European Journal of Operational Research 141 (2002) 410–420

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

www.elsevier.com/locate/dsw

Heuristic algorithms for the three-dimensional bin packing problem

Andrea Lodi [*], Silvano Martello [1], Daniele Vigo

Dipartimento di Elettronica, Informatica e Sistemistica, Universita di Bologna, Viale Risorgimento, 2, 40136 Bologna, Italy

Received 30 October 2000; accepted 3 May 2001

Abstract

The *Three-Dimensional Bin Packing Problem* (3BP) consists of allocating, without overlapping, a given set of three-dimensional rectangular items to the minimum number of three-dimensional identical finite bins. The problem is NP-hard in the strong sense, and finds many industrial applications. We introduce a Tabu Search framework exploiting a new constructive heuristic for the evaluation of the neighborhood. Extensive computational results on standard benchmark instances show the effectiveness of the approach with respect to exact and heuristic algorithms from the literature. © 2002 Elsevier Science B.V. All rights reserved.

Keywords: Cutting; Packing; Heuristic algorithms; Tabu Search

1. Introduction

Given a set of n three-dimensional rectangular *items*, each characterized by width w_j, height h_j and depth d_j ($j \in J = \{1, \ldots, n\}$), and an unlimited number of identical three-dimensional rectangular containers (*bins*) having width W, height H and depth D, the *Three-Dimensional Bin Packing Problem* (3BP) consists of orthogonally packing, without overlapping, all the items into the minimum number of bins. It is assumed that the items have fixed orientation, i.e., they cannot be rotated. We will call *base* of an item (resp. bin) its $w_j \times h_j$ (resp. $W \times H$) side. We assume, without loss of generality, that all the input data are positive integers, and that $w_j \leqslant W$, $h_j \leqslant H$ and $d_j \leqslant D$ ($j \in J$).

The problem is strongly NP-hard as it generalizes the (one-dimensional) *Bin Packing Problem* (1BP), 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. Problem 3BP also generalizes the *Two-Dimensional Bin Packing Problem* (2BP), which consists of determining the minimum number of identical rectangular bins of width W and height H needed to orthogonally pack a given set of n rectangles having width w_j and height h_j ($j \in J$). Another related problem is

[*] Corresponding author. Tel.: +39-051-644-3029; fax: +39-051-644-3073.

E-mail addresses: alodi@deis.unibo.it (A. Lodi), smartello@deis.unibo.it (S. Martello), dvigo@deis.unibo.it (D. Vigo).

[1] Tel.: +39-051-2093022; fax: +39-051-2093073.

0377-2217/02/$ - see front matter © 2002 Elsevier Science B.V. All rights reserved.
PII: S0377-2217(02)00134-0

the *Three-Dimensional Strip Packing Problem* (3SP), in which there is a single bin (*strip*) of width W, depth D and infinite height, and we want to pack all the items into a strip of minimum height.

Three-dimensional packing problems have relevant practical interest in industrial applications such as, e.g., cutting of foam rubber in arm-chair production, container and pallet loading, and packaging design. Although 3BP is a simplified version of real-world problems, in many cases it arises as a subproblem.

Martello et al. [8] presented exact and heuristic approaches to three-dimensional packing problems. Færø et al. [4] recently proposed a Guided Local Search heuristic for 3BP, based on iterative solution of constraint satisfaction problems. As far as 1BP and 2BP are concerned the reader is referred to the surveys by Coffman et al. [1,2] and Lodi et al. [6], and to the annotated bibliography by Dyckhoff et al. [3].

In this paper we present a new approximation algorithm for 3BP, and discuss its use as subordinate heuristic within a Tabu Search metaheuristic approach derived from that proposed by Lodi et al. [7] for 2BP. The approximation algorithm is presented in Section 2, and the Tabu Search approach in Section 3. Finally, Section 4 gives the outcome of extensive computational experiments. A preliminary version of the algorithms is described in Lodi [5].

2. Heuristic algorithm

The 3BP heuristic algorithm we propose packs the items by *layers*. The *floor* of the first layer coincides with the base of a bin, and the items are packed with their bases on it. The floor of each subsequent layer in the bin is defined by the height of the tallest item packed into the layer below.

