &\geq 1 \\
3q^1_{13} + 3q^1_{19} + q^1_{26} + q^1_{27} + 2q^1_{29} - Q^1 + M(1 - y_3) &\geq 1 \\
3q^2_{13} + 3q^2_{19} + q^2_{26} + q^2_{27} + 2q^2_{29} - Q^2 + My_3 &\geq 1 \\
4q^1_{13} + 3q^1_{19} + q^1_{26} + q^1_{27} + q^1_{29} - Q^1 + M(1 - y_4) &\geq 1 \\
4q^2_{13} + 3q^2_{19} + q^2_{26} + q^2_{27} + q^2_{29} - Q^2 + My_4 &\geq 1 \\
y_1 = 1, \mathbf{y} &\text{ binary} \\
\mathbf{q}^1, \mathbf{q}^2 \geq 0 &\text{ and integer}
\end{aligned}$$

The optimal solution of IPX2(4) yields $y_1 = 1, y_2 = 0, y_3 = 1$ and $y_4 = 1$, the non-zero weights are $q^1_{19} = 1$ and $q^2_{29} = 1$, yielding $Q^1 = 2$ and $Q^2 = 4$. Then, the DP with those bidimensional weights finds the feasible solution shown in Figure 2, with value $Z^5 = 6175$. That solution is certified to be optimal and Algorithm X2 terminates.

5. Primal Dual Heuristic X2D

Morabito and Pureza (2010) proposed DP_AOG, a primal dual heuristic that combines the dynamic programming with space-state relaxation with the And/Or Graph Heuristic

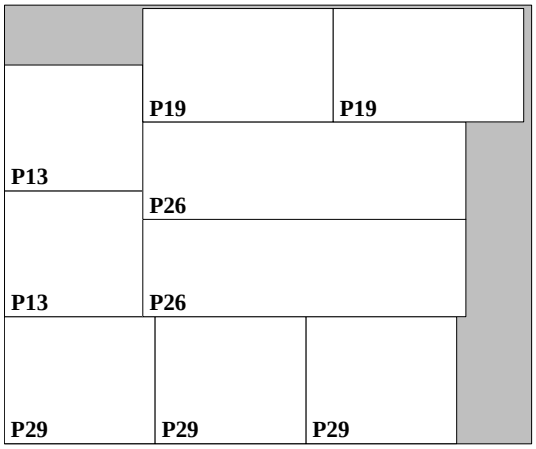

Figure 2: Optimal solution of instance CW4, value 6175.

(Morabito et al., 1992). In the same spirit, we propose X2D, a primal dual heuristic that combines the improved space-state relaxations presented in the previous section with primal elements, able to find feasible solutions.

5.1. Solutions from the Dynamic Programming Matrix

Algorithms X and X2 can only find feasible solutions that correspond to the optimal solution of a DP. In fact, they can only find feasible solutions in their last iteration and only for instances where $Z1^*$ and $Z2^*$, respectively, are equal to the value of the optimal CTGCP solution. It would be good to use the significant computational effort already spent in solving each DP for also trying to find feasible solutions in every iteration of those algorithms.

The first observation is that, according to equations (6) and (14), we calculate $V(L, W, Q)$ as the best of several solutions, corresponding to all ways of performing the first guillotine cut in the original object. It may happen that some suboptimal solutions are feasible, even though the optimal one is unfeasible. The computational effort for checking each of those solutions is negligible compared with the time spent by the DP. Moreover, checking the suboptimal solutions is worthy: in several instances, good lower bounds are obtained from Algorithms X and X2 in this way. For example, the optimal solution of CW4 depicted in Figure 2 can be found as a suboptimal solution of the dynamic programming in the third iteration of Algorithm X.

The second observation is that some solutions found by the DP, optimal or suboptimal, are near-feasible (very few items exceed their demands) and have value larger than Z_{LB}. In those cases, a feasibility heuristic is applied to "fix" those near-feasible solutions into feasible ones, replacing an item in excess with a smaller item with available demand. Those replacements usually decrease the value of a solution, but sometimes the new value is still larger than Z_{LB}. Algorithm DP_AOG also contains a feasibility heuristic.

5.2. Algorithm X2H

Suppose an unidimensional weight vector $\mathbf{q}$ with several positive components. In order to obtain a valid upper bound, it is necessary to calculate $V(L, W, Q)$, where $Q = \sum_{i=1}^{m}(D_i q_i)$. That value of Q comes from the fact that is possible that all optimal CTGCP solutions use exactly D_i copies from each item i with $q_i > 0$. However, in practice, it is very unlikely that this happens. This means that if Q_{heu} is a value a little smaller than Q, $V(L, W, Q_{heu})$ will not be a valid upper bound. However, there is often still a chance of that an optimal CTGCP solution (or at least an improving solution) will appear in the corresponding DP matrix. The proposed Algorithm X2H is an heuristic based on that observation, to be used if X2 terminates without a certificate of optimality.

