Journal of Mathematical Modelling and Algorithms in Operations Research manuscript No.
(will be inserted by the editor)

A new search procedure for the two-dimensional orthogonal packing problem

S. Grandcolas · **C. Pinto**

Received: date / Accepted: date

Abstract In this paper we propose a new exact procedure for the two-dimensional orthogonal packing problem, based on F. Clautiaux et al. approach [4]. The principle consists in searching first the positions of the items on the horizontal axis, so as that, at each position, the sum of the heights of the items does not exceed the height of the bin. Each time a valid placement of all the items is encountered, another procedure determines if it can be extended to a solution of the packing problem, searching the positions of the items on the vertical axis. Novel aspects of our approach include a simple and efficient search procedure, which only generates restricted placements, at least in a first stage, in order to reduce the search space, and the memorization of unsuccessful configurations, which are then used to detect dead-ends. We tested our implementation on a selection of orthogonal packing problems and strip packing problems, and we compared our results with those of recent successful approaches.

Keywords Orthogonal packing · strip packing.

1 Introduction

In a two-dimensional space, the Orthogonal Packing Problem (OPP-2) consists in packing a set of rectangles inside a given rectangular bin, so as no two rectangles overlap. The rectangles are defined by their widths and heights, and their orientations are fixed (their edges must be orthogonal to the edges of the bin). This problem is NP hard. In many recent approaches, the positions of the items are considered independently in each dimension. A global constraint ensures that the items do not overlap simultaneously in all the dimensions. In this paper we focused on this type of approaches.

This work has been supported by the Region Provence-Alpes-Cote-d'Azur and the ICIA Technologies company `http://www.iciatechnologies.com`.

S. Grandcolas
LSIS – UMR CNRS 7296 Avenue Escadrille Normandie-Niemen, F-13397 Marseille Cedex 20 - France
E-mail: stephane.grandcolas@lsis.org

C. Pinto
LSIS – UMR CNRS 7296 Avenue Escadrille Normandie-Niemen, F-13397 Marseille Cedex 20 - France
E-mail: cedric.pinto@lsis.org

S. P. Fekete and J. Schepers first proposed a modelization based on interval graphs [8]. For each dimension of the space a graph represents the intersections of the projections of the items on the axis corresponding to this dimension. If

- each graph is an interval graph,
- in each graph no stable set has a weight greater than the size of the bin in the corresponding dimension,
- no edge occurs simultaneously in all the graphs, that is no two items intersect in all the dimensions,

then the problem is solvable and a solution can be generated from the graphs. Since neither the effective positions nor the relative positions of the items are fixed, the same tuple of graphs can represent many different packings. The approach can be generalised to higher-dimensional spaces. S. P. Fekete and J. Schepers developed a method based on this modelization [9]. The principle consists in enumerating all the possible tuples of graphs, looking for a tuple that satisfies the three properties. Starting with empty graphs, the procedure explores a search tree, in which at each node a new edge is inserted in one of the graphs. While adding new edges, the graphs can loose or retrieve the interval graph property, and then this property must be tested at each node. As well, each time a new edge is added, the maximal weights of the stable sets must be calculated in the graphs which violate the stable sets property. These verifications require expensive computations. However, using relevant bounds, S. P. Fekete and J. Schepers obtained very good results compared to the best known classical approaches. In a two-dimensional space, in the worst case the cost of the search is $O(2^{n^2})$ if there are n items. Recently, S. Grandcolas and C. Pinto proposed a SAT encoding of S. P. Fekete and J. Schepers modelization [10]. To eliminate the unfeasible stable sets, their idea consists in adding, for each stable set, a clause to force at least two intervals of the set to overlap. The problem is that the number of clauses grows exponentially with the number of items, making this encoding impractical when there are many items.

F. Clautiaux et al. [4] proposed an exact method in two-dimensional spaces named TBSP, for *two-step branching procedure*. The method consists in solving with a branch and bound a relaxation of the packing problem in which the vertical positions of the items are ignored, but the cumulated heights of the items which overlap the same position are constrained not to exceed the height of the bin. This is called the *outer* phase. Each time a solution of the relaxed problem is discovered, an *inner* branch and bound procedure determines if it can be extended to a solution of the packing problem by searching for the vertical positions of the items. If this *inner* phase fails, the search continues. In most cases the solutions of the relaxed problem are easily extended to global solutions. Then, in general, the total computation time corresponds to the time to solve the relaxed problem. A solution of the relaxed problem can represent many different packings, since the vertical positions of the items are ignored. This is not comparable to S. P. Fekete and J. Schepers modelization, but the search space during the outer phase is then strongly reduced. Using reduction procedures and relevant bounds F. Clautiaux et al. obtained good results on a selection of problems that they generated. Note that the relaxed problem is equivalent to a single resource scheduling problem with no precedence constraints, and techniques from this field can be transposed in the domain of packing.

In a more recent paper, F. Clautiaux et al. [5] describe a constraint-based scheduling model for the orthogonal packing problem. Their idea is to represent the relaxed problem defined in [4] together with constraints which involve the vertical positions of the items. Energetic reasoning, a concept developed for scheduling problems, helps to detect the unfeasability and to derive necessary conditions. They also compute minimal bounds of the

unrecoverable area solving subset-sum problems, to verify that there is enough space for the items when the positions of some items are fixed or strongly constrained. Integrating these techniques in a branch-and-bound algorithm F. Clautiaux et al. obtained very good results.

Very interesting results have been published the last few years. For perfect packing problems, M. Kenmochi et al. [15] proposed several branch and bound algorithms. Solutions are built by packing items one at a time in the bin. The best results were obtained building *staircase placements*, that is placements of the items in the bin such that the upper borders of the top items are even lower than the items are on the right. This approach can also be used to solve strip packing problems. Most recently J.F. Côté, M. Dell' Amico et M. Lori [6] developed an innovative two-phases approach for the strip packing problem. During phase one, a first attempt to solve a relaxation of the problem is made, using a branch and bound algorithm. If no solution is found, and the unfeasability cannot be proved before a given limit of the number of expanded nodes is reached, then a second attempt is made using Benders' decomposition, a modelisation based on the solution of a linear model. Relying on CPLEX to solve the linear program, this approach gives impressive results.

We propose in this paper an exact method for the two-dimensional orthogonal packing problem based on the approach of F. Clautiaux et al. [4]. The core of the method is a procedure that searches for the horizontal positions of the items, that we call *placements*. This procedure explores the search space in a very efficient way, generating only placements in which no item, which is not already at the leftmost position in the bin, can be moved immediately to the left without violating the height constraint. In some cases there exists no solution of the packing problem that corresponds to such placements, and then some placements that do not satisfy the property must also be generated. Additional techniques help to avoid useless explorations. At each node of the search tree, if the exploration does not succeed, a two-dimensional representation of the remaining free vertical space is memorized. We call it the *profile* of the placement. If later a configuration with a similar or worse profile is encountered, the branch will be abandoned. Profiles are also used for calculating minimal bounds of the wasted space with a knapsack algorithm.

