A note on Robot-packable and Orthogonal variants of the three-dimensional bin packing problem

Edgar den Boef, Jan Korst, Silvano Martello, David Pisinger, Daniele Vigo

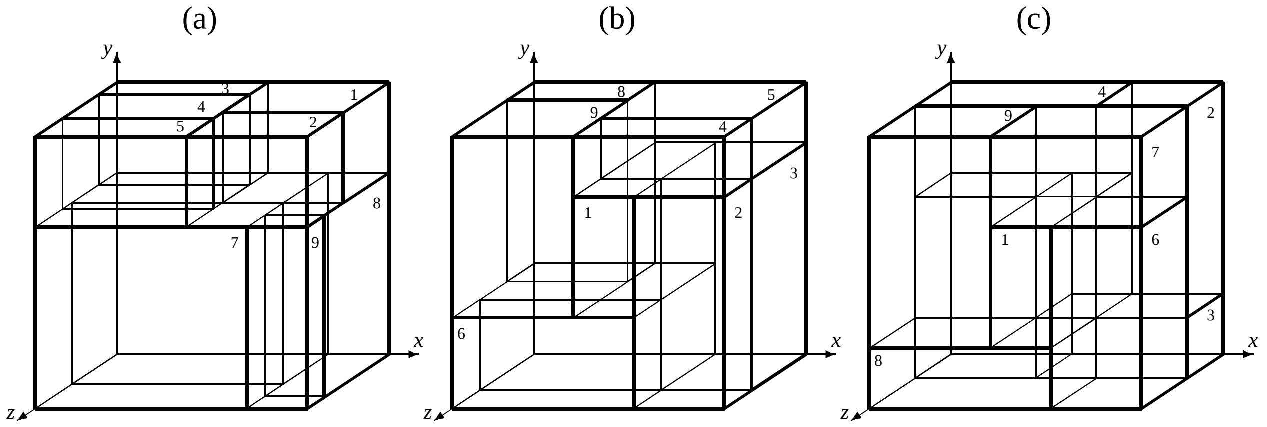
Abstract

In the three-dimensional bin packing problem the task is to orthogonally pack a given set of rectangular items into a minimum number of three-dimensional rectangular bins. We give a characterization of the algorithm proposed by Martello, Pisinger and Vigo (2000) for the exact solution of the problem, showing that not all orthogonal packings can be generated by the proposed algorithm. The packings however have the property of being robot packings, which is of great relevance to the industry. A new (and correct) algorithm for solving the general orthogonal packing problem is presented based on constraint programming. Extensive computational experiments compare the algorithm for the three-dimensional bin packing when solving general orthogonal packings and when restricted to robot packings.

1 Introduction

In the three-dimensional bin packing problem we are given a set of n rectangular-shaped *items* (boxes) $j = 1, 2, \dots, n$, each having a width w_j, height h_j and depth d_j, and an unlimited number of identical three-dimensional *bins* (containers) having width W, height H and depth D. The task is to orthogonally pack all the items into a minimum number of bins. We assume that the items may not be rotated, i.e., that they are packed with each edge parallel to the corresponding bin edge. Such packings will be denoted as *general* packings.

If additionally it is demanded that the items can be cut out of the bin through sequential guillotine cuts (i.e., face-to-face cuts parallel to the faces of the bin), the packing is said to be *guillotine cuttable*. A *robot packing* is a packing which can be achieved by successively placing items starting from the bottom-left-behind corner, and such that each item is in front of, right of, or above each of the previously placed items. The name comes from the fact that robots used for packing boxes in the industry are equipped with a rectangular hand parallel to the base of the bins, which is covered with vacuum cells for lifting the boxes. To avoid collisions, it is demanded that no already packed box is positioned in front of, right of, or above the destination of the current box.



