The Three-Dimensional Bin Packing Problem

Silvano Martello[*], **David Pisinger**[†], **Daniele Vigo**[*]
[*]DEIS, Univ. of Bologna, Viale Risorgimento 2, Bologna
[†]DIKU, Univ. of Copenhagen, Univ.parken 1, Copenhagen

Abstract

The problem addressed in this paper is that of orthogonally packing a given set of rectangular-shaped items into the minimum number of three-dimensional rectangular bins. The problem is strongly NP-hard and extremely difficult to solve in practice. Lower bounds are discussed, and it is proved that the asymptotic worst-case performance ratio of the continuous lower bound is $\frac{1}{8}$. An exact algorithm for filling a single bin is developed, leading to the definition of an exact branch-and-bound algorithm for the three-dimensional bin packing problem, which also incorporates original approximation algorithms. Extensive computational results, involving instances with up to 90 items, are presented: it is shown that many instances can be solved to optimality within a reasonable time limit.

1 Introduction

We are given a set of n rectangular-shaped *items*, each characterized by width w_j, height h_j and depth d_j ($j \in J = \{1, \dots, n\}$), and an unlimited number of identical three-dimensional containers (*bins*) having width W, height H and depth D. The *Three-Dimensional Bin Packing Problem* (3D-BPP) consists of orthogonally packing all the items into the 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. We also assume, without loss of generality, that all the input data are positive integers satisfying $w_j \leq W$, $h_j \leq H$ and $d_j \leq D$ ($j \in J$). No further restriction is present: the items need not be packed in layers, and we do not impose the so-called *guillotine* constraint, which requires that the patterns be such that the items can be obtained by sequential face-to-face cuts parallel to the faces of the bin (patterns satisfying such constraint constitute *guillotine packings*). Problem 3D-BPP is strongly NP-hard, since it is a generalization of the well-known (one-dimensional)

Bin Packing Problem (1D-BPP), in which a set of n positive values w_j has to be partitioned into the minimum number of subsets so that the total value in each subset does not exceed a given bin capacity W. It is clear that 1D-BPP is the special case of 3D-BPP arising when $h_j = H$ and $d_j = D$ for all $j \in J$. Another important related problem arises when $d_j = D$ for all $j \in J$: we have in this case the *Two-Dimensional Bin Packing Problem* (2D-BPP), calling for the determination of the minimum number of identical rectangular bins of size $W \times H$ needed to pack a given set of rectangles of sizes $w_j \times h_j$ $(j \in J)$.

For a general classification of packing and loading problems we refer to Dyckhoff [8], Dyckhoff and Finke [9], and Dyckhoff, Scheithauer and Terno [10]. Further details on 1D-BPP can be found in Martello and Toth [16] and Coffman, Garey and Johnson [6], while for 2D-BPP the reader is refereed to Martello and Vigo [18].

The 3D-BPP is closely related to other three-dimensional container loading problems:

- *Knapsack Loading.* In the knapsack loading of a container each item has an associated profit, and the problem is to choose a subset of the items that fits into a single container (bin) so that maximum profit is loaded. If the profit of an item is set to its volume, this corresponds to the minimization of wasted space. Heuristics for the knapsack loading problem have been presented in Gehring, Menscher and Meyer [11] and Pisinger [19].

- *Container Loading.* In this version, all the items have to be packed into a single bin, which does however have an infinite heigth. The problem is thus to find a feasible solution such that the heigth to which the bin is filled is minimized. The first heuristic for the container loading problem was presented by George and Robinson [12], but several variants have been presented since then. Bischoff and Marriott [2] compare 14 different heuristics based on the George-Robinson framework. Heuristics with asymptotic worst-case performance guarantee have been presented by Li and Cheng [15].

- *Bin Packing.* The 3D-BPP calls for a solution where all the items are packed into bins which have finite sizes, and the objective is to find a solution using the smallest possible number of bins. An approximation algorithm for the three-dimensional bin-packing problem was presented by Scheithauer [20]. Chen, Lee and Shen [3] consider a generalization of the problem, where the bins may have different sizes. An integer programming formulation is developed and a small instance with $n = 6$ items is solved to optimality using an MIP solver.

To our knowledge no algorithms for the exact solution of 3D-BPP have been published, thus we start this presentation by considering a number of lower bounds. In Section 2 we determine the worst-case behavior of the so-called continuous lower bound for 3D-BPP. It is proved, through a constructive algorithm, that the asymptotic worst-case performance ratio of the continuous bound is $\frac{1}{8}$. New bounds are introduced and analyzed in Section 3. In Section 4 we present an exact algorithm for selecting a subset of items which can be packed into a single bin by maximizing the total volume packed. These results are used in Section 5 to obtain two approximation algorithms and an exact branch-and-bound algorithm. An extensive computational testing is presented in Section 6, showing that the exact algorithm is able to solve instances with up to 90 items to optimality within reasonable time.

In the following we will denote by Z the optimal solution value of 3D-BPP. The volume of item j will be denoted by $v_j = w_j h_j d_j$, the total volume of the items in J by $V = \sum_{j=1}^{n} v_j$, and the bin volume by $B = WHD$.

2 The continuous lower bound

An obvious lower bound for 3D-BPP comes from a relaxation in which each item j is cut into $w_j h_j d_j$ unit-length cubes, thus producing a *continuous lower bound*

$$L_0 = \left\lceil \frac{\sum_{j=1}^{n} v_j}{B} \right\rceil \tag{1}$$

L_0 can be computed in $O(n)$ time. In this section we examine its worst-case behavior.

Let $Z(I)$ be the value of the optimal solution to an instance I of a minimization problem, $L(I)$ the value provided by a generic lower bound L and let $\rho(I) = L(I)/Z(I)$. The *absolute worst-case performance ratio* of L is then defined as the smallest real number ρ such that $\rho(I) \geq \rho$ for any instance I of the problem. The *asymptotic worst-case performance ratio* of L is instead the smallest real number ρ^∞ such that there is a (large) positive integer value N for which $\rho(I) \geq \rho^\infty$ for all instances I satisfying $Z(I) \geq N$.

It is known that, for 1D-BPP, the absolute worst-case performance ratio of the continuous lower bound ($\lceil \sum_{j=1}^{n} w_j / W \rceil$) is $\frac{1}{2}$ (see, e.g., Martello and Toth [16]). Recently Martello and Vigo [18] proved that, for 2D-BPP, the continuous lower bound ($\lceil \sum_{j=1}^{n} w_j h_j / (WH) \rceil$) has absolute worst-case performance ratio equal to $\frac{1}{4}$.

Theorem 1 *The asymptotic worst-case performance ratio of lower bound L_0 is $\frac{1}{8}$.*

Proof We prove the thesis by introducing a heuristic algorithm which, given an instance I of 3D-BPP with a sufficiently large solution value $Z(I)$, produces a feasible solution requiring a number of bins arbitrarily close to $8L_0(I)$.

The heuristic applies, at each iteration, an algorithm (called H2D hereafter) proposed by Martello and Vigo [18] for the two-dimensional bin packing problem. Given an instance $\widetilde{I}$ of 2D-BPP, algorithm H2D determines a feasible solution requiring, say, $U(\widetilde{I})$ bins such that the first $U(\widetilde{I}) - 2$ of them are filled, on average, to at least one quarter of their area, i.e.,

$$\sum_{i=1}^{U(\widetilde{I})-2} A_i \geq (U(\widetilde{I}) - 2)\frac{HW}{4}$$

where A_i denotes the total area occupied by the rectangles packed by H2D into bin i. The heuristic algorithm for 3D-BPP works as follows.

1. Partition the items, according to their depth, into a number $q = \max\{1, \lceil \log D \rceil\}$ of subsets $J_0, \dots, J_{q-1}$, using the following rule:

   $$\text{item } j \text{ is assigned to set } J_i \quad \text{iff} \quad \frac{D}{2^{i+1}} \leq d_j < \frac{D}{2^i}$$

   but assign to set J_0 also the items with $d_j = D$. In other words, J_0 contains all the items with a depth of at least $D/2$, J_1 contains all the items with a depth less than $D/2$ and at least $D/4$, and so on.