In fact, X2H uses bidimensional weights and starts by considering all the n unfeasible solutions found along Algorithms X and X2, all of them have value larger or equal to the current Z_{UB}. The idea is fixing values for $Q_{heu} = (Q^1_{heu}, Q^2_{heu})$ and then choosing the weights $\mathbf{q}$ by solving the model IPX2H(n, Q_{heu}), that is obtained by replacing Constraints (18) and (19) in IPX2(n) by:

$$\sum_{i=1}^{m}(b_i^j q_i^1) - Q^1_{heu} + M(1 - y_j) \geq 1 \qquad \forall j = 1, \ldots, n \tag{24}$$

$$\sum_{i=1}^{m}(b_i^j q_i^2) - Q^2_{heu} + M y_j \geq 1 \qquad \forall j = 1, \ldots, n. \tag{25}$$

Model IPX2H(n, Q_{heu}) is always feasible, even if IPX2(n) is not. This happens because IPX2H(n) can always satisfy Constraints (24) and (25) by using arbitrarily large values for $(\mathbf{q}^1, \mathbf{q}^2)$. In fact, if $(\mathbf{q}^1, \mathbf{q}^2)$ is such that $Q^1 = \sum_{i=1}^{m}(D_i q_i^1) > Q^1_{heu}$ or $Q^2 = \sum_{i=1}^{m}(D_i q_i^2) > Q^2_{heu}$, the value $V(C, L, Q_{heu})$ does not provide a valid upper bound. However, the corresponding solution may be feasible and better than Z_{LB}, the current best lower bound. However, it may also be another unfeasible solution. Procedure DescentX2H, described next, is a search for a feasible solution using a fixed value of Q_{heu}. It always starts using the n unfeasible solutions available at the end of Algorithm X2, but each subsequent unfeasible solution found by the DP is incorporated into IPX2H (n) in order to be eliminated in the next iterations. It was called a "descent" because the successive values of Z^n are likely to decrease, the procedure stops whenever $Z^n \leq Z_{LB}$.

Algorithm DescentX2H($MaxIter, Q_{heu}, n, Z_{LB}$)

1: Solve IPX2H(n, Q_{heu})
2: Update vector $\mathbf{q}$ with optimal solution of IPX2H(n, Q_{heu});
3: $n = n + 1$;
4: Solve the relaxed DP with bidimensional vector $\mathbf{q}$ and Q_{heu}, obtaining a solution $\mathbf{b}^n$ with value Z^n;
5: **if** ($\mathbf{b}^n$ is feasible **and** $Z^n > Z_{LB}$) **then** $Z_{LB} = Z^n$;
6: **if** ($Z^n \leq Z_{LB}$ **or** $n > MaxIter$) **then return** (Z_{LB});
7: **Goto** 1.

Algorithm X2H consists in calling DescentX2H, perhaps for different values of Q_{heu}. After

some experiments, we decided to perform a single descent, with Q_{heu} set to the value of Q obtained by the solution of IPX2$(n-1)$. Even when X2 terminates because IPX2(n) is infeasible, IPX2$(n-1)$ is certainly feasible.

5.3. G-2D Heuristic

Since Herz (1972), it is known that there is always an optimal solution where the guillotine cuts are applied only in points such that the resulting smaller rectangle has exactly the sum of the length of the items produced in the bottom of it (in case of a vertical cut) or exactly to the sum of the width of the items produced in the left of it (in case of a horizontal cut). However, it was verified that by only using cuts that correspond to the length or width of a single item it is possible to obtain optimal or high quality solutions for most practical instances. This experimental observation was recently confirmed by the extensive tests in Furini et al. (2016). The G2-D heuristics for CTGCP, first proposed in Velasco et al. (2008) and improved in Velasco and Uchoa (2014), produce solutions by repeatedly cutting *strips, a subrectangle defined by the dimensions of a single item.* For a given rectangle, there are several strips that can be cut from it. Those algorithms consider some possible strips and evaluate their strip values. Those values are not calculated exactly, they correspond to the values obtained by a constructive algorithm that try to fit items with positive residual demand into it. It must be decided which cut to perform. This could be done in a deterministic way by greedily taking the largest strip value. However, it is better to use the GRASP concepts for randomly selecting, among a list of candidates, the strip to be cut and for repeating the whole procedure many times in order to perform a diversified search in the solution space. The procedure is reactive (Prais and Ribeiro, 2000), because some parameters are dynamically adjusted based on the results of previous iterations.

G-2D Heuristic is used in X2D in two situations:

1. As a stand-alone heuristic, for building solutions from scratch. In that context, 50,000 iterations of G2-D are run in the instances where Algorithms X and X2 not performed well, as measured by the gap between the lower and upper bounds in the end of Algorithm X2. The main reason for including G-2D in X2D was to make X2D more competitive with DP_AOG, that also includes a pure primal heuristic (And/Or Graph Heuristic).

2. In the feasibility heuristic applied to near-feasible solutions obtained during Algorithms X, X2 and X2H. In those cases, a single G-2D iteration is performed to replace items in excess with combinations of smaller items with available demand. This is only done for unfeasible solutions with up to two pieces in excess. Actually, as G2-D in that context works over quite restricted areas, the resulting replacing patterns are likely to be simple, composed by few items. The feasibility heuristic in Algorithm DP_AOG also contains a procedure for replacing items in excess by combinations of items.

5.4. Complete X2D Heuristic

The X2D heuristic is composed by the following Phases:

1. Execute Algorithm X with $MaxIter = 100$ and $MaxQ = 20$. Feasible solutions may be also be obtained from suboptimal solutions in the DP matrix, perhaps with help of the feasibility heuristic. X2D stops if a certificate of optimality is obtained.

2. Execute Algorithm X2 with $MaxIter = 100$ and $MaxQ = 30$. X2 is initialized with the n unfeasible solutions found in X. Again, feasible solutions may be also be obtained from the DP matrix, perhaps using the feasibility heuristic. X2D stops if a certificate of optimality is obtained.

3. If $(Z_{UB} - Z_{LB})/Z_{LB} > 0.3\%$, execute 50,000 iterations of G2-D.

4. Execute X2H with a single descent, with Q_{heu} set to the value of Q obtained by the solution of IPX2$(n-1)$.

X2D is stopped at any point, if a time limit of 600 seconds is reached.

