ELSEVIER

European Journal of Operational Research 112 (1999) 158–166

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

Theory and Methodology

Approximation algorithms for the oriented two-dimensional bin packing problem

Andrea Lodi [1], Silvano Martello [*], Daniele Vigo [2]

Dipartimento di Elettronica, Informatica e Sistemistica, University of Bologna, Viale Risorgimento, 2-40136-Bologna, Italy

Received 2 April 1997; accepted 30 September 1997

Abstract

Given a set of rectangular items which may not be rotated and an unlimited number of identical rectangular bins, we consider the problem of packing each item into a bin so that no two items overlap and the number of required bins is minimized. The problem is strongly NP-hard and finds practical applications in cutting and packing. We discuss a simple deterministic approximation algorithm which is used in the initialization of a tabu search approach. We then present a tabu search algorithm and analyze its average performance through extensive computational experiments.

Keywords: Packing; Cutting; Heuristics; Tabu search

1. Introduction

Given n rectangular *items* y, each characterized by a height h_j and a width $w_j j = 1, \ldots, n$, and an unlimited number of identical rectangular *bins*, each having height H and width W, the *Oriented Two-Dimensional Bin Packing Problem* (2D-BPP) is to pack each item into a bin so that no two items overlap and the number of required bins is minimized. We assume that the items cannot be rotated. We also assume, without loss of generality, that all input data are positive integers and that $h_j \leqslant H,\ w_j \leqslant W\ \ (j = 1, \ldots, n)$. According to the typology given by Dykhoff [1], the problem can be classified as 2/V/I/M.

The problem is known to be strongly NP-hard and finds many practical applications. For example, in the glass industry and in the wood industry it is requested to cut rectangular items in demand from standardized stock sheets by minimizing the trim loss, i.e., by using the minimum number of stock sheets. According to the specific application, the items in demand may have a prefixed orientation (e.g., in the cutting of corrugated iron) or may be rotated (usually by 90°). As previously stated, this paper deals with the *oriented case* in which no item rotation is allowed. In cutting applications it is frequently required that the resulting patterns are *guillotine-cuttable*, i.e., the items

* Corresponding author. E-mail: smartello@deis.unibo.it
[1] E-mail: alodi@deis.unibo.it
[2] E-mail: dvigo@deis.unibo.it

0377-2217/99/$ – see front matter
PII S0377-2217(97)00388-3

can be obtained through a sequence of edge-to-edge cuts parallel to the edges of the bin: as will be pointed out later, the algorithms we propose satisfy such a constraint.

Approximation algorithms for 2D-BPP have been given by Chung et al. [2], Berkey and Wang [3] and Frenk and Galambos [4], whereas Bengtsson [5] and El-Bouri et al. [6] considered the case with 90° rotation of the items. An exact branch-and-bound approach for 2D-BPP has been developed by Martello and Vigo [7]. Surveys on packing problems can be found in Dyckhoff and Finke [8], and Dowsland and Dowsland [9], while an annotated bibliography has recently been presented by Dyckhoff et al. [10].

This paper presents a tabu search approach to 2D-BPP. The algorithm is initialized with the solution obtained by using a simple and fast heuristic which proved to be an excellent starting point for the subsequent search. In Section 2 we describe this heuristic and discuss its worst case performance. In Section 3 we define the neighborhoods used and give the relevant details of the tabu search algorithm. The results of extensive computational experiments are given in Section 4.

2. Initial feasible solution

In this section we describe and analyze a simple heuristic used to initialize the tabu search algorithm. Although its absolute worst case performance is poor and the solutions it produces require, on average, a high number of bins, it has experimentally proved to provide a very good starting point for the subsequent tabu search. Indeed, better starting solutions (e.g., those provided by algorithms FFF and FBS, see Section 3) proved to be much harder to modify through our tabu search. On the other hand, starting with trivial solutions typically increases the number of iterations needed to obtain good local optima.