2. For each nonempty subset J_i do the following:
   - let $\widetilde{I}_i$ be the instance of 2D-BPP defined by $|J_i|$ rectangles having sizes w_j and h_j for each $j \in J_i$, and bin sizes W and H;
   - apply algorithm H2D to $\widetilde{I}_i$, thus obtaining a solution of value $U(\widetilde{I}_i) = u_i + 2$ with the first u_i bins filled, on average, to at least one quarter of their area;
   - derive the corresponding three-dimensional solution using $u_i + 2$ *bin slices* with width W, height H and depth $D/2^i$. Observe that, by definition of J_i, the first u_i slices are filled, on average, to at least one eighth of their volume, i.e.,

     $$\sum_{k=1}^{u_i} V_k \geq \frac{D}{2^{i+1}} u_i \frac{HW}{4} = \frac{1}{8} u_i \frac{B}{2^i}$$

     where V_k denotes the total volume occupied by the items packed into bin slice k. Call *fractional bin slices* the last two (possibly less filled) slices.

3. Observe that all the bin slices have width W, height H and a depth which is a "power-of-2" fraction of D. Hence consider all the bin slices, apart from the fractional ones, according to decreasing depth and combine them into full bins of depth D by simply filling up the bins with slices from one end. This will give a number u of bins which have a total filling of no less than $u\frac{B}{8}$, plus at most one possibly less filled bin, $u + 1$.

4. Consider now the fractional bin slices and combine them, as done in Step 3, into full bins of depth D. Observe that at most four new bins are needed, since there are two such slices per subset and their total depth is bounded by

$$\sum_{i=0}^{q-1} 2\frac{D}{2^i} \leq 4D$$

We have thus obtained a feasible solution which uses $u+5$ bins. The total volume of all the items is $V = \sum_{j=1}^{n} v_j \geq u\frac{B}{8}$. Since $Z(I) \leq u+5$ we obtain

$$L_0(I) = \left\lceil \frac{V}{B} \right\rceil \geq \frac{u}{8} \geq \frac{Z(I)}{8} - \frac{5}{8} \tag{2}$$

To show that the bound is tight it is sufficient to consider an instance in which n is a multiple of eight, $W = H = D = a$ (where a is a very large even value) and, for each $j \in J$, $w_j = h_j = d_j = \frac{a}{2}+1$. The optimal solution value is clearly $Z = n$. If we denote with $\frac{a^3}{8}+\Delta$ the volume of an item, we have $L_0 = \lceil \frac{n}{8} + \frac{n\Delta}{a^3} \rceil = \frac{n}{8}+1$ (by assuming $a \geq \sqrt[3]{n\Delta}$), so the ratio L_0/Z is arbitrarily close to $\frac{1}{8}$ for sufficiently large n. $\square$

As previously mentioned, a variant of the problem may admit rotation of the items.

Corollary 1 *The asymptotic worst-case performance ratio of lower bound L_0 is $\frac{1}{8}$ even if rotation of the items (by any angle) is allowed.*

Proof Given any instance I, let $Z^r(I)$ denote the solution value if the items may be rotated. Since $Z^r(I) \leq Z(I)$, we immediately have from (2) that $L_0(I) \geq \frac{Z^r(I)}{8} - \frac{5}{8}$. On the other hand, the tightness example above holds for this variant too. $\square$

3 New lower bounds

The continuous lower bound of the previous section is likely to produce tight values when the item sizes are small with respect to the bin size. The lower bounds presented in this section are better suited to the cases in which there are relatively large items.

Our first bound is obtained by reduction to the one-dimensional case. Let

$$J^{WH} = \left\{ j \in J : w_j > \frac{W}{2} \text{ and } h_j > \frac{H}{2} \right\} \tag{3}$$

and define

$$\begin{aligned} L_1^{WH} = \left| \left\{ j \in J^{WH} : d_j > \frac{D}{2} \right\} \right| + \max_{1 \leq p \leq \frac{D}{2}} & \left\{ \left\lceil \frac{\sum_{j \in J_s(p)} d_j - (|J_\ell(p)|D - \sum_{j \in J_\ell(p)} d_j)}{D} \right\rceil, \right. \\ & \left. \left\lceil \frac{|J_s(p)| - \sum_{j \in J_\ell(p)} \lfloor \frac{D-d_j}{p} \rfloor}{\lfloor \frac{D}{p} \rfloor} \right\rceil \right\} \end{aligned} \tag{4}$$

where

$$\begin{aligned} J_\ell(p) &= \{ j \in J^{WH} : D - p \geq d_j > \tfrac{D}{2} \} && (5) \\ J_s(p) &= \{ j \in J^{WH} : \tfrac{D}{2} \geq d_j \geq p \} && (6) \end{aligned}$$

Theorem 2 *L_1^{WH} is a valid lower bound for 3D-BPP.*
Proof By definition, items of J^{WH} can possibly be packed into the same bin only by placing them one behind the other. Hence, the relaxed instance of 3D-BPP consisting of only such items is equivalent to an instance of 1D-BPP defined by the values d_j ($j \in J^{WH}$) with bin size D. A valid lower bound for 3D-BPP is thus given by any valid lower bound for 1D-BPP. Equation (4) has the form $L_1^{WH} = \alpha + \max_{1 \le p \le \frac{D}{2}} \{\beta(p), \gamma(p)\}$ combining two such bounds: $\alpha + \max_{1 \le p \le \frac{D}{2}} \{\beta(p)\}$ has been introduced by Martello and Toth [17], and $\alpha + \max_{1 \le p \le \frac{D}{2}} \{\gamma(p)\}$ by Dell'Amico and Martello [7]. $\square$

Corollary 2 *A valid lower bound for 3D-BPP is*

$$L_1 = \max\{L_1^{WH}, L_1^{WD}, L_1^{HD}\} \tag{7}$$

where L_1^{WH} is defined by (3)-(6), while L_1^{WD} (resp. L_1^{HD}) are obtained from (3)-(6) by interchanging h_j (resp. w_j) with d_j and H (resp. W) with D.
Proof Immediate, since $J^{WD} = \{j \in J : w_j > \frac{W}{2}$ and $d_j > \frac{D}{2}\}$ and $J^{HD} = \{j \in J : h_j > \frac{H}{2}$ and $d_j > \frac{D}{2}\}$, i.e., L_1^{WD} and L_1^{HD} are equivalent to L_1^{WH} over a rotated instance. $\square$

A straightforward computation of L_1 would require pseudo-polynomial time. However,

Proposition 1 *L_1 can be computed in $O(n^2)$ time.*
Proof It has been proved in Martello and Toth [17] and Dell'Amico and Martello [7] that only the p values corresponding to actual item sizes need be considered. $\square$

Proposition 2 *Lower bounds L_0 and L_1 do not dominate each other.*
Proof For the instance introduced in the tightness proof of Theorem 1 we have $\{j \in J^{WH} : d_j > \frac{D}{2}\} = J$ so from (4) we obtain $L_1 = n$ $(= Z)$, while $L_0 = \frac{n}{8} + 1$. Now consider a similar instance with n multiple of eight, W, H and D even and, for each $j \in J$, $w_j = \frac{W}{2}$, $h_j = \frac{H}{2}$ and $d_j = \frac{D}{2}$. In this case $Z = \frac{n}{8} = L_0$ whereas $L_1 = 0$, since $J^{WH} = J^{WD} = J^{HD} = \emptyset$. The latter instance also shows that the worst-case performance of L_1 can be arbitrarily bad. $\square$

A better bound, which explicitly takes into account the three item dimensions, is provided in the following.

