CIRRELT

Centre interuniversitaire de recherche
sur les réseaux d'entreprise, la logistique et le transport

Interuniversity Research Centre
on Enterprise Networks, Logistics and Transportation

An Exact Algorithm for the Two-Dimensional Orthogonal Packing Problem with Unloading Constraints

Jean-François Côté
Michel Gendreau
Jean-Yves Potvin

April 2013

CIRRELT-2013-26

Bureaux de Montréal :
Université de Montréal
C.P. 6128, succ. Centre-ville
Montréal (Québec)
Canada H3C 3J7
Téléphone : 514 343-7575
Télécopie : 514 343-7121

Bureaux de Québec :
Université Laval
2325, de la Terrasse, bureau 2642
Québec (Québec)
Canada G1V 0A6
Téléphone : 418 656-2073
Télécopie : 418 656-2624

www.cirrelt.ca

An Exact Algorithm for the Two-Dimensional Orthogonal Packing Problem with Unloading Constraints

Jean-François Côté[1,2], Michel Gendreau[1,3], Jean-Yves Potvin[1,2,*]

[1] Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT)

[2] Department of Computer Science and Operations Research, Université de Montréal, P.O. Box 6128, Station Centre-Ville, Montréal, Canada H3C 3J7

[3] Department of Mathematics and Industrial Engineering, École Polytechnique de Montréal, P.O. Box 6079, Station Centre-ville, Montréal, Canada H3C 3A7

Abstract. This paper describes a branch-and-cut algorithm for solving a two-dimensional orthogonal packing problem with unloading constraints, which often occurs as a subproblem of mixed vehicle routing and loading problems. At each node of the branching tree, cuts are generated to find a feasible integer solution to a one-dimensional contiguous bin packing problem. If an integer solution is obtained, an auxiliary problem is then solved to find a feasible two-dimensional packing of the items. Different techniques to reduce the size of the solution space and uncover infeasibility are also described. A numerical comparison with the best known exact method is reported at the end on benchmark instances.

Keywords: Vehicle routing, unloading constraints, packing, branch-and-cut.

Acknowledgements. Financial support for this work was provided by the Natural Sciences and Engineering Research Council of Canada (NSERC). This support is gratefully acknowledged.

---

* Corresponding author: Jean-Yves.Potvin@cirrelt.ca

Dépôt légal – Bibliothèque et Archives nationales du Québec
Bibliothèque et Archives Canada, 2013

1 Introduction

Typically, when solving a mixed vehicle routing and loading problem, a delivery route is first generated and then a feasibility check is performed to determine if the goods can be feasibly packed inside the vehicle. This feasibility problem, in two dimensions (2D) or three dimensions (3D), is far from obvious and the work reported here focuses on this issue.

Let us consider the delivery route $\mathcal{R}$ of two-dimensional rectangular objects (called items, thereafter), which is defined as a sequence of cardinality R over a set of customers $J = \{1, 2, ..., R\}$. Without loss of generality, we will assume in the following that customer j is the j-th customer in the sequence, that is, customer 1 is delivered first, customer 2 is delivered second, etc. Let us also consider the set of items $I = \{1, ..., n\}$ to be delivered to the customers, where each item $i \in I$ is characterized by its width w_i, its height h_i, as well as its associated customer and delivery order $seq_i \in \{1, 2, ..., R\}$ in route $\mathcal{R}$. Note that all items delivered to a given customer have the same delivery order. The loading area of the vehicle (called bin, thereafter) has width W and height H. Given these assumptions, we want to know if the items can fit inside the bin, that is, without overlap and in such a way that the items of any given customer are directly available at delivery time (i.e., the items can be moved out by directly pulling each one of them outside of the bin without moving any other item). The latter characteristic will be referred to as the *unloading constraints*. It should be noted that the *unloading constraints* are also referred to as *sequential loading*, *rear loading*, *multi-drop* or *LIFO* constraints in the literature. We prefer *unloading constraints* because these constraints relate to the deliveries.

Figure 1 presents an example for a route that starts from the depot 0 and then visits the customers 1, 2, 3, 4 and 5. Figures 1(a) and 1(b) show two possible packings for this route: the first one satisfies the unloading constraints while the second one does not (note that the items are taken out from the top of the bin, which corresponds to the rear of the vehicle). In the second case, the items of customers 2 must clearly be moved to allow the items of customer 1 to be unloaded.

This problem is the classical *2D Orthogonal Packing Problem* (2OPP) with the addition of unloading constraints (UL). The 2OPP is often found as a subproblem of the *2D Strip Packing Problem* (2SPP) and the *2D Bin Packing Problem* (2BPP). Recent exact methods for the 2OPP can be found in [7, 11, 14, 24], while exact methods for the 2SPP are found in [2, 3, 6, 10, 23]. A recent work on the 2SPP with unloading constraints is also reported in [26] where a GRASP heuristic, previously proposed in [1], and two approximation algorithms are used to solve the problem.

The 2OPP-UL occurs in several papers as a subproblem of mixed vehicle routing and loading problems, when unloading constraints apply as well. In these problems, a fleet of vehicles must visit a set of customers while minimizing the total traveled distance. In addition, the items in the route of each vehicle must be feasibly packed.

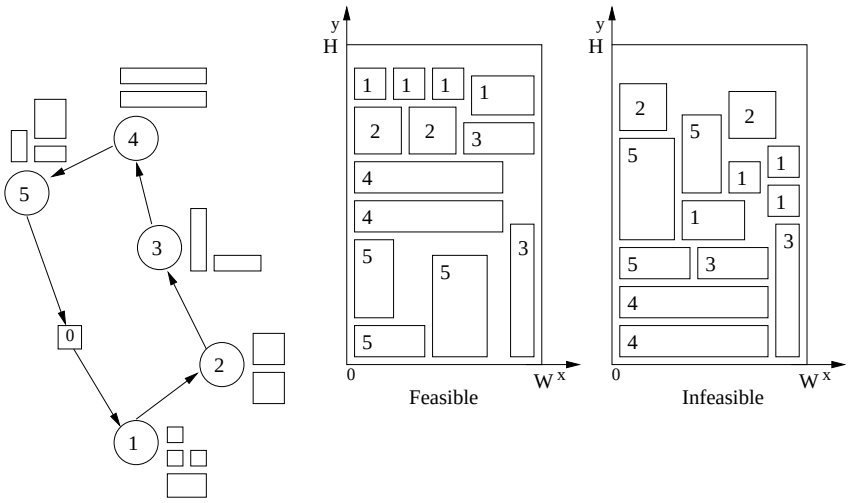

Figure 1: Packing examples

The literature on this problem distinguishes four main variants, depending if the items can be rotated by 90 degrees (*non-oriented*) or not (*oriented*) and if unloading constraints apply or not:

- $2|OU|$: two-dimensional oriented with unloading constraints;
- $2|OR|$: two-dimensional oriented without unloading constraints;
- $2|NU|$: two-dimensional non-oriented with unloading constraints;
- $2|NR|$: two-dimensional non-oriented without unloading constraints.

Our work focuses on the $2|OU|$ variant. The $2|OR|$ and $2|NR|$ variants are related to the 2OPP and a vast literature can be found on the subject. We believe that the work reported here can also be generalized to the non-oriented variant $2|NU|$.

The heuristics reported in the literature for mixed vehicle routing and loading problems are often local search heuristics which are designed to improve the current solution by applying some modification to it (see, for example, [12, 15, 16, 22, 29]). A modification to a vehicle route is accepted only if all constraints related to the packing are satisfied. This is typically verified with simple heuristics like the Bottom-Left, Bottom-Left Fill and Touching Perimeter heuristics [29].

Only a few exact algorithms are reported in the literature for these problems due, in particular, to the inherent difficulty of the packing. In [20], a branch-and-cut algorithm is proposed where the classical Bottom-Left heuristic if first used and, if it fails, an exact branch-and-bound algorithm based on the work in [23] is applied. More recently, the authors in [9] describe a branch-and-cut-and-price

algorithm where a similar, but more involved, tree-based search is proposed. A lower bound that takes into account the unloading constraints is also reported. The reader is referred to the exhaustive surveys in [5, 19] for the three-dimensional case.

When considering the 2OPP, a relaxation known as the *One-dimensional Contiguous Bin Packing Problem* (1CBP) can be used to generate an initial lower bound on the required height of the bin. This relaxation is defined as follows:

Definition 1.1 *Given that each item i of width w_i is cut into h_i slices of unitary height, we ask for the packing of all slices into a minimum number of bins of capacity W, in such a way that if the first slice of item i is packed into bin j, then the kth slice is packed into bin $j + k$, $k \in \{1, 2, ..., h_i\}$.*

The number of bins obtained is a lower bound on the required height H. That is, if the resulting lower bound is larger than H then the problem is infeasible, otherwise the problem remains undecided. Note that if we have a feasible solution for the 2OPP, a feasible solution of the 1CBP can be obtained by considering the y coordinate of the bottom-left corner of each item as the height position of its starting slice in the 1CBP. Conversely, one could try to build a feasible solution for the 2OPP from a 1CBP solution by using the height position of the first slice of every item as the set of y-coordinates and then by determining a set of x-coordinates that does not lead to any overlap. This kind of two-phase approach can be found in [10], although the authors go the other way around by first solving for the x-coordinates and then, for the y-coordinates. Intuitively, this approach looks good in our case, given that the width W (x-axis) of a vehicle's loading area is typically smaller than its height H (y-axis). It means that the first problem is smaller, while the second problem exhibits a larger degree of freedom when suitable y-coordinates must be found. Unfortunately, a preliminary computational study has shown that the structure of the x-coordinates obtained by solving the first problem seems to preclude, in most cases, the identification of y-coordinates that satisfy the unloading constraints.