The algorithm is based on a technique developed by Martello and Vigo [7] for proving the worst-case performance of the continuous lower bound for 2D-BPP. We start by packing all the items into a strip having height H and infinite width: the strip is then cut so as to produce a feasible 2D-BPP solution. Let (x, y) be a coordinate system with the origin in the bottom left corner of the strip. For each item j we define an attribute $class(j) = \min\{r:\ h_j \geqslant \frac{H}{2^{r+1}}, r \text{ integer}\}$: observe that any horizontal unit of strip containing 2^r items of class r has at least half of its area occupied by such items. Hence we define the set of admissible vertical coordinates for any item of class r as $V(r) = \{\frac{t}{2^r}H:\ t = 0, 1, \ldots, 2^r - 1\}$, and we pack the items according to non-increasing class, at admissible vertical coordinates in such a way that the total occupied area before a current horizontal coordinate $\overline{x}$ is no less than $\overline{x}H/2$. The resulting strip is then subdivided into *slices* of width W and, for each slice, a feasible packing requiring one or two bins is determined. The algorithm can be outlined as follows. Set X contains the horizontal coordinates, greater than $\overline{x}$, corresponding to right edges of already packed items.

Algorithm IH

0. for $j := 1$ **to** n **do**
  if $h_j < H$ **then** $class(j) := \lceil \log_2(H/h_j) \rceil - 1$
  else $class(j) := 0$;
  sort (and renumber) the items by nondecreasing $class(j)$ values;
  $\overline{x} := 0$;
  $X := \emptyset$;
1. for $j := 1$ **to** n **do**
  begin
    if all admissible vertical coordinates at $\overline{x}$ are occupied **then**
      begin
        $\overline{x} := \min\{x:\ x \in X\}$;
        $X := X \setminus \{\overline{x}\}$
      end;
    place item j with its bottom left corner in $(\overline{x}, \overline{y})$, where $\overline{y}$ is the lowest unoccupied admissible vertical coordinate of $V(class(j))$, at $\overline{x}$; $X := X \cup \{\overline{x} + w_j\}$
  end;
2. $k := \lfloor \overline{x}/W \rfloor$ (**comment:** observe that no item can terminate after $(k+2)W$);
  subdivide the strip into $k+2$ slices of width W, starting from the origin;
  for $i := 1$ **to** k **do**
  begin
    pack the items entirely contained in slice i,

if any, into a new bin;
pack the items crossing the right boundary of slice i, if any, into a new bin
end;
pack the items terminating after $\bar{x}$, if any, into a new bin;
pack the remaining items of slice $k+1$, if any, into a new bin
end.

Fig. 1 gives an example of packing obtained at the end of Step 1. (After renumbering, we have $[class(j)] = [0,1,1,1,2,2,2,2,2,3]$; observe that ties may be broken arbitrarily.) At Step 2 we have $k = 2$: two bins are then introduced for slice 1 (with contents $\{1\}$ and $\{2,3\}$), two for slice 2 (with contents $\{4,5\}$ and $\{6,7\}$), one for $\{9,10\}$ and one for $\{8\}$.

It is easily seen that algorithm IH can be implemented so as to run in $O(n \log n)$ time. Indeed, by using a heap for storing set X, the operations involving set X at each iteration of the *for* loop in Step 1 may be performed in $O(\log n)$ time. On the other hand, whenever an item is placed in $(\bar{x},\bar{y})$, the next unoccupied admissible vertical coordinate at $\bar{x}$ (if any) can be easily determined in constant time.

It is also clear that the patterns produced by IH are guillotine-cuttable. The items can indeed be obtained through the following recursive operations. Let $\tilde{H}$ be the height of the current strip: obtain each item j having $h_j \geqslant \tilde{H}/2$ through a vertical edge-to-edge cut, possibly followed by an horizontal edge-to-edge cut; subdivide the remaining part of the strip into two strips of height $\tilde{H}/2$ through an horizontal edge-to-edge cut, and so on, recursively.

Given any instance I of a minimization problem P, let $z(I)$ be the value of the optimal solution to I, and $U(I)$ the value provided by a heuristic algorithm U. The *worst-case performance ratio* of U is defined as the smallest real number ρ such that

$$\frac{U(I)}{z(I)} \leqslant \rho \quad \text{for all instances } I \text{ of } P.$$

