An AND/OR-graph Approach to the Container Loading Problem

REINALDO MORABITO* and MARCOS ARENALES†
*Universidade Federal de Sao Carlos, Brazil and †Universidade de Sao Paulo, Brazil

The container loading problem consists of packing boxes of various sizes into available containers in such a way as to optimize an objective function. In this paper we deal with the special case where there is just one available container and the objective is to maximize the total volume (or the total utility value, supposing that each box has a utility value) of the loaded boxes. We firstly present three heuristic solution methods for the unconstrained problem. Two of them solve the original three-dimensional problem by layers and by stacks reducing it into several problems with lower dimensions. The third one consists of representing possible loading patterns as complete paths in an AND/OR-graph. Bounds and heuristics are proposed in order to reduce the solution space. A proper heuristic is also given to treat the constrained problem by using the AND/OR-graph approach. Moreover, computational results are presented by solving a number of examples.

Key words: cutting and packing problems, search graph, branch and bound method, heuristics

INTRODUCTION

The container loading problem is a three-dimensional problem from within the general category of cutting and packing. Parallelepipeds (i.e. boxes) of various sizes have to be packed into (or cut from) larger paralellepipeds (i.e. containers) in such a way as to optimize certain functions. In addition to the geometric constraints (no overlap is allowed), other constraints have to be held, for example, cargo loading stability and density.

Suppose initially that the containers are enough to arrange all the boxes. Consider the following combinatorial optimization problem:

\[
\text{load all the boxes into the containers in such a way}\nonumber \\
\text{the total volume of the used containers is minimized.} 
\]

A subset of containers is chosen to load all the boxes. Note that if the containers are identical then problem (1) can be stated so as to minimize the necessary number of containers.

Now suppose that the containers are not enough to arrange all the boxes. Consider this second combinatorial optimization problem:

\[
\text{load the maximum volume of boxes into the available containers.} 
\]

In this way a subset of boxes have to be chosen and placed into the available containers.

Since the 1950s various authors have proposed different approaches to the one-dimensional or two-dimensional cutting and packing problems. A few of them considered three dimensions. Surveys are given in Dyckhoff (1990), Dowsland and Dowsland (1992) and Dyckhoff and Finke (1992). Sweeney and Paternoster (1992) and Dyckhoff and Finke gathered references to more than 400 works.

The container loading problem is probably the most important three-dimensional problem. Bischoff and Marriot (1990) distinguished cases where a cargo has to be transported in several containers [problem (1)] and cases where the maximum volume of a cargo must be loaded in only one container [an instance of problem (2)]. The authors have also mentioned cases where the cargo is loaded by its weight and its distribution and cases where the objective is to get the maximum utilization of the container as a function of volume and value of the loaded cargo.

A suitable choice of container size in terms of the cargo density permits a better use of its volumetric capacity. Haessler and Talbot (1990) studied a three-dimensional loading problem
considering only cargo of low density. These authors have also argued that in several situations the volumetric capacity limits the quantity of boxes, before weight constraints were reached (see also George and Robinson, 1980; Gehring et al., 1990).

In this work we are concerned with loading just one container and we consider only the spatial constraints (geometric combination of the boxes into the container) and the stability of the cargo.

**PROBLEM FORMULATION**

Consider a set of boxes which are grouped into \( m \) types. All boxes type \( i, \ i = 1, \ldots, m, \) are characterized by their length, width and height \((l_i, w_i, h_i)\) and the number of boxes \( b_i. \) Also consider a set of containers which are grouped into \( n \) types. All containers type \( k, \ k = 1, \ldots, n, \) are characterized by their size \((L_k, W_k, H_k)\) and are available in \( B_k \) units. Also suppose that the boxes are loaded into the containers following a fixed orientation.