6. Computational Results

Algorithm X2D was tested in the following environment: single core of a processor i7-4790 at 3.6 GHz, 16GB RAM and Windows 8 (64 bits) OS. The algorithms were coded in C and compiled in Microsoft Visual Studio 2010. CPLEX 12.6 solved the IPs in Algorithms X, X2 and X2H. The tests were performed over instances available in the ESICUP web page (http://paginas.fe.up.pt/∼esicup/datasets). The first set of 30 instances corresponds to the most classical instances from the literature: 14 of them are unweighted (Classes WANG, OF and CU), the remaining 16 (Classes ChW and CW) are weighted. The second set contains 450 unweighted random instances generated by Morabito and Pureza (2010). They are divided into 3 classes: instances in Class 1 are moderately constrained (relatively large demands), those in Class 2 are more constrained (small demands), instances in Class 3 are highly constrained (unitary demands). Each class is divided into 10 groups of 15 instances. Each group is denoted by R_m_X, where the number of distinct items m can be 10, 20, 30, 40 or 50; and X can be S or L, representing small or large items, respectively. The third set of 20 instances is composed by the hard APT instances proposed in Alvarez-Valdés et al. (2002): 10 of them are unweighted (APT30-39), the remaining 10 are weighted (APT40-49).

Algorithm X2D is first compared (on the variant without rotation) with Algorithm DP_AOG in Morabito and Pureza (2010). That comparison is very relevant not only because the latter algorithm has the best published heuristic results, but also because both algorithms are primal-dual heuristics heavily based in the DPSSR of Christofides and Hadjiconstantinou (1995). Therefore, the superior performance of X2D attests the effectiveness of the proposed improvements in that

DPSSR, the main contribution of this paper. The times for DP_AOG were obtained in a processor Pentium IV 2.99GHz. According to PassMark web page (http://www.passmark.com), that processor is about 3.5 times slower than the one used in our tests.

Table 1 contains the comparison over the first set of classical instances. The value $\bar{m}$ is defined as $\sum_{i=1}^{m} D_i$. For Algorithm X2D, Z_{LB} and Z_{UB} are the lower and upper bounds obtained, $bestT$ is the time (in seconds) when the best feasible solution was found and $totalT$ is the total time (seconds) spent. For DP_AOG, only the lower bounds are available. The authors only indicate the cases where the lower bounds were certificate to be optimal. Those cases are maked with a $*$. In order to keep the notation consistent, lower bounds obtained by X2D certificate to be optimal are also marked with $*$, even though that information is already available in column Z_{UB}. It can be seen that both methods obtain all the optimal solutions. However, X2D is significantly faster and obtains two additional certificates of optimality.

Table 2 is a comparison over the second set, for Classes 1, 2 and 3. Each entry below a group name R_m_X correspond to results for the 15 instances in that group. Rows Z_{UB}, Z_{LB}, $bestT$, $totalT$ and $Gap(\%)$ are averages of upper bounds, lower bounds, time to best (seconds), total time (seconds), and percent gap. Rows $CO(\%)$ are the percentage of instances where a certificate of optimality was obtained. Columns $Avg.$ are aggregated averages over 75 instances from 5 groups. Due to the improvements in DPSSR of Christofides and Hadjiconstantinou (1995), the upper bounds from X2D are often significantly better than those from DP_AOG. They were better in 21 groups, equal in 7 groups, and worse only in groups Class 3 R_30_S and Class 3 R_40_S. Those last two results are explained by the fact that Algorithms X and X2 are being truncated by parameters $MaxIter$ and $MaxQ$. The lower bounds by X2D are also usually better, but the difference is smaller. They were better in 6 groups, equal in 22 groups and worse in groups Class 2 R_40_S and Class 3 R_20_S. As a result, X2D obtain better values of $Gap(\%)$ and $CO(\%)$ in most cases. Moreover, even taking the difference in machine speeds into account, X2D is almost always faster. The overall conclusions of the experiments reported in Table 2 are the following:

- Both primal-dual heuristics based on DPSSR of Christofides and Hadjiconstantinou (1995) (X2D and DP_AOG) are clearly better on instances with large items and with larger demands. Anyway, the gaps obtained are good, except for the instances in Class 3 with small demands. In those cases, even the improved DPSSR in X2D fails to obtain high-quality upper bounds, as a result, the lower bounds obtained from the DP matrices (even after trying the feasibility heuristic) are also poorer. In those instances the pure primal heuristic G2-D is likely to be executed and sometimes improves the lower bounds significantly. We guess that DP_AOG is also helped a lot by the pure primal And/Or Graph Heuristic in those cases.

Table 3 is a comparison of X2D with Algorithm A1, the best exact algorithm in the literature, proposed in Dolatabadi et al. (2012). As the test instances APT are larger, we increase the time limit in X2D to 900 seconds. The times for A1 were obtained in a Intel Dual CPU T3400 at

