A maximal-space algorithm for the container loading problem

F. Parreño [‡], R. Alvarez-Valdes [†], J.F. Oliveira [§][*], J.M. Tamarit [†],

[†] University of Valencia, Department of Statistics and Operations Research, Burjassot, Valencia, Spain
[§] Faculty of Engineering, University of Porto, Portugal
[*] INESC Porto – Instituto de Engenharia de Sistemas e Computadores do Porto, Portugal
[‡] University of Castilla-La Mancha. Department of Computer Science, Albacete, Spain

Abstract

In this paper a greedy randomized adaptive search procedure (GRASP) for the container loading problem is presented. This approach is based on a constructive block heuristic that builds upon the concept of maximal-space, a non-disjoint representation of the free space in a container.

This new algorithm is extensively tested over the complete set of Bischoff and Ratcliff problems, ranging from weakly heterogeneous to strongly heterogeneous cargo, and outperforms all the known non-parallel approaches that, partially or completely, have used this set of test problems. When comparing against parallel algorithms, it is better on average but not for every class of problem. In terms of efficiency, this approach runs in much less computing time than that required by parallel methods. Thorough computational experiments concerning the evaluation of the impact of algorithm design choices and internal parameters on the overall efficiency of this new approach are also presented.

Keywords: Container loading; 3D packing; heuristics; GRASP

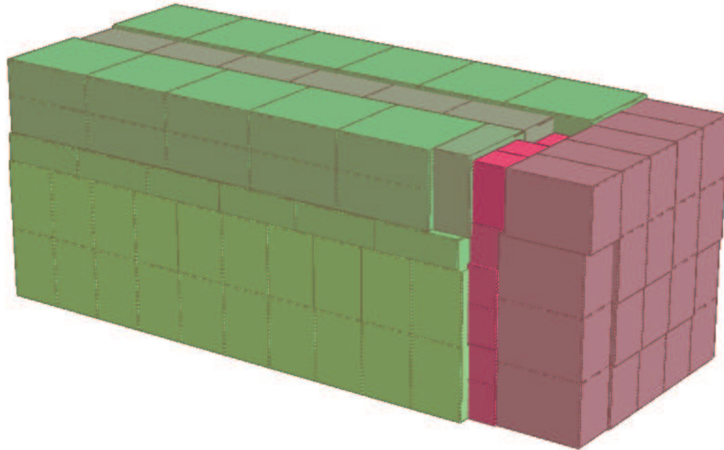


Figure 1: *A solution of Instance BR_3_79 [2]*

1 Introduction

The Single Container Loading Problem (CLP) is a three-dimensional packing problem in which a large parallelepiped has to be filled with smaller parallelepipeds, available in different sizes and limited quantities, so that empty space is minimized (Figure 1). Under the improved typology for cutting and packing problems proposed by Wäscher et al. [21], the CLP can be classified as a three-dimensional rectangular single large object placement problem (3-dimensional rectangular SLOPP). This is an NP-hard problem as the NP-hard one-dimensional knapsack problem can be transformed into the 3D SLOPP.

From the applications point of view, this problem arises in practice whenever containers or trucks have to be filled/loaded with boxes, so that the usage of the container is maximized. The minimization of empty space inside the containers is not only an economic requirement but also an ecological issue, given the impact that goods transportation has on the global effect of human activities in our planet's sustainability.

In 1980 George and Robinson [14] proposed the first fairly sophisticated heuristic for the container loading problem. It was a wall-building procedure and since then many variants and improvements have been developed [1], including, more recently, meta-heuristic approaches based on this heuristic [16]. In 1997 Gehring and Bortfeldt [12] presented a genetic algorithm for the container loading problem, based on a column-building procedure. This was

the first of an important series of papers by those same authors who, between 1998 and 2002, presented a Tabu Search [5], a hybrid Genetic Algorithm [6] and a parallel version of the genetic algorithm [13]. In the same line of work, Bortfeldt, Gehring and Mack developed a parallel tabu search algorithm [7] and a parallel hybrid local search algorithm which combines simulated annealing and tabu search, their best algorithm so far [15]. Eley [10] proposes an algorithm that combines a greedy heuristic, which generates blocks of boxes, with a tree-search procedure. Moura and Oliveira [16] develop a GRASP algorithm based on a modified version of George and Robinson [14] constructive algorithm. All the previously mentioned papers have in common the use of a set of 1500 problems, distributed over 15 classes with different characteristics as benchmark problems for the computational experiments. These problems were generated by Bischoff and Ratcliff [2]. Pisinger [17] presents a wall-building heuristic, which seems to perform quite well, but unfortunately the author does not use Bischoff and Ratcliff's problems for his computational experiments.

When speaking about real-world container loading problems space usage is the most important objective, but other issues have to be taken into account, such as cargo stability, multi-drop loads or weight distribution ([2], [4], [8], [20]). Among these additional considerations, cargo stability is the most important one. Sometimes it is explicitly taken into account in the design of the algorithms and in the definition of feasible solutions for the container loading problem. At other times the algorithms' results are also evaluated against standard measures of cargo stability. However, when container volume usage is very high, stability almost becomes a consequence of this high cargo compactness. Some freight companies value space utilization so highly that they use foam pieces to fill the remaining gaps and increase stability without compromising space usage.

In this paper we present a new algorithm for the container loading problem that is based on an original heuristic enhanced by a GRASP solution space search strategy. At the core of this approach is the concept of maximal-spaces that explains the efficiency of the algorithm. When considering Bischoff and Ratcliff's problems, this new algorithm outperforms the previously published approaches, taking much less computational time, mainly in the harder classes of problems.

The remainder of this paper is organized as follows: in the next section the basic constructive algorithm, based on the concept of maximal-space, is presented in its several variants. In Section 3 the GRASP implementation is presented in its constitutive parts, and in Section 4 thorough computational experiments are described and results presented. Finally, in Section 5 conclusions are drawn.

2 A constructive algorithm

At the base of any GRASP algorithm is a constructive heuristic algorithm. The constructive algorithm proposed in this paper is a block heuristic. The box blocks may take the shape of columns or layers. A layer is a rectangular arrangement of boxes of the same type, in rows and columns, filling one side of the empty space. Other authors have used the term wall to refer to the same concept. We prefer the more general term layer because it can be put vertically or horizontally in the empty space. As usual in block heuristics, the feasible placement positions are described as lists of empty spaces and each time a block is placed in an empty space, new spaces are generated. The present block algorithm differs from the existing approaches because it represents the feasible placement points as a set of non-disjoint empty spaces, the maximal-spaces. This representation brings additional complexity to space management procedures but induces increased flexibility and quality in container loading problem solutions. Additionally, this heuristic starts to place boxes from the eight corners of the selected empty space, according to well-defined selection criteria. Details on this new constructive heuristic are given in the rest of this section .