Given the above observations, we propose a branch-and-cut algorithm to solve the 2OPP-UL, where the 1CBP is first solved to get y-coordinates, followed by a so-called x-check problem to get the corresponding x-coordinates. Several preprocessing routines and inequalities that take into account the unloading constraints are also proposed to tighten the problem.

The remainder of the paper is organized as follows. A basic mathematical model is first introduced in Section 2. Then, the outline of the problem-solving methodology is explained in Section 3. Various inequalities for LP relaxations of the 1CBP are reported in Section 4, including a description of the separation routines associated with these inequalities. Section 5 describes the various preprocessing routines, while the lower bounds are presented in Section 6. Computational results are finally reported in Section 7.

2 Model

In the following, the mathematical notation is first presented. Then, the notion of normal patterns is introduced to reduce, without impairing optimality, the set of solutions to be examined. After these preliminaries, an integer programming model for the 2OPP-UL is reported.

2.1 Notation

The following notation will be used throughout the paper:

- I_i : set of items with delivery order i (same customer),
- $I_i^{=}$: set of items delivered at the same time than item i,
- $I_i^{<}$: set of items delivered before item i,
- $I_i^{<,w}$: set of items delivered before item i of width $> w$,
- $I_i^{>}$: set of items delivered after item i,
- $I_i^{>,w}$: set of items delivered after item i of width $> w$,
- $\Sigma_{(i)}^{>} = \sum_{\substack{k \in I \setminus \{i\} \\ seq_k > seq_i}} w_k h_k$: total area of the items delivered after item i,
- $\Sigma_{(i)}^{>,w} = \sum_{\substack{k \in I \setminus \{i\} \\ seq_k > seq_i \\ w_k > w}} w_k h_k$: total area of the items of width $> w$ delivered after item i ,
- $\Sigma_{(i,j)}^{<} = \sum_{\substack{k \in I \setminus \{i,j\} \\ seq_i < seq_k < seq_j}} w_k h_k$: total area of the items delivered after item i but before item j,
- $\Sigma_{(i,j)}^{<,w} = \sum_{\substack{j \in I \setminus \{i,j\} \\ seq_i < seq_k < seq_j \\ w_k > w}} w_k h_k$: total area of the items of width $> w$ delivered after item i but before item j.

This notation is extended in the following to include items that are delivered at the same time than item i (i.e., the $<$ and $>$ signs are replaced by $\leq$ and $\geq$ signs, respectively).

2.2 Normal Patterns

In general, the bottom-left corner of a given item $i \in I$ can be found at every (x, y) coordinate where $x \in [0, W - w_i]$ is the width position and $y \in [0, H - h_i]$ is the

height position. This set of coordinates can however be reduced by considering only normal patterns [18], which are defined as follow:

$$P_i^W = \{x = \sum_{j \in I \backslash \{i\}} w_j \xi_j : 0 \leq x \leq W - w_i, \xi_j \in \{0,1\}, j \in I \backslash \{i\}\} \quad (1)$$

$$P_i^H = \{y = \sum_{j \in I \backslash \{i\}} h_j \xi_j : 0 \leq y \leq H - h_i, \xi_j \in \{0,1\}, j \in I \backslash \{i\}\} \quad (2)$$

When the unloading constraints are considered, the set P_i^H can be reduced by observing that items to be delivered after item i cannot be over it:

$$P_i^H = \{y = \sum_{j \in I_i^{\geq} \backslash \{i\}} h_j \xi_j : 0 \leq y \leq H - h_i, \xi_j \in \{0,1\}, j \in I_i^{\geq} \backslash \{i\}\} \quad (3)$$

We define $P^W = \bigcup_{i \in I} P_i^W$ and $P^H = \bigcup_{i \in I} P_i^H$ as the set of all possible width and height positions. We also define the set $P_i^W(r)$ as the set of positions that cover width position r. The set $P_i^H(t)$ is defined similarly for height position t:

$$P_i^W(r) = \{x \in P_i^W : [r - w_i + 1]^+ \leq x \leq r\}, \ i \in I, r \in P^W \quad (4)$$

$$P_i^H(t) = \{y \in P_i^H : [t - h_i + 1]^+ \leq y \leq t\}, \ i \in I, t \in P^H$$

2.3 Mathematical model

Given the decision variables

- x_{ir} is 1 if the bottom-left corner of item i is located at width position r, 0 otherwise, $i \in I$, $r \in P_i^W$;
- y_{it} is 1 if the bottom-left corner of item i is located at height position t, 0 otherwise, $i \in I$, $t \in P_i^H$,

we formulate the *2OPP-UL* as follow :

$$\text{(2OPP-UL)} \qquad \sum_{r \in P_i^W} x_{ir} = 1 \qquad i \in I \tag{5}$$

$$\sum_{t \in P_i^H} y_{it} = 1 \qquad i \in I \tag{6}$$

$$\sum_{r \in P_i^W(\bar{r})} x_{ir} + \sum_{r \in P_j^W(\bar{r})} x_{jr} + \sum_{t \in P_i^H(\bar{t})} y_{it} + \sum_{t \in P_j^H(\bar{t})} y_{jt} \leq 3 \qquad i \in I, j \in I_i^{=}, \bar{r} \in P^W, \bar{t} \in P^H \tag{7}$$

$$y_{i\bar{t}} + \sum_{r \in P_i^W(\bar{r})} x_{ir} + \sum_{r \in P_j^W(\bar{r})} x_{jr} + \sum_{\substack{t \in P_j^H(\bar{t}) \\ t \leq \bar{t}+h_i-1}} y_{jt} \leq 3 \qquad i \in I, j \in I_i^{<}, \bar{r} \in P^W, \bar{t} \in P_i^H \tag{8}$$

$$x_{ir} \in \{0,1\} \qquad r \in P_i^W, i \in I \tag{9}$$

$$y_{it} \in \{0,1\} \qquad t \in P_i^H, i \in I \tag{10}$$

Constraints (5) and (6) impose a (x,y) coordinate for each item. Constraints (7) insure that no pair of items covers the same (x,y) coordinate. In particular, if items i and j overlap, it is possible to find a coordinate (x,y) such that the left hand side of constraint (7) equals 4. The unloading constraints are imposed through (8). If item i is to be delivered after j and both items cover the same x coordinate (i.e., the summation of the two terms involving the x-coordinates equals 2), then item j cannot be below i (i.e., the summation of the two terms involving the y-coordinates must be smaller than or equal to 1). The number of constraints of types (7) and (8) is *pseudo*-polynomial and grows very quickly with W and H, thus leading to a model which is not really useful in practice. Accordingly, the next section will describe an alternative approach to tackle this problem.

3 Problem-solving methodology

In the following, we first describe preprocessing techniques aimed at reducing the number of solutions to be examined. Then, instead of solving directly the 2OPP-UL model, a two-phase approach similar to the one reported in [10] for the 2SPP is applied. Basically, at each node of our branch-and-cut algorithm, a modified 1CBP (formulated as a *binary program*) is solved to find y coordinates. Then, the corresponding x-check problem is addressed in the second phase.

3.1 Preprocessing

Some preprocessing is first applied to tighten the problem. In this process, infeasibility might be uncovered. The algorithmic flow is the following:

1. Apply height position restrictions

2. Apply lifting
3. Calculate lower bounds SPP, L_2, L_3^H, L_3^W
4. From bottom to top-fill do
   (a) Apply precedence relations
   (b) Apply normal pattern dominance
   (c) Apply normal pattern removal
5. Select best fill
6. Calculate L_4^H, L_4^W

In step 1, the minimum and maximum possible height positions for each item are determined (section 5.1). This is followed by lifting procedures aimed at artificially increasing the item sizes (section 5.2). Then, some lower bounds are calculated (section 6). Different ways of filling the bin, either from the bottom, from the top, or from the bottom and top are considered (section 5.5), while applying normal pattern dominance and normal pattern removal to reduce as much as possible the set of normal patterns (section 5.4). The filling associated with the smallest set of normal patterns is then selected. At the end, two sophisticated bounds, which exploit the current set of normal patterns, are applied (section 6) .

At each step, infeasibility tests are performed to detect if the current set of normal patterns for any given item becomes empty or if the minimum and maximum height positions of any given item are not coherent with the dimensions of the bin (e.g., if the minimum height position of item i must be larger than $H - h_i$).

3.2 Modified 1CBP

The modified 1CBP has the same variables than the 2OPP-UL and is defined as follow :

$$
\begin{aligned}
(1CBP\ ) \qquad & \sum_{t \in P_i^H} y_{it} = 1 && i \in I && (11)\\
& \sum_{i \in I} \sum_{\bar{t} \in P_i^H(t)} w_i y_{i\bar{t}} \ \leq W && t \in P^H && (12)\\
& \sum_{y_{it} \in S} y_{it} \leq |S| - 1 && S \in \mathcal{S} && (13)\\
& y_{it} \in \{0,1\} && i \in I,\ t \in P_i^H && (14)
\end{aligned}
$$

Basically, this is a standard 1CBP but with the number of bins fixed at H. Constraints (11) state that the first slice of each item must be assigned to a bin. Constraints (12) ensure that the capacity W of each bin is satisfied. Finally, constraints (13) correspond to *Benders' feasibility cuts*. The set $\mathcal{S}$ contains all subsets $S = \{y_{it}\} \in \mathcal{S}$ such that an infeasibility occurs in the x-check problem when all variables in S are equal to 1. Given that set $\mathcal{S}$ can be very large, these constraints are relaxed in the initial formulation of the problem.