Given any pair of integers (p, q), with $1 \le p \le \frac{W}{2}$ and $1 \le q \le \frac{H}{2}$, define

$$K_v(p,q) = \{j \in J : w_j > W - p \text{ and } h_j > H - q\} \tag{8}$$

$$K_\ell(p,q) = \{j \in J \setminus K_v(p,q) : w_j > \tfrac{W}{2} \text{ and } h_j > \tfrac{H}{2}\} \tag{9}$$

$$K_s(p,q) = \{j \in J \setminus (K_v(p,q) \cup K_\ell(p,q)) : w_j \ge p \text{ and } h_j \ge q\} \tag{10}$$

and let

$$L_2^{WH}(p,q) = L_1^{WH} + \max\left\{0, \left\lceil \frac{\sum_{j \in K_\ell(p,q) \cup K_s(p,q)} v_j - (D \cdot L_1^{WH} - \sum_{j \in K_v(p,q)} d_j)WH}{B} \right\rceil \right\} \tag{11}$$

and

$$L_2^{WH} = \max_{1 \le p \le \frac{W}{2}; 1 \le q \le \frac{H}{2}} \left\{ L_2^{WH}(p,q) \right\} \tag{12}$$

Theorem 3 *L_2^{WH} is a valid lower bound for 3D-BPP.*

Proof We will show that, for any pair (p,q), $L_2^{WH}(p,q)$ is a valid lower bound for the relaxed instance of 3D-BPP consisting of only the items in $K_v(p,q) \cup K_\ell(p,q) \cup K_s(p,q)$. For any pair (p,q), $K_v(p,q) \cup K_\ell(p,q)$ coincides with set J^{WH} (see (3)), so L_1^{WH} is a valid lower bound on the number of bins needed for such items. The second term in (11) increases this value through a lower bound on the number of additional bins needed for the items in $K_s(p,q)$. To this end observe that an item of $K_\ell(p,q) \cup K_s(p,q)$ and one of $K_v(p,q)$ could be packed into the same bin only by placing them one behind the other. In other words, with respect to our relaxed instance, any item of $K_v(p,q)$ of sizes $w_j \times h_j \times d_j$ can be seen as a larger *equivalent item* of sizes $W \times H \times d_j$. Hence, the total volume of the L_1^{WH} bins which can be used for the items of $K_s(p,q)$ is given by BL_1^{WH} minus the volume of the equivalent items of $K_v(p,q)$, minus the volume of the items of $K_\ell(p,q)$. It follows that at least

$$\max\left\{0, \left\lceil \frac{\sum_{j \in K_s(p,q)} v_j - (B \cdot L_1^{WH} - WH \sum_{j \in K_v(p,q)} d_j - \sum_{j \in K_\ell(p,q)} v_j)}{B} \right\rceil \right\} \tag{13}$$

additional bins are needed for the items of $K_s(p,q)$, hence the thesis. □

Corollary 3 *A valid lower bound for 3D-BPP is*

$$L_2 = \max\{L_2^{WH}, L_2^{WD}, L_2^{HD}\} \tag{14}$$

where L_2^{WD} (resp. L_2^{HD}) are obtained from (8)-(11) and (12) by interchanging h_j (resp. w_j) with d_j and H (resp. W) with D.

Proof Immediate, since L_2^{WD} and L_2^{HD} are equivalent to L_2^{WH} over a rotated instance. □

Proposition 3 *L_2 can be computed in $O(n^2)$ time.*

Proof We show how to compute L_2^{WH} in $O(n^2)$ time. First observe that L_1^{WH} is independent of p and q, hence it can be computed once (in $O(n^2)$ time). We now prove that in the computation of (12) it is sufficient to consider the values of p and q which correspond to distinct values of w_j and h_j, respectively. Indeed, given any p value, let q_1 and q_2 (with $q_1 < q_2$) be two distinct q values such that $K_s(p,q_1) = K_s(p,q_2)$, and note that the increase of q from q_1 to q_2 may cause one or more items to move from $K_\ell(p,q)$ to $K_v(p,q)$, i.e., $K_\ell(p,q_2) = K_\ell(p,q_1) \setminus K$ and $K_v(p,q_2) = K_v(p,q_1) \cup K$. Hence, $L_2^{WH}(p,q_1) \le L_2^{WH}(p,q_2)$ since for each item $i \in K$ the value of the numerator in (11) is increased by $d_i(WH - w_i h_i) \ge 0$. Therefore, only distinct values $q = h_j \le \frac{H}{2}$ need be considered since they induce different sets $K_s(p,q)$, while all the intermediate q values produce dominated lower bounds. Analogously, we obtain that for any q value only the values $p = w_j \leq \frac{W}{2}$ need be considered.

We conclude the proof by showing that given a p value, the computation of the bounds $L_2(p,q)$ for all q may be parametrically performed in $O(n)$ time. Indeed, let us assume that items are renumbered according to nondecreasing h_j values. The determination of the initial sets $K_v(p,h_1), K_\ell(p,h_1)$ and $K_s(p,h_1)$, as well as the computation of $L_2^{WH}(p,h_1)$ may be clearly performed in $O(n)$ time. As to the remaining q values, the computation of the corresponding bounds simply requires the updating of $\sum_{j \in K_\ell(p,q) \cup K_s(p,q)} v_j$ and $\sum_{j \in K_v(p,q)} d_j$. Indeed $K_v(p,q) \cup K_\ell(p,q) = J^{WH}$ is invariant while, for each new q value, some items may move from $K_\ell(p,q)$ to $K_v(p,q)$, and some new items may enter $K_s(p,q)$. Hence, for each p value the overall computation requires $O(n)$ time. $\Box$

Although the worst-case time complexity of L_2 is equal to that of L_1, it should be noted that the computational effort required to compute L_2 is considerably greater than that required by L_1. However,

Proposition 4 *L_2 dominates both L_0 and L_1.*

Proof For $p = q = 1$ we have $K_v(p,q) = \{j \in J : w_j = W \text{ and } h_j = H\}$ and $K_v(p,q) \cup K_\ell(p,q) \cup K_s(p,q) = J$. Hence, from (11)

$$L_2^{WH}(1,1) = L_1^{WH} + \max\left\{0, \left\lceil \frac{\sum_{j \in J} v_j - BL_1^{WH}}{B} \right\rceil\right\} = \max\left\{L_1^{WH}, \left\lceil \frac{\sum_{j \in J} v_j}{B} \right\rceil\right\} \qquad (15)$$

from which $L_2^{WH} \geq \max\left\{L_1^{WH}, L_0\right\}$. Analogously we have $L_2^{WD} \geq \max\left\{L_1^{WD}, L_0\right\}$ and $L_2^{HD} \geq \max\left\{L_1^{HD}, L_0\right\}$. $\Box$

4 Filling a single bin

In Section 5 we describe an enumerative algorithm for the exact solution of 3D-BPP. This algorithm repeatedly solves associated subproblems in which all the items of a given subset $\overline{J}$ have to be packed, if possible, into a single bin. Also this problem is NP-hard in the strong sense, since solving a special case in which all the items have the same depth $d_j = D$ and the same height $h_j = 1$ answers the question whether an instance of the one-dimensional bin packing problem admits a solution requiring no more than H bins. In the present section we describe a branch-and-bound algorithm for the exact solution of an optimization version of the problem (maximize the total volume of the packed items), which is also used within one of the heuristic algorithms presented in Section 5.3.

The problem we consider is that of selecting a subset $\overline{J}' \subseteq \overline{J}$ of items and assigning coordinates (x_j, y_j, z_j) to each item $j \in \overline{J}'$ such that no item goes outside the bin, no two items overlap, and the total volume of the items in $\overline{J}'$ is a maximum. For the two-dimensional case, a knapsack loading version of the problem has been considered in Hadjiconstantinou and Christofides [13]. In the following we present a non trivial generalization of the two-dimensional approach to the three-dimensional case, and give an effective algorithm for this problem. We assume that the origin of the coordinate system is in the left-bottom-back corner of the bin.

At each node of the branch-decision tree, described in more detail in Section 4.2, a current partial solution, which packs the items of a certain subset $I \subset \overline{J}$, is increased by selecting in turn each item $j \in \overline{J} \setminus I$, and generating descendant nodes by placing j into all the admissible points. By placing an item into a point p we mean that its left-bottom-back corner is positioned at p.

Let (x_p, y_p, z_p) be the coordinates of point p: obviously item j cannot be placed at p if $x_p + w_j > W$ or $y_p + h_j > H$ or $z_p + d_j > D$. The set of admissible points to be considered may be further drastically reduced through the following properties.