We follow an iterative process in which we combine two elements: a list $\mathcal{B}$ of types of boxes still to be packed, initially the complete list of boxes, and a list $\mathcal{L}$ of empty maximal-spaces, initially containing only the container C. At each step, a maximal-space is chosen from $\mathcal{L}$ and, from the boxes in $\mathcal{B}$ fitting into it, a type of box and a configuration of boxes of this type are chosen to be packed. That usually produces new maximal-spaces going into $\mathcal{L}$ and the process goes on until $\mathcal{L} = \emptyset$ or none of the remaining boxes fit into one of the remaining maximal-spaces.

- Step 0: *Initialization*

  $\mathcal{L}$={C}, the set of empty maximal-spaces.
  $\mathcal{B} = \{b_1, b_2, \ldots, b_m\}$, the set of types of boxes still to be packed.
  $q_i = n_i$ (number of boxes of type i to be packed).
  $\mathcal{P} = \emptyset$, the set of boxes already packed.

- Step 1: *Choosing the maximal-space in $\mathcal{L}$*

  We have a list $\mathcal{L}$ of empty maximal-spaces. These spaces are called maximal because at each step they are the largest empty parallelepiped that can be considered for filling with rectangular boxes. We can see an example for a 2D problem in Figure 2. Initially we have an empty rectangle and when we cut a piece at its bottom left corner, two maximal-spaces are generated. These spaces do not have to be disjoint. In the

same Figure 2 we see two more steps of the filling process with the maximal-spaces generated.

Working with maximal-spaces has two main advantages. First, we do not have to decide which disjoint spaces to generate, and then we do not need to combine disjoint spaces into new ones in order to accommodate more boxes. We represent a maximal-space by the vertices with minimum and maximum coordinates.

To choose a maximal-space, a measure of its distance to the container's corners is used. For every two points in R^3, $a = (x_1, y_1, z_1)$ and $b = (x_2, y_2, z_2)$, we define the distance $d(a, b)$ as the vector of components $|x_1 - x_2|$, $|y_1 - y_2|$ and $|z_1 - z_2|$ ordered in non-decreasing order. For instance, if $a = (3, 3, 2)$ and $b = (0, 5, 10)$, $d(a, b) = (2, 3, 8)$, resulting from ordering the differences 3, 2, 8 in non-decreasing order. For each new maximal-space, we compute the distance from every corner of the space to the corner of the container nearest to it and keep the minimum distance in the lexicographical order.

$$d(S) = min\{d(a, c), a \text{ vertex of } S,\ c \text{ vertex of container } C\}$$

For instance, if we have the maximal-space $S_1 = \{(4, 4, 2),\ (6, 6, 10)\}$ in a (10,10,10) container, the corner of S_1 nearest to a corner of the container is (6,6,10) and the distance to that corner is $d(S_1) = (0, 4, 4)$. At each step we take the maximal-space with the minimum distance to a corner of the container and use the volume of the space as a tie-breaker. If we have a second maximal-space $S_2 = \{(1, 1, 2), (4, 4, 9)\}$, we have $dist(S_1) = (0, 4, 4)$ and $dist(S_2) = (1, 1, 1)$. From among these spaces we would choose space S_1 to be filled with boxes. At each step, the chosen space will be denoted by S^*.

The corner of the maximal-space with the lowest distance to a corner of the container will be the corner in which the boxes will be packed. The reason behind that decision is to first fill the corners of the container, then its sides and finally the inner space.

- Step 2: *Choosing the boxes to pack*

  Once a maximal-space S^* has been chosen, we consider the types of boxes i of $\mathcal{B}$ fitting into S^* in order to choose which one to pack. If $q_i > 1$, we consider the possibility of packing a column or a layer, that is, packing several copies of the box arranged in rows and columns, such that the number of boxes in the block does not exceed q_i. In Figures 3 and 4 we can see different possibilities for packing a box type i with

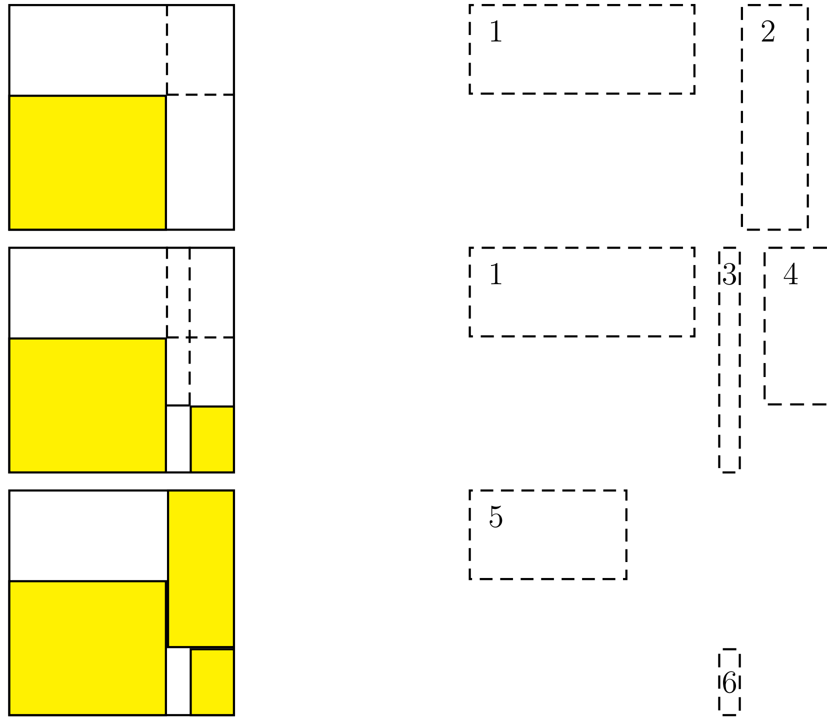
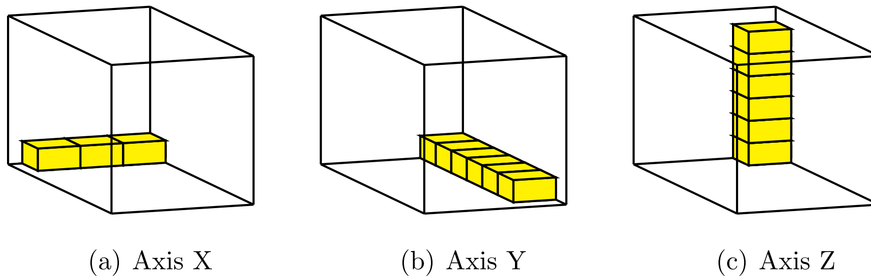


Figure 2: *Maximal-spaces in two dimensions*

$q_i = 12$. Figure 3 shows alternatives for packing a column of boxes, while in Figure 4 we consider different ways of packing a layer with this box type. If some rotations of the pieces are allowed, the corresponding configurations will be considered.



