Algorithm 864: General and robot-packable variants of the three-dimensional bin packing problem.

Article · January 2007

Source: DBLP

CITATIONS
56

READS
1,323

5 authors, including:

David Pisinger
Technical University of Denmark
169 PUBLICATIONS **11,040** CITATIONS
SEE PROFILE

Daniele Vigo
University of Bologna
176 PUBLICATIONS **11,530** CITATIONS
SEE PROFILE

Edgar den Boef
14 PUBLICATIONS **160** CITATIONS
SEE PROFILE

Jan Korst
Philips
141 PUBLICATIONS **4,493** CITATIONS
SEE PROFILE

Algorithm 864: General and Robot-Packable Variants of the Three-Dimensional Bin Packing Problem

SILVANO MARTELLO
University of Bologna
DAVID PISINGER
University of Copenhagen
DANIELE VIGO
University of Bologna
and
EDGAR DEN BOEF and JAN KORST
Philips Research Laboratories

We consider the problem of orthogonally packing a given set of rectangular-shaped boxes into the minimum number of three-dimensional rectangular bins. The problem is NP-hard in the strong sense and extremely difficult to solve in practice. We characterize relevant subclasses of packing and present an algorithm which is able to solve moderately large instances to optimality. Extensive computational experiments compare the algorithm for the three-dimensional bin packing when solving general orthogonal packings and when restricted to robot packings.

Categories and Subject Descriptors: G.4 [**Mathematical Software**]: *Algorithm design and analysis*; G.2.1 [**Discrete Mathematics**]: Combinatorics—*Combinatorial algorithms*

General Terms: Algorithms, Experimentation

Additional Key Words and Phrases: Bin packing, exact algorithms, constraint programming, branch-and-bound

S. Martello and D. Vigo were supported by the Italian Ministry of Education, University and Research (MIUR).
D. Pisinger thanks The Danish Natural Science Research Council (SNF) for the support of this project.
Authors' current addresses: S. Martello and D. Vigo, DEIS, University of Bologna, Viale Risorgimento 2, 40136 Bologna, Italy; email: {smartello,dvigo}@deis.unibo.it; D. Pisinger, DIKU, University of Copenhagen, Universitetsparken 1, DK-2100 Copenhagen, Denmark; email: pisinger@diku.dk; E. den Boef, Quintiq, Goudsbloemvallei 12-28 5237, MJs-Hertogenbosch, The Netherlands; email: edgar.den.boef@quintiq.com; J. Korst, Philips Research Laboratories Eindhoven, Building WY 2.23, Prof. Holstlaan 4, 5656 AA Eindhoven, The Netherlands; email: jan.korst@philips.com.

ACM Reference Format:
Martello, S., Pisinger, D., Vigo, D., den Boef, E., and Korst, J. 2007. Algorithm 864: General and robot-packable variants of the three-dimensional bin packing problem. ACM Trans. Math. Softw. 33, 1, Article 7 (March 2007), 12 pages DOI = 10.1145/1206040.1206047 http://doi.acm.org/10.1145/1206040.1206047

1. INTRODUCTION

The *Three-Dimensional Bin Packing Problem* asks for an orthogonal packing of a given set of rectangular-shaped boxes into a minimum number of three-dimensional rectangular bins. Each *box* j ($j = 1, \ldots, n$) is characterized by a width w_j, height h_j, and depth d_j. An unlimited number of identical three-dimensional *bins*, having width W, height H, and depth D is available. The boxes have fixed orientation, that is, they may not be rotated, but no further restriction is imposed. Such packings will be hereafter denoted as *general* packings. Without loss of generality, we assume in the following that all input data is positive integers. Coordinates originate from the bottom-left-behind corner of a bin, and (x_j, y_j, z_j) is the point where the bottom-left-behind corner of box j is positioned.

The problem has several industrial applications in cutting and loading contexts. The reader is referred to Dyckhoff et al. [1997] for an annotated bibliography on this subject, and to Lodi et al. [2002a, 2002b] for recent surveys.

