Computers & Operations Research 40 (2013) 3081–3090

Contents lists available at ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/caor

A GRASP/Path Relinking algorithm for two- and three-dimensional multiple bin-size bin packing problems

R. Alvarez-Valdes [a,*], F. Parreño [b], J.M. Tamarit [a]

[a] University of Valencia, Department of Statistics and Operations Research, Burjassot, Valencia, Spain
[b] University of Castilla-La Mancha, Department of Mathematics, Albacete, Spain

ARTICLE INFO

Available online 11 April 2012

Keywords:
Bin packing
Heuristics
GRASP
Path Relinking

ABSTRACT

The three-dimensional multiple bin-size bin packing problem, MBSBPP, is the problem of packing a set of boxes into a set of bins when several types of bins of different sizes and costs are available and the objective is to minimize the total cost of bins used for packing the boxes. First we propose a GRASP algorithm, including a constructive procedure, a postprocessing phase and some improvement moves. The best solutions obtained are then combined into a Path Relinking procedure for which we have developed three versions: static, dynamic and evolutionary. An extensive computational study, using two- and three-dimensional instances, shows the relative efficiency of the alternatives considered for each phase of the algorithm and the good performance of our algorithm compared with previously reported results.

1. Introduction

The three-dimensional multiple bin-size bin packing problem can be defined as follows: given a set of n three-dimensional rectangular items *(boxes)*, each characterized by width w_j, height h_j, depth d_j and a demand q_j, $j \in J$, and a set of m types of three-dimensional rectangular containers *(bins)* of width W_i, height H_i, depth D_i and cost C_i, $i \in I$, find an orthogonal packing, without overlapping, of all the items into the bins at minimum cost. We assume, without loss of generality, that all the input data are positive integers. We also assume that $\forall j \in J\ \exists\, i \in I \,|\, w_j \le W_i,\ h_j \le H_i,\ d_j \le D_i$, that is, every item fits into at least one bin type, as otherwise no solution to the problem exists. The number of available bins of each type is unlimited. If no items fit into a bin type, then that type of bin may be deleted from the problem. Also, as Pisinger and Sigurd [26] point out, dominated bin types may be removed. If bin types i_1 and i_2 satisfy $C_{i_2} \ge C_{i_1}$, $W_{i_2} \le W_{i_1}$, $H_{i_2} \le H_{i_1}$, $D_{i_2} \le D_{i_1}$ we may remove bin type i_2, since we can always find an optimal solution which does not make use of the dominated bin type i_2.

The two-dimensional multiple bin-size packing problem addresses the same question for two-dimensional bins (W_i,H_i) and boxes (w_j,h_j) and can be considered a special case of the three-dimensional problem when $d_j = D_i = D, \forall j \in J, \forall i \in I$. According to the typology proposed by Wäscher et al. [32], multiple bin packing problems are classified as MBSBPP (multiple bin-size bin packing problem). The problem generalizes the well-known one-dimensional bin packing problem (1BP), hence it is strongly NP-hard.

Three-dimensional packing problems, and particularly MBSBPP, are of particular practical interest in logistics, as in order to be transported the boxes required have to be packed into bins of different sizes at different costs. In fact, our research has been motivated by a real problem in which a company works as a logistics center, storing and distributing a large variety of goods. Every morning the clients' orders, involving different types of products packed into boxes of different sizes, have to be put together into larger boxes, or bins, to be delivered to the clients. Several types of bins are available, each of them with different dimensions and a different cost, and the question is to choose the most appropriate bins so as to minimize the transportation costs. Apart from that obvious application, MBSBPP also appears in other industrial problems such as the cutting of foam rubber in armchair production and packaging design and, as rotation of the boxes is not allowed, the model can also be used for solving several scheduling problems.

Friesen and Langston [13] presented one of the first approaches to the one-dimensional variable-size bin packing problem. In this case, an unbounded number of bins from a finite collection of bin sizes are available to pack a set of items into. The objective is to minimize the total space used in packing. Correia et al. [7] proposed several formulations for the 1D-MBSBPP and developed new valid inequalities. They used a set of instances randomly generated by Monaci [23], who also presented lower bounds and an exact algorithm.

* Corresponding author. Tel.: +34 963544308; fax: +34 963543238.
E-mail address: ramon.alvarez@uv.es (R. Alvarez-Valdes).

0305-0548/$ - see front matter © 2012 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.cor.2012.03.016

Seiden and van Stee [31] established new upper and lower bounds for multidimensional generalizations of bin packing, including the d-dimensional variable-size bin packing. Epstein and van Stee [9] developed an online optimal bounded space algorithm for variable-sized multidimensional bin packing. Epstein and Levin [10] proposed an asymptotic polynomial-time approximation scheme for the generalized problem, where an available bin size is associated with a fixed cost, which may be smaller or larger than its size. Chen et al. [6] considered the three-dimensional version of the problem. They presented an analytical model, including the consideration of multiple containers, multiple object sizes, object orientations, and the possibility of objects overlapping in the container.

The first algorithm for the multidimensional problem was provided by Hopper and Turton [18] for the two-dimensional problem. They considered a variant with bounded numbers of bins of each type and uniform bin costs. In Pisinger and Sigurd [26], a mathematical formulation and a column generation procedure are also developed and used for solving the two-dimensional MBSBPP. They considered a version of the problem in which there are no bounds in the number of bins of each type that can be used, studying several bounds from the continuous lower bound to an integer programming lower bound, and proposed a branch-and-price algorithm based on the formulation by Gilmore and Gomory [14,15].

The work by Brunetta and Grégoire [5] for the three-dimensional case is motivated by a real case in a biscuit factory with smaller objects of 15 different sizes (in this case the smaller objects are boxes of confectionary). They propose a tree-search algorithm that implicitly explores the solution space in order to maximize the volumetric utilization of multiple-sized containers. In their paper they consider a version of the problem where there are bounds in the number of bins of each type that can be used. They test their procedure on the real data and on a set of instances which they generate considering uniform bin costs and instances with up to 1000 boxes.

Another algorithm for the three-dimensional-MBSBPP is proposed by Ertek and Kilic [11], who propose a beam search algorithm, a greedy algorithm and a tree search enumeration algorithm to solve a packing problem in a major automobile manufacturer in Turkey. They generated 90 instances, considering the cost to be an exponential function of the volume.

Wu et al. [33] study a variant of the three-dimensional bin packing problem (3D-BPP), where bin height can be adjusted to the cartons packed into it. The bins and cartons to be packed are assumed to be rectangular in shape. The objective function is the filling ratio and they designed a genetic algorithm for the problem.

Recently Ortmann et al. [24] have studied and improved level heuristics for this problem, considering values for the bins equal to their volume. They have used and compared 11 level heuristics for the problem, with an objective function consisting of maximizing the average filling ratio of the bins.

