Exact method for the two-dimensional orthogonal packing problem

François Clautiaux, Jacques Carlier, Aziz Moukrim

Laboratoire HeuDiaSyC, UMR CNRS 6599
Université de Technologie de Compiègne, BP 20529, 60205 Compiègne, France
Email: {francois.clautiaux, jacques.carlier, aziz.moukrim}@hds.utc.fr

Abstract

The two-dimensional orthogonal packing problem ($2OPP$) consists of determining if a set of rectangles (items) can be packed into one rectangle of fixed size (bin). In this paper we propose two exact algorithms for solving this problem. The first algorithm is an improvement on a classical branch&bound method, whereas the second algorithm is based on a two-step enumerative method. We also describe reduction procedures and lower bounds which can be used within the branch&bound method. We report computational experiments for randomly generated benchmarks, which demonstrate the efficiency of both methods.

Key words: two-dimensional orthogonal packing problem, exact method, branch&bound
PACS:

1 Introduction

The two-dimensional orthogonal packing problem ($2OPP$) consists of determining if a set of rectangles (items) can be packed into one rectangle of fixed size (bin). This problem occurs in industry when rectangular pieces of steel, wood, or paper have to be cut from a larger rectangle. It can also be used to model the layout of a newspaper. It is *NP-complete* as it generalizes the classical one-dimensional bin-packing problem [1]. This problem is an issue in the *two-dimensional bin-packing* ($2BP$), *two-dimensional knapsack* ($2KP$), and *strip-packing* problems ($2SP$).

A $2OPP$ instance D is a pair (A, B). A is the set of items a_i to pack. $B = (W, H)$ is a bin of width W and height H. An item a_i has a width w_i and a height h_i ($w_i, h_i \in \mathbb{N}$). We consider the version of the problem in which the

items cannot be rotated. To simplify the notation we usually denote an item a_i by its size (w_i, h_i). The position of the item a_i, denoted by (x_i, y_i), corresponds to the coordinates of its bottom-left corner.

Several lower bounds have been proposed for $2BP$, all of them can be used for $2OPP$. Martello and Vigo [2] and Boschetti and Mingozzi [3,4] propose lower bounds dedicated to this problem, whereas Fekete and Schepers [5] apply one-dimensional functions called *Dual Feasible Functions (DFF)* separately on each dimension and show that this method can be used for both two and three dimensional bin packing problems. In a recent paper [6] we propose new lower bounds following the framework defined by Fekete and Schepers. We use a discrete version of DFF and propose a new class of functions called *Data-Dependent DFF*. These functions behave as DFF for a given instance. The bounds obtained dominate previous ones [2–4,7] and improve on the best results for well-known established benchmarks derived from the literature. Recently, Caprara et al. [8] have proposed an enumerative method to find the best pair of DFF for a given two-dimensional bin-packing instance.

Although several papers deal with heuristics for $2BP$ or $2OPP$ [4,9,10] (see [11,12] for extensive surveys), few exact methods exist. Hadjiconstantinou and Christofides [13] study the two-dimensional knapsack and $2OPP$. They propose a general framework for a branch&bound method but they do not report any computational results. Martello and Vigo [2] introduce an exact method for $2BP$. Part of their work is devoted to $2OPP$: their algorithm explicitly enumerates all feasible packings. It generates many redundancies since the same item can be packed in a given position several times during the enumeration. Scheithauer [14] lists many redundancies and proposes methods for reducing their number.

In order to avoid exploring redundant solutions, Fekete and Schepers [15] introduce a graph theoretical model. They show that any *packing class* (*i.e.* a set of packings with common properties) can be associated with a pair of interval graphs. The authors also propose an enumerative algorithm [5] to construct such interval graphs. It dramatically reduces the number of redundancies compared to the method proposed by Martello and Vigo [2], and outperforms all previous methods for $2OPP$.

In this paper we propose two branch&bound methods for $2OPP$. The first algorithm ($LMAO$) is an improvement on the method proposed by Martello and Vigo [2]. Instead of testing the packing of items in each dominant coordinate, the algorithm enumerates the packings of items only in the leftmost-downward position denoted by c. It also tests the possibility of not packing any item in c.

The second method ($TSBP$) is a two step branch&bound method based on a

new relaxation of the problem. In the first step (*outer branch&bound method*), all solutions of a relaxed problem are enumerated. For each solution found, a second enumerative method is run (*inner branch&bound method*) to seek a solution for the initial problem corresponding to the current solution. The relaxation consists in cutting each item (w_i, h_i) into h_i strips of width w_i. All strips related to a given item have to be packed at the same x-coordinate, even if all parts are not contiguous. Using this relaxation, the problem consists in finding a suitable set of x-coordinates for the items. When a solution is sought for the relaxed problem (*i.e.* a valid list of x-coordinates for the items), the inner branch&bound method tries to add a set of y-coordinates to obtain a solution for the initial problem. If there is no solution for the orthogonal packing problem, the inner branch&bound method may not be launched. If the instance is feasible, the number of states enumerated is also dramatically reduced: many non-feasible configurations are not enumerated and the inner branch&bound method is rarely launched more than once.

At each step of the enumeration, the quality of the lower bounds is improved on. The idea is to make use of the packed items to transform the instance into another instance with larger items. In the outer branch&bound method, we avoid all symmetries related to the y-axis, but redundant solutions can still be reached. So we propose methods to avoid many redundancies in the search tree. These methods are different from [14], as they can be applied even when the y-coordinates are not fixed yet.

Numerous benchmarks are available for $2BP$ or knapsack problems, but we want our method to be tested against difficult $2OPP$ instances to allow a better comparison with the other methods. Several available benchmarks for $2OPP$ are related to feasible sets of items. We wanted to test our method against both feasible and non feasible instances. So we have generated our own instances. We take into account the number of items, the size of the bin and the discrepancy between the area of the bin and the global area of the items to genereate those difficult $2OPP$ instances. This approach is similar to the approach of Hopper and Turton [16]. We report computational experiments testing our methods against the new randomly-generated benchmarks. We compare our algorithms to the method proposed by Martello and Vigo [2], and Fekete and Schepers [5]. The computational results confirm the interest of our method, as it outperforms the method of Martello and Vigo and is competitive compared to the method of Fekete and Schepers.

In section 2 we show how our reduction procedures can be applied to the feasibility problem and how they can be improved. Section 3 describes our lower bounds whereas section 4 is devoted to our new branch&bound method. Section 5 deals with redundancies and how they can be avoided using our methods. In section 6 we introduce the new benchmarks and we report computational experiments.

2 Reduction procedures

Before applying an approximate or an exact method to the feasibility problem it is interesting to reduce the size of the instance. In this section we propose several reduction procedures related to special cases where an optimal way to pack a set of items is known. We also show that a reduction procedure, described in [6] for $2BP$, can be adapted to $2OPP$ to obtain better results.

2.1 First reduction procedures

If the height of an item a_i is equal to the height of the bin, and there is a solution for $2OPP$, there is a solution such that a_i is packed at position $(0, 0)$. Also the property holds for the width. So at each step of the algorithm, if there exists such an item, it is removed and the size of the bin is updated. This method can be generalized to items which cannot be packed above any other item. It can remove several items when used with a preprocessing phase which increases the size of large items, or within an enumerative method.

We now consider the case where four items can only be packed as a *frame* around the bin (Figure 1). Let (A, B) be a $2OPP$ instance, and a_i, a_j, a_k and a_l four items of A such that $h_i + h_k = h_j + h_l = H$ and $w_i + w_l = w_j + w_k = W$. The following proposition holds.

Proposition 1 *There exists a feasible packing for (A, B) if and only if there is a feasible packing for (A, B) such that a_i is packed in position $(0, 0)$, a_l is packed in position $(w_i, 0)$, a_k is packed in position $(0, h_i)$, and a_j is packed in position (w_k, h_l).*

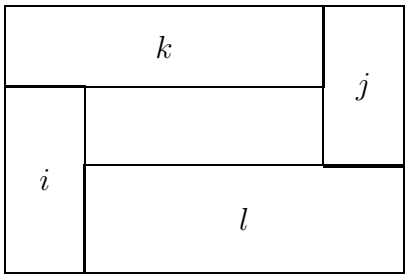

Fig. 1. Frame configuration

The reduction procedure is directly deduced from the proposition. If four such items are found, they are removed and the size of the bin is updated.