The paper is organized as follows: in the next section we define canonical placements, which represent the positions of the items on the axis. In the third section we describe the search procedure. In the fourth section we present additional techniques to improve the search. Finally, in the fifth section, we report the results of our approach on classical problems, and we compare its performances with those of the best known approaches.

2 Placements

Given a rectangular bin of width W and height H, and a set of n rectangular items of widths $w_1, \ldots, w_n$ and heights $h_1, \ldots, h_n$, the orthogonal packing problem consists in determining if the items can be packed in the bin with their sides parallel to the sides of the bin, in such a way that no two items overlap. The positions of the items in the bin are defined by their positions on the x-axis and on the y-axis (the x-axis corresponds to the width of the bin and the y-axis to the height). The position 0 on the x-axis corresponds to the left side of the bin, and the position 0 on the y-axis corresponds to the bottom of the bin. The widths, the heights and the positions of the items are integers. Since the width of the area occupied by two items packed side by side is the sum of their widths, we can consider that the right side of the first one and the left side of the second one are at the same position. In fact the projection of item i on the x-axis is the interval $[p, p+w_i[$ if p is the position of i on the x-axis and w_i is

its width. Similarly the projection of item i on the y-axis is the interval $[p, p+h_i[$ if p is the position of i on the y-axis and h_i is its height.

In this section we describe how the positions of the items on the two axis are represented, and we introduce some constraints on the positionnement of the items which aim at reducing the size of the search space.

Definition 1 [Placements]. A placement on the x-axis is a sequence $P = ((i_1, p_1), \ldots, (i_m, p_m))$ where $i_1, \ldots, i_m$ are items and $p_1, \ldots, p_m$ are integers, such that, for each pair (i_j, p_j), $p_j + w_{i_j} \leq W$ holds, and either $p_j = 0$, or there exist $k,\ 1 \leq k \leq m$, such that $p_j = p_k + w_{i_k}$. Placement P satisfies the *height constraint* if at each position p_j on the x-axis, $H_P(p_j) \leq H$, where $H_P(p) = \sum_{k \in \{1,\ldots,m\}/p_k \leq p < p_k + w_{i_k}} h_{i_k}$.
Similarly, a placement on the y-axis is a sequence $P = ((i_1, p_1), \ldots, (i_m, p_m))$ such that, for each pair (i_j, p_j), $p_j + h_{i_j} \leq H$ holds, and either $p_j = 0$, or there exist $k,\ 1 \leq k \leq m$, such that $p_j = p_k + h_{i_k}$. Placement P satisfies the *width constraint* if at each position p_j, $W_P(p_j) \leq W$, where $W_P(p) = \sum_{k \in \{1,\ldots,m\}/p_k \leq p < p_k + h_{i_k}} w_{i_k}$.

Placements are used to represent the positions of the items in the bin. The items are necessarily inside the bin, either at the position 0 or side by side with another item. This idea was introduced by J. C. Herz [11] for the two-dimensional stock cutting, and is also well-known in the domain of scheduling (see Stinson J. et al. [18]). Packings of this type are called *gapless packings* (S. P. Fekete and J. Schepers [8]).

Definition 2 [Consistent placements]. If P_x is a placement of the items on the x-axis, and P_y is a placement of the items on the y-axis, P_x and P_y are *consistent* if and only if, for any two items i and j, if (x_i, y_i) and (x_j, y_j) denote the positions of i and j in P_x and P_y

$$x_i + w_i \leq x_j \text{ or } x_i \geq x_j + w_j \text{ or } y_i + h_i \leq y_j \text{ or } y_i \geq y_j + h_j$$

Consistent placements represent the gapless solutions of packing problems. Indeed, each item is inside the bin (from the definition of the placements), and no two items overlap (from the definition of consistent placements). Then, searching for a solution of a packing problem amounts to searching a placement of the items on the x-axis P_x and a placement of the items on the y-axis P_y which are consistent. Moreover P_x and P_y satisfy necessarily the height and width constraints (if P_x does not satisfy the height constraint then there exists a set of items which overlap each others in P_x and which cannot be positionned on the y-axis inside the bin with no overlapping). From now, any placement to which we will refer is supposed to satisfy the height or width constraint depending on the axis.

Definition 3 [Canonical placements]. A placement $P = ((i_1, p_1), \ldots, (i_m, p_m))$ is *ordered* if, for each pair (i_j, p_j) with $j < m$, $p_j \leq p_{j+1}$, and if $p_j = p_{j+1}$ then $i_j \prec i_{j+1}$, where $\prec$ is a given *a priori* order on the items.
The placement P is *locally minimal* if, for each pair (i_j, p_j) such that $p_j > 0$, $H_P(p_j - 1) + h_{i_j} > H$ if P is a placement on the x-axis, and $W_P(p_j - 1) + w_{i_j} > W$ if P is a placement on the y-axis.
The placement P is *canonical* if P is ordered, locally minimal, and P satisfies the height constraint (resp. the width constraint) if P is a placement on the x-axis (resp. on the y-axis).

The ordering constraint helps, while generating placements, to avoid mutiple generation of the same configuration (modulo a rearrangement of the sequence). A similar technique is used by S. Martello et al. [16] and later R. Alvarez-Valdes et al. [2] in their branch and bound algorithms, when successive choices can be done indifferently in any order. Local minimality constrains the placements of the items in one dimension, ignoring their positions in the other dimension. It is quite different from gapless packings. This concept is fairly close to the *left shift rule* proposed by E. Demeulemeester et al. in [7] in the domain of scheduling: if a task can be moved backward it must be.

Property 1 If there exists a placement P_x of the items on the x-axis which satisfies the height constraint, then there also exists a canonical placement of the items on the x-axis which satisfies the height constraint.

Proof. A canonical placement P'_x can easily be constructed from P_x. The processus consists first in reordering placement P_x so as that two consecutive pairs (i_j, p_j) and (i_{j+1}, p_{j+1}) satisfy $p_j < p_{j+1}$ or $p_j = p_{j+1}$ but $i_j \prec i_{j+1}$, then in shifting the items one after the other as much as possible on the left (that is while the height constraint is satisfied) considering the items in the order in which they occur in the placement, and finally in reordering again the placement, as in the first step. The placement P'_x obtained this way is canonical by construction, and P'_x *dominates* the placement P_x. The property also applies to placements on the y-axis.

Then any feasible orthogonal packing problem admits a canonical placement of the items on the x-axis. Indeed, there exists a placement of the items on the x-axis which corresponds to a solution, and then there also exists a canonical placement. This statement can help while searching consistent placements: if there exists no canonical placement of the items on the x-axis (or on the y-axis) then the problem is not feasible.