Gilmore and Gomory (1965) presented a method to solve problem (1) based on the simplex method (since it can be modeled as a linear program) and a subproblem being solved at each iteration to generate a loading pattern. This subproblem is given by:

\[
\begin{align*}
\text{maximize} & \quad \sum_{i=1}^{m} v_i a_i \\
\text{subject to} & \quad \text{the condition that the m-vector } (a_1, a_2, \ldots, a_m) \\
& \quad \text{corresponds to a three-dimensional packing pattern}
\end{align*}
\]  

where \( a_i \) is an integer variable representing the number of boxes type \( i \) in the pattern and \( v_i \) is the simplex multiplier (or a function of it) associated with a particular basis. This method works well when \( b_i \) is considerably greater than \( a_i. \)

Now consider just one container of size \((L, W, H)\). Note that if \( v_i \) in (3)] is the volume of box type \( i, \) then we have a particular case of problem (2) with just one available container.

Let \([z]\) denote the largest integer number less than or equal to \( z. \) If the quantity \( b_i \) is not large enough (i.e. \( b_i < [L/l_i][W/w_i][H/h_i] \)), then the variable \( a_i \) has to be superiorly bounded by \( b_i. \) In this case problem (3) is called a constrained problem; otherwise, it is called an unconstrained problem.

The purpose of this work is to present heuristic solution methods to the unconstrained and constrained problems and to show their computational performance. The additional constraints \( a_i \leq b_i, i = 1, \ldots, m, \) impose considerable difficulties to solving problem (3). This will be discussed later (AND/OR-graph Approach), where an AND/OR-graph approach is proposed to solve both the unconstrained and constrained problem. The methods in the next two sections are limited to dealing with the unconstrained problem.

**SOLVING THE PROBLEM BY LAYERS**

In order to get a three-dimensional packing pattern we simplify the possibilities in two phases (Fig. 1). Firstly, horizontal layers of size \((L, W)\) are arranged with boxes that have the same height (i.e. at most \( m \) two-dimensional packing problems are solved) and finally these layers are chosen to be piled up along the height \( H (i.e. a one-dimensional knapsack problem is solved).

This scheme of solving problem by layers generalizes the method of Gilmore and Gomory (1965) for two-dimensional problems with a 2-staged cutting pattern. Of course, each two-dimensional problem in the first phase (which is to build a layer) can be heuristically solved imposing the constraint of two stages and so only one-dimensional knapsack problems should be solved. In the next section we give another way to generalize the method of Gilmore and Gomory.

**SOLVING THE PROBLEM BY STACKS**

Similarly to the above method, we first define stacks of height \( H \) by piling up boxes (i.e. various one-dimensional knapsack problems are solved) and then we choose which stacks should build the
loading pattern by arranging them over the floor \((L, W)\) of the container (i.e. a two-dimensional packing problem is solved). This procedure was initially proposed by Gilmore and Gomory (1965) but we did not find computational experiences using it in the literature. Figure 2 illustrates an instance of this method using only stacks of size \((l_i, w_i, H)\), \(i = 1, \ldots, m\).

The methods described in this section and in the earlier section provide heuristic solutions to the three-dimensional packing problem, reducing it to solve a number of one-dimensional and two-dimensional problems (see Morabito, 1992 for additional details). In the next section, we generalize a third method initially proposed to the two-dimensional cutting problem (Morabito and Arenales, 1992) and by this method the three-dimensional problem is directly approached without reducing it into lower dimensions.

**AND/OR-GRAPH APPROACH**

In the following, we describe a procedure to generate all possible unconstrained and constrained three-dimensional guillotine cutting patterns. Note that this approach discards possible optimal non-guillotine patterns to the problem.

**Guillotine pattern**

Let us consider the container \((L, W, H)\). Guillotine cuts are performed on \((L, W, H)\) yielding intermediary boxes which are successively cut to obtain the ordered boxes \((l_i, w_i, h_i)\), \(i = 1, \ldots, m\). Figure 3a shows a sequence of cuts performed on the container \((L, W, H)\) (A in the figure), producing the cutting pattern illustrated in Fig. 3b.

Initially a guillotine cut is done on the length \(L\) of container A (cut 1 in Fig. 3b), yielding two
intermediary boxes B and C, called successors of A. Next, both of these boxes are independently cut. The box B is horizontally cut (cut 2 in Fig. 3b) producing the successors D and E. The box C is cut on the weight W providing F and G. This sequence of cuts generates a cutting pattern for the container \((L, W, H)\) which is illustrated in Fig. 3b (note that the intermediary boxes B and C are not indicated in this pattern but only the final boxes D, E, F and G).

