Ann Oper Res (2010) 179: 203–220
DOI 10.1007/s10479-008-0449-4

A hybrid GRASP/VND algorithm for two- and three-dimensional bin packing

F. Parreño · R. Alvarez-Valdes · J.F. Oliveira · J.M. Tamarit

Published online: 25 October 2008

Abstract The three-dimensional bin packing problem consists of packing a set of boxes into the minimum number of bins. In this paper we propose a new GRASP algorithm for solving three-dimensional bin packing problems which can also be directly applied to the two-dimensional case. The constructive phase is based on a maximal-space heuristic developed for the container loading problem. In the improvement phase, several new moves are designed and combined in a VND structure. The resulting hybrid GRASP/VND algorithm is simple and quite fast and the extensive computational results on test instances from the literature show that the quality of the solutions is equal to or better than that obtained by the best existing heuristic procedures.

Keywords Cutting · Packing · Heuristic algorithms · GRASP · VND

1 Introduction

The three-dimensional bin packing problem (3BP) consists of determining the minimum number of three-dimensional rectangular containers (*bins*) into which a given set of n three-dimensional rectangular items (*boxes*) can be orthogonally packed without overlapping. All bins are of identical known dimensions (W, H, D) and each box i is of dimensions (w_i, h_i, d_i), $i = 1, \ldots, n$. We assume, without loss of generality, that all the input data are

F. Parreño
Department of Mathematics, University of Castilla-La Mancha, Albacete, Spain

R. Alvarez-Valdes (✉) · J.M. Tamarit
Department of Statistics and Operations Research, University of Valencia, Burjassot, Valencia, Spain
e-mail: ramon.alvarez@uv.es

J.F. Oliveira
Faculty of Engineering of the University of Porto, Porto, Portugal

J.F. Oliveira
INESC Porto—Instituto de Engenharia de Sistemas e Computadores do Porto, Porto, Portugal

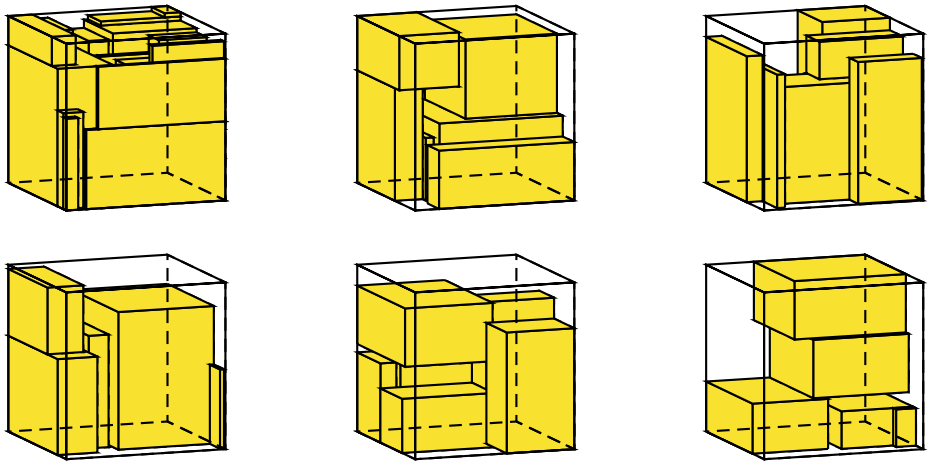


Fig. 1 Example class 7 instance 2 with 40 items

positive integers and that $w_i \leq W$, $h_i \leq H$ and $d_i \leq D$, $i = 1, \ldots, n$. It is also assumed that the orientation of the items is fixed, i.e. they cannot be rotated. We can see an instance in Fig. 1 in which 40 boxes are packed into 6 bins. The two-dimensional bin packing problem (2BP) addresses the same question for two-dimensional bins (W, H) and boxes (w_i, h_i) and can be considered a special case of 3BP when $d_i = D, \forall i = 1, \ldots, n$. According to the typology proposed by Wäscher et al. (2007), bin packing problems are classified as SBSBPP (Single Bin-Size Bin Packing Problems). The problem is strongly NP-hard as it generalizes the strongly NP-hard one-dimensional bin packing problem.

Three-dimensional packing problems are of relevant practical interest in industrial applications such as loading cargo into pallets, containers or vehicles, or packaging design. Although 3BP is a simplified version of real-world problems, in many cases it arises as a subproblem of other complex problems not only in cutting and packing but also in some scheduling problems.

An exact method for the 3BP was proposed by Martello et al. (2000). Their initial proposal only solved robot-packable problems (den Boef et al. 2005), but was later modified for solving the general problem (Martello et al. 2007). Fekete and Schepers (2004a), Fekete et al. (2007) proposed a general framework for the exact solution of multi-dimensional packing problems. A branch and bound algorithm for the two-dimensional case was also developed by Martello and Vigo (1998).

Lower bounds for the 3BP appeared in the above mentioned paper by Martello et al. (2000), and that by Fekete and Schepers (2004b). More recently, Boschetti (2004) has introduced new bounds which dominate previous ones. Boschetti and Mingozzi (2003a, 2003b) propose new lower bounds for the two-dimensional case.

Several constructive and metaheuristic algorithms have been designed for solving large bin packing problems. Faroe et al. (2003) proposed a Guided Local Search heuristic for 3BP and 2BP, based on the iterative solution of constraint satisfaction problems. Lodi et al. have developed tabu search algorithms based on new constructive procedures for two-dimensional (Lodi et al. 1999) and three-dimensional (Lodi et al. 2002) cases, and a unified tabu search code for general multi-dimensional bin packing problems (Lodi et al. 2004). More recently, Crainic et al. (2008) have developed a two-level tabu search algorithm, in which the first level aims to reduce the number of bins and the second optimizes the packing

of the bins, using the representation proposed by Fekete and Schepers (2004a), Fekete et al. (2007). This two-level scheme results in a more flexible structure and the computational results show their algorithm equals or outperforms previous approaches.

For the two-dimensional bin packing problem (2BP), Boschetti and Mingozzi (2003b) proposed an effective constructive heuristic that assigns a score to each object, considers the objects according to decreasing values of the corresponding scores, updates the scores using a specified criterion, and iterates until either an optimal solution is found or a maximum number of iterations has been performed. More recently, Monaci and Toth (2006) have designed a set-covering-based heuristic approach in which in a first phase a large number of columns are generated by heuristic procedures (including the Boschetti and Mingozzi algorithm (2003b)) and by a time-limited execution of the exact algorithm by Martello and Vigo (1998). In the second phase they use these columns, corresponding to efficient ways of filling one bin, for solving a set-covering problem which gives the solution to the original bin packing problem.

In this paper we propose a new GRASP (*Greedy Randomized Adaptive Search Procedure*) algorithm for solving two and three dimensional bin packing problems. The constructive phase is based on a heuristic we developed for the container loading problem (Parreño et al. 2008a). In the improvement phase, several new moves are designed, tested and finally combined in a VND (*Variable Neighborhood Descent*) structure (Hansen and Mladenovic 2005). The resulting hybrid GRASP/VND algorithm is simple and fast and the extensive computational results show the solutions to be equal to or better than those obtained by the best, more complex existing procedures. The paper is organized as follows. Section 2 describes some preprocessing techniques developed to reduce the dimensions of the problem. Section 3 presents the constructive and improvement phases of the algorithm, as well as a diversification procedure. In Sect. 4 the algorithm is tested on a large set of test instances from the literature and compared with the best reported heuristics. Finally, in Sect. 5 some conclusions are drawn.

