Neighborhood structures for the container loading problem: a VNS implementation

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

This paper presents a Variable Neighborhood Search (VNS) algorithm for the container loading problem. The algorithm combines a constructive procedure based on the concept of maximal-space, with five new movements defined directly on the physical layout of the packed boxes, which involve insertion and deletion strategies.

The new algorithm is tested on the complete set of Bischoff and Ratcliff problems, ranging from weakly to strongly heterogeneous instances, and outperforms all the reported algorithms which have used those test instances.

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

1 Introduction

In today's global economy a continuously increasing quantity of goods are shipped over long distances in containers. The efficient loading of these containers, that is, the minimization of the empty spaces inside them, is not only an economic requirement but also an ecological issue due to the adverse consequences of increased traffic on environmental resources. In this paper we address the container loading problem and propose efficient algorithms for solving it.

The Single Container Loading Problem (CLP) is a three-dimensional packing problem in which a large parallelepiped, the container, has to be filled with smaller parallelepipeds, the boxes, which are of different sizes and

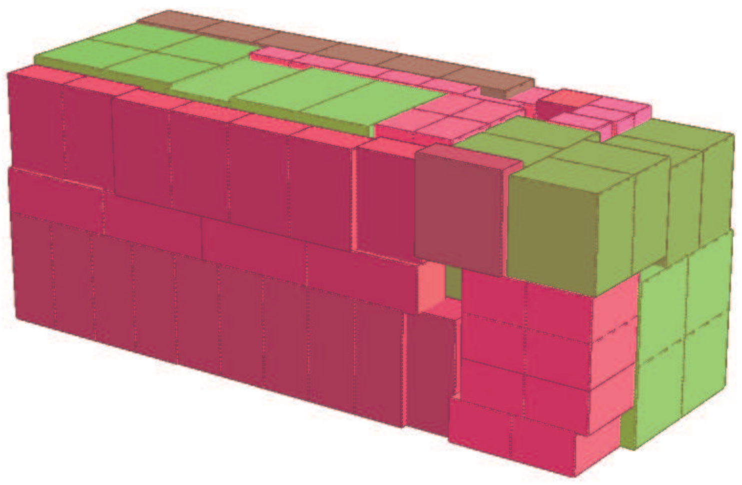


Figure 1: *Instance 9 of Class 7 from Table 5*

limited quantities, so that empty space is minimized (Figure 1). Within the improved typology for cutting and packing problems proposed by Wäscher et al. [19], 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.

Many procedures have been proposed for solving the CLP, from wall-building algorithms [11, 17] to complex metaheuristic schemes, such as Tabu Search [4], GRASP [15, 16], Simulated Annealing, genetic [9] and hybrid algorithms [5]. Sometimes these procedures include parallelization [10, 6, 13], resulting in highly complex and time-consuming algorithms designed to obtain high percentages of container utilization. All metaheuristic approaches involve some kind of neighborhood structure within which to move from one feasible solution to another. The moves are usually applied on the lists representing the solutions, changing the order in which boxes are packed, rather than on the physical layout of the boxes.

In this paper we follow a different approach. We define several movements directly on the boxes and consider the geometric consequences of moving, inserting or deleting boxes from a given packing. These moves are studied separately and then combined in a Variable Neighborhood Search (VNS) procedure. As in most of the previously reported papers, our algorihm uses the set of 1500 problems distributed over 15 classes with different characteristics generated by Bischoff and Ratcliff [1], which can be seen as the benchmark problems for computational experiments on CLP. The new algorithm outperforms the previously published approaches, taking only a fraction of the computational time, mainly in the hardest classes of problems.

When speaking about real-world container loading problems, space usage is the most important objective but other issues such as cargo stability, multi-drop loads or weight distribution ([1], [18], [7]) are also taken into account. Among these additional considerations, cargo stability is the most important one. Sometimes it is explicitly taken into account in the design of the algorithms. At other times the algorithms' results are also evaluated against standard measures of cargo stability. Our VNS algorithm does not take stability considerations into account explicitly, but we have included a postprocessing phase which compacts the solutions and makes them very stable according to the usual stability measures. Moreover, as container volume usage is very high, stability almost becomes a consequence of this high cargo compactness. If necessary, the little empty space left can be filled with foam rubber, as some freight companies do, to ensure proper support of the boxes.

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 section 3 several moves are defined, the corresponding neighborhoods are built and a VNS algorithm implementation is presented. In section 4 thorough computational experiments are described and results presented. Finally, in section 5 conclusions are drawn.