In some relevant practical applications, it is demanded that the boxes can be obtained from the bin through a sequence of *guillotine cuts*, that is, face-to-face cuts parallel to the faces of the bin: these packings are said to be *guillotine cuttable*. In other industrial contexts, boxes have to be packed by robots equipped with a "hand" parallel to the base of the bins, which is covered with vacuum cells for lifting the boxes. When the boxes have to be palletized or packed into a container that is open from three sides while loading, the robot is usually located at one of the corners. To simplify the packing operations, and to avoid collisions between the hand and the boxes, it can be demanded that no already packed box be positioned in front of, right of, or above the destination of the current box: a *robot packing* is a packing which can be achieved by successively placing boxes starting from the bottom-left-behind corner, and such that each box is in front of, right of, or above each of the previously placed boxes.

Note that each guillotine-cuttable packing is also a robot packing. Consider a guillotine-cuttable packing and a feasible sequence (tree) of cuts: by first packing the boxes in the bottom, left, or back part (depending on the cutting direction) of each cut, a feasible robot packing is obtained. The converse is not true, that is, not all robot packings are guillotine cuttable, and some sets of items can be robot packable even though they are not guillotine cuttable, as shown by the example depicted in Figure 1.

Martello et al. [2000] proposed a branch-and-bound algorithm for the three-dimensional bin packing problem. den Boef et al. [2005] have shown that this algorithm correctly solves the robot packing variant of the problem, but does not generate all orthogonal packings, thus possibly failing the detection of an

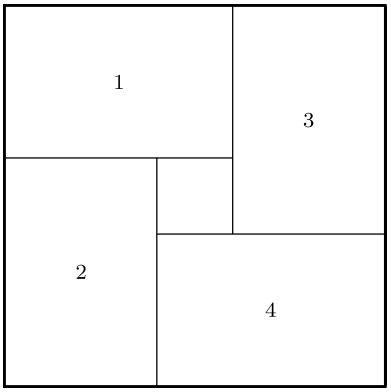

Fig. 1. A two-dimensional robot packing that is not guillotine cuttable. The four items cannot be packed in the given square bin so as to be guillotine cuttable.

optimal general packing. This behavior is due to an inner procedure used to determine an optimal packing for a single bin.

We present here an extension of the algorithm in Martello et al. [2000], obtained through a new single-bin packing procedure that combines the original enumerative approach with a new constraint programming approach, so that general as well as robot-packable versions of the problem can be solved. The corresponding C language implementation, `binpack3d` is provided. In Section 2 we give an overview of the main algorithm, while Section 3 describes the single-bin filling subproblem. Section 4 presents the overall structure of the algorithm. Finally, Section 5 reports computational experiments to evaluate the average performance of the code, as an algorithm for solving both general and robot-packable variants of the problem. Section 6 presents our conclusions and discusses future research topics.

2. THE ALGORITHM

The algorithm is based on the two-level decomposition approach presented by Martello and Vigo [1998] for the two-dimensional bin packing problem. A *main branching tree* assigns the boxes to the bins, without specifying their actual position. The actual feasibility of this assignment is then checked by an inner level. The boxes are initially sorted by nonincreasing volume.

The lower bound used by the algorithm is L_2, whose detailed description is given in Martello et al. [2000]. Bound L_2 is determined through three lower bound computations, each relative to a different dimension of bins and boxes. Consider, say, the depth, and let $p \leq W/2$ and $q \leq H/2$ be two given threshold values. The corresponding bound $L_2^{WH}(p, q)$ is obtained by extracting from the box set three subsets of boxes: (i) *very large* (those with $w_j > W - p$ and $h_j > H - q$); (ii) *large* (among the remaining boxes, those with $w_j > W/2$ and $h_j > H/2$); (iii) *small* (among the remaining boxes, those with $w_j \geq p$ and $h_j \geq q$). The other (smaller) boxes are disregarded. The bound is then computed, on the basis of the depths of the considered boxes, by determining, through a simpler bound L_1^{WH}, the minimum number of bins needed to allocate the