2 Preprocess

We have designed a procedure to be applied before calling the heuristic algorithms in order to transform the original instance into an equivalent problem which can be more easily solved.

If there is a box j such that $w_j > W/2$, $h_j > H/2$ and $d_j > D/2$, we build the set $S_j = \{i \mid w_i \leq W - w_j \text{ or } h_i \leq H - h_j \text{ or } d_i \leq D - d_j\}$. The boxes in S_j are the only ones that can be packed into the same bin as box j. Then, if there is a feasible packing of boxes $\{j\} \cup S_j$, this packing optimally fills one bin and we can solve a reduced problem without these boxes and add this bin to the solution.

The existence of this packing is heuristically checked and therefore it may be that the packing exists and we are not able to find it. As this test is used to discard boxes from the problem resolution, the failure of the heuristic will just mean that we will solve a larger problem. First, we compute the total volume of box j, $v_j = w_j * h_j * d_j$, and the boxes in S_j, $V_j = \sum_{i \in S_j} w_i * h_i * d_i$. If $v_j + V_j \leq W * H * D$, we try to pack the boxes using the constructive algorithm which will be described in the next section. If $v_j + V_j > W * H * D$, the boxes do not fit into one bin and in principle the procedure cannot be applied. However, if there is a box $k \in S_j$ such that $w_k = \min_{i \in S_j}\{w_i\}$, $h_k = \min_{i \in S_j}\{h_i\}$, $d_k = \min_{i \in S_j}\{d_i\}$, that is, minimal in S_j in all dimensions, we consider the set $S_j \setminus k$. If $v_j + V_j - v_k \leq W * H * D$ and, by applying the heuristic procedure, we can pack box j and the boxes in $S_j \setminus k$ into one

bin, then this packing is still optimal (the bin containing box j cannot be filled in any better way). We can then solve a reduced problem with the remaining boxes, including the box k.

3 GRASP algorithm

The GRASP algorithm was developed by Feo and Resende (1989) to solve hard combinatorial problems. For an updated introduction, refer to Resende and Ribeiro (2003). GRASP is an iterative procedure combining a randomized constructive phase and an improvement phase. In the subsections we describe a constructive procedure, a randomization strategy which will be embedded in the constructive process, some movements for the improvement phase, and a diversification phase which can be included in the iterative structure.

3.1 The constructive phase

The constructive procedure follows an iterative process in which at the beginning of each iteration we have a list of boxes still to be packed $\mathcal{B}$ and a new empty bin E. Initially, this list contains the complete list of boxes and, as the bins are filled, it is reduced until all the boxes have been packed.

At each iteration, given the list and dimensions of the remaining boxes and the dimensions of the bin, we apply a constructive algorithm we originally designed for solving the Container Loading Problem (Parreño et al. 2008a, 2008b). Each application of the constructive algorithm fills one bin and the procedure is used until all the boxes are packed. The details of the algorithm can be found in the references and we will only outline its main steps here.

1. Step 0: *Initialization*
   $\mathcal{S}$={E}, the set of empty maximal spaces.
   $\mathcal{B} = \{b_1, b_2, \ldots, b_m\}$, the set of boxes still to be packed.
2. Step 1: *Choosing the maximal space in* $\mathcal{S}$
   We have a list $\mathcal{S}$ of empty maximal spaces. These spaces are called maximal because at each step they are the largest empty parallelepiped available to be filled with rectangular boxes. We can see an example for a 2D problem in Fig. 2. Initially we have an empty rectangle and when we pack a piece into its bottom left corner, two maximal spaces are generated. These spaces do not have to be disjoint. In the same Fig. 2 we see one more step of the filling process with the maximal spaces generated.

   At each step we take the maximal space with the minimum distance to a corner of the bin and use the volume of the space as a tie-breaker. The corner of the maximal space with the lowest distance to a corner of the bin will be the corner into which the boxes will be packed. The reason behind that decision is to first fill the corners of the bin, then its sides and finally the inner space.
3. Step 2: *Choosing the boxes to pack*
   Once a maximal space S^* has been chosen, we consider the boxes i of $\mathcal{B}$ fitting into S^* in order to choose which one to pack. If there are several boxes with the same dimensions, we consider the possibility of packing a layer, that is, packing several of these boxes arranged in rows and columns.

   Two criteria have been considered to select the configuration of boxes:

   (i) *Best-Volume*:
   The box or layer producing the largest increase in the volume occupied by boxes.

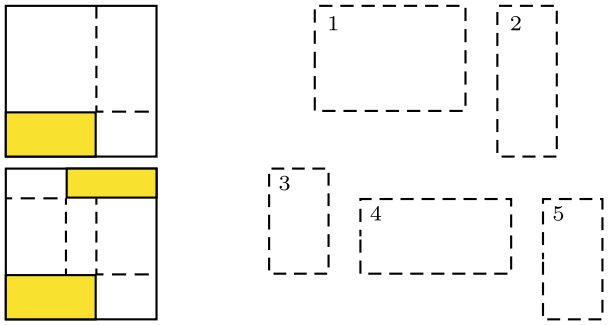


Fig. 2 Maximal spaces in two dimensions

(ii) *Best-Fit*:
The box or layer which fits best into the maximal space. We compute the distance from each side of the layer to each side of the maximal space and order these distances in a vector in non-decreasing order. The box or layer is chosen using the lexicographical order.

4. Step 3: *Updating the list* $\mathcal{S}$
Unless the layer fits exactly in space S^*, packing it produces new empty maximal spaces, which will replace S^* on the list $\mathcal{S}$. Moreover, as the maximal spaces are not disjoint, the layer being packed can intersect with other maximal spaces which will have to be reduced. Therefore, we have to update the list $\mathcal{S}$. Once the new spaces have been added and some of the existing ones modified, we check the list and eliminate possible inclusions. Figure 2 shows this process. When the second box is packed in maximal space 1, the space into which it is packed is eliminated from the list and is replaced by two new spaces 3 and 4. Maximal space 2 is also affected by the packing thereby defining a new, reduced, maximal space 5.

The list $\mathcal{B}$ is also updated and the maximal spaces that cannot accommodate any of the boxes still to be packed are eliminated from $\mathcal{S}$. If $\mathcal{S} = \emptyset$ or $\mathcal{B} = \emptyset$, the procedure ends. Otherwise, it goes back to Step 1.

In the deterministic procedure described above, we have introduced a randomization strategy at Step 2 when selecting the type of box and the configuration to pack. We consider all feasible configurations of all types of boxes fitting into S and evaluate them according to the objective function (Best-Volume or Best-Fit). The configuration is selected at random from among a restricted set of candidates composed of the best $100\delta\%$ of the blocks, where $0 \leq \delta \leq 1$ is a parameter to be determined.