(a)

i	w_i	h_i	d_i	x_i	y_i	z_i
1	4	3	5	5	6	0
2	4	3	4	5	6	5
3	5	3	2	0	6	0
4	5	3	4	0	6	2
5	5	3	3	0	6	6
6	7	6	5	0	0	0
7	7	6	4	0	0	5
8	2	6	7	7	0	0
9	2	6	2	7	0	7

(b)

i	w_i	h_i	d_i	x_i	y_i	z_i
1	2	4	9	4	3	0
2	3	7	3	6	0	6
3	3	7	6	6	0	0
4	5	2	3	4	7	6
5	5	2	6	4	7	0
6	6	3	3	0	0	6
7	6	3	6	0	0	0
8	4	6	3	0	3	0
9	4	6	6	0	3	3

(c)

i	w_i	h_i	d_i	x_i	y_i	z_i
1	2	4	9	4	2	0
2	3	7	4	6	2	0
3	5	2	4	4	0	0
4	6	3	4	0	6	0
5	4	6	4	0	0	0
6	3	6	5	6	0	4
7	5	3	5	4	6	4
8	6	2	5	0	0	4
9	4	7	5	0	2	4

Figure 1: (a) Guillotine cuttable packing; (b) robot packing, which cannot be obtained by guillotine cuts; (c) general packing, which cannot be obtained by robot packings. The tables give the dimensions (w_j, h_j, d_j) and coordinates (x_j, y_j, z_j) of the items.

Each guillotine cuttable packing is also a robot packing. Consider indeed a guillotine cuttable packing and a feasible sequence (tree) of cuts. By first packing the items in the bottom, left or back part (depending on the cutting direction) of each cut, a feasible robot packing is obtained.

We assume in the following that coordinates originate from the bottom-left-behind corner of the bin, and that (x_j, y_j, z_j) is the point where the bottom-left-behind corner of item j is positioned. The three different packing strategies are illustrated in Figure 1, where the bin size is $9 \times 9 \times 9$. Figure 1(a) shows a guillotine cuttable pattern. A first cut, parallel to the x-z plane at height $y = 6$ separates items 1–5 from items 6–9; two cuts, parallel to the y-z plane, will further separate the two resulting subsets, and a final series of cuts parallel to the x-y plane will produce the items. Figure 1(b) gives an example of robot packing. A feasible packing is produced by the item number sequence: 7, 6, 8, 9, 1, 3, 2, 5, 4. Note that this pattern is not guillotine cuttable: any face-to-face initial cut would cut through some items. Moreover, no guillotine packing using a single bin exists for this item set, as one can see by considering the first cut, which is either parallel to the x-y plane, to the x-z plane or to the y-z plane: for each of these cases, it can be checked that no feasible cut separating two completely filled regions is possible. A general packing, which cannot be obtained by robot packings, is finally shown in Figure 1(c). Robot packing can only start with item 5; the unique next feasible robot packing is that of item 3, then that of item 8. At this point, no further item can be added without leaving unreachable holes. Moreover, no robot packing using a single bin exists for this item set (as we checked by running a robot packing algorithm).

In Martello, Pisinger and Vigo [7], a branch-and-bound algorithm for the exact solution of the three-dimensional bin packing problem with general packings was presented. The algorithm consists of a main depth-first branching tree in which, at each node, the current item is repeatedly assigned to each bin already containing some items (plus, possibly, to a new bin), without specifying the actual placements of the items. The feasibility of the assignment of the current subset of items to a single bin is checked as follows. First, lower bounds are computed for the subset: if one of them gives a value greater than one, the node is immediately pruned. Otherwise, heuristic algorithms are executed: if a solution requiring a single bin is obtained, the node is accepted. If none of the two cases occurs, the optimal solution for the sub-instance is determined through a subroutine ONEBIN: the node is then accepted if a single bin solution is found, or pruned otherwise. The ONEBIN algorithm is a constructive branch-and-bound algorithm which, at each step, places an item in a feasible point, according to the following rule (see Property 2 of Martello, Pisinger and Vigo [7]): each new item is placed such that it is in front of, right of, or above each of the previously placed items. More formally, when successively placing items $1,2,\ldots,m$, for each pair i,j with $i<j$ (i.e., item i is placed before item j), at least one of the following three equations should be true:

$$\begin{array}{l} x_i + w_i \leq x_j \\ y_i + h_i \leq y_j \\ z_i + d_i \leq z_j. \end{array} \tag{1}$$

Obviously the packing patterns generated in this way are robot packable. However, not all general packings satisfy this property. Indeed, packing (c) in Figure 1 cannot be achieved by the ONEBIN algorithm as for items 1, 4, and 9, irrespective of the order in which they are packed, the last placed item violates the above property.