large and very large boxes, as well as the minimum number of additional bins needed for the small ones. Lower bound L_2^{WH} is then the maximum $L_2^{WH}(p,q)$ value, computed over all (p,q) pairs. The two other bounds, L_2^{WD} (respectively L_2^{HD}) are obtained in the same way, by interchanging h_j (respectively w_j) with d_j and H (respectively W) with D, thus providing the overall bound $L_2 = \max\{L_2^{WH}, L_2^{WD}, L_2^{HD}\}$.

At the root node, we execute in sequence two heuristic algorithms, H1 and H2, whose detailed descriptions are given in Martello et al. [2000]. Algorithm H1 is based on a level approach where a number of layers of dimension $W \times H \times d$, $(d \leq D)$ are constructed: the resulting layers are then combined into full bins of depth D by solving a one-dimensional bin packing problem defined on the depths of the layers. Algorithm H2 is a greedy approach which fills every bin as far as possible, by heuristically solving a single-bin filling problem in which the objective is to maximize the volume filled. These algorithms can produce different solutions by rotating the dimensions of bins and boxes; hence the computations are performed three times, and the best solution is selected as the incumbent. If lower bound L_2 does not prove that this solution is optimal, the enumeration begins.

The search is recursively performed, following a depth-first strategy, by an algorithm called `rec_binpack`. At each decision node, let u denote the incumbent solution value and $M = \{1, \ldots, m\}$ the set of bins used to allocate boxes in the ascendant nodes. A bin of M is called *open* if further boxes may be possibly placed into it; otherwise it is called *closed*. The branching phase assigns the next free box, in turn, to all the open bins; in addition, if $|M| < u - 1$ the box is also assigned to a new bin (hence opening it).

As previously mentioned, the exploration of a generated node requires checking the actual feasibility of the associated assignment. This is performed by an algorithm called `onebin_decision`. Assume that the node has assigned a box k to a bin i already containing a nonempty subset J of boxes such that $\sum_{j \in J \cup \{k\}} w_j h_j d_j \leq WHD$. The subproblem of establishing whether $J \cup \{k\}$ can be packed into a single bin is in itself NP-hard. We solve it through the following steps.

(1) Compute lower bound L_2 for the subproblem;
(2) if $L_2 > 1$ then fathom the node (no feasible packing exists); otherwise,
(3) if robot packing is desired then
    (3.1) execute, for the subproblem, a branch-and-bound search to enumerate all single-bin robot packings through algorithm `onebin_robot`, described in detail in Section 3.1; otherwise, that is, if general packing is desired,
    (3.2) execute, for the subproblem, algorithm `onebin_general`, a constraint programming-based approach described in detail in Section 3.2, to enumerate all possible packings;
(4) if a single-bin packing is found, accept the assignment; otherwise, fathom the node.

When the assignment of box k to bin i is accepted, an attempt is made to close bin i. To this end, for each unassigned box k', we compute lower bound

L_2 for the subinstance defined by the boxes of $J \cup \{k\} \cup \{k'\}$. If $L_2 > 1$ for each k', then we know that no further box can be placed into i and we close the bin. If this fails, let K be the subset of those unassigned boxes k' for which we have obtained $L_2 = 1$: if heuristic H1 or H2 finds a single-bin solution for the subinstance defined by the boxes in $J \cup \{k\} \cup K$, then we know that no better placing is possible for these boxes, so we assign all of them to bin i and close it.

Whenever a bin is closed, lower bound L_2 is computed for the instance defined by all the boxes not currently assigned to closed bins: if $u \leq L_2 + c$, where c is the number of closed bins, we backtrack.