In this paper we develop metaheuristic algorithms for the three-dimensional and two-dimensional multiple-sized bin packing problem. The bin types may have variable costs, i.e. costs which are not proportional to the size of the bin, and the objective is to minimize the cost of the bins used. We first develop a GRASP algorithm and then some Path Relinking strategies to combine the best solutions obtained in the iterative process. We also propose two simple lower bounds, improving existing bounds by Monaci [23] and by Pisinger et al. [26]. The algorithms are tested on the set of two-dimensional test problems generated by Pisinger and Sigurd [26] and on a new set of three-dimensional instances that we have generated in order to assess the contribution of their different elements to the quality of the solutions. The final implementation of the algorithm is then compared with other existing procedures, showing its superior performance.

2. GRASP algorithm

The GRASP algorithm was developed by Feo and Resende [12] to solve hard combinatorial problems. For an updated introduction, refer to Resende and Ribeiro [28]. Each GRASP iteration consists of a constructive phase where a solution is built, and an improvement phase, usually consisting of a simple local search. The construction phase is iterative, greedy, randomized and adaptive. In this section we describe a GRASP algorithm for the variable-size bin packing problem.

2.1. Constructive algorithm

We have a list $\mathcal{B}$ of boxes still to be packed, initially the complete list of boxes, ordered by the non-decreasing number of bin types into which each box fits. The first tie-break criterion is the non-decreasing cost of the cheapest bin type into which the box fits. If necessary, secondary criteria are non-increasing box volume and non-increasing w_j, h_j and d_j. Therefore, large and difficult boxes are considered first.

While $\mathcal{B}$ is non-empty, we choose the first box in $\mathcal{B}$, b^*, for packing. Among the bin types into which b^* fits, we choose the bin type with the minimum cost per unit of volume. Ties are broken by non-decreasing bin type volume. If there are several boxes with the same dimensions as b^*, we build and pack a layer, that is, a rectangular array of these boxes arranged in rows and columns.

For the particular case in which the cost of the bin type is equal to its volume, the selection of the bin is slightly modified. If the total volume of the remaining pieces to pack is less than the volume of some of the existing bin types, then we choose the cheapest bin type into which all these pieces fit.

Once a bin of the selected type is opened to be partially filled by b^* (or a layer of boxes b^*), it is filled with the strategy used in Parreño et al. [25], the algorithm *Fill*. When no more boxes can be packed into this bin, the list $\mathcal{B}$ is updated and the process goes on until all the boxes are packed.

The algorithm *Fill* works on the maximal spaces of the bin which can be used to pack new boxes. In this case, as the selected box is packed into a new bin, three maximal spaces are created (two in the case of two-dimensional problems, as appears in Fig. 1). The algorithm *Fill* uses un updated list of maximal spaces and the list of boxes still to be packed and tries to fill the volume

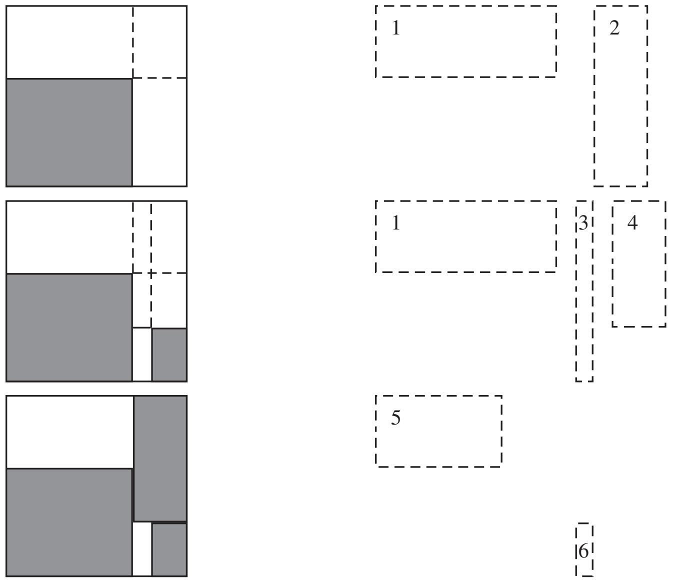

Fig. 1. Maximal-spaces in two dimensions.

of the bin as much as possible. The details of the algorithm can be found in [25]. For completeness we will outline its main steps here.

ALGORITHM FILL

- Step 0: *Initialization.*
  $\mathcal{S}$, the empty maximal spaces created when packing the selected box.
  $\mathcal{B}$, the set of boxes still to be packed.
- Step 1: *Choosing the maximal space in $\mathcal{S}$.*
  We have a list $\mathcal{S}$ of empty maximal spaces, which are the largest empty parallelepiped available for filling with boxes.
  At each step we take the maximal space with the minimum distance to a corner of the bin and use the volume of space as a tie-breaker. The corner of the maximal space with the shortest distance to a corner of the bin will be the corner into which the box will be packed. The reason behind that decision is to fill the corners of the bin first, then its sides and finally the inner space.
- Step 2: *Choosing the boxes to pack.*
  Once a maximal space S^* has been chosen, we take from the ordered list $\mathcal{B}$ the first box i fitting into S^*. If there are several boxes of the same dimensions, we consider the possibility of packing a layer, that is, packing several of these boxes arranged in rows and columns.
- Step 3: *Updating the list $\mathcal{S}$.*
  Unless the box or layer fits exactly into 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 box or 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 (see Fig. 1).
  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.

2.2. Randomization strategies

In order to produce different solutions we randomize both selections, the box to be packed and the bin type to pack it into.

We have studied two types of randomization procedures for selecting the box to be packed. In the first one, the box is selected at random from among a restricted set of candidates, *RCL_1*, composed of the first $100\delta\%$ of the boxes on the list $\mathcal{B}$, where $0 \le \delta \le 1$ is a parameter to be determined. In the second alternative, introduced by Resende and Werneck [29], a restricted candidate list, *RCL_2*, is built by selecting a fraction γ $(0 \le \gamma \le 1)$ of the elements on the list $\mathcal{B}$ at random. Then, the first element of *RCL_2* on the ordered list $\mathcal{B}$ is selected.