2 A constructive algorithm

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 $\mathcal{S}$ of empty maximal spaces, initially containing only the container C. At each step, a maximal space is chosen from $\mathcal{S}$ and from the boxes in $\mathcal{B}$ fitting into it one type of box and then one configuration of boxes of this type are chosen to be packed. That usually produces new maximal spaces going into $\mathcal{S}$ and the process goes on until $\mathcal{S} = \emptyset$ or none of the remaining boxes fit into any of the remaining maximal spaces. We give a simplified description of the algorithm below. More details can be found in [16].

- Step 0: *Initialization* $\mathcal{S}$={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.

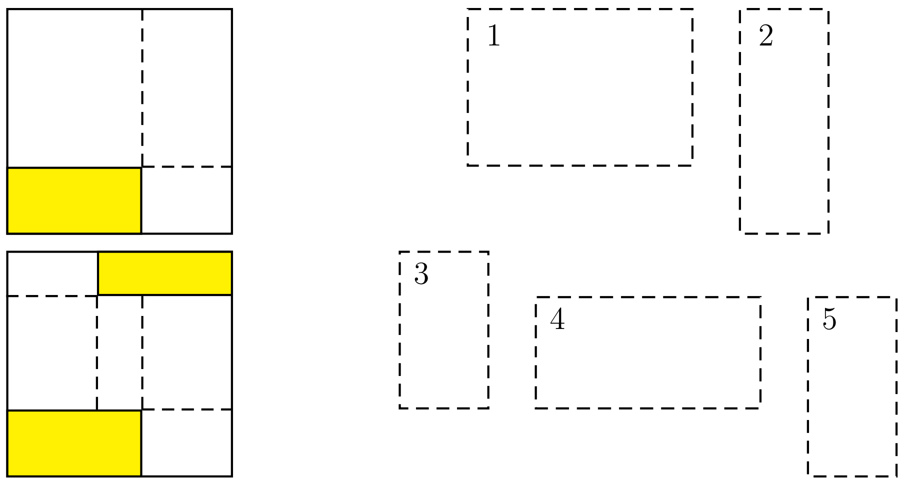


Figure 2: *Maximal spaces in two dimensions*

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

  We have a list $\mathcal{S}$ of empty maximal spaces. These spaces are called maximal because at each step they are the largest empty parallelepiped available 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 pack a piece into its bottom left corner, two maximal spaces are generated. These spaces do not have to be disjoint. In the same Figure 2 we see one more step of the filling process with the maximal spaces generated.

  At each step we take the maximal space with the minimum distance to a corner of the container and use the volume of the space as a tie-breaker. The corner of the maximal space with the lowest distance to a corner of the container will be the corner into 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 layer, that is, packing several copies of the box arranged in rows and columns, such that the number of boxes in the layer does not exceed q_i. In Figure 3 we can see different possibilities for building a layer with boxes of dimensions $(9, 5, 3)$ on the base of the container with dimensions $(20, 20, 20)$. Similar configurations could be built on the other sides of the container. If some rotations of the pieces are not allowed, the corresponding configurations will not be considered.

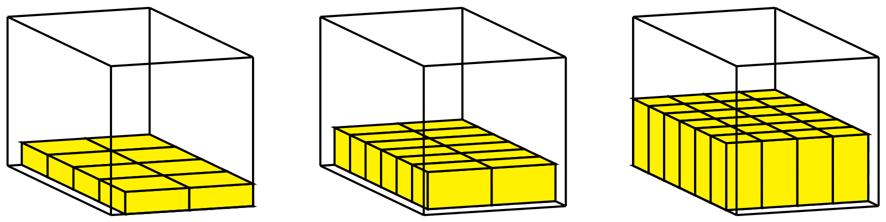


Figure 3: *Building layers with a piece in different positions*

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

i *Best-Volume:*

The layer producing the largest increase in the volume occupied by boxes.

ii *Best-Fit:*

The layer which best fits into the maximal space. We compute the distance from each side of the layer to each side of the maximal space and put these distances in a vector in non-decreasing order. The layer is chosen using the lexicographical order.

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

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

3 Variable neighborhood search algorithms

Variable neighborhood search (VNS) is a metaheuristic procedure developed by Mladenovic and Hansen [14] for solving hard combinatorial problems. For an updated introduction, refer to Hansen and Mladenovic [12].

VNS explores the solution space through a systematic change of neighborhoods. The procedure is based on the fact that a local minimum for a first type of move is not necessarily a minimum for another type of move. The VNS tries to avoid being trapped in local minima by considering more than one neighborhood. It has been successfully applied to diverse combinatorial optimization problems.

We have used five neighborhood structures, N_1 to N_5, considering different insertion and elimination procedures. These structures will be defined in the following subsections.

3.1 Definition of movements

The solution space in which we move is composed of feasible solutions only. In this space we will define several moves for going from one solution to another. An initial solution is obtained by applying the constructive algorithm described in the previous section.