In Section 2 we describe an MILP model for the general single-bin packing problem, based on the modeling technique proposed by Onodera, Taniguchi and Tamaru [10] for a two-dimensional case. The model may also be seen as a specialization of that proposed by Chen, Lee and Shen [4] for the multiple-container loading problem. Solving this model by a standard MILP solver may replace the ONEBIN procedure in the algorithm, thus making it possible to solve the general three-dimensional bin packing problem. Computational experiments, however, indicated that solving the model through commercial software (CPLEX 7.0) was quite time consuming, and only problems of small size could be solved. Thus, in Section 3, a new algorithm is proposed based on constraint programming. Use of domain restriction and constraint propagation makes it possible to determine infeasible packings early in the search tree, resulting in a much better performing algorithm. In Section 4 we finally compare the two versions of the algorithm, dealing with general and robot packing, respectively. These extensive computational experiments, comprising more than one year of total CPU time, have solved considerably more instances to optimality than previously reported in the literature.

2 Formulation of the single-bin packing problem

The decision problem whether a given subset of items fits into a single bin of size $W \times H \times D$ may be formulated as follows. Let the items be denoted as $1,2,\ldots,m$, and let us introduce the binary decision variables ℓ_{ij} (left), r_{ij} (right), u_{ij} (under), a_{ij} (above), b_{ij} (behind), f_{ij} (in front) for items i,j with $i<j$. As no two items i,j may overlap, we must demand that they are located left, right, under, above, behind or in front of each other. Thus

$$\ell_{ij}+r_{ij}+u_{ij}+a_{ij}+b_{ij}+f_{ij}\geq 1 \qquad i,j=1,2,\ldots,m,\;\; i<j \tag{2}$$

If just one of the relations is satisfied, no overlap will occur. However, more than one relation is allowed, e.g., $\ell_{ij}=b_{ij}=a_{ij}=1$. If i is located on the left of j, i.e., $\ell_{ij}=1$, then we have that $x_i+w_i\leq x_j$. In general we get, for $i,j=1,2,\ldots,m$ with $i<j$, the logical constraints:

$$\begin{array}{lcl} \ell_{ij}=1 & \Rightarrow & x_i+w_i\leq x_j \\ r_{ij}=1 & \Rightarrow & x_j+w_j\leq x_i \\ u_{ij}=1 & \Rightarrow & y_i+h_i\leq y_j \\ a_{ij}=1 & \Rightarrow & y_j+h_j\leq y_i \\ b_{ij}=1 & \Rightarrow & z_i+d_i\leq z_j \\ f_{ij}=1 & \Rightarrow & z_j+d_j\leq z_i \end{array} \tag{3}$$

Obviously we must also demand that $0\leq x_j\leq W-w_j$, $0\leq y_j\leq H-h_j$ and $0\leq z_j\leq D-d_j$, to ensure that no part of item j exceeds the bin.

By linearizing (3), the single-bin packing problem can be formulated as the following MILP problem. (See, e.g., Mitra et al. [9] for a general discussion on how logical constraints can be linearized.)

$$\begin{array}{ll} \ell_{ij}+r_{ij}+u_{ij}+a_{ij}+b_{ij}+f_{ij}\geq 1 & i,j=1,2,\ldots,m,\;\; i<j \\ x_i-x_j+W\ell_{ij}\leq W-w_i & i,j=1,2,\ldots,m,\;\; i<j \\ x_j-x_i+Wr_{ij}\leq W-w_j & i,j=1,2,\ldots,m,\;\; i<j \\ y_i-y_j+Hu_{ij}\leq H-h_i & i,j=1,2,\ldots,m,\;\; i<j \\ y_j-y_i+Ha_{ij}\leq H-h_j & i,j=1,2,\ldots,m,\;\; i<j \\ z_i-z_j+Db_{ij}\leq D-d_i & i,j=1,2,\ldots,m,\;\; i<j \\ z_j-z_i+Df_{ij}\leq D-d_j & i,j=1,2,\ldots,m,\;\; i<j \\ 0\leq x_j\leq W-w_j & j=1,2,\ldots,m \\ 0\leq y_j\leq H-h_j & j=1,2,\ldots,m \\ 0\leq z_j\leq D-d_j & j=1,2,\ldots,m \\ \ell_{ij},r_{ij},u_{ij},a_{ij},b_{ij},f_{ij}\in\{0,1\} & i,j=1,2,\ldots,m,\;\; i<j \end{array} \tag{4}$$