It is difficult to determine the value of δ (γ) that gives the best average results. The principle of reactive GRASP, proposed for the first time by Prais and Ribeiro [27], 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 $\mathcal{D}=\{\delta_1,\delta_2,\dots,\delta_m\}$, but after a certain number of iterations, N, 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. In this way, the randomization procedure of GRASP reacts to the results obtained in the iterative process, tuning the parameter to the more suitable values for each instance. The procedure is described in Fig. 2, following Delorme et al. [8].

```
begin Reactive GRASP
1       $\mathcal{D} = \{\delta_1, \delta_2, ..., \delta_m\}$,
2       $V_{best} = \infty$;   $V_{worst} = 0$
3       $n_{\delta^*} = 0$, number of iterations using $\delta^*$, $\forall\, \delta^* \in \mathcal{D}$
4       $Sum_{\delta^*} = 0$, sum of values of solutions obtained using $\delta^*$
5       $P(\delta = \delta^*) = p_{\delta^*} = 1/|\mathcal{D}|, \forall\, \delta^* \in \mathcal{D}$
6       $numIter = 0$
7       while $numIter < maxIter$
8         Choose $\delta^*$ from $\mathcal{D}$ with probability $p_{\delta^*}$
9         $n_{\delta^*} = n_{\delta^*} + 1$
10        $numIter = numIter + 1$
11        Perform an iteration using $\delta^*$, obtaining $s$ with value $V$
12        if $V < V_{best}$ then $V_{best} = V$
13        if $V > V_{worst}$ then $V_{worst} = V$
14        $Sum_{\delta^*} = Sum_{\delta^*} + V$
15        if $mod(numIter, N) = 0$, then:
16                $average_\delta = Sum_\delta / n_\delta, \forall\, \delta^* \in \mathcal{D}$
17                $eval_\delta = \left(\frac{V_{worst} - average_\delta}{V_{worst} - V_{best}}\right)^\alpha \quad \forall \delta \in \mathcal{D}$
18                $p_\delta = eval_\delta / (\sum_{\delta' \in \mathcal{D}} eval_{\delta'}) \quad \forall \delta \in \mathcal{D}$
19      end-while
end
```

Fig. 2. Reactive GRASP.

From among the bin types into which the selected box fits, instead of choosing the bin type with lowest cost we use a biased randomized strategy. If we can use bin types whose costs are $C_i, i \in I$, the probability of choosing bin type i is set equal to: $p_i = C_i / \sum_i C_i$

When using Algorithm *Fill* [25] to fill the bins, we can also randomize the selection of the corner to pack each box into. The corner is randomly chosen for each iteration of the GRASP algorithm.

2.3. Improvement phase

Each solution built at the randomized constructive phase is the starting point for a local search procedure in which we try to improve the solution. In this phase we have the bins of the solution ordered by non-increasing efficiency, defined as the volume of the boxes packed into the bin divided by the cost of the bin.

Before proceeding to the local search improving phase, we try two simple improving procedures, which can be considered as a postprocessing phase of the solution.

First, we try to pack the contents of two bins into one larger bin whose cost is lower than the sum of the costs of the original bins. For each pair of bins in the solution we consider another bin whose volume is larger than the total volume of the packed boxes and try to pack all these boxes into the new bin using the deterministic constructive algorithm *Fill*.

The second procedure tries to pack the boxes of one bin into another bin of lower cost. For each bin in the solution, we consider bins with a lower cost whose volumes are larger than the total volume of the boxes and try to pack them using the deterministic constructive algorithm *Fill*.

For the local search phase, we have developed two methods. The first method consists of eliminating the last $k\%$ of bins in the solution (for instance, the last 40%) and packing again the list of boxes corresponding to these bins with the deterministic constructive algorithm.

The second alternative is to eliminate r bins from the solution and pack the corresponding boxes using the deterministic constructive algorithm. We have taken $r = 2, 3, 4$. The subsets are created in a systematic way, considering the bins in non-decreasing order of efficiency. We only study subsets of r bins in which at least one of them has an efficiency rate lower than the average efficiency of all the bins in the solution. Even with this limitation, the number of neighbors of a solution can be very large and so we have limited the search to a maximum of 500 neighbors.

3. Path Relinking

Path Relinking (PR) is a method for integrating intensification and diversification strategies in the context of Tabu Search [16]. This approach generates new solutions by exploring trajectories that connect high quality solutions which have been collected along the iterative process to form the *elite set* (ES). Starting from one of these solutions, called the initiating solution, a path is generated in the solution space that leads towards another solution, called the guiding solution. This is done by selecting moves that introduce the attributes of the guiding solution into the new solutions.

Laguna and Martí [20] adapted PR in the context of GRASP as a form of intensification. The relinking in this context consists of finding a path between a solution found with GRASP and a chosen elite solution. Therefore, the relinking concept has a different interpretation within GRASP since the solutions found from one GRASP iteration to the next are not linked by a sequence of moves (as in the case of Tabu Search). Resende and Ribeiro [28] present numerous examples of GRASP with PR.

3.1. New solutions obtained in the relinking process

In this section we explore the adaptation of GRASP with PR to the variable-size bin packing problem. We follow a constructive strategy, inserting the bins from a solution B, one at a time, into a solution A. Bins in B appearing in A with the same set of boxes are obviously not considered for insertion. When we insert a bin from B into A, we remove all the bins in A containing boxes packed in that bin, to prevent a repetition of boxes. Then we pack the boxes which have been left unpacked, using the deterministic constructive algorithm. At the end of the process, we will have reproduced solution B, but new solutions will have been generated along the path. When inserting the bins from solution B, we first insert the bins with the largest efficiency. Another possibility could be the insertion of bins into solution A in a random order.

Figs. 3–6 show an example of the procedure. In Fig. 3 we have solution A with seven bins, A_1–A_7, and in Fig. 4 solution B with six bins, B_1–B_6, both of them ordered by efficiency. As B_2 is equal to A_3 and B_4 is equal to A_5 (the same type of bin and the same list of boxes), these bins are not considered for insertion. In Fig. 5 we introduce the bin B_1 into solution A, removing bins A_1, A_6 and A_7 which contain boxes packed in B_1. Therefore, the first four bins in the new solution C are bins A_2, A_3, A_4, A_5, which have not been affected by the insertion of bin B_1, and the fifth is B_1. The remaining boxes, which were initially packed into bins A_1, A_6 and A_7, are not packed yet into any bin. In Fig. 6 these unpacked boxes have been packed, producing a complete combined solution C. In this case, all these boxes fit into one bin and the new solution is better than both A and B.

3.2. Building the elite set

If we only consider a quality criterion to populate the elite set, we could simply build it with the best $|ES|$ solutions generated by GRASP. However, Resende and Werneck [29] have empirically found that an application of PR to a pair of solutions is likely to be unsuccessful if the solutions are very similar. Therefore, to construct

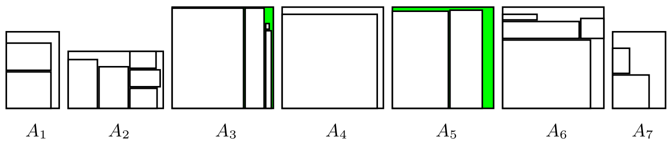

Fig. 3. Example class 10, instance 9. Solution A of value 50 095.

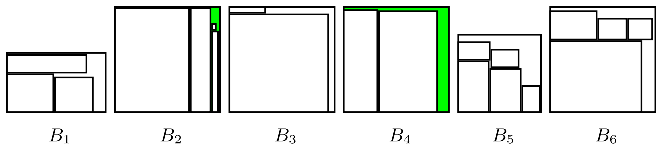

Fig. 4. Example class 10, instance 9. Solution B of value 49 444.