In order to produce an "effective" packing into layers one has to consider two main, possibly conflicting, issues: (i) to obtain a good "vertical" filling, by packing items with similar height in the same layer; (ii) to obtain a good "horizontal" filling, by effectively solving the two-dimensional packing of the item bases on the layer floors. To this end, the proposed algorithm, that will be called *Height first–Area second* (HA), chooses the best of two possible solutions (each enhancing one of the two issues), and can logically be subdivided into two phases:

1. The items are partitioned into clusters characterized by non-increasing height, and a layered strip packing solution is determined. The first 3BP solution is then constructed by combining the obtained layers into finite bins through a 1BP algorithm.
2. The items are re-sorted by non-increasing area of their base and re-allocated to the current layers, possibly modifying the layer heights, and the second solution is obtained through the 1BP algorithm.

The two phases above are described in more detail in the following sections.

2.1. Phase 1

The algorithm starts by sorting the items by non-increasing height. The items are then partitioned into clusters, each characterized by a different height, as follows. The first (tallest) item j defines the first cluster with height h_j, which includes all items having height h_k satisfying $h_k \geqslant \beta h_j$, where β ($\beta \in [0, 1]$) is a prefixed parameter. The tallest item s for which $h_s < \beta h_j$ defines the next cluster with height h_s, and so on. The items in each cluster are then sorted by non-increasing $w_j d_j$ value, and the item set is renumbered, by increasing cluster, following the new order.

Note that the value of β determines the way the items are partitioned into clusters. A value close to zero produces very few clusters and disregards the item heights, while a value close to one produces a very large number of clusters and disregards the area of the item bases. A good value should appropriately take into account both information. Through preliminary experimental evaluations we determined $\beta = 0.75$ as the value producing, on average, the partitions leading to the best packings.

The first item of the first cluster initializes the first layer: it is packed with its back left corner in the back left corner of the floor. The next items are packed one at a time, by increasing cluster and index. Given any point p on a layer floor, we will say that an item is packed in *position* p if its back left corner is on p.

Consider any packing pattern relative to a layer ℓ: it is well known that there exists an equivalent pattern (called *normal pattern*) in which no item base can be moved leftwards or backwards. We will always consider normal patterns. At each iteration, the next (tallest) unpacked item j is packed in the layer ℓ^* and the normal position p^* determined through a *score* $S(j,\ell,p)$, whose three terms (see Eq. (1) below) take into account, respectively, three different factors:

(i) the fraction of the perimeter of the base of j which touches the edges of the floor of ℓ or the edges of items already packed in ℓ;
(ii) the already packed portion of the floor of ℓ;
(iii) the relative difference between the height of ℓ and the height of j.

Let H_ℓ denote the height of layer ℓ, and J_ℓ the set of items already packed in ℓ. Let $P(j,\ell,p)$ be the perimeter of the base of j which will touch the edges of the floor of ℓ or the edges of items in J_ℓ if j is packed in layer ℓ in position p. The score is then

$$S(j,\ell,p) = \rho\frac{P(j,\ell,p)}{2w_j+2d_j} + \mu\frac{\sum_{k\in J_\ell} w_k d_k}{WD} - (1-\rho-\mu)\frac{|H_\ell - h_j|}{H_\ell}, \qquad (1)$$

where ρ and μ are prefixed real values satisfying $\rho,\mu \in [0,1]$ and $\rho+\mu \leqslant 1$. We set $S(j,\ell,p)=0$ if packing j in position p on the floor of ℓ would cause some portion of j to overlap another item or to go outside the bounds of the floor.

The next item, j, is packed, in order of preference:

(1) in an initialized layer having height no less than h_j (if any), in the maximum score position;
(2) in an initialized layer having height less than h_j (if any), in the maximum score position (by appropriately increasing the layer's height);
(3) in a new layer, with its back left corner in the back left corner of the floor.