Of course, a number of other cuts could be performed on each box, producing different cutting patterns. If all cutting possibilities on each box were investigated, including the no-cut option (called 0-cut), we could generate all guillotine cutting patterns. Observe that, for each successor box, a subproblem similar to the original is obtained.

The boxes in Fig. 3a can be represented as nodes in an oriented graph. The initial node represents the container \((L, W, H)\). If we know all the cutting patterns for every successor of a particular node, we have all cutting patterns related to this node. In this case we say the node is solved. A node representing a box \((l, w, h)\) such that \(l < \min\{l_i, i = 1, \ldots, m\}\), \(w < \min\{w_i, i = 1, \ldots, m\}\) or \(h < \min\{h_i, i = 1, \ldots, m\}\) accepts only a 0-cut (such a node corresponds to a waste on the cutting pattern or an empty space in the packing pattern). A node representing a box from a 0-cut is not cut any more and is called final.

A cut performed on a box is represented by an AND-arc in the oriented graph pointing to two successor nodes corresponding to the boxes obtained (for example, the arc linking A with B and C in Fig. 3a). These arcs define the relationship among the nodes. The set of all nodes and arcs is called an AND/OR-graph.

Consider the following sequence of arcs: from the initial node we choose one AND-arc (and only one) and from each successor node we choose one AND-arc (and only one) and so on, until all nodes found are final. This sequence is called a complete path in the AND/OR-graph. There is an onto function from the set of all complete paths in the above defined AND/OR-graph to the set of all guillotine cutting patterns, that is, for every guillotine cutting pattern there exists at least one complete path in the AND/OR-graph whose sequence of cuts (arcs) yields it (different complete paths can produce the same cutting pattern). If a final node represents an ordered box \((l, w, h)\) then its value is \(v_i\); otherwise it is zero. The value of a complete path is defined as the sum of the values of the final nodes in the complete path.

Therefore, the problem of determining the best guillotine cutting pattern consists of determining the most valuable complete path in the above described AND/OR-graph. This path is determined as soon as the initial node is solved.

A search strategy is a particular way of traversing the graph or a way of enumerating its nodes; it defines a method to solve the problem. Later (A Search Strategy), we present a search strategy to solve container loading problems. Morábito and Arenales (1992) used it in order to solve staged and constrained two-dimensional cutting problems. Other strategies are presented by Nilsson (1971) and Pearl (1984).

The complete enumeration of the nodes is, in general, computationally unfeasible. It is desirable in
the search to avoid equivalent cutting patterns, i.e. different patterns that yield the same quantity of boxes type \(i\) (see Fig. 4). In the next section we describe some rules to avoid some of them. The search can also be reduced by using bounds to get rid of nonpromising paths, implicitly enumerating the nodes (a branch and bound method). In a later section (Lower and Upper Bounds) we provide lower and upper bounds for each node. Although the generation of a number of nodes can be avoided, the search can still be computationally unfeasible. In the section of Heuristics we show how the bounds can also be used to define additional heuristics that enable us to find good and computationally feasible solutions.

![Fig. 4. Three equivalent cutting patterns.](image)

So far we have shown how unconstrained guillotine cutting patterns can be represented in an AND/OR-graph (since no restriction on the number of boxes was imposed). In the sequence we show how constrained guillotine cutting patterns can be generated by a graph search.

**Constrained pattern**

If there exists a limit \(b_i\) on the number of boxes type \(i\) available to be packed in the container and \(b_i < [L/l_i][W/w_i][H/h_i], i = 1, \ldots, m\), then the problem is called constrained. Let us consider a node \(N\) corresponding to a box to be cut. The decision to pack boxes type \(i\) in the intermediary box represented by \(N\) is not independent of the decision to pack other quantities of boxes type \(i\) in other nodes that belong to the same path that includes \(N\).