(a) Axis X (b) Axis Y (c) Axis Z

Figure 3: *Three different alternatives for a column*

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

i The block of boxes producing the largest increase in the objective function. This is a greedy criterion in which the space is filled with

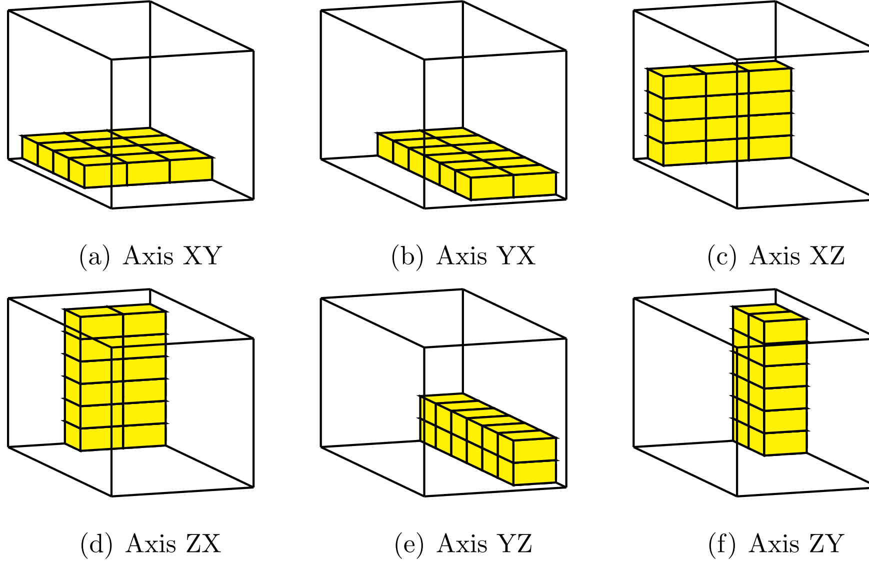


Figure 4: *Six different alternatives for a layer*

the block producing the largest increase in the volume occupied by boxes.

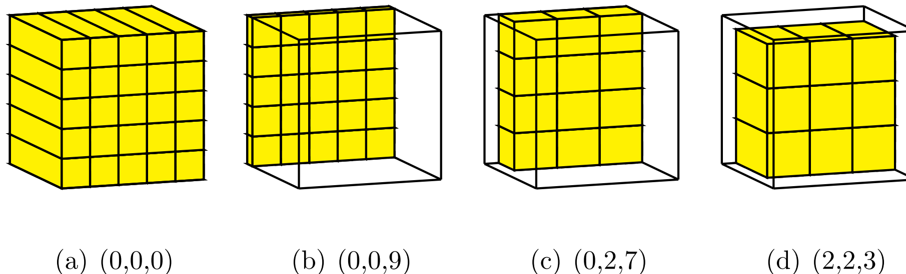
ii The block of boxes which fits best into the maximal-space. We compute the distance from each side of the block to each side of the maximal-space and order these distances in a vector in non-decreasing order. The block is chosen using again the lexicographical order. We can see an example in Figure 5, in which we consider several alternatives for filling an empty space of $(20, 20, 10)$. The block (a) completely fills the space and its distance is (0,0,0); the block (b) completely fills one side of the space and its distance is (0,0,9); for the block (c), only its height matches the space height and its distance is (0,2,7); finally, none of the dimensions of block (d) matches the space dimensions and its distance is (2,2,3).

For both criteria, ties are broken by choosing the configuration with the minimum number of boxes. In Section 4 we test both criteria and make a decision about which one will be used in the final implementation.

After choosing a block with r_i boxes, we update $\mathcal{P}$ with the type i and the number of boxes packed and set $q_i = q_i - r_i$. If $q_i = 0$, we remove piece i from the list $\mathcal{B}$.

- Step 3: *Updating the list $\mathcal{L}$*

Unless the block fits into space S^* exactly, packing the block produces



(a) (0,0,0) (b) (0,0,9) (c) (0,2,7) (d) (2,2,3)

Figure 5: *Best fit of the empty space*

new empty maximal-spaces, which will replace S^* in the list $\mathcal{L}$. Moreover, as the maximal-spaces are not disjoint, the block being packed can intersect with other maximal-spaces which will have to be reduced. Therefore, we have to update the list $\mathcal{L}$. Once the new spaces have been added and some of the existing ones modified, we check the list and eliminate possible inclusions. For the sake of simplicity, Figure 2 illustrates this process for a two-dimensional problem. When the second box is packed in maximal-space 2, this space is eliminated from the list and is replaced by two new spaces 3 and 4. When the third piece is packed into maximal-space 4, as it is completely filled it just disappears, but spaces 1 and 3 are also partially filled and must be reduced, producing spaces 5 and 6.

3 GRASP Algorithm

The GRASP algorithm was developed by Feo and Resende [11] to solve hard combinatorial problems. For an updated introduction, refer to Resende and Ribeiro [19]. GRASP is an iterative procedure combining a constructive phase and an improvement phase. In the constructive phase a solution is built step by step, adding elements to a partial solution. In order to choose the element to be added, a greedy function is computed, which is dynamically adapted as the partial solution is built. However, the selection of the element is not deterministic. A randomization strategy is added to obtain different solutions at each iteration. The improvement phase, usually consisting of a simple local search, follows the constructive phase.

3.1 The constructive phase

In our algorithm the constructive phase corresponds to the constructive algorithm described in Section 2, introducing a randomization procedure

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 chosen objective function (the best increase of volume or the best fit into the space). 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 [18], 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. The procedure is described in Figure 6, following Delorme et al. [9]. In this figure the parameter α is fixed at 10, as in [18].

3.2 Improvement phase

Each solution built in the constructive phase is the starting point for a procedure in which we try to improve the solution. The procedure proposed here consists of eliminating the final $k\%$ blocks of the solution (for instance, the final 50%) and filling the empty spaces with the deterministic constructive algorithm. At Step 2 of the constructive algorithm we again consider the two different objective functions described in Section 2, the largest increase in the volume used and the best fit into the empty maximal-space being considered.