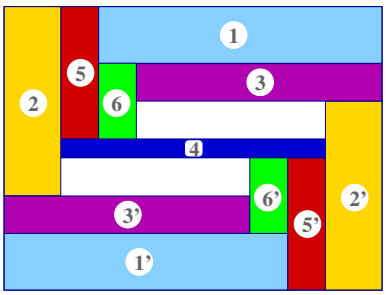

Fig. 1 A solution of a packing problem which does not correspond to a minimal placement

However a packing problem can be feasible, but have no solution whose projection on the x-axis (or the y-axis) is a canonical placement. Consider for example the problem with 11 items of sizes 15×3, 15×3, 3×10, 3×10, 13×2, 13×2, 14×1, 2×7, 2×7, 2×4 and 2×4, and a bin of size 20×15. There exists a unique solution (modulo an horizontal or vertical flip and the interchangeability of the identical items), depicted in figure 1. To be convinced of this, just note that (1) items 1, 2, 1' and 2' are engaged in pairs forming a circle, since three of them cannot overlap neither on the x-axis nor on the y-axis, and (2)

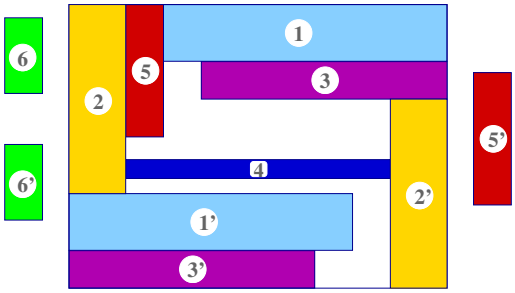

Fig. 2 Items 5 and 5' cannot be both packed above or under item 4

items 5 and 5' are on either side of item 4 (see figure 2). In this solution the position of item 6' is not minimal (there is enough vertical free space immediately to the left of item 6').

The simplest way to construct a placement consists in extending an initially empty sequence iteratively, adding new pairs $(item, position)$ at the end of the sequence. The addition of a new pair is called an *extension*. We will only consider the extensions which satisfy the height (or width) constraint and the ordering constraint, so as to only generate ordered placements which satisfy the height (or width) constraint. An extension is *locally minimal* if the position of the new item is either 0 or a position p such that the item could not be positioned at $p-1$ not violating the height constraint. A placement constructed with locally minimal extensions starting with an empty sequence is canonical. Remark that there can be no locally minimal extension with a given item i, even if there exists an extension (not locally minimal) with i. This situation occurs if, p_m being the position of the last pair of the placement, item i can be positioned at $p_m - 1$, whithout violating the height constraint. The potential extensions of a placement P with item i can be generated starting from the position p of the last pair of the placement (or the position 0 if P is empty), and enumerating the positions of the right sides of the items of P which are greater than p, in increasing order. The positions such that i should be outside of the bin or such that the height constraint is violated, are ignored. If p_{min} denotes the smallest of the remaining positions, if p_{min} is 0 or if $H_P(p_{min}-1) > H$ then p_{min} is the locally minimal extension of P with i. In the other case there is no locally minimal extension of P with i.

According to a well-known principle introduced by J. C. Herz [11], a feasible problem has solutions in which any item is moved as down and as left as possible. For consistent placements this idea translates into the following property.

Property 2 If P_x and P_y are consistent placements respectively on the x-axis and on the y-axis, then there exists a placement P'_y of the items on the y-axis such that:

- P_x and P'_y are consistent,
- for any item i, either i is at the position 0 in P'_y, or there exists an item j such that, if (x_i, y_i) and (x_j, y_j) are the positions of i and j in P_x and P'_y, $x_i \leq x_j < x_i + w_i$ or $x_j \leq x_i < x_j + w_j$, and $y_i = y_j + h_j$.

Proof. Placement P'_y is constructed from P_y, first decreasing the positions of the items as much as possible while no two items overlap simultaneously on both axis, considering the items in the order in which they occur in P_y, then in reordering the placement. P'_y satisfies necessarily the *width constraint*. Indeed, if C is a set of items which overlap one with any

other in P'_y, since no two items of C overlap in P_x the sum of their widths is less than the width of the bin.

3 Search procedure

The method that we propose is based on F. Clautiaux et al. approach [4]. It consists in searching all the placements of the items on the x-axis (the outer phase), and for each placement in searching a consistent placement on the y-axis (the inner phase). The inner phase is theoretically less costly, since the positions of the items on the x-axis have been already assigned, and then the search space is strongly reduced. The backtracking search procedure is described below (function `SearchPacking`). An innovative aspect of this procedure is, in the outer phase, the restriction of the search to canonical placements, at least in a first stage. Indeed, if some canonical placements on the x-axis are discovered but no consistent placement on the y-axis exists, then, to be complete, the procedure must also generate non-canonical placements.

Function `SearchPacking`(P, S, W, H)

```
begin
    if S = ∅ then
        if SearchVerticalPositions(P,W,H) =TRUE then
            return SUCCESS;
        else
            return NOCONSISTENT;
    if there exists (S,Φ) ∈ 𝒯 such that Φ ⊑ profile(P) then
        return FAILURE;
    if MaximalAvailableArea(P,S,W,H) < Area(S) then
        return FAILURE;
    M := LocallyMinimalExtensions(P,S,W,H);
    V := ∅;
    r := FAILURE;
    while M ≠ ∅ do
        (i, p) := SelectMin(M);
        M := M \ {(i, p)};
        V := V ∪ {i};
        status := SearchPacking(P ∪ {(i, p)}, S \ {i}, W, H);
        if status = SUCCESS then
            return SUCCESS;
        if status = NOCONSISTENT then
            r := NOCONSISTENT;
            M := M ∪ NextExtension(i, p, P, W, H);
    if r = FAILURE and V = S then
        𝒯 := 𝒯 ∪ {(S, profile(P))};
    return r;
end
```

The function `SearchPacking` has four parameters: P a placement of items on the x-axis, S a set of items which are not in P, W and H the width and height of the bin (the widths and heights of the items are omitted to simplify the function). The function returns SUCCESS if there exists a packing whose projection on the x-axis is an extension of P. Else, if there exist placements of all the items on the x-axis which extend P, but no consistent placement on the y-axis, then the function returns NOCONSISTENT. Finally, if there is no placement of all the items on the x-axis extending P, the function returns FAILURE. First the function is called with an empty placement and the set of items of the packing problem.

The function `SearchPacking` explores a tree, choosing at each node a new extension for the placement P. If the set S is empty (all the items are positioned in P), the function `SearchVerticalPositions` is called, to search a (not necessarily canonical) placement of the items on the y-axis which is consistent with P. If S is not empty (some items are not positioned on the x-axis), a loop iterates through the set M of the possible extensions of P with the items of S, which are locally minimal. For each extension (i, p) a recursive call determines wether there exists a packing including this extension or not. If the result is SUCCESS then the function returns SUCCESS: a packing exists which extends placement P. If the result is NOCONSISTENT, and if there are extensions of P with i at positions greater than p, then the function `NextExtension` returns the extension with the smallest position. This extension is necessarily not locally minimal. It is inserted in M, forcing the search to consider again the choice of item i to extend P. If there is no extension with i at a position greater than p, the function `NextExtension` returns the emptyset. Note that the placements obtained with the new extension cannot be generated with any other extension of M. Finally, if no recursive call of the function `SearchPacking` returns SUCCESS and if some placements containing all the items were discovered during the current exploration, then the function returns NOCONSISTENT, in order to indicate to the ancestor nodes that branches with non-minimal extensions must be eventually explored. If no placement on the x-axis has been found, the function returns FAILURE, and the triplet $(S, \text{profile}(P), V)$, where V is the set of the variables with which P has been extended in the current call, is added to the table $\mathscr{T}$, so as to avoid useless explorations if similar configurations are encountered (see next section).