Property 1 *Any packing of a bin can be replaced by an equivalent packing where no item may be moved leftwards, downwards or backwards.*

Proof Obvious. A similar property was observed by Christofides and Witlock [4] for the two-dimesional case. □

Property 2 *An ordering of the items in an optimal solution exists such that, if $i < j$,*

$$x_i + w_i \le x_j \quad \text{or} \quad y_i + h_i \le y_j \quad \text{or} \quad z_i + d_i \le z_j \tag{16}$$

Proof Given any optimal solution, define an associated digraph with a vertex for each item and an arc from vertex i to vertex j if and only if (16) holds. It is clear that the resulting digraph is acyclic, since otherwise we would have a "cycle" of items in which two items are both one to the right of the other, or one above the other, or one behind the other. It is well-known that the vertices of an acyclic digraph can be re-numbered so that $i < j$ if arc (i, j) exists. □

It follows that the enumeration scheme for filling a single bin can be limited to only generate solutions which: (i) satisfy Property 1, and (ii) are such that the sequence in which the items are assigned (starting from the root node) constitute an item numbering satisfying Property 2.

An important consequence of (ii) is that at any decision node, where the items of I are already packed and item $j \in \overline{J} \setminus I$ is selected for branching, j may only be placed at points p such that no item of I has some part right of p, or above p, or in front of p. In other words, the items of I define an "envelope" separating the two regions where the items of $\overline{J} \setminus I$ may

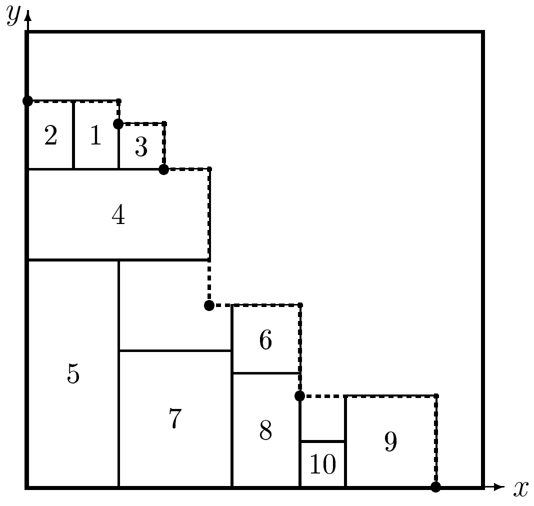

Figure 1: Two-dimensional single bin filling. The envelope associated with the placed items is marked by a dashed line, while black points indicate corner points in $\widehat{C}(I)$.

(resp. may not) be placed. More formally, a new item may only be placed in the set

$$S(I) = \{(x, y, z) : \forall i \in I, x \geq x_i + w_i \ \text{ or } \ y \geq y_i + h_i \ \text{ or } \ z \geq z_i + d_i\} \tag{17}$$

Figure 1 shows, for the two-dimensional case (in which the feasible region is $\widehat{S}(I) = \{(x, y) : \forall i \in I, x \geq x_i + w_i \ \text{ or } \ y \geq y_i + h_i\}$), a set of already placed items and the corresponding envelope. Observe that, due to Property 1, the next item j may only be placed at points where the slope of the envelope changes from vertical to horizontal (black points in Figure 1): such points are called in the following *corner points* of the envelope. In the next subsection we show how the set of feasible corner points can be efficiently determined.

4.1 Finding possible positions for placing an item

We solve this problem by repeatedly solving two-dimensional problems obtained by considering the x-y faces of a subset of items. We thus start with the description of the two-dimensional algorithm.

Given an item set I, it is quite easy to find, in two dimensions, the set $\widehat{C}(I)$ of corner points of the envelope associated with the feasible region $\widehat{S}(I)$ defined by the x-y faces of the items in I. Following Property 2, let us order the items according to their *end-points* $(x_j + w_j, y_j + h_j)$, so that the values of $y_j + h_j$ are nonincreasing, breaking ties by the largest value of $x_j + w_j$ (see Figure 1). The following algorithm for determining the corner points set, consists of three phases: First, *extreme items* are found, i.e., items whose end-point coincides with a point where the slope of the envelope changes from horizontal to vertical. In the second phase, corner points are defined at intersections between the lines leading out from end-points of extreme items. Finally, infeasible corners, where no further item of $\overline{J} \setminus I$ can fit, are removed. Thus we get the following algorithm:

```
algorithm 2D-CORNERS:
begin
  if $I = \emptyset$ then $\widehat{C}(I) := \{(0,0)\}$ and return;
  comment: Phase 1: identify the extreme items $e_1, \ldots, e_m$;
  $\overline{x} := m := 0$;
  for $j := 1$ to $|I|$ do
    if $(x_j + w_j > \overline{x})$ then $m := m+1$; $e_m := j$; $\overline{x} := x_j + w_j$;
  comment: Phase 2: determine the corner points;
  $\widehat{C}(I) := \{(0, y_{e_1} + h_{e_1})\}$;
  for $j := 2$ to $m$ do $\widehat{C}(I) := \widehat{C}(I) \cup \{(x_{e_{j-1}} + w_{e_{j-1}}, y_{e_j} + h_{e_j})\}$;
  $\widehat{C}(I) := \widehat{C}(I) \cup \{(x_{e_m} + w_{e_m}, 0)\}$;
  comment: Phase 3: remove infeasible corner points;
  for each $(x'_j, y'_j) \in \widehat{C}(I)$ do
    if $x'_j + \min_{i \in \overline{J} \setminus I}\{w_i\} > W$ or $y'_j + \min_{i \in \overline{J} \setminus I}\{h_i\} > H$ then $\widehat{C}(I) := \widehat{C}(I) \setminus \{(x'_j, y'_j)\}$
end.
```

Consider the example in Figure 1: the extreme items are 1, 3, 4, 6 and 9, and the resulting corner points are indicated by black dots; Phase 3 could remove some of the first and/or last corner points.

The time complexity of 2D-CORNERS is $O(|I|)$, plus $O(|I| \ \log |I|)$ for the initial item sorting, i.e., $O(n)$ plus $O(n \log n)$.

Assume that the resulting corner points are $\widehat{C}(I) = \{(x'_1, y'_1), \ldots, (x'_\ell, y'_\ell)\} \neq \emptyset$. Then the area occupied by the the envelope is

$$A(I) = x'_1 H + \sum_{i=2}^{\ell} (x'_i - x'_{i-1}) y'_{i-1} + (W - x'_\ell) y'_\ell \tag{18}$$

where the first (resp. last) term is nonzero whenever the first (resp. last) corner point found in Phase 2 has been removed in Phase 3. In the special case where $\widehat{C}(I) = \emptyset$, we obviously set $A(I) = WH$.

Algorithm 2D-CORNERS can be used to determine the set $C(I)$ of corner points in three dimensions, where I is the set of three-dimensional items currently packed into the bin. One may apply the algorithm for $z = 0$ and for each distinct z coordinate where an item of I ends, by increasing values. For each such coordinate z', 2D-CORNERS can be applied to the subset of those items $i \in I$ which end after z', i.e., such that

$$z_i + d_i > z' \tag{19}$$

adding the resulting corner points to $C(I)$. In this way, one may however obtain some false corner points, since they are corner points in the two-dimensional case, but not in the three-dimensional case (see, e.g., Figure 2). Such points are easily removed by dominance, where we say that a corner point (x'_a, y'_a, z'_a) dominates another corner point (x'_b, y'_b, z'_b) that is "hidden" behind it. Formally this may be written as

$$x'_a = x'_b \text{ and } y'_a = y'_b \text{ and } z'_a < z'_b \tag{20}$$

where we have equalities in the first two expressions since (19) ensures that all the items in front of z'_k are chosen, and thus no corner point will be generated inside the three-dimensional envelope. This is done in the following algorithm, where the generation of corner points ends as soon as a z coordinate is found such that no further item could be placed after it.

```
algorithm 3D-CORNERS:
begin
  if $I = \emptyset$ then $C(I) := \{(0,0,0)\}$; return;
  $T := \{0\} \cup \{z_i + d_i : i \in I\}$;
  sort $T$ by increasing values, and let $T = \{z'_1, \dots, z'_r\}$;
  $C(I) := \widehat{C}(I_0) := \emptyset$; $k := 1$;
  while $k \le r$ and $z'_k + \min_{i \in \overline{J} \setminus I}\{d_i\} \le D$ do
    begin
      $I_k := \{i \in I : z_i + d_i > z'_k\}$;
      apply 2D-CORNERS to $I_k$ yielding $\widehat{C}(I_k)$;
      for each $(x'_j, y'_j) \in \widehat{C}(I_k)$ do   (comment: add true corner points to $C(I)$)
        if $(x'_j, y'_j) \notin \widehat{C}(I_{k-1})$ then $C(I) := C(I) \cup \{(x'_j, y'_j, z'_k)\}$;
      $k := k + 1$
    end
end.
```