Theorem 1. *The worst-case performance ratio of algorithm IH is 4.*

Proof. Let z^* denote the optimal number of bins and $\bar{z}$ the number of bins produced by IH. It has been proved in [7] that for any instance of 2D-BPP we have $\bar{z} \leqslant 4\text{LB}$, where LB is a lower bound on z^*. Hence $\bar{z} \leqslant 4z^*$. To prove that this ratio is tight it is sufficient to consider an instance with $n = 6$, $h_1 = H/2$, $w_1 = W/2 + 1$, $h_2 = h_3 = h_4 = h_5 = h_6 = H/4$, $w_2 = w_3 = W/2$, $w_4 = w_6 = W/2 - 1$, $w_5 = W$. The strip packing solution determined by algorithm IH is shown in Fig. 2(a): we then obtain four bins containing, respectively, $\{1\}$, $\{2,3\}$, $\{5,6\}$ and $\{4\}$. The optimal single bin solution is shown in Fig. 2(b). □

The worst-case performance of IH is thus quite poor. Chung et al. [2] presented a hybrid algorithm, HFF, whose solution value satisfies $\text{HFF}(I)/z(I) < \frac{17}{8} + 5/z(I)$. This performance is

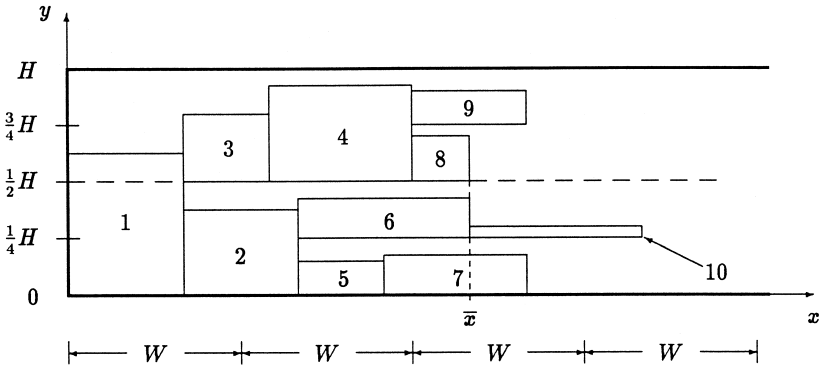

Fig. 1. Example of the strip packing solution determined by algorithm IH.

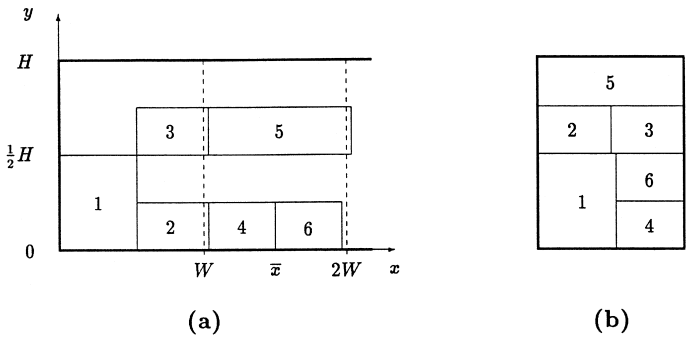

Fig. 2. Worst-case example for algorithm IH.

better than that of IH for any instance I with $z(I) \geqslant 3$. However, as previously mentioned, the structure of the solutions provided by IH experimentally proved to be particularly suited for the tabu search initialization.

3. Tabu search

The algorithm starts by computing a lower bound LB as described in [7] and by applying two simple heuristics, called FFF (*Finite First-Fit*) and FBS (*Finite Best Strip*), described in [3].

The FFF heuristic initially sorts the items by non-increasing height and then iteratively packs each of them into the first initialized bin in which it fits, or into a new bin if no such bin exists.

The FBS heuristic initially sorts the items by non-increasing height and consists of two phases. First the items are packed into an infinite height strip using a best-fit algorithm to select the level for each item: the resulting strip packing is made up of "shelves", each corresponding to a different level, having width equal to the strip width and a different height. In the second phase the shelves are packed into finite bins using the well-known best-fit heuristic for the one-dimensional bin packing problem (see, Johnson et al. [11]). Algorithms FFF and FBS may be implemented to run in $O(n^2)$ and $O(n \log n)$ time, respectively. It is known that both FFF and FBS produce patterns that satisfy the guillotine constraint.