More precisely, after the initial sorting, clustering and re-numbering, Phase 1 consists of executing the following:

```
**procedure** PACK:
    **for** $j := 1$ **to** $n$ **do**
        $S^* := 0$;
        **for each** normal position $p$ in an
        initialized layer $\ell$ satisfying $H_\ell \geqslant h_j$
        **do**
            compute $S(j,\ell,p)$;
            $S^* := \max\{S^*, S(j,\ell,p)\}$
        **end for**;
        **if** $S^* > 0$ **then** pack item $j$ in the layer
        and position corresponding to $S^*$
        **else**
            **for each** normal position $p$ in an
            initialized layer $\ell$ satisfying
            $H_\ell < h_j$ **do**
                compute $S(j,\ell,p)$;
                $S^* := \max\{S^*, S(j,\ell,p)\}$
            **end for**;
            **if** $S^* > 0$ **then**
                pack item $j$ in the layer and
                position corresponding to
                $S^*$, and set the layer height
                to $h_j$
            **else**
                initialize a new layer with
                height $h_j$, and pack item $j$
                into it
    **end for**
**end**.
```

Let $H_1,\ldots,H_g$ be the heights of the layers produced by procedure PACK. A finite bin packing solution is obtained by solving a 1BP instance defined by values H_k $(k=1,\ldots,g)$ and capacity H. We used exact algorithm MTP by Martello and Toth [9], with a limit of 50 backtrackings. Due to the limited number of resulting layers, this limit experimentally turned out to almost always produce, for the instances we used, the optimal 1BP solutions. Higher values, which could be appropriate for larger instances, would however affect the computing times of the Tabu Search algorithm described in the following section, in which algorithm HA is executed at each move.

2.2. Phase 2

The items are directly sorted and re-numbered by non-increasing $w_j d_j$ value. The g layers produced by Phase 1 are initialized, with heights $H_1, \ldots, H_g$, but with no item packed into them. Procedure PACK of Section 2.1 is then executed, and a new finite bin solution is obtained as at the end of Phase 1, by obviously disregarding empty layers (if any). The best of the two solutions is finally selected.

Example 1. We illustrate algorithm HA with a numerical example. Let $n = 5$, $W = H = D = 10$, $w_1 = 4$, $h_1 = d_1 = 10$, $w_2 = 5$, $h_2 = 9$, $d_2 = 6$, $w_3 = 6$, $h_3 = 7$, $d_3 = 6$, $w_4 = h_4 = d_4 = 5$, $w_5 = 6$, $h_5 = 1$, $d_5 = 10$. The items are already sorted by non-increasing height. By assuming $\beta = 1$, each item defines a cluster. By using $\rho = 0.3$ and $\mu = 0.7$, we obtain the packing into layers produced by PACK in Phase 1 as depicted in Fig. 1(a).

It is easy to see that the layers of Fig. 1(a) produce, in Phase 2, a finite bin solution using 3 bins. In Phase 3 the new order of the items, sorted by non-increasing $w_j d_j$ value, is: 5, 1, 3, 2, 4. The layers produced by PACK by using $\rho = 0.2$ and $\mu = 0.3$ are depicted in Fig. 1(b). The third (empty) layer is disregarded and the new finite bin solution uses 2 bins.

The two steps of HA can also be executed on the two instances we can obtain by interchanging widths, heights and depths, i.e., by creating layers whose floors are, respectively, on the height–depth plane, or on the height–width plane. In this way we obtain three different solutions, among which to select the best one (in terms of the number of used bins).

3. A Tabu Search approach

Lodi et al. [7] developed a unified Tabu Search framework for two-dimensional packing problems, by defining a search scheme and a neighborhood which are independent of the specific problem to be solved. In practice, the approach can be used for any variant of 2BP, by just changing a subordinate inner heuristic. Due to this characteristic, it was not difficult to adapt this framework to 3BP as well. The main modifications needed were the use of algorithm HA of Section 2 as inner heuristic, the extension of the so-called 'filling function' (see [7]) to the three-dimensional case, the way forbidden moves are stored in the tabu lists, and the diversification action. The resulting algorithm is briefly described hereafter.