The improvement phase is only called if the objective function value of the solution of the constructive phase $V \geq V_{worst} + 0.50(V_{best} - V_{worst})$, where V_{worst} and V_{best} correspond to the worst and best objective function values of solutions obtained in previous GRASP iterations.

4 Computational experiments

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 most recent and efficient algorithms proposed for the container loading problem.

Initialization:

$\mathcal{D} = \{0.1, 0.2, \ldots, 0.9\}$, set of possible values for δ

$V_{best} = 0; \quad V_{worst} = \infty$

$n_{\delta^*} = 0$, number of iterations with δ^*, $\forall\, \delta^* \in \mathcal{D}$.

$Sum_{\delta^*} = 0$, sum of values of solutions obtained with δ^*.

$P(\delta = \delta^*) = p_{\delta^*} = 1/|\mathcal{D}|\,, \forall\, \delta^* \in \mathcal{D}$

$numIter = 0$

While $(numIter < maxIter)$

{

Choose δ^* from $\mathcal{D}$ with probability p_{δ^*}.

$n_{\delta^*} = n_{\delta^*} + 1$

$numIter = numIter + 1$

Apply Constructive Phase with δ^*obtaining solution S with objective value V

Apply Improvement Phase obtaining solution S' with value V'

If $V^{'} > V_{best}$ then $V_{best} = V^{'}$.

If $V^{'} < V_{worst}$ then $V_{worst} = V^{'}$

$Sum_{\delta^*} = Sum_{\delta^*} + V^{'}$

If $mod(numIter, 500) == 0$:

$$eval_\delta = \left(\frac{mean_\delta - V_{worst}}{V_{best} - V_{worst}}\right)^\alpha \quad \forall \delta \in \mathcal{D}$$

$$p_\delta = \frac{eval_\delta}{\left(\sum_{\delta' \in \mathcal{D}} eval_{\delta'}\right)} \quad \forall \delta \in \mathcal{D}$$

}

Figure 6: *Reactive Grasp*

4.1 Test problems

The tests were performed on 1500 problems generated by Bischoff and Ratcliff [2] and Davies and Bischoff [8]. The 1500 instances are organized into 15 classes of 100 instances each. The number of box types increases from 3 in BR1 to 100 in BR15. Therefore, this set covers a wide range of situations, from weakly heterogenous to strongly heterogenous problems. The number of boxes of each type decreases from an average of 50.2 boxes per type in BR1 to only 1.30 in BR15. The total volume of the boxes is on average 99.46% of the capacity of the container, but as the boxes' dimensions have been generated independently of the container's dimensions, there is no guarantee that all the boxes of one instance can actually fit into the container. Therefore, this figure has to be used as an upper bound (probably quite loose) on the maximum percentage of volume which the boxes can fill.

Following the presentation of authors with whom we compare our results, in the tables of this computational experiment section for each class we present the average value of the solutions obtained on its 100 instances.

4.2 Choosing the best strategies

Table 1 compares the results obtained by the constructive algorithm of Section 2, when the block to pack is selected according to each one of the four strategies considered in Step 2, obtained by combining the two objective functions (volume, best-fit) with the two types of configurations (columns, layers). This table, like all other tables in this section, shows the percentages of container volume occupied by boxes in the solution. The tables also show the aggregate results for just the first seven classes, in order to compare with authors who only use these classes in their computational experience. The results of Table 1 have been obtained on the complete set of 1500 test instances. Therefore, they can be directly compared with other algorithms. If we compare them with the constructive and metaheuristic algorithms in Table 5, we can see that our initial simple constructive procedure can already compete with these algorithms, especially for strongly heterogeneous classes.

In order to choose the best combination of configuration and objective function we apply statistical analysis to the data summarized in Table 1. First, we use multivariate analysis of variance for repeated measures, considering that for each instance we have the results obtained by the four strategies. These strategies define the intra-subject factor, while the instance classes define the inter-subject factor. This analysis allows us to consider separately the effect of the strategies and the instance classes on the results obtained and also to make pairwise comparisons. As this analysis requires some

normality conditions, we also perform a non-parametric analysis for related measures, the Friedman test, and the Wilcoxon test for pairwise comparisons. Both types of tests show that the effect of the different algorithmic strategies is statistically very significant ($p < 0.001$). The pairwise comparisons also show significant differences for each pair of strategies. These conclusions are illustrated in Figure 7. The use of the criterion Best_Fit for the objective function produces results which are significantly worse than using an objective function based on the increase in the volume occupied. With this objective function, building layers is significantly better than building columns, but this difference only corresponds to the weakly heterogeneous instances of the first classes. For strongly heterogeneous problems, with few copies of each box type, building layers is almost equivalent to building columns and the results are very similar. In fact, over the 1500 instances, building layers gets better results in 833 cases, while building columns obtains better results in 623 and there are 44 ties. A first consequence of this analysis is that we will use the objective function based on the increase of volume and we will build layers of boxes when filling empty spaces.

	Columns		**Layers**	
	Best-fit	**Volume**	**Best-fit**	**Volume**
BR1	82,36	82,14	**85,33**	84,34
BR2	82,22	82,93	**85,15**	84,57
BR3	81,96	84,53	85,40	**85,92**
BR4	81,97	85,19	85,48	**86,71**
BR5	82,39	85,16	85,49	**86,52**
BR6	82,83	85,55	84,70	**86,87**
BR7	83,08	86,16	84,75	**86,31**
BR8	83,35	86,23	84,12	**86,35**
BR9	83,75	86,00	83,89	**86,29**
BR10	83,32	**86,04**	84,18	85,84
BR11	83,54	85,90	83,59	**85,99**
BR12	83,52	85,69	83,94	**85,82**
BR13	83,62	85,47	83,75	**85,58**
BR14	83,30	85,45	83,71	**85,50**
BR15	82,82	85,67	83,93	**85,73**
Mean B1-B7	82,40	84,52	85,19	**85,89**
Overall mean	82,94	85,21	84,49	**85,89**

***The best values appear in bold**

Table 1: *Constructive phase: selecting the configuration*

We have also tested whether the new distance defined in Step 1 of the constructive algorithm, denoted as lexicographical distance, really is better than the usual Euclidean distance. Figure 8 shows the results obtained with both distances. The statistical tests mentioned above, when applied to these data, show that using the new distance for selecting the space to fill produces significantly better results that using the classical Euclidean distance ($p < 0.001$).