Let UB denote the value of the best heuristic solution found among those given by FFF and FBS. If UB > LB the tabu search algorithm is initialized with the solution produced by algorithm IH. The search is based on a main loop where, at each iteration, one of two possible neighborhoods is explored. Let z denote the number of bins used in the current solution. The algorithm accepts moves that either decrease z or re-distribute items among the z bins, while moves increasing z are never accepted. As soon as no acceptable move is possible, a restart is performed.

Both neighborhoods consist of moves involving the items of a particular bin, which is called the *weakest bin* and is defined as follows. Let S_i denote the set of items that, in the current solution, are packed into bin i ($i = 1, \ldots, z$). The weakest bin is then defined as the one which minimizes the quantity

$$\varphi(i) = \alpha \frac{\sum_{j \in S_i} h_j w_j}{HW} - \frac{|S_i|}{n} \tag{1}$$

that gives an estimate of the difficulty of emptying bin i (α is a prefixed non-negative parameter). At each iteration we consider an item j currently packed into the weakest bin b and we try to remove j from b: the first neighborhood tries to directly pack j into a different bin, while the second neighborhood tries to re-combine the items of two different bins so that one of them can accomodate j. The core of the algorithm can be outlined as follows:

determine the weakest bin b;
while a *stopping criterion* is not satisfied **do**

```
begin
  repeat
    perform the next move according to the
    first neighborhood;
    if S_b = ∅ then determine the new weak-
    est bin b
  until no move is possible;
  repeat
    perform the best move according to the
    second neighborhood;
    determine the new weakest bin b̄
  until b ≠ b̄;
  b := b̄
end
```

Given a subset of items S, let $\mathrm{FBS}(S)$ denote the number of $H \times W$ bins used by algorithm FBS for packing all items of S. Both neighborhoods use FBS for re-combining subsets of items (note indeed that determining the *optimal* packing of a single bin is already an NP-hard problem).

The *first neighborhood* is obtained by determining the weakest bin b and by considering, in turn, each non-tabu item j currently packed into b: $\mathrm{FBS}(\{j\} \cup S_i)$ is then iteratively computed for bins $i \neq b$. As soon as a bin i^* is found for which $\mathrm{FBS}(\{j\} \cup S_{i^*}) = 1$, item j is moved to this bin, it is inserted in the first tabu-list, and the next item of b is considered. An aspiration criterion is used: if b contains a single item, then the search is performed even if this item is tabu, since the possible move would improve z. Whenever the weakest bin becomes empty, the new weakest bin is determined and the search continues with the first neighborhood. As soon as no move is possible, the search explores the second neighborhood. The first tabu list, which stores the indices of recently moved items, is preserved for the next exploration of the first neighborhood.

The second tabu list contains *scores* given by the total area of a subset of items representing a possible move. Also the *second neighborhood* considers, in turn, items j currently packed into the weakest bin b. For each of these j and for each pair of bins h and k, $(h, k \neq b)$, set $S = \{j\} \cup S_h \cup S_k$ is defined, and $\mathrm{FBS}(S)$ is computed. Four cases may occur:

1. $\mathrm{FBS}(S) = 1$, i.e., z is decreased by one or two units: the move is immediately performed and the exploration is resumed from the first neighborhood;
2. $\mathrm{FBS}(S) = 2$, i.e., item j can be removed from bin b: the move is immediately performed. Two subcases are then possible:
   2.1. $|S_b| = 0$, i.e., the move has decreased z by one: the exploration is resumed from the first neighborhood;
   2.2. $|S_b| > 0$: the new weakest bin $\overline{b}$ is determined among b, h, and k. The search continues with the second neighborhood if $\overline{b} \equiv b$, or with the first neighborhood, otherwise;
3. $\mathrm{FBS}(S) = 3$: let T be the set of items packed by FBS into the bin with minimum $\varphi(i)$ among the three resulting bins. Two subcases are possible:
   3.1. $\mathrm{FBS}(T \cup (S_b \setminus \{j\})) > 1$: the move would increase z, hence it is not accepted;
   3.2. $\mathrm{FBS}(T \cup (S_b \setminus \{j\})) = 1$: a *score*, defined by the ϕ value of the bin that packs the items in $T \cup (S_b \setminus \{j\})$, is assigned to the move. If this score is not tabu, the best acceptable move not improving z is possibly updated;
4. $\mathrm{FBS}(S) > 3$: the move is not accepted since it would increase z.