We distinguish five types of movements: layer reduction, column insertion, box insertion, and emptying a region of the container and filling it again with two alternative procedures. In layer reduction, a layer of the solution is removed from the current solution. In column insertion, a new column is added to the solution in one of the empty spaces. In box insertion, a piece is included in the solution in one of the empty spaces. Finally the fourth and fifth moves consist of choosing two maximal spaces, emptying the smallest parallelepiped containing both of them and filling it again by using the constructive procedure with the two different objective functions described above, *Best-Volume* and *Best-Fit*.

We first present a scheme of the procedures and then some examples of their application in Figures 4, 5 and 6:

1. *Layer reduction*

    1. Initialization:
       $\mathcal{Y}$= the list of layers in the current solution
    2. Choosing the layer to reduce
       Take Y, one of the layers of $\mathcal{Y}$ with k columns and l rows
       Select the number r of columns (rows) to eliminate,

$$1 \leq r \leq k \quad (1 \leq r \leq l),$$

(if $r = k$ or $r = l$, the layer may disappear completely).

3. Move the remaining layers to the nearest corner of the container, measured by the lexicographical distance.

   If none if them is moved, go to step 1.
4. Update the list $\mathcal{S}$ of maximal spaces.
5. Fill the empty maximal spaces by applying the constructive algorithm with the *Best-volume* objective function.

In Step 3, if none of the layers is moved, no new spaces are generated apart from the space previously occupied by Y and filling the same space again will not produce any improvement.

2. *Column insertion*

   1. Initialization:
      $\mathcal{B}$= the list of boxes outside the container in the current solution
      $\mathcal{S}$= the list of empty maximal-spaces
   2. Choosing the space into which to insert new boxes: Take $S \in \mathcal{S}$
   3. Choosing the box to be inserted: Take $B \in \mathcal{B}$.
   4. Put box B into the corner of S nearest to a corner of the container.
   5. Choose a possible direction for building a column of boxes.

      In at least one dimension the box is larger than the space, but it is possible that in the other dimensions more than one box can fit into the space. In this case, we do not consider packing an individual box but a column with as many boxes as possible, in such a way that the column does not exceed the dimensions of the space in more than one direction.

      Let us suppose that S has dimensions (l_S, w_S, h_S) and B dimensions (l_B, w_B, h_B). If $l_S \leq l_B$, $w_S \leq w_B$ and $h_S \leq h_B$, then only one box B is inserted into S. But if, for instance, $l_S < l_B$ but $w_S > k * w_B$ and $h_S > m * h_B$, we can insert a column of k boxes along the width or a column of m boxes along the height of S.
   6. Remove the overlapping boxes of the container.
   7. Update the list $\mathcal{S}$ of maximal spaces.
   8. Fill the empty maximal spaces by applying the constructive algorithm with the *Best-volume* objective function.

3. *Box insertion*

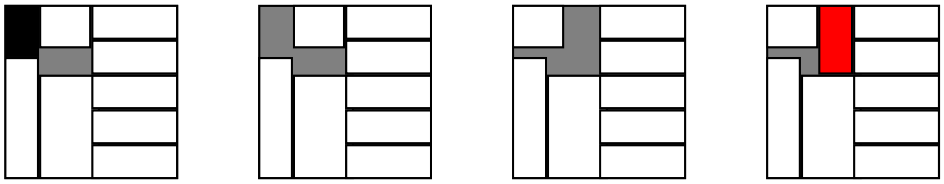
   1. Initialization:
      $\mathcal{B}$= the list of boxes outside the container in the current solution
      $\mathcal{S}$= the list of empty spaces
   2. Choosing a box to insert: Take $B \in \mathcal{B}$.
   3. Choosing the space into which insert this piece: Take $S \in \mathcal{S}$
   4. Choosing the position of B in S: Consider packing box B in every possible corner of S. Put the piece into the corner and remove the overlapping boxes.
   5. Update the list $\mathcal{S}$ of maximal spaces.
   6. Fill the empty maximal spaces by applying the constructive algorithm with the *Best-volume* objective function.

4.-5. *Emptying a region*

   1. Initialization:
      $\mathcal{Y}$= the list of layers in the current solution
      $\mathcal{S}$= the list of empty spaces
   2. Take a first space $S_1 \in \mathcal{S}$.
   3. From among the spaces which are smaller than S_1, take a second space $S_2 \in \mathcal{S}$.
   4. Create the smallest parallelepiped P containing S_1 and S_2. If P has the same volume as any emptied region explored before, go to 2, because it is probably the same region and will not be studied. Otherwise, remove all the boxes overlapping with P.
   5. Update the list $\mathcal{S}$ of maximal spaces.
   6. Fill the empty spaces by applying the constructive algorithm.