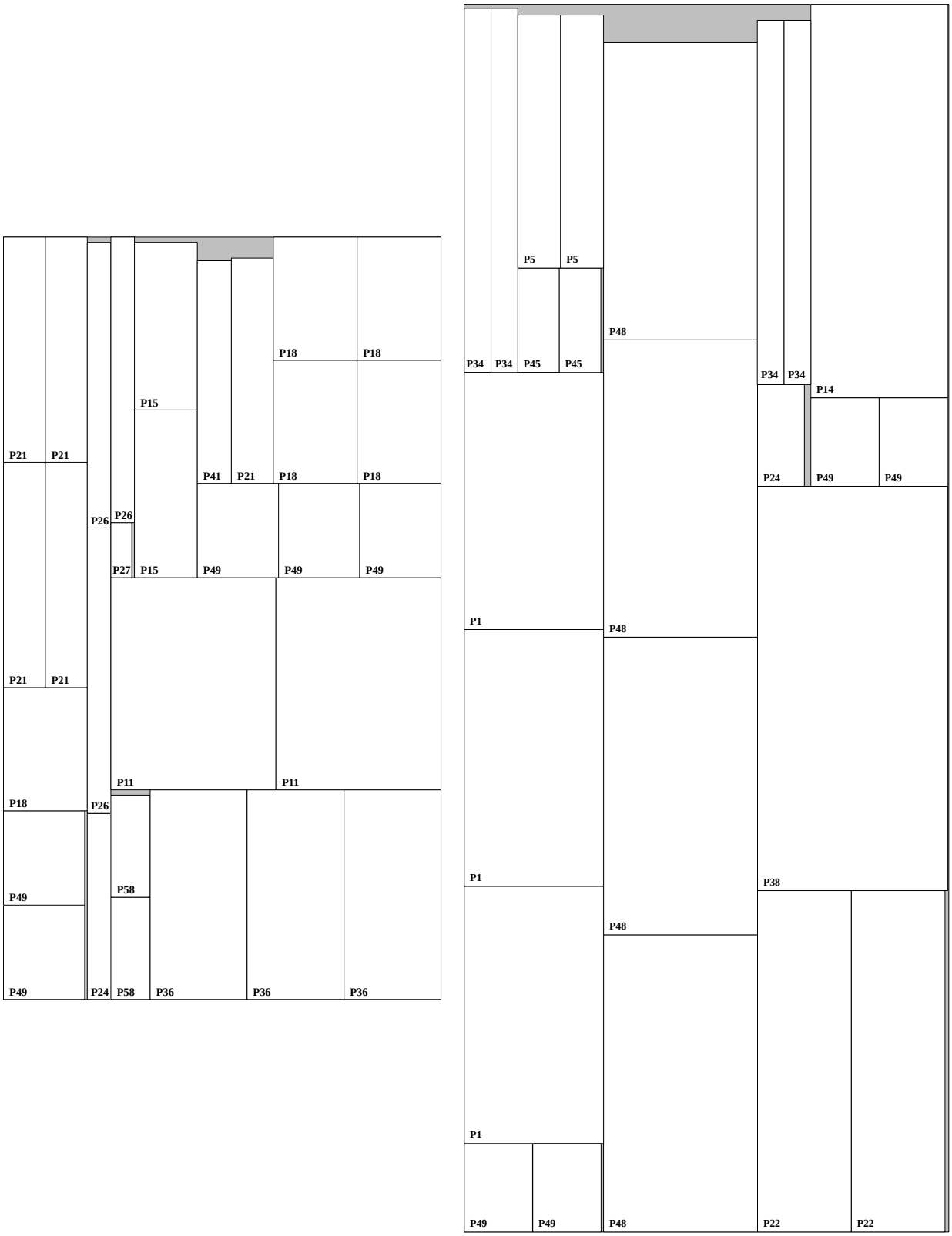

Figure 3: Improved solutions for instances APT42 and APT43 (no rotation).

2.16GHz, a processor about 3.1 times slower than the processor used in this work. The comparison is made in the ATP instances. It should be noted that A1 received as input the values of an external lower bound (taken from Hifi (2004)) and an external upper bound (taken from Chen (2008)), the times for computing those bounds are not included in their article. For example, on APT30 Algorithm A1 already received the information that the optimal solution value was 140,904, it took 2.43 seconds in order to actually find a solution with that value. Disregarding the external bound issue, A1 performed better than X2D, obtaining 18 solutions certificated to be optimal. Algorithm X2D obtained 16 of those optimal solutions, but only 8 of them could be certified. However, X2D clearly outperformed A1 in the open instances APT42 and APT43. In those cases, A1 could not improve its received external bounds in its time limit of 1 hour. In contrast, X2D not only improved the best known solutions for those two instances but also improved their best upper bounds, as marked in bold. That behavior is somehow expected:

- Exact Algorithm A1 uses clever acceleration tricks, but is still based on a worst-case exponential time enumeration. As so, it can solve many instances to optimality in reasonable times, but it can also fail completely in some harder/larger instances. On the other hand, Primal Dual Heuristic Algorithm X2D is based on a DPSSR that has a pseudo-polynomial worst-case complexity (assuming limited $MaxIter$ and $MaxQ$), so (unless L and W are too large) it may produce reasonable results even for those harder/larger instances.

The solution with value 33,598 for APT42 is depicted in the left of Figure 3. That solution was directly found (the feasibility heuristic was not called) as a suboptimal solution of the DP during Phase 2, in an iteration where $n = 47$ and $Q = (12, 7)$. The solution with value 217,288 for APT43 in the right of Figure 3 was directly found as a suboptimal solution of the DP during Phase 1, in an iteration where $n = 7$ and $Q = 20$.

The algorithms proposed in this paper can be easily adapted to the CTGCP variant that allows item rotations. There are only two differences: (1) The discretization points should be calculated as the conic combinations of both item lengths and widths; (2) the base of the DP recursions, Equations (5) and (13), should consider the possibility that an item is rotated. Besides that, Algorithms X, X2 and X2H remain the same. Tables 4, 5 and 6 present results for the CTGCP. Unhappily, there are no recent algorithms in the literature for comparisons. It seems that allowing rotations make the instances a bit easier for X2D. The total number of solutions certified to be optimal increases from 342 to 377.