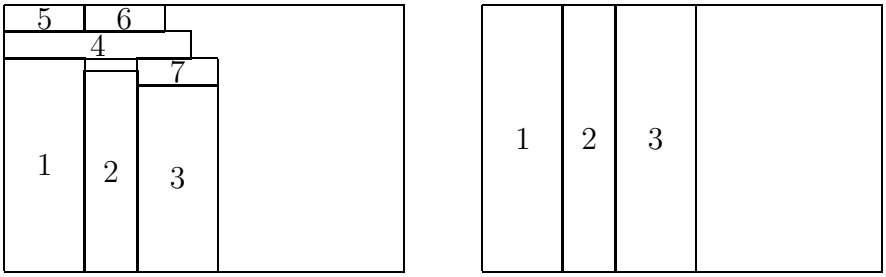

Fig. 2. v_1^* procedure

2.2 Improving a reduction procedure

In a recent paper [6] we propose a reduction procedure denoted by v_1. The same idea is used by Martello et al. for the strip-packing problem [17]. Let D be an instance of OPP and p an integer, $1 \leq p \leq \frac{1}{2}H$. We define the set of tall items $A_{tall} = \{a_i \in A : h_i > H - p\}$, and the set of shallow items $A_{shallow} = \{a_i \in A : h_i \leq p\}$. $m = |A_{tall}|$. For each item a_i of A_{tall} a bin B_i of size $(H - h_i, w_i)$ is created. Consider $D' = (A_{shallow}, \{B_1, \dots B_m\})$ the following decision problem: *¨Is there a feasible packing for $A_{shallow}$ in the bins $B_1, \dots B_m$?¨* .

Proposition 2 *[6] If D' has a feasible solution then applying function $v_1^{p,H}$*

$$v_1^{p,H} : [0,W] \times [0,H] \rightarrow [0,W] \times [0,H]$$
$$a_i \mapsto \begin{cases} (w_i, H) \text{ if } h_i > H - p \\ (0,0) \text{ if } h_i \leq p \\ (w_i, h_i) \text{ otherwise} \end{cases}$$

on A does not modify the feasibility of D.

For $2OPP$, both tall and shallow items are removed and the size of the bin is updated. For $2BP$, if a shallow item has too large a width, it cannot be packed in the area above a tall item. The reduction procedure can also be improved on for $2OPP$, as shallow items may be packed in the area above several tall items. The procedure obtained is denoted by v_1^*. Figure 2 illustrates the fact that v_1^* can reduce the size of the instance when v_1 cannot. We use a greedy algorithm to solve the decision problem created. For a given value of p, the sets A_{tall} and $A_{shallow}$ are computed. Items of A_{tall} are packed on the left-hand side of the bin in decreasing order of height. Then we try to pack the small items above them following the *bottom-left decreasing* rule.

3 Lower Bounds

A classical lower bound for $2BP$ is the continuous bound L_0. It is equal to the total area of the items divided by the area of one bin. $L_0 = \lceil \frac{\sum_{a_i \in A} w_i h_i}{WH} \rceil$. This bound can also be applied to a $2OPP$ instance. One way of improving this method is to modify the size of the items so that the obtained instance is feasible if the initial instance is feasible. It can be done using *Dual-Feasible Functions* (DFF) independently for each dimension [18]. The original concept of DFF has been proposed by Johnson [19] and first used for the $1BP$ by Lueker [20] and then by Fekete and Schepers [7]. It has been generalized for higher dimensional problems by Fekete and Schepers [18]. We use the framework proposed in [18] in a previous paper [6], using a discretisation of DFF. We now give a definition of the *discrete DFF*.

Definition 3 *A* Discrete DFF f *is a discrete application from* $[0, X]$ *to* $[0, X']$ *(X and X' integers) such that* $x_1 + x_2 + \ldots + x_k \leq X \Rightarrow f(x_1) + f(x_2) + \ldots + f(x_k) \leq f(X) = X'$.

We also introduced the concept of *Data-Dependent DFF (DDFF)* [6]. These functions behave as DFF with the instance for which they have been generated (see [6] for more details). We also propose analytic families of DFF, similarly to Fekete and Schepers [7]. So we introduce two families of DFF (f_0^k and f_2^k), and a family of DDFF (f_1^k) [6], which are inspired by the work of Boschetti and Mingozzi [3], and lead to bounds which dominate those proposed by [3]. The three functions are defined as follows. A more complete description can be found in [6].

- Let $k = 1, \ldots, \frac{C}{2}$. $f_0^k(x) = 0$ if $x < k$, $f_0^k(x) = x$ if $k \leq x \leq C - k$, and $f_0^k(x) = C$ if $C - k < x \leq C$.
- Let $k = 1, \ldots, \frac{C}{2}$. The data-dependent DFF f_1^k is defined for the value C and a list of integer values $c_1, c_2, \ldots, c_n$ ($I = \{1, \ldots, n\}$). We introduce the set $J = \{i \in I : \frac{1}{2}C \geq c_i \geq k\}$ and for a given integer Y, $M_C(Y, J)$ denotes the maximum number of items c_i such that $i \in J$ which can be packed together in a container of size Y. It can be computed in linear time if the items are sorted by increasing order of size. $f_1^k(x) = 0$ if $x < k$, $f_1^k(x) = 1$ if $k \leq x \leq \frac{1}{2}C$, and $f_1^k(x) = M_C(C - x, J)$ if $\frac{1}{2}C < x$.
- Let $k = 1, \ldots, \frac{C}{2}$. $f_2^k(x) = 2\lfloor \frac{x}{k} \rfloor$ if $x < \frac{1}{2}C$, $f_2^k(x) = \lfloor \frac{C}{k} \rfloor$ if $x = \frac{1}{2}C$, and $f_2^k(x) = 2\lfloor \frac{C}{k} \rfloor - 2\lfloor \frac{C-x}{k} \rfloor$ if $x > \frac{1}{2}C$.

The obtained lower bound is $L^{DDFF} = \max_{\substack{1 \leq k \leq \frac{W}{2}, 1 \leq l \leq \frac{H}{2}, \\ u \in \{0,1,2\}, v \in \{0,1,2\}}} \left\{ \left\lceil \sum_{a_i \in A} \frac{f_u^k(w_i) f_v^l(h_i)}{f_u^k(W) f_v^l(H)} \right\rceil \right\}$.

For $2OPP$ any reduction procedure or lower bound can be applied to the instance obtained after using the (D)DFF. The reduction related to large

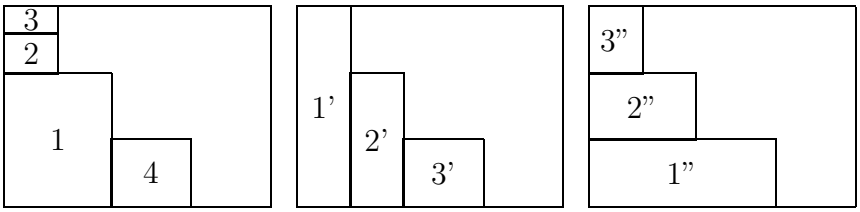

Fig. 3. Creating new instances

items becomes more efficient and can improve the quality of the lower bound in many cases. As function f_1^k is data-dependent, the results can be improved using a simple method: at each step of the algorithm, an item is removed from the instance, and the lower bound applied on the residuous instance. Note that this method would not improve the value of the bound if only DFF were used.