The function `SearchVerticalPositions` follows the same scheme as the function `SearchPacking`, except that all the extensions are explored (even those which are not canonical) excluding those that are not consistent with the placement on the x-axis. However, it can be deduced from property 2 that it suffices to consider the vertical placements in which any item touches either the bottom side of the bin or the upper border of another item. The evaluation of an upper bound of the available area and the memorisation of the profiles are useless in this phase, and then searching the vertical positions can be very costly.

We tested several heuristics to choose the most promising extensions first. It seems that the best strategy consists in considering the extensions in increasing order of their positions, choosing at first the ones with the smaller ranks in case of equality. Indeed, the items are then positioned as much as possible on the left of the bin, and the wasted area should be limited. Furthermore, the extensions corresponding to identical items, that is items with the same width and the same height, are equivalent: if one of them turns out to be unsuccessful it will be the same with the others. Then, at each node of the search tree, no more than one extension for each class of identical items is to be considered. Finally, at each node of the search tree the function `MaximalAvailableArea` is called to compute an upper bound of the area which is available to pack the items of S, considering the placement P. If the result is less than the total area of the items of S, the current branch is abandonned. This evaluation is described in the next section.

4 Profiles

Placements can be represented in a two-dimensional space by contiguous vertical strips, delimited by the positions of the items on the x-axis, where the height of each strip is the sum of the heights of the items which overlap the strip (see the diagram on the left in figure 3). A *profile* is a slightly different representation which better reflects the wasted space corresponding to a placement.

Definition 4 [Profiles]. Given a non-empty ordered placement $P = ((i_1, p_1), \ldots, (i_m, p_m))$, if $r_1, \ldots, r_m$ are the positions of the right sides of the items in P sorted in increasing order of their values, and r_0 is the value 0, then the profile of P denoted profile(P) is the polygon defined by the set of points

$$\{(r_0,0),(r_0,e_1),(r_1,e_1),(r_1,e_2),(r_2,e_2),\ldots,(r_{m-1},e_m),(r_m,e_m),(r_m,e_{m+1})\}$$

where $e_{m+1} = 0$ and $\forall i,\ 0 < i \leq m$, if $r_i \leq p_m$ then $e_i = H$ else $e_i = max\{e_{i+1}, H_P(r_{i-1})\}$.

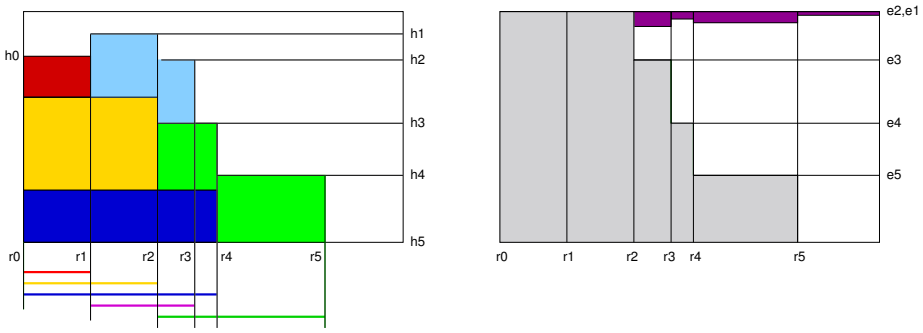

Fig. 3 A placement and its profile

Figure 3 shows the profile of a placement of a set of items. The profile is the light grey area in the drawing on the right, on the inside of the rectangle that represents the bin. It is composed of contiguous vertical strips with decreasing heights, and the strips at the left of the position p_m occupy the entire height of the bin. Profiles resemble Martello and al. envelopes [17], with the difference that envelopes are based on the positions of the items on both axis.

The free area above each strip of the profile can only be occupied by the items which are not yet in the placement. The combination of items which maximizes the vertical occupation can be calculated with a knapsack algorithm. It gives a lower bound of the wasted space (represented in figure 3 with violet rectangles at the top of the strips). Added to the surface areas of the items which are not already in the placement, it constitutes a lower bound of the free available space which is necessary to place the remaining items, and it helps to detect dead-ends while searching for a placement. The same computation can be performed in the horizontal direction, if we consider the profile as a series of slices stacked one on another. Combining the wasted areas in both directions, making sure not double counting the areas which could overlap, provides an even better lower bound.

Other approaches exploit similar evaluations. In their constraint programming model F. Clautiaux et al. [5] evaluate the wasted space on each portion of the x-axis on which the available height and the set of eligible items are constant. In their branch and bound algorithm M. Kenmochi et al. [15] use a bounding rule called *DP cut* which is based on the evaluation of the space that cannot be filled in *staircase placements*. For strip packing problems, R. Alvarez-Valdes et al. [2] proposed a lower bound, denoted L_7, which is evaluated using a knapsack algorithm. The bound is evaluated only once, before any item has been packed. An effort is made to reflect the fact that the same item cannot be used to maximise the occupation at each level in the bin.

Failures memorization

The result of the function `SearchPacking` only depends on the set of items S, on the profile of the placement P, and on the rank and position of the last item in P. If the result is FAILURE, that is if there is no canonical placement of the items of S on the x-axis extending P, a simple idea consists in memorizing these informations in order to avoid a useless exploration next time the same configuration is encountered. Similar techniques have been already experimented in the domain of scheduling (see for example the *cutset dominance rule* proposed by E. Demeulemeester et al. [7]). Intuitively, if P and Q are two placements of the same set of items, then if the profile of P is contained in the profile of Q, then P has more chances than Q to be extended to a placement of all the items. Unfortunately this is not true if we consider only locally minimal extensions. Figure 4 shows two placements P and Q of items i_1, i_3 and i_4. The profile of placement P is contained in the profile of placement Q. However, if the ordering of the items is $i_1 \prec i_2 \prec i_3 \prec i_4$, Q can be extended with item i_2, while there is no locally minimal extension of P with i_2, since i_2 should have the same position as i_3, and $i_2 \prec i_3$.