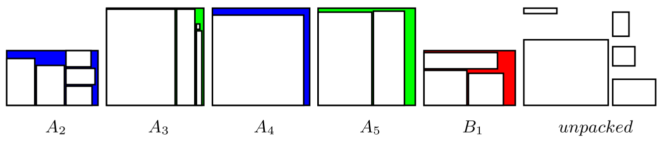

Fig. 5. Inserting B_1 into solution A and removing bins in A with boxes of B_1.

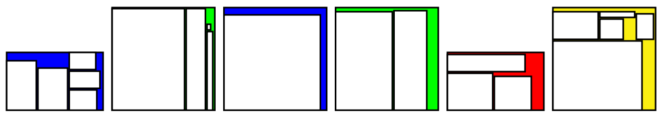

Fig. 6. Packing the remaining boxes. New solution C of value 48 858.

ES we will consider both quality and diversity in the same way as Resende et al. [30]. Initially ES is empty and we apply GRASP for $|ES|$ iterations and populate it with the solutions obtained. We order the solutions in ES from the best (s^1) to the worst ($s^{|ES|}$). Then, in the following GRASP iterations, we test whether the generated (constructed and improved) solution s' qualifies for entry into ES. Specifically, if s' is better than the best s^1, it is put into the set. Moreover, if it is better than the worst s^b and is sufficiently different from the other solutions in the elite set ($d(s',ES) \geq D$), it is also put into ES. The *ES* set is kept ordered from best to worst solution values.

We have defined two measures of distance, both based on the differences of bins used in each solution. If we have a vector x^t with the number of bins of each type used in a solution s^t, and vectors x^i for the solutions s^i in *ES*, and we have *m* types of available bins, we define:

$$d_1(s^t,ES) = \sum_{i=1}^{|ES|}\sum_{j=1}^{m} |x_j^i - x_j^t|$$

$$d_2(s^t,ES) = \min_{\{i=1,\ldots,|ES|\}} \sum_{j=1}^{m} |x_j^i - x_j^t|$$

The first measure, d_1, can be very large if there are big differences between some of the solutions in the elite set and the new solution, but this new solution can be very similar to others in the elite set.

In the second measure, d_2, we only take into account the minimum of these differences, which is the minimum Hamming distance from x^t to the vectors belonging to ES. For example, a value of 2 in this distance means that the new solution and all the solutions from the elite set have a difference of at least two bins. The parameter *D* is a distance threshold value that tries to reflect the term *sufficiently different*. For the first measure we use a large value, for instance $D=2|ES|$, and for the second measure a small value, $D=2$ or $D=1$.

To keep the size of ES constant, whenever we add a solution to this set, we remove another one. To maintain the quality and diversity, we remove the closest solution to x' in ES from those which are worse than it in value.

3.3. *Variants of the Path Relinking algorithm*

We have studied three variants of the PR: a static variant, a dynamic variant, and evolutionary Path Relinking, as in Resende et al. [30]. Given two solutions *x*, *y* to be linked, we apply PR in both directions, i.e. *PR*(*x*,*y*) from *x* to *y* and *PR*(*y*,*x*) from *y* to *x*. The best solution generated in both paths is subjected to the local search method for improved outcomes.

In the first static variant, Fig. 7, the elite set is created during the GRASP iterative process (lines 2–8) and when GRASP ends we apply Path Relinking to generate new solutions between all pairs of solutions in the elite set (lines 9–12).

The second alternative implementation of GRASP with PR consists of a dynamic update of the elite set as introduced in Laguna and Marti [19] (Fig. 8). In this design, each solution *s* generated with GRASP is directly subjected to the PR algorithm, which is applied between *s* and a solution s^j selected from ES. The selection is probabilistically made according to the value of the solutions.

Evolutionary Path Relinking (EvPR) was introduced by Resende and Werneck [29] as a postprocessing phase for GRASP with PR (Fig. 9). Initially we use the dynamic variant of GRASP with PR for a given number of iterations, *LocalIter* (lines 6–13). When this number is reached, the procedure stops and a Path Relinking phase is applied to the current members of ES. The solutions obtained with this latter application of PR are considered to be candidates for entry into ES and PR is again applied to them as long as new solutions enter ES (lines 15–23). This way we say that ES evolves. When no new solutions enter into ES, the process returns to the dynamic variant of GRASP with PR in an outer global iteration for a given number of iterations, *GlobalIter* (lines 4–25).

4. Computational results

In this section we will present the results of the computational experiments. Our goal is to show that on average the new algorithm produces high quality solutions for the two- and three-dimensional bin-packing problems appearing in the literature. We also show that the 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.

4.1. *Instances*

The first instances published (see OR-library, [2]) for the two-dimensional-MBSBPP were those generated by Hopper and Turton [18]. However, those instances have an upper bound on the use of each bin type, and the bin costs are uniform. Hence, Pisinger and Sigurd [26] created a new set of instances with

```
begin Static GRASP+PR
1    $NumIter$ = total number of iterations
2    Apply GRASP for $|ES|$ iterations to build the initial Elite Set
3    $iter = |ES| + 1$
4    while $iter \leq NumIter$ do
5        Apply GRASP to obtain a new solution $s$
6        Determine if $s$ enters the Elite Set according to Section 3.2
7        $iter = iter + 1$
8    end-while
9    for $i = 1$ to $|ES| - 1$ and $j = i + 1$ to $|ES|$ do
10       Apply Path Relinking to $s^i$ and $s^j$
11       Apply Local Search to the best solution obtained in the process
12   end-for
13   return $s^{best}$
end
```

Fig. 7. Static PR.

```
begin Dynamic GRASP+PR
1    $NumIter$ = total number of iterations
2    Apply GRASP for $|ES|$ iterations to build the initial Elite Set
3    $iter = |ES| + 1$
4    while $iter \leq NumIter$ do
5        Apply GRASP to obtain a new solution $s$
6        Select $s^j$ from $ES$ at random, according to its quality
7        Apply Path Relinking to $s$ and $s^j$
8        Apply Local Search to the best solution obtained in the process
9        Determine if the new solution enters $ES$ according to Section 3.2
10       $iter = iter+1$
11   end-while
12   return $s^{best}$
end
```