3.1 Modification of the instance

Consider an enumeration process such that the x-coordinate and then the y-coordinate of items are fixed. The partial or total packings provide information which can be used to update the lower bounds and the reduction procedures. One easy way of improving the bounds is to check if each remaining item a_i can be packed in the remaining free areas. More powerful methods can be designed. The idea is to aggregate the items which are packed side-by-side to create new instances which are more constrained than the initial instance (Figure 3). The lower bounds and the reduction procedures are applied to these instances to obtain better results. The method is based on a geometric observation. Consider the polygon ψ formed by the set A_1 of items packed in the bin. If A_1 is replaced in the initial instance D by another set A'_1 such that items of A'_1 can be packed in ψ, the following property holds:

Proposition 4 *If there is no feasible solution for $A - A_1 + A'_1$ in B, then there is no feasible solution for A in B such that items of A_1 are packed in ψ.*

PROOF. *If the original instance D has a feasible solution such that A_1 is packed to form the polygon ψ, a solution for the modified instance D' can be found by packing the items of A'_1 to form the same polygon. The other items are packed the same way and a feasible solution for D' is built.*

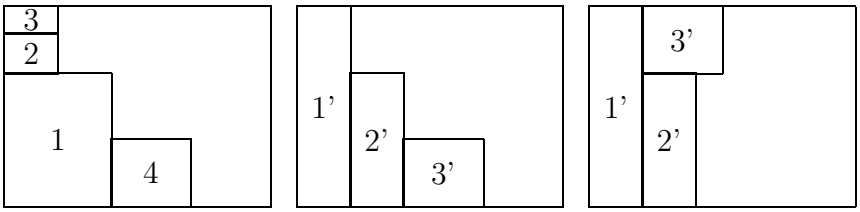

Fig. 4. A configuration to avoid

Given a set of packed items A_1, we create a set A'_1 which is more constrained than A_1. The idea is to maximize the height or the width of the created items to obtain two instances (Figure 3). If several transformed items which are packed side by side are shallow they can be packed one above the other in the new instance. For example in Figure 4, the third configuration is not allowed, otherwise the bounds may be weaker, as geometric information are lost. To avoid this situation we operate a second modification on the items. If the packed items have been cut into vertical strips, we add to the new items a height equal to H and the height of the bin is updated to $2H$. So the bounds can take into account the fact that these items cannot be packed one above the other. If the packed items do not fit the width of the bin a dummy item with size (w^*, H) is created, w^* being the free width to the right of the packed items (Figure 5). Note that after the second transformation, item $3'$ cannot be packed above item $2'$. The same operation can be realized when the width is considered. We denote the new instance obtained as D''. The reduction procedures described in section 2 and our lower bounds can be applied to D''. The results are improved on because the problem is more constrained as the number of large items increases.

Proposition 5 *If there is no feasible solution for the new instance D'' obtained there is no feasible solution for the initial instance D including the current partial packing.*

PROOF. *If there is a solution for D including the packing of the items of A_1 in ψ there exists a solution for D'. It can be found by packing the modified items in such a way that the remaining area is the same as in D. The remaining items of $A \setminus A_1$ are packed in the same way as in the solution for D. So if there is a solution for D including the packing of A_1 in ψ there is a packing for the modified instance.*

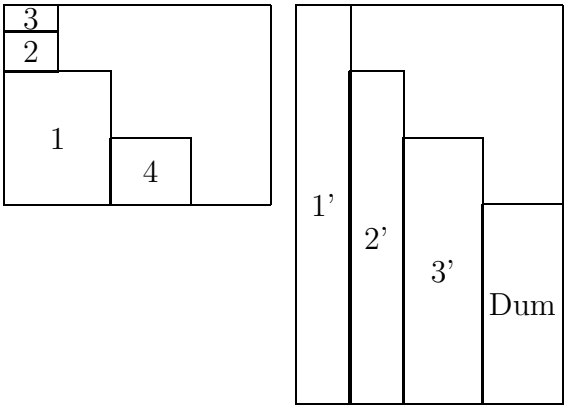

Fig. 5. Improving the new instances

4 Exact methods

We propose in this section two new exact methods for $2OPP$. The first algorithm is an improvement on a branch&bound method proposed by Martello and Vigo [2]. The second algorithm is based on a two-step enumerative method. The idea is to enumerate all the solutions of a relaxed problem. During this step only the x-coordinates of the items are fixed. For each configuration found a second exact algorithm searches for a valid list of y-coordinates to obtain a solution for the initial problem.

4.1 Classical methods

Several methods in the literature [13,2] build a configuration step by step following the *leftmost-downward* strategy. As Martello and Vigo [2] have applied an efficient implementation of this algorithm for $2BP$, we will denote this method MV. The coordinates considered for an item a_i are those in which a_i has its left edge adjacent either to the right edge of a previously packed item or to the left edge of the bin, and its bottom edge adjacent either to a packed item or to the bottom edge of the bin [21]. In Figure 6 the left configuration is dominated whereas the right configuration is non-dominated. At each step there is a finite list of l positions and a list of n_1 items. There are two choices to make: the item to be packed and the position to be used. This leads to $l \times n_1$ child nodes. This method generates a large number of redundancies which have to be handled to obtain an efficient algorithm. A given configuration can be found in different parts of the search tree. For example, in Figure 6, the right packing can be obtained by the two following orders : $\pi_1 = [(a_1,(0,0)),(a_2,(3,0)),(a_3,(5,0)),(a_4,(0,8))]$ and $\pi_2 = [(a_1,(0,0)),(a_4,(0,8)),(a_2,(3,0)),(a_3,(5,0))]$.

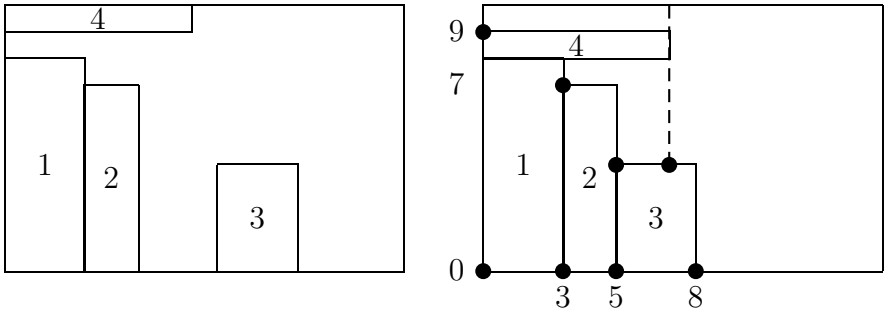

Fig. 6. The *leftmost downward* rule

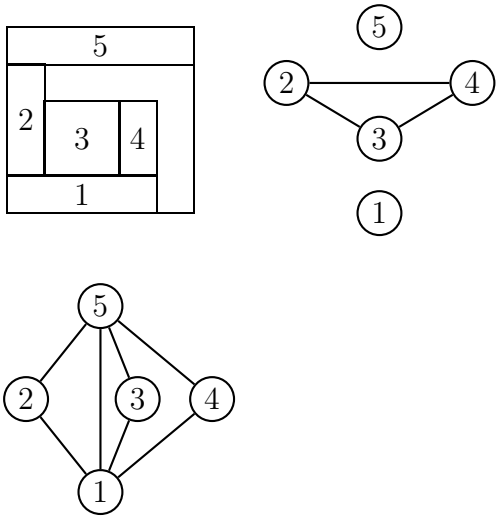

Fig. 7. Interval graphs model

Fekete and Schepers [15,5] propose a new model for the feasibility problem. They show that a pair of interval graphs can be associated with any *packing class* (*i.e.* a set of packings with common properties). The interest of this concept is that a large number of symmetries are removed as only one packing by class is enumerated. A graph $G_d = (V, E_d)$ is associated with each dimension, where $|V| = n$ is the number of items in the bin and $d \in \{w, h\}$ the dimension considered. In the two graphs each vertex v_i is associated with an item a_i. An edge is added in the graph G_w (respectively G_h) between two vertices v_i and v_j if the projections of items a_i and a_j on the horizontal (respectively vertical) axis overlap (see Figure 7). The authors prove that the packing class related to a pair of interval graphs G_w and G_h is feasible if and only if

(1) each stable set S of G_w is such that $\sum_{v_i \in S} w_i \leq W$
(2) each stable set S of G_h is such that $\sum_{v_i \in S} h_i \leq H$
(3) $E_h \cap E_w = \emptyset$

The authors propose a branch&bound method to search for a pair of inter-

val graphs with the properties described above. This method avoids a large number of redundancies compared to the classical method, and outperforms all previous methods.