3. THE SINGLE-BIN FILLING SUBPROBLEM

When the exploration of a node is not concluded by Steps 1–2 above, the feasibility of the assignment of the given subset of boxes to a single bin is determined, at Step 3, through one out of two complete enumeration searches.

Computational experiments showed that in many cases the algorithm is executed for a very small set of boxes. Hence, we found it fruitful to implement specialized routines for instances with up to three boxes, based on direct enumeration of all distinct placings of the boxes, avoiding symmetric or dominated solutions like those described by Scheithauer [1997].

3.1 Search for a Feasible Robot Packing

The `onebin_robot` algorithm establishes whether the set of boxes currently assigned to a bin can actually be packed into it, by restricting the search to robot packings. This is obtained by considering the problem of finding a solution that packs the largest possible volume into the bin. In Martello et al. [2000], it was shown that only solutions where the boxes are moved backward, downward, and leftward as far as possible need be considered.

Within such settings, a new box may be placed at any intersection between three boxes (or bin sides); hence a naive test would demand $O(n^3)$ time. An $O(n^2)$ test can be obtained by observing that every subsolution defines an *envelope* below which no new box can be placed, and where a set of *corner points* defines the undominated positions for placing a new box (see Figure 2). At any decision node, let V_p be the volume of the already placed boxes, and V_e the volume below the envelope. Observe that $V_p \leq V_e$ since the envelope may identify some encapsulated volume that cannot be used (like that within the dotted lines in Figure 2). Since no solution that packs a volume larger than $WHD - (V_e - V_p)$ can be obtained by the descendant nodes, if this upper bound does not exceed the current best solution, we may backtrack. The branching operation is easily performed by recursively placing each unpacked box, in turn, into all the feasible corner points. The incumbent solution is updated whenever a pattern is found which fills a larger volume.

At each decision node, the envelope and the set of feasible corner points are obtained, in $O(n^2)$ time, by determining a sequence of $O(n)$ two-dimensional envelopes and corner points, as shown in Martello et al. [2000].

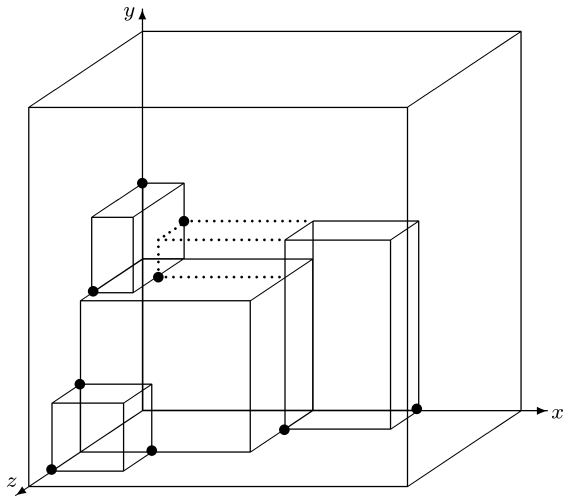

Fig. 2. Envelope for the robot-packing single-bin filling. Black dots show the corner points for future box insertions. Dotted lines show an encapsulated volume that cannot be used.

3.2 Search for a Feasible General Packing

The second packing algorithm, `onebin_general`, establishes whether a given set of boxes can be packed into the bin using general packing patterns. Unlike robot packing, we cannot fill the bin from one corner. Instead, a solution approach based on constraint programming has been implemented.

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 et al. [1999]). A packing is feasible if all boxes are placed within the bin, and no two boxes overlap. The latter criteria is satisfied if for each pair of boxes i, j, box i is left of, right of, under, above, behind, or in front of box j. This means that for each pair of boxes i, j we may introduce the set of relative positions $M_{ij} = \{$ `l, r, u, a, b, f` $\}$. To avoid symmetries, boxes 1 and 2 have the restricted domain $M_{12} = \{$`l, u, b`$\}$.