The time complexity of 3D-CORNERS is $O(n^2)$. Indeed, there are at most $|I|+1$ distinct z coordinates in T, and each set $\widehat{C}(I_k)$ is derived by 2D-CORNERS in $O(|I_k|)$ time, since the sorting of the items can be done once and for all at the beginning of the algorithm. For each z'_k value, the check of corner points prior to their addition to $C(I)$ requires $O(|\widehat{C}(I_{k-1})|)$ time in total, since both $\widehat{C}(I_{k-1})$ and $\widehat{C}(I_k)$ are ordered by increasing x'_j and decreasing y'_j. The overall complexity follows from the observation that both $|I|$ and $|\widehat{C}(I_k)|$ are $O(n)$.

Assuming that k^* is the index of the last I_k generated by 3D-CORNERS, the volume $V(I)$ occupied by the envelope associated with I is then

$$V(I) = \sum_{k=2}^{k^*} (z'_k - z'_{k-1}) A(I_{k-1}) + (D - z'_{k^*}) A(I_{k^*}) \tag{21}$$

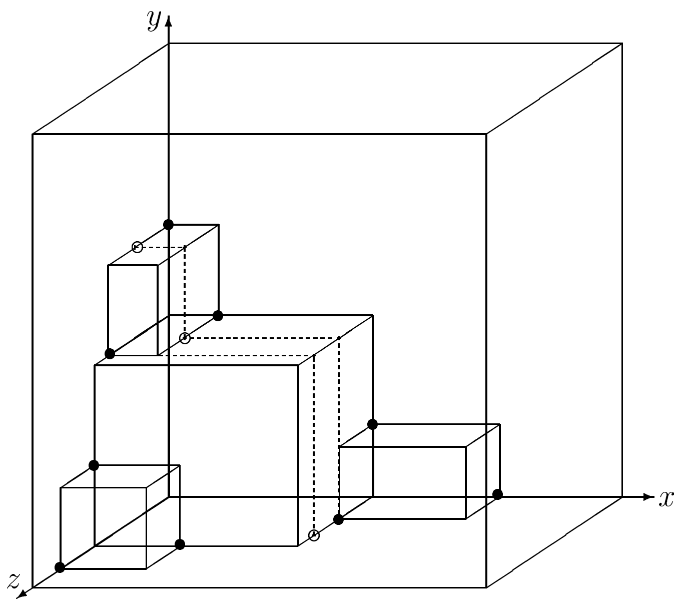

Figure 2: Three-dimensional single bin filling. Corner points $C(I)$ are found by applying algorithm 2D-CORNERS five times, one for each value of z'_k. True corner points are black dots, false corner points are empty circles.

where the last term is nonzero whenever $k^* < r$.

4.2 ONEBIN: an exact algorithm for filling a single bin

We can now easily derive, in a recursive way, a branch-and-bound algorithm, called ONEBIN, for finding the best filling of a single bin using items from a given set $\overline{J}$. Initially no item is placed, so $C(\emptyset) = \{(0,0,0)\}$. At each iteration, given the set $I \subset \overline{J}$ of currently packed items, set $C(I)$ is determined through 3D-CORNERS, together with the corresponding volume $V(I)$. If F is the total volume achieved by the current best filling, we may backtrack whenever

$$\sum_{i \in I} v_i + \Big(B - V(I)\Big) \leq F \tag{22}$$

since even if the remaining volume was completely filled, we would not improve on F.

If no more items fit into the bin (i.e., if $C(I) = \emptyset$) we possibly update F, and backtrack. Otherwise, for each position $(x', y', z') \in C(I)$ and for each item $j \in \overline{J} \setminus I$, we assign the item to this position (provided that its end-points do not exceed the bin limits), and call the procedure recursively.

The best performance of the branch-and-bound algorithm was obtained when the items were a-priori ordered according to nonincreasing volumes. This is not surprising, as such ordering is consistent with the good performance of the well-known best-fit decreasing algorithm for 1D-BPP.

Although the branch-and-bound approach based on 3D-CORNERS considerably limits the enumeration compared to a naive technique trying all placings of items, practical experiments have shown that the algorithm is very time consuming. However, in the branch-and-bound algorithm decribed in the next section the single bin filling has often to be solved for a small subset $\overline{J}$ of items. Thus a specific procedure was derived for solving the subproblem when $|\overline{J}| \leq 4$, through direct evaluation of all possible placings.

For the case of two items, there are only three arrangements in which the items may be placed with respect to each other: one beside the other, one above the other, one behind the other. Thus these three arrangements are simply tested.

With three items it is obvious that a guillotine packing always exists. Thus the problem may be reduced to that of placing two items at one side of the cut, and the remaining item at the other side. Hence, there are three ways in which the items can be partitioned, and for each partition the cut can be made in three orthogonal orientations: the two items at one side of the cut can then be handled as the packing problem with two items.

With four items the case is more involved, since non-guillotine packing is also possible (when the four items lay on the same plane). If the items are guillotine packed, depending on how the first cut separates the items, the possible placings can be handled as the packing problem with two or three items. As to non-guillotine packings, for reasons of symmetry, we may fix one of the items in a corner, and consider the 3! placings of the remaining items in the three remaining corners of the same plane, checking that no two items overlap.

5 Exact and approximation algorithms for 3D-BPP

The exact algorithm for 3D-BPP is based on the two-level decomposition principle presented by Martello and Vigo [18] for 2D-BPP. A *main branching tree* assigns items to bins without specifying their actual position, while a specialized version of the algorithm of Section 4 is used, at certain decision nodes, to test whether a subset of items can be placed inside a single bin and to determine the placing when the answer is affirmative. In order to construct a good starting solution, two heuristic algorithms have been defined, one based on a first-fit-decreasing approach and one based on an m-cut version of the ONEBIN algorithm of Section 4.2. In the next sections we describe the main components of our exact algorithm.

5.1 Main Branching Tree

The main branching tree assigns items to the different bins without specifying their actual position. The items are previously sorted according to nonincreasing volumes, and the exploration follows a depth-first strategy. Let Z be the incumbent solution value and $M =$

$\{1, \ldots, m\}$ the current set of bins used to allocate items in the ascendant decision nodes. A bin of M is called *open* if there is no evidence that no further item can be placed into it; otherwise, it is called *closed*.

At each decision node the next free item is assigned, in turn, to all the open bins; in addition, if $|M| < Z - 1$ the item is also assigned to a new bin (hence opening it).

When an item k is assigned to a bin i (*current bin*) which already contains, say, a subset $J_i \neq \emptyset$ of items, the actual feasibility of the assignment is checked as follows. First, lower bound L_2 is computed for the sub-instance defined by the items of $\overline{J} = J_i \cup \{k\}$: if $L_2 \geq 2$ the node is immediately killed. Otherwise, the heuristics of Section 5.3 are executed in sequence for the sub-instance: if a solution requiring a single bin is obtained, the assignment is accepted. If, instead, no such solution is found, the optimal solution for the sub-instance is determined through the algorithm of Section 5.2: the node is then accepted if a single bin solution is found, or killed otherwise.