4.2 Improving on the classical branch&bound method: LMAO

We propose an improvement on the classical method in order to avoid a large number of repetitions in the branch&bound method. At each node of the search tree our method generates at most $n_i + 1$ child nodes corresponding to the packing of the n_i items in a single position or not using this position.

We now describe this method in detail. We define the concept of *active* and *non-active* positions. Initially all positions are active. The leftmost downward active position (*i.e.* the position with the smallest y-coordinate among the positions with the smallest x-coordinates) is denoted by c. At each node of the search tree the algorithm tests the packing of each item at position c. The possibility of not packing any item in c is also tested. In this case it is not interesting to pack any item in c in a future step. So this position is said inactive. We denote the exact method obtained by *LMAO (Left-Most Active Only)*.

When an item a_i is packed at position $c = (x, y)$ the area left to a_i is removed if there is such an area. Indeed the possibility of packing an item at this position is enumerated in another part of the search tree. Practically speaking, a dummy item is created and packed to the left of a_i to ensure that no item will be packed at this position and to increase the total area of the items. In this case a new leftmost downward position is created at y-coordinate $y_i + h_i$. This new coordinate is active. Note that when the dummy item is added, the total area of the items (including the dummy items) can become larger than the area of the bin.

Definition 6 *A LMAO order is a sequence of choices related to a given solution using algorithm LMAO. It is composed of choices of a given item a_i to pack in the current dominant position c: (a_i, c) and choices not to pack any item in the current leftmost-downward active position: $\overline{c}$.*

This method has the advantage of reducing the number of orders on the items which lead to the same solution. In fact, this order is unique for a given solution. For example the configuration of Figure 6 can only be reached with the following order.

$$\pi = [(a_1, (0, 0)), (a_4, (0, 8)), \overline{(0, 9)}, (a_2, (3, 0)), \overline{(3, 7)}, (a_3, (5, 0))]$$

Proposition 7 *For a given leftmost downward configuration P there is only one $LMAO$ order π which leads to P.*

PROOF.

Let π_1, π_2 be two distinct $LMAO$ orders and j be the first index for which $\pi_1(j) \neq \pi_2(j)$. As the two partial configurations are the same before step j, two cases are possible:

(1) $\pi_1(j) = (a_i, c)$ and $\pi_2(j) = (a_k, c)$, $k \neq i$. In this case item a_i cannot have the same position in the two obtained configurations because a_k uses this coordinate when we follow $LMAO$ order π_2.
(2) $\pi_1(j) = \overline{c}$ and $\pi_2(j) = (a_k, c)$. In this case position c will not be inspected any more following order π_1 and thus item a_k cannot be packed in position c.

The configurations obtained in the two cases are not equivalent.

Theorem 8 *$LMAO$ is a valid method for $2OPP$.*

PROOF. *We show that for any feasible configuration P enumerated by MV there is a $LMAO$ order which leads to P. As MV is valid, the theorem immediately holds. Let P be a feasible configuration obtained by MV. We construct a new MV order π_1 from P by selecting iteratively the remaining item which is packed in the leftmost-downward position available at the current step. $\pi_1 = [(a_{i_1}, c_1), (a_{i_2}, c_2), \ldots, (a_{i_n}, c_n)]$. From π_1 it is possible to obtain a $LMAO$ order π_2 which leads to P. For each choice (a_{i_j}, c_j) in π_1 made with MV, we give an equivalent sequence of choices in the $LMAO$ order. Two cases can occur.*

(1) c_j is the current leftmost-downward position. The $LMAO$ choice consists in packing item a_j in this corner, and a partial configuration is obtained, which is equivalent to the configuration obtained with MV.
(2) c_j is not the current leftmost downward position c_k. It is an immediate consequence of the construction of π_1 that no item is packed in position c_k in P (otherwise, the current $LMAO$ choice would be to pack this item in c_k). The $LMAO$ choice consists in declaring c_k inactive, and then recursively declaring the current leftmost-downward active position inactive while c_j is not reached. The construction of π_2 ensures that all skipped *positions are not used in P. So after several iterations, as the two current configurations are the same, position c_j is reached. Then the next $LMAO$ choice consists in packing a_{i_j} in position c_j.*

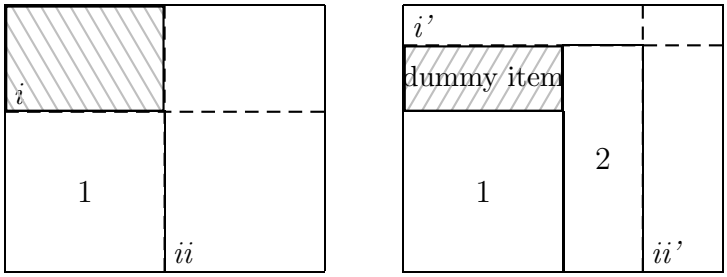

Fig. 8. LMAO

Figure 8 illustrates the two main principles of $LMAO$. The situation on the left occurs if no item is packed in the leftmost-downward position i. The position ii becomes the leftmost downward active position. The right situation occurs if item 2 is packed at position ii. A dummy item is inserted in the area to the left of item 2. A new active position i' is created. Note that $LMAO$ and MV enumerate an identical set of packings. The only difference is that many redundancies do not occur with $LMAO$ and thus do not have to be handled by any expensive methods. When a position c becomes inactive, we add a dummy item of height 1 in c. As the size of the items are integers, when an item overlaps with the corresponding area the minimum overlapped area has height 1.

4.3 two-step-algorithm: TSBP

Our goal is to aggregate in a first step a large number of packings in order to determine that a whole set of packings are not feasible. For this purpose we propose a two-step method composed of two branch&bound methods. The *outer branch&bound method* only determines the x-coordinates of the items. We relax the constraint related to the height of the items (*i.e.* several horizontal strips of an item are allowed not to be contiguous). For each solution found in the first step an *inner branch&bound method* is launched. This method determines the y-position of the items according to the x-coordinates chosen in the first step. Each configuration in the outer branch&bound method is related to a set of packings. Thus when a pruning is performed in the outer tree, a large number of non-feasible packings are not enumerated. We denote the whole method by *TSBP (Two-Step Branching Procedure)*. Other papers deal with mathematical models for the two-dimensional knapsack problem using horizontal or vertical strips [22–24], but none of them add the constraint that all strips of a same item have to be packed at the same x-coordinate. We now describe the two steps of the algorithm.

The outer branch&bound method determines the x-coordinates of the items. During this step, the constraints related to the height of the items are not

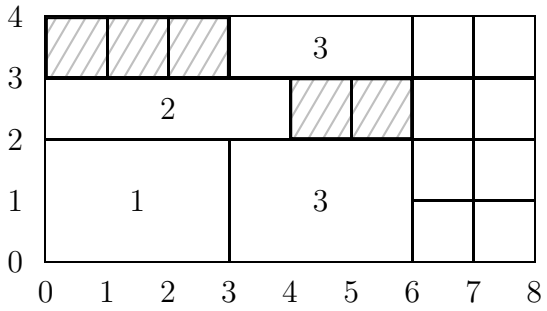

Fig. 9. Outer branch&bound method

taken into account. Several horizontal strips of the same item can be packed in non-contiguous y-positions. We keep several constraints: the total height of the items packed at a given x-coordinate has to be less than the height of the bin H. Moreover the horizontal strips of the same item have to be packed at the same x-coordinate (see Figure 9).