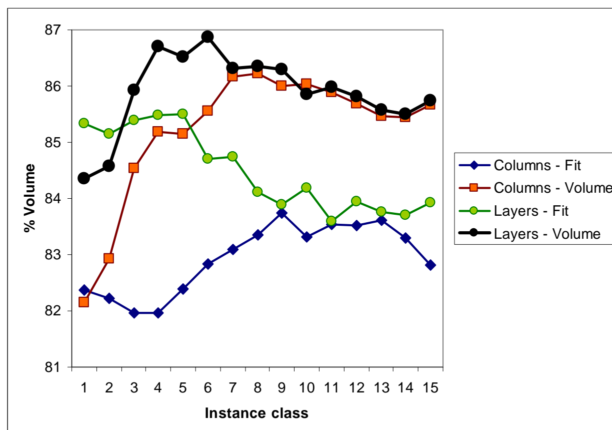


Figure 7: *Comparing objectives and strategies*

In order to choose the best strategies for the GRASP algorithm, we have done a limited computational study using only the first 10 instances of each class of problems and setting a limit of 5000 iterations. Hence, the results of Tables 2 and 3 cannot be compared with other algorithms.

Table 2 explores possible values for the percentage of removed blocks in the improvement procedure. The values considered are 10%, 30%, 50%, 70% and 90%, as well as a strategy consisting of choosing this value at random from [25%,75%]. We again use the parametric ANOVA procedure and the non-parametric Friedman test to compare these values. The pairwise comparisons distinguish strategies removing 10% and 30% as significantly worse than the others, but between the other four values, 50%, 70%, 90% or at random from [25%,75%], there are no significant differences. Therefore, we choose the value 50% because it requires less computational effort and therefore less computing time.

The good results obtained when removing high percentages of the blocks may seem strange at first sight. One explanation could be that the quality of the solutions depends critically on the layout of the first boxes packed. The randomized constructive phase provides a wide set of alternatives for these first packing steps. For the remaining boxes, at each iteration we have two alternatives: the one obtained by continuing with the randomized procedure and the one obtained by completing the solution in a deterministic way. The overall results show a significant improvement when the two alternatives are combined.

Finally, in Table 3 we compare three different improvement methods, differing in the objective function used when filling the container again with

Figure 8: *Comparing distances for selecting the space to fill*

the deterministic algorithm. The first one uses volume, the second the best-fit criterion and the third uses both criteria, repeating the procedure twice. The statistical tests show that there are very significant differences between them, but the main reason for that result is the relatively poor performance of the best-fit criterion, as can be seen in Figure 9. If we eliminate this alternative and compare the other two strategies, the differences are not clear. The Wilcoxon test gives a $p-value$ of 0.039. This is mainly due to the fact that in classes 7 to 15 both strategies produce very similar results. However, Figure 9 shows and the tests confirm that for classes 1 to 6 there is a significant difference in favor of the strategy using both criteria. As the increase in running time is not too large and we are looking for an algorithm working well for all classes, we decided to use this strategy in the final implementation.

4.3 Studying the random component of the algorithm

The GRASP algorithm has a random component in the constructive phase. In the experiments in the previous subsection each instance was run only once, but in order to assess the effect of the associated randomness, we run the algorithm 10 times for each of the first 10 instances of each class, with a limit of 5000 iterations per run. The results are summarized in Table 4. The second column shows the results when running the algorithm just once. Columns 3 to 6 show the results for 5 runs and columns 7 to 8 the results of 10 runs. Finally, column 10 shows the results obtained by running the algorithm only once, but with a limit of 50000 iterations.

The ranges of variation between the best and the worst solutions obtained

Problem	Removing Pieces 10%	30%	50%	70%	90%	Random 25%-75%
BR1	92,55	92,36	**92,65**	92,63	92,27	92,64
BR2	93,11	93,42	**93,44**	93,38	93,25	**93,44**
BR3	92,88	93,05	93,32	93,25	93,10	**93,33**
BR4	92,60	92,91	92,87	92,91	92,56	**93,00**
BR5	92,31	92,52	**92,64**	92,51	92,11	92,41
BR6	91,76	92,11	92,44	**92,49**	91,85	92,37
BR7	91,12	91,34	91,59	**91,73**	91,38	91,46
BR8	89,69	90,21	90,83	**91,09**	90,86	91,05
BR9	88,97	89,76	90,42	**90,63**	90,66	90,33
BR10	88,36	89,16	89,79	89,78	**90,19**	89,84
BR11	88,02	88,44	89,34	**89,69**	89,66	89,31
BR12	87,15	88,12	88,85	89,13	**89,24**	88,95
BR13	86,72	87,67	88,15	88,25	**88,91**	88,31
BR14	86,82	87,62	88,36	88,45	**88,71**	88,21
BR15	86,85	87,89	88,25	88,55	**88,75**	88,29
Mean B1-B7	92,33	92,53	**92,71**	92,70	92,36	92,67
Overall mean	89,93	90,44	90,86	**90,96**	90,90	90,86

***The best values appear in bold**

Table 2: *Percentage of removed pieces*

	Without improving	Objective function for the improvement method					
		Volume		Best-fit		Volume+Best-fit	
Problem	**Vol.(%)**	**Vol.(%)**	**Time**	**Vol.(%)**	**Time**	**Vol.(%)**	**Time**
BR1	92,09	92,65	0,88	92,84	1,04	**92,95**	1,11
BR2	92,86	93,46	1,85	93,79	2,17	**93,95**	2,39
BR3	92,24	93,32	3,50	93,21	3,87	**93,54**	4,39
BR4	91,95	92,87	5,06	**93,21**	5,63	93,05	6,53
BR5	91,36	92,59	6,43	**93,02**	7,09	93,01	8,21
BR6	91,11	92,41	9,40	92,20	10,42	**92,72**	12,14
BR7	90,13	91,58	14,51	91,34	15,99	**91,62**	18,80
BR8	89,20	**90,87**	29,65	90,39	32,35	90,74	38,69
BR9	88,08	90,42	45,74	89,61	50,16	**90,43**	60,44
BR10	87,31	**89,79**	71,20	89,02	77,54	89,69	95,94
BR11	86,83	**89,37**	98,60	88,64	107,33	89,29	135,12
BR12	86,72	**88,95**	134,79	88,09	147,07	**88,95**	182,96
BR13	86,16	88,15	176,94	87,91	194,19	**88,22**	240,71
BR14	85,99	**88,36**	230,60	87,34	253,25	88,25	326,26
BR15	86,26	**88,30**	284,83	87,33	311,56	**88,30**	387,41
Mean B1-B7	91,68	92,70	5,95	92,80	6,60	**92,98**	7,88
Overall mean	89,22	90,87	74,27	90,53	81,31	**90,98**	99,75