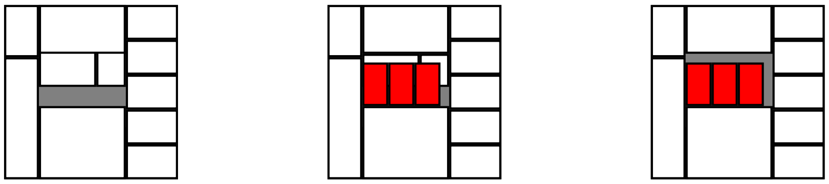
   According to the objective function used in Step 6, *Best-Volume* or *Best-Fit*, we have two different moves.

In Figures 4, 5 and 6 we can see some examples of these moves for a two-dimensional problem. In Figure 4 we have an example of the reduction of a layer in the solution. In fact, as the layer is composed of only one box, reduction amounts to elimination. Figure 5 shows an example of column insertion. The box width exceeds the space width, but along the length of S a column of three boxes can be inserted. Finally Figure 6 shows an example of emptying a region. First, two empty spaces are selected, then all the pieces that overlap with the smallest rectangle that contains them are removed and finally the resulting empty space is filled by applying the constructive algorithm.



(a) Selecting (b) Removing (c) Moving to the corner (d) Filling

Figure 4: *Layer reduction*



(a) Selecting the empty space (b) Inserting a column (c) Eliminating overlap

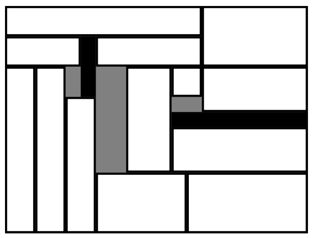
Figure 5: *Column insertion*

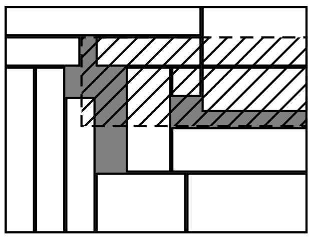
The four movements are complementary. There are two insertion moves and three based on eliminating boxes and filling the resulting empty spaces again. The difference between the insertion of a box and insertion of a column is double: firstly, in the second move we can insert more than one box, and secondly, when inserting one box we consider inserting it into each corner of each maximal space.

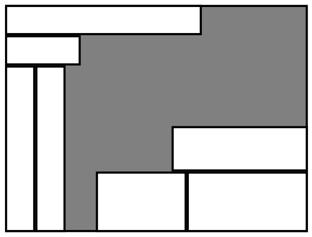
3.2 Neighborhood structures

Using the movements described above, we can build five different neighborhoods.

1. *N_1, based on layer reduction*
   The neighbors of a solution are the solutions obtained by applying the layer reduction move, considering at Step 2 each layer $Y \in \mathcal{Y}$ (with k columns and l rows) and for the chosen layer each possible reduction of r columns ($1 \leq r \leq k$) and each possible reduction of r rows ($1 \leq r \leq l$).


(a) Selecting the spaces


(b) Defining the region


(c) Removing overlapping boxes

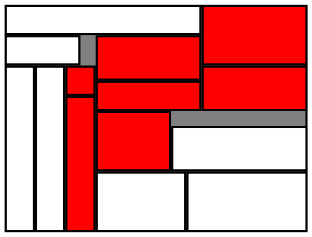

(d) Filling

Figure 6: *Emptying a region*

2. *N_2, based on column insertion*
   The neighbors of a solution are the solutions obtained by applying the column insertion move, taking at Step 2 a maximal-space S in non-decreasing order of volume, considering at Step 3 each box B outside the solution and studying at Step 4 each possible direction in which a column can be built.

3. *N_3, based on box insertion*
   The neighbors of a solution are the solutions obtained by applying the box insertion move, taking at Step 2 a box B outside the solution in non-decreasing order of volume, considering at Step 3 each maximal-space S in non-decreasing order of volume and studying at Step 4 each possible corner of S to pack B into.

4-5. *N_4 and N_5, based on emptying a region*
   The neighbors of a solution are the solutions obtained by applying the emptying move, taking at Step 2 a maximal-space S_1 in non-decreasing order of volume and considering at Step 3 each maximal-space S_2, smaller than S_1, in non-decreasing order of volume.

The sets of possible neighbors are very large and therefore we do not explore them fully, but at each iteration we explore only 1000 moves for the first three neighborhoods and 100 for the last ones. For the moves based on emptying regions of the container, we take special care only to study moves producing different regions in order to avoid repetitions.