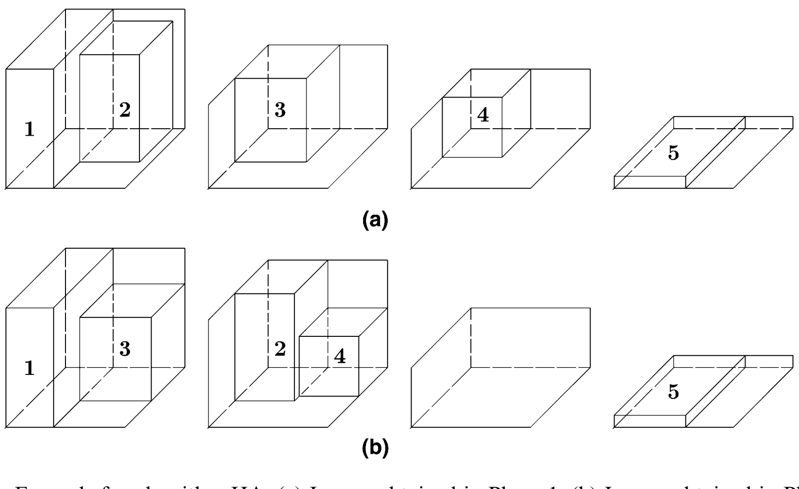

Fig. 1. Example for algorithm HA. (a) Layers obtained in Phase 1. (b) Layers obtained in Phase 3.

Table 1
Results for $n < 50$; Digital Alpha 533 MHz; 15 CPU seconds time limit for Tabu Search; average values over 10 instances

n	Class	H1/V	H2/V	HA/V	TS/V	BB/V
10	1	1.14	1.00	1.08	1.00	1.00
	2	1.13	1.00	1.00	1.00	1.00
	3	1.03	1.00	1.00	1.00	1.00
	4	1.00	1.00	1.00	1.00	1.00
	5	1.15	1.00	1.00	1.00	1.00
	6	1.13	1.00	1.05	1.05	1.00
	7	1.33	1.05	1.05	1.05	1.00
	8	1.22	1.00	1.10	1.05	1.00
Average		1.14	1.01	1.04	1.02	1.00
15	1	1.07	1.03	1.00	1.00	1.00
	2	1.15	1.00	1.05	1.02	1.00
	3	1.09	1.00	1.02	1.02	1.00
	4	1.02	1.01	1.00	1.00	1.00
	5	1.15	1.07	1.03	1.00	1.00
	6	1.13	1.02	1.00	1.00	1.00
	7	1.47	1.10	1.18	1.18	1.00
	8	1.22	1.07	1.05	1.05	1.00
Average		1.16	1.04	1.04	1.03	1.00
20	1	1.12	1.03	1.00	1.00	1.00
	2	1.04	1.00	1.02	1.00	1.00
	3	1.10	1.04	1.02	1.00	1.00
	4	1.02	1.00	1.00	1.00	1.00
	5	1.25	1.05	1.08	1.08	1.00
	6	1.12	1.03	1.00	1.00	1.00
	7	1.35	1.05	1.08	1.02	1.00
	8	1.09	1.00	1.00	1.00	1.00
Average		1.14	1.03	1.02	1.01	1.00
25	1	1.13	1.02	1.03	1.00	1.00
	2	1.17	1.05	1.06	1.03	1.00
	3	1.09	1.05	1.03	1.00	1.00
	4	1.02	1.01	1.00	1.00	1.00
	5	1.21	1.06	1.05	1.03	1.00
	6	1.15	1.04	1.02	1.00	1.00
	7	1.46	1.10	1.07	1.07	1.07
	8	1.27	1.06	1.04	1.02	1.00
Average		1.19	1.05	1.04	1.02	1.01
30	1	1.12	1.03	1.01	1.00	1.00
	2	1.17	1.05	1.04	1.01	1.00
	3	1.10	1.01	1.03	1.02	1.00
	4	1.01	1.01	1.00	1.00	1.00
	5	1.19	1.04	1.02	1.02	1.00
	6	1.18	1.04	1.03	1.00	1.00
	7	1.56	1.18	1.20	1.09	1.13
	8	1.21	1.05	1.04	1.00	1.00
Average		1.19	1.05	1.05	1.02	1.02
35	1	1.20	1.09	1.07	1.01	1.00
	2	1.11	1.03	1.03	1.00	1.00
	3	1.14	1.03	1.03	1.00	1.00
	4	1.01	1.01	1.00	1.00	1.00
	5	1.24	1.09	1.08	1.02	1.09