***The best values appear in bold**

Table 3: *Results of improvement methods*

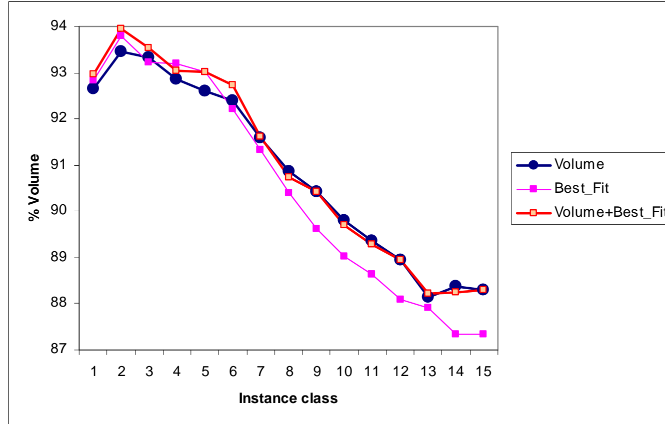


Figure 9: *Comparing improvement methods*

when running the algorithm 5 or 10 times can be obtained by subtracting columns 2 and 3, and columns 5 and 6. The average range for 5 runs is 0.80% and for 10 runs is 1.06%. Even with this iteration limit of 5000 iterations, the algorithm is very stable. If the iteration limit is increased, this variation is lower still.

Table 4 also indicates, and the statistical tests confirm, that if we choose the average of the occupied volume as the measure for showing the performance of the algorithms, the average volumes obtained by running the algorithm once do not differ significantly from those obtained if we run the algorithm five or ten times and then calculate the averages of those five or ten results (columns 5 and 9).

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 used by running it once with a limit of 50000 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 δ are updated after a given number of iterations. The results of the previous iterations should guide this procedure to favor values of δ producing best solutions. Therefore, the results obtained using the algorithm once up to 50000 iterations should be at least equal to and possibly better than those obtained by running it 10 independent times. The results of Table 4 show a slight advantage of the algorithm with 50000 iterations, but the statistical tests performed on the results summarized in columns 6 and 10 do not

	1 run 5000 iter	5 runs 5000 iter			10 runs 5000 iter			1 run 50000 iter
		Min	Max	Mean	Min	Max	Mean	
BR1	92.95	92.40	93.11	92.71	92.29	93.41	92.82	93.42
BR2	93.95	93.68	94.22	94.00	93,51	94.28	93.95	94.28
BR3	93.54	93.08	94.04	93.58	92.97	94.10	93.46	94.05
BR4	93.05	92.71	93.61	93.25	92.62	93.82	93.26	94.03
BR5	93.01	92.59	93.48	93.00	92,57	93.68	93.06	93.79
BR6	92.72	92.32	93.21	92.72	92,12	93.31	92.66	93.21
BR7	91.62	91.32	92.23	91.70	91,24	92.51	91.74	92.55
BR8	90.74	90.38	91.37	90.81	90.33	91.71	90.87	91.66
BR9	90.43	90.11	90.89	90.45	90.03	91.05	90.45	91.12
BR10	89.69	89.38	90.12	89.70	89.30	90.21	89.72	90.50
BR11	89.29	89.05	89.85	89.36	88.93	90.01	89.37	90.02
BR12	88.95	88.57	89.35	88.95	88.47	89.61	88.95	89.43
BR13	88.22	87.99	88.77	88.34	87.89	88.82	88.28	88.79
BR14	88.25	87.94	88.66	88.24	87.86	88.73	88.21	88.77
BR15	88.23	88.08	88.66	88.33	88.04	88.79	88.35	88.89
Mean BR1-BR7	92.98	92.59	93.41	92.99	92.48	93.59	92.99	93.62
Mean BR1-BR15	90.98	90.64	91.44	91.01	90.54	91.60	91.01	91.63

Table 4: *Studying the algorithm's randomness*

show any 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 50000 iterations. Second, the learning mechanism of the Reactive GRASP does not seem to have any significant effect on the long term quality of the solutions. The preliminary experiments we ran showed that this way of determining δ performed better than other alternatives, but its effect seems to diminish in the long run.

4.4 Comparison with other algorithms

As a consequence of the results obtained in the exploratory experiments described in previous subsections, for the following computational tests the constructive phase of the GRASP will use the lexicographical distance at Step 1, building layers of boxes at Step 2 and an objective function based on the increase of volume. The value of the parameter δ in the randomization procedure will be determined by the Reactive GRASP strategy. In the improvement phase, the final 50% of the blocks will be removed and the empty space filled twice, once with each objective function.

The way in which our algorithm packs the boxes, starting by packing from the corners, then the sides and finally the center of the container, does not guarantee cargo stability and does not produce a workable packing sequence. Therefore, a postprocessing phase in which the solution is compacted is absolutely necessary. We have developed a compacting procedure which takes each box from bottom to top and tries to move it down, along axis Z, until it

is totally or partially supported by another box. Then, a similar procedure takes boxes from the end to the front of the container and tries to move them to the end, along axis Y. Finally, there is a procedure moving them from left to right, along axis X. The three procedures are called iteratively while there are boxes which have been moved. When the compacting phase is finished, the empty spaces are checked for the possibility of packing some of the unpacked boxes. In the final solution every box is supported totally or partially by other boxes and the boxes are given in an order in which they can really be packed into the container.

The complete computational results on the whole set of 1500 instances appear in Tables 5 and 6. These two tables include a direct comparison with the results of the best algorithms proposed in the literature, which have benchmarked themselves against these test problems. Therefore, we compare our algorithm against 16 approaches. Among them we can distinguish two types of algorithms: algorithms that pack the boxes so that they are completely supported by other boxes; and algorithms that do not consider this constraint. In the first group we have the following algorithms:

- **H_BR**: a constructive algorithm by Bischoff and Ratcliff [2];
- **H_B_al**: a constructive algorithm by Bischoff, Janetz and Ratcliff [3];
- **H_B**: a heuristic approach by Bischoff [4];
- **GA_GB**: a genetic algorithm by Gehring and Bortfeldt [12];
- **TS_BG**: a tabu search approach by Bortfeldt and Gehring [5];
- **H_E**: a greedy constructive algorithm with an improvement phase by Eley [10];
- **G_M**: a GRASP approach by Moura and Oliveira [16].