For each x-coordinate $x < W$, we denote the height of the horizontal strips at this position by $K(x)$. These strips a_i are those packed at position x but also those packed at position $x_i < x$, $x_i + w_i > x$. Before the first step no item is packed in the bin. $K(x) = 0, \forall x$. When an item a_i is packed at position $x = 0$, the value $K(x)$ is updated for $x = 0, \ldots, w_i - 1$ ($K(x) \leftarrow K(x) + h_i$). At a given step of the branch&bound method, the algorithm enumerates all sets of items A_j which can be packed at the current x-coordinate x. The value x is the smallest value for which $K(x) < H$. The set A_j has to obey this rule: $\sum_{a_i \in A_j} h_i \leq H - K(x)$. By construction, the values $K(x)$ are non-increasing so the constraint is satisfied for all x-coordinates greater than x. If for a given x-coordinate x, we have $K(x) < H$ after having selected A_j, a dummy item of size $(1, H - K(x))$ is created so that $K(x) = H$. This dummy item increases the total area of the items and may increase the value of lower bound. Practically speaking, by no means all values of $K(x)$ are interesting. The only x-coordinates tested are those related to the end of an item a_i ($x = x_i + w_i$). When a dummy item is added its size is the difference between the current x-coordinate and the first x-coordinate where the set of items is different.

In Figure 9 the current x-coordinate is $x = 6$. The first hashed surface is a dummy item which has been added to make up the height between position 0 and position 3. Such an item is not necessary between positions 3 and 4 because the set of items occupies the entire height. It is an arbitrary figure: item 1 is neither above nor below item 2. A feasible packing can be obtained by choosing y-position 3 for item 2.

If at a given step of the algorithm an item goes beyond the right-hand side of

the bin, or if the total area of the items including the dummy items is larger than the total area of the bin, the enumeration is stopped. At each leaf of the search tree, each item has a fixed x-coordinate. The list of x-coordinates obtained is related to a set of packings, feasible or not. A method has to be run to determine if there is a feasible packing under the constraints added during the first step.

Lemma 9 *The outer branch&bound method enumerates all feasible configurations of the relaxed problem.*

PROOF. *A feasible configuration of the relaxed problem is related to a list of items packed for each x-coordinate. At each step of the algorithm, all possibilities are tested for the set A_j to pack at the current x-coordinate x. So each feasible configuration is enumerated by the algorithm.*

Theorem 10 *If the method used for the inner method is valid, $TSBP$ is valid.*

PROOF. *The validity of the method holds on the fact that* all *feasible configurations of the relaxed problem are enumerated. Suppose a given solution s is found using only the inner method. If s is feasible for the bin-packing problem the x-coordinates of the items in s are feasible for the relaxed problem. So because of Lemma 9 it is enumerated by the outer tree and the inner method is launched.*

For the inner branch&bound method we use the $LMAO$ method. It is modified in the following way: an item a_i is candidate to be packed at position (x_j, y_j) if its x-coordinate has been fixed to x_j in the outer tree.

5 Redundancies in the outer tree

The two-step method avoids a large number of redundancies, but methods can be applied to reduce their number further. For the classical branch&bound method MV, Scheithauer [14] lists several classes of redundancies and proposes methods to avoid some of these repetitions. For the outer branch&bound method in $TSBP$, detecting the redundancies is more difficult. We have to propose *dominance* relations and ensure that a non-feasible configuration cannot dominate a feasible configuration. We now show that two redundancies can be avoided during the tree search: a *block equivalence* and a *pseudo-symmetry*. For both cases we propose a class of non-dominated configurations which are actually enumerated during the branch&bound method.

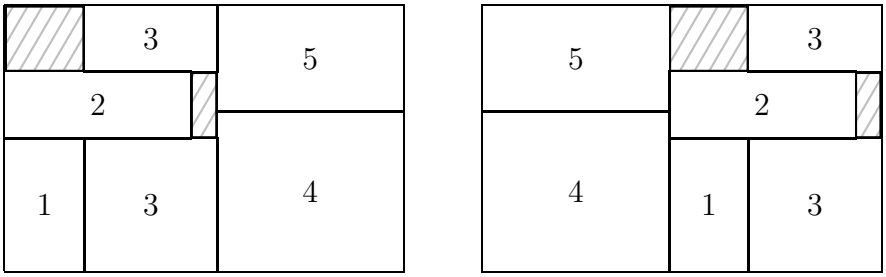

Fig. 10. Block equivalence

5.1 Block equivalence

We consider cases where a configuration is composed of two independent *blocks*.

Definition 11 *A configuration is said to be composed of two independent* blocks *if there exist two sets A_1 and A_2, $A = A_1 \cup A_2$, such that no piece of an item of A_1 is on the same x-coordinate as an item of A_2:*

$$\exists A_1, A_2 \text{ such that } A_1 \cup A_2 = A, \forall a_i \in A_1, \forall a_j \in A_2, x_i + w_i \leq x_j \text{ or } x_j + w_j \leq x_i$$

Figure 10 illustrates the equivalence between two configurations composed of the same blocks. In this case, the two blocks are independent and thus inverting the positions of blocks A_1 and A_2 leads to an equivalent set of packings. We enumerate only one of the two configurations. For this purpose we select before the enumeration an item a^* which does not change during the enumeration. If item a^* is in the left-hand block, the enumeration goes on, otherwise, the branch is cut. This proposition can be generalized if the configuration is composed of more than two blocks. In this case another method has to be used to avoid exploring redundant solutions.

In order to choose one configuration among the two equivalent configurations, we introduce a dominance rule based on lexicographic labels *lbl*.

Definition 12 *Let a_i and a_j be two items. $lbl(a_i) \succ lbl(a_j)$ if and only if one of the following cases holds.*

(1) $h_i > h_j$
(2) $h_i = h_j$ *and* $w_i > w_j$

Definition 13 *Let P_1 and P_2 be two blocks respectively composed of the sets A_1 and A_2. Let a_j^i be the i^{th} element of A_j. Without loss of generality we*

suppose that the a_j^i are sorted by decreasing labels. $lbl(A_1) \succ lbl(A_2)$ if and only if

(1) $a_1^i = a_2^i$ for $i = 1, \ldots, |A_2|$ and $|A_1| > |A_2|$
(2) there exists i such that $lbl(a_1^i) \succ lbl(a_2^i)$ and no $k < i$ such that $lbl(a_2^k) \succ lbl(a_1^k)$.

Note that due to the definition $lbl(A_j) \succ lbl(\emptyset)$ for all A_j.

Proposition 14 *Let (A, B) be a 2OPP instance and P a configuration composed of two or more independent blocks $P_1, P_2, \ldots, P_k$. If P is feasible then there is an equivalent feasible configuration P' composed of the same independent blocks sorted in decreasing order of $lbl(P_j)$.*

PROOF. *Immediate.*

If two consecutive blocks are not sorted by decreasing values of lbl, the enumeration stops. A simple method to ensure that the blocks can be sorted by decreasing values of P_j after the construction of a given block P_1 is to check that there is no unpacked item a_i such that $lbl(a_i) \succ lbl(a_k)$ for all items a_k of P_1. For the left-hand configuration of Figure 10, as soon as the x-coordinates of 1, 2 and 3 are selected, the enumeration stops. Indeed as 4 is the tallest item, subsequent blocks cannot be sorted by decreasing values of $lbl(P_j)$.

5.2 Pseudo-symmetry

In the classical method, if two configurations are symmetrical with respect to the y-axis only one has to be enumerated. If this configuration does not respect the leftmost-downward rule, it is not enumerated either.

In the outer branch&bound method, there is no real axial symmetry, as the y-coordinates are not fixed. One way of avoiding exploring symmetrical configurations is to select two items a_1^* and a_2^* before the enumeration, and then we enumerate only solutions where $x_1^* \leq x_2^*$. If a_2^* is packed before a_1^*, a cut is made. As we are dealing with pseudo-symmetry instead of real symmetry, a problem can occur. Suppose two feasible pseudo-symmetrical configurations are such that $x_1^* > x_2^*$. This can happen because a translation is applied after the symmetry. In this case, no configuration is enumerated. So, we have to verify that such a case cannot occur. We only apply the method if $h_1^* + h_2^* > H$. Indeed if the items are translated, their respective positions cannot be modified without overlapping.