Table 1 (*continued*)

n	Class	H1/V	H2/V	HA/V	TS/V	BB/V
	6	1.21	1.10	1.07	1.05	1.00
	7	1.49	1.26	1.20	1.13	1.19
	8	1.35	1.20	1.09	1.08	1.16
Average		1.22	1.10	1.07	1.04	1.05
40	1	1.17	1.09	1.07	1.03	1.01
	2	1.15	1.07	1.07	1.03	1.03
	3	1.13	1.08	1.04	1.03	1.02
	4	1.02	1.00	1.00	1.00	1.00
	5	1.37	1.24	1.17	1.13	1.19
	6	1.17	1.10	1.07	1.03	1.00
	7	1.60	1.30	1.26	1.17	1.29
	8	1.30	1.17	1.10	1.06	1.02
Average		1.24	1.13	1.10	1.06	1.07
45	1	1.17	1.11	1.07	1.03	1.06
	2	1.16	1.07	1.07	1.04	1.04
	3	1.17	1.10	1.10	1.05	1.05
	4	1.02	1.00	1.00	1.00	1.00
	5	1.24	1.13	1.06	1.04	1.09
	6	1.18	1.12	1.08	1.02	1.00
	7	1.69	1.45	1.37	1.32	1.42
	8	1.32	1.18	1.13	1.06	1.06
Average		1.24	1.15	1.11	1.07	1.09

The first incumbent solution is obtained by executing algorithm HA. Let z be the number of bins used in the incumbent solution. The search is started by generating an initial (naive) current solution which packs each item into a separate bin. Let z^c be the number of bins used in the current solution.

A move modifies the current solution by changing the packing of a subset S of items, trying to empty a specified target bin. Let S_i be the set of items currently packed into bin i: the *target bin t* is the one minimizing, over all bins i, the *filling function*

$$\varphi(S_i) = \alpha \frac{\sum_{j\in S_i} w_j h_j d_j}{WHD} - \frac{|S_i|}{n} \tag{2}$$

(α is a pre-specified positive weight), which gives a measure of the easiness of emptying the bin.

Once the target bin has been selected, subset S is defined so as to include one item, j, from the target bin and the current contents of k other bins. The new packing for S is obtained by executing algorithm *HA* on S. The value of parameter k, which defines the size and the structure of the current neighborhood, is automatically updated during the search. Initially, k is set to 1.

If the move packs the items of S into k (or less) bins, i.e., item j has been removed from the target bin, a new item is selected, a new set S is defined accordingly, and a new move is performed. Otherwise S is changed by selecting a different set of k bins, or a different item from the target bin (when all possible configurations of k bins have been attempted for the current j). Whenever a move produces a solution using less than z bins, the incumbent solution is updated.

We halt the current search when one of the two situations occurs: (i) all items j from the target bin and all possible configurations of k bins have been attempted without finding a feasible move; (ii) ℓ moves have been attempted without improving the incumbent solution (where ℓ is a prefixed parameter). In both cases, if $k < k_{\max}$ (where $k_{\max}$ is a prefixed upper limit) then the value of k is increased by one and the new (enlarged) neighborhood is explored; otherwise a diversification action is performed.

There are a tabu list and a tabu tenure τ_k for each value of k. A tabu list stores, for each forbidden move, the sum of the filling function values of the $k+1$ involved bins.