It is difficult to determine the value of δ that gives the best average results. The principle of reactive GRASP, proposed for the first time by Prais and Ribeiro (2000), is to let the algorithm find the best value of δ in a small set of allowed values. The parameter δ is initially taken at random from a set of discrete values $\{0.1, \ldots, 0.8, 0.9\}$, but after a certain number of iterations the relative quality of the solutions obtained with each value of δ is taken into account and the probability of values consistently producing better solutions is increased.

At every iteration of the GRASP algorithm, except the first one, the constructive phase has a target, a maximum number of bins to use, which is set to the best current solution minus one. If the first solution requires n bins to pack all the boxes, in the subsequent iterations the constructive phase will use at most $n-1$ bins. If the boxes do not fit into $n-1$ bins, the solution will consist of $n-1$ lists of boxes packed into the $n-1$ available bins plus a list of unpacked boxes. If at a given iteration all the boxes are packed into the available bins, the target is decreased by one unit for the following iterations.

3.2 Improvement phase

Each solution built at the constructive phase is the starting point for a procedure in which we try to improve the solution. Before applying any move we order the bins in the solution by non-increasing occupied volume. In this way, the last bins in the solution will have more empty spaces and will be the more likely candidates to be involved in improving moves. Inside each bin, the order of the boxes is the order in which they were inserted into the solution. We have studied several alternatives:

- The first procedure starts by eliminating the last $k\%$ items in the solution (for instance, the last 50%), in the order described above. These items correspond to the less occupied bins. We choose the value k at random from $[30, 90]$. The removed items plus the unpacked items are then packed again using the deterministic constructive procedure. In this call of the deterministic algorithm we can use the objective function *Best-Volume* or the objective function *Best-Fit*. Both alternatives have been tested and their results will appear in the next section.

  In Fig. 3 we can see an example with seventeen boxes. At a given iteration four bins are being used in the constructive phase and one box is left unpacked. First, the last five items are removed. Then the six currently unpacked boxes are considered for packing using the deterministic constructive procedure. In this case all items are packed and a feasible solution with only four bins is found. Therefore, in the next iterations of the GRASP algorithm the target will be to use only three bins.
- A second procedure consists of considering the bins whose occupied volume is lower than the average occupancy and removing the last $k\%$ pieces packed from each one of them. The partially emptied bins are then filled again using the deterministic constructive algorithm (with the Best-Volume or Best-Fit objective function).
- In a third procedure for each bin whose occupancy is below average the bin is split into two parts, randomly selecting whether the cut is to be horizontal or vertical, and one of the two resulting sides of the bin (upper/lower, left/right), again randomly selected, is emptied. That involves removing pieces whose volume is mostly included in the part

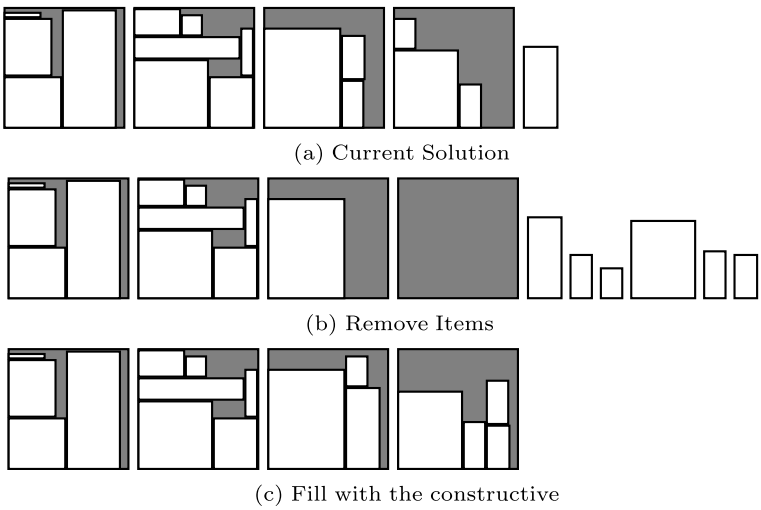
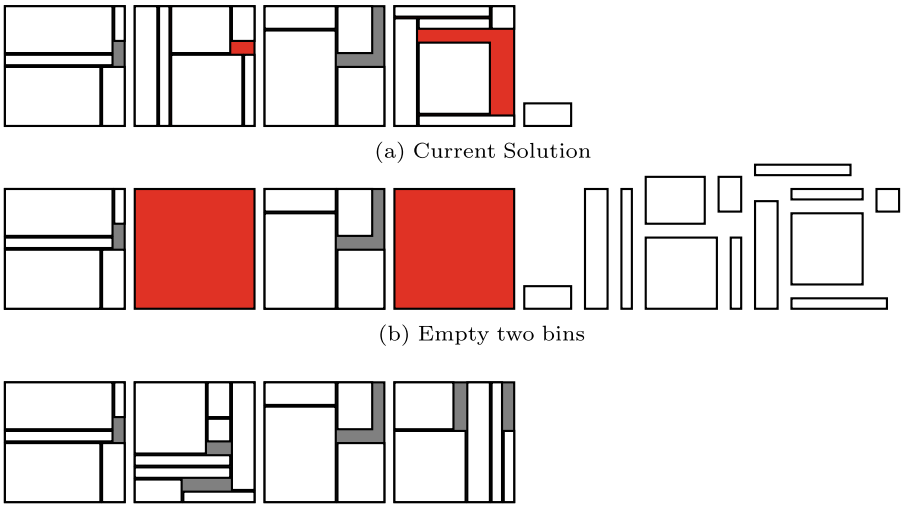


Fig. 3 Instance 10 21, first move



(a) Current Solution

(b) Empty two bins

(c) Pack first the unpacked item and then the removed boxes

Fig. 4 Instance 1 17, fourth move

being emptied. The partially emptied bins are filled using the deterministic constructive algorithm (with the Best-Volume or Best-Fit objective function).

- The fourth procedure consists of a local search which is only called if we have more than two bins in the current solution. The move consists of removing all the boxes from two bins in the current solution and completing the partial solution, with the added condition that the first item packed in the first bin is one of the unpacked items. This way we pack items which were previously difficult to pack. The neighborhood consists of all pairs of bins in which at least one of them has an occupancy which is below the average.

  We have proved two variants, one in which as soon as we find an improving move we make it (First-improve) and another in which we examine the neighborhood completely before making a move (Best-improve). A preliminary computational experiment shows that the complete exploration of the neighborhood produces better results.

  An example of this kind of improvement can be seen in Fig. 4, in which we remove the boxes in bins 2 and 4. We pack the only box which was not included in any of the bins of the solution in the first empty bin, and then the remaining boxes with the constructive algorithm.

A move is considered to improve the solution if the total volume of boxes packed into the bins is increased.

The improvement phase is only called if the solution of the constructive phase is considered to be promising, that is, if it is considered a good starting point to improve the best known solution. More specifically, we try to improve a solution if the volume of packed boxes $V \geq V_{total} - (V_{total}/(nBest_{bins} - 1))$, where V_{total} is the total volume of the boxes and $nBest_{bins}$ corresponds to the minimum number of bins used.