Let \(f_i(N)\) be the maximum quantity of boxes type \(i\) that can be packed in the box at \(N\) (if \(N\) is the initial node, then \(f_i(N) = b_i, i = 1, \ldots, m\)). Let \((N_1, N_2)\) be a pair of successors from \(N\), obtained by a guillotine cut. The following problem has to be solved:

\[
\begin{align*}
\text{maximize} \quad & \sum_{i=1}^{m} v_i a_i^1 + v_i a_i^2 \\
\text{subject to:} \quad & (a_1^1, a_2^1, \ldots, a_m^1) \text{ is a guillotine packing pattern for } N_1, \\
& (a_1^2, a_2^2, \ldots, a_m^2) \text{ is a guillotine packing pattern for } N_2, \\
& a_1^1 + a_1^2 \leq f_i(N), i = 1, \ldots, m. 
\end{align*}
\]

(4)

It is not a trivial task to solve problem (4). In the section on Heuristics we provide a heuristic solution for it. Note that if \(f_i(N), i = 1, \ldots, m\), were large enough (unconstrained problem), problem (4) could be decomposed into two independent problems for solving \(N_1\) and \(N_2\). In the next section we present some rules to avoid equivalent patterns.

**AVOIDING EQUIVALENT PATTERNS**

**Normal patterns**

Herz (1972) showed for two-dimensional problems that the guillotine cuts can, without loss of generality, be integer nonnegative linear combinations of the sizes of the ordered rectangles. This can be extended to three-dimensional problems. Let \(l_0 = \text{minimum}\{l_i, i = 1, \ldots, m\}\). We can reduce the cuts along the length \(L\) to the set \(\mathcal{X}\):
\[ X = \left\{ x \mid x = \sum_{i=1}^{m} \alpha_i l_i, 1 \leq x \leq L - l_0, 0 \leq \alpha_i \leq b_i \text{ and integer} \right\} \]

(similarly, sets \( \gamma \) and \( \zeta \) can be defined for the cuts along the width \( W \) and the height \( H \), respectively). The sets \( X, \gamma \) and \( \zeta \) are called discretization sets. Later on we extend the Christofides and Whitlock (1977) formulae to generate the discretization sets.

**Symmetry**

Herz also proved that a kind of duplication can be avoided by defining for each node \( N \) a set \( X(N) \) as the following. Let \( (x, y, z) \) be a box represented by \( N \)

\[ X(N) = \left\{ x_1 \mid x_1 = \sum_{i=1}^{m} \alpha_i l_i \text{ and } (if w_i > y \text{ or } h_i > z \Rightarrow \alpha_i = 0), 1 \leq x_1 \leq \lfloor x/2 \rfloor, 0 \leq \alpha_i \leq b_i \text{ and integer} \right\} \]  

(5)

[similarly, set \( \gamma(N) \) and \( \zeta(N) \)]. Note in (5) that the set \( X(N) \) does not depend on \( y \) or \( z \).

**Exclusion**

Consider again a node \( N \) representing a box \( (x, y, z) \). If an ordered box \( (l_i, w_i, h_i) \) is such that \( w_i > y \) or \( h_i > z \), then the length \( l_i \) can be excluded from consideration in order to produce \( X(N) \). So,

\[ X(N) = \left\{ x_1 \mid x_1 = \sum_{i=1}^{m} \alpha_i l_i \text{ and } (if w_i > y \text{ or } h_i > z \Rightarrow \alpha_i = 0), 1 \leq x_1 \leq \lfloor x/2 \rfloor, 0 \leq \alpha_i \leq b_i \text{ and integer} \right\} \]  

(6)

[similarly, with \( \gamma(N) \), if \( l_i > x \) or \( h_i > z \) and with \( \zeta(N) \), if \( l_i > x \) or \( w_i > y \)]. Observe in (6) that \( X(N) \) now depends on \( y \) and \( z \).

Christofides and Whitlock (1977) proposed recursive formulae to build the discretization sets for two-dimensional problems. Here we extend them to three-dimensional problems. The discretization sets are determined for the initial node that represents the container \((L, W, H)\) and, after that, the sets \( X(N), \gamma(N) \) and \( \zeta(N) \) are easily found for any node \( N \).