3.3 Variable neighborhood descent

A first implementation to combine several neighborhoods in a deterministic way is given by the variable neighborhood descent (VND). Its steps are the following:

```
Initialization: Select the set of neighborhood structures $N_p$, for $p = 1, ..., 5$,
and start from the initial solution $x$ obtained by the constructive algorithm.
Repeat the following sequence until no improvement is obtained:
    (1) Set $p \leftarrow 1$
    (2) Repeat the following steps until $p = p_{max}$
        (a) Exploration of neighborhood.
            Find the best neighbor $x'$ of $x (x' \in N_p(x))$;
        (b) Move or not.
            If the solution $x'$ obtained is better than $x$, set $x \leftarrow x'$ and $p \leftarrow 1$.
            Otherwise, set $p \leftarrow p + 1$;
```

We have also tested a variant in which instead of finding the best solution in the neighborhood, a move is made as soon as a solution x' that improves x is found. This variant will be denoted as VND_{first}.

3.4 VNS

The basic VNS includes not only a systematic search of several neighborhoods, as in VND, but also some stochastic moves designed to escape from local optima, a strategy known as *shaking.* Its steps are the following:

Initialization: Select the set of neighborhood structures N_p, for $p = 1, ..., p_{max}$, that will be used in the shaking phase, and the set of neighborhoods N_l, for $l = 1, .., l_{max}$, which will be used in the local search. Start from the initial solution x obtained by the constructive algorithm.

```
Repeat the following sequence until the stopping condition is met:
(1) Set $p \leftarrow 1$
(2) Repeat the following steps until $p = p_{max}$
```

```
(a) Shaking. Generate a neighbor $x'$ of $x$ at random, using the $p^{th}$ neighborhood.
(b) Local search by VND.
    (b1) Set $l \leftarrow 1$;
    (b2) Repeat the following steps until $l = l_{max}$
      - Exploration of neighborhood.
        Find the best neighbor $x''$ of $x'$ in $N_l(x)$;
      - Move or not.
        If the solution $x''$ is better than $x'$, set $x' \leftarrow x''$ and $l \leftarrow 1$.
        Otherwise, set $l \leftarrow l + 1$;
(c) Move or not. If this local optimum is better than the incumbent, move there ($x \leftarrow x''$) and continue the search with $N_1$; otherwise, set $p \leftarrow p + 1$.
```

Initially we use the same set of neighborhoods, N_1 to N_5, for the shaking step and for the VND. A preliminary computational experiment shows that the moves based on inserting a box or a column hardly transform the solution, and the opposite occurs for the other moves involving elimination of boxes. We then have a balance between intensification and diversification shaking. Nevertheless, we have defined a stronger shaking movement in which we eliminate a percentage of the boxes in the current solution and fill the container again using the constructive procedure with the *Best-volume* objective function. This percentage varies between 10% and 30% of the boxes in the solution. This move defines a new neighborhood, N_6, which is only used in the Shaking phase.

4 Computational results

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 algorithms we have compared them with the most recent and efficient algorithms proposed for the container loading problem on the set of 1500 problems suggested by Bischoff et al. [2] and Davies and Bischoff [7]. Those instances are subdivided into 15 test cases each with 100 instances which are referred to as BR1 to BR15. The number of different box types in each class are 3, 5, 8, 10, 12, 15, 20, 30, 40, 50, 60, 70, 80, 90 and 100. According to the decreasing average number of boxes per type, the problem structure changes gradually from weakly heterogeneous to strongly heterogeneous. In test class BR1 there are on average 50.15 boxes for each box type, but in test class BR15 the average number is only 1.33. The average of available cargo is over

99.46% of the capacity of the container and for each individual instance it never exceeds the volume of the container. As the dimensions of the boxes were generated independently of the dimensions of the container, there is no guarantee that all the boxes will fit into the container. What is more likely is that the given percentage should be seen as a (quite loose) upper bound on the volume of the container that optimal packing could attain.

All computational results tables present average values for the 100 instances of each class. Each instance was only solved once. As the procedures have a random component, if we had solved them more than once, we could have obtained different solutions. However, if we compute the average for each instance over several runs and then the average of these averages over the 100 instances of each class, it is well-known that we would not obtain significantly different results. Obviously, if we compute the best value over several runs for each instance and then the averages for each class we would obtain better results, but with running times increased as many times as the runs we executed. Our option was to keep the running times low and therefore we decided to run each instance just once.

In order to choose the best strategies for defining the *VND* and *VNS* algorithms, we have done a limited computational study using some of the problem instances described above. The first series of experiments compare separate local searches in the five defined neighborhood structures N_1, N_2, N_3, N_4 and N_5, using the first 10 instances of each class. The results appear in Table 1. As can be expected, the table shows that the searches in N_1, N_2, N_3 are faster, but produce solutions of worse quality. The opposite holds for the neighborhoods N_4 and N_5, which give better results but require much longer running times.