In summary, the first move, in which the last $k\%$ boxes in the solution are removed, has been chosen because it has worked well in the Container Loading Problem (Parreño et al. 2008a). The second move is a variation of the first one in which the percentage of boxes to be removed is applied to every bin of those below average. The third move is an aggressive variation of the second, allowing different parts of the bins to be emptied. Finally, the fourth

move looks for pairs of complementary bins which can be best filled by combining the boxes initially contained in both of them, plus the items outside the solution, in a different way.

Having defined four improvement moves, we can choose just one of them, according to certain computational experiments, or combine them somehow to produce a more powerful improvement phase. In the next section a computational comparison of the effect of each individual move with both objective functions is presented. We have also designed three possible combination strategies:

1. *All moves*:
   Each move, in the order in which they have been described, is applied to the solution while it improves the current solution.
2. *VND* (*Variable Neighborhood Descent*):
   Define the neighborhoods N_1 to N_4 corresponding to the four moves and apply the *VND* scheme as described in Hansen and Mladenovic (2005). Note that given a feasible solution x, its neighborhoods N_1, N_2 and N_3 only contain one solution, so we are considering quite a special *VND* algorithm, but the basic scheme can be applied to this case:
   Initialization: Select the set of neighborhood structures N_p, for $p = 1, \ldots, 4$, and start from the initial solution x obtained by the constructive algorithm.
   Repeat the following sequence until no improvement is obtained:
   (1) Set $p \leftarrow 1$
   (2) Repeat the following steps while $p <= 4$
      (a) Exploration of neighborhood.
         Find the best neighbor x' of $x (x' \in N_p(x))$;
      (b) Move or not.
         If the solution x' obtained is better than x, set $x \leftarrow x'$ and $p \leftarrow 1$.
         Otherwise, set $p \leftarrow p + 1$;
3. VND_{seq} (*Sequential Variable Neighborhood Descent*):
   In VND_{seq} the neighborhoods are used in the same order as in *VND*, but in the Local Search when a solution is improved using the p^{th} neighborhood, instead of setting $p = 1$ and going back to the first neighborhood, the algorithm proceeds sequentially to the $(p + 1)^{\text{th}}$.

3.3 Diversification phase

Throughout the search we store a vector with the frequency with which each item is not packed into the available bins. We can use this information to diversify the search. After a given number of iterations without improving the best known solution, we perform a certain number of diversification iterations in which the constructive process is modified. Every time we start packing a new bin, we pack into it the unpacked piece with the largest value in the frequency vector and then go on with the usual filling process. In this way, the pieces which are more difficult to pack and are left unpacked most of the time go into a bin for at least a given number of iterations. A preliminary study showed that putting more than one of these difficult pieces into the same bin consistently produced bad solutions and therefore we put just one of these pieces into every new bin.

Whenever the solution is improved and the total number of bins is reduced, the frequency vector is initialized.

4 Computational experiments

In this section we will present the results of the computational experiments. Our goal is to show that the new algorithm on average produces similar or better solutions than all the other known algorithms for the two- and three-dimensional bin-packing problem. We also show that the high quality solutions for medium-large instances can be constructed in seconds on a standard laptop.

The above algorithm was coded in C++ and run on a Pentium Mobile at 1500 MHz with 512 Mbytes of RAM. In order to assess the relative efficiency of our algorithm we have compared it with the guided local search algorithm by Faroe et al. (2003), the tabu search procedures by Lodi et al. for two-dimensional (1999) and three-dimensional cases (2002), the heuristic proposed by Boschetti and Mingozzi (2003b) and the recent set-covering algorithm by Monaci and Toth (2006) and TS^2PACK by Crainic et al. (2008).

4.1 3D instances

For three-dimensional bin packing, the standard benchmark is the set of 320 problems generated by Martello et al. (2000). The instance generator is available at http://www.diku.dk/~pisinger/codes.html. These instances have already been used by Faroe et al. (2003) (GLS) and by Lodi et al. (2002) (TS3).

The instances are organized into 8 classes with 40 instances each, 10 instances for each value of $n \in \{50, 100, 150, 200\}$.

4.2 2D instances

For two-dimensional bin packing there are several sets of commonly used instances. First, we consider those generated by Berkey and Wang (1987) and by Martello and Vigo (1998). Each class is composed of 50 instances, 10 instances for each value of $n \in \{20, 40, 60, 80, 100\}$, so a set of 500 instances has been considered. As for the lower bounds, we used those proposed by Martello and Vigo (1998) and by Boschetti and Mingozzi (2003a). All instances, and the corresponding best known solution values, are available at http://www.or.deis.unibo.it/research_pages/ORinstances/ORinstances.htm. Finally, we consider other instances in the literature: instances *cgcut*, proposed by Christofides and Whitlock (1977), *gcut* and *ngcut*, proposed by Beasley (1985a, 1985b), all available in the ORLIB library, http://ww.ms.ic.ac.uk/info.html, see Beasley (1990). These instances are two-dimensional cutting problems that were transformed into 2D-BPP as explained by Faroe et al. (2003). We also used the *beng* set, proposed by Bengtsson (1982), available in the PackLib^2 (Fekete and Van der Veen 2007), http://mo.math.nat.tu-bs.de/packlib/index.html. These classes have been used by Faroe et al. (2003) (GLS), Lodi et al. (1999) (TS3), Boschetti and Mingozzi (2003b) (HBP) and Monaci and Toth (2006) (SCH).

4.3 Choosing the best strategies

Table 1 presents the results obtained by the constructive algorithm in Sect. 2. At Step 2 of this algorithm, when selecting the new maximal space into which to pack new boxes, the original procedure chose the space closest to one of the eight corners in the three-dimensional container. Using this strategy, the container was first filled at the corners, then at the sides and finally at the center. We compared this strategy with some alternatives in which only the four corners at the base of the container were used to choose the space and the results were

Table 1 Results of the constructive algorithm according to selection of the corner

		LB	Fixed by preproc.	One corner	Two corners			Four corners	Eight corners
					X	*Y*	*Z*		
2D class	1	993	34	1008	1010	1011		1013	
	2	124	0	129	129	129		130	
	3	687	1	725	727	728		740	
	4	119	0	127	127	128		126	
	5	883	15	918	917	919		915	
	6	108	0	116	116	116		116	
	7	813	5	853	844	857		855	
	8	826	1	858	861	850		863	
	9	2130	222	2138	2138	2138		2138	
	10	490	6	523	524	517		523	
2D literature		230	3	240	241	240		241	
Overall 2D		7403	287	7635	7634	7633		7660	
3D class	1	1207	1	1330	1340	1327	1323	1347	1355
	2	1194	1	1312	1308	1325	1303	1325	1337
	3	1208	1	1322	1318	1318	1333	1316	1341
	4	2878	8	2960	2960	2960	2960	2960	2960
	5	620	1	746	755	746	744	757	782
	6	881	4	988	992	995	991	1004	1013
	7	519	0	628	632	628	625	639	660
	8	735	1	862	866	862	866	873	893
Overall 3D		9242	17	10148	10171	10161	10145	10221	10341
Overall		**16645**	**304**	**17783**	**17805**	**17794**		**17881**	

better if the eight corners were considered. However, it is not clear whether a strategy working well for CLP instances, in which the container is very large relative to the dimensions of the boxes, will also work well for bin packing problems in which the boxes are much bigger compared with the size of the bins. Therefore, our first comparison involved the use of the constructive algorithm using as a reference for choosing the maximal space only one corner, the origin, two corners, considering different sides of the bin, X, Y, Z, the four corners of the floor of the bin and finally the eight corners of the bin. Obviously, for two-dimensional problems, this last option does not apply.