Suppose \( l_1 \leq l_2 \leq \ldots \leq l_m \) and let \( F_i(x_1) \) be the minimum of the maximum width of the boxes \( 1, 2, \ldots, i \) whose lengths can be combined to sum \( x_1 \). So,

\[ F_i(x_1) = \min \left\{ F_{i-1}(x_1); \max \left\{ w_i; \min \{ F_{i-1}(x_1 - \rho l_i), 1 \leq \rho \leq \min \{ \lfloor x_1/l_i \rfloor, b_i \text{ and integer} \} \} \right\} \right\} \]

\[ l_i \leq x_1 \leq \lfloor L/2 \rfloor, \]

\[ F_i(x_1) = F_{i-1}(x_1), x_1 < l_i, \]

where \( F_0(x_1) = \infty, x_1 = 1, \ldots, \lfloor L/2 \rfloor \), and \( F_i(0) = 0, i = 0, 1, \ldots, m \) (similarly, let \( G_i(x_1) \) be the minimum of the maximum height of the boxes \( l_1, 2, \ldots, i \) whose lengths can be combined to sum \( x_1 \)).

If \( F_i(x_1) < \infty \) and \( G_i(x_1) < \infty \) then \( x_1 = \sum_{j=1}^{i} \alpha_j l_j, 0 \leq \alpha_j \leq b_j \text{ and } \alpha_j \text{ integer} \). Since \( F_i(x_1) \) and \( G_i(x_1) \) are independent of the graph nodes, they can be generated before starting the search process. After that, the set \( X(N) \) can be easily found using \( F_m(x_1) \) and \( G_m(x_1) \). If \( F_m(x_1) \leq y \) and \( G_m(x_1) \leq z \), then \( x_1 \in X(N) \).

The discretization set \( X(N) \) in (6) is rewritten as

\[ X(N) = \{ x_1 \mid F_m(x_1) \leq y \text{ and } G_m(x_1) \leq z, 1 \leq x_1 \leq \lfloor x/2 \rfloor \} \]  

(7)

[similarly, set \( \gamma(N) \) and \( \zeta(N) \)].
Cut ordering

Suppose that the node $N$ representing the box $(x, y, z)$ is cut at $x_1, x_1 \in X(N)$ yielding $(x_1, y, z)$ and $(x - x_1, y, z)$. Then suppose $(x - x_1, y, z)$ is cut at $x_2, x_2 \in X(N)$, producing $(x_2, y, z)$ and $(x - x_1 - x_2, y, z)$. These three boxes could also be produced by cutting the box $(x, y, z)$ first at $x_2$ and then cutting $(x - x_2, y, z)$ at $x_1$.

Christofides and Whitlock (1977) remarked that this duplication can be avoided, without loss of optimality, by introducing an arbitrary order to the successive cuts on the length (similarly to successive cuts on the width and height).

The symmetry and cut ordering rules will be heuristics rules when combined together with a greedy heuristic presented later to solve problem (4).

STABILITY OF THE LOADING

It is also necessary to consider the stability of the boxes which are piled up. Figure 5 depicts an unstable loading due to the nonoccupied space under box 3. Note that the methods presented previously (Solving the Problem by Layers; Solving the Problem by Stacks) tend to yield stable loadings.

An additional condition can be imposed on the search graph in order to avoid unstable loading. For this, after a cut along the height, the newly generated boxes will only be cut along their heights. The idea is to produce layers (intermediary boxes) which will be filled in with identical boxes (see lower bounds in next section). These layers could be, if necessary, exchanged between each other to try to get stable loading. Note that this condition avoid the unstable loading in Fig. 5 since after the cut, indicated by $zz'$, the lower box should be cut at $xx'$ to produce boxes 1 and 2. Figure 6a illustrates an intermediary box after cuts along the height and an unstable loading. After exchanging layers, Fig. 6b shows the same pattern but stable. Of course, this procedure, as well as the earlier ones, is not free from producing unstable loading since it can occur if a rotation of a box can be avoided along the length but not along the width. Furthermore, even if stable loading is obtained, there is no guarantee of optimality.

LOWER AND UPPER BOUNDS

In this section we present some bounds of great utility in the search process. Consider a node $N$ representing the box $(x, y, z)$ and let

$$B(N) = \{ i | l_i \leq x, w_i \leq y, h_i \leq z, i = 1, \ldots, m \}$$

be the set of the boxes that can be packed in the box at node $N$.