We have used the five neighborhoods N_i, $i = 1, \dots 5$, in the implementation of the *VND* algorithm. The order in which the five neighborhoods are considered can influence the quality of the solution produced by the *VND* algorithm. Therefore, we have adopted a strategy consisting of alternating neighborhoods based on elimination and neighborhoods based on insertion. First, neighborhood N_4, based on emptying a region and filling it again with *Best-Volume* objective; then N_2, inserting a column; N_5, emptying a region and filling it again with *Best-Fit* objective; N_3, inserting a box; and finally N_1, removing a layer of the solution. In Table 2 we compare the results obtained using this order (*VNS_42531*) with the alternative of first using the simplest and fastest moves and calling the more complex moves only when they fail to produce improved solutions, that is, using the neighborhoods in order N_3, N_2, N_1, N_4 and N_5 (*VNS_32145*). The results show that taking the neighborhoods in the proposed order produces much better results than taking them in order of complexity, though the computing times are longer

								Emptying a region			
		N_3		N_2		N_1		N_4		N_5	
	Construc.	**Box Insertion**		**Column Insertion**		**Layer reduction**		**Best-Volume**		**Best-Fit**	
Problem	**Vol.(%)**	**Vol.(%)**	**Time**	**Vol.(%)**	**Time**	**Vol.(%)**	**Time**	**Vol.(%)**	**Time**	**Vol.(%)**	**Time**
BR_1	84,34	87,60	0,01	88,92	0,01	87,75	0,01	**92,08**	0,04	**92,08**	0,04
BR_2	85,61	87,23	0,02	89,36	0,02	88,41	0,01	91,84	0,05	**92,10**	0,07
BR_3	85,81	87,82	0,02	88,55	0,02	87,94	0,02	91,73	0,10	**92,45**	0,17
BR_4	87,07	87,29	0,02	88,21	0,02	88,72	0,03	**92,99**	0,23	92,66	0,17
BR_5	86,46	87,40	0,02	88,68	0,03	88,58	0,05	**92,00**	0,23	91,58	0,30
BR_6	88,21	88,40	0,04	89,43	0,04	88,79	0,06	91,22	0,31	**91,79**	0,37
BR_7	85,96	84,65	0,05	85,81	0,04	85,77	0,12	**90,47**	0,66	90,12	0,50
BR_8	85,96	86,71	0,10	87,27	0,12	87,17	0,33	89,07	1,01	**89,08**	1,12
BR_9	86,23	86,65	0,12	86,80	0,11	87,38	0,49	**89,29**	1,77	89,23	1,90
BR_10	85,72	86,46	0,17	86,20	0,16	86,93	0,71	**88,63**	2,38	88,62	2,06
BR_11	85,85	86,76	0,20	86,99	0,31	87,62	1,47	**88,81**	3,59	88,58	3,80
BR_12	85,18	87,04	0,34	86,86	0,30	87,34	1,51	**88,66**	5,14	88,40	5,18
BR_13	85,40	85,93	0,62	85,71	0,50	86,19	3,36	**87,83**	9,28	87,13	6,53
BR_14	84,87	85,72	0,72	85,87	0,74	86,18	4,42	**87,67**	9,70	87,11	9,09
BR_15	85,41	85,19	0,87	85,31	0,89	85,70	6,39	**87,27**	16,84	86,70	12,90
Mean	85,87	86,72	0,22	87,33	0,22	87,36	1,27	**89,97**	3,42	89,84	2,95

***The best values appear in bold**

Table 1: *Comparing the neighborhoods*

	VND_42531		VND_32145	
Problem	**Vol.(%)**	**Time**	**Vol.(%)**	**Time**
BR_1	**92,82**	0,08	92,54	0,28
BR_2	93,26	0,19	**93,39**	0,83
BR_3	**93,10**	0,34	91,94	1,85
BR_4	**93,73**	0,67	92,60	1,35
BR_5	**92,73**	0,98	91,60	1,73
BR_6	**92,67**	2,07	91,09	2,23
BR_7	**91,38**	2,21	90,54	3,25
BR_8	**90,56**	5,62	90,55	6,23
BR_9	**90,73**	8,18	89,76	7,03
BR_10	**89,94**	10,95	89,71	8,74
BR_11	**90,22**	16,88	88,90	12,00
BR_12	**89,88**	19,28	89,55	19,37
BR_13	**88,75**	27,25	88,60	20,82
BR_14	**88,73**	34,78	88,38	33,30
BR_15	**88,70**	59,01	88,24	29,11
Mean	**91,15**	12,57	90,49	9,87

***The best values appear in bold**

Table 2: *Comparing the orders in which the neighborhoods are used*

for strongly heterogeneous classes.