Due to the large number of decision variables, the MILP formulation (4), however, turned out to be difficult to solve through commercial software (CPLEX 7.0). This confirms analogous results obtained by Chen, Lee and Shen [4] with the original container loading model. Thus a different approach based on techniques from constraint programming was used.

3 A constraint programming approach

Constraint programming has recently been recognized as a promising tool for solving decision problems which are difficult to formulate in ILP form (see, e.g., Brailsford, Potts and Smith [3]).

In our problem, the domain of each pair of items i, j is $M_{ij} = \{$l, r, u, a, b, f$\}$ meaning that they can be placed relative to each other as given by the six relations. To avoid symmetries, items 1 and 2 have the restricted domain $M_{12} = \{$l, u, b$\}$.

The problem is solved recursively by an algorithm, called CSPBIN, where at each iteration two items i and j are considered, and one of the relations $\ell_{ij}, r_{ij}, u_{ij}, a_{ij}, b_{ij}, f_{ij}$ is imposed. The feasibility of the imposed relations is then checked, and if it can be proved that no solution exists, the algorithm backtracks. Otherwise the algorithm calls itself recursively. If all pairs of items i, j have been imposed a relation, and a feasible assignment of coordinates exists, the algorithm terminates with a positive answer.

The feasibility of a partially ordered problem is checked as follows. First, the items are ordered according to the ℓ_{ij} and r_{ij} relations. Then, for a given item j the position x_j is defined as the maximum of $x_i + w_i$ for all items i located left to item j (while x_j is set to 0 if no item is located left of j). If $x_j + w_j > W$ for some j then the relations cannot be satisfied. The algorithm runs in $O(n^2)$ time since the topological ordering takes $O(n^2)$ time and the assignment of coordinates can be done in $O(n^2)$ as there are $O(n^2)$ relations ℓ_{ij}, r_{ij}. The same approach is used for relations u_{ij}, a_{ij} and b_{ij}, f_{ij}.

To further speed up the solution algorithm, domain restriction and constraint propagation is applied. Domain restriction is obtained by the Forward Check (FC) approach proposed by Haralick and Elliot [6]. Having chosen a relation between items i, j, we may remove all other relations from M_{ij} as we know that only one relation needs to be chosen.

We use the Mainaining Arc Consistency (MAC) approach proposed by Gaschnig [5] to propagate the chosen constraints. The MAC algorithm is a more sophisticated FC algorithm which at any step keeps the problem arc consistent. The present problem is arc consistent if for all possible assignments of a relation between two boxes i and j there is a feasible assignment of coordinates to the boxes such that all constraints are satisfied. If this is not possible then the relation is removed from the domain of M_{ij}. In the case where a domain becomes empty the problem cannot be satisfied, and we backtrack.

Although the use of MAC means that the time complexity of each node in the search tree grows considerably, the number of nodes investigated decreases correspondingly. Sabin and Freuder [12] showed that the additional effort of MAC pays off when dealing with hard problems.

The MAC strategy is implemented as follows. For each recursive call of CSPBIN (i.e., for each new assignment of a relation) we run through all pairs of items i and j with $|M_{ij} \geq 2|$ and test feasibility of each relation $R \in M_{ij}$. If the packing becomes infeasible with the additional relation imposed, the relation is removed from the domain. If M_{ij} in this way becomes empty, we may conclude that no feasible packing can be obtained by following this branch, and thus we may backtrack in CSPBIN. If only one relation is left in a domain M_{ij} then this relation is fixed. All reductions in the domain achieved by the forward propagation are pushed to a stack, such that the domains quickly can be restored upon backtracking from CSPBIN.