Fig. 8. Dynamic PR.

```
begin Evolutionary GRASP+PR
1    $Globaliter$ = number of global iterations
2    Apply GRASP for $|ES|$ iterations to build the initial Elite Set
3    $iter1 = 1$
4    while $iter1 \leq Globaliter$ do
5        $iter2=1$
6        while $iter2 \leq Localiter$ do
7            Apply GRASP to obtain a new solution $s$
8            Select $s^j$ from $ES$ at random, according to its quality
9            Apply Path Relinking to $s$ and $s^j$
10           Apply Local Search to the best solution obtained in the process
11           Determine if the new solution enters $ES$ according to Section 3.2
12           $iter2= iter2+1$
13       end-while
14       $Evol = True$
15       while $Evol=True$ do
16           $Evol= False$
17           for $i = 1$ to $|ES| - 1$ and $j = i + 1$ to $|ES|$ do
18               Apply Path Relinking to $s^i$ and $s^j$
19               Apply Local Search to the best solution obtained in the process
20               Determine if the new solution enters $ES$ according to Section 3.2
21               if $ES$ is updated, then $Evol=True$
22           end-for
23       end-while
24       $iter1= iter1+1$
25   end-while
26   return $s^{best}$
end
```

Fig. 9. Evolutionary PR.

variable bin costs, based on the characteristics of the instances proposed by Berkey and Wang [3] and by Martello and Vigo [21]. The original instances were classified into 10 classes, varying the characteristics (dimension and shape) of the pieces. Each class was composed of 50 instances, 10 for each value of $n \in \{20, 40, 60, 80, 100\}$, so there were a total of 500 instances. Instead of a unique bin type, Pisinger and Sigurd [26] included five bin types for each instance. The bin sizes were picked uniformly at random from $[W/2,W] \times [H/2,H]$, where W, H are the bin width and height from the original problem.

The costs C_k (as they appear in the van Vuuren web page: www.vuuren.co.za) are a function of the bin areas W_kH_k: $c_k := \lfloor 1.2W_kH_k - 0.2WH \rfloor$. This cost function imposes a relatively cheaper cost on smaller bins.

For three-dimensional bin packing, the standard benchmark for the classical 3DBPP with one bin type is the set of 320 problems, generated by Martello et al. [22], also divided into 8 classes with 40 instances each, 10 instances for each $n \in \{50, 100, 150, 200\}$. The instance generator is available at http://www.diku.dk/~pisinger/codes.html. Based on these instances we have created a new set of instances with five bin types in each instance. The bin sizes have been picked uniformly at random from $[W/2,W] \times [H/2,H] \times [D/2,D]$, where W, H, D are the bin width, height and depth from the original problem. The bin costs, C_k are set as follows:

$$c_k := \left\lceil 10000\frac{1.2V_k - 0.2WHD}{WHD} \right\rceil.$$

To compare the different versions we have used the gap $(UB-LB)/UB$, where UB is the solution of each algorithm and LB is the lower bound obtained using the linear relaxation of an integer formulation of the problem [1].

For the preliminary computational results we have used two instances of each class and type. That produces a set of 100 two-dimensional instances and 64 three-dimensional instances, large enough to apply statistical analysis to the solutions for comparing the different strategies.

4.2. Choosing the best strategies

The first two comparisons involve the two basic elements of the GRASP algorithm: the randomization procedures of the constructive phase and the methods for the improvement phase. In both cases, we set a computational time limit of 15 s for two-dimensional and 150 s for three-dimensional problems, or a maximum number of 50 000 iterations. These time limits are similar to those used in previous studies on the bin packing problem [25].

Table 1 compares the results of two types of randomization procedures embedded in the constructive procedure. We compare the two types of restricted candidate list, *RCL_1* and *RCL_2*. For both types of lists, we consider randomizing only the selection of the box and also the possibility of randomizing the choice of the bin. A fifth strategy consists of randomizing only the choice of the bins, with probabilities inversely proportional to the ratio cost/volume. The table also includes the results of the constructive deterministic algorithm in order to assess the improvement obtained by the randomization procedures. For the reactive GRASP in Fig. 2 we used the set of values $\mathcal{D}=\{0.1,0.2,\ldots,0.9\}$. The probabilities p_i are initially set at 1/9 and are revised every $N=500$ iterations. The parameter α is fixed at 10, as in Prais and Ribeiro [27]. The same values are used for parameter γ.

For the analysis of the results we use a non-parametric analysis for related measures, the Friedman test, and the Wilcoxon test for pairwise comparisons [17]. There are significant differences between the five strategies and the best results are obtained by randomizing only the selection of the bin. But as the results are very similar to those obtained by *RCL_2*, including the random selection of the bin, and this strategy produces more diversity, this is the strategy used for the following experiments.

In Table 2 we compare four different improvement methods, including the postprocessing in all of them. The first one consists of removing the last $k\%$ of the pieces. As no value of k has been shown to be consistently the best in a preliminary test, it is randomly chosen at each iteration from the interval (0.3,0.9). The second, third and fourth methods consist of removing two, three and four bins (respectively) from the solution and filling them again with the deterministic procedure. The last one is called *all moves* because each move, in the order in which they have been described, is applied to the solution. For comparison, in column 3 we include the results of the randomized constructive algorithm, using *RCL_2* for the selection of the box and also randomizing the selection of the bin. We again use the non-parametric Friedman test to compare these values. The statistical tests show that there are very significant differences between them, and the best option is to use all moves in order.

Table 3 shows the results obtained by the three PR versions of the algorithm: static, dynamic and evolutionary. In all the cases, $|ES|=20$ (as in [30]). In order to give similar computational time to the three versions, we set the following limits. In the static PR, GRASP runs for 15 s for two-dimensional instances and 150 s for three-dimensional instances and then PR is applied. In the dynamic PR, the total running time for the combined GRASP/PR procedure is set at 15/150 s. For the evolutionary version of PR, we run seven global iterations and at each iteration GRASP/PR runs for 1.5/15 s and then PR is applied to the elite set. The statistical analysis shows that there are differences between the

Table 1
The effect of the randomization procedures.

Instances		Constr.	RCL_1		RCL_2		Only bin
			Box	Boxes+Bin	Piece	Boxes+Bin	
2D classes	**1**	8.23	7.61	8.16	8.46	6.88	7.66
	2	6.13	5.02	1.82	3.14	2.52	2.55
	3	16.71	13.62	13.10	12.86	12.40	12.74
	4	9.62	9.62	6.10	6.15	9.62	6.15
	5	14.56	13.16	13.40	13.41	12.26	12.68
	6	5.42	5.42	2.32	3.32	2.09	2.99
	7	15.62	11.84	14.24	12.85	12.64	12.06
	8	12.47	11.78	12.90	12.50	11.92	11.17
	9	4.14	4.56	7.68	6.81	5.45	4.71
	10	12.58	10.10	9.85	10.41	9.21	10.10
Overall 2D		**10.55**	**9.27**	**8.96**	**8.99**	**8.50**	**8.28**
3D classes	**1**	22.25	20.86	23.58	20.96	23.83	20.52
	2	14.59	14.70	16.25	14.81	16.26	14.07
	3	18.80	18.89	20.59	18.84	20.89	17.74
	4	4.72	8.47	9.60	8.50	9.72	4.67
	5	21.49	17.83	16.87	17.81	16.90	20.27
	6	18.49	13.65	15.46	15.62	13.48	15.63
	7	31.99	25.70	25.86	25.78	25.91	30.59
	8	26.07	22.37	22.47	22.36	22.54	24.78
Overall 3D		**19.80**	**17.81**	**18.83**	**18.08**	**18.69**	**18.54**
Overall		**14.16**	**12.61**	**12.81**	**12.54**	**12.48**	**12.28**

Table 2
Results of improvement methods.