When the current node is accepted, an attempt is made to close the current bin i. For each free item k' we compute lower bound L_2 for the sub-instance defined by the items of $\overline{J} \cup \{k'\}$. If the lower bound is greater than one for each k' then the bin is closed, since we know that no further item could be placed into it. Otherwise, the following dominance rule is tested. Let K be the subset of those free items for which the above lower bound computation has given value one: if one of the heuristics of Section 5.3 can find a single bin solution for the sub-instance defined by the items in $\overline{J} \cup K$, then we know that no better placing is possible for these items, 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 items not currently assigned to closed bins: if $Z \leq L_2 + c$, where c is the number of closed bins, we backtrack. Since closed bins are seldom completely filled, this improved bound generally increases as the branching propagates.

5.2 Single bin filling

When a decision node is neither killed by the lower bound, nor accepted by the heuristics, the feasibility of the current assignment of the items in $\overline{J}$ is tested through algorithm ONEBIN of Section 4.2. Since we are only interested in finding a solution where all the items in $\overline{J}$ are packed, we initialize the current best filling to $F = \sum_{\ell \in \overline{J}} v_\ell - 1$: if the algorithm returns an unchanged value of F then no filling exists with all the items inside the same bin.

5.3 Approximation algorithms

In order to obtain a good upper bound at the root node of the branching tree, and to limit the number of executions of ONEBIN, two different heuristics were defined. The two approaches computationally proved to have a “complementary” behavior: for many instances a poor performance of one of them corresponded to a good performance of the other one, although, on average, the second one gives better results.

The first heuristic is based on a layer building principle derived from shelf approaches used by several authors for 2D-BPP (see, e.g., Chung, Garey and Johnson [5] and Berkey and Wang [1]). In such approaches the rectangles are sorted by nonincreasing height and packed, from left to right, in rows forming shelves: the first shelf is the bottom of the two-dimensional bin; when a new shelf is needed, it is created along the horizontal line which coincides with the top of the tallest rectangle packed into the highest shelf. For the three-dimensional case we first construct *bin slices* having width W, height H and different depths. Each slice is obtained through a two-dimensional shelf algorithm applied to a subset containing the deepest items not yet packed: since the shelf algorithm discards items that do not fit into the current packing, the input subset consists of items with a total x-y area close to twice the available area. The slices are then combined into three-dimensional bins. Let $\overline{J}$ be the set of items to be packed. The algorithm works as follows.

```
**algorithm** H1:
**begin**
  $T := \overline{J}$; sort the items of $T$ by nonincreasing depth;
  **while** $T \neq \emptyset$ **do**
    **begin**
      let $T = \{j_1, \dots, j_{|T|}\}$; $k := \max\{r : \sum_{s=1}^{r} w_{j_s} h_{j_s} < 2WH\}$;
      **comment:** construct a single slice of depth $d_{j_1}$;
      $T' := \{j_1, \dots, j_k\}$; sort the items of $T'$ by nondecreasing height;
      **for each** item $j \in T'$ **do**
        **if** $j$ fits into an existing shelf **then** pack $j$ into the lowest shelf where it fits
        **else if** there is enough vertical space for $j$ **then** pack $j$ into a new shelf;
      remove the packed items from $T$
    **end**;
  combine the slices into bins of depth $D$ through a 1D-BPP algorithm
**end**.
```

For the final step we used the FORTRAN code MTP, provided in the book by Martello and Toth [16], with a limit of 10^6 backtrackings. By rotating the instance, algorithm H1 will construct bin slices with width W, depth D and different heights, or bin slices with heigth H, depth D and different widths. Thus at the root node H1 is run three times, corresponding to these rotations, and the best solution is taken.

The second heuristic repeatedly fills a single bin. Let $\overline{J}$ be the set of items to be packed.

```
algorithm H2:
begin
  $T := \overline{J}$; sort the boxes of $T$ by nondecreasing volume;
  while $T \neq \emptyset$ do
    apply ONEBIN to $T$ and remove the packed items from $T$
end.
```

Since the solution times of algorithm ONEBIN may be unacceptable when $|T|$ is large, the branching scheme of Section 4.2 has been changed to an m-cut enumeration as described in Ibaraki [14], where at each decision node only the first m branches are considered. Good values for m were experimentally found to be $m = 4$ when $|T| \leq 10$, $m = 3$ when $10 < |T| \leq 15$ and $m = 2$ for larger problems. Moreover, a limit of 5000 decision nodes was imposed on each filling of a bin, and the best solution found within this limit was returned. Because of the limit on the number of branches, also algorithm H2 produces different solutions if the instance is rotated, hence this algorithm too is run three times at the root node. At other decision nodes, where the heuristics are used to find a single bin filling (see Section 5.1), we first run H2 once and, if it fails, we run H1 once. Computational experiments showed indeed that the instance rotation very rarely helps in the single bin case. The experiments also proved that it is not convenient to use the heuristics, at each decision node, to produce complete feasible solutions starting from the partial solution associated with the node.

6 Computational Experiments

To our knowledge no test instances have been published for the three-dimensional bin packing problem. Thus in our computational experiments we have chosen to generate three-dimensional instances by generalizing some classes of randomly generated two-dimensional instances. Also a new class of "all-fill" instances was introduced. For short in the following "u.r." means "uniformly random" (or "uniformly randomly"). All test instances are available on web site `http://www.diku.dk/~pisinger/codes.html`.

The first classes of instances are generalizations of the instances considered by Martello and Vigo in [18]. The bin size is $W = H = D = 100$ and five types of items are considered:

- *Type 1*: w_j u.r. in $[1, \frac{1}{2}W]$, h_j u.r. in $[\frac{2}{3}H, H]$, d_j u.r. in $[\frac{2}{3}D, D]$.
- *Type 2*: w_j u.r. in $[\frac{2}{3}W, W]$, h_j u.r. in $[1, \frac{1}{2}H]$, d_j u.r. in $[\frac{2}{3}D, D]$.
- *Type 3*: w_j u.r. in $[\frac{2}{3}W, W]$, h_j u.r. in $[\frac{2}{3}H, H]$, d_j u.r. in $[1, \frac{1}{2}D]$.
- *Type 4*: w_j u.r. in $[\frac{1}{2}W, W]$, h_j u.r. in $[\frac{1}{2}H, H]$, d_j u.r. in $[\frac{1}{2}D, D]$.

- *Type 5*: w_j u.r. in $[1, \frac{1}{2}W]$, h_j u.r. in $[1, \frac{1}{2}H]$, d_j u.r. in $[1, \frac{1}{2}D]$.

We 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 is a generalization of the instances presented by Berkey and Wang [1]. The three classes may be described as:

Class 6: bin size $W = H = D = 10$; items with w_j, h_j, d_j u.r. in $[1, 10]$.

Class 7: bin size $W = H = D = 40$; items with w_j, h_j, d_j u.r. in $[1, 35]$.

Class 8: bin size $W = H = D = 100$; items with w_j, h_j, d_j u.r. in $[1, 100]$.

Finally, a new difficult class of *all-fill* problems has been introduced (*Class 9*). These instances have a known solution with three bins, since the items are generated by cutting the bins into smaller parts. For a problem with n items, bins 1 and 2 are cut into $\lfloor n/3 \rfloor$ items each, while bin 3 is cut into $n - 2\lfloor n/3 \rfloor$ items. The cutting is made using the following recursion, where J is the set of items which should be generated, and the initial values of w, h and d are W, H and D, respectively.

```
procedure CUT($J$, $w$, $h$, $d$)
begin
  if $|J| = 1$ then generate an item with size $(w, h, d)$;
  else
    if $|J| = 5$ then generate a non-guillotine cutting pattern as shown in Figure 3;
    else
      begin
        u.r. partition $J$ into $J_1$ and $J_2$;
        u.r. execute one of the following three steps:
        1. generate $w'$ u.r. in $[1, w-1]$; call CUT($J_1$, $w'$, $h$, $d$); call CUT($J_2$, $w - w'$, $h$, $d$);
        2. generate $h'$ u.r. in $[1, h-1]$; call CUT($J_1$, $w$, $h'$, $d$); call CUT($J_2$, $w$, $h - h'$, $d$);
        3. generate $d'$ u.r. in $[1, d-1]$; call CUT($J_1$, $w$, $h$, $d'$); call CUT($J_2$, $w$, $h$, $d - d'$)
      end
end.
```