Within the branch-and-cut algorithm, the x-check problem is called each time a binary solution to the current 1CBP linear relaxation is obtained. If the x-check problem is feasible, then we have a feasible solution to the 2OPP-UL. Otherwise, a Benders' cut of type (13) is added to the 1CBP model. Such an inequality is very weak because it typically removes a single (x, y) coordinate from the solution space. However, stronger cuts can be generated by applying an idea similar to the one in [10]. Let S be the subset of variables equal to 1 in a solution of the 1CBP for which the corresponding x-check problem is infeasible. Let $S' \subset S$ be such that the corresponding x-check problem is also infeasible. Then, any modification to the y coordinates of the items in $S \backslash S'$ induces a cut that subsumes the cut generated by S. Based on this observation, the set S is reduced as much as possible to obtain a minimal infeasible set (MIS) [8].

To find a MIS, the variables in $S = \{y_{it}\}$ are considered sequentially from the one with the smallest height position to the one with the largest height position. We remove the chosen variable from S and solve the x-check problem for the reduced set. If it is feasible, the variable is put back in S, otherwise a smaller set that is still infeasible is found. We repeat the procedure until all variables have been considered. Although the order of removal of the variables from set S can lead to different MISs, a single order was considered here (no significant improvement was observed in preliminary tests using multiple MISs). Note also that the resulting inequality cannot be lifted as in [10], because the infeasibility of the x-check problem can come from the unloading constraints.

Apart from the Bender's cuts, additional cuts are generated when the solution of the current 1CBP linear relaxation is fractional. These inequalities are described in Section 4.

3.3 x-check problem

The model for the x-check problem is obtained by replacing the y_{it} variables in the 2OPP-UL model by their optimal values in the 1CBP problem. Unfortunately, the resulting model is still too large to be solved in practice with CPLEX. The x-check problem is thus addressed with an enumerative tree search-based approach, which proves to be very efficient. Let $\bar{y}_i$ be the y-coordinate of item i in the 1CBP solution. The algorithm starts by filling the bottom of the bin and then moves up progressively. At the root node, no item is in the bin. Then, for each item i such that

$\bar{y}_i = 0$ a child node is created with i at coordinate $(0, 0)$ in the bin. For subsequent nodes, let (x_{cur}, y_{cur}) be the current lowest leftmost position. For each item i such that $\bar{y}_i = y_{cur}$, a child node is created with item i at (x_{cur}, y_{cur}) if the item fits without overlap and if the unloading constraints are satisfied. An empty node is also created where coordinates (x_{cur}, y_{cur}) to $(x_{cur} + e_x, y_{cur} + e_y)$ are forbidden with $e_x = \min_{i \in I'}\{\min_{r \in P_i^W}\{r : r > x_{cur}\}\}$, $e_y = \min_{i \in I'}\{\bar{y}_i : \bar{y}_i > y_{cur}\}$ and I' the set of items that are not yet in the bin.

A node is fathomed when the needed empty space is larger than the available empty space. This simple scheme proved the feasibility or infeasibility of every x-check problem after generating only a few nodes in the branching tree.

4 Inequalities

Before describing our inequalities, we need to introduce the concept of *conflicting items* or, equivalently, *conflicting variables*. Let Y be the set of variables in our 1CBP. Let also y_{it_i} and y_{jt_j} be two variables in Y. Then, these variables are *conflicting* if they cannot cover a common x-coordinate without overlapping or violating the unloading constraints. Figure 2 b) shows an example where the variables associated with items 3 and 4 are conflicting. Clearly, at the time of delivery, item 4 needs to be moved to reach item 3, thus violating the unloading constraints. Two variables y_{it_i} and y_{jt_j} are conflicting if they satisfy one of these two conditions:

- $seq_i < seq_j$ and $t_i < t_j + h_j$,
- $seq_i = seq_j$ and $t_i < t_j + h_j$ and $t_j < t_i + h_i$,

Two variables y_{it} are also considered to be conflicting if they are associated with the same item $i \in I$. The reason is that only one of these variables should be equal to 1, see equation (11).

A subset $C \subseteq Y$ is *mutually conflicting* if for every pair of variables $y_{it_i}, y_{jt_j} \in C$, y_{it_i} is conflicting with y_{jt_j}. For example, the set $C = \{y_{1t_1}, y_{2t_2}, y_{3t_3}, y_{4t_4}\}$ is mutually conflicting in both Figures 2 a) and b). In the case of Figure 2 b), the solution is not UL-feasible because the summation over the widths of the items in set C exceeds W. Conversely, the same summation is equal to W in Figure 2 a), which is a UL-feasible solution. This observation can be generalized through the following:

Proposition 4.1 *In any feasible 2OPP-UL solution, the summation over the widths of the items in every mutually conflicting set is smaller than or equal to W.*

Proof. By contradiction, let us assume that we have a set $C = \{y_{i_1t_1}, y_{i_2t_2}, ..., y_{i_kt_k}\}$ of mutually conflicting variables in a feasible *2OPP-UL* solution such that $seq_{i_1} \geq$

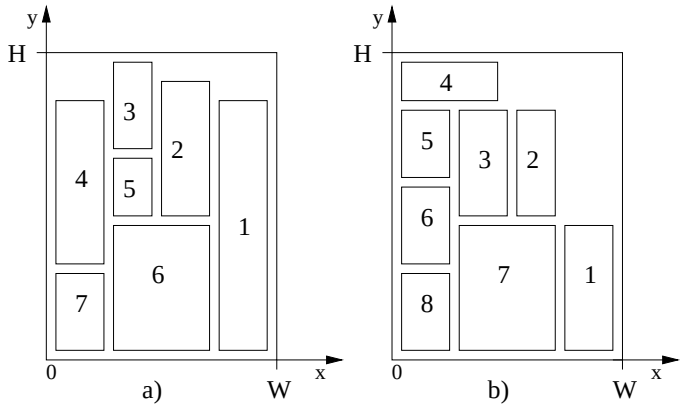

Figure 2: Mutually conflicting sets

$seq_{i_2} \geq ... \geq seq_{i_k}$ and the summation over the widths of the corresponding items is strictly larger than W (for the sake of simplicity, it is assumed that the i_j's are all different). Since variable $y_{i_1 t_1}$ is conflicting with $y_{i_2 t_2}$, the required width for the two corresponding items is $w_{i_1} + w_{i_2}$. The same applies to $y_{i_3 t_3}$ which is conflicting with $y_{i_1 t_1}$ and $y_{i_2 t_2}$, thus requiring a width of $w_{i_1} + w_{i_2} + w_{i_3}$. Following the same line of reasoning, the required width is $\sum_{j=1}^{k} w_{i_j}$ which is strictly larger than W (by hypothesis). This observation contradicts the fact that the 2OPP-UL solution is feasible. This proof can be easily extended to the case where the i_j's are not all different.

These observations led to the conflict inequalities which are descried in the next section.

4.1 Conflict inequalities

Let us consider the following proposition:

Proposition 4.2 *In every feasible 2OPP-UL solution, we have:*

$$\sum_{y_{it} \in C} w_i y_{it} \leq W \qquad \forall C \in \mathcal{Y} \text{ such that } \sum_{i \in I(C)} w_i > W, \tag{15}$$

where $\mathcal{Y}$ is the set of all mutually conflicting subsets of Y. Inequalities (15) are classical *knapsack constraints*. For any given mutually conflicting set C, we can instead use the following *cover inequality*:

$$\sum_{y_{it} \in C} y_{it} \leq |I(C)| - 1 \tag{16}$$

where $I(C)$ is the set of items associated with the variables in C. One can observe that inequality (16) is a special case of the classical *GUB cover inequality*. We recall that C is a GUB cover if $\sum_{i \in I(C)} w_i > W$ and if no two variables are associated with the same item. Set C can be transformed into a GUB cover by removing all variables but one that belong to the same item. In addition, set C is a minimal GUB cover if no proper subset of C is a GUB cover. Assuming that C is both mutually conflicting and a minimal GUB cover, we can consider an extension E such that $C \subseteq E$, E is mutually conflicting and $\max_{i \in I(C)}\{w_i\} \leq \min_{i \in I(E) \setminus I(C)}\{w_i\}$. Inequality (16) can then be extended and dominated by:

$$\sum_{y_{it} \in E} y_{it} \leq |I(C)| - 1 \tag{17}$$

We prefer here to use a special type of *GUB lifted cover inequality* because a single weight w_i is associated with the y_{it} variables, for any given item i. Let C be a minimal GUB cover and E be a mutually conflicting set such that $C \subseteq E$. Then we can derive the following inequality:

$$\sum_{y_{it} \in E, i \in I(C)} y_{it} + \sum_{y_{it} \in E, i \notin I(C)} \alpha_i y_{it} \leq |I(C)| - 1 \tag{18}$$

We will refer to the α_i's as the *lifting coefficients*. They can be computed by solving a series of knapsack problems [17]. We will refer to these inequalities as *conflict inequalities*.

4.2 Max flow inequalities

Another type of inequality is obtained by considering any mutually conflicting set C such that $\sum_{i \in I(C)} w_i \leq W$. For example, let us assume that items 1, 5 and 6 in Figure 3 are fixed at their current position. Then, we can identify the feasible areas for all other items. For example, the items to be delivered after 1 but before 5 must be in areas A_1, B_5, B_6 or E.