Lower bounds – constrained problem

Very simple lower bounds can be defined for node $N$ from trivial cutting patterns. A cutting pattern that uses only a type of box, called homogeneous cutting, provides:
If \( E_j(N) \) is much smaller than \( \frac{y}{w_j} \), the homogeneous cutting will produce a large waste which can be used to pack other boxes.

If we arrange the box \( j \) into layers (we could arrange them into walls as well), we will fill \( \theta_j \) layers, where

\[
\theta_j = \min\left\{ \frac{z}{h_j}, \delta_j(N) \left( \frac{x}{l_j} \right) \right\} \tag{9}
\]

[given that \( \theta_j \leq \left\lfloor \frac{z}{h_j} \right\rfloor \) and \( \theta \left( \frac{x}{l_j} \frac{y}{w_j} \right) \leq \delta_j(N) \)] that can be a non-integer number. By convenience, we approximate it to the nearest integer number, that is, \( \theta_j \leftarrow \left\lfloor \theta_j + 0.5 \right\rfloor \). So, a box \((x, y, z - \theta_j h_j)\) is left which can also be filled with another homogeneous solution (not including the box \( j \)). This leads to an improved lower bound:

\[
\mathcal{L}^1(N) = \mathcal{L}^0(N) + \mathcal{L}^0(x, y, z - \theta_j h_j)
\]

where \( \mathcal{L}^0(x, y, z - \theta_j h_j) \) is computed according to (8) excluding box \( j \), that is,

\[
\mathcal{L}^0(x, y, z - \theta_j h_j) = \max_{i \in \mathbb{B}(N) \setminus \{j\}} \left\{ v_i \min\left\{ \left[ \frac{x}{l_i} \right], \left[ \frac{y}{w_i} \right], \left[ \frac{z - \theta_j h_j}{h_i} \right], \delta_i(N) \right\} \right\}. \tag{10}
\]

If box type \( k \) is chosen to fill the leftover box \((x, y, z - \theta_j h_j)\) we determine the approximated integer value to \( \theta_k \) [see (9)], and we can define another lower bound:

\[
\mathcal{L}^2(N) = \mathcal{L}^1(N) + \mathcal{L}^0(x, y, z - \theta_j h_j - \theta_k h_k).
\]

This procedure can be continued while the leftover box allows the fitting of any other available box to be packed. It is interesting to note that a good lower bound can substantially reduce the number of generated nodes (as we shall see later on in this section) allowing us to solve larger problems (since it avoids a lot of branchings), but its computational cost can be high and can take a long time (since it is evaluated at each node). A computational study is necessary in order to decide among \( \mathcal{L}^0, \mathcal{L}^1, \mathcal{L}^2 \), etc.

### Upper bounds

A simple upper bound can also be defined at node \( N \) that represents a box \((x, y, z)\). Consider a relaxation by taking only the constraint involving volume. So, we have an upper bound:

\[
\mathcal{U}(N) = \max \sum_{i \in \mathbb{B}(N)} v_i a_i
\]

subject to:

\[
\sum_{i \in \mathbb{B}(N)} (l_i w_i h_i) a_i \leq (x y z) \tag{10}
\]

\[0 \leq a_i \leq \delta_i(N), i \in \mathbb{B}(N).\]

If we eliminate the condition \( a_i \leq \delta_i(N), i \in \mathbb{B}(N) \), we will get a very easy upper bound given by:

\[
\mathcal{U}^0(N) = \max \{ v_i (x/l_i) (y/w_i) (z/h_i), i \in \mathbb{B}(N) \}
\]

with \( \mathcal{U}^0(N) = 0 \) if \( \mathbb{B}(N) = \emptyset \).