The problem is solved recursively by an algorithm, `onebin_general`, where at each iteration two boxes i and j are considered, and one of the relative positions in M_{ij} is imposed. The feasibility of the currently 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 boxes i, j have had imposed on them a relation, and a feasible assignment of coordinates exists such that no box exceeds the bin, the algorithm terminates with the value `true`. If the recursive search did not find a feasible packing, `onebin_general` returns the value `false`.

The feasibility of a partially ordered problem is checked separately for each dimension. For symmetry reasons, we will only describe the algorithm for the x-coordinates. Hence, assume that those boxes that are forced to be left or right of each other are sorted in topological order as follows: if $M_{ij} = \{$ `l` $\}$ then $i \prec j$

and if $M_{ij} = \{ \texttt{r} \}$ then $j \prec i$. Next, for a given box j, its position x_j is found iteratively as

$$x_j = \max_{i \,:\, i \prec j} x_i + w_i$$

while x_j is set to zero if we maximize over the empty set. 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 $O(n^2)$ time as there are $O(n^2)$ relations `l, r`. If $x_j + w_j > W$ for some j then box j exceeds the bin, and we backtrack.

To further speed up the solution algorithm, domain reduction and constraint propagation are applied. Domain reduction is obtained by the Forward Check (FC) approach proposed by Haralick and Elliot [1980]. Having chosen a relation between boxes i and j, we may remove all other relations from M_{ij} as we know that only one relation needs to be chosen.

We use the Maintaining Arc Consistency (MAC) approach proposed by Gaschnig [1979] 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 [1994] 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 algorithm `onebin_general` (i.e., for each new assignment of a relation) we run through all pairs of boxes i and j with $|M_{ij}| \geq 2$ and test the 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 `onebin_general`. If only one relation is left in a domain M_{ij} then this relation is fixed. Having fixed a relation, in order to maintain arc consistency it may be necessary to run through an additional reduction. However, preliminary computational experiments showed that this does not pay off. All reductions in the domain achieved by the forward propagation are pushed to a stack, such that the domains can be quickly restored upon backtracking from `onebin_general`.

The best performance of algorithm `onebin_general` was obtained by sorting the boxes according to nonincreasing volume, and then branching on pairs of boxes $(1, 2)$, $(1, 3)$, $(2, 3)$, $(1, 4)$, $(2, 4)$, $(3, 4)$, $(1, 5)\ldots$. In this way the, relative position of the largest boxes was settled at an early stage of the algorithm, and infeasibility could quickly be detected. This complies with the First Fail Strategy proposed by Barták [1998].

```
1.   copy the input information to internal structures;
2.   compute lower bound L_2;
3.   execute heuristics H1 and H2, and set u to the best solution value found;
4.   while no optimal solution is found or stopping criterion is met do
          comment: perform the tree enumeration;
4.1       assign the next box to an open bin or a new bin;
4.2       check the feasibility of the assignment with onebin_decision;
4.3       if the assignment is not feasible then backtrack
          else
               check the current solution for (non-)optimality;
               check node limit and time limit;
               for each open bin do
                    if the bin can be closed then
                         compute a new lower bound and possibly backtrack
               end for
          end if
5.   return the best solution found.
```

Fig. 3. Structure of algorithm `binpack3d` .

4. PSEUDOCODE VERSION OF BINPACK3D

For a better understanding of algorithm `binpack3d`, we report its general structure in Figure 3. Initial lower bounds and heuristic solutions are computed at the root node (see Section 2). If the problem is not solved to optimality, the tree enumeration is recursively performed through algorithm `rec_binpack` (see Sections 2 and 3).

As observed in Section 2, different upper- and lower-bound values may be obtained by considering the instance according to the three different orientations. To this purpose, the computations are performed for a single orientation: the overall lower- and upper-bound values are then obtained making use of a routine that rotates the dimensions as $w \leftarrow h \leftarrow d \leftarrow w$ and $W \leftarrow H \leftarrow D \leftarrow W$.