Table 1 shows the sum of bins required for each class of problems. For the two-dimensional case, the table includes the aggregate results of the 10 classes generated by Berkey and Wang (1987) and by Martello and Vigo (1998) and also the other sets of instances grouped under the heading of *Literature*. For the three-dimensional case, the results obtained for the eight classes generated by Martello et al. (2000) are reported.

As a reference, Table 1 also includes for each type of problem the best reported lower bounds, which will be described in more detail in the following subsections, and the number of boxes fixed in the preprocess described in Sect. 2.

Table 2 Individual improvement methods

		Constr.	Random. constr.	Best-Volume 1	2	3	Best-Fit 1	2	3	Best-Volume 4	Best-Fit 4
2D class	1	1008	997	997	998	997	997	997	997	997	997
	2	129	124	124	124	124	124	124	124	124	124
	3	725	700	700	700	701	700	698	700	698	696
	4	127	123	123	123	124	123	123	124	123	123
	5	918	899	899	899	899	899	898	898	894	895
	6	116	112	112	112	112	112	112	112	112	111
	7	853	831	831	831	831	831	829	831	828	828
	8	858	836	836	835	836	836	836	838	835	835
	9	2138	2130	2130	2130	2130	2130	2130	2130	2130	2130
	10	523	509	509	509	509	509	506	509	507	506
2D literature		240	231	231	231	231	231	231	231	230	231
Overall 2D		7635	7492	7492	7492	7494	7492	7484	7494	7478	7476
3D class	1	1330	1281	1281	1281	1281	1281	1281	1283	1273	1276
	2	1312	1269	1269	1268	1267	1269	1264	1267	1258	1260
	3	1322	1278	1277	1275	1277	1278	1274	1276	1271	1271
	4	2960	2941	2941	2941	2942	2941	2941	2941	2941	2941
	5	746	719	719	718	719	719	718	718	707	709
	6	988	961	961	960	960	961	959	959	957	955
	7	628	599	599	595	599	599	598	597	592	593
	8	862	829	829	829	830	829	830	830	821	823
Overall 3D		10148	9877	9876	9867	9875	9877	9865	9871	9820	9828
Overall		**17783**	**17369**	**17368**	**17359**	**17369**	**17369**	**17349**	**17365**	**17298**	**17304**

In order to decide if the differences observed in the table can be considered statistically significant, we have applied the Friedman non-parametric test to compare the six alternatives and the Wilcoxon test for pairwise comparisons (Gibbons and Chakraborti 2003). For the two-dimensional case, the four-corners strategy is significantly worse than the others, which in turn can be considered to produce similar results. For the three-dimensional case, the first four alternatives, involving one or two corners, are again significantly better than the others, with eight-corners performing worse than four-corners. From this analysis, the clear conclusion is that any of the strategies using one or two corners is a good choice for the constructive algorithm. An alternative is to choose one of them. Another possibility would be to use all four of them, selecting one of them at each iteration in a deterministic or random way.

In Table 2 we compare the four improvement methods, used independently, with both objective functions. In all cases, the GRASP algorithm is run for 50000 iterations and at each iteration one of the four good strategies for choosing the corners is randomly selected. Columns 3 and 4 contain the results corresponding to the constructive deterministic algorithm and the constructive randomized algorithm, without an improvement phase. We have again used the Friedman test to jointly compare several alternatives and the Wilcoxon test

Table 3 Combined improvement methods

		All moves	*VND$_{seq}$*	*VND*	*VND$_{seq}$ +Divers.*	*VND +Divers.*
Class	1	997	997	997	997	997
	2	124	124	124	124	124
	3	696	696	697	696	696
	4	123	123	123	122	122
	5	895	893	893	893	893
	6	112	111	112	111	111
	7	828	827	828	827	827
	8	835	835	835	835	835
	9	2130	2130	2130	2130	2130
	10	504	503	503	503	503
Literature		*230*	*230*	*230*	*230*	*230*
Overall 2D		*7474*	*7469*	*7472*	*7468*	*7468*
3D	1	1275	1273	1274	1273	1273
	2	1259	1258	1258	1258	1258
	3	1271	1270	1270	1270	1270
	4	2940	2940	2940	2940	2940
	5	707	707	706	706	706
	6	954	953	953	953	953
	7	592	593	590	593	593
	8	819	818	816	818	818
Overall 3D		*9817*	*9812*	*9807*	*9811*	*9811*
Overall		**17291**	**17281**	**17279**	**17279**	**17279**

for pairwise comparisons. For two-dimensional problems the statistical analysis shows that moves 1 and 3 do not improve the randomized constructive procedure, while moves 2 and 4 improve it significantly. Among these two moves, there is a significant difference in favor of move 4, while the objective function used does not produce significant differences. In summary, for two-dimensional problems, move 4 with any of the objective functions is the best option, followed by move 2 with Best-Fit objective function.

For the harder three-dimensional problems, the results are quite similar but the differences between the strategies are larger. Moves 1 and 3 do not improve the randomized constructive algorithm, while move 2, either with Best-Volume or Best-Fit, improves it significantly. However, move 4 with any objective function is again significantly better than move 2.

If we had to choose just one move for the implementation of the algorithm, these considerations for the quality of the solution would have to be balanced with their CPU time requirements. However, as all these moves are quite fast and we are looking for high-quality solutions, instead of choosing one move, we have explored different ways of combining them.

Table 3 presents the results of the three strategies for the combination of improvement moves. Columns 6 and 7 also include a diversification phase. This phase is not called until half the total number of iterations or the total running time is reached. From this moment,

every time 500 iterations are spent without improving the best known solution, the diversification constructive process is used for 100 iterations. For two-dimensional problems, the results of the three strategies are quite similar, though the statistical tests show that the strategy *All moves* performs significantly worse than VND_{seq} and the two alternatives including diversification, while these three alternatives do not differ significantly. For three-dimensional problems, significant differences are observed between the strategy *All moves* and all the others, but these four alternatives do not differ among themselves. Therefore, the statistical analyses do not provide a clear indication about which strategy to use. We decided to include the strategy $VND_{seq} + Diversification$ in the final implementation of our algorithm because we think that it is convenient to include a procedure which explicitly takes into account the existence of awkward pieces, pieces that due to their size or shape are usually left out of good configurations, thereby producing an increase in the final complete solution. This situation has been explicitly considered by Bischoff et al. (1995) in their work for the multiple pallet problem. Our diversification strategy does not help to improve the solutions of the test problems, but could be useful in some other cases.