Table 3 shows the results obtained by three versions of the algorithm: the standard *VND*, VND_{first} and VND_{seq}. In VND_{seq} the neigborhoods are used in the same order as VND, but in the Local Search, when a solution is improved using the p^{th} neigborhood, instead of setting $p = 1$ and going back to the first neighborhood, the algorithm proceeds sequentially to the $(p+1)^{th}$. Both VND and VND_{seq} present similar results; however, VND_{seq} requires a slightly lower computational effort.

Table 4 compares several implementations of the VNS algorithm. Columns 2 and 3 correspond to a simple implementation, VNS_{red}, in which only neighborhood N_4 is used in shaking and local search. Columns 4 and 5 correspond to VNS_{seq}, which is similar to VND_{seq}, which uses the five neighborhoods, but in the Local Search when a solution is improved using the p^{th} neighborhood, instead of setting $p = 1$ and going back to the first neighborhood, the algorithm proceeds to the $(p + 1)^{th}$. Columns 6 and 7 correspond to VNS, a standard implementation of the algorithm as described in section 3.4, running for 15 iterations. The number of iterations in these three procedures has been set to obtain similar running times. The results are quite similar,

	VND_First		*VND*		*VND_Seq*	
Problem	**Vol.(%)**	**Time**	**Vol.(%)**	**Time**	**Vol.(%)**	**Time**
BR_1	92,63	0,11	92,82	0,08	**92,93**	0,13
BR_2	93,27	0,20	93,26	0,19	**93,58**	0,26
BR_3	92,96	0,55	**93,10**	0,34	93,06	0,52
BR_4	92,77	0,60	**93,73**	0,67	93,29	0,75
BR_5	92,36	1,23	**92,73**	0,98	92,35	1,14
BR_6	91,97	1,15	**92,67**	2,07	92,08	1,20
BR_7	90,55	2,44	91,38	2,21	**91,89**	2,22
BR_8	90,69	4,63	90,56	5,62	**91,61**	5,42
BR_9	90,49	6,34	**90,73**	8,18	90,08	7,15
BR_10	89,61	11,05	89,94	10,95	**90,41**	10,34
BR_11	89,55	13,08	**90,22**	16,88	89,85	15,53
BR_12	89,36	14,40	**89,88**	19,28	89,75	21,62
BR_13	88,31	21,93	88,75	27,25	**89,08**	29,82
BR_14	88,70	39,76	**88,73**	34,78	88,37	32,86
BR_15	88,05	39,93	**88,70**	59,01	88,58	35,49
Mean	90,75	10,49	**91,15**	12,57	91,13	10,96

***The best values appear in bold**

Table 3: *Comparing VND strategies*

	VNS_{red} **60 Iter**		VNS_{Seq} **18 Iter**		VNS **15 Iter**	
Problem	**Vol.(%)**	**Time**	**Vol.(%)**	**Time**	**Vol.(%)**	**Time**
BR_1	94,07	1,84	**94,85**	2,98	94,56	1,97
BR_2	**95,19**	3,60	95,10	5,60	94,93	4,75
BR_3	94,54	9,46	**94,97**	11,06	94,55	7,40
BR_4	**94,85**	10,21	94,52	15,12	94,41	13,44
BR_5	94,11	17,63	**94,19**	22,62	**94,19**	18,59
BR_6	93,35	29,17	93,61	31,71	**93,63**	26,01
BR_7	92,70	41,41	**93,38**	58,00	92,99	43,73
BR_8	91,94	88,45	**92,36**	122,05	92,23	104,92
BR_9	91,50	118,97	91,69	141,84	**92,05**	123,88
BR_10	91,08	169,90	**92,04**	218,05	91,81	182,46
BR_11	90,96	248,00	91,16	309,12	**91,22**	272,24
BR_12	90,83	295,56	90,94	375,65	**91,23**	404,09
BR_13	90,12	436,72	**90,88**	502,25	90,50	527,07
BR_14	89,92	515,50	**90,46**	640,32	90,05	620,18
BR_15	89,74	686,03	**90,35**	788,24	90,22	764,01
Mean	92,33	178,16	**92,70**	216,31	92,57	207,65

***The best values appear in bold**

Table 4: *Comparing VNS alternatives*

but we can see that the VNS versions which use several neighborhoods are better than that using just one. If we compare the two versions of the VNS using the five neighborhoods, we see that VND_{seq} produces slightly better results in similar computing times. The reason may lie in the fact that we have changed the order in which the neigborhoods are searched, first putting the more efficient and time-consuming one. It is faster to follow the complete cycle of neighborhoods instead of going back to the first one whenever an improved solution is found, and the time saved is well-used making more iterations.