In the second group we have the following algorithms:

- **HGA_BG**: a hybrid genetic algorithm by Gehring and Bortfeldt [6];
- **PGA_GB**: a parallel genetic algorithm by Gehring and Bortfeldt [13];
- **PTSA**: a parallel tabu search algorithm by Bortfeldt, Gehring and Mack [7];
- **TSA**: a tabu search algorithm by Bortfeldt, Gehring and Mack [7];

- **SA**: a simulated annealing algorithm by Mack, Bortfeldt and Gehring [15];
- **HYB**: a hybrid algorithm by Mack, Bortfeldt and Gehring [15];
- **PSA**: a parallel simulated annealing algorithm by Mack, Bortfeldt and Gehring [15];
- **PHYB**: a parallel hybrid algorithm by Mack, Bortfeldt and Gehring [15];
- **PHYB_XL**: a massive parallel hybrid algorithm by Mack, Bortfeldt and Gehring [15];

In Table 5 we compare our GRASP algorithm (5000 iterations) with fast constructive procedures and metaheuristic algorithms requiring moderate computing times. The computing times cannot be compared directly because each author has used a different computer. In general, the running times for all approaches are lower for the weakly heterogenous problems and increase with the number of different box types. The times required by algorithms H_B_al [3] and H_BR [2] are not reported, but obviously these algorithms are very fast. H_B [4] was run on a Pentium IV at 1.7 GHz, with times ranging on average from 26.4 seconds for BR1 to 321 seconds for BR15. H_E [10] ran on a Pentium at 200 Mhz with a time limit of 600 seconds which was reached only for 26 out of the 700 instances of BR1-BR7. Algorithms GA_GB, TS_GB and HGA_GB were run on a Pentium at 400 Mhz, with average times of 12, 242 and 316 seconds respectively. The parallel PGA ran on five Pentiums at 400 Mhz and took on average 183 seconds. Finally, G_M was run on a Pentium IV at 2.4 GHz, with an average time of 69 seconds.

We can compare the fast constructive algorithms H_B_al and H_BR with our constructive algorithm, whose results appear in the last column of Table 1, using a t-test. We have tested for each class whether the average volume obtained by our algorithm is significantly better than the reported averages of H_B_al and H_BR. The statistical tests conclude that for each class our constructive algorithm outperforms both procedures, except for class BR1, for which our results do not differ significantly from those obtained by H_BR ($p = 0.12$).

The other algorithms in Table 5 have been compared with our GRASP algorithm. Again, we have compared the average results obtained with our algorithm on the 100 instances of each class with the averages reported for each class by the other algorithms. The statistical tests confirm that our GRASP algorithm gets significantly better results for every class. The differences are more important for strongly heterogeneous problems. In fact,

as Bortfeldt et al. state in their papers, their algorithms are best suited for weakly heterogeneous classes. In particular, their Tabu Search algorithm, TS_BG, obtains very good results in relatively short times in the first seven classes. In any case, the Bortfeldt et al. algorithms will be more adequately compared in Table 6, where their more recent and powerful procedures are considered.

Table 6 compares GRASP with more recent metaheuristics that require longer computing times, especially in their parallel versions. In this case, we have allowed our algorithm to run up to an extremely high number of iterations, 200000, to assess its performance for longer running times. The last row of the table shows the reported average running times. They are included only as a reference, because the algorithms have been run on different computers. Mack et al. [15] use a Pentium-PC at 2Ghz for the serial procedures, a LAN of four of these computers for the parallel implementation of PSA and PHYB and a LAN of sixty-four computers for the PHYB.XL implementation.

An approximate comparison, according to the reported running times, would be to compare our version of the GRASP algorithm with 50000 iterations with the serial methods and GRASP with 200000 iterations with the parallel methods. In the first set of comparisons, we compare the best serial method, HYB, with our GRASP with 50000 iterations. The statistical tests show that our algorithm outperforms the serial methods for all classes, except for BR1 in which the $p - value$ is 0.052 and the equality cannot be rejected at a level $\alpha = 0.05$. In a second series of comparisons, we compare the best results of the three parallel methods, PTSA, PSA and PHYB with our GRASP with 200000 iterations. Again, our algorithm produces significantly better results for all classes ($p < 0.01$), except for BR1 in which the $p - value$ is 0.114. Finally, we compare the massively parallel method PHYB.XL with GRASP with 200000 iterations. In this case, equality cannot be rejected for classes BR1, BR2 and BR5 (with $p - values$ of 0.48, 0,52 and 0.66), PHYB.XL is better for classes BR3 and BR4 and GRASP is better for classes BR6 and BR7. Again, the Mack et al. algorithms work very well for weakly heterogeneous classes but our procedure obtains progressively better results when heterogeneity increases.

5 Conclusions

We have developed a new GRASP algorithm which obtains good results for all classes of test instances, from weakly to strongly heterogeneous problems, especially for the latter. We think that these good results are due to three

Problem class	H_B_al	H_BR	GA_GB	TS_BG	HGA_BG	PGA_GB	H_B	H_E	G_M	GRASP	Time
BR1	81,76	83,37	86,77	92,63	87,81	88,1	89,39	88	89,07	**93,27**	1,27
BR2	81,7	83,57	88,12	92,7	89,4	89,56	90,26	88,5	90,43	**93,38**	2,32
BR3	82,98	83,59	88,87	92,31	90,48	90,77	91,08	89,5	90,86	**93,39**	4,62
BR4	82,6	84,16	88,68	91,62	90,63	91,03	90,9	89,3	90,42	**93,16**	6,52
BR5	82,76	83,89	88,78	90,86	90,73	91,23	91,05	89	89,57	**92,89**	8,58
BR6	81,5	82,92	88,53	90,04	90,72	91,28	90,7	89,2	89,71	**92,62**	12,23
BR7	80,51	82,14	88,36	88,63	90,65	91,04	90,44	88	88,05	**91,86**	19,25
BR8	79,65	80,1	87,52	87,11	89,73	90,26	—	—	86,13	**91,02**	38,20
BR9	80,19	78,03	86,46	85,76	89,06	89,5	—	—	85,08	**90,46**	63,10
BR10	79,74	76,53	85,53	84,73	88,4	88,73	—	—	84,21	**89,87**	97,08
BR11	79,23	75,08	84,82	83,55	87,53	87,87	—	—	83,98	**89,36**	136,50
BR12	79,16	74,37	84,25	82,79	86,94	87,18	—	—	83,64	**89,03**	183,21
BR13	78,23	73,56	83,67	82,29	86,25	86,7	—	—	83,54	**88,56**	239,80
BR14	77,4	73,37	82,99	81,33	85,55	85,81	—	—	83,25	**88,46**	307,62
BR15	75,15	73,38	82,47	80,85	85,23	85,48	—	—	83,21	**88,36**	394,66
Mean B1-B7	81,97	83,38	88,30	91,26	90,06	90,43	90,55	88,79	89,73	**92,94**	7,83
Mean	80,17	79,2	86,39	87,15	88,61	88,97	—	—	86,74	**91,05**	101,00