Finally, Table 7 is aimed at showing the contribution of each element of X2D for its results. The rows are organized by phases. Columns CO and Z_{UB} shows how many certificates of optimality and best upper bounds were obtained in Phases 1 and 2. Column Z_{UB} show how many times the best lower bound found for an instance was found in each phase, from Phase 1 to 4. The next columns detail that information, for Phases 1, 2 and 4: how many times the lower bound was found directly as an optimal solution of the DP, by the feasibility heuristic over an optimal

DP solution, as a suboptimal solution in the DP matrix, and by the feasibility heuristic over a suboptimal DP solution.

Instances					OPT	X2D				DP_AOG		
name	m	$\bar{m}$	L	W		Z_{LB}	Z_{UB}	$bestT$	$totalT$	Z_{LB}	$bestT$	$totalT$
WANG1	20	42	33	69	2277	*2277	2277	0.01	0.01	*2277	0.1	0.1
WANG2	20	42	39	70	2694	*2694	2694	0.01	0.01	*2694	0.1	0.2
WANG3	20	42	40	70	2721	*2721	2721	0.01	0.02	*2721	0.1	0.6
OF1	10	23	70	40	2737	*2737	2737	0.03	0.05	*2737	0.1	11.2
OF2	10	24	70	40	2690	*2690	2690	0.11	0.25	2690	0.1	23.6
CU1	25	82	100	125	12330	*12330	12330	0.01	0.01	*12330	0.1	0.5
CU2	35	90	150	175	26100	*26100	26100	0.01	0.01	*26100	0.2	1.4
CU3	45	158	134	125	16723	*16723	16723	0.01	0.02	*16723	0.3	5.8
CU4	45	113	285	354	99495	*99495	99495	0.01	0.05	*99495	3.8	35.0
CU5	50	120	456	385	173364	*173364	173364	0.02	0.12	*173364	2.7	873.2
CU6	45	124	356	447	158572	*158572	158572	0.02	0.02	*158572	1.3	11.8
CU7	25	56	563	458	247150	247150	247392	0.30	0.64	247150	0.1	1800.0
CU8	35	78	587	756	433331	*433331	433331	0.01	0.01	*433331	0.5	26.8
CU9	25	76	856	785	657055	*657055	657055	0.01	0.01	*657055	0.1	19.7
CU10	40	129	794	985	773772	773772	774954	2.33	52.61	773772	69.2	1800.0
CU11	50	134	977	953	924696	924696	925276	0.16	600.00	924696	333.6	1293.1
Average								0.18	40.86		25.8	368.9
ChW1	7	16	15	10	244	*244	244	0.10	0.11	*244	0.1	0.1
ChW2	10	23	40	70	2892	*2892	2892	0.34	0.34	*2892	0.8	38.4
ChW3	20	62	40	70	1860	*1860	1860	0.03	0.05	*1860	0.1	19.9
CW1	25	67	125	105	6402	*6402	6402	0.01	0.02	*6402	1.1	25.6
CW2	35	63	145	165	5354	*5354	5354	0.05	0.44	5354	0.7	1118.8
CW3	40	96	267	207	5689	*5689	5689	0.03	0.03	*5689	0.5	26.9
CW4	39	86	465	387	6175	*6175	6175	0.09	0.24	*6175	1.3	46.8
CW5	35	91	524	678	11659	*11659	11659	0.10	0.10	*11659	0.8	1268.5
CW6	55	149	781	657	12923	*12923	12923	0.13	0.13	*12923	33.3	890.6
CW7	45	123	276	374	9898	*9898	9898	0.01	0.01	*9898	4.1	10.7
CW8	60	168	305	287	4605	*4605	4605	0.05	0.08	*4605	20.8	340.5
CW9	50	131	405	362	10748	*10748	10748	0.12	0.12	*10748	7.0	121.3
CW10	60	130	992	970	6515	*6515	6515	0.15	0.15	*6515	2.8	182.2
CW11	60	114	982	967	6321	*6321	6321	0.32	1.22	*6321	2.0	1375.0
Average								0.11	0.22		5.4	390.4

Table 1: Comparison with Algorithm DP_AOG on 20 classical instances (no rotation), unweighted and weighted.

7. Conclusions

The main contribution of this article was Algorithm X, an improved scheme based on integer programming for updating the weights in a DPSSR for CTCGP. Unless the scheme is truncated, it always obtains the optimal weights. The general principle behind the new scheme, that is valid to any kind of combinatorial relaxation, is the following: *a relaxation can only improve the current dual bound if it forbids all the known unfeasible solutions with value better than that bound.* Additional contributions are Algorithm X2, a new DPSSR for CTCGP that uses bidimensional weights, and a full primal-dual heuristic called X2D. Extensive tests with X2D show that it can indeed obtain optimal or near-optimal solutions in many cases. Comparisons with the best

	Class 1	R_10_S	R_20_S	R_30_S	R_40_S	R_50_S	Avg.	R_10_L	R_20_L	R_30_L	R_40_L	R_50_L	Avg.
D	Z_{LB}	9834	9976	10000	10000	10000	9962.0	9129	9595	9835	9861	9892	9662.4
P	Z_{UB}	9838	9978	10000	10000	10000	9963.2	9129	9598	9838	9880	9898	9668.6
	bestT	8.3	7.8	3.1	1.4	0.3	4.2	< 0.1	1.1	< 0.1	< 0.1	< 0.1	0.3
A	*totalT*	46.3	79.6	7.3	5.8	4.2	28.70	0.8	6.7	7.2	12.8	8.5	7.2
O	Gap(%)	0.04	0.02	0	0	0	0.01	0	0.03	0.03	0.19	0.06	0.06
G	CO(%)	93.3	86.7	100	100	100	96.0	100	93.3	93.3	80.0	93.3	92.0
	Z_{LB}	9834	9976	10000	10000	10000	9962.0	9129	9595	9835	9861	9892	9662.4
X	Z_{UB}	9835	9977	10000	10000	10000	9962.4	9129	9595	9835	9863	9892	9662.8
2	*bestT*	0.1	0.2	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1
D	*totalT*	0.1	1.5	< 0.1	< 0.1	0.3	0.1	< 0.1	< 0.1	< 0.1	0.1	< 0.1	< 0.1
	Gap(%)	0.01	0.01	0	0	0	< 0.01	0	0	0	0.02	0	< 0.01
	CO(%)	93.3	93.3	100	100	100	97.3	100	100	100	93.3	100	98.7