Instances		Without improving	Removing last k %	Emptying			All moves
				2 bins	3 bins	4 bins	
2D classes	**1**	6.68	5.88	6.26	7.11	7.09	5.84
	2	2.52	2.49	2.49	2.49	3.10	2.49
	3	12.40	11.43	11.73	12.07	11.80	11.20
	4	9.62	5.73	5.73	5.71	5.81	5.73
	5	12.26	12.04	12.31	12.58	12.42	11.70
	6	2.09	2.00	2.00	2.00	2.63	2.00
	7	12.64	10.76	11.79	12.19	11.98	10.53
	8	11.92	10.01	10.81	11.13	11.16	9.98
	9	5.45	3.84	3.78	4.11	4.19	3.78
	10	9.21	9.21	9.58	9.51	9.65	9.24
Overall 2D		**8.50**	**7.34**	**7.65**	**7.89**	**7.98**	**7.25**
3D classes	**1**	23.83	22.94	20.82	23.26	23.75	20.85
	2	16.26	14.83	13.89	15.68	15.84	13.47
	3	20.89	19.80	18.50	20.49	20.29	18.37
	4	9.72	4.15	3.58	4.71	5.14	3.61
	5	16.90	16.88	17.03	17.14	17.61	17.03
	6	13.48	13.58	14.08	14.25	14.59	12.84
	7	25.91	25.39	26.20	25.38	25.50	25.44
	8	22.54	22.69	22.38	22.78	22.85	22.71
Overall 3D		**18.69**	**17.53**	**17.06**	**17.96**	**18.20**	**16.79**
Overall		**12.48**	**11.32**	**11.32**	**11.82**	**11.97**	**10.97**

Table 3
Path Relinking methods.

Instances		Static	Dynamic	Evolutionary
2D classes	**1**	5.80	5.57	5.47
	2	2.49	2.49	2.49
	3	11.22	10.82	10.77
	4	5.73	5.74	5.88
	5	11.73	11.36	11.43
	6	2.00	2.00	2.00
	7	10.48	10.24	10.35
	8	9.98	9.58	9.61
	9	3.78	3.78	3.78
	10	9.16	8.83	8.99
Overall 2D		**7.24**	**7.04**	**7.08**
3D classes	**1**	20.78	19.51	19.47
	2	13.44	11.56	11.61
	3	18.37	16.99	16.78
	4	3.61	3.44	3.48
	5	17.03	16.63	16.88
	6	12.75	12.04	12.26
	7	25.28	25.16	25.25
	8	22.71	22.19	22.13
Overall 2D		**16.75**	**15.94**	**15.98**
Overall		**10.95**	**10.51**	**10.55**
Time 2D		17.20	14.11	17.36
Time 3D		167.72	150.09	150.64

Table 4
Distances in dynamic Path Relinking.

Instances		Without improvement	Distance		
			d_1. $D=2\|ES\|$	d_2. $D=2$	d_2. $D=1$
2D classes	**1**	5.69	5.57	5.62	5.62
	2	2.49	2.49	2.49	2.49
	3	11.11	10.82	10.89	10.96
	4	5.74	5.74	5.71	5.73
	5	11.69	11.36	11.43	11.40
	6	2.00	2.00	2.00	2.00
	7	10.56	10.24	10.26	10.31
	8	9.92	9.58	9.74	9.63
	9	3.78	3.78	3.78	3.78
	10	9.06	8.83	8.82	8.85
Overall 2D		**7.20**	**7.04**	**7.07**	**7.08**
3D classes	**1**	21.16	19.51	19.74	19.59
	2	13.12	11.56	12.13	11.79
	3	18.31	16.99	17.15	16.85
	4	3.57	3.44	3.48	3.48
	5	17.03	16.63	16.76	16.64
	6	13.15	12.04	12.14	12.29
	7	25.51	25.16	25.19	25.29
	8	22.92	22.19	22.09	22.17
Overall 3D		**16.85**	**15.94**	**16.08**	**16.01**
Overall		**10.97**	**10.51**	**10.59**	**10.56**

three methods but these differences are due to the static PR, which works worse than the other two. Both the dynamic and evolutionary PR present similar results.

Finally, in Table 4 we compare different distances for building the elite set ES. In the previous experiments we used the distance d_1 with $D=2|ES|$. Now, for the first measure d_1 the threshold is set again at $D=2|ES|$ and for the second measure d_2 we try values $D=2$ and $D=1$. The results of distance d_1 with $D=2|ES|$ and d_2 with $D=1$ are comparable; however, the results obtained by d_2 with $D=2$ are worse, maybe because it is more difficult for good solutions to enter into the elite set. The table also includes in column 3 the results obtained by the dynamic version of PR with distance d_1 and $D=2|ES|$, but in the case in which we do not apply the improvement phase to the solutions obtained in the Path Relinking phase. It is clear that improving the solutions obtained in the Path Relinking method is quite effective.

Table 5
Studying the algorithm randomness.

Instances		1 run	15/150 s						150/ 1500 s
			5 runs			10 runs			1 run
			Min.	Max.	Mean	Min.	Max.	Mean	
2D classes	**1**	5.57	5.42	5.80	5.66	5.41	5.80	5.65	5.28
	2	2.49	2.49	2.49	2.49	1.16	2.49	2.25	1.16
	3	10.82	10.79	11.06	10.92	10.66	11.08	10.90	10.64
	4	5.74	5.73	5.74	5.73	5.54	5.74	5.71	5.54
	5	11.36	11.28	11.65	11.47	11.26	11.69	11.50	11.24
	6	2.00	2.00	2.00	2.00	2.00	2.00	2.00	1.82
	7	10.24	10.11	10.45	10.29	10.10	10.49	10.31	10.06
	8	9.58	9.53	9.92	9.68	9.40	9.92	9.67	9.51
	9	3.78	3.78	3.78	3.78	3.78	3.78	3.78	3.78
	10	8.83	8.65	9.03	8.88	8.57	9.08	8.91	8.59
Overall 2D		**7.04**	**6.98**	**7.19**	**7.09**	**6.79**	**7.21**	**7.07**	**6.76**
3D classes	**1**	19.51	18.87	19.75	19.39	18.87	20.04	19.45	18.86
	2	11.56	11.16	12.16	11.71	11.08	12.25	11.64	10.87
	3	16.99	16.20	17.49	16.85	16.20	17.55	16.87	15.95
	4	3.44	3.44	3.59	3.48	3.44	3.59	3.48	3.38
	5	16.63	16.43	17.05	16.75	16.43	17.14	16.81	16.24
	6	12.04	11.73	12.67	12.16	11.73	12.81	12.28	11.84
	7	25.16	24.79	25.51	25.18	24.43	25.64	25.20	24.32
	8	22.19	21.65	22.72	22.17	21.55	22.81	22.13	20.94
Overall 3D		**15.94**	**15.53**	**16.37**	**15.96**	**15.47**	**16.48**	**15.98**	**15.30**
Overall		**10.51**	**10.32**	**10.77**	**10.55**	**10.17**	**10.82**	**10.55**	**10.09**