The best performance of algorithm CSPBIN was obtained by sorting the items according to non-increasing volume, and then branching on pairs of items $(1,2)$, $(1,3)$, $(2,3)$, $(1,4)$, $(2,4)$, $(3,4)$, $(1,5)\ldots$. In this way the largest items were placed relatively to each other at an early stage of the algorithm and infeasibility could quickly be detected. This complies with the *Fail First Strategy* proposed by Barták [1].

4 Computational Experiments

The algorithm by Martello, Pisinger and Vigo [7] was modified as follows. When the actual feasibility of the assignment of the current subset of items to a single bin has to be checked, lower bounds and heuristic algorithms are first executed as described in Section 1. If they fail in (respectively) pruning or accepting the node, algorithm CSPBIN is executed for a final decision on node acceptance. We refer to the resulting algorithm as the *general packing algorithm*, and if the execution of algorithm CSPBIN is replaced by the execution of the original algorithm ONEBIN, we refer to it as the *robot packing algorithm*.

Both versions of the algorithm were tested on eight classes of random instances, generated according to different distributions. The first group of classes (1–5) are generalizations of the instances considered by Martello and Vigo in [8]. The bin size is $W = H = D = 100$ and five types of items are considered. Type 1: $w_j \in [1, \frac{1}{2}W]$, $h_j \in [\frac{2}{3}H, H]$, $d_j \in [\frac{2}{3}D, D]$. Type 2: $w_j \in [\frac{2}{3}W, W]$, $h_j \in [1, \frac{1}{2}H]$, $d_j \in [\frac{2}{3}D, D]$. Type 3: $w_j \in [\frac{2}{3}W, W]$, $h_j \in [\frac{2}{3}H, H]$, $d_j \in [1, \frac{1}{2}D]$. Type 4: $w_j \in [\frac{1}{2}W, W]$, $h_j \in [\frac{1}{2}H, H]$, $d_j \in [\frac{1}{2}D, D]$. Type 5: $w_j \in [1, \frac{1}{2}W]$, $h_j \in [1, \frac{1}{2}H]$, $d_j \in [1, \frac{1}{2}D]$. We then obtained five classes of instances as follows. For *Class k* ($k = 1, \ldots, 5$), each item is of type k with probability 60%, and of the other four types with probability 10% each.

The second group of classes (6–8) is a generalization of the instances presented by Berkey and Wang [2]. The bin size is always $W = H = D$, with $W = 10$ for Class 6, $W = 40$ for Class 7 and $W = 100$ for Class 8. The item sizes are obtained by uniformly randomly generating w_j, h_j and d_j in $[1, 10]$, $[1, 35]$ and $[1, 100]$, respectively for the three classes. The last class (Class 9) are the so-called *all-fill* problems, obtained by cutting three bins into smaller parts through guillotine and non-guillotine cuttings. A complete description of this class can be found in Martello, Pisinger and Vigo [7].

For each class, we considered instances of different size, with n ranging between 10 and 50. The algorithms were implemented in C, and the experiments were run on a Sun4 Sparc Ultra-Enterprise 400 MHz. A time limit of 300,000 seconds was given to each instance, and 10 instances were solved for each class and size of a problem. The time limit was chosen very huge in order to solve as many instances as possible to optimality, as this makes it possible to evaluate the quality of the bounds in more detail than in previous papers published on this subject. All the obtained solutions are available from the internet at the first author's webpage (`www.diku.dk/~pisinger`). In the following tables, results for the general packing algorithm are presented at the top of the tables, while results for the robot packing algorithm are given at the bottom. Both algorithms were run on the same computer, so the solution times are directly comparable.