Pathological situations may occur, in which the procedure returns items with null volume since it is not possible to arrange the items of J within (w, h, d), but in this case a new problem is simply generated from scratch. The exact code was implemented in C, and the experiments were run on a HP9000/C160 160 Mhz. The outcome of our experiments is given in Tables I to VII. A time limit of 1000 seconds was given to each instance, and 10

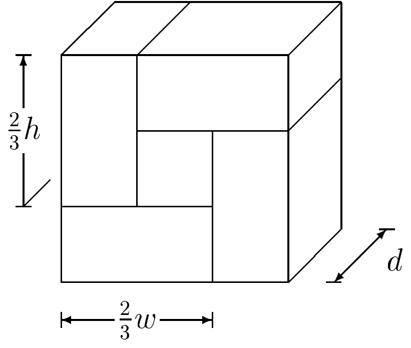

Figure 3: A non-guillottine cuttable pattern with five items

instances were generated for each class and size of a problem. Fine-tuning of the algorithm showed that the best performance was obtained if the principle of closing a bin, described in Section 5.1, was only tested at branching nodes not farther than 16 nodes from the root: this value appears to be surprisingly robust, at least for all the instances of our experiments, and independent of the size and the class.

Tables I and II give the number of instances solved to proved optimality and the average CPU time required for the solution, computed over all the solved instances. Nearly all the instances with $n \leq 30$ and 84% of those with $n \leq 50$ were solved to optimality within reasonable computing times. In total we solved more than 63% of the problems within the given time limit (i.e., 740 out of 1170). Almost all the instances of Classes 4 and 6, where large items are more frequent, were solved to optimality also for large values of n. On the other hand the instances of Classes 5, 7 and 9, where the average number of items per bin is much higher, turned out to be harder. Indeed, L_1 and L_2 are clearly more effective if large items are present (and cannot improve L_0 if all the items are small). Additional tests with instances having similar characteristics confirmed this trend. The single bin filling algorithm is the most time consuming component: for instances of "average" difficulty, such as Class 9 with $n = 40$, it took, on average, about 99% of the total CPU time. For the same instances we had to determine a total of 33,686 single bin fillings: about 40% of them, involving no more than four items, were solved through the special procedure (see Section 4.2), while for the remaining 60%, involving five to twenty items, the full branch-and-bound algorithm was executed.

Tables III–V show the average deviation of the lower bounds values L_i, $i = 0, 1, 2$ computed at the root node of the branch-decision tree, with respect to the optimal solution value Z. The deviation is defined as $100 \cdot (Z - L_i)/Z$, $i = 0, 1, 2$, and is computed for the solved instances only. In Proposition 4 we proved that L_2 dominates both L_0 and L_1, but it is interesting to see that also in practice L_2 is considerably better than both of the other bounds. In fact, over all the solved instaces, the average deviation of L_2 is 9.6%, whereas those of L_0 and L_1 are 28.4% and 14.4%, respectively. Table VI shows the average deviation of the best upper bound value U found by algorithms H1 and H2 at the root node, with respect to the optimal solution value. In this case the deviation is defined as $100 \cdot (U - Z)/Z$, and is again computed for the solved instances only. It may be noted that the heuristic solutions are generally very good, but for the very difficult "all-fill" instances of Class 9. Over all solved instances, the average deviation of U is 5.3%, whereas the average deviation of U obtained by excluding the instances of Class 9 is 3.3%. Finally, Table VII shows the average deviation of the best upper bound value U with respect to the best lower bound value L_2, computed over all instances.

Table I: Number of instances solved to proved optimality

	Class									
n	1	2	3	4	5	6	7	8	9	Total
10	10	10	10	10	10	10	10	10	10	90
15	10	10	10	10	10	10	10	10	10	90
20	10	10	10	10	10	10	10	10	10	90
25	10	10	10	10	10	10	9	10	10	89
30	10	10	10	10	10	10	5	10	10	85
35	9	9	10	10	7	10	5	6	9	75
40	9	8	8	10	4	10	3	8	8	68
45	5	6	4	10	8	10	1	7	2	53
50	5	2	4	10	3	10	2	4	—	40
60	1	1	—	9	1	6	—	5	—	23
70	—	—	—	6	2	4	—	4	—	16
80	—	—	1	4	—	4	—	2	—	11
90	—	—	—	2	—	7	—	1	—	10
Total	79	76	77	111	75	111	55	87	69	740

Table II: Average solution times expressed in HP9000/C160 160 Mhz CPU seconds

	Class								
n	1	2	3	4	5	6	7	8	9
10	0.03	0.02	0.03	0.01	0.14	0.08	0.13	0.11	0.00
15	0.05	0.04	0.06	0.01	0.26	0.15	0.24	0.30	0.00
20	0.19	0.11	0.10	0.02	0.43	0.48	0.38	0.31	0.12
25	1.02	0.27	1.70	0.03	9.08	0.80	5.94	0.82	0.34
30	78.17	8.46	9.50	0.15	1.10	15.55	3.05	1.26	56.46
35	2.63	112.70	42.32	0.17	2.13	4.18	61.64	2.17	6.36
40	72.16	30.61	268.04	0.21	1.25	36.24	15.58	66.39	220.37
45	115.65	209.21	35.11	0.29	7.36	72.80	0.97	14.67	498.90
50	248.62	101.86	30.01	22.24	238.73	68.56	1.34	27.17	—
60	74.25	250.63	—	18.99	9.01	139.68	—	257.73	—
70	—	—	—	11.78	355.20	211.31	—	298.74	—
80	—	—	4.03	4.74	—	31.76	—	137.16	—
90	—	—	—	223.13	—	131.31	—	308.66	—

Table III: Average deviation of lower bound L_0 with respect to the optimal solution value

	Class								
n	1	2	3	4	5	6	7	8	9
10	24.2	37.1	38.9	45.5	40.0	31.0	40.0	28.6	0.0
15	31.9	31.1	33.3	42.5	37.9	31.8	30.0	27.0	0.0
20	30.0	34.8	30.0	46.3	30.8	27.8	34.2	32.7	0.0
25	27.0	30.0	33.8	46.1	30.4	23.7	28.2	30.9	0.0
30	30.2	30.0	29.1	44.8	29.6	20.6	31.8	28.3	0.0
35	25.6	26.7	27.2	46.4	33.3	18.2	26.9	28.9	0.0
40	23.7	27.9	29.3	47.3	31.0	19.3	38.1	26.2	0.0
45	28.8	28.9	24.5	47.8	28.6	17.7	50.0	25.8	0.0
50	26.6	32.1	26.4	46.3	25.8	14.3	35.3	25.6	0.0
60	21.4	26.7	—	47.8	30.0	14.5	—	21.6	0.0
70	—	—	—	48.0	25.9	12.1	—	22.0	0.0
80	—	—	27.8	48.1	—	14.9	—	19.2	0.0
90	—	—	—	48.6	—	14.6	—	18.8	0.0

Table IV: Average deviation of lower bound L_1 with respect to the optimal solution value

	Class								
n	1	2	3	4	5	6	7	8	9
10	6.1	11.4	22.2	1.5	16.0	27.6	32.0	17.9	3.3
15	17.0	17.8	22.9	1.1	24.1	29.5	20.0	13.5	6.7
20	13.3	13.6	18.3	0.8	15.4	24.1	15.8	14.3	6.7
25	12.2	15.7	12.7	1.9	19.6	22.0	20.5	16.4	16.7
30	14.0	16.2	16.3	3.5	16.7	29.4	27.3	18.3	3.3
35	9.8	10.5	15.5	2.4	14.6	26.0	23.1	15.6	16.7
40	13.4	12.8	16.3	2.1	13.8	18.2	23.8	16.9	10.0
45	13.6	13.2	18.4	2.2	19.0	18.8	12.5	14.5	10.0
50	12.5	7.1	15.1	2.4	19.4	22.4	11.8	18.6	20.0
60	7.1	13.3	—	1.5	20.0	18.4	—	11.8	36.7
70	—	—	—	2.0	11.1	20.7	—	12.0	10.0
80	—	—	0.0	1.1	—	13.4	—	7.7	16.7
90	—	—	—	0.9	—	16.2	—	18.8	26.7