5. COMPUTATIONAL EXPERIMENTS

For each class, we considered instances of different size, with n ranging between 10 and 50. The code was implemented in ANSI-C, and the experiments were run on Pentium 4 running at 3 GHz. A time limit of 3600 s was given to each instance, and 10 instances were solved for each class and size of a problem.

The experiments were performed on the nine classes of random instances described in Martello et al. [2000]. For the first five classes, the bin size was $W = H = D = 100$ and five types of boxes were considered by uniformly

Table I. Number of Instances, out of 10, Solved to Proved Optimality in 3600 s

Packing	n	Class 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	9	9
	35	10	10	10	10	10	10	8	9	9
	40	10	9	8	10	9	10	7	9	3
	45	8	6	4	10	6	9	5	9	0
	50	7	3	7	10	5	10	6	9	0
Total: 723		85	78	79	90	80	89	76	85	61
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	8	10	7	9	10
	30	10	10	10	10	9	10	7	7	10
	35	10	10	10	10	5	10	5	8	10
	40	10	9	8	10	4	10	2	10	9
	45	10	7	4	10	4	10	1	8	2
	50	7	4	9	10	3	10	3	5	0
Total: 694		87	80	81	90	63	90	55	77	71

randomly generating the box sizes in different intervals, namely:

—*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: $w_j \in [1, \frac{1}{2}W]$, $h_j \in [1, \frac{1}{2}H]$, $d_j \in [1, \frac{1}{2}D]$.

Classes k ($k = 1, \ldots, 5$) were then obtained by generating each box according to type k with probability 60%, and according to the other four types with probability 10% each. Classes 6 to 8 may be described as follows:

—*Class* 6: $W = H = D = 10$; w_j, h_j, d_j uniformly random in $[1, 10]$;
—*Class* 7: $W = H = D = 40$; w_j, h_j, d_j uniformly random in $[1, 35]$;
—*Class* 8: $W = H = D = 100$; w_j, h_j, d_j uniformly random in $[1, 100]$.

Finally, *Class* 9 consisted of difficult *all-fill* instances having a known solution with three bins: the boxes were generated by randomly cutting the bins into smaller parts as follows. Bins 1 and 2 were cut into $\lfloor n/3 \rfloor$ boxes each, while bin 3 was cut into $n - 2\lfloor n/3 \rfloor$ boxes. The cutting was made using a recursive approach which repeatedly cuts the bin into two parts using a guillotine cut until five boxes remain. These boxes were then cut using a nonguillotine pattern.

The entries in Table I give the number of instances, out of 10, solved to optimality using a time limit of 3600 s. The bottom line reports the total number of instances solved to optimality for each class and packing version. The general

Table II. Average Solution Times in Seconds, as Average of 10 Instances Using a Time Limit of 3600 s

Packing	n	Class 1	2	3	4	5	6	7	8	9
General	10	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	15	0.00	0.00	0.00	0.00	0.01	0.00	0.02	0.00	0.00
	20	0.01	0.01	0.01	0.00	0.01	0.01	0.16	0.00	0.01
	25	0.05	0.04	0.10	0.00	0.09	0.19	1.76	0.91	0.11
	30	15.18	1.09	3.50	0.00	1.41	7.34	93.73	360.35	468.81
	35	61.07	102.39	26.30	0.00	3.01	2.90	764.97	390.11	546.46
	40	217.87	614.06	861.15	0.00	448.32	15.79	1151.20	365.25	2859.09
	45	1049.05	1493.49	2163.92	0.01	1464.79	363.07	1952.55	450.54	3600.01
	50	1391.89	2696.83	1712.14	4.00	1971.17	4.22	1604.58	434.98	3600.02
Robot	10	0.00	0.00	0.00	0.00	0.00	0.00	7.81	0.00	0.00
	15	0.00	0.00	0.00	0.00	0.04	0.01	4.85	0.02	0.00
	20	0.01	0.01	0.01	0.00	3.96	0.04	207.76	4.31	0.00
	25	0.11	0.03	0.18	0.00	843.37	0.09	1425.30	361.05	0.05
	30	10.41	1.01	1.90	0.00	381.42	5.00	1308.55	1080.11	19.78
	35	36.67	35.75	11.03	0.00	1819.74	1.53	2033.36	950.86	74.75
	40	114.53	449.87	766.76	0.00	2184.86	12.84	2881.79	136.94	1250.15
	45	748.34	1299.86	2162.28	0.01	2161.50	11.28	3240.01	751.35	3247.02
	50	1262.65	2523.43	1333.56	3.52	2532.84	5.49	2742.58	1875.95	3600.01