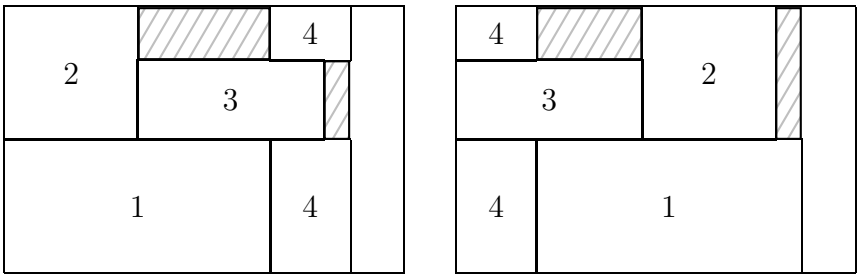

Fig. 11. Pseudo-symmetry

Proposition 15 *Let (A, B) be a 2OPP instance and a_1^*, a_2^* two items of A, $h_1^* + h_2^* > H$. If there is a non-dominated feasible packing for A in B, there is a non-dominated feasible packing for A in B such that $x_1^* \leq x_2^*$.*

PROOF. *Suppose there is a feasible packing P for A in B. If $x_1^* \leq x_2^*$, the proposition is true. If $x_1^* > x_2^*$, a (possibly dominated) packing P' can be obtained by applying an axial symmetry to this configuration. In this configuration, $x_1^* < x_2^*$. The corresponding non-dominated configuration P'' can be obtained by translating items to the left or downward. As $h_1^* + h_2^* > H$, the translation cannot change the relative position of a_1^* and a_2^*. So P'' is a non-dominated packing such that $x_1^* \leq x_2^*$.*

Note that if a_1^* and a_2^* are not chosen such that $lbl(a_2^*) \succ lbl(a_1^*)$, this method can be used with the method which considers the block redundancies. Moreover the pseudo-symmetry can also occur inside a block. In this case a slightly different procedure can be applied. The reference items a_i and a_k are the two lexicographically highest items of the block. The configuration retained is when $x_i \leq x_j$.

6 Computational experiments

We generated our own benchmarks in order to obtain instances which are difficult to solve (*i.e.* where the lower and upper bounds are not equal). The first results obtained on randomly generated instances show that $TSBP$ dramatically reduces the computing time in many cases compared to $LMAO$ and MV. It solves instances which cannot be handled by the other methods. Then we studied the efficiency of $TSBP$ for a large number of benchmarks generated according to two parameters: the number of items and the total area of the items. We also tested the reduction procedures, the method to

avoid redundancies and our lower bounds within the enumeration. The reduction procedures and the method to avoid redundancies reduce the computing time, but it is not always interesting to use the lower bounds within the enumeration process. Finally we tested the optimized method against the method of Fekete and Schepers [5]. $TSBP$ solves instances which are not handled by the graph theoretical method.

All programs are implemented in C on a PC (Pentium IV 2,6 GHz). The results of Fekete and Schepers [5] have been computed by the authors with a PC (Pentium IV 3 Ghz). The limit of time given for one instance is 15 minutes. If the method tested is not able to find the solution within this time, we consider it cannot solve the problem. In each table, two results are reported: the number of nodes enumerated and the time needed to solve the problem. All computing times are reported in seconds. For each instance, we report its feasibility (F, N for feasible, and non-feasible). Symbol $-$ means that the method is not able to solve the instance within 15 minutes.

6.1 Results for the enumerative methods

The discrepancy between the area of the bin and the total area of the items is an important parameter for the difficulty of an instance. We denote this parameter by ϵ, $\sum_{a_i \in A} w_i h_i = (1-\epsilon)WH$. If they are equal, a non-feasible configuration may be detected easily with the lower bounds. On the other hand, if the discrepancy is too large, a solution may also be easily obtained. In order to compare the efficiency of our methods with the classical method, we generated a set of difficult $2OPP$ instances with bin of size $(20, 20)$. The difficulty of an instance is determined by parameter ϵ and the number of items n. In the next section, we analyze the influence of ϵ on the difficulty of the benchmarks. As our bounds detect a large number of non-feasible configurations, we only report results for instances which are not closed by the lower bounds.

The purpose of Table 1 is to show that our branching scheme is dramatically more efficient than the scheme of Martello and Vigo. For each instance, the results are reported for three methods: the classical method MV, the new branch&bound method $LMAO$ and the two-step method $TSBP$ (Table 1). The reduction procedures are launched at the root node. No lower bounds or methods to avoid redundancies are used, even at the root node, so the results only depend on the branching scheme.

$TSBP$ is dramatically more efficient for the non-feasible instances. Practically speaking the inner branch&bound method is rarely launched. For two *easy* feasible instances $LMAO$ is more efficient than $TSBP$ (for these instances, a

Instance			Nodes			CPU Time		
ϵ	n	feas.	MV	$LMAO$	$TSBP$	MV	$LMAO$	$TSBP$
00	10	N	498940	914	386	6	0	0
00	15	N	44108432	6972444	45016	-	87	0
00	23	N	56882064	67408024	40569046	-	-	254
00	23	N	42608212	68155560	154997737	-	-	-
02	17	F	52758236	71765888	1706527	-	-	8
02	20	F	43113104	58100032	70690	-	-	0
02	22	F	1736591	58108864	108	28	-	0
02	20	N	53653556	73548088	171891273	-	-	-
03	10	N	4389743	198654	17122	55	2	0
03	15	N	54482588	79027208	22580963	-	-	90
03	16	N	46469712	68238856	24091255	-	-	106
03	17	N	41706968	73210184	147246675	-	-	720
03	18	N	43368960	5001083	2621993	-	67	12
04	15	F	48638400	66949008	4249	-	-	0
04	17	F	65952336	29908064	15937	-	377	0
04	19	F	57670816	5423174	25300	-	58	0
04	20	F	58401756	33	195	-	0	0
04	15	N	54766908	71919648	3340753	-	-	12
04	17	N	62933956	77090992	80888040	-	-	379
04	18	N	61351128	71299008	76571868	-	-	364
05	15	F	68788024	49042968	82098	-	523	0
05	18	F	59957500	73878872	4249532	-	-	29
05	20	F	23184128	68833	24836	254	1	0
05	15	N	63948032	84383344	13465274	-	-	50
05	17	N	51738224	78107640	44862814	-	-	209
05	15	N	63457040	80207048	211846864	-	-	-
07	15	F	65526928	20388888	822	-	219	0
07	10	N	5377856	424144	83795	63	3	0
07	15	N	57912524	83727840	12420952	-	-	53
07	15	N	60273792	79483544	221997738	-	-	-
08	15	F	67319104	21165528	50689	-	218	0
08	15	N	60148728	91504040	38186062	-	-	148
10	10	N	16777620	741677	27644	200	6	0
10	15	N	68136960	92775504	29091982	-	-	104
10	15	N	54992568	77875552	259205325	-	-	-
13	10	N	37400760	2606968	164016	411	22	0
13	15	N	87402296	4595959	451458	-	35	1
13	15	N	44129192	86488752	259296268	-	-	-
15	10	N	84001872	7191894	110200	-	59	0
15	15	N	70194208	88272600	6287316	-	-	22
20	15	F	41817	263837	81	1	3	0
20	15	F	50799084	72909992	7781	-	-	0

Table 1
Comparison between the three enumerative methods

solution is found in one of the first branches of the tree). It can be explained by the fact that the two-step method has to use two small branch&bound methods instead of only one in the first method. For the feasible instances for which the solution is harder to find, $TSBP$ is more efficient. The results confirm that the classical method MV, to be efficient, has to include many methods to reduce the number of redundancies which occur (see Scheithauer [14], for example). A large number of these redundancies are handled by $LMAO$ and $TSBP$.