Tables I and II compare the number of instances solved to optimality using a time limit of 300,000 and 1,000 seconds, respectively. With the huge time limit, the general version is able to solve more instances in most of Classes 1–8, while the robot packing version is better in Class 9. The better performance for Classes 1–8 may be explained by the fact that the general version has more freedom in arranging the items and thus may find solutions using fewer bins. In this way, having tighter upper bounds, the algorithms may terminate faster as the lower bounds used by both algorithms are the same. For Class 9, the behavior can be explained by the fact that the generated instances are all robot packable, so the robot packing version has the advantage that it only needs to search the relevant solution space. When it comes to the time limit of 1,000 seconds, the general packing algorithm is slightly better for most of the Berkey-Wang instances (see Classes 7 and 8) while the robot packing algorithm is slightly better for most of the the Martello-Vigo instances (see Classes 1–4). As before, the robot packing algorithm is better for Class 9. By considering the relative size of the items with respect to the size of the bins, it may be observed that the general packing algorithm is slightly better for instances in which the item size is relatively small (Classes 5, 7, and 8) while the robot-packing algorithm is slightly better for instances with relatively large items (Classes 1-4).

Table I: Number of instances, out of ten, solved to proved optimality in 300,000 seconds.

		Class								
packing	*n*	1	2	3	4	5	6	7	8	9
general	10	10	10	10	10	10	10	10	10	10
	15	10	10	10	10	10	10	10	10	10
	20	10	10	10	10	10	10	10	10	10
	25	10	10	10	10	10	10	10	10	10
	30	10	10	10	10	10	10	10	10	9
	35	10	10	10	10	10	10	10	10	10
	40	10	9	8	10	10	10	10	10	7
	45	10	8	7	10	10	10	10	10	2
	50	8	7	10	10	10	10	10	10	0
robot	10	10	10	10	10	10	10	10	10	10
	15	10	10	10	10	10	10	10	10	10
	20	10	10	10	10	10	10	10	10	10
	25	10	10	10	10	10	10	10	10	10
	30	10	10	10	10	10	10	10	10	10
	35	10	10	10	10	9	10	10	10	10
	40	10	9	9	10	9	10	9	10	10
	45	10	8	7	10	9	10	10	10	9
	50	8	8	10	10	4	10	9	10	1

Table II: Number of instances, out of ten, solved to proved optimality in 1,000 seconds.

		Class								
packing	*n*	1	2	3	4	5	6	7	8	9
general	10	10	10	10	10	10	10	10	10	10
	15	10	10	10	10	10	10	10	10	10
	20	10	10	10	10	10	10	10	10	10
	25	10	10	10	10	10	10	10	10	10
	30	10	10	10	10	10	10	9	10	9
	35	9	8	10	10	10	10	10	8	10
	40	9	8	5	10	8	10	7	9	2
	45	5	5	4	10	9	9	4	8	0
	50	5	2	4	10	4	9	5	8	0
robot	10	10	10	10	10	10	10	10	10	10
	15	10	10	10	10	10	10	10	10	10
	20	10	10	10	10	10	10	10	10	10
	25	10	10	10	10	10	10	9	10	10
	30	10	10	10	10	10	10	6	10	10
	35	10	10	10	10	7	10	6	6	10
	40	9	8	8	10	6	10	3	8	8
	45	6	6	4	10	8	10	1	7	2
	50	5	2	4	10	4	10	2	4	0

Table III compares the optimal solution values obtained by the two algorithms. As the robot packable solutions are a subset of all packings, the solutions found by the general packing algorithm should be smaller. However, there are only a few instances for which the general packing algorithm is able to find better solutions than the robot packing algorithm, while in general the solutions found are the same for both variants of the problem. Over a total of 764 instances solved to optimality by both algorithms, only in one case (Class 6, $n = 30$, instance no. 8) the general packing algorithm found a solution better than the one found by the robot packing algorithm. Notice that if an instance was not solved to optimality, the reported solution value may be larger than the optimum.

Table III: Average solution values found.