For the complete computational results we use the complete set of 1500 instances generated by Bischoff and Ratcliff [1], instead of just solving the first ten instances of each class as has been done in previous tables. We use the algorithm VNS_{seq} but we have changed its stopping criterion. As this algorithm is much faster than other previous algorithms against which it is going to be compared, we let it run for a minimum of 30 iterations and then go on until no improvement is found in the last 5 iterations or a maximum of 60 iterations is reached. Using this new stopping criterion, the average times are increased around 10% of the times reported in Table 4, but the average results are slightly improved. As the algorithm does not take into account cargo stability considerations, a postprocessing phase has been added in which the solution is compacted. 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 of the container to the front 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 some boxes are moved. When the compacting phase is finished, the empty spaces are checked for the possibility of packing some of the unpacked boxes. In some cases, the results obtained by the VNS_{seq} are improved after this compacting procedure.

Table 5 compares the VNS_{seq} algorithm with the most recent and efficient metaheuristic procedures reported in the literature. The Table includes results from the parallel algorithms proposed by Mack et al. [13]: a parallel simulated annealing algorithm, *PSA_MB*; a parallel hybrid algorithm, *PHYB_MB*; and a massive parallel hybrid algorithm *PHYB_XL_MB*; and our *GRASP* algorithm [16] with iteration limits of 5000, 50000 and 200000. For all these algorithms, except GRASP with 5000 iterations, the authors only report the results on classes BR1 to BR7. Mack et al. [13] use computers Pentium-PC at 2Ghz, a LAN of 4 computers for the parallel implementation of *PSA* and *PHYB* and a LAN of 64 computers for the *PHYB.XL* implementation. Algorithms *GRASP* and *VNS* run on a Pentium Mobile at 1.5 GHz.

VNS takes much shorter running times (except when compared with GRASP_5000) and obtains the best results on average and in each individual class, except for BR_7 in which GRASP_200000 obtains a better result. The lower half of the table compares VNS_{seq} with *GRASP_5000* because the other algorithms have not been run on these classes. In this case, VNS_{seq} clearly outperforms GRASP_5000 with moderately longer running times. In general, the improvements obtained by the VNS_{seq} are very important, considering the high efficiency of the other algorithms and the distance to the upper bound.

5 Conclusions

We have developed a new heuristic algorithm based on variable neighborhood search for the container loading problem. We have proposed several new neighborhoods based on the elimination of layers, insertion of columns or boxes and a stronger move based on emptying a region of the container.

	Parallel methods			*GRASP*			
Class	PSA	$PHYB$	$PHYB.XL$	*5000 iter*	*50000 iter*	*200000 iter*	VNS_{seq}
BR_1	93,24	93,41	93,70	93,27	93,66	93,85	**94,93**
BR_2	93,61	93,82	94,30	93,38	93,90	94,22	**95,19**
BR_3	93,78	94,02	94,54	93,39	94,00	94,25	**94,99**
BR_4	93,40	93,68	94,27	93,16	93,80	94,09	**94,71**
BR_5	92,86	93,18	93,83	92,89	93,49	93,87	**94,33**
BR_6	92,27	92,64	93,34	92,62	93,22	93,52	**94,00**
BR_7	91,22	91,68	92,50	91,86	92,64	92,94	**93,53**
Mean B1-B7	92,91	93,20	93,78	92,94	93,53	93,82	**94,34**
Average times B1-B7	81	222	596	8	77	302	28
BR_8				91,02			**92,78**
BR_9				90,46			**92,18**
BR_10				89,87			**91,92**
BR_11				89,36			**91,46**
BR_12				89,03			**91,20**
BR_13				88,56			**91,08**
BR_14				88,46			**90,65**
BR_15				88,36			**90,38**
Mean B8-B15				89,39			**91,46**
Overall mean				91,05			**92,89**
Overall mean times				101			296

***The best values appear in bold**

Table 5: *Comparison of algorithms*

The computational results show that these ideas work well for a large set of instances. Our experimentation shows that the VNS methodology competes favorably with the best known algorithms for the container loading problem.

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.

References

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

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

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

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

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

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

[7] 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.

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

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

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

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

[12] Hansen, P. and Mladenovic, N. (2005) Variable neighborhood search, in *Search methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, E.Burke and G.Kendall, Eds., Springer, pp. 211-238.

[13] 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.

[14] Mladenovic, N. and Hansen, P. (1997) Variable neighborhood search,*Computers and Operations Research*, 24, 1097-1100.

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

[16] Parreño, F., Alvarez-Valdes, R., Oliveira, J.F. and Tamarit, J.M. (2007) Maximal-space algorithm for the container loading problem *Technical Report TR03-2007*, Department of Statistics and Operations Research, University of Valencia.

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

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

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