	Class 2	R_10_S	R_20_S	R_30_S	R_40_S	R_50_S	Avg.	R_10_L	R_20_L	R_30_L	R_40_L	R_50_L	Avg.
D	Z_{LB}	9640	9909	9958	9996	10000	9900.6	9230	9635	9842	9867	9902	9695.2
P	Z_{UB}	9829	9961	9998	10000	10000	9957.6	9253	9660	9842	9870	9904	9705.8
	bestT	19.1	46.2	85.4	48.3	26.1	45.00	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1
A	*totalT*	942.3	929.9	1206.4	350.6	31.6	692.20	8.3	19.8	7.6	12.6	9.9	12
O	Gap(%)	1.92	0.52	0.40	0.04	0	0.58	0.25	0.26	0	0.03	0.02	0.11
G	CO(%)	6.7	6.7	6.7	66.7	100	37.4	86.7	60.0	93.3	86.7	93.3	84.0
	Z_{LB}	9644	9909	9960	9995	10000	9901.6	9230	9635	9842	9867	9902	9695.2
X	Z_{UB}	9717	9935	9981	10000	10000	9926.6	9242	9635	9842	9867	9902	9697.6
2	*bestT*	7.5	0.5	6.8	36.7	8.6	12.00	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1
D	*totalT*	89.7	74.3	92.8	60	8.6	65.10	0.4	< 0.1	< 0.1	< 0.1	< 0.1	0.1
	Gap(%)	0.75	0.26	0.21	0.05	0	0.25	0.13	0	0	0	0	0.03
	CO(%)	20.0	20.0	26.7	66.7	100	46.7	93.3	100	100	100	100	98.7

	Class 3	R_10_S	R_20_S	R_30_S	R_40_S	R_50_S	Avg.	R_10_L	R_20_L	R_30_L	R_40_L	R_50_L	Avg.
D	Z_{LB}	8115	9575	9671	9807	9770	9387.6	8746	9322	9685	9767	9823	9468.6
P	Z_{UB}	9263	9923	9880	9890	10000	9791.2	8883	9469	9745	9832	9881	9562.0
	bestT	0.7	89.5	194.7	425.7	1.73	142.50	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1
A	*totalT*	752.2	532.6	638.6	670.6	1442.4	807.30	11.9	42.4	47	46.2	59.2	41
O	Gap(%)	12.39	3.51	2.12	0.84	2.30	4.23	1.54	1.55	0.62	0.66	0.59	0.99
G	CO(%)	0	0	0	0	0	0	46.7	33.3	40.0	26.7	26.7	34.7
	Z_{LB}	8116	9544	9751	9865	9903	9435.8	8746	9322	9685	9767	9823	9468.6
X	Z_{UB}	8910	9844	9952	9996	9998	9740.0	8769	9364	9708	9778	9842	9492.2
2	*bestT*	9.9	108.1	120.2	85.5	184.9	101.70	0.1	< 0.1	0.1	0.1	0.2	0.1
D	*totalT*	39.4	167.6	355.3	279.7	257.4	219.90	0.6	1.5	3.6	2	5.3	2.6
	Gap(%)	8.91	3.05	2.02	1.31	0.95	3.25	0.26	0.45	0.24	0.11	0.19	0.25
	CO(%)	6.7	0	0	0	0	1.3	86.7	73.3	53.3	80.0	40.0	66.7

Table 2: Comparison with Algorithm DP_AOG on 450 instances (no rotation) by Morabito and Pureza (2010).

Instances					OPT/	X2D				A1			
name	m	$\bar{m}$	L	W	BKS	Z_{LB}	Z_{UB}	$bestT$	$totalT$	$ExtZ_{LB}$	$ExtZ_{UB}$	Z_{LB}	$totalT$
APT30	38	192	152	927	140904	*140904	140904	23.67	23.67	140904	140904	*140904	2.43
APT31	51	258	964	856	823976	823976	824483	150.24	900.00	823976	824931	*823976	178.99
APT32	56	249	124	307	38068	*38068	38068	23.98	23.98	38068	38068	*38068	0.37
APT33	44	224	983	241	236611	236588	236678	744.03	900.00	236611	236818	*236611	40.62
APT34	27	130	456	795	361398	361398	362219	176.17	900.00	361197	362520	*361398	79.08
APT35	29	153	649	960	621021	620948	622161	18.89	900.00	621021	622644	*621021	15.36
APT36	28	153	244	537	130744	130744	130800	1.43	900.00	130744	130744	*130744	18.06
APT37	43	222	881	440	387276	*387276	387276	24.27	277.31	387276	387276	*387276	48.03
APT38	40	202	358	731	261395	261395	261572	117.68	900.00	261395	261698	*261395	44.63
APT39	33	163	501	538	268750	268750	268926	602.00	900.00	268750	268750	*268750	33.70
Average								128.24	662.50				46.13
APT40	56	290	138	683	67154	*67154	67154	1.04	23.41	67154	67654	*67154	25.86
APT41	36	177	367	837	206542	*206542	206542	2.14	59.18	206542	215699	*206542	229.30
APT42	59	325	291	167	33503	**33598**	**33680**[1]	280.27	900.00	33503	34098	33503	3600.00
APT43	49	259	917	362	214651	**214840**	**217288**	24.05	900.00	214651	222570	214651	3600.00
APT44	39	196	496	223	73868	*73868	73868	93.54	194.20	73868	74887	*73868	19.23
APT45	33	156	578	188	74691	*74691	74691	0.53	7.40	75808[2]	75888	*74691	19.73
APT46	42	197	514	416	149911	*149911	149911	0.62	0.62	149911	151813	*149911	37.00
APT47	43	204	554	393	150234	150234	150659	286.57	900.00	150234	153747	*150234	51.80
APT48	34	167	254	931	167660	167660	167835	144.38	900.00	167660	170914	*167660	75.93
APT49	25	119	449	759	219354	219354	220383	254.31	900.00	218388	226346	*219354	2680.39
Average								108.74	478.48				1033.92