		Class								
packing	*n*	1	2	3	4	5	6	7	8	9
general	10	3.3	3.5	3.6	6.6	2.5	2.9	2.5	2.8	3.0
	15	4.7	4.5	4.8	8.7	2.9	4.4	3.0	3.7	3.0
	20	6.0	6.6	6.0	12.3	3.9	5.4	3.8	4.9	3.0
	25	7.4	7.0	7.1	15.4	4.6	5.9	4.3	5.5	3.0
	30	8.6	8.0	8.6	17.2	5.4	6.7	4.0	6.0	3.1
	35	9.3	9.5	10.3	21.1	6.3	7.7	5.0	7.0	3.0
	40	11.0	10.7	11.5	24.3	6.6	8.8	5.8	7.7	3.4
	45	12.2	12.6	12.1	27.6	7.4	9.6	6.0	8.3	3.8
	50	13.5	13.8	13.3	29.4	8.3	9.8	7.4	9.2	4.7
robot	10	3.3	3.5	3.6	6.6	2.5	2.9	2.5	2.8	3.0
	15	4.7	4.5	4.8	8.7	2.9	4.4	3.0	3.7	3.0
	20	6.0	6.6	6.0	12.3	3.9	5.4	3.8	4.9	3.0
	25	7.4	7.0	7.1	15.4	4.6	5.9	4.3	5.5	3.0
	30	8.6	8.0	8.6	17.2	5.4	6.8	4.0	6.0	3.0
	35	9.3	9.5	10.3	21.1	6.4	7.7	5.0	7.0	3.0
	40	11.0	10.7	11.5	24.3	6.7	8.8	5.9	7.7	3.0
	45	12.2	12.6	12.1	27.6	7.4	9.6	6.0	8.3	3.1
	50	13.5	13.8	13.3	29.4	9.0	9.8	7.5	9.2	4.3

Table IV compares the number of branch-and-bound nodes in the search tree of the two algorithms for small instances. The values are not reported in the case that the counter overflowed due to the size of the instances. It is seen that the search trees are of the same magnitude for the two variants of the problem. Finally, Table V compares the average solution times, in seconds, computed over all the instances solved to optimality within the time limit of 300,000 seconds. Although solution times cannot be compared between the two versions if not all instances are solved by both of them, it is confirmed that the general packing (resp. robot-packing) algorithm is slightly better for instances with relatively small (resp. large) item sizes.

5 Conclusion

We have pointed out an inaccuracy in the paper by Martello, Pisinger and Vigo [7], showing that the proposed algorithm only considers robot packings, and not all general packings. A new MILP model has been presented which makes it possible to correct the algorithm in [7]. It has been shown that constraint programming is a promising approach for modeling and solving the one-bin packing problem. The very extensive computational experiments have made it possible to

Table IV: Number of investigated nodes in the branch-and-bound tree.

		Class								
packing	*n*	1	2	3	4	5	6	7	8	9
general	10	11	2	8	0	1	3	3	2	0
	15	34	18	121	1	4	8	4	5	0
	20	116	649	71	2	5	14	8	7	4
	25	446	623	1095	8	9	26	21	19	20
	30	419780	6392	103572	19	14	1492	41	27	77
	35	4179435	1828851	1125642	79	18	684	118	177	106
robot	10	11	2	8	0	1	3	3	2	0
	15	34	18	121	1	4	8	4	6	0
	20	116	649	71	2	5	15	8	7	4
	25	446	625	1095	8	10	28	23	21	20
	30	419905	6392	103572	19	16	1752	40	28	113
	35	4179409	1865316	1125706	79	27	640	122	184	107

Table V: Average solution times in seconds, computed for the instances solved to optimality within the time limit of 300,000 seconds.