At the end of the exploration the best non-improving acceptable move (if any) is performed, its score is stored in the second tabu list, and the search continues with the first neighborhood. For both tabu lists we used static tabu tenures of length τ_1 and τ_2, respectively.

The *stopping criteria* adopted are quite standard. The search terminates if (i) a feasible solution of value LB is determined, or (ii) a prefixed maximum number of iterations is attained, or (iii) a prefixed time limit is reached.

The algorithm also includes *restart actions*. If no acceptable move is found, or if the incumbent solution value z was not improved during the last μ moves, we modify the incumbent solution as follows. We renumber the bins so that $\varphi(i) \geqslant \varphi(i+1)$ for all i and execute algorithm IH on the items of $\bigcup_{i=1}^{\lfloor z/2 \rfloor} S_i$, adding to the obtained solution the incumbent packings of bins $\lfloor z/2 \rfloor + 1, \ldots, z$. The resulting solution is then used to re-start the search.

Table 1
Results on problem instances from the literature. Time limit of 100 CPU seconds on a Silicon Graphics INDY R4000sc

Name	n	LB	FFF	FBS	TS		Time
beng1	20	4	4	4	4	*	0.01
beng2	40	6	7	7	7		100.02
beng3	60	9	10	9	9	*	0.01
beng4	80	11	12	12	12		100.06
beng5	100	14	16	15	14	*	0.01
beng6	40	2	2	2	2	*	0.01
beng7	80	3	3	3	3	*	0.01
beng8	120	5	5	5	5	*	0.01
cgcut1	16	2	2	2	2	*	0.01
cgcut2	23	2	3	3	2	*	0.01
cgcut3	62	23	26	26	23	*	0.01
gcut1	10	4	5	5	5	*	100.02
gcut2	20	6	7	7	6	*	50.11
gcut3	30	8	9	8	8	*	0.01
gcut4	50	13	15	15	14	*	100.03
gcut5	10	3	4	4	4		100.02
gcut6	20	6	8	8	7	*	100.02
gcut7	30	10	12	12	12		100.01
gcut8	50	12	15	14	14		100.03
gcut9	10	3	3	3	3	*	0.01
gcut10	20	7	8	8	8		100.03
gcut11	30	8	10	10	9	*	100.03
gcut12	50	16	17	17	16	*	23.56
gcut13	32	2	2	2	2	*	0.01
ngcut1	10	2	3	3	3	*	100.02
ngcut2	17	3	4	4	4	*	100.02
ngcut3	21	3	4	4	4		100.02
ngcut4	7	2	2	2	2	*	0.01
ngcut5	14	3	4	4	3	*	0.01
ngcut6	15	2	3	3	3	*	100.02
ngcut7	8	1	2	2	1	*	0.01
ngcut8	13	2	2	2	2	*	0.01
ngcut9	18	3	4	4	4		100.02
ngcut10	13	3	3	3	3	*	0.01
ngcut11	15	2	3	3	3		100.02
ngcut12	22	3	4	4	4		100.02

We finally observe that the algorithm starts with a solution produced by IH (hence guillotine-cuttable) and modifies it through FBS, thus the resulting patterns satisfy the guillotine constraint.

4. Computational results

The tabu search algorithm was coded in FORTRAN 77 and run on a Silicon Graphics INDY R4000sc 100MHz on test instances from the literature. The values we used for the parameters of the tabu search algorithm are $\alpha = 5.0$, $\tau_1 = 3$, $\tau_2 = 5$, $\mu = 50$.

Given any instance of 2D-BPP, by interchanging H with W and h_j with w_j ($j = 1, \ldots, n$), we obtain an equivalent instance for which, however, a heuristic algorithm can produce a different solution. Hence, for each instance, we executed both the initial upper bound computation and the tabu search twice,

getting the best solution: an overall limit of 100 CPU seconds was assigned for the solution of each instance (50 seconds per execution).

Table 1 gives the results obtained on instances from the literature with up to 120 items. Problems `bengl`–`beng8` are the instances used in [5] for the variant of 2D-BPP where 90° rotation of the items is allowed. The remaining problems are instances proposed in the literature for other two-dimensional cutting problems and transformed into 2D-BPP instances by using the relative bin and item sizes. In particular problems `cgcutl`–`cgcut3` are described in Christofides and Whitlock [12], while problems `gcutl`–`gcutl3` and `ngcutl`–`ngcutl2` are described in Beasley [13,14], respectively. Problems `bengl`–`beng8` and `ngcutl`–`ngcutl2` were originally proposed to test algorithms not satisfying the guillotine constraint.