4.4 Computational comparisons with other algorithms

The algorithm of Faroe et al. (2003) was run on a Digital 500 au workstation with a 500 MHz 21164 CPU (SPECint2000 value of 161) and the Lodi et al. (1999, 2002) and Monaci and Toth (2006) algorithms were run on a Digital Alpha 533 MHz, a similar computer to the one used by Faroe et al. The algorithms were run with a time limit of 1000 seconds for each instance of 3BP and 100 seconds for each instance of 2BP. Our computer has a SPECint2000 value of approximately 1000 and can therefore be considered about seven times faster than the other computers. Hence we set a computational time limit at 150 seconds for the 3BP and 15 seconds for instances of 2BP, or a maximum number of 50000 iterations in all cases. The recent TS^2PACK by Crainic et al. (2008) was run on a Pentium IV 2000 MHz, but the authors made their computing times equivalent to those of the Digital 500 workstation and therefore the same comparisons can be applied.

Table 4 shows the computational results corresponding to three-dimensional test instances, comparing the solutions obtained by our GRASP algorithm with the GLS algorithm by Faroe et al. (2003), the TS3 algorithm by Lodi et al. (2002) and the TS^2PACK by Crainic et al. (2008). Each row of the table gives the average over the 10 instances for each class and number of boxes. Column 4 shows L_2, the corresponding lower bound obtained by Martello et al. (2000). Columns 5 to 7 contain details of our GRASP algorithm: the iteration in which the best solution was obtained, the total running time and the best solution found. The last four rows show aggregate results, first for Classes 1, 4, 5, 6, 7, 8 which allow the comparison with GLS (Faroe et al. 2003) and TS^2PACK (Crainic et al. 2008), and then for all classes for a complete comparison with TS3 (Lodi et al. 2002). Finally, Column 11 shows the number of times our solution matches the corresponding lower bound.

The results show that our GRASP algorithm consistently equals or outperforms algorithm TS3 for all subclasses. GRASP also gets equal or better results than GLS for all subclasses, except some subclasses of Class 7 and one subclass of Class 8. When comparing GRASP with TS^2PACK, our algorithm gets better results in 8 cases while TS^2PACK gets better results in 5, with 11 ties. The aggregate results favor GRASP against the other three procedures. If we apply statistical analysis to the results, the Friedman test shows a very significant difference between the four algorithms. This difference appears in the pairs GRASP-TS3 and TS^2PACK-TS3, but the differences between GRASP and GLS, GLS and TS3, GRASP and TS^2PACK, GLS and TS^2PACK are not significant (*p-values* of 0.105, 0.058, 0.202 and 0.357 respectively in the Wilcoxon test).

Table 4 Results for the three-dimensional instances[a]

Class	Bin size	Averages of ten instances for each class and size								
		n	L_2	GRASP			GLS	TS3	TS^2PACK	Opt.
				Best iter	Time	Sol				
1	$100 \times 100 \times 100$	50	12.5	3.8	47.2	**13.4**	**13.4**	**13.4**	**13.4**	3
		100	25.1	48.8	98.4	**26.6**	26.7	**26.6**	26.7	1
		150	34.7	1476.6	118.3	**36.4**	37.0	36.7	37.0	0
		200	48.4	77.8	143.3	**50.9**	51.2	51.2	51.1	0
2	$100 \times 100 \times 100$	50	12.7	44.3	82.1	**13.8**	–	**13.8**	–	1
		100	24.1	9.1	120.2	**25.7**	–	**25.7**	–	0
		150	35.1	360.6	113.5	**36.9**	–	37.2	–	1
		200	47.5	838.4	138.3	**49.4**	–	50.1	–	0
3	$100 \times 100 \times 100$	50	12.3	298.3	71.3	**13.3**	–	**13.3**	–	3
		100	24.7	77.6	78.4	**26.0**	–	**26.0**	–	2
		150	36.0	29.4	121.4	**37.6**	–	37.7	–	0
		200	47.8	2070.9	147.3	**50.0**	–	50.5	–	0
4	$100 \times 100 \times 100$	50	28.7	0.2	77.4	**29.4**	**29.4**	**29.4**	**29.4**	4
		100	57.6	0.4	135.0	59.0	59.0	59.0	**58.9**	1
		150	85.2	1.2	135.1	**86.8**	**86.8**	**86.8**	**86.8**	1
		200	116.3	3.0	135.1	**118.8**	119.0	**118.8**	**118.8**	1
5	$100 \times 100 \times 100$	50	7.3	152.8	102.2	**8.3**	**8.3**	8.4	**8.3**	2
		100	12.9	115.8	141.1	**15.0**	15.1	**15.0**	15.2	0
		150	17.4	472.3	144.5	**20.1**	20.2	20.4	**20.1**	0
		200	24.4	1242.4	150.0	**27.1**	27.2	27.6	27.4	0
6	$10 \times 10 \times 10$	50	8.7	22.8	95.7	**9.8**	**9.8**	9.9	**9.8**	1
		100	17.5	111.6	126.9	**19.0**	19.1	19.1	19.1	0
		150	26.9	120.9	134.6	**29.2**	29.4	29.4	**29.2**	0
		200	35.0	1684.6	150.0	**37.4**	37.7	37.7	37.7	0
7	$40 \times 40 \times 40$	50	6.3	1.7	142.9	**7.4**	**7.4**	7.5	**7.4**	0
		100	10.9	15.9	150.0	12.5	**12.3**	12.5	**12.3**	0
		150	13.7	33.6	150.0	16.0	**15.8**	16.1	**15.8**	0
		200	21.0	179.7	150.0	**23.5**	**23.5**	23.9	**23.5**	0
8	$100 \times 100 \times 100$	50	8.0	100.8	140.1	**9.2**	**9.2**	9.3	**9.2**	0
		100	17.5	13.2	121.7	18.9	18.9	18.9	**18.8**	1
		150	21.3	118.4	118.6	24.1	**23.9**	24.1	**23.9**	1
		200	26.7	201.3	150.0	**29.8**	29.9	30.3	30.0	0
Average classes 1, 4–8			28.50	258.32	127.42	**30.36**	30.43	30.50	30.41	
Total classes 1, 4–8			6840			**7286**	7302	7320	7298	16
Average all classes			28.88	310.26	122.83	**30.67**		30.82		
Total all classes			9242			**9813**		9863		23

[a]The best values appear in bold

Table 5 Results for the two-dimensional instances. Part I[a]