Let $t_i^{min} = \min_{y_{jt} \in C}\{t : j = i\}$ and $t_i^{max} = \max_{y_{jt} \in C}\{t : j = i\}$. A maximum flow problem, defined on an appropriate network, can be solved to know if the remaining free items can lead to a feasible solution given that the items associated with set C are fixed at their current position. More precisely, the network would be as follow:

- a source node s,
- a destination node t,
- for each $i \in I(C)$, nodes A_i and B_i, where A_i stands for the area immediately under item i and B_i for the area immediately above it,

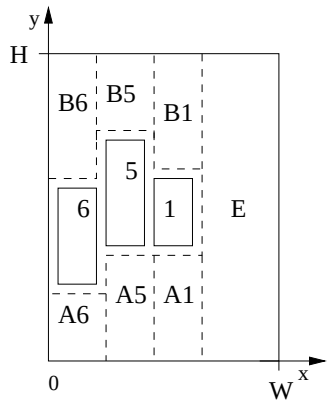

Figure 3: Areas associated with mutually conflicting sets

- a node E for the remaining area,
- for each $i \in I \backslash I(C)$, a node k_i,
- for each $i \in I(C)$, an arc from s to A_i with a capacity equal to $t_i^{max} w_i$,
- for each $i \in I(C)$, an arc from s to B_i with a capacity equal to $(H - t_i^{min} - h_i) w_i$,
- an arc from s to E with a capacity equal to $H(W - \sum_{i \in I(C)} w_i)$,
- for each $i \in I(C)$ and $j \in I \backslash I(C)$, an arc from A_i to k_j with a capacity equal to $\min\{w_i, w_j\} h_j$ if $h_j \leq t_i^{max}$ and $seq_i \leq seq_j$,
- for each $i \in I(C)$ and $j \in I \backslash I(C)$, an arc from B_i to k_j with a capacity equal to $\min\{w_i, w_j\} h_j$ if $h_j \leq H - t_i^{min} - h_i$ and $seq_i \geq seq_j$,
- for each $i \in I \backslash I(C)$, an arc from E to k_i with a capacity equal to $min\{w_i, W - \sum_{i \in I(C)} w_i\} h_i$
- for each $i \in I \backslash I(C)$, an arc from k_i to t with a capacity equal to $h_i w_i$.

Let $maxflow(C)$ be the maximum flow from s to t in this network. We then have :

Proposition 4.3 *If $maxflow(C)$ is strictly smaller than $\sum_{i \in I \backslash I(C)} h_i w_i$ then the mutually conflicting set C leads to an infeasible solution. The partial solution can be forbidden through the following constraints:*

$$\sum_{y_{it} \in C} y_{it} \leq |I(C)| - 1 \qquad \forall C \in \mathcal{Y} \text{ such that } maxflow(C) < \sum_{i \in I \backslash I(C)} h_i w_i \qquad (19)$$

We will refer to these inequalities as the *max flow inequalities*.

4.3 Separation procedures

In the previous section, we introduced some properties of feasible *2OPP-UL* solutions. In many cases, the 1CBP solution given to the x-check problem will not satisfy these properties. It is thus useful to look for a violated property before calling the x-check problem. The inequality generated from a violated property will also be stronger as it will focus on what truly makes the solution infeasible. Accordingly, at each node of our branch-and-cut algorithm, we first check if the current solution violates inequalities (15) or (19).

A simple heuristic and an exact method have been developed to separate violated inequalities. Given that these inequalities are based on mutually conflicting sets, the first algorithm generates conflicting sets in a heuristic way, while the second one generates all of them.

Starting with an empty set, the heuristic first adds a randomly chosen variable to this set. Then, at each iteration, a new randomly chosen variable in considered and added to the set if it is conflicting with the other variables already in the set. At the end, we check if the resulting set violates a conflict inequality. If this is the case, the mutually conflicting set is kept. The method is called n times, where n is the number of items.

The exact method generates all mutually conflicting sets using a tree search-based algorithm. First, the y_{it} variables are sorted in ascending order based on item index i and height position t. Let $Ord(y_{it})$ be the order of variable y_{it}. At the root node, for which there is no mutually conflicting set, a child node is created for each variable and the associated conflicting set is a singleton with only this variable. In subsequent nodes, a new variable is added to the set only if it is of higher order than any variable in the set and if the variable is in conflict with all variables in the set. Each mutually conflicting set produced in this way is checked to see if it violates an inequality. If it does, the conflicting set is kept.

If the number of variables with a positive value is high, there might just be too many mutually conflicting sets. Based on preliminary experiments, the exact method is called only if this number is smaller than or equal to 1.1 times the number of items n. Otherwise, the heuristic method is used. Note also that each inequality must be violated by a value greater than or equal to 0.2 and that a maximum of 10 inequalities are added at each call.

The resulting inequalities are strengthened before they are added to the model. For a conflict inequality of type (15), a corresponding inequality of type (18) is added as it cuts fractional parts of the feasible domain. To generate an inequality of type (18) from an inequality of type (15), set C is first transformed into a minimal GUB cover by removing the variables with the largest weights. Then, variables with a positive value that are in conflict with all variables already in C are added to the set (considering these variables first proved to generate stronger inequalities). Then, the same procedure is applied to the remaining variables. We then find the

lifting coefficients for the resulting extension by solving a series of *knapsack problems* (see [21] for details). As previously mentioned, only one extension is created by considering the variables to be added to C in random order.

For a max-flow inequality (19), the extension $E = \{y_{it} \in Y : i \in I(C) \text{ and } t_i^{min} \leq t \leq t_i^{max}\}$ is used. That is, only the variables associated with the items that are already in C and such that t is between t_i^{min} and t_i^{max} are considered.

5 Preprocessing

In this section, the various procedures that are applied during the initial preprocessing phase are described.

5.1 Height position restrictions

If the first items to be delivered are at the bottom of the bin, the corresponding solution is likely to be infeasible due to the unloading constraints. Figure 4 a) shows that the items to be delivered before item i must necessarily be in areas B and C. That is, if they are in area A, item i must be moved to get to these items, which is forbidden. Similarly, the items to be delivered after i must be in areas A and C. Figure 4 b) shows that if item i is at some height position $y_i \in [0, H]$, the unloading constraints might force some items outside of the bin (we recall that the height position of an item is defined as the height position of its bottom-left corner).

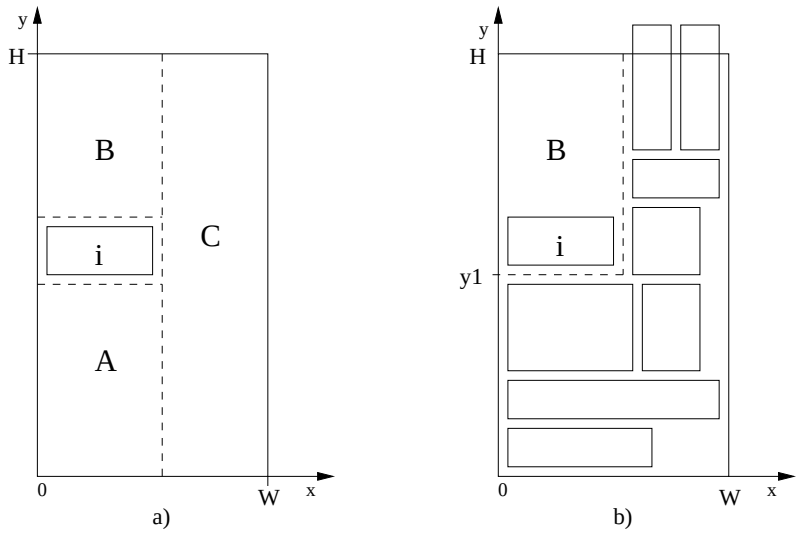

Figure 4: Height position restrictions

Motivated by some lower bounds reported in [9], we define y_i^{min} and y_i^{max} as the minimal and maximal height positions of item $i \in I$:

$$y_i^{min,1} = \max\left\{\left\lceil\frac{\Sigma_{(i)}^{>,W-w_i}}{W}\right\rceil, \max_{j\in I_i^{>,W-w_i}}\{h_j\}\right\} \tag{20}$$

$$y_i^{min,2} = \left\lceil\frac{[\Sigma_{(i)}^{>} - H(W-w_i) + \sum_{j\in I_i^{\leq,w_i}} h_j(w_j-w_i)]^+}{w_i}\right\rceil \tag{21}$$

$$y_i^{max,1} = H - \max\left\{\left\lceil\frac{\Sigma_{(i)}^{<,W-w_i}}{W}\right\rceil, \max_{j\in I_i^{<,W-w_i}}\{h_j\}\right\} \tag{22}$$

$$y_i^{max,2} = H - \left\lceil\frac{[\Sigma_{(i)}^{<} - H(W-w_i) + \sum_{j\in I_i^{\geq,w_i}} h_j(w_j-w_i)]^+}{w_i}\right\rceil \tag{23}$$

$$y_i^{min} = \max\{y_i^{min,1}, y_i^{min,2}\} \tag{24}$$

$$y_i^{max} = \min\{y_i^{max,1}, y_i^{max,2}\} \tag{25}$$

Equations (20) and (21) are obtained by considering only the items that must lie below some item i. More precisely, the value $y_i^{min,1}$ corresponds to the maximum between a continuous relaxation over all items that must lie below i and the item of maximum height that must also lie below i. The value $y_i^{min,2}$ is the height position obtained by pushing item i downward as long as the bottom of the bin is not reached or an item in area A, by moving to area C, does not force an item outside of the bin. Similar values in equations (22) and (23) are obtained by considering the items that must lie above i.