Table 3: Comparison with Algorithm A1 on 30 instances (no rotation) by Alvarez-Valdés et al. (2002).
[1] Our UB contradicts the LB of 34,015 reported in Chen (2008). So, we believe that the LB of 33,598 is indeed the new best.
[2] This LB, reported in Chen (2008), could not be found by any of the exact methods tried in Dolatabadi et al. (2012). According to our code, the optimal solution of APT45 indeed has value 74,691.

Instance	X2D			
name	Z_{LB}	Z_{UB}	$bestT$	$totalT$
WANG1	*2277	2277	< 0.01	< 0.01
WANG2	*2716	2716	< 0.01	< 0.01
WANG3	*2771	2771	0.02	0.02
OF1	*2757	2757	0.02	0.53
OF2	*2769	2769	0.27	0.31
CU1	*12500	12500	< 0.01	< 0.01
CU2	*26200	26200	< 0.01	0.02
CU3	*16750	16750	< 0.01	< 0.01
CU4	*100230	100230	0.05	0.33
CU5	*174705	174705	0.08	0.02
CU6	*158572	158572	0.02	0.02
CU7	*255684	255684	0.02	0.02
CU8	*438383	438383	0.03	0.22
CU9	*659648	659648	0.02	0.09
CU10	*779239	779239	0.16	0.16
CU11	928170	928215	0.38	600.00
Average			0.06	37.61

Instance	X2D			
name	Z_{LB}	Z_{UB}	$bestT$	$totalT$
ChW1	*260	260	0.02	0.23
ChW2	*2901	2901	0.03	0.94
ChW3	*1920	1920	0.23	1.05
CW1	*6766	6766	0.02	0.30
CW2	*5689	5689	0.03	0.28
CW3	*5744	5744	0.06	7.61
CW4	*7496	7496	0.52	0.52
CW5	*11659	11659	0.47	0.47
CW6	*13203	13203	0.34	19.25
CW7	*10880	10880	0.02	0.02
CW8	*4736	4736	0.02	0.13
CW9	*11479	11479	0.27	0.27
CW10	*6835	6835	1.39	1.39
CW11	*6784	6784	1.53	2.48
Average			0.35	2.49

Table 4: Results on 20 classical instances (with rotation), unweighted and weighted.

Class 1	R_10_S	R_20_S	R_30_S	R_40_S	R_50_S	Avg.	R_10_L	R_20_L	R_30_L	R_40_L	R_50_L	Avg.
Z_{LB}	9961	9999	10000	10000	10000	9992.0	9286	9774	9904	9929	9971	9772.8
Z_{UB}	9961	9999	10000	10000	10000	9992.0	9286	9776	9904	9929	9971	9773.2
bestT	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1
totalT	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	0.1	< 0.1	0.1	< 0.1	< 0.1
Gap(%)	0	0	0	0	0	0	0	0.02	0	0	0	< 0.01
CO(%)	100	100	100	100	100	100	100	93.3	100	100	100	98.7

Class 2	R_10_S	R_20_S	R_30_S	R_40_S	R_50_S	Avg.	R_10_L	R_20_L	R_30_L	R_40_L	R_50_L	Avg.
Z_{LB}	9850	9976	10000	10000	10000	9965.4	9546	9797	9909	9945	9966	9832.6
Z_{UB}	9887	9989	10000	10000	10000	9975.2	9546	9797	9909	9945	9966	9832.6
bestT	18.2	1.4	2.5	0.7	0.6	4.70	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1
totalT	175.1	90.4	2.5	0.7	0.6	53.90	< 0.1	< 0.1	0.1	0.1	< 0.1	< 0.1
Gap(%)	0.37	0.13	0	0	0	0.10	0	0	0	0	0	0
CO(%)	20.0	46.7	100	100	100	73.3	100	100	100	100	100	100

Class 3	R_10_S	R_20_S	R_30_S	R_40_S	R_50_S	Avg.	R_10_L	R_20_L	R_30_L	R_40_L	R_50_L	Avg.
Z_{LB}	8413	9781	9887	9949	9967	9599.4	9183	9621	9803	9900	9927	9686.8
Z_{UB}	9205	9950	10000	10000	10000	9831.0	9188	9624	9812	9910	9933	9693.4
bestT	3.2	80.4	91	13.3	33.2	44.20	0.1	0.1	0.7	0.1	< 0.1	0.2
totalT	75.2	180.5	161.9	49.3	41.5	101.70	0.7	0.4	1.9	1.8	1.7	1.3
Gap(%)	8.60	1.70	1.13	0.51	0.33	2.45	0.05	0.03	0.09	0.10	0.06	0.07
CO(%)	0	0	0	6.7	13.3	4.0	93.3	93.3	80.0	60.0	73.3	80.0

Table 5: Results on 450 instances (with rotation) by Morabito and Pureza (2010).