Table III. Average Number of Branch-and-Bound Nodes (in Thousands)

Packing	n	Class 1	2	3	4	5	6	7	8	9
General	10	0	0	0	0	0	0	0	0	0
	15	0	0	0	0	0	0	0	0	0
	20	0	0	0	0	0	0	0	0	0
	25	0	0	1	0	0	0	0	0	0
	30	419	6	103	0	0	1	0	0	0
	35	4179	1829	1125	0	0	0	0	0	0
	40	17163	4146	21409	0	0	4	0	0	0
	45	31083	38449	60636	1	0	19	0	0	0
	50	45604	84678	60629	674	0	20	0	3	0
Robot	10	0	0	0	0	0	0	0	0	0
	15	0	0	0	0	0	0	0	0	0
	20	0	0	0	0	0	0	0	0	0
	25	0	0	1	0	0	0	0	0	0
	30	420	6	103	0	0	1	0	0	0
	35	4179	1863	1125	0	0	0	0	0	0
	40	17163	9402	40950	0	0	4	0	0	0
	45	51837	59719	115812	1	0	19	0	0	0
	50	65470	192349	80268	674	0	20	0	3	0

algorithm was able to solve 723 out of 810 instances, while the robot-packable algorithm was able to solve 694 of them. The difference may be explained by the fact that the general version has more freedom in arranging the boxes and thus may find solutions using fewer bins. In this way, having tighter upper bounds, the algorithm may terminate faster as the lower bounds used by both algorithms are the same. For Class 9, the picture changed as the robot-packable algorithm was able to solve 71 instances while the general algorithm only solved 61 instances (out of 90). This behavior can be explained by the fact that the

Table IV. Average Number of Iterations in Procedure `onebin_decision` (in Thousands)

Packing	n	Class 1	2	3	4	5	6	7	8	9
General	10	0	0	0	0	0	0	0	0	0
	15	2	3	9	0	0	0	1	0	1
	20	10	22	9	0	0	1	5	0	8
	25	39	42	94	0	2	16	36	68	59
	30	12224	820	3876	0	5	1627	42912	149919	193643
	35	27063	102944	31566	1	7	1715	309928	102222	363543
	40	206458	598374	720682	8	166802	9161	430270	167427	1267382
	45	1098733	1621442	1795338	24	674970	205247	816829	193217	1579066
	50	1676059	3062527	2299310	17432	861892	3718	672255	152052	1266272
Robot	10	0	0	0	0	0	3	4123	0	0
	15	1	2	3	0	16	4	2467	11	0
	20	23	14	6	0	2098	27	100877	2087	4
	25	127	35	223	0	285958	55	513293	174724	45
	30	13651	1455	2742	0	129160	3024	400023	401562	14725
	35	23214	54554	20954	2	548005	1168	895675	511179	28287
	40	160077	644205	785467	12	1387156	13591	1436016	94393	493016
	45	1137480	2139452	2439088	27	1576640	8260	1781172	509878	1343846
	50	1966457	4570412	1990263	17027	989142	3821	975509	878014	1380453

generated instances were all robot packable, so the robot packing version had the advantage that it only needed to search the relevant solution space. As a whole, Classes 4 and 6 were generally the easiest to solve while Class 9 tended to be the most difficult.