		Class								
packing	*n*	1	2	3	4	5	6	7	8	9
general	10	0.02	0.01	0.02	0.00	0.07	0.04	0.07	0.06	0.00
	15	0.03	0.03	0.06	0.00	0.15	0.08	0.14	0.11	0.00
	20	0.08	0.11	0.07	0.01	0.21	0.17	0.45	0.16	0.07
	25	0.30	0.25	0.68	0.02	0.52	1.56	6.44	0.56	0.57
	30	57.14	4.22	12.82	0.08	4.98	28.83	395.53	1.04	111.94
	35	224.87	380.59	91.34	0.08	13.04	6.94	4092.04	5787.13	229.47
	40	759.50	1083.92	645.78	0.10	5247.60	64.60	15850.02	4023.11	38073.20
	45	6227.55	29763.78	63820.07	0.14	10701.19	6709.44	34492.75	4575.79	237947.20
	50	32355.35	44822.16	16411.43	12.97	55402.25	6681.26	21709.16	508.01	–
robot	10	0.02	0.01	0.01	0.00	0.07	0.04	0.06	0.06	0.00
	15	0.02	0.02	0.03	0.00	0.12	0.08	0.12	0.15	0.01
	20	0.10	0.05	0.05	0.01	0.21	0.24	0.19	0.15	0.06
	25	0.52	0.13	0.88	0.02	4.49	0.40	8528.78	0.41	0.17
	30	37.54	4.26	4.67	0.07	0.53	7.74	27547.76	0.63	26.68
	35	93.97	101.62	20.15	0.09	42576.91	2.11	47281.62	33248.95	92.85
	40	281.79	304.04	13054.89	0.11	50593.16	18.61	71820.94	341.95	1221.78
	45	1767.06	6452.40	18538.22	0.15	27639.62	36.83	134727.58	6782.45	99269.02
	50	4775.12	30719.51	4756.58	11.09	3486.47	34.06	62678.03	38693.36	176005.47

find an optimal solution for more problem instances than previously presented in the literature, thus making it possible to study quite large instances with respect to the quality of upper and lower bounds (see [11] for more detailed computational results). The extensive computational experiments are also promising from a different point of view as they show that the solution times do not grow as fast as one could expect. This could indicate that in a couple of years we will be able to systematically solve instances with 50 items to optimality.

6 Acknowledgements

The first author would like to thank Sun Microsystems for providing resources for the computational experiments.

References

[1] R. Barták (1998), "Online Guide to Constraint Programming", available at `http://kti.mff.cuni.cz/~bartak/constraints/`.

[2] J.O. Berkey and P.Y. Wang (1987) "Two Dimensional Finite Bin Packing Algorithms", Journal of the Operational Research Society 38, 423–429.

[3] S.C. Brailsford, C.N. Potts and B.M. Smith (1999), "Constraint Satisfaction Problems: Algorithms and Applications", European Journal of Operational Research 119, 557–581

[4] C.S. Chen, S.M. Lee and Q.S. Shen (1995), "An Analytical Model for the Container Loading Problem", European Journal of Operational Research 80, 68–76.

[5] J. Gaschnig (1979). "Performance Measurement and Analysis of Certain Search Algorithms" Tech. rep. CMU-CS-79-124, Carnegie-Mellon University, 1979.

[6] R.M. Haralick and G.L. Elliot (1980) "Increasing Tree Search Efficiency for Constraint Satisfaction Problems", Artificial Intelligence 14, 263–313.

[7] S. Martello, D. Pisinger and D. Vigo (2000), "The Three-Dimensional Bin Packing Problem", Operations Research 48, 256–267.

[8] S. Martello and D. Vigo (1998), "Exact Solution of the Two-Dimensional Finite Bin Packing Problem", Management Science 44, 388–399.

[9] G. Mitra, C. Lucas, S. Moody and E. Hadjiconstantinou (1994), "Tools for Reformulating Logical Forms into Zero-One Mixed Integer Programs", European Journal of Operational Research 72, 263–277.

[10] H. Onodera, Y. Taniguchi and K. Tamaru (1991), "Branch-and-Bound Placement for Building Block Layout", *Proceedings 28th ACM/IEEE Design Automation Conference, 433–439.*

[11] D. Pisinger, E. den Boef, J. Korst, S. Martello and D. Vigo (2001), “Robot-Packable and General Variants of the Three-Dimensional Bin Packing Problem”, Technical report 01/05, DIKU, University of Copenhagen, Denmark.

[12] D. Sabin and E.C. Freuder (1994), “Contradicting Conventional Wisdom in Constraint Satisfaction”, In: A. Borning (ed.), *Proceedings of the Second International Workshop on Principles and Practice of Constraint Programming, PPCP’94, Rosario, Orcas Island, Washington, USA*, 874, 10–20.