Table 2
Results for $n \geqslant 50$; Digital Alpha 533 MHz; 60 CPU seconds time limit for Tabu Search; average values over 10 instances

n	Class	H1/V	H2/V	HA/V	TS/V	BB/V
50	1	1.19	1.14	1.08	1.04	1.06
	2	1.25	1.14	1.11	1.08	1.08
	3	1.20	1.14	1.11	1.05	1.07
	4	1.02	1.01	1.00	1.00	1.00
	5	1.34	1.28	1.13	1.11	1.24
	6	1.20	1.12	1.07	1.01	1.00
	7	1.49	1.32	1.27	1.17	1.28
	8	1.40	1.29	1.21	1.13	1.24
Average		1.26	1.18	1.12	1.07	1.12
60	1	1.26	1.15	1.14	1.08	1.10
	2	1.21	1.15	1.11	1.07	1.09
	3	1.22	1.14	1.12	1.08	1.09
	4	1.02	1.02	1.01	1.01	1.01
	5	1.45	1.31	1.25	1.16	1.31
	6	1.15	1.13	1.06	1.02	1.00
	7	1.56	1.41	1.28	1.22	1.38
	8	1.25	1.17	1.10	1.04	1.09
Average		1.26	1.18	1.13	1.08	1.13
70	1	1.24	1.15	1.13	1.09	1.11
	2	1.22	1.15	1.11	1.07	1.09
	3	1.22	1.16	1.13	1.08	1.11
	4	1.05	1.02	1.02	1.02	1.02
	5	1.30	1.27	1.16	1.11	1.25
	6	1.22	1.20	1.11	1.05	1.05
	7	1.65	1.47	1.33	1.25	1.46
	8	1.36	1.27	1.15	1.09	1.17
Average		1.28	1.21	1.14	1.09	1.16
80	1	1.19	1.17	1.12	1.08	1.09
	2	1.19	1.15	1.12	1.08	1.10
	3	1.17	1.13	1.09	1.06	1.09
	4	1.03	1.03	1.02	1.02	1.02
	5	1.36	1.33	1.18	1.12	1.33
	6	1.20	1.20	1.11	1.06	1.08
	7	1.45	1.39	1.24	1.18	1.38
	8	1.36	1.34	1.19	1.15	1.25
Average		1.24	1.22	1.13	1.09	1.17
90	1	1.18	1.14	1.10	1.07	1.09
	2	1.18	1.14	1.11	1.07	1.09
	3	1.17	1.15	1.11	1.07	1.09
	4	1.04	1.04	1.03	1.03	1.03
	5	1.32	1.34	1.20	1.14	1.30
	6	1.15	1.18	1.06	1.03	1.02
	7	1.49	1.46	1.29	1.19	1.45
	8	1.35	1.32	1.20	1.15	1.24
Average		1.24	1.22	1.14	1.09	1.16
100	1	1.19	1.14	1.10	1.07	1.09
	2	1.19	1.16	1.12	1.07	1.10
	3	1.17	1.14	1.10	1.07	1.10
	4	1.05	1.04	1.02	1.02	1.03
	5	1.37	1.40	1.22	1.17	1.36

Table 2 (*continued*)

n	Class	H1/*V*	H2/*V*	HA/*V*	TS/*V*	BB/*V*
	6	1.25	1.26	1.14	1.09	1.10
	7	1.46	1.41	1.23	1.16	1.41
	8	1.23	1.25	1.14	1.10	1.16
Average		1.24	1.22	1.13	1.09	1.17

Two different diversification actions are performed alternatively. The first one ('soft' diversification) simply consists in selecting as target bin the one having the second smallest filling function value (see (2)). The second one ('hard' diversification) consists in re-packing into separate bins the items currently packed in the $\lfloor z^c/2 \rfloor$ bins with the smallest $\varphi(S_i)$ value. In both cases, all tabu lists are reset to empty, and the search is restarted with $k = 1$.

Preliminary computational experiments allowed to set the parameters needed by the Tabu Search to the following values: $\alpha = 1.5$, $k_{\max} = 3$, $\tau_k = 20$ for all k, and $\ell = 100 - 25(k-1)$. (More details on these experiments can be found in Lodi [5].)

4. Computational experiments

The algorithms of the two previous sections were coded in FORTRAN 77 and run on a Digital Alpha 533 MHz. For algorithm HA of Section 2, the parameters were experimentally set to the following values: $\beta = 0.75$; $\rho = 0.3$ for Phase 1 and $\rho = 0.2$ for Phase 3; $\mu = 0.7$ for Phase 1 and $\mu = 0.3$ for Phase 3. The algorithms were tested on eight classes of instances proposed by Martello et al. [8]. (The instance generator is available at http://www.diku.dk/~pisinger/codes.html.) The same classes have been used by Færø et al. [4] for testing their Guided Local Search heuristic (GLS) for 3BP.