4.3. Studying the random component of the algorithm

According to the results obtained in previous subsections, the proposed GRASP/PR algorithm uses in the constructive phase the candidate list *RCL_2* randomizing both selections of box and bin with a reactive strategy to update the parameter values. In the improvement phase, all four moves developed in Section 2.3 are applied. The Dynamic Path Relinking strategy is applied at each iteration of GRASP, as it is explained in Section 3.3.

The GRASP/PR algorithm has a random component in the constructive phase. In the experiments in the previous subsections each instance was run only once, but to assess the effect of the associated randomness we run the complete GRASP/PR algorithm 10 times for each of the first two instances of each class and type, with a limit of 50 000 iterations or 15 s per run for two-dimensional problems or 150 s for three-dimensional problems. The results are summarized in Table 5. The third column shows the results when running the algorithm just once. Columns 4–6 show the results for 5 runs, and columns 7–9 the results for 10 runs. Finally, column 10 shows the results obtained by running the algorithm only once, but with a limit of 500 000 iterations or 10 times the previous time limits. The ranges of variation between the best and the worst solutions obtained when running the algorithm 5 or 10 times can be obtained by subtracting columns 4 and 5, and columns 7 and 8.

Obviously, running the algorithm 10 times and keeping the best solution produces better results than running it just once, but the interesting question is whether this extra computing effort of running the algorithm 10 independent times would not be better employed by running it once with a limit of 500 000 iterations. If the GRASP iterations were independent, the results should be the same. In our implementation, the iterations are linked by the reactive GRASP procedure in which the probabilities of the values of parameter γ are updated after a given number of iterations. The results of the previous iterations should guide this procedure to

Table 6
Final results.

Instances		LB	Constr.	Random.	Improv.	Dynamic PR		Iter.
						W/o Imp.	W/Imp.	
2D classes	**1**	1668.22	8.70	8.49	6.64	6.63	6.55	2582.30
	2	1338.62	5.70	1.82	1.22	0.81	0.85	3922.52
	3	17 073.08	17.23	15.23	13.22	13.16	12.97	3025.64
	4	14 704.30	12.14	6.12	5.82	5.39	5.14	5049.94
	5	143 163.94	15.99	13.36	11.98	11.93	11.63	2917.36
	6	111 825.52	10.88	4.38	3.91	3.86	3.79	4056.54
	7	130 224.58	15.28	14.65	11.35	11.37	11.05	3255.62
	8	131 705.22	13.32	13.99	10.59	10.46	10.30	3321.90
	9	330 290.86	4.54	5.32	3.99	3.99	3.98	1417.98
	10	81 753.42	13.50	10.30	9.32	9.27	9.08	4166.80
Overall 2D		**96 374.78**	**11.73**	**9.37**	**7.80**	**7.69**	**7.53**	**3371.66**
3D classes	**1**	226 551.40	21.27	23.47	20.38	20.34	18.89	2890.03
	2	253 805.60	14.63	15.86	13.55	13.50	12.10	4245.33
	3	236 982.10	18.89	20.72	18.35	18.35	16.87	3330.45
	4	559 487.55	4.08	4.71	2.96	2.95	2.90	1924.30
	5	134 126.88	20.35	16.69	16.90	16.92	16.45	5286.45
	6	188 948.25	19.50	16.38	13.58	13.45	13.07	6335.40
	7	95 387.75	29.97	25.15	24.60	24.71	24.32	6847.88
	8	153 434.45	22.46	19.74	19.61	19.79	18.94	4095.38
Overall 3D		**231 090.50**	**18.89**	**17.84**	**16.24**	**16.25**	**15.44**	**4369.40**
Overall		**148 946.76**	**14.52**	**12.67**	**11.10**	**11.03**	**10.62**	**3761.02**

favor values of γ producing best solutions. Therefore, the results obtained using the algorithm once up to 500 000 iterations or 10 times the previous times should be at least equal to and possibly better than those obtained by running it 10 independent times. The results in Table 5 show a slight advantage of the algorithm with 500 000 iterations, and the statistical tests performed on the results summarized in columns 7 and 10 do show a significant difference between them. Therefore, we can draw two conclusions. First, instead of running the algorithm 10 times and reporting the best results obtained, we can run it once with 500 000 iterations. Second, the learning mechanism of the reactive GRASP does seem to have significant effect on the long-term quality of the solutions. The preliminary experiments we ran showed that this way of determining γ performed better than other alternatives.

Table 6 contains the complete computational results for all the instances. Column 4 shows the results of the deterministic constructive algorithm, column 5 the results of the randomized constructive algorithm, column 6 the results of the GRASP algorithm without Path Relinking, and columns 7 and 8 the results of the complete algorithm including the dynamic Path Relinking procedure, without and with the improvement of the new solutions found in the PR process. Column 3 shows the average values of the lower bound obtained for each instance. Column 9 shows the average number of iterations of the algorithm.

4.4. *Comparing with the results of Ortmann et al. [24]*

In order to assess the relative efficiency of our algorithm we have tried to compare it with existing results, but the only available results for the same type of problem we are considering here are those recently published by Ortmann et al. [24]. They proposed some level heuristics for the problem, but setting the bin costs at equal to their volume. Therefore, we have changed our objective function and considered the objective of maximizing the utilization of the bins, defined as the total area of the items divided by the total area of the bins used. Table 7 corresponds to the results obtained on the set of two-dimensional instances with this objective function. The table compares our deterministic constructive procedure and our complete GRASP+PR algorithm with the eight best-performing algorithms reported by Ortmann et al. [24]. We can see that our constructive algorithm obtains slightly better results than the maximum of the eight algorithms. The running time for the constructive algorithm is less than 8 ms, and is therefore comparable to the running times required by any other simple heuristic. The GRASP+PR algorithm has a time limit of 15 s and obviously obtains much better results. The main result in Table 7 is the significative improvement that can be obtained by using a complex metaheuristic procedure instead of a simple heuristic algorithm, if the required running time is available. The last column shows the average number of iterations of the algorithm.

Table 7
Maximizing volume utilization.