The solution to problem (10) can be obtained in the following way: Suppose that \( \mathbb{B}(N) = \{i_1, i_2, \ldots, i_j\} \) and

\[
(v_{i_1}/(l_{i_1} w_{i_1} h_{i_1})) \geq (v_{i_2}/(l_{i_2} w_{i_2} h_{i_2})) \geq \ldots \geq (v_{i_j}/(l_{i_j} w_{i_j} h_{i_j}))
\]
where
\[ \bar{a}_{ij} = \min \left\{ b_i(N), \frac{xyz}{(l_i w_i h_i)} \right\} \]
\[ \bar{a}_k = \min \left\{ b_k(N), \frac{xyz - \sum_{j=1}^{k-1} (l_j w_j h_j) \bar{a}_{ij}}{(l_k w_k h_k)} \right\}, k = 2, \ldots, r. \]

For simplicity, the computational results (given later) were produced with the lower bound \( \mathcal{L}^0 \) in (8) and the upper bound \( \mathcal{U}^0 \).

A branch and bound method

The previous lower and upper bounds, that we generically denote here by \( \mathcal{L} \) and \( \mathcal{U} \), can be used to implicitly enumerate some nodes of the graph. For convenience, every box in a final node \( N \) which is obtained through a 0-cut is filled in with a homogeneous packing (or composed homogeneous packing) and so it is valued \( \mathcal{L}(N) \).

Let \( \mathcal{V}(N) \) be the best value attached to node \( N \) at the present stage of the search, called the current value (it is given by a lower bound or by a known complete path emanating from \( N \)). The value of \( \mathcal{V}(N) \) is always updated when a better solution is obtained from the successors of \( N \). For example, if \( \mathcal{V}(N) < \mathcal{L}(N_1) + \mathcal{L}(N_2) \), where \((N_1, N_2)\) is a successor-pair of \( N \), then \( \mathcal{V}(N) \) is updated to \( \mathcal{L}(N_1) + \mathcal{L}(N_2) \).

Note that if \( \mathcal{V}(N) \geq \mathcal{U}(N_1) + \mathcal{U}(N_2) \), then \( N_1 \) and \( N_2 \) need not be explicitly considered. Also observe that if \( \mathcal{V}(N) = \mathcal{U}(N) \) then \( \mathcal{V}(N) \) provides the best value to node \( N \) and we say that the node is solved. These remarks characterize a branch and bound method (the branchings were defined previously, AND/OR-graph Approach). When the initial node is solved we have found the optimum packing pattern to the container \((L, W, H)\).

There are a number of strategies to traverse the graph, that is, different ways to choose a node to be branched. The number of nodes can be very large (in spite of implicit enumeration) and heuristic search strategies can be designed to discard a number of paths. In the Search Strategy we describe one which we implemented. Before we describe the search strategy we provide other heuristics to discard nonpromising paths.

HEURISTICS

The bounds defined in the previous section can also be used in order to produce heuristics that reduce the search space. These heuristics are used to eliminate branchings which seem unlikely to lead to optimal solutions. A greedy heuristic is also proposed to find good solutions to problem (4).

Heuristic 1 (H1)

Consider a node \( N \) and a successor-pair \((N_1, N_2)\). Consider also attached to node \( N \) an upper bound \( \mathcal{U} \) and the current value \( \mathcal{V} \) as defined in the previous section. We expect \( \mathcal{U}(N_1) + \mathcal{U}(N_2) \) to being 'substantially' larger than \( \mathcal{V}(N) \), otherwise it might be a sign that \( \mathcal{V}(N) \) will not be overcome by the branching, leading to \((N_1, N_2)\).

Let \( \lambda_1 \) be a previously defined fraction. We define heuristic H1 as:

If:
\[ (1 + \lambda_1) \mathcal{V}(N) \geq \mathcal{U}(N_1) + \mathcal{U}(N_2) \]
Then:
abandon the branching leading to \( N_1 \) and \( N_2 \).

Note that if \( \lambda_1 = 0 \), the previous procedure to discard branchings is not a heuristic anymore. In the algorithm, \( \lambda_1 \) will be taken approximately to zero.
Heuristic 2 (H2)

Consider again a node \( N \) and its successor-pair \((N_1,N_2)\). The lower bounds \( \mathcal{L}(N_1) \) and \( \mathcal{L}(N_2) \) yield a new feasible solution to node \( N \). Remember that the current value \( \mathcal{V}(N) \) can be greater, smaller or equal to \( \mathcal{L}(N_1) + \mathcal{L}(N_2) \). However, if \( \mathcal{V}(N) \) is substantially greater than \( \mathcal{L}(N_1) + \mathcal{L}(N_2) \), it might be a sign that \( \mathcal{V}(N) \) will not be overcome by the value to be obtained from this branching.