Instance	X2D			
name	Z_{LB}	Z_{UB}	*bestT*	*totalT*
APT30	*140904	140904	771.18	771.18
APT31	824878	825184	723.32	900.00
APT32	*38068	38068	424.94	458.79
APT33	*236903	236903	0.39	0.39
APT34	361952	362520	499.10	900.00
APT35	622518	622965	144.88	900.00
APT36	130965	130988	0.13	900.00
APT37	387439	387640	19.54	900.00
APT38	261625	261698	288.06	900.00
APT39	269278	269376	38.16	900.00
Average			290.97	753.04

Instance	X2D			
name	Z_{LB}	Z_{UB}	*bestT*	*totalT*
APT40	*67294	67294	103.19	226.49
APT41	210713	211613	140.04	900.00
APT42	33756	34010	880.95	900.00
APT43	218820	219654	727.45	900.00
APT44	76122	76306	235.88	900.00
APT45	*74691	74691	1.15	16.70
APT46	*150983	150983	18.48	56.73
APT47	152778	152951	89.38	900.00
APT48	170678	171896	135.95	900.00
APT49	222248	224987	727.90	900.00
Average			306.04	659.99

Table 6: Results on 30 instances (with rotation) by Alvarez-Valdés et al. (2002).

	CO	Z_{UB}	Z_{LB}	Opt.DP	Feas.Opt	Sub.DP	Feas.Sub
Without Rotation							
Phase 1 (X)	286	329	382	134	12	214	22
Phase 2 (X2)	56	171	74	7	0	45	22
Phase 3 (G-2D)	-	-	34	-	-	-	-
Phase 4 (X2H)	-	-	10	1	0	8	1
With Rotation							
Phase 1 (X)	316	373	403	147	14	201	41
Phase 2 (X2)	61	127	60	13	0	35	12
Phase 3 (G-2D)	-	-	36	-	-	-	-
Phase 4 (X2H)	-	-	1	0	0	1	0

Table 7: Effectiveness of each X2D Phase.

previous heuristic and exact algorithms for the CTCGP without rotation are made. We also provide results for the important CTCGP variant that permits rotations.

Modern codes are very efficient on solving linear IPs with only a few dozen variables and constraints. In fact, at least in our experiments, the time for solving the IPs in Algorithms X, X2 and X2H was always negligible with respect to the time spent to solve the DP recursions. Nevertheless, in the future it could be interesting to devise fast heuristics for solving the IPs with quadratic objective function that arise when bidimensional weights are used.

Acknowledgments

This work was partially supported by the Brazilian research agencies CNPq and Faperj.

References

R. Alvarez-Valdés, A. Parajón, J. M. Tamarit, A tabu search algorithm for large-scale guillotine (un)constrained two-dimensional cutting problems, Computers & Operations Research 29 (7) (2002) 925 – 947.

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

Y. Chen, A recursive algorithm for constrained two-dimensional cutting problems, Computational Optimization and Applications 41 (3) (2008) 337–348.

N. Christofides, E. Hadjiconstantinou, An exact algorithm for orthogonal 2-d cutting problems using guillotine cuts, European Journal of Operational Research 83 (1) (1995) 21 – 38.

N. Christofides, C. Whitlock, An algorithm for two-dimensional cutting problems, Oper. Res. 25 (1) (1977) 30–44.

G. Cintra, F. Miyazawa, Y. Wakabayashi, E. Xavier, Algorithms for two-dimensional cutting stock and strip packing problems using dynamic programming and column generation, European Journal of Operational Research 191 (1) (2008) 61 – 85.

V.-D. Cung, M. Hifi, B. Le Cun, Constrained two-dimensional cutting stock problems a best-first branch-and-bound algorithm, International Transactions in Operational Research 7 (3) (2000) 185–210.

M. Dolatabadi, A. Lodi, M. Monaci, Exact algorithms for the two-dimensional guillotine knapsack, Computers & Operations Research 39 (1) (2012) 48–53.

F. Furini, E. Malaguti, D. Thomopulos, Modeling two-dimensional guillotine cutting problems via integer programming, INFORMS Journal on Computing 28 (4) (2016) 736–751.

P. C. Gilmore, R. E. Gomory, Multistage cutting stock problems of two and more dimensions, Operations Research 13 (1) (1965) 94–120.

J. C. Herz, Recursive computational procedure for two-dimensional stock cutting, IBM J. Res. Dev. 16 (5) (1972) 462–469.

M. Hifi, Dynamic programming and hill-climbing techniques for constrained two-dimensional cutting stock problems, Journal of Combinatorial Optimization 8 (1) (2004) 65–84.

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

R. Morabito, V. Pureza, A heuristic approach based on dynamic programming and and/or-graph search for the constrained two-dimensional guillotine cutting problem, Annals of Operations Research 179 (1) (2010) 297–315.

M. Prais, C. C. Ribeiro, Reactive GRASP: An application to a matrix decomposition problem in TDMA traffic assignment, INFORMS Journal on Computing 12 (3) (2000) 164–176.

A. Velasco, G. G. Paula Junior, E. Vieira Neto, Um algoritmo heurístico baseado na GRASP para o problema de corte bidimensional guilhotinado e restrito, Gepros: Gestão da Produção, Operações e Sistemas 3 (1) (2008) 129–141.

A. Velasco, E. Uchoa, Geração de padrões de corte bidimensionais guilhotinados via GRASP, in: Proceedings of XLVI SBPO, Salvador, 1–12, 2014.