Instances		Max Ortmann	Constructive	GRASP/PR	Iterations
2D classes	**1**	88.6	89.77	94.91	3005.48
	2	85.4	87.27	95.70	2579.08
	3	82.2	81.63	90.48	3728.02
	4	83	81.95	93.45	4171.42
	5	81.4	81.10	88.56	3268.42
	6	80.7	78.75	91.39	3174.70
	7	80.9	82.91	89.15	3585.12
	8	81.6	83.43	88.59	3490.48
	9	73.2	73.64	75.63	1428.76
	10	85.5	84.19	91.95	4883.20
No. of boxes	**20**	75.6	75.61	84.81	9917.21
	40	80.3	81.13	89.86	3561.72
	60	83.3	83.88	91.28	1683.17
	80	85	84.93	91.58	900.67
	100	86.5	86.81	92.33	712.79
Overall		**82.25**	**82.46**	**89.98**	**3331.47**

5. Conclusions

We have designed a new GRASP/Path Relinking algorithm for the variable bin-size bin packing problem. We have developed a new constructive algorithm, adapting procedures which were successful for the container loading problem, and used it in a GRASP framework, defining some improvement moves.

The best solutions from the GRASP procedure have been combined in a Path Relinking structure, allowing the algorithm to obtain high-quality solutions. This paper is one of the first implementations of Path Relinking procedures for packing problems. This methodology has been extensively used in other types of combinatorial problems and here we show that it is possible to implement it successfully in packing problems.

Two extensive computational experiments with two- and three-dimensional instances show that the proposed algorithm obtains good results. 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 or imposing upper bounds on the number of copies of each type of bin.

Acknowledgments

This study has been partially supported by the Spanish Ministry of Science and Technology, DPI2008-02700 and DPI2011-24977, cofinanced by FEDER funds, by Project PII2I09-0073-4576, Consejeria de Educación y Ciencia, Junta de Comunidades de Castilla-La Mancha, and by Generalitat Valenciana, ACOMP /2009/264.

References

[1] Alvarez-Valdes R, Parreño F, Tamarit JM. Lower bounds for two- and three-dimensional multiple bin-size bin packing problems. Technical report, Department of Statistics and Operations Research, University of Valencia; 2011.
[2] Beasley JE. OR-library: distributing test problems by electronic mail. Journal of the Operational Research Society 1990;41:1069–72.
[3] Berkey JO, Wang PY. Two dimensional finite bin packing algorithms. Journal of the Operational Research Society 1987;38:423–9.
[4] Brunetta L, Gregoire P. A general purpose algorithm for three-dimensional packing. INFORMS Journal on Computing 2005;17:328–38.
[5] Chen CS, Lee SM, Shen QS. Analytical model for the container loading problem. European Journal of Operational Research 1995;80(1):68–76.
[6] Correia I, Gouveia L, Saldanha-da-Gama F. Solving the variable size bin packing problem with discretized formulations. Computers and Operations Research 2008; 35(6):2103–13.
[7] Delorme X, Gandibleux X, Rodriguez J. GRASP for set packing problems. European Journal of Operational Research 2003;153(3):564–80.
[8] Epstein L, Van Stee R. On variable-sized multidimensional packing. In: Lecture notes in computer science, vol. 3221; 2004. p. 287–98.
[9] Epstein L, Levin A. An APTAS for generalized cost variable sized bin packing. SIAM Journal on Computing 2008;38(1):411–28.
[10] Ertek G, Kilic K. Decision support for packing in warehouses. In: Lecture notes in computer science, vol. 4263; 2006. p. 115–24.
[11] Feo T, Resende MGC. A probabilistic heuristic for a computationally difficult set covering problem. Operations Research Letters 1989;8:67–71.
[12] Friesen DK, Langston MA. Variable sized bin packing. SIAM Journal on Computing 1986;15:222–30.
[13] Gilmore PC, Gomory RE. A linear programming approach to the cutting stock problem. Operations Research 1961;9:849–59.
[14] Gilmore PC, Gomory RE. A linear programming approach to the cutting stock problem 1/2 part II. Operations Research 1963;13:94–119.
[15] Glover F, Laguna M. Tabu Search. Boston: Kluwer; 1997.
[16] Gibbons JD, Chakraborti S. Non parametric statistical inference.4th ed. New York: Marcel Dekker; 2003.
[17] Hopper E, Turton CH. An empirical study of meta-heuristics applied to 2D rectangular bin packing. Studia Informatica 2002;2(1):77–92.
[18] Laguna M, Marti R. GRASP and path relinking for 2-layer straight line crossing minimization. INFORMS Journal on Computing 1999;11:44–52.
[19] Laguna M, Marti R. Scatter search: methodology and implementations in C. Boston: Kluwer; 2003.
[20] Martello S, Vigo D. Exact solution of the two-dimensional finite bin packing problem. Management Science 1998;44(1):388–99.
[21] Martello S, Pisinger D, Vigo D. The three-dimensional bin packing problem. Operations Research 2000;48:256–67.
[22] Monaci M. Algorithms for packing and scheduling problems. PhD thesis, University of Bologna; 2002.
[23] Ortmann FG, Ntene N, van Vuuren JH. New and improved level heuristics for the rectangular strip packing and variable-sized bin packing problems. European Journal of Operational Research 2010;203(2):306–35.
[24] Parreño F, Alvarez-Valdes R, Oliveira JF, Tamarit JM. A hybrid GRASP/VND algorithm for two- and three-dimensional bin packing. Annals of Operations Research 2010;179:203–20.
[25] Pisinger D, Sigurd M. The two-dimensional bin packing problem with variable bin sizes and costs. Discrete Optimization 2005;2(2):154–67.
[26] Prais M, Ribeiro CC. Reactive GRASP: an application to a matrix decomposition problem in TDMA traffic assignment. INFORMS Journal on Computing 2000;12:164–76.
[27] Resende MGC, Ribeiro CC. Greedy randomized adaptive search procedures. In: Glover F, Kochenberger G, editors. Handbook of Metaheuristics. Boston: Kluwer; 2003. p. 219–49.
[28] Resende MGC, Werneck RF. A hybrid heuristic for the p-median problem. Journal of Heuristics 2004;10:59–88.
[29] Resende MGC, Marti R, Gallego M, Duarte A. GRASP and Path Relinking for the max–min diversity problem. Computers and Operations Research 2010;37(3): 498–508.
[30] Seiden SS, van Stee R. New bounds for multidimensional packing. Algorithmica 2003;36(3):261–93.
[31] Waescher G, Haussner H, Schumann H. An improved typology of cutting and packing problems. European Journal of Operational Research 2007;183(3): 1109–30.
[32] Wu Y, Li W, Goh M, de Souza R. Three-dimensional bin packing problem with variable bin height. European Journal of Operational Research 2010;202(2): 347–55.