For each problem the table gives the problem name, the number of items, the values of LB, FFF, FBS and TS (tabu search solution) and the computing time, expressed in seconds, required by the tabu search. An asterisk indicates the instances where TS is optimal (the values of the optimal solutions were taken from [7]).

In Table 2 we present the results for six classes of instances randomly generated as in [3]. Each

Table 2
Results on the random problem instances proposed by Berkey and Wang. Time limit of 100 CPU seconds on a Silicon Graphics INDY R4000sc

Items	*Bins*	*n*	FFF/LB	FBS/LB	TS/LB	Time	$\leqslant$ B & B
$[1,10] \times [1,10]$	10×10	20	1.165	1.136	1.061	40.01	10
		40	1.122	1.090	1.072	85.13	8
		60	1.096	1.074	1.047	90.03	9
		80	1.083	1.060	1.030	67.89	8
		100	1.074	1.061	1.035	96.92	7
$[1,10] \times [1,10]$	30×30	20	1.100	1.100	1.100	0.01	9
		40	1.100	1.100	1.100	0.01	10
		60	1.150	1.150	1.150	30.00	10
		80	1.067	1.067	1.033	15.00	10
		100	1.058	1.058	1.033	10.00	10
$[1,35] \times [1,35]$	40×40	20	1.197	1.177	1.177	80.01	7
		40	1.177	1.136	1.110	90.01	8
		60	1.138	1.106	1.078	80.13	7
		80	1.131	1.099	1.073	90.02	8
		100	1.123	1.091	1.072	100.04	7
$[1,35] \times [1,35]$	100×100	20	1.000	1.000	1.000	0.01	10
		40	1.100	1.100	1.100	0.01	10
		60	1.200	1.200	1.200	40.00	10
		80	1.100	1.100	1.100	30.01	10
		100	1.100	1.100	1.067	20.00	10
$[1,100] \times [1,100]$	100×100	20	1.140	1.140	1.106	60.01	9
		40	1.105	1.105	1.087	80.01	6
		60	1.111	1.099	1.062	91.29	8
		80	1.119	1.089	1.060	92.22	6
		100	1.121	1.091	1.070	97.64	6
$[1,100] \times [1,100]$	300×300	20	1.000	1.000	1.000	0.01	10
		40	1.400	1.400	1.400	0.01	10
		60	1.100	1.100	1.050	10.00	10
		80	1.000	1.000	1.000	0.01	10
		100	1.133	1.100	1.100	30.00	10

class is characterized by a different size of the bins and by the ranges in which the item sizes were uniformly randomly generated. The first three columns give the intervals in which the sizes of the items were uniformly randomly generated, the (fixed) bin sizes and the number of items. The values in each row are referred to 10 random problem instances. We give the average ratios of FFF, FBS and TS with respect to LB, and the average tabu search computing time (expressed in seconds). The last column ($\leqslant$B & B) gives the number of instances in which TS was at least equal to the solution value obtained by a branch-and-bound algorithm for the exact solution of the problem, presented in [7]; since the execution of such algorithm was interrupted after 300 CPU seconds, the solutions it provided were not necessarily optimal.

Table 3 examines new randomly generated instances described in [7], having a different mix of items. Four types of items were considered, each defined by different intervals in which the sizes were randomly generated (see Table 3). Each class is characterized by a different percentage of items generated for each type. The table gives, for each pair of size intervals, the percentage of items that were uniformly randomly generated in such interval (the bin size was always 100×100), the number of items, plus the same information as in Table 2. In this case too the values in each row are referred to 10 random problem instances.

The results of Tables 1–3 show a very good behavior of the proposed tabu search algorithm. The TS/LB ratios are considerably lower than the FFF/LB and FBS/LB ratios (remind that algorithms FFF and FBS are classical and effective methods from the literature). Also the comparison with the branch-and-bound approach is satisfactory. For 26 out of 36 instances from the literature the exact solution was obtained, while for 90% of the randomly generated instances of Tables 2 and 3 the tabu search approach attained the same performance, in terms of solution quality, as a branch-and-bound algorithm.