6.2 Tuning $TSBP$

As we only consider x-coordinates in the first step of $TSBP$, the number of nodes enumerated can be different if the role of the x-coordinates and the y-coordinates are exchanged. It corresponds to a rotation of the problem of angle $\frac{\pi}{2}$. The instance obtained is feasible if and only if the original instance is feasible. The problem is to determine which orientation minimizes the computing time needed. The method we use is the following: for each orientation, we compute the number of nodes generated for the first level. This number is equal to the number of possible sets A_j to pack at the first x-coordinate. This is not an optimal decision rule, but in most of the cases it allows us to find the best orientation for the problem. So for MV and $LMAO$ we solve the problem for the original instance, whereas for $TSBP$ we consider in several cases the instance obtained after rotation. The computing time of the decision procedure is included in the computing time of $TSBP$.

Our method may not perform well when there is a small non feasible subset of items, we apply the following method: we first solve the instance related to the two largest items, and then recursively add smaller items one by one while the obtained instance is feasible. In several cases, small subsets are sufficient for the instance not to be feasible. The drawback of this method is to increase the computing time for feasible instances.

6.3 Benchmarks: computational analysis

For our tests we generated benchmarks according to parameter ϵ. The method we used to generate benchmarks has similarities with the method of Hopper and Turton [16]. The idea is to obtain both feasible and non-feasible problem instances. The first step of the algorithm consists in randomly generating a set of values whose sum is equal to $(1-\epsilon)WH$. These values are the areas of the items in the created instance. Then the values are factorized to get the width and the height of the items.

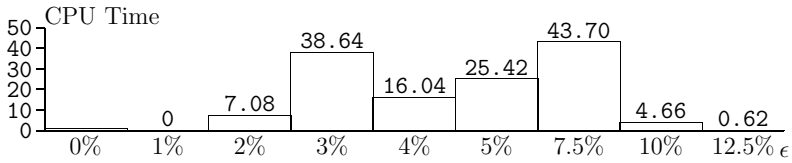

Fig. 12. Difficulty of the feasible benchmarks for $n = 15$

Table 2, Figure 12 and Figure 13 illustrate the fact that the value of ϵ affects the difficulty and even the solvability of the instances. For $\epsilon = 0\%$ no feasible instances were generated, and for $\epsilon = 50\%$ and $\epsilon = 75\%$ no non-feasible instances were generated. These cases are shown in Table 2 as $n.s.i$ (no such instances). For each benchmark, the bin is of size $(20, 20)$ (area: 400) and the number of items n is equal to 15. For each value of ϵ we report the number of nodes visited on average to determine the exact solution of the 50 first instances generated. The method used is $TSBP$ applied after the reduction procedures and the lower bounds. The method to avoid redundancies is not used within $TSBP$.

	Non-Feasible		Feasible		Feasibility		
ϵ	nodes	CPU time	nodes	CPU time	N	F	X
0	20102.34	0.22	n.s.i.	n.s.i	50	0	0
1%	448220.88	8.29	14104.00	0.00	49	1	0
2%	1782186.38	37.08	378055.94	7.08	37	13	0
3%	3310969.25	69.79	1929031.00	38.64	28	22	0
4%	7156912.00	147.28	865186.56	16.04	25	24	1
5%	4581232.00	96.27	1352387.50	25.42	11	38	1
7,5%	2192336.50	38.33	2566824.25	43.70	9	37	4
10%	0.00	0.00	503232.06	4.66	7	41	2
12,5%	0.00	0.00	62060.40	0.62	3	45	2
25%	0.00	0.00	216222.69	1.51	3	47	0
50%	n.s.i	n.s.i.	218606.52	1.41	0	49	1
75%	n.s.i	n.s.i.	580315.25	3.60	0	50	0

Table 2
Difficulty of the benchmarks for $n = 15$ and different values of ϵ

The more difficult instances in average are generated with $\epsilon = 3\%$ or $\epsilon = 5\%$ for non-feasible configurations. For the feasible configurations, the results are less clear, but the more difficult instances are related to values of ϵ between 3% and 7.5%. When $\epsilon = 0$ the bounds work very well, as few branch&bound methods are actually launched. It also appears that although the average CPU

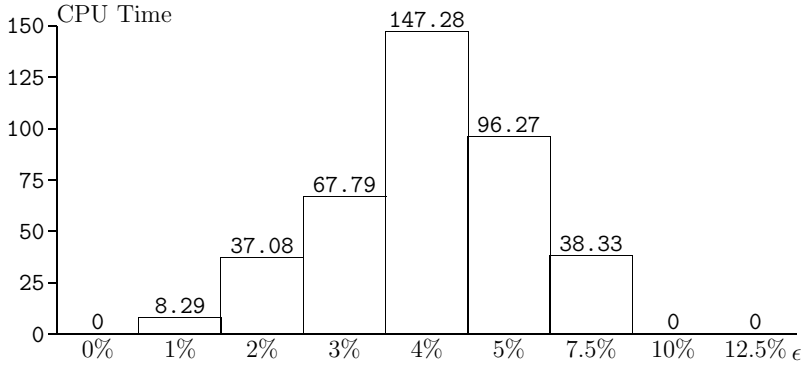

Fig. 13. Difficulty of the non-feasible benchmarks for $n = 15$

time used is smaller for values near 10%, the number of unsolved instances is larger. For large values of ϵ the problem becomes easy.

6.4 Reduction Procedures and methods for pruning the search tree

We have added several procedures to reduce the computing time of the method: the reduction procedures described in Section 2 (rp), the methods to handle redundancies (rh) and our lower bounds (lb). Together with the methods to handle redundancies, we check that each remaining item can be packed in the current available areas. We sum up the results in Table 3.

The reduction procedures are run on the initial instance as a preprocessing, and included in the lower bounds. When an instance is modified by a (D)DFF, the reduction procedures are run on the residual instance. The computing time is reduced by a wide range for many test cases. The procedures which handle redundancies in the tree search decrease the number of explored nodes for a small algorithmic cost, so they also reduce the computing time for most of the instances. The reduction depends on the instance: it can be considerable in some cases, and marginal for others. The lower bounds are useful to reduce the number of nodes in the tree search, but their computing time is large. So they may increase the overall computing time of the method. However in several cases the time needed for the method to find a solution is reduced when the lower bounds are used, so they can be useful.

6.5 Comparison with the method of Fekete and Schepers [5]

In Table 4, we compare our results to the method of Fekete and Schepers [5]. These results were sent by J. Van der Ween and S. Fekete. They were obtained by the authors using a Pentium IV 3 Ghz. The two methods are complemen-

Instance			Nodes				CPU Time			
ϵ	n	feas.	init	rp	rp+rh	rp+rh+lb	init	rp	rp+rh	rp+rh+lb
00	10	N	1	1	1	1	0	0	0	0
00	15	N	273008	181697	110943	96920	0	1	1	2
00	23	N	-	39240877	9342562	5968406	-	258	66	86
00	23	N	-	-	13383832	9057985	-	-	118	289
02	17	F	2969891	1814574	1171696	784796	23	9	6	12
02	20	F	608735	609161	487753	487230	10	14	12	12
02	22	F	175426	175426	175380	174943	1	4	4	4
02	20	N	1106	1	1	1	0	1	1	1
03	10	N	3100	3100	1222	417	0	0	0	0
03	15	N	5119	5119	4169	3707	0	1	1	1
03	16	N	4251766	4251766	1771831	1592400	15	17	9	32
03	17	N	242635	508053	343290	313007	0	3	2	4
03	18	F	3467378	3418946	2719210	2605815	16	18	17	22
04	15	F	8178	9053	6698	3949	0	1	1	1
04	17	F	2328156	2328156	2103026	1942682	24	27	25	26
04	19	F	2114390	2114390	1465752	1075159	8	11	8	7
04	20	F	6673	6673	6163	5876	0	3	3	3
04	15	N	63207	63207	43256	42844	0	1	1	1
04	17	N	13889	1	1	1	0	1	1	1
04	18	N	-	7315303	1689391	434824	-	36	11	7
05	15	F	503436	503436	382690	334434	2	3	2	3
05	18	F	62890632	62804148	40122815	20245458	300	293	220	126
05	20	F	39429	39429	39429	39387	0	2	2	2
05	15	N	61379	1	1	1	0	0	0	0
05	17	N	5393	2158	1429	993	0	1	1	1
05	15	N	158	1	1	1	0	0	0	0
07	15	F	232534	232464	129878	90219	0	1	1	1
07	10	N	5363	5206	1903	1758	0	0	0	0
07	15	N	1	1	1	1	0	0	0	0
07	15	N	1180	1180	677	651	0	1	1	1
08	15	F	32853635	32901221	27666296	22658934	130	128	130	117
08	15	N	303	414	267	261	0	1	1	1
10	10	N	153	1	1	1	0	0	0	0
10	15	N	1085	1	1	1	0	0	0	0
10	15	N	33545	33545	17618	17603	0	1	1	1
13	10	N	3846	3846	1889	1468	0	0	0	0
13	15	N	4233	93	91	91	0	0	0	0
13	15	N	262	1	1	1	0	0	0	0
15	10	N	621	621	331	331	0	0	0	0
15	15	N	4293	3396	1117	1117	0	0	0	0
20	15	F	867	747	747	747	0	1	1	1
20	15	F	4607934	4355764	4355764	4355492	34	34	36	44