The obtained values y_i^{min} and y_i^{max} in equations (24) and (25) can be improved in two different ways. First, by considering items that cannot be beside i. Let us suppose, for example, that item j is to be delivered after i and that $w_i + w_j > W$. In this case, $y_j^{min} + h_j$ is clearly a lower bound on y_i^{min}. The second improvement can be obtained by considering only normal patterns. In fact, the value y_i^{min} should be the minimum height position of item i only if it corresponds to a normal pattern (i.e., the bottom of item i touches the top of another item or the bottom of the bin). If this is not the case, it can be reset to the minimum height position larger than y_i^{min} which corresponds to a normal pattern. The same line of reasoning can be applied to y_i^{max}. We thus obtain:

$$y_i^{min} = \max\{y_i^{min,1}, y_i^{min,2}, \max_{j\in I_i^{>,W-w_i}}\{y_j^{min} + h_j\}, \min\{t : t \in P_i^H\}\} \tag{26}$$

$$y_i^{max} = \min\{y_i^{max,1}, y_i^{max,2}, \min_{j \in I_i^{<,W-w_i}} \{y_j^{max} - h_i\},\ \max\{t : t \in P_i^H\}\} \tag{27}$$

5.2 Lifting

As shown by previous authors [2, 6], if we do not find any combination of widths such that their sum is equal to W, then it is possible to reduce the width W as some unused space will remain. The same idea can be applied to the height H. The values W^* and H^* in equations (28) and (29) can be obtained through dynamic programming. At the end, we reset W to W^* and H to H^*.

$$W^* = \max\{\sum_{i \in I} z_i w_i : \sum_{i \in I} z_i w_i \leq W,\ z_i \in \{0,1\}, i \in I\} \tag{28}$$

$$H^* = \max\{\sum_{i \in I} z_i h_i : \sum_{i \in I} z_i h_i \leq H,\ z_i \in \{0,1\}, i \in I\} \tag{29}$$

A similar procedure can then be applied to the item sizes. The increase in width and height of item $i \in I$ is given by the following equations:

$$\Delta w_i = W - \max\{w_i - \sum_{j \in I \setminus \{i\}} z_j w_j : \sum_{j \in I \setminus \{i\}} z_j w_j \leq W - w_i,\ z_j \in \{0,1\}, j \in I \setminus \{i\}\} \tag{30}$$

$$\Delta h_i = H - \max\{h_i - \sum_{j \in I \setminus \{i\}} z_j h_j : \sum_{j \in I \setminus \{i\}} z_j h_j \leq H - h_i,\ z_j \in \{0,1\}, j \in I \setminus \{i\}\} \tag{31}$$

Clearly, not all items can be found at a given height position $t \in P^H$, either because it is not part of their normal patterns or because y_i^{min} is too large or y_i^{max} is too small for some item $i \in I$. Thus, it might well happen that the width occupied by the set of remaining items I^t at height position t can only be smaller than W. In this case, we can further reduce the width W to W_t :

$$W_t = \max\{\sum_{i \in I^t} z_i w_i : \sum_{i \in I^t} z_i w_i \leq W : z_i \in \{0,1\}, i \in I^t\} \qquad t \in P^H \tag{32}$$

We can extend this line of reasoning to individual items. Let us assume that item i is at height position t. Then, we can consider the other items one by one and use the previously described maximum flow algorithm to detect if it can be beside item i or not (see section 4.2). After removing all items that cannot be beside i, there might always be an empty space beside i. In this case, we can increase the width of w_i to w_{it} to cover the empty space. The w_{it} values can be obtained by solving (in pseudo-polynomial time) subset sum-like problems using dynamic programming.

5.3 Precedence relations

Let us consider items i and j such that i is delivered after j (i.e., $seq_i > seq_j$). We want to know if there is a feasible solution where height position t_j is smaller than t_i. If there is no such solution, we have a precedence relation between items i and j, because the height position of i must be smaller than or equal to the the height position of j for a solution to be feasible.

Formally, we identify two different types of precedence relations : a weak precedence relation where $t_i \leq t_j$ and a strong precedence relation where $t_i + h_i \leq t_j$. Furthermore, we consider a side-by-side relation where $t_i \leq t_j < t_i + h_i$ or $t_j \leq t_i < t_j + h_j$ always holds. The sets of weak precedence, strong precedence and side-by-side relation associated with item i are denoted $H(i)^-$, $H(i)^+$ and $H(i)^=$, respectively.

In the following, we present conditions for such precedence relations to occur between items i and j.

Condition 1 : If $y_i^{max} + h_i \leq y_j^{min}$, then $j \in H^+(i)$.

Condition 2 : If $y_i^{max} \leq y_j^{min}$ then $j \in H^-(i)$.

Condition 3 : If $maxflow(C) < \sum_{k \in I \setminus \{i,j\}} h_k w_k$ for all $t_i \in P_i^H$ and $t_j \in P_j^H$ such that $t_j < t_i$ and $C = \{y_{it_i}, y_{jt_j}\}$ is a mutually conflicting set, then $j \in H^-(i)$.

Condition 4 : If $w_i + w_j > W$, then $j \in H^+(i)$..

Condition 5 : If $maxflow(C) < \sum_{k \in I \setminus \{i,j\}} h_k w_k$ for all $t_i \in P_i^H$ and $t_j \in P_j^H$ such that $t_j < t_i + h_i$ and $C = \{y_{it_i}, y_{jt_j}\}$ is a mutually conflicting set, then $j \in H^+(i)$.

Condition 6 : If $maxflow(C) < \sum_{k \in I \setminus \{i,j\}} h_k w_k$ for all $t_i \in P_i^H$ and $t_j \in P_j^H$ such that $t_i > t_j$ or $t_i < t_j$ and $C = \{y_{it_i}, y_{jt_j}\}$ is a mutually conflicting set, then $j \in H^=(i)$.

From Conditions 1 to 3, we obtain inequalities (33) where the height position of item j is forced to be greater than or equal to the height position of item i, when i is weakly preceding j. From Conditions 4 and 5, we get inequalities (34) where the height position of item j is forced to be greater than or equal to the height position of item i plus its height h_i, when i is strongly preceding j. Finally, Condition 6 leads to inequalities (35) where items i and j are forced to be side-by-side.

$$y_{it} \leq \sum_{\substack{\bar{t} \in P_j^H \\ \bar{t} \geq t}} y_{j\bar{t}} \qquad i \in I, j \in H^-(i),\ t \in P_i^H \tag{33}$$

$$y_{it} \leq \sum_{\substack{\bar{t} \in P_j^H \\ \bar{t} \geq t + h_i}} y_{j\bar{t}} \qquad i \in I, j \in H^+(i),\ t \in P_i^H \tag{34}$$

$$y_{it} \leq \sum_{\bar{t} \in P_j^H(t)} y_{j\bar{t}} \qquad i \in I, j \in H^{=}(i),\ t \in P_i^H, \tag{35}$$

5.4 Reconsidering the normal patterns

In this section, we describe two different ways of reducing the initial set of normal patterns.

5.4.1 Normal pattern dominance

It is possible to reduce the set of normal patterns by considering a dominance relation among the patterns associated with a given item. Let i be an item and $t_i \in P_i^H$ some height position corresponding to a pattern. If i is at height position t_i and nothing fits over i, then nothing will fit over i at any height position $t > t_i$. Let $t_i^{min} \in P_i^H$ be the lowest height position such that nothing fits over item i. Then every height position $t \in P_i^H$ such that $t \geq t_i^{min}$ can be removed. This type of relation is called *Normal Pattern Dominance*.

5.4.2 Normal pattern reduction

Previously, we introduced a feasibility test based on the solution of a maximum flow problem to know if a partial solution based on a mutually conflicting set of items leads to an infeasible solution. This test can also be used to prove the feasibility of a normal pattern. Suppose that item i is at height position $t_i \in P_i^H$. If all mutually conflicting sets that include t_i fail the feasibility test then t_i can be removed from P_i^H. But since there might be a huge number of mutually conflicting sets, a more viable approach is the following. For any given item i and height position $t_i \in P_i^H$, t_i can be removed from P_i^H if there is an item j such that for all $t_j \in P_j^H$ and mutually conflicting set $C = \{y_{it_i}, y_{jt_j}\}$, $maxflow(C)$ is smaller than $\sum_{k \in I \setminus \{i,j\}} h_k w_k$. The value t_i is then added to a set INF_i that contains infeasible height positions for item i.

By using the previously defined y_i^{min} values, y_i^{max} values and INF_i sets, an improved procedure can be designed to calculate a reduced set of normal patterns. The pseudo-code of this procedure is found in Algorithm 1 for the typical case where the bin is filled from the bottom to the top.

The normal patterns are generated by a call to the method *Calculate Normal Patterns*. Three sets *Cur*, *Prev* and *Glob* are first initialized with position 0 which is the bottom of the bin. Set *Prev* contains all normal patterns generated up to (but not including) the current customer j. Note that *Cur* is reset to *Prev* before generating the patterns of each item associated with customer j. The patterns

for item $i \in I_j$ are generated recursively by the method *Generate Patterns* if the current customer has more than one item. When the set of all patterns associated with item i has been produced, the global variable P_i^H is assigned with this set. The newly generated patterns are also added to set *Glob*. When all items of the current customer are done, *Prev* is set to *Glob* just before handling the next customer. The methods *Generate Patterns* and *Generate Patterns Recursively* consider all possible ordered combinations of items in set I and make sure that the corresponding patterns are feasible normal patterns.

The complexity of this algorithm is $O(Hn(o-1)!)$ where H is the height of the bin, n is the number of items and $o = \max_{j=1,\ldots,R} |I_j|$. The algorithm performs well when the number of items per customer is small. However, if $(o-1)!$ is larger than n, it is better to use the dynamic algorithm based on equations (1) and (2), as reported in [18], which is in the order of $O(Hn^2)$.