***The best values appear in bold**

Table 5: *GRASP computational results. Part I*

	Serial methods			Parallel methods				GRASP		
Class	**TSA**	**SA**	**HYB**	**PTSA**	**PSA**	**PHYB**	**PHYB.XL**	**5000 Iter**	**50000 Iter**	**200000 Iter**
BR1	93.23	93.04	93.26	93.52	93,24	93,41	93,70	93,27	93,66	**93,85**
BR2	93.27	93.38	93.56	93.77	93,61	93,82	**94,30**	93,38	93,90	94,22
BR3	92.86	93.42	93.71	93.58	93,78	94,02	**94,54**	93,39	94,00	94,25
BR4	92.40	92.98	93.30	93.05	93,40	93,68	**94,27**	93,16	93,80	94,09
BR5	91.61	92.43	92.78	92.34	92,86	93,18	93,83	92,89	93,49	**93,87**
BR6	90.86	91.76	92.20	91.72	92,27	92,64	93,34	92,62	93,22	**93,52**
BR7	89.65	90.67	91.20	90.55	91,22	91,68	92,50	91,86	92,64	**92,94**
Mean volume	92.00	92,53	92,86	92,70	92,91	93,20	93,78	92,94	93,53	**93,82**
Mean time	38	72	205	121	81	222	596	8	77	302

***The best values appear in bold**

Table 6: *GRASP computational results. Part II*

main reasons. First, the use of maximal-spaces allows us to have, at each step, the best possible spaces into which a box can fit. Second, the way in which the spaces are filled, though it seems counterintuitive, keeps maximal empty spaces larger and available for packing new boxes. Third, the GRASP scheme permits the combination of randomized and deterministic packing strategies. We have performed a preliminary computational test to determine the best strategies and parameter values from among those we have designed. The resulting algorithm produces good quality solutions in short computing times and can improve them if longer times are available. By adding a post processing phase in which the solutions are compacted, they become more stable and are sometimes even improved in terms of volume utilization. Nevertheless, more complex metaheuristics, based on the same ideas but adding more powerful improvement schemes, could improve the results further. The next phase of our research will address this aspect.

Acknowledgements

This study has been partially supported by the Spanish Ministry of Science and Technology, DPI2005-04796, and by Project PBI-05-022, Consejeria de Ciencia y Tecnologia, Junta de Comunidades de Castilla-La Mancha.

We also thank professor Ana Moura, from the Polytechnic Institute of Leiria, Portugal, for the fruitful discussions on this work.

References

[1] Bischoff, E.E. and Marriot, M.D. (1990) A Comparative Evaluation of Heuristics for Container Loading, *European Journal of Operational Research*, 44, 267-276.

[2] Bischoff, E.E. and Ratcliff, M.S.W. (1995) Issues in the Development of Approaches to Container Loading, *Omega*, 23, 377–390.

[3] Bischoff, E.E. Janetz, F. and Ratcliff, M.S.W. (1995) Loading Pallets with Nonidentical Items, *European Journal of Operational Research*, 84, 681–692.

[4] Bischoff, E.E. (2006) Three dimensional packing of items with limited load bearing strength, *European Journal of Operational Research*, 168, 952–966.

[5] Bortfeldt, A. and Gehring, H. (1998) A Tabu Search Algorithm for Weakly Heterogeneous Container Loading Problems, *OR Spectrum*, 20, 237–250.

[6] Bortfeldt, A. and Gehring, H. (2001) A Hybrid Genetic Algorithm for the Container Loading Problem, *European Journal of Operational Research*, 131, 143–161.

[7] Bortfeldt, A. Gehring, H. and Mack, D. (2003) A Parallel Tabu Search Algorithm for Solving the Container Loading Problem, *Parallel Computing* 29, 641–662.

[8] Davies, A.P. and Bischoff, E.E. (1998) Weight distribution considerations in container loading. Working Paper, European Business Management School, Statistics and OR Group, University of Wales, Swansea.

[9] Delorme, X. and Gandibleux, X. and Rodriguez, J. (2003) GRASP for set packing problems, *European Journal of Operational Research* 153, 564-580.

[10] Eley, M. (2002) Solving Container Loading Problems by Block Arrangement, *European Journal of Operational Research*, 141, 393–409.

[11] Feo, T. and Resende, M.G.C. (1989) A Probabilistic Heuristic for a Computationally Difficult Set Covering Problem, *Operations Research Letters* 8, 67-71.

[12] Gehring, H. and Bortfeldt, A. (1997) A Genetic Algorithm for Solving the Container Loading Problem, *International Transactions in Operational Research*, 4, 401–418.

[13] Gehring, H. and Bortfeldt, A. (2002) A Parallel Genetic Algorithm for Solving the Container Loading Problem, *International Transactions in Operational Research*, 9, 497–511.

[14] George, J. A. and Robinson, D. F. (1980) A Heuristic for Packing Boxes into a Container, *Computers and Operations Research*, 7, 147-156.

[15] Mack, D. Bortfeldt, A. and Gehring, H. (2004) A Parallel hybrid local search algorithm for the container loading problem, *International Transactions in Operational Research* 11, 511–533.

[16] Moura, A. and Oliveira, J.F. (2005) A GRASP approach to the Container-Loading Problem *IEEE Intelligent Systems*, 20, 50-57.

[17] Pisinger, D. (2002) Heuristics for the container loading problem, *European Journal of Operational Research*, 141, 382-392.

[18] Prais, M. and Ribeiro, C.C. (2000) Reactive GRASP: An application to a matrix decomposition problem in TDMA traffic assignment, *INFORMS Journal on Computing* 12, 164-176.

[19] Resende, M.G.C and Ribeiro, C.C. (2003) Greedy Randomized Adaptive Search Procedures, in *Handbook of Metaheuristics*, F.Glover and G.Kochenberger, Eds., Kluwer Academic Publishers, pp. 219-249.

[20] Ratcliff, M. S. W. and Bischoff, E. E. (1998) Allowing for weight considerations in container loading, *OR Spectrum*, 20, 65-71.

[21] Wäscher, G. and Haussner, H. and Schumann, H. An improved typology of cutting and packing problems, *European Journal of Operational Research*, in Press.