For Classes 1–5, the bin size is $W = H = D = 100$ and the following five types of items are considered:

type 1: w_j uniformly random 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 uniformly random 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 uniformly random 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 uniformly random 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 uniformly random in $[1, \frac{1}{2}W]$, h_j in $[1, \frac{1}{2}H]$, d_j in $[1, \frac{1}{2}D]$;

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. Classes 6–8 are as follows:

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

In Tables 1–3, the proposed algorithms are compared with the algorithms proposed by Martello et al. [8], namely heuristic algorithms H1 and H2 and exact algorithm BB. For each class, 140 instances were solved, 10 for each value of $n \in \{10, 15, 20, 25, 30, 35, 40, 45, 50, 60, 70, 80, 90, 100\}$, yielding a total of 1120 instances. In Tables 1 and 2, we report, for each algorithm, the average ratio (*solution value*)/V, where V is the optimal solution value if known, or the highest lower bound value otherwise. V was computed by running the branch-and-bound algorithm of [8] with a time limit of 2000 CPU seconds on a HP PA9000/782 (200 MHz). The Tabu Search algorithm, TS, was run with a time limit of 15 CPU seconds for instances with $n < 50$, and with a time limit of 60 CPU seconds for instances with $n \geqslant 50$. The computing times of algorithm HA, as well as those of H1 and H2, were negligible.

Although the proposed algorithms can only obtain solutions composed by layers (thus solving a more constrained packing problem), Tables 1 and 2 show a satisfactory behavior of HA and TS

Table 3
Aggregated results for Tabu Search; entries give the number of instances out of 80 (1120 in total)

n	10	15	20	25	30	35	40	45	50	60	70	80	90	100	*Total*	*Percentage*
TS $\leqslant$ BB	77	74	77	75	77	75	73	75	79	78	78	78	78	79	1073	95.80
TS $<$ BB	0	0	0	1	1	10	8	13	21	29	36	39	39	47	244	21.79
TS opt	77	74	77	74	74	67	57	50	43	29	20	15	13	4	674	60.18

with respect to H1 and H2. In particular, HA dominates H1 on all instances, and dominates H2 for $n > 30$. The Tabu Search approach was effective in improving the solutions found by HA: with few exceptions for very small instances, it dominates all other heuristic algorithms. Furthermore, for sufficiently large *n* values, it produced solutions consistently better than the best incumbent solutions obtained by branch-and-bound within a much larger time limit.

The good behavior of TS is further illustrated in Table 3, where we report aggregated results comparing it with the branch-and-bound algorithm, BB. For each value of *n*, we give the number of instances (over 80) where TS: (i) was at least as good as BB; (ii) was strictly better than BB, and (iii) obtained a proven optimal solution.

Finally, in Table 4, we compare our TS approach with the Guided Local Search heuristic, GLS, recently proposed by Færø et al. [4]. This algorithm is not restricted to layer packings. We ran TS on the same instances considered in [4], i.e., Classes 1–8 with $n \in \{50, 100, 150, 200\}$. The entries give, for each algorithm and for different time limits, the average solution value (number of bins) over 10 instances. The values in columns GLS are taken from [4], where, for Classes 2 and 3, the authors say that the results were very similar to those of Class 1 (without explicitly reporting them). The experiments of Færø et al. [4] were performed on a Digital workstation 500 a.u. 500 MHz, similar in speed to the Digital Alpha 533 MHz we used. The results show a satisfactory behavior of both approaches. In one third of the cases, they produced exactly the same average solution values. For Classes 1 (hence, presumably, 2 and 3 as well) and 4, TS is generally better than GLS. The reverse holds for Classes 5, 7 and 8, while equivalent performances can be observed for Class 6. It turns out that the two approaches complement each other well.