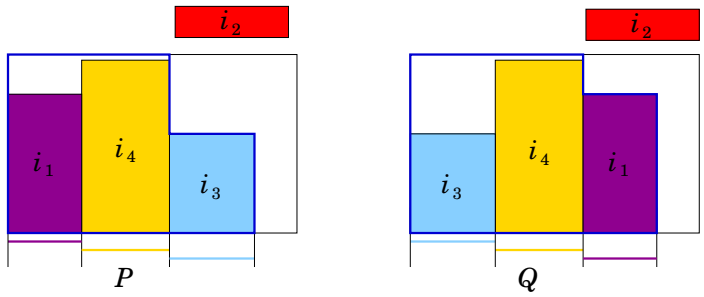

Fig. 4 The profile of P is contained in the profile of Q, but P cannot be extended with i_2

To work around this problem we propose to memorize unsuccessful configurations uniquely when there exists an ordered and locally minimal extension for each not-yet-placed item. Indeed, if P satisfies this condition, if there exists a sequence E_Q such that the concatenation of Q and E_Q, denoted $Q \cdot E_Q$, is a valid placement, then there exists a sequence E_P such that $P \cdot E_P$ is a valid placement. To simplify the proof we will first suppose that P is canonical. The placement $P \cdot E_P$ is obtained from $P \cdot E_Q$, using the transformation described

in the proof of Property 1. Since the profile of P is contained in the profile of Q, the sequence $P \cdot E_Q$ is a placement (the items are inside the bin and it satisfies the height constraint). The transformation lets the positions of the items in P unchanged since P is canonical, and the final positions of the other items are all greater or equal to the position p_{last} of the last item i_{last} of P, since there is a locally minimal extension of P with each item not in P. For the same reason, for each item i in E_P, whose position after the transformation is p_{last}, $i_{last} \prec i$ holds. Then if P cannot be extended to a placement of all the items, Q cannot either. If P is not canonical the transformation must be restricted to E_Q, to let the positions of the items in P unchanged.

Memorization works as follows: each time the function `SearchPacking` returns FAILURE, if there is an extension for each item of S, then the set S and the profile of P are memorized in the hash table $\mathscr{T}$. When a new node is expanded, the function looks in the hash table for a configuration with the same set of free items and a profile contained in the current profile, to avoid a useless exploration. Remark that memorization does not fit well with other techniques intended to prune the search tree, like pseudo-symmetry breaking or blocks ordering [4]. Pseudo-symmetry breaking aims at avoiding the generation of packings and their symmetrics with respect to the x-axis or y-axis. A simple way to implement this constraint in placements consists in forcing two items i and j to always occur the first one before the second one in the placements. To be consistent with the memorization (that is to ensure that the above proof remains true), the transformation described in the proof of Property 1 must preserve the order in which i and j occur in the placement. For placements on the x-axis this is achieved choosing i with a smaller height than j. We did not use these techniques which are not very useful as the profiles are memorized.

5 Experimental results

We have implemented the function `SearchPacking` and we have compared the computation times with those of other well-known approaches. In this implementation the items are ordered in decreasing order of their areas (this ordering defines the ranks of the items), and at each node of the search tree the valid extensions are explored in increasing order of their positions. Moreover, before starting the exploration, the problems are simplified, removing the items which cannot be placed with any other items one beside the other (resp. one above the other) and reducing the size of the bin accordingly. Apart from this basic initial simplification, no technique is used to detect non-feasible instances or to modify the sizes of the items or the sizes of the bin before the search.

First we evaluated the efficiency of the memorization of the profiles, and the gain provided by the evaluation of the wasted space with a knapsack algorithm, as described in section *Profiles*. We have reported in table 1 the computation times in seconds and the average numbers of nodes which are explored per second, on a selection of problems from F. Clautiaux. We considered three alternatives for the evaluation of the wasted space: either the wasted space is calculated only in the vertical direction, or it is calculated in both the vertical and horizontal direction and these values are combined to deduce a lower bound, or the wasted space is not evaluated, and the algorithm backtracks when there is not enough free space for the items that remain to be packed. We reported only the results of the hardest problems. First it may be noticed that combining the vertical wasted space with the horizontal wasted space is better but does not produce significant gains. In fact, since the wasted spaces corresponding to the horizontal strips and those corresponding to the vertical strips

Wasted space		*vertical only*		*vert. and horizontal*		*no evaluation*	
Memorization		*yes*	*no*	*yes*	*no*	*yes*	*no*
E03X18	*S*	0.15	0.20	0.09	0.27	0.14	0.76
E20F15	*S*	0.00	0.15	0.00	0.25	0.00	0.40
E20X15	*S*	0.12	0.07	0.07	0.11	0.13	0.19
E00N23	*U*	1.30	2.57	1.15	2.47	2.56	5.03
E00X23	*U*	24.83	22.33	23.74	20.76	71.90	36.81
E02N20	*U*	5.35	8.14	4.40	7.03	8.27	16.70
E03N16	*U*	0.82	1.07	0.92	1.17	1.50	1.59
E04N17	*U*	0.15	1.06	0.15	0.83	0.18	1.86
E05X15	*U*	1.37	11.45	1.41	9.57	1.62	29.48
E07X15	*U*	0.73	6.63	0.77	4.92	0.88	10.10
E10N15	*U*	0.08	1.88	0.04	1.50	0.05	3.13
E10X15	*U*	0.80	4.62	0.76	4.05	0.81	6.46
E13X15	*U*	0.15	11.20	0.15	7.59	0.15	22.40
total time		35.85	71.37	33.65	60.52	88.19	134.91
nodes per sec		619597	2451753	643185	3418031	368385	3624383

Table 1 Computation times for a selection of F. Clautiaux instances with different options

can overlap, it would not be correct to just add them together. We must subtract an upper bound of the surface of the intersections, and then considering the two dimensions does not necessarily produce significantly better bounds. On the contrary, if the wasted space is not at all taken into account, then the search is strongly penalized, especially if the problem is hard.

The memorization of the profiles helps to avoid useless explorations, and in general it speeds the search. However it is costly, even more if many profiles have been stored. In particular, if the wasted space is not evaluated but the profiles are memorized, the exploration is very slow (in average 368385 nodes are explored per second). Indeed many profiles are memorized, and searching if a dominating profile exists for the current set of free items is then very costly. In the worst case (see the problem E00X23) it is better not to memorize, since the advantage is smaller than the cost. However, in the general case, the best strategy consists in combining the computation of the waste space with the memorization of the profiles.

We also compared the results of our approach with the results of the best-known approaches on orthogonal packing problems (OPP-2) and on strip packing problems (SPP-2) (pack a set of items into a strip of fixed width and minimum height). For OPP-2 we used the selection of problems proposed by Clautiaux et al. [4]. Table 2 summarizes the results. We reported the characteristics of the problems, and the computation times of different approaches, that can be found in the literature: Fekete et al. in [9] (**FS**), Clautiaux et al. in [4] (**Cl07**), Clautiaux et al. in [5] (**Cl08 er** for the version with energetic reasoning and **Cl08 ss** for the version using the solutions of subset-sum problems, and no preprocessing methods or lower bounds evaluation at the root node in both versions), Grandcolas and Pinto in [10] (**GP**) (a SAT encoding of Fekete and Schepers characterization that uses an external SAT solver), Joncour et al. in [14] (**JP**) (an approach based on the characterization of Fekete et al. using MPQ-trees), and the results of the approach that we described in this paper (**SMP**[1]).