Table V: Average deviation of lower bound L_2 with respect to the optimal solution value

	Class								
n	1	2	3	4	5	6	7	8	9
10	6.1	11.4	19.4	1.5	16.0	17.2	28.0	14.3	0.0
15	12.8	13.3	16.7	1.1	24.1	20.5	10.0	10.8	0.0
20	13.3	12.1	16.7	0.8	15.4	14.8	13.2	12.2	0.0
25	9.5	8.6	11.3	1.9	13.0	13.6	17.9	10.9	0.0
30	11.6	10.0	12.8	3.5	13.0	13.2	22.7	13.3	0.0
35	6.1	7.0	10.7	2.4	10.4	11.7	11.5	11.1	0.0
40	6.2	8.1	9.8	2.1	10.3	6.8	23.8	10.8	0.0
45	6.8	7.9	6.1	2.2	14.3	9.4	12.5	9.7	0.0
50	6.2	3.6	7.5	2.4	12.9	11.2	11.8	11.6	0.0
60	0.0	6.7	—	1.5	20.0	9.2	—	9.8	0.0
70	—	—	—	2.0	11.1	8.6	—	10.0	0.0
80	—	—	0.0	1.1	—	9.0	—	7.7	0.0
90	—	—	—	0.9	—	7.7	—	12.5	0.0

Table VI: Average deviation of upper bound U with respect to the optimal solution value

	Class								
n	1	2	3	4	5	6	7	8	9
10	0.0	0.0	0.0	0.0	0.0	0.0	5.0	0.0	0.0
15	0.0	0.0	0.0	0.0	3.3	1.7	10.0	6.7	0.0
20	2.0	0.0	4.0	0.0	5.0	3.3	5.0	0.0	6.7
25	1.7	4.7	3.1	0.0	5.8	4.0	3.3	5.8	30.0
30	2.9	3.9	1.2	0.0	4.2	3.7	0.0	4.8	40.0
35	7.9	3.4	3.0	0.0	0.0	9.8	5.4	4.0	40.0
40	7.5	3.7	6.0	0.0	1.2	9.9	1.1	11.1	53.3
45	4.3	2.5	4.1	0.4	2.5	11.1	0.0	8.9	56.7
50	5.5	1.4	3.2	0.8	2.9	11.5	0.0	4.9	66.7
60	0.7	0.7	—	0.6	0.0	6.5	—	7.3	60.0
70	—	—	—	0.0	0.8	4.9	—	8.0	83.3
80	—	—	1.1	0.0	—	4.3	—	3.8	93.3
90	—	—	—	0.2	—	7.8	—	1.2	103.3

Table VII: Average deviation of upper bound U with respect to lower bound L_2

	Class								
n	1	2	3	4	5	6	7	8	9
10	8.3	15.0	29.2	2.5	30.0	28.3	48.3	25.0	0.0
15	16.5	18.3	22.0	1.1	47.5	30.7	19.2	23.3	0.0
20	19.2	14.5	27.7	0.8	26.7	20.5	25.8	18.3	6.7
25	13.2	14.0	18.7	2.4	23.7	20.2	30.8	21.8	30.0
30	16.8	15.7	16.6	3.9	22.3	20.2	37.5	25.9	40.0
35	15.7	11.6	15.5	2.5	18.1	25.0	33.5	28.1	40.0
40	16.1	14.7	17.3	2.2	34.1	19.3	39.4	28.1	53.3
45	15.6	12.5	12.1	3.2	25.2	23.0	45.6	30.4	56.7
50	17.4	14.4	17.1	3.5	31.6	25.6	33.7	35.2	66.7
60	15.2	15.6	14.0	2.8	33.3	23.5	41.0	23.3	60.0
70	14.9	14.8	15.8	3.6	31.0	24.1	47.2	32.2	83.3
80	17.0	14.9	12.8	2.7	32.6	22.5	37.9	35.2	93.3
90	13.4	14.0	13.8	3.4	30.1	21.3	45.0	32.8	103.3

7 Conclusions

Our paper is the first work on exact algorithms for the three-dimensional bin-packing problem; thus several important aspects have been addressed. We have presented a number of lower bounds, and compared the theoretical as well as practical performance of them. The branch-and-bound algorithm for the filling of a single bin plays a central role in our overall algorithm for 3D-BPP, and it may be useful for other research projects in the field of cutting and packing. Finally we have demonstrated that the framework proposed by Martello and Vigo [18] for the exact solution of 2D-BPP may be adapted to the three-dimensional case with small modifications. The computational results illustrate the applicability of our results.

Acknowledgments

The first and third authors acknowledge Ministero dell'Università e della Ricerca Scientifica e Tecnologica (MURST) and Consiglio Nazionale delle Ricerche (CNR), Italy, for the support of this project. The second author would like to thank the EC Network DIMANET for supporting the research by European Research Fellowship No. ERBCHRXCT-94 0429.

References

[1] J.O. Berkey and P.Y. Wang. Two dimensional finite bin packing algorithms. *Journal of the Operational Research Society*, 38:423–429, 1987.

[2] E.E. Bischoff and M.D. Marriott. A comparative evaluation of heuristics for container loading. *European Journal of Operational Research*, 44:267–276, 1990.

[3] C.S. Chen, S.M. Lee, and Q.S. Shen. An analytical model for the container loading problem. *European Journal of Operational Research*, 80:68–76, 1995.

[4] N. Christofides and C. Whitlock. An algorithm for two-dimensional cutting problems. *Operations Research*, 25:30–44, 1977.

[5] F.K.R. Chung, M.R. Garey, and D.S. Johnson. On packing two-dimensional bins. *SIAM Journal of Algebraic and Discrete Methods*, 3(1):66–76, 1982.

[6] E.G. Coffman, Jr., M.R. Garey, and D.S. Johnson. Approximation algorithms for bin packing: A survey. In D.S. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems.* PWS Publishing Company, Boston, 1997.

[7] M. Dell'Amico and S.Martello. Optimal scheduling of tasks on identical parallel processors. *ORSA Journal on Computing*, 7(2):191–200, 1995.

[8] H. Dyckhoff. A typology of cutting and packing problems. *European Journal of Operational Research*, 44:145–159, 1990.

[9] H. Dyckhoff and U. Finke. *Cutting and Packing in Production and Distribution.* Physica Verlag, Heidelberg, 1992.

[10] H. Dyckhoff, G. Scheithauer, and J. Terno. Cutting and Packing (C&P). In M. Dell'Amico, F. Maffioli, and S. Martello, editors, *Annotated Bibliographies in Combinatorial Optimization.* John Wiley & Sons, Chichester, 1997.

[11] M. Gehring, K. Menscher, and M. Meyer. A computer-based heuristic for packing pooled shipment containers. *European Journal of Operational Research*, 44:277–288, 1990.

[12] J.A. George and D.F. Robinson. A heuristic for packing boxes into a container. *Computers and Operations Research*, 7:147–156, 1980.

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

[14] T. Ibaraki. *Enumerative Approaches to Combinatorial Optimization – Part 2*, volume 11 of *Annals of Operations Research*. Baltzer, Basel, 1987.

[15] K. Li and K.-H. Cheng. On three-dimensional packing. *SIAM Journal on Computing*, 19:847–867, 1990.

[16] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, Chichester, 1990.

[17] S. Martello and P. Toth. Lower bounds and reduction procedures for the bin packing problem. *Discrete Applied Mathematics*, 28:59–70, 1990.

[18] S. Martello and D. Vigo. Exact solution of the two-dimensional finite bin packing problem. *Management Science*, 1998. (to appear).

[19] D. Pisinger. The container loading problem. In *Proceedings NOAS'97*, 1997.

[20] G. Scheithauer. A three-dimensional bin packing algorithm. *J. Inform. Process. Cybernet.*, 27:263–271, 1991.