Table II gives the average CPU times with a time limit of 3600 s. If an instance was not solved to proved optimality within the time limit, the time spent is included in the computation. It can be seen that most instances up to 30 items were solved within a reasonable time. Classes 4 and 6 were particularly easy to solve, and the solution times seldom exceeded a few seconds.

Table III reports the number of branch-and-bound nodes in the main search tree, measured in thousands. Table IV shows the corresponding number of iterations in procedure `onebin_decision`, measured in thousands. It can be seen that Classes 1 to 3 demanded quite a lot of branch-and-bound nodes, while Classes 5 to 9 seldom branched but mainly spent time in the `onebin_decision` procedure. Class 4 used neither many branch-and-bound nodes nor many iterations in `onebin_decision`.

6. CONCLUSIONS AND FUTURE RESEARCH TOPICS

To our knowledge, the present code is the first implementation of an algorithm for the three-dimensional bin-packing problem. The code may be used as a whole for solving this kind of problem, or the exact algorithm for filling a single bin may be used as a subproblem in other applications. The computational experiments in the present article, as well as those in Martello et al. [2000], have demonstrated that the algorithm finds good solutions within a short computing time.

A closer study of the algorithm shows that more than 95% of the time is spent in routines related to the single-bin filling problem. Hence future research should be focused on solving this decision problem.

REFERENCES

BARTÁK, R. 1998. Online guide to constraint programming. Available online at http://kti.mff.cuni.cz/~bartak/constraints/.

BRAILSFORD, S., POTTS, C., AND SMITH, B. 1999. Constraint satisfaction problems: Algorithms and applications. *European J. Operat. Res. 119*, 557–581.

DEN BOEF, E., KORST, J., MARTELLO, S., PISINGER, D., AND VIGO, D. 2005. Erratum to the three-dimensional bin packing problem: Robot-packable and orthogonal variants of packing problems. *Operat. Res. 53*, 735–736.

DYCKHOFF, H., SCHEITHAUER, G., AND TERNO, J. 1997. Cutting and Packing. In *Annotated Bibliographies in Combinatorial Optimization*, M. Dell'Amico, F. Maffioli, and S. Martello, Eds. John Wiley & Sons, Chichester, England, 393–413.

GASCHNIG, J. 1979. Performance measurement and analysis of certain search algorithms. Tech. rep. CMU-CS-79-124. Carnegie-Mellon University, Pittsburgh, PA.

HARALICK, R. AND ELLIOT, G. 1980. Increasing tree search efficiency for constraint satisfaction problems. *Artific. Intell. 14*, 263–313.

LODI, A., MARTELLO, S., AND MONACI, M. 2002a. Two-dimensional packing problems: A survey. *European J. Operat. Res. 141*, 3–13.

LODI, A., MARTELLO, S., AND VIGO, D. 2002b. Recent advances on two-dimensional bin packing problems. *Discr. Appl. Math. 123*, 379–396.

MARTELLO, S., PISINGER, D., AND VIGO, D. 2000. The three-dimensional bin packing problem. *Operat. Res. 48*, 256–267.

MARTELLO, S. AND VIGO, D. 1998. Exact solution of the two-dimensional finite bin packing problem. *Manage. Sci. 44*, 388–399.

SABIN, D. AND FREUDER, E. 1994. Contradicting conventional wisdom in constraint satisfaction. In *Proceedings of the Second International Workshop on Principles and Practice of Constraint Programming, PPCP'94, Rosario, Orcas Island, WA, USA, May 1994*, A. Borning, Ed. Lecture Notes in Computer Science, vol. 874. Springer-Verlag, Berlin and Heidelberg, Germany, 10–20.

SCHEITHAUER, G. 1997. Equivalence and dominance for problems of optimal packing of rectangles. *Ricerca Operat. 83*, 3–34.

Received May 2003; revised January 2006; accepted April 2006