Times are in seconds. **FS** times were obtained with a Pentium 4 3.2 GHz (and a time out of 900 seconds that we use to compute the mean time), **Cl07** times with a Pentium 4 2.6 GHz,

[1] *search and memorize for packing*

Cl08 times with a Pentium M 1.8 GHz, **GP** times with a Pentium 4 3.2 GHz, and **JP** times with a Pentium 4 3 GHz. For all our experiments we run **SMP** on a machine equipped with a Xeon processor at 2.4 GHz (we estimate between 1.5 and 2 the ratio between a Pentium 4 3.2 GHz processor and a Xeon 2.4 GHz processor, and around 4 the ratio between a Pentium M 1.8 GHz and a Xeon 2.4 GHz). In the version that was used the wasted space was obtained from an evaluation in the two dimensions, and the profiles are systematically memorized.

Instance			**FS**	**Cl07**	**Cl08 er**	**Cl08 ss**	**GP**	**JP**	**SMP**
Name	*feas.*	*n*	*time*	*time*	*time*	*time*	*time*	*time*	*time*
E02F17	*F*	17	7	12	0.01	0.00	4.95	30	0.00
E02F20	*F*	20	-	12	0.03	0.03	5.46	2	0.04
E02F22	*F*	22	167	4	0.00	0.01	7.62	2	0.00
E03X18	*F*	18	0	22	0.09	0.11		24	0.09
E04F15	*F*	15	0	1	0.02	0.03		60	0.00
E04F17	*F*	17	13	26	0.03	0.01	0.64	9	0.02
E04F19	*F*	19	560	7	0.01	0.01	3.17	14	0.01
E04F20	*F*	20	22	3	0.79	1.05	5.72	0	0.00
E05F15	*F*	15	0	3	0.01	0.01		0	0.00
E05F18	*F*	18	0	126	0.01	0.00		0	0.00
E05F20	*F*	20	491	2	0.97	0.96	6.28	6	0.00
E07F15	*F*	15	0	1	0.08	0.14		0	0.00
E08F15	*F*	15	0	117	0.00	0.00		0	0.00
E20F15	*F*	15	0	1	0.24	0.54		1	0.00
E20X15	*F*	15	0	44	0.00	0.00		8	0.07
E00N10	*N*	10	0	0	0.01	0.01		0	0.00
E00N15	*N*	15	0	2	0.00	0.02		2	0.10
E00N23	*N*	23	-	86	0.48	0.43		87	1.27
E00X23	*N*	23	-	289	5.98	5.40		-	23.62
E02N20	*N*	20	0	1	0.01	0.01		0	4.40
E03N10	*N*	10	0	0	0.01	0.00		0	0.00
E03N15	*N*	15	0	1	0.29	0.53		21	0.25
E03N16	*N*	16	2	32	0.57	1.12	39.90	51	0.74
E03N17	*N*	17	0	4	0.02	0.03	4.44	45	0.19
E04N15	*N*	15	0	1	0.52	1.09		7	0.09
E04N17	*N*	17	0	1	0.00	0.01		3	0.14
E04N18	*N*	18	10	7	0.05	0.05	161.34	7	0.08
E05N15	*N*	15	0	0	6.08	13.19		4	0.06
E05N17	*N*	17	0	1	0.10	0.22		1	0.08
E05X15	*N*	15	2	0	3.17	6.88		110	1.28
E07N10	*N*	10	0	0	0.01	0.00		0	0.00
E07N15	*N*	15	0	0	0.01	0.00		0	0.05
E07X15	*N*	15	0	1	0.17	0.31		79	0.61
E08N15	*N*	15	0	1	0.08	0.12		3	0.04
E10N10	*N*	10	0	0	0.00	0.00		0	0.00
E10N15	*N*	15	0	0	1.10	1.96		0	0.04
E10X15	*N*	15	0	1	0.21	0.32		50	0.60
E13N10	*N*	10	0	0	0.02	0.03		0	0.00
E13N15	*N*	15	0	0	0.00	0.00		0	0.00
E13X15	*N*	15	0	0	0.65	1.16		2	0.15
E15N10	*N*	10	0	0				0	0.00
E15N15	*N*	15	0	0	0.01	0.01		0	0.00
mean times			94.62	19.26	0.53	0.87		57.81	0.81

Table 2 Computation times in seconds to solve a selection of OPP-2 instances proposed by F. Clautiaux

On OPP-2 feasible instances **SMP** outperforms the other approaches. The order in which the extensions are considered, starting with the leftmost extensions, seems very effective. Indeed, for feasible instances it plays a significant role, guiding the search towards the choices which appear to be the most promising. On unfeasible instances **Cl08 er** and **Cl08 ss** are clearly the best performers. Their mean times are better than those of **SMP**, and the machine used for **SMP** is faster. F. Clautiaux et al. approach is particularily efficient when the problems are hard, see for example the problems E00N23, E00X23 or E04N18. An explanation for this is that, in their constraint-based model, the positions of the items on the x-axis and on the y-axis are constrained together. Each time the position of an item on the x-axis is fixed, constraint propagation helps to eliminate some positions of the free variables on the x-axis, but also on the y-axis. Thus the inconsistency can be detected much earlier, even if the relaxed problem has no solution. That is particularly true in the case of problems which are highly constrained, such as the instances E00N23 and E00X23 in which the total area of the items equals the area of the bin. It is also true for "easy" problems: for example **Cl08 er** solves the instances E07N10, E01N10, E13N15 or E15N15 exploring a unique node, while **SMP** explores between 3000 and 5000 nodes (in the same time). However, there are a few instances (E05N15, E10N15 or E13X15) on which **SMP** gets good results. Remark that, using the preprocessing methods and the lower bounds described in [4] at the root node, the performances of **Cl08 er** and **Cl08 ss** on these instances change completely (see the times in [5]). This explains the performances of **FS** and **Cl07** on these instances.

In the outer phase the items are packed as much as possible on the left side of the bin. This is pertinent when the problem is tightly constrained. If the bin is very large compared to the total area of the items, this strategy is inappropriate. The items share many intersections, and it becomes very hard to assign vertical positions to the items, exactly as if the bin was narrow. Then the procedure spent most of the time in the inner phase. To address this problem, we propose to consider, uniquely in the outer phase, a bin with a reduced height. This additional constraint makes the placements on the x-axis more homogeneous, and the search for the vertical positions of the items easier. If no solution is found with the reduced height then the search is reiterated with the real height, to confirm or invalidate the infeasibility. This idea is of great interest when solving strip packing problems with a *decrease height strategy*. In this case, the algorithm solves a series of problems which are all feasible except the last one. Table 3 shows the numbers of expanded nodes and the computation times for solving the same problem while varying the height of the bin (this example is based on the problem E05F20 from F. Clautiaux).