Algorithm 1 Calculate Normal Patterns

Require: y_i^{min}, y_i^{max} and INF_i
  $Prev \leftarrow \{0\}$
  $Cur \leftarrow \{0\}$
  $Glob \leftarrow \{0\}$
  for $j = R$ to 1 **do**
    for $i \in I_j$ **do**
      $Cur \leftarrow Prev$
      if $I_j \backslash \{i\} \neq \emptyset$ **then**
        Generate Patterns($I_j \backslash \{i\}$, Cur)
      end if
      $P_i^H \leftarrow \{t \in Cur \backslash INF_i : y_i^{min} \leq t \leq y_i^{max}\}$
      for $t \in P_i^H$ **do**
        $Cur \leftarrow Cur \cup \{t + h_i\}$
      end for
      $Glob \leftarrow Glob \cup Cur$
    end for
    $Prev \leftarrow Glob$
  end for

5.5 Top-fill versus bottom-fill

Typically, a bottom-left approach is used to generate the set of normal patterns (as in Algorithm 1). More precisely, the bottom of every item must be in contact with either the bottom of the bin or the top of another item. Furthermore, the left side of each item must be in contact with the left side of the bin or the right side of another item.

Due to the unloading constraints, solutions to our problem exhibit a special

Algorithm 2 Generate Patterns

Require: I : set of items, P : set of patterns
  $min_y \leftarrow \min_{i \in I} y_i^{min}$
  $max_y \leftarrow \max_{i \in I} y_i^{max}$
  for $t = max_y$ to min_y **do**
    if $t \in P$ **then**
      for $i \in I$ **do**
        Generate Patterns Recursively$(t + h_i,\ I \backslash \{i\},\ P)$
      end for
    end if
  end for

Algorithm 3 Generate Patterns Recursively

Require: t: a starting position, I : set of items, P : set of patterns
  if $t \in P$ **then**
    Exit
  end if
  $P \leftarrow P \cup \{t\}$
  for $i \in I$ **do**
    if $y_i^{min} \leq t \leq y_i^{max}$ and $t \notin INF_i$ **then**
      Generate Patterns Recursively$(t + h_i,\ I \backslash \{i\},\ P)$
    end if
  end for

structure which can be exploited by choosing to fill the bin from the bottom, from the top or from both the bottom and the top (mixed fill), in the hope of reducing the set of normal patterns. With regard to the mixed approach, if the items with delivery order j (customer j) fill the bin from the bottom, then the items with delivery order k (customer k) with $k > j$ also fill the bin from the bottom. Conversely, if the items with delivery order j fill the bin from the top, then the items with delivery order k with $k < j$ also fill the bin from the top.

For a sequence of R customers, there are $R + 1$ ways to choose the cut point between the customers whose items will fill the bin from the bottom and the customers whose items will fill the bin from the top. It is then possible to choose, among these $R + 1$ cut points, the one that leads to the smallest number of normal patterns. Figure 5 presents an example of a mixed fill where the cut point is chosen between customers 3 and 4, that is, items of customers 1, 2 and 3 fill the bin from the top and items of customers 4, 5 and 6 fill the bin from the bottom.

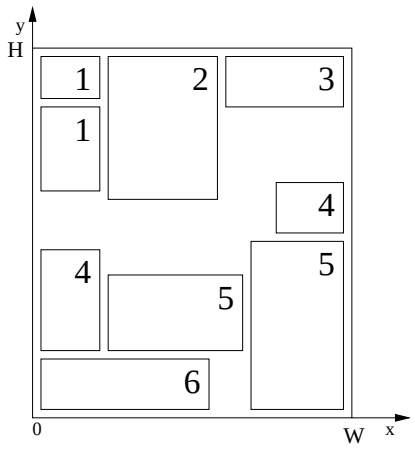

Figure 5: Mixed Fill

Proposition 5.1 *If a solution is feasible for a given cut point, it is feasible for every cut point.*

Proof. Let us assume that a feasible solution s_1 is obtained when the cut point is in position 1, that is, just before the first customer (which corresponds to filling the bin from the bottom, as it is typically done). Then, a feasible solution s_2 for the cut point in position 2, i.e. between customers 1 and 2, is obtained from s_1 by pushing the items of customer 1 up until the top of the bin is reached. Similarly, a feasible solution s_3 for the cut point in position 3, i.e. between customers 2 and 3, is obtained from s_2 by pushing the items of customer 2 up until the top of the bin or the bottom of another item is reached. The result is obtained by repeating this argument until the cut point is in position $R + 1$.

This proposition also means that if a solution is infeasible for a given cut point, it is infeasible for every cut point. Thus, we are guaranteed not to miss any feasible solution. As indicated in the demonstration of the above proposition, there might also be a gap between the top-filled and bottom-filled items. Inspired by the work in [6], a constraint is associated with each item and each height position to force down (up) a top-filled item so that its bottom (top) touches another item which must be delivered after (before) it. These constraints are the following:

$$y_{it} \leq \sum_{j=seq_i}^{R} \sum_{\substack{k \in I_j \\ k \neq j}} y_{k(t-h_k)} \qquad t \in P_i^H,\ i \in I \text{ and } i \text{ is bottom-filled} \tag{36}$$

$$y_{it} \leq \sum_{j=1}^{seq_i} \sum_{\substack{k \in I_j \\ k \neq j}} y_{k(t+h_k)} \qquad t \in P_i^H,\ i \in I \text{ and } i \text{ is top-filled} \tag{37}$$

6 Lower bounds

In this section we present some lower bounds on the required height of the bin.

6.1 Simple Lower Bounds

First, we can use the lower bound for the *SPP* reported in [23]. This so-called continuous lower bound, denoted L_1, calculates the number of strips of width W and height 1 needed to accommodate all items.

$$L_1 = \left\lceil \frac{\sum_{i \in I} w_i h_i}{W} \right\rceil \tag{38}$$

From the definition of y_i^{min} and y_i^{max} in equations (26) and (27), the following lower bound can also be derived:

$$L_2 = \max_{i \in P} \{y_i^{min} + h_i + (H - y_i^{max})\} \tag{39}$$

It is worth noting that this bound improves upon the one reported in [9], through the addition of the third term in the numerator of $y_i^{min,2}$ and $y_i^{max,2}$ in equations (21) and (23) and through the consideration of the third and fourth components in the definition of y_i^{min} and y_i^{max} in equations (26) and (27).

6.2 Lower Bounds based on the Cutting Stock Problem

The bounds presented in this section are based on the Gilmore-Gomory formulation of the *Cutting Stock Problem* (CSP). We define a cutting pattern as a subset of items $I' \subset I$. A pattern is said to be h-feasible if $\sum_{i \in I'} h_i \leq H$ and w-feasible if $\sum_{i \in I'} w_i \leq W$. Let $\mathcal{K}^H$ and $\mathcal{K}^W$ be the sets of all h-feasible and w-feasible cutting patterns. Let also the variable v_k be equal to the number of times cutting pattern k appears in a solution with $a_{ik} = 1$ if item i is in cutting pattern k, 0 otherwise. The CSP based on w-feasible cutting patterns (CSP^W) can be formulated as follow :

$$(CSP^W) \qquad \min \sum_{k \in \mathcal{K}^W} v_k \tag{40}$$

$$\sum_{k \in \mathcal{K}^W} a_{ik} v_k \geq h_i \qquad i \in I \tag{41}$$

$$v_k \geq 0 \text{ and integer} \qquad k \in \mathcal{K}^W \tag{42}$$

The optimal solution value of (CSP^W) provides a lower bound on the required height of the bin. Thus, an instance is infeasible if the obtained value is larger than H. A CSP^H can be defined similarly to obtain a lower bound on the required width. We will call these two lower bounds L_3^H and L_3^W, respectively.

The CSP is famous for the strength of its linear relaxation and it is often the case that the round up value of the linear relaxation is optimal. Column (cutting pattern) generation is typically used to solve this problem and it is well known that the pricing subproblem is a knapsack problem (KP), where the item values come from the dual variables. We refer to [4] for more details on this issue in the context of the 2OPP. A column of negative reduced cost generated at a given width or height position often corresponds to subsets of items that cannot be side by side or one over the other because the corresponding KP does not account for any unloading structure. Clearly, integrating some unloading structure can only improve L_3^W and L_3^H.

For example, we can improve L_3^H to obtain L_4^H by considering only the items $i \in I$ that can be at some height position, based on the current set P^H and the values y_i^{min} and y_i^{max}. Although we might still end up with infeasible columns, this approach has proven to be very effective.

We can also improve L_3^W to obtain L_4^W. Here, we must generate a feasible *stack* of items of negative reduced cost at some width position, as illustrated in Figure 6. For generating stacks, an algorithm similar in spirit to the one used for generating normal patterns is applied (see section 5.4). Let $V(j, h)$ be the sum of the item values over all items that are below position h in an optimal stack for customers with a delivery order between j and R. Also, let S be a subset of items associated with

a given customer and consider that function $\delta(S, h)$ returns the minimum height required by the items in S, if they are all below position h (if this is not feasible then $\delta(s, h) = -\infty$). To find a stack of negative reduced cost, one needs to solve the following recursion for $V(1, H)$:

$$V(j,h) = \max\{\ V(j+1,h), \\ \max_{s \subseteq I_j}\{\ \sum_{i \in S} \lambda_i + V(j+1, \delta(S,h))\ \}\ \}$$

It should be noted that $V(j, h) = \infty$ for $j > R$ in this recursion.