Table 3
Reducing the search tree

tary, as they perform differently depending on the instance. It appears that the graph theoretical method fails to find a solution in the 15 minutes allowed for three instances. No results were found after one hour for $\epsilon = 0$ and $n = 23$, and 1200 secondes were needed to find a solution for $\epsilon = 2$, $n = 20$. However the method of Fekete and Schepers remains of great interest, as it outperforms our method for several instances (see $\epsilon = 7$, $n = 15$ and $\epsilon = 13$, $n = 15$, for example). The computing time of each node of the tree search is large, but the size of the tree is smaller in many cases. It seems that the graph theoretical method performs well compared to $TSBP$ when a small subset of items is sufficient for the instance to be non feasible. This can be explained by the fact that the graph-theoretical branching scheme consider pairs of items and may concentrate the search on the more constrained items, whereas our method has to deal with all items, even the smaller. In the cases where all items are needed (the two instances with $n = 23$, for example), our branching scheme is competitive.

7 Conclusion

We propose new reduction procedures and a new exact method in two phases to pack a set of rectangles in a rectangular bin. Practically speaking, using the outer branch&bound method decreases the number of nodes visited by a wide range compared to the method of Martello and Vigo. We also propose methods to handle a part of the redundancies which occur in the branch&bound method. The lower bounds we propose may increase the computing time of the method for some instances, but may also be useful. Work should be done to reduce their computing time. Our method is complementary with the method of Fekete and Schepers, as it returns better results for several instances. It seems that the graph theoretical model is not the only one to lead to interesting results and other ways can still be explored to improve results for $2OPP$.

Acknowledgements

We would like to thank Prof. Fekete and Jan Van der Ween who run their $2OPP$ program on our benchmarks and sent us the results.

References

[1] M. R. Garey, D. S. Johnson, Computers and intractability, a guide to the theory of NP-completeness, Freeman, New York, 1979.

Instance			Nodes		CPU Time	
ϵ	n	feas.	Graph	TSBP	Graph	TSBP
00	10	N	1	1	0	0
00	15	N	127	96920	0	2
00	23	N	-	5968406	-	**86**
00	23	N	-	9057985	-	**289**
02	17	F	28631	784796	**7**	12
02	20	F	-	487230	-	**12**
02	22	F	190617	174943	167	**4**
02	20	N	1	1	0	1
03	10	N	1	417	0	0
03	15	N	13	3707	0	1
03	16	N	9891	1592400	**2**	32
03	17	N	431	313007	0	4
03	18	F	574	2605815	**0**	22
04	15	F	933	3949	0	1
04	17	F	20270	1942682	**13**	26
04	19	F	786057	1075159	560	**7**
04	20	F	22796	5876	22	**3**
04	15	N	35	42844	0	1
04	17	N	1	1	0	1
04	18	N	24593	434824	10	**7**
05	15	F	1410	334434	0	3
05	18	F	262	20245458	**0**	126
05	20	F	547708	39387	491	**2**
05	15	N	1	1	0	0
05	17	N	1	993	0	1
05	15	N	18369	1	2	0
07	15	F	92	90219	0	1
07	10	N	17	1758	0	0
07	15	N	1	1	0	0
07	15	N	61	651	0	1
08	15	F	433	22658934	**0**	117
08	15	N	1	261	0	1
10	10	N	5	1	0	0
10	15	N	77	1	0	0
10	15	N	7	17603	0	1
13	10	N	17	1468	0	0
13	15	N	1	91	0	0
13	15	N	1	1	0	0
15	10	N	25	331	0	0
15	15	N	1	1117	0	0
20	15	F	325	747	0	1
20	15	F	36	4355492	**0**	44

Table 4
Comparison with the method of Fekete and Schepers

[2] S. Martello, D. Vigo, Exact solution of the two-dimensional finite bin packing problem, Management Sci. 44 (1998) 388–399.

[3] M. Boschetti, A. Mingozzi, The two-dimensional finite bin packing problem. part I: New lower bounds for the oriented case, 4OR 1 (2003) 27–42.

[4] M. Boschetti, A. Mingozzi, The two-dimensional finite bin packing problem. part II: New lower and upper bounds, 4OR 1 (2003) 135–147.

[5] S. Fekete, J. Schepers, An exact algorithm for higher-dimensional orthogonal packing., Revised for Operations Research.

[6] J. Carlier, F. Clautiaux, A. Moukrim, The two-dimensional bin-packing problem. New reduction procedures and lower bounds, Submitted.

[7] S. Fekete, J. Schepers, New classes of fast lower bounds for bin packing problems, Mathematical Programming 91 (2001) 11–31.

[8] A. Caprara, M. Locatelli, M. Monaci, Bilinear packing by bilinear programming, in: IPCO XI, 2005.

[9] J. O. Berkey, P. Y. Wang, Two-dimensional finite bin-packing algorithms, Journal of Operational Research Society 38 (1987) 423–429.

[10] A. Lodi, S. Martello, D. Vigo, Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems, INFORMS J. Comput. 11 (1999) 345–357.

[11] A. Lodi, S. Martello, M. Monaci, Two-dimensional packing problems: a survey, European Journal of Operational Research 141 (2002) 241–252.

[12] A. Lodi, S. Martello, D. Vigo, Recent advances on two-dimensional bin packing problems, Discrete and Applied Mathematics 123 (2002) 379–396.

[13] E. Hadjiconstantinou, N. Christofides, An exact algorithm for general, orthogonal, two-dimensional knapsack problem, European Journal of Operational Research 83 (1995) 39–56.

[14] G. Scheithauer, Equivalence and dominance for problems of optimal packing of rectangles, Ricerca Operativa 83 (1998) 3–34.

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

[16] E. Hopper, B. C. H. Turton, Problem generators for rectangular packing problems, Stud. Inform. Univ. 2 (1) (2002) 123–136.

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

[18] S. Fekete, J. Schepers, A general framework for bounds for higher-dimensional orthogonal packing problems, Mathematical Methods of Operations Research 60 (2004) 311–329.

[19] D. S. Johnson, Near optimal bin packing algorithms, dissertation, Massachussetts Institute of Technology, Cambridge, Massachussetts (1973).

[20] G. S. Lueker, Bin packing with items uniformly distributed over intervals [a,b], in: Proc. of the 24th Annual Symposium on Foundations of Computer Science (FOCS 83), IEEE Computer Society, 1983, pp. 289–297.

[21] N. Christofides, C. Whitlock, An algorithm for two-dimensional cutting problems, Operations Research 25 (1977) 30–44.

[22] G. Scheithauer, LP-based bounds for the container and multi-container loading problem, International Transactions in Operational Research 6 (1999) 199–213.

[23] A. Amaral, A. N. Letchford, An improved upper bound for the two-dimensional non-guillotine cutting problem, submitted to publication.

[24] M. Boschetti, E. Hadjiconstantinou, A. Mingozzi, New upper bounds for the two-dimensional orthogonal non guillotine cutting stock problem, IMA Journal of Management Mathematics 13 (2002) 95–119.