Three approaches are presented. In the first one (column *normal search*) the function `SearchPacking` is just called to solve the instance. In the second approach (column *90-degree rotation*) a 90-degree rotation is applied to the problem just before calling the search procedure. In the third one, the search procedure is called for each height, but in the outer phase the height of the bin is assigned a median value between the current height and a lower bound obtained by dividing the total area of the items by the width of the bin. In the inner phase the real height is used. If a solution is found with the reduced height then the problem is also feasible with the real height. In the other case the search is reitered with the real height in both phases, to confirm the unfeasibility. For each height or width we reported the feasibility (or $-$ if the timeout is reached), the number of expanded nodes $N1$ in the outer phase, the number of expanded nodes $N2$ in the inner phase, and the computation time. With the first approach the computation times are sometimes very important whereas the

[2] 141190 to discover the unfeasibility with the lowered height, and 20 to find the feasibility with the real height

	normal search				90-degree rotation				reduced height in phase 1			
h	*feas.*	*N*1	*N*2	*time*	*feas.*	*N*1	*N*2	*time*	*feas.*	*N*1	*N*2	*time*
26	–	21	48303478	>60	S	20	68057	0.15	S	20	20	0.0
25	–	20	23990744	31.66	S	20	58312	0.13	S	20	20	0.0
24	S	20	20	0.0	S	20	43696	0.10	S	20	20	0.0
23	–	21	41089487	>60	S	20	33778	0.09	S	20	118	0.0
22	S	20	20	0.0	S	20	20362	0.06	S	20	20	0.0
21	S	20	110	0.0	S	20	12203	0.04	S	20	20	0.0
20	S	20	20	0.0	S	1593	280141	0.85	S	141210^2	20	0.26
19	U	44966	0	0.08	U	58081	0	0.08	U	44966	0	0.10
18	U	0	0	0.0	U	0	0	0.0	U	0	0	0.0
17	U	0	0	0.0	U	0	0	0.0	U	0	0	0.0

Table 3 Computations times to solve a packing problem for different heights of the bin

bin is high and the problem easy. There is no memorization in the second phase, and then searching the vertical positions can generate huge explorations (see for example the number of expanded nodes in the second phase when the height is 26). The third implementation corrects this defect: easy problems are easy to solve. The computation time is the most important (0.26 seconds) when the height is 20 (the optimal value). The search procedure has been called two times for this instance: first, with a lowered height in the outer phase, it returns failure, and secondly using the current height in both phases, it returns success with a negligible computation time.

It appears clearly from these results that the best strategy for the strip packing problem consists in increasing the height of the strip, starting with a minimal bound. We reported in table 4 the results published by S. Martello et al. [16] (**MMV**), R. Alvarez-Valdes et al. [2] (**AVPT**), M. Boschetti and L. Montaletti [3] (**BM**), and M. Kenmochi et al. [15] (**ST** for STAIRCASE and **G-ST** for G-STAIRCASE), on a selection of strip packing problems. These approaches are based on branch and bound algorithms, and use lower bounds, dominance rules and problem relaxation techniques. They appear to be the best exact approaches yet, if we do not include the approach of J.F. Côté, M. Dell' Amico et M. Lori [6] which makes use of CPLEX to solve linear programs, and then should be considered in a class by itself. We also reported the results that we obtained with an increase-height strategy (the height of the bin is incremented one by one while the problem is not feasible starting from a lower bound) and with a decrease-height strategy using a reduced height in the outer phase as described above. For each instance we have indicated the number of items, the width of the strip and the optimal height. **MMV** used a Pentium III 800 MHz with a time limit of 3,600 seconds, **AVPT** computation times were obtained with a Pentium 4 at 2 Ghz with a time limit of 1,200 seconds, **BM** computation times with a Pentium M 725 1.60 GHz and a time limit of 1,200 seconds, and **ST** and **G-ST** with a Pentium 4 3.0GHz and a time limit of 3,600 seconds. We made our experimentations on a Linux machine equipped with a Xeon 2.4 GHz processor (a machine three times faster than the machine used by R. Alvarez-Valdes et al.) and a time limit of 1,500 seconds. Times are in seconds, and a $-$ denotes that the program reached timeout.

The increase-height strategy outperforms the decrease-height strategy. By analysing precisely the causes of the poor performances of the latter, we observed that in many cases the program spent most of the time in the inner phase, that is searching the vertical positions of the items, even if a reduced height is used in the outer phase. The results are identical if a 90-degree rotation is applied to the problem before calling the search procedure. In fact,

Instance				**MMV**	**AVPT**	**BM**	**ST**	**G-ST**	**incr.**	**decr.**
Name	*n*	*w*	*opt.*	*time*	*time*	*time*	*time*	*time*	*time*	*time*
ht1	16	20	20	10.84	0.03	3.84	0.10	0.07	0.00	0.02
ht2	17	20	20	3043.25	0.36	149.98	0.12	0.07	0.00	1.91
ht3	16	20	20	500.75	0.02	1.22	0.08	0.10	0.00	2.57
ht4	25	40	15	8.26	0.06	611.70	0.08	0.11	0.01	–
ht5	25	40	15	20.29	0.03	300.95	0.09	0.06	0.05	142.51
ht6	25	40	15	16.94	0.02	25.79	0.11	0.06	0.00	1349.63
ht7	28	60	30	–	1.70	654.56	0.12	0.10	9.54	–
ht8	29	60	30	–	–	732.05	71.40	76.97	0.35	–
ht9	28	60	30	0.00	8.55	669.90	0.10	0.13	17.07	–
cgcut1	16	10	23	11.48	0.02	0.69	0.10	0.12	0.00	0.20
cgcut2	23	70	64	–	–	–	–	–	–	–
gcut1	10	250	1016	0.00	0.02	0			1.17	0.00
gcut2	23	250	1187	–	–	7.41			–	–
ngcut1	10	10	23	0.05	2.11	0.08	2080.04	0.39	0.00	0.00
ngcut2	17	10	30	11.31	3.11	0.47	–	–	0.01	0.01
ngcut3	21	10	28	27.01	0.00	1.62	0.09	0.10	0.00	0.02
ngcut4	7	10	20	0.00	0.03	0.14	3.63	0.14	0.00	0.00
ngcut5	14	10	36	0.00	0.00	0.34	0.11	0.07	0.00	0.01
ngcut6	15	10	31	727.20	4.39	0.84	–	147.31	0.02	0.31
ngcut7	8	20	20	0.00	0.00	0.38	–	0.10	0.00	0.00
ngcut8	13	20	33	53.09	3.39	15.39	30.51	0.50	0.00	0.03
ngcut9	18	20	50	–	56.13	286.55	–	1971.64	0.12	4.46
ngcut10	13	30	80	0.18	2.61	6.58	–	113.98	0.06	0.00
ngcut11	15	30	52	483.01	11.88	107.47	–	7.71	0.03	0.09
ngcut12	22	30	87	0.00	0.03	1.41	–	–	63.93	12.16
beng1	20	25	30	911.58	5.41	26.95	0.53	0.93	0.0	0.21
beng2	40	25	57	–	0.41	72.55	1.13	22.89	314.95	–