Class	Bin size	Averages of ten instances for each class and size									
		n	*LB**	GRASP			SCH	GLS	TS3	HBP	Opt.
				Best iter	Time	Sol					
1	10 × 10	20	7.1	0.5	0.0	**7.1**	**7.1**	**7.1**	**7.1**	**7.1**	10
		40	13.4	1.7	0.0	**13.4**	**13.4**	**13.4**	13.5	**13.4**	10
		60	19.7	3.8	4.5	**20.0**	**20.0**	20.1	20.1	20.1	7
		80	27.4	1.2	1.5	**27.5**	**27.5**	**27.5**	28.2	**27.5**	9
		100	31.7	6.6	0.0	**31.7**	**31.7**	32.1	32.6	31.8	10
2	30 × 30	20	1.0	0.0	0.0	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	10
		40	1.9	11.9	0.0	**1.9**	**1.9**	**1.9**	2.0	**1.9**	10
		60	2.5	1.6	0.0	**2.5**	**2.5**	**2.5**	2.7	**2.5**	10
		80	3.1	1.8	0.0	**3.1**	**3.1**	**3.1**	3.3	**3.1**	10
		100	3.9	0.4	0.0	**3.9**	**3.9**	**3.9**	4.0	**3.9**	10
3	40 × 40	20	5.1	1.3	0.0	**5.1**	**5.1**	**5.1**	5.5	**5.1**	10
		40	9.2	5.2	3.0	**9.4**	**9.4**	**9.4**	9.7	9.5	8
		60	13.6	29.8	4.6	**13.9**	**13.9**	14.0	14.0	14.0	7
		80	18.7	190.9	4.1	**18.9**	**18.9**	19.1	19.8	19.1	8
		100	22.1	243.1	4.9	**22.3**	**22.3**	22.6	23.6	22.6	8
4	100 × 100	20	1.0	0.0	0.0	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	10
		40	1.9	0.1	0.0	**1.9**	**1.9**	**1.9**	**1.9**	**1.9**	10
		60	2.3	0.3	3.0	**2.5**	**2.5**	**2.5**	2.6	**2.5**	8
		80	3.0	118.7	1.9	**3.1**	3.2	3.3	3.3	3.3	8
		100	3.7	0.4	1.5	**3.8**	**3.8**	**3.8**	4.0	**3.8**	9
5	100 × 100	20	6.5	0.4	0.0	**6.5**	**6.5**	**6.5**	6.6	**6.5**	10
		40	11.9	2.2	0.0	**11.9**	**11.9**	**11.9**	**11.9**	**11.9**	10
		60	17.9	11.0	1.5	**18.0**	**18.0**	18.1	18.2	**18.0**	9
		80	24.1	2.8	9.0	**24.7**	**24.7**	24.9	25.1	24.8	4
		100	27.9	194.7	5.2	**28.2**	**28.2**	28.8	29.5	28.7	7
6	300 × 300	20	1.0	0.0	0.0	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	10
		40	1.5	27.9	3.0	**1.7**	**1.7**	1.8	1.9	1.8	8
		60	2.1	48.0	0.1	**2.1**	**2.1**	2.2	2.2	**2.1**	10
		80	3.0	0.0	0.0	**3.0**	**3.0**	**3.0**	**3.0**	**3.0**	10
		100	3.2	0.4	3.0	**3.4**	**3.4**	**3.4**	**3.4**	**3.4**	8

Table 5 presents the results obtained for the two-dimensional test instances. It compares the solutions of our GRASP algorithm against the SCH algorithm by Monaci and Toth (2006), the GLS algorithm by Faroe et al. (2003), the TS3 algorithm by Lodi et al. (1999) and the HBP algorithm by Boschetti and Mingozzi (2003b). Again, each row shows the averages over 10 instances for each class and box count. Column LB^* indicates the lower bound reported by Monaci and Toth (2006), computed by applying all the lower-bounding procedures from the literature and an exact algorithm for a long computing time. Columns 5 to 7 contain details of our GRASP algorithm: the iteration in which the best solution was

Table 5 (*Continued*)

Class	Bin size	Averages of ten instances for each class and size									
		n	*LB**	GRASP			SCH	GLS	TS3	HBP	Opt.
				Best iter	Time	Sol					
7	100 × 100	20	5.5	0.8	0.0	**5.5**	**5.5**	**5.5**	**5.5**	**5.5**	10
		40	10.9	20.3	3.0	**11.1**	**11.1**	11.3	11.4	**11.1**	8
		60	15.6	4.2	4.5	15.9	**15.8**	15.9	16.2	16.0	8
		80	22.4	1.1	12.0	**23.2**	**23.2**	**23.2**	**23.2**	**23.2**	2
		100	26.9	6.8	3.1	**27.1**	**27.1**	27.5	27.7	27.4	8
8	100 × 100	20	5.8	0.5	0.0	**5.8**	**5.8**	**5.8**	**5.8**	**5.8**	10
		40	11.2	3.0	1.5	**11.3**	**11.3**	11.4	11.4	**11.3**	9
		60	15.9	256.7	4.2	**16.1**	16.2	16.3	16.2	16.2	7
		80	22.3	5.8	1.6	**22.4**	**22.4**	22.5	22.6	22.6	9
		100	27.4	5.6	6.1	**27.8**	27.9	28.1	28.4	28.0	5
9	100 × 100	20	14.3	0.0	0.0	**14.3**	**14.3**	**14.3**	**14.3**	**14.3**	10
		40	27.8	0.3	0.0	**27.8**	**27.8**	**27.8**	**27.8**	**27.8**	10
		60	43.7	6.9	0.1	**43.7**	**43.7**	**43.7**	43.8	**43.7**	10
		80	57.7	0.1	0.0	**57.7**	**57.7**	**57.7**	**57.7**	**57.7**	10
		100	69.5	0.2	0.0	**69.5**	**69.5**	**69.5**	**69.5**	**69.5**	10
10	100 × 100	20	4.2	33.9	0.0	**4.2**	**4.2**	**4.2**	4.3	4.3	10
		40	7.4	1.6	0.0	**7.4**	**7.4**	**7.4**	7.5	**7.4**	10
		60	9.8	362.4	4.5	**10.0**	10.1	10.2	10.4	10.2	7
		80	12.3	59.9	9.4	12.9	**12.8**	13.0	13.0	13.0	5
		100	15.3	28.3	9.2	**15.9**	**15.9**	16.2	16.6	16.2	4
Average			14.35	34.10	2.21	**14.48**	14.49	14.57	14.72	14.55	
Total			7173			**7241**	7243	7284	7360	7275	430

[a]The best values appear in bold

obtained, the total running time and the best solution found. Finally, column 11 shows the number of times the GRASP algorithm solution matches the lower bound, proving its optimality.

Finally Table 6, with the same structure as Table 5, compares the results of these algorithms on a set of two-dimensional instances which are well-known in the literature. For these problems, no differences are observed, except for one instance of Bengtsson (1982) in which GRASP improves the SCH solution.

The results show that for these two-dimensional instances the new GRASP algorithm equals or outperforms GLS, TS3 and HBP for all subclasses. When comparing with SCH, a very sophisticated algorithm that had obtained the best known results for these problems, the results are almost identical, with GRASP improving four SCH solutions and SCH improving two GRASP solutions. The last column shows that for most of the instances GRASP, and also SCH, get proven optimal solutions. That indicates that both are extremely good and their results on these benchmark problems are difficult to improve on. The statistical analysis confirms that GRASP and SCH are significantly better than GLS, TS3 and HBP.