Acknowledgements

We thank the Ministero dell'Università e della Ricerca Scientifica e Tecnologica (MURST) and the Consiglio Nazionale delle Ricerche (CNR), Italy, for the support given to this project. The

Table 4
Comparison of metaheuristic approaches

Class	n	GLS			TS		
		tl = 60	tl = 150	tl = 1000	tl = 60	tl = 150	tl = 1000
1	50	13.4	13.4	13.4	13.4	13.4	13.4
	100	26.9	26.7	26.7	27.1	26.6	26.6
	150	37.5	37.2	37.0	37.3	37.3	36.7
	200	52.8	52.1	51.2	51.9	51.6	51.2
2	50	–	–	–	13.8	13.8	13.8
	100	–	–	–	25.8	25.7	25.7
	150	–	–	–	37.5	37.4	37.2
	200	–	–	–	50.5	50.5	50.1
3	50	–	–	–	13.4	13.3	13.3
	100	–	–	–	26.3	26.3	26.0
	150	–	–	–	38.1	38.0	37.7
	200	–	–	–	50.6	50.6	50.5
4	50	29.4	29.4	29.4	29.4	29.4	29.4
	100	59.0	59.0	59.0	59.0	59.0	59.0
	150	87.1	86.9	86.8	86.8	86.8	86.8
	200	119.9	119.7	119.0	118.8	118.8	118.8
5	50	8.3	8.3	8.3	8.4	8.4	8.4
	100	15.1	15.1	15.1	15.1	15.1	15.0
	150	20.7	20.3	20.2	20.7	20.7	20.4
	200	27.8	27.5	27.2	28.0	27.8	27.6
6	50	9.8	9.8	9.8	9.9	9.9	9.9
	100	19.3	19.1	19.1	19.3	19.1	19.1
	150	29.5	29.4	29.4	29.7	29.4	29.4
	200	38.5	38.0	37.7	38.2	38.0	37.7
7	50	7.4	7.4	7.4	7.5	7.5	7.5
	100	12.5	12.3	12.3	12.6	12.5	12.5
	150	16.1	15.8	15.8	16.5	16.3	16.1
	200	24.4	24.1	23.5	24.6	24.4	23.9
8	50	9.2	9.2	9.2	9.3	9.3	9.3
	100	18.9	18.9	18.9	19.0	18.9	18.9
	150	24.5	24.1	23.9	24.6	24.6	24.1
	200	30.6	30.1	29.9	31.0	30.9	30.3

computational experiments have been executed at the Laboratory of Operations Research of the University of Bologna (Lab. OR). Thanks are also due to Matteo Porcù for his help with programming.

References

[1] E.G. Coffman Jr., G. Galambos, S. Martello, D. Vigo, Bin packing approximation algorithms: Combinatorial analysis, in: D.-Z. Du, P.M. Pardalos (Eds.), Handbook of Combinatorial Optimization, Kluwer Academic Publishers, Boston, MA, 1999, pp. 151–207.

[2] E.G. Coffman Jr., M.R. Garey, D.S. Johnson, Approximation algorithms for bin packing: A survey, in: D.S. Hochbaum (Ed.), Approximation Algorithms for NP-Hard Problems, PWS Publishing Company, Boston, 1997, pp. 46–93.

[3] H. Dyckhoff, G. Scheithauer, J. Terno, Cutting and packing (C&P), in: M. Dell'Amico, F. Maffioli, S. Martello (Eds.), Annotated Bibliographies in Combinatorial Optimization, Wiley, Chichester, 1997, pp. 393–413.

[4] O. Færø, D. Pisinger, M. Zachariasen, Guided local search for the three-dimensional bin packing problem, Technical paper, DIKU, University of Copenhagen, 1999.

[5] A. Lodi, Algorithms for two-dimensional bin packing and assignment problems, Ph.D. Thesis, University of Bologna, Italy, 2000.
[6] A. Lodi, S. Martello, M. Monaci, Two-dimensional packing problems: A survey, European Journal of Operational Research, this issue.
[7] A. Lodi, S. Martello, D. Vigo, Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems, INFORMS Journal on Computing 11 (1999) 345–357.
[8] S. Martello, D. Pisinger, D. Vigo, The three-dimensional bin packing problem, Operations Research 48 (2000) 256–267.
[9] S. Martello, P. Toth, Knapsack Problems: Algorithms and Computer Implementations, Wiley, Chichester, 1990.