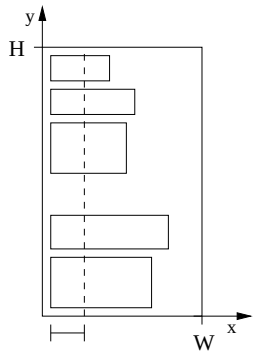

Figure 6: Stacking

If either L_4^H or L_4^W is larger than H or W, the items cannot fit within an area of size HW, but the value obtained is not necessarily a lower bound on the required height or width of the bin. For example, if $L_4^H = H + \Delta H$ with $\Delta H > 1$, the items might well fit within an area of size (H+1)W, because increasing H by only one unit typically leads to many new feasible height and width positions in sets P^H and P^W.

We end this section by observing that the lower bounds for the Strip Packing Problem or the Two-Dimensional Bin Packing Problem are also valid lower bounds for the 2OPP-UL. We refer the reader to [2, 6] for an exhaustive list of lower bounds for the SPP.

7 Computational results

In this section, we first analyze the lower bounds presented in section 6. Then, the impact of the preprocessing is quantified. Finally, our branch-and-cut algorithm is compared with the algorithm in [20], where the authors implemented the tree search-based enumeration scheme proposed in [23] with additional fathoming criteria. This is the best know exact algorithm for solving packing problems with

Type	# Items per cust.	Vertical		Horizontal		Homogeneous	
		Height	Width	Height	Width	Height	Width
1	1	1	1	1	1	1	1
2	[1,2]	[.4H,.9H]	[.1W,.2W]	[.1H,.2H]	[.4W,.9W]	[.2H,.5H]	[.2W,.5W]
3	[1,3]	[.3H,.8H]	[.1W,.2W]	[.1H,.2H]	[.3W,.8W]	[.2H,.4H]	[.2W,.4W]
4	[1,4]	[.2H,.7H]	[.1W,.2W]	[.1H,.2H]	[.2W,.7W]	[.1H,.4H]	[.1W,.4W]
5	[1,5]	[.1H,.6H]	[.1W,.2W]	[.1H,.2H]	[.1W,.6W]	[.1H,.3H]	[.1W,.3W]

Table 1: Types of instances

unloading constraints. The authors were kind enough to give us their code for this comparison. Our branch-and-cut algorithm is implemented in C++ and calls Cplex 12.5. The tests were performed on a 2.2 GHz AMD Opteron 275 processor running under Linux.

7.1 Instances

The Two-Dimensional Loading Capacitated Vehicle Routing Problem ($2L-CVRP$) instances reported in [16] were used for testing purposes. As indicated in Table 1, there are 5 different types of instances, with 36 instances of each type, for a total of 180 instances. The height H and width W of the bin are equal to 40 and 20, respectively. The number of items per customer is indicated in column *Items per cust.* In types 2 to 5, each item also has one of three different dimensions, referred to as *vertical*, *horizontal* and *homogeneous*. The exact number of items per customer and dimension values were randomly generated in the intervals shown in Table 1. The largest instances have up to 255 customers, 786 items and a fleet of 51 vehicles.

We also created additional instances by simply modifying the dimensions of the bin, as indicated below. With 180 instances in each class, we end up with a total of 900 different $2L-CVRP$ instances.

- Class 1 : H = 40, W = 20
- Class 2 : H = 32, W = 25
- Class 3 : H = 50, W = 16
- Class 4 : H = 80, W = 14
- Class 5 : H = 130, W = 14

7.2 Lower Bounds

Tables 2, 3 and 4 compare our lower bounds on the packing instances generated from the 180 original $2L-CVRP$ instances in [16]. Note that these bounds are useless for Type 1 because all items have their width and height equal to 1. For this comparison, solutions (sets of routes) were generated for each $2L-CVRP$ instance

with the $ALNS$ heuristic in [25]. For efficiency purposes, some bounds taken from [2, 6, 23] were calculated before inserting a customer into a route. These bounds could detect, in particular, situations where the additional area required by the items of the current customer would lead to infeasibility by exceeding the total area of the bin. Accordingly, lower bound L_1 is automatically satisfied. For each $2L-CVRP$ instance, the $ALNS$ heuristic was run for 20,000 iterations and a maximum of 800 best solutions produced during the search were kept. The total number of routes (or packing instances) in these solutions appears in column *# Instances* of Table 2. The average number of items per packing instance is also shown in column *# Items*. After applying the height position restrictions and the lifting procedure, the lower bounds SPP, L_2, L_2^{COR}, L_3^H and L_3^W were applied on each individual packing instance to detect infeasibility (see the algorithmic flow in section 3.1). In the case of SPP, it should be noted that we really have a collection of lower bounds, including the dual functions in [6] and the so-called L_8 bound in [2]. Also, L_2^{COR} corresponds to the bound reported in the work of Cordeau et al. [9], which is improved by L_2. The value in each entry is the percentage of infeasible packing instances detected by the corresponding lower bound. The CPU time in seconds is only shown for L_3^H and L_3^W, because the other ones are too small.

Table 3 then shows the performance of the preprocessing routines, without the application of L_4^H and L_4^W, and the resulting number of undecided packing instances. Then, the performance of L_4^H and L_4^W, which are applied to the undecided instances at the end of the preprocessing, is reported (see the algorithmic flow in section 3.1).

Overall, we can see that the performance substantially diminishes from the type 2 instances to the type 5 instances, which are clearly the most difficult ones. Lower bounds L_3^H and L_3^W largely outperform the simple bounds SPP, L_2 and L_2^{COR} (with our L_2 bound only slightly better than L_2^{COR} on instances of type 2). The preprocessing routines, as a whole, are quite good, but L_4^H and L_4^W still allow a substantial number of additional infeasible instances to be detected.

Table 4 shows the percentage of packing instances that were found infeasible only by the corresponding lower bound or only by the preprocessing routines. It also reports the percentage of instances where L_2 was better than SPP and L_4^H was better than L_4^W (by detecting infeasibility while the other did not). We can see that L_4^H is worth considering even if it is not as good as L_4^W in Table 2. Note also that the simple bounds are still useful, in spite of the 0.0% value in each entry, because they can detect infeasible instances quickly, that is, before going into the more sophisticated preprocessing routines.

7.3 Impact of preprocessing

For this study, we started with the routes produced by the $ALNS$ heuristic when applied to the five classes of $2L-CVRP$ instances, while keeping a maximum of 800 visited solutions for each instance. The feasibility of the obtained routes (packing

Type	# Instances	# Items	SPP Inf.	L_2^{COR} Inf.	L_2 Inf.	L_3^H Inf.	L_3^H sec	L_3^W Inf.	L_3^W sec
2	84571	8.7	24.8%	2.8%	3.0%	38.5%	0.008	40.3%	0.007
3	81637	11.0	8.6%	1.2%	1.2%	20.0%	0.011	25.5%	0.011
4	81806	13.2	3.3%	0.2%	0.2%	7.5%	0.013	12.9%	0.012
5	70132	19.8	0.0%	0.0%	0.0%	0.1%	0.015	0.3%	0.016

Table 2: Comparison of lower bounds SPP, L_2^{COR}, L_2, L_3^H and L_3^W

Type	# Instances	# Items	Preprocessing Inf.	Preprocessing sec	# Instances	L_4^H Inf.	L_4^H sec	L_4^W Inf.	L_4^W sec
2	84571	8.7	89.3%	0.002	9066	13.3%	0.010	23.9%	0.007
3	81637	11.0	54.3%	0.024	37292	11.1%	0.011	23.1%	0.008
4	81806	13.2	20.4%	0.089	65154	6.1%	0.012	16.4%	0.008
5	70132	19.8	0.2%	0.389	70020	0.2%	0.014	0.7%	0.011

Table 3: Comparison of L_4^H and L_4^W after preprocessing

instances) was then assessed with a simple heuristic reported in [20], which is derived from the classical Bottom-Left heuristic. If a packing instance could be proven feasible, it was discarded. Then, a simple enumeration procedure was applied for 20 seconds and, again, if the feasibility or infeasibility of the packing instance could be proven, it was discarded. A maximum of 8 undecided packing instances were kept for each $2L-CVRP$ instance. At the end, a total of 2,183 packing instances were collected. Clearly, this process led to difficult packing instances, which is exactly what we wanted. The number of instances collected by type and class are shown in Table 5. We note, in particular, that all instances of type 1 were discarded, while only one instance of type 2 was still undecided. Thus, the packing instances of type 1 and 2 are quite easy to solve.

Table 6 shows the average CPU time and number of max flow problems generated during the preprocessing phase (since this number has a significant impact on the CPU time) over each class, based on the 2,183 remaining instances. The average number of items per instance and dimensions of the bin for each class are also shown. Then, Table 7 compares the impact of each individual preprocessing routine. The values shown on each line correspond to the increase in percentage in the number of normal patterns for each class when the corresponding preprocessing routine is removed from the basic implementation (with all routines). In this Table, L stands for lifting, HR for height restrictions, PR for precedence relations, NPD for Normal Pattern dominance, NPR for normal pattern removal and TBF for mixed top,

Type	SPP	L_2^{COR}	L_2	L_3^H	L_3^W	Preprocessing	L_4^H	L_4^W	$L_2 >$ SPP	$L_4^H > L_4^W$
2	0.0%	0.0%	0.0%	0.0%	0.0%	44.9%	0.2%	1.1%	0.5%	0.2%
3	0.0%	0.0%	0.0%	0.0%	0.0%	27.2%	0.6%	5.0%	0.6%	1.3%
4	0.0%	0.0%	0.0%	0.0%	0.0%	10.6%	0.7%	6.2%	0.1%	1.4%
5	0.0%	0.0%	0.0%	0.0%	0.0%	0.1%	0.1%	0.3%	0.0%	0.1%

Table 4: Comparison of different lower bounds

	Type					
Class	1	2	3	4	5	Total
1	0	0	30	198	200	428
2	0	1	128	198	200	527
3	0	0	2	155	206	363
4	0	0	0	187	182	369
5	0	0	11	246	239	496
Total	0	1	171	984	1027	2183

Table 5: Number of instances by class and type

	Class 1	Class 2	Class 3	Class 4	Class 5	All
CPU time (sec.)	0.3	0.1	0.5	1.6	9.5	2.6
# Max flow	16 683.2	9 194.0	27 960.6	77 845.9	313 626.7	93 942.4
# Items	17.6	16.6	18.6	23.9	36.9	23.0
$[H, W]$	[40,20]	[32,25]	[50,16]	[80,14]	[130,14]	

Table 6: Preprocessing with all routines

bottom-fill. Table 7 also includes the average number of maximum flow problems and the CPU time when the corresponding preprocessing routine is removed. The method with the largest impact is clearly TBF. Without it, the number of normal patterns increases by 22% to 57% when compared to the implementation with all routines. The HR method has also a significant impact on the CPU time because its removal substantially increases the number of maximum flow problems to be solved.