Table 4 Computation times in seconds to solve strip packing problems (SPP-2)

the increase-height strategy leads to solve unfeasible problems, or problems highly constrained, and then the evaluation of the wasted space helps efficiently to detect dead-ends and to prune the search space. Compared to the other approaches, the increase-height strategy gives good results on *ht* problems. These problems were proposed by E. Hopper and B.C.H. Turton [12]. They are particular in that their optimal solutions are perfect packings. In this case, the increase search strategy starts with the optimal height, and then the algorithm solves a unique instance. Furthermore, there must be no wasted space in the bin for this instance. Then the procedure backtracks whenever a dimension of the free space cannot be perfectly occupied with free items. The increase-height strategy produces also good results on NGCUT problems. The best performer on the whole selection of problems is **AVPT**. This approach combines many techniques. Lower and upper bounds are calculated, these last ones using the GRASP algorithm[1]. Then a branch and bound procedure determines the optimal height. In many cases the lower and upper bounds which are computed in the first phase coincide, and then there is no need to call the branch and bound procedure. In the other cases, the optimality is not always established before the time limit. When the number

of items is large these techniques are very efficient, while our search procedure tends to perform huge explorations. On the other side, when the problems are small but hard, the size of the search tree is bounded, and the techniques employed in our approach are efficient to avoid redundant explorations and to capture symmetries. In fact when there are many items, memorizing the profiles becomes inappropriate since the number of subsets becomes very large, and knapsack computations become very costly. Even with a good amount of memory we were not able, for example, to solve the problems $beng_i$ when i is greater than 2, while **AVPT** or **ST** easily solve all beng instances.

Finally we tested our procedure on optimal rectangle packing problems. Table 5 compares the CPU times that we obtained with those of Huang and Korf [13], on oriented equal-perimeter benchmark (problem n consists in packing n oriented rectangles of sizes $1 \times n$, $2 \times (n-1)$,..., $n \times 1$ in a box of minimal area). We successfully solved instances 1, 2,..., 23, and 24. We did not get equally good results on consecutive-square instances. The general approach consists in generating all the possible bounding boxes, then in solving the corresponding packing problems in order of increasing area, until all optimal solutions are discovered. Column *tested* indicates the number of boxes which were tested, and column *opt.* contains the optimal boxes. Computation times are in the format *days* : *hours* : *minutes* : *seconds*. For instance 22 it takes 6 hours 17 minutes to prove that 51×40 is not a valid box, while the optimal box 60×34 had been already discovered.

Instance			**HK13**	**SMP**
n	*tested*	*opt.*	*time*	*time*
19	12	42×32 56×24	2:15	1:01
20	11	42×37	7:51	6:47
21	9	51×35	11:20	11:12
22	15	60×34	9:12:37	7:00:22
23	16	61×38	3:22:50:38	6:27:08
24	18	69×38 57×46		18:32:41

Table 5 Computation times required on the oriented equal-perimeter benchmark

6 Conclusion

In this paper, we described a procedure to determine the feasibility of orthogonal packing problems. The procedure is based on F. Clautiaux et al. approach [4]. It consists in solving a relaxation of the initial problem, so as to fix the positions of the items on the horizontal axis. Each time a solution of the relaxed problem is discovered, another procedure is called to determine if this solution can be extended to a solution of the packing problem. This two-phase strategy appears to be less costly than the generation of all the placements of the items in the bin. We proposed a new search procedure which aims to improve this approach. Experimentations on classical benchmarks show the usefulness of this procedure. The performances are comparable to those of the best known approaches on many instances. We plan to implement the computation of more relevant bounds, in particular using DFF functions: to prove

the unfeasability during the search, the procedure which is used to solve the relaxed problem could also be used to detect the unfeasability of the DFF-transformation of the problem (in place of the continuous lower bound). The connections between the positions of the items on the horizontal axis and the positions of the items on the vertical axis should be also investigated, so as to establish simple rules that could help to detect inconsistencies while solving the relaxed problem.

References

1. Alvarez-Valdes, R., Parreño, F., Tamarit, J.: Reactive grasp for the strip-packing problem. Computers & Operations Research **35**(4), 1065 – 1083 (2008). DOI 10.1016/j.cor.2006.07.004
2. Alvarez-Valdes, R., Parreño, F., Tamarit, J.: A branch and bound algorithm for the strip packing problem. OR Spectrum **31**, 431–459 (2009)
3. Boschetti, M.A., Montaletti, L.: An exact algorithm for the two-dimensional strip-packing problem. Oper. Res. **58**, 1774–1791 (2010)
4. Clautiaux, F., Carlier, J., Moukrim, A.: A new exact method for the two-dimensional orthogonal packing problem. European Journal of Operational Research **183**(3), 1196–1211 (2007)
5. Clautiaux, F., Jouglet, A., Carlier, J., Moukrim, A.: A new constraint programming approach for the orthogonal packing problem. Comput. Oper. Res. **35**, 944–959 (2008)
6. Côté, J.F., Dell'Amico, M., Iori, M.: Combinatorial benders' cuts for the strip packing problem. Operations Research **62**(3), 643–661 (2014)
7. Demeulemeester, E., Herroelen, W.: A branch-and-bound procedure for the multiple resource-constrained project scheduling problem. Manage. Sci. **38**(12), 1803–1818 (1992)
8. Fekete, S.P., Schepers, J.: A combinatorial characterization of higher-dimensional orthogonal packing. Mathematics of Operations Research **29**(2), 353–368 (2004)
9. Fekete, S.P., Schepers, J., van der Veen, J.: An exact algorithm for higher-dimensional orthogonal packing. Operations Research **55**(3), 569–587 (2007)
10. Grandcolas, S., Pinto, C.: A sat encoding for multi-dimensional packing problems. In: Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, *Lecture Notes in Computer Science*, vol. 6140, pp. 141–146 (2010)
11. Herz, J.C.: Recursive computational procedure for the two dimensional stock cutting. IBM J. Res. Development **16**, 462–469 (1972)
12. Hopper, E., Turton, B.C.H.: An empirical investigation of meta-heuristic and heuristic algorithms for a 2d packing problem. European Journal of Operational Research **128**, 34–57 (2000)
13. Huang, E., Korf, R.E.: Optimal rectangle packing: An absolute placement approach. J. Artif. Int. Res. **46**(1), 47–87 (2013)
14. Joncour, C., Pêcher, A., Valicov, P.: Mpq-trees for the orthogonal packing problem. Journal of Mathematical Modelling and Algorithms **11**(1), 3–22 (2012)
15. Kenmochi, M., Imamichi, T., Nonobe, K., Yagiura, M., Nagamochi, H.: Exact algorithms for the 2-dimensional strip packing problem with and without rotations (2008)
16. Martello, S., Monaci, M., Vigo, D.: An exact approach to the strip-packing problem. Journal on Computing **15**(3), 310–319 (2003)
17. Martello, S., Pisinger, D., Vigo, D.: The three-dimensional bin packing problem. Operations Research **48**(2), 256–267 (2000)
18. Stinson, J., Davis, E., Khumawala, B.: Multiple resource-constrained scheduling using branch and bound. AIIE Transactions **10**, 252–259 (1978)