Table 3
Results on the random problem instances proposed by Martello and Vigo. Time limit of 100 CPU seconds on a Silicon Graphics INDY R4000sc

$h_j \in [1, \frac{H}{2}]$	$[\frac{2}{3}H, H]$	$[\frac{H}{2}, H]$	$[1, \frac{H}{2}]$	n	FFF/LB	FBS/LB	TS/LB	Time	$\leqslant$B & B
$w_j \in [\frac{2}{3}W, W]$	$[1, \frac{W}{2}]$	$[\frac{W}{2}, W]$	$[1, \frac{W}{2}]$						
70%	10%	10%	10%	20	1.098	1.098	1.062	30.09	9
				40	1.107	1.107	1.070	70.01	7
				60	1.077	1.077	1.044	64.91	9
				80	1.073	1.058	1.041	91.95	10
				100	1.045	1.041	1.030	73.79	9
10%	70%	10%	10%	20	1.173	1.157	1.078	40.01	9
				40	1.093	1.084	1.022	30.69	10
				60	1.062	1.062	1.025	46.72	10
				80	1.067	1.063	1.028	71.42	8
				100	1.062	1.062	1.030	83.75	10
10%	10%	70%	10%	20	1.014	1.007	1.000	0.01	10
				40	1.022	1.019	1.014	40.01	10
				60	1.018	1.016	1.009	40.04	10
				80	1.020	1.016	1.014	80.07	10
				100	1.018	1.015	1.009	50.03	10
10%	10%	10%	70%	20	1.137	1.137	1.137	50.01	7
				40	1.145	1.089	1.075	50.01	9
				60	1.146	1.119	1.084	80.17	9
				80	1.147	1.131	1.088	96.15	9
				100	1.137	1.104	1.073	100.09	10

Acknowledgements

This work was supported by Ministero dell'Università e della Ricerca Scientifica e Tecnologica, and by Consiglio Nazionale delle Ricerche, Italy. The first author gratefully acknowledges the support by the Department of Mathematics of the Swiss Federal Institute of Technology (Lausanne) and in particular by Prof. Th.M. Liebling. We thank an anonymous referee for several useful comments.

References

[1] H. Dyckhoff, A typology of cutting and packing problems, European Journal of Operational Research 44 (2) (1990) 145–159.
[2] F.K.R. Chung, M.R. Garey, D.S. Johnson, On packing two-dimensional bins, SIAM Journal on Algebraic and Discrete Methods 3 (1) (1982) 66–76.
[3] J.O. Berkey, P.Y. Wang, Two dimensional finite bin packing algorithms, Journal of Operational Research Society 38 (1987) 423–429.
[4] J.B. Frenk, G.G. Galambos, Hybrid next-fit algorithm for the two-dimensional rectangle bin-packing problem, Computing 39 (1987) 201–217.
[5] B.E. Bengtsson, Packing rectangular pieces – a heuristic approach, The Computer Journal 25 (1982) 353–357.
[6] A. El-Bouri, N. Popplewell, S. Balakrishnan, A. Alfa, A search based heuristic for the two-dimensional bin-packing problem, INFOR 32 (4) (1994) 265–274.
[7] S. Martello, D. Vigo, Exact solution of the two-dimensional finite bin packing problem, Management Science (to appear).
[8] H. Dyckhoff, U. Finke, Cutting and Packing in Production and Distribution, Physica Verlag, Heidelberg, 1992.
[9] K.A. Dowsland, W.B. Dowsland, Packing problems, European Journal of Operational Research 56 (1) (1992) 2–14.
[10] 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.
[11] D.S. Johnson, A. Demers, J.D. Ullman, M.R. Garey, R.L. Graham, Worst-case performance bounds for simple one-dimensional packing algorithms, SIAM Journal on Computing 3 (1974) 299–325.
[12] N. Christofides, C. Whitlock, An algorithm for two-dimensional cutting problems, Operations Research 25 (1977) 30–44.
[13] J.E. Beasley, Algorithms for unconstrained two-dimensional guillotine cutting, Journal of Operational Research Society 36 (1985) 297–306.
[14] J.E. Beasley, An exact two-dimensional non-guillotine cutting tree search procedure, Operations Research 33 (1985) 49–64.