Table 6 Results for two-dimensional instances. Part II[a]

Class	n	Instances	LB^*	GRASP			SCH	GLS	TS3	Opt.
				Best iter	Time	Sol				
cgcut	16–62	3	27	0.33	0.01	**27**	**27**	**27**	**27**	3
gcut	10–50	13	104	1.46	0.01	**104**	**104**	**104**	108	13
ngcut	7–22	12	32	0.42	0.08	**32**	**32**	**32**	36	12
beng 1–8	20–120	8	54	10.25	0.21	**54**	55			8
beng 9–10	160–200	2	13	6.50	0.00	**13**				2

[a]The best values appear in bold

5 Conclusions

We have designed a new GRASP algorithm, based on maximal-spaces, for the bin-packing problem. We have developed a new constructive algorithm, adapting procedures which were successful for the container loading problem and defining some new improvement moves. These improvement moves have been combined in a VND structure, allowing the algorithm to obtain high-quality solutions. Two extensive computational experiments with three and two dimensional instances already used by other authors show that the new heuristic on average obtains similar or better solutions than the other existing algorithms for the three-dimensional and two-dimensional bin-packing problems.

Finally, we would like to mention that the algorithm is quite flexible and could be adapted to accommodate other conditions or constraints, such as the possibility of rotating the items.

Acknowledgements This study has been partially supported by the Spanish Ministry of Science and Technology, DPI2005-04796, cofinanced by FEDER funds, and by Project PCI08-0048-8577, Consejeria de Ciencia y Tecnologia, Junta de Comunidades de Castilla-La Mancha.

References

Beasley, J. E. (1985a). Algorithms for unconstrained two-dimensional guillotine cutting. *Journal of the Operational Research Society*, *36*, 297–306.

Beasley, J. E. (1985b). An exact two-dimensional non-guillotine cutting tree search procedure. *Operations Research*, *33*, 49–64.

Beasley, J. E. (1990). OR-library: Distributing test problems by electronic mail. *Journal of the Operational Research Society*, *41*, 1069–1072.

Bengtsson, B. E. (1982). Packing rectangular pieces—a heuristic approach. *The Computer Journal*, *25*, 353–357.

Berkey, J. O., & Wang, P. Y. (1987). Two dimensional finite bin packing algorithms. *Journal of the Operational Research Society*, *38*, 423–429.

Bischoff, E. E., Janetz, F., & Ratcliff, M. S. W. (1995). Loading pallets with non-identical items. *European Journal of Operational Research*, *84*, 681–692.

Boschetti, M. A. (2004). New lower bounds for the finite three-dimensional bin packing problem. *Discrete Applied Mathematics*, *140*, 241–258.

Boschetti, M. A., & Mingozzi, A. (2003a). Two-dimensional finite bin packing problems. Part I: New lower and upper bounds. *4OR*, *1*, 27–42.

Boschetti, M. A., & Mingozzi, A. (2003b). Two-dimensional finite bin packing problems. Part II: New lower and upper bounds. *4OR*, *2*, 135–147.

Christofides, N., & Whitlock, C. (1977). An algorithm for two-dimensional cutting problems. *Operations Research*, *25*(1), 30–44.

Crainic, T. G., Perboli, G., & Tadei, R. (2008). *TS*^{2}*PACK*: A two-level tabu search for the three-dimensional bin packing problem. *European Journal of Operational Research*. doi:10.1016/j.ejr.2007.06.063.

den Boef, E., Korst, J., Martello, S., Pisinger, D., & Vigo, D. (2005). Erratum to 'The three-dimensional bin packing problem': robot-packable and orthogonal variants of packing problems. *Operations Research*, *53*, 735–736.

Faroe, O., Pisinger, D., & Zachariasen, M. (2003). Guided local search for the three-dimensional bin-packing problem. *INFORMS Journal on Computing*, *15*(3), 267–283.

Fekete, S. P., & Schepers, J. (2004a). A combinatorial characterization of higher-dimensional orthogonal packing. *Mathematics of Operations Research*, *29*(2), 353–368.

Fekete, S. P., & Schepers, J. (2004b). A general framework for bounds for higher-dimensional orthogonal packing problems. *Mathematical Methods of Operations Research*, *60*(2), 311–329.

Fekete, S. P., & Van der Veen, J. C. (2007). PackLib^2: An integrated library of multi-dimensional packing problems. *European Journal of Operational Research 183*, 1131–1135.

Fekete, S. P., Schepers, J., & Van der Veen, J. C. (2007). An exact algorithm for higher-dimensional packing. *Operations Research*, *55*, 569–587.

Feo, T. A., & Resende, M. G. C. (1989). A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, *8*, 67–71.

Gibbons, J. D., & Chakraborti, S. (2003). *Non parametric statistical inference* (4th ed.). New York: Marcel Dekker.

Hansen, P., & Mladenovic, N. (2005). Variable neighborhood search. In E. Burke & G. Kendall (Eds.), *Search methodologies: introductory tutorials in optimization and decision support techniques* (pp. 211–238). Berlin: Springer.

Lodi, A., Martello, S., & Vigo, D. (1999). Approximation algorithms for the oriented two-dimensional bin packing problem. *European Journal of Operational Research*, *112*(1), 158–166.

Lodi, A., Martello, S., & Vigo, D. (2002). Heuristic algorithms for the three-dimensional bin packing problem. *European Journal of Operational Research*, *141*(2), 410–420.

Lodi, A., Martello, S., & Vigo, D. (2004). TSpack: a unified tabu search code for multi-dimensional bin packing problems. *Annals of Operations Research*, *131*, 203–213.

Martello, S., & Vigo, D. (1998). Exact solution of the two-dimensional finite bin packing problem. *Management Science*, *44*(3), 388–399.

Martello, S., Pisinger, D., & Vigo, D. (2000). The three-dimensional bin packing problem. *Operations Research*, *40*, 256–267.

Martello, S., Pisinger, D., Vigo, D., den Boef, E., & Korst, J. (2007). Algorithm 864: Algorithms for general and robot-packable variants of the three-dimensional bin packing problem. *ACM Transactions on Mathematical Software*, *33*, 1.

Monaci, M., & Toth, P. (2006). A set-covering-based heuristic approach for bin-packing problems. *INFORMS Journal on Computing*, *18*(1), 71–85.

Parreño, F., Alvarez-Valdes, R., Oliveira, J. F., & Tamarit, J. M. (2008a). A maximal-space algorithm for the container loading problem. *INFORMS Journal on Computing*, *20*, 412–422.

Parreño, F., Alvarez-Valdes, R., Oliveira, J. F., & Tamarit, J. M. (2008b). Neighborhood structures for the container loading problem: a VNS implementation. *Journal of Heuristics*. doi:10.1007/s10732-008-9081-3.

Prais, M., & Ribeiro, C. C. (2000). Reactive GRASP: An application to a matrix decomposition problem in TDMA traffic assignment. *INFORMS Journal on Computing*, *12*(3), 164–176.

Resende, M. G. C., & Ribeiro, C. C. (2003). Greedy randomized adaptive search procedures. In F. Glover & G. Kochenberger (Eds.), *Handbook of metaheuristics* (pp. 219–249). Dordrecht: Kluwer Academic.

Wäscher, G., Haussner, H., & Schumann, H. (2007). An improved typology of cutting and packing problems. *European Journal of Operational Research*. *183*, 1109–1130.