7.4 Comparison with another exact algorithm

The results of the comparison between our algorithm and the one reported in [20] on the 2,183 difficult packing instances with unloading constraints are shown in Table 8. Column *# Inst.* is the number of instances of each type in each class. Then, columns *Fea.* and *Inf.* show the number of proven feasible and infeasible instances, within the allowed 1,200 seconds of computation time, while the average CPU time in seconds is shown in column *sec.* Column *Solved* reports the total number of solved instances (either feasible or infeasible).

We observe that our algorithm performs very well in comparison with the one reported by Iori et al. [20]. In particular, we can solve 1088 new instances (while only one previously solved instance was not solved by our algorithm). The computation

Routine	Class 1	Class 2	Class 3	Class 4	Class 5	# Max flow	CPU time (sec.)
L	0.47%	1.28%	0.06%	0.08%	0.17%	84 003.0	2.2
HR	0.02%	0.00%	1.02%	2.10%	4.29%	286 133.6	8.5
PR	0.03%	0.00%	0.00%	0.18%	0.56%	98 421.2	2.8
NPD	9.79%	11.70%	6.98%	1.91%	0.42%	94 935.9	2.6
NPR	0.03%	0.00%	0.00%	0.18%	0.56%	43 517.0	1.5
TBF	55.26%	57.77%	50.83%	37.09%	22.84%	92 248.7	2.6

Table 7: Impact of each preprocessing routine

		#	*Iori et al. (2007)*					*Our Branch & Cut*				
Class	Type	Inst.	Fea.	sec.	Inf.	sec.	Solved	Fea.	sec.	Inf.	sec.	Solved
1	3	30	4	34.1	26	111.7	30	4	0.2	26	0.4	30
	4	198	17	164.9	106	377.2	123	27	1.1	171	1.2	198
	5	200	5	509.9	1	595.3	6	53	196.1	108	95.4	161
2	2	1	0		1	0.3	1	0		1	0.1	1
	3	128	4	2.0	124	61.3	128	4	0.1	124	0.3	128
	4	198	17	175.9	110	380.9	127	23	0.8	175	0.6	198
	5	200	9	309.3	4	730.6	13	64	63.0	117	43.9	181
3	3	2	0		2	19.6	2	0		2	0.1	2
	4	155	20	223.0	53	431.9	73	29	3.0	126	3.4	155
	5	206	9	337.5	0		9	54	199.4	103	118.4	157
4	4	187	18	298.5	30	366.7	48	41	90.1	141	60.0	182
	5	182	3	280.6	0		3	37	200.6	46	222.9	83
5	3	11	0		5	555.6	5	2	14.3	9	6.3	11
	4	246	8	421.7	0		8	45	268.5	119	217.5	164
	5	239	0		0		0	2	602.3	11	296.8	13
Total		2183	114		462		576	385		1279		1664

Table 8: Comparison of two algorithms

times are also significantly smaller. It should be noted that the smallest undecided instance has 20 items, as compared with 13 items for the algorithm of Iori et al. Conversely, the largest instance solved with our algorithm has 52 items, as compared with 29 items for Iori et al.

8 Conclusion

This paper has demonstrated the effectiveness of a new branch-and-cut algorithm for a two-dimensional packing problem with unloading constraints through numerical results on a set of benchmark instances. The next step will now consist in integrating this algorithm into a problem-solving methodology for a mixed vehicle routing and loading problem. Alternative approaches, in particular dynamic programming, are also currently considered to tackle the packing problem.

Acknowledgments. Financial support for this work was provided by the Canadian Natural Sciences and Engineering Research Council (NSERC). This support is gratefully acknowledged.

References

[1] Alvarez-Valdes R., Parreño F., Tamarit J.M., A GRASP algorithm for constrained two-dimensional non guillotine cutting problems. Journal of the Operational Research Society 56, 414-425, 2005.

[2] Alvarez-Valdes R., Parreño F., Tamarit J.M., A branch and bound algorithm for the strip packing problem. OR Spectrum 31, 431-459, 2009.

[3] Arahori Y., Imamichi T., Nagamochi H., An exact strip packing algorithm based on canonical forms, Computers & Operation Research 39, 2991-3011, 2012.

[4] Belov G., Rohling H., A branch-and-price graph-theoretical algorithm for orthogonal-packing feasibility. Technical report, Preprint MATH-NM-10-2009, Technische Universität Dresden, 2009.

[5] Bortfeldt A., Wäscher G., Container loading problems : A state-of-the-art review. Technical Report, 2012.

[6] Boschetti M.A., Montaletti L., An exact algorithm for the two-dimensional strip-packing problem, Operations Research 58, 1774-1791, 2010.

[7] Caprara A., Monaci M., Bidimensional packing by bilinear programming. Mathematical Programming 118, 75-108, 2009.

[8] Codato G., Fischetti M., Combinatorial Benders cuts for mixed-integer linear programming. Operations Research 54, 756-766, 2006.

[9] Cordeau J.-F., Iori M., Ropke S., Vigo D., Branch-and-cut-and-price for the capacitated vehicle routing problem with two-dimensional loading constraints. ROUTE 2007, Jekyll Island, U.S.A., May 2007.

[10] Côté J.-F., Iori M., Dell'Amico M., Combinatorial Benders cuts for the strip packing problem, Technical Report, DISMI, University of Modena and Reggio Emilia, 2013.

[11] Clautiaux F., Carlier J., Moukrim A., A new exact method for the two-dimensional orthogonal packing problem. European Journal of Operational Research 183, 1196-1211, 2007.

[12] Duhamel C., Lacomme P., Quilliot A., Toussaint H., A multi-start evolutionary local search for the two-dimensional loading capacitated vehicle routing problem, Computers & Operation Research 38, 617-640, 2011.

[13] Fekete S., Schepers J., A combinatorial characterization of higher-dimensional orthogonal packing, Mathematics of Operations Research 29, 353-368, 2004.

[14] Fekete S., Schepers J., van der Veen J.C., An exact algorithm for higher-dimensional orthogonal packing, Operations Research 55, 569-587, 2007.

[15] Fuellerer G., Doerner K.F., Hartl R.F., Iori M., Ant colony optimization for the two-dimensional loading vehicle routing problem. Computers & Operations Research 36, 655-673, 2009.

[16] Gendreau M., Iori M., Laporte G., Martello S., A tabu search heuristic for the vehicle routing problem with two-dimensional loading constraints. Networks 51, 4-18, 2008.

[17] Gu Z., Nemhauser G.L., Savelsbergh M.W.P., Lifted cover inequalities for 0-1 integer programs: Complexity. INFORMS Journal on Computing 11, 117-123, 1999.

[18] Herz J.C., Recursive computational procedure for two-dimensional stock cutting. IBM Journal of Research and Development 16, 462-469, 1972.

[19] Iori M., Martello S., Routing problems with loading constraints, TOP 18, 4-27, 2010.

[20] Iori M., Salazar-González J.-J., Vigo D., An exact approach for the vehicle routing problem with two-dimensional loading constraints. Transportation Science 41, 253-264, 2007.

[21] Kaparis K., Letchford A.N., Separation algorithms for 0-1 knapsack polytopes. Mathematical Programming 124, 69-91, 2010.

[22] Leung S.C.H., Zhang Z., Zhang D., Hua X., Lim M.K., A meta-heuristic algorithm for heterogeneous fleet vehicle routing problems with two-dimensional loading constraints. European Journal of Operational Research 225, 199-210, 2013.

[23] Martello S., Monaci M., Vigo D., An exact approach to the strip-packing problem. INFORMS Journal on Computing 15, 310-319, 2003.

[24] Mesyagutov M., Scheithauer G., Belov G., LP bounds in various constraint programming approaches for orthogonal packing. Computers & Operation Research 39, 2425-2438, 2012.

[25] Ropke S., Pisinger D., An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows, Transportation Science 40, 455-472, 2006.

[26] da Silveira J.L.M., Miyazawa F.K., Xavider E.C., Heuristics for the strip packing problem with unloading constraints. Computers & Operation Research 40, 991-1003, 2013.

[27] Wäscher G., Haussner H., Schumann H., An improved typology of cutting and packing problems. European Journal of Operational Research 183, 1109-1130, 2007.

[28] Wolsey L.A., Valid inequalities for 0-1 knapsacks and MIPS with generalised upper bound constraints. Discrete Applied Mathematics 29, 251-261, 1990.

[29] Zachariadis E.E., Tarantilis C.D., Kiranoudis C.T., A guided tabu search for the vehicle routing problem with two-dimensional loading constraints. European Journal of Operational Research 3, 729-743, 2009.