Initially we thought of abandoning all the branchings from \( N \) whose lower bounds were somewhere inferior to \( \mathcal{V}(N) \). However, as the search becomes deeper, better solutions for \( N \) are obtained and the value \( \mathcal{V}(N) \) is updated. As \( \mathcal{V}(N) \) becomes larger, it makes more difficult to carry out new branchings from \( N \). Hence we propose the use of \( \mathcal{L}(N) \) instead of \( \mathcal{V}(N) \) to compare with \( \mathcal{L}(N_1) + \mathcal{L}(N_2) \).

Let \( \lambda_2 \) be a previously defined fraction. Heuristic H2 is defined as:

\[
\text{If:} \quad \lambda_2 \mathcal{L}(N) \geq \mathcal{L}(N_1) + \mathcal{L}(N_2) \\
\text{Then:} \quad \text{abandon the branching leading to } N_1 \text{ and } N_2.
\]

Note that if \( \lambda_2 = 0 \), the above procedure to discard branchings is not a heuristic any more. In the algorithm, \( \lambda_2 \) will be taken to be approximately one.

Heuristic 3 (H3)

This heuristic deals with problem (4), since it is not an easy problem to be solved. We propose a greedy heuristic in order to get a good solution. Let \( \delta_i(N) \) be the maximum quantity of boxes type \( i \) that can be packed in the box at \( N \). Heuristic H3 is defined as follows: initially we consider node \( N_1 \) with the limit \( \delta_i(N), i = 1, \ldots, m \), and after determining a cutting pattern for the box represented by \( N_1 \), we consider node \( N_2 \) with the limit \( \delta_i(N) - \delta_i^1, i = 1, \ldots, m \), where \( \delta_i^1 \) is the quantity of boxes type \( i \) already used in \( N_1 \).

Observe that symmetry and ordering cut rules together with H3 consist of a new heuristic, since the order to get solution by the greedy heuristic is important. In Computational Results we consider the use of the symmetry rule or not.

A SEARCH STRATEGY

In this section we present the search strategy used to traverse the graph described previously (AND/OR-graph Approach). It is a hybrid strategy combining two basic strategies:

(i) **Back-tracking (BT)** is an important implementation of depth-first search. It always chooses the newly not-final generated node to be explored (one successor is generated).

(ii) **Hill-climbing (HC)** is a search strategy based upon local optimization that after expanding a node (all its successors are generated) chooses the best successor to be expanded, and discards all the remaining ones.

If a depth bound is imposed to BT, those strategies can be combined by firstly generating all nodes up to the depth bound (using BT strategy) and then the best path is chosen (using HC strategy) whose not-final nodes are again expanded up to the depth bound and so on.

Let DB be a positive integer number denoting a depth bound to BT.

**Algorithm BT–HC**

1. Let ROOT be a list that at the beginning contains only the initial node (a node in ROOT is called a root-node). Define DB the depth bound for each expanding from a root node.
2. While ROOT is not empty, do:
   3. Let \( s \) be the first node in ROOT. Generate all the successors of the root-node \( s \), using the BT strategy and respecting DB. Take \( s \) out from ROOT.
   4. Choose the most valuable path from \( s \) and discard the remaining paths (HC strategy). If there are nodes in this path whose depth is equal to DB and are not final, put them in ROOT.
Remarks

(i) In step 3, the generation of the successors from the root-node $s$ should take into account the rules discussed previously (AND/OR-graph Approach, Avoiding Equivalent Patterns, Stability of the Loading), the branch and bound method discussed in Lower and Upper Bounds and the heuristics discussed in the previous section.

(ii) For simplicity, an AND/OR-tree was implemented instead of an AND/OR-graph, that is, there is no checking test to see if a newly generated node has already been generated. Indeed, only the most valuable path is kept in the memory. This makes the BT–HC algorithm almost independent of memory limitation, quite the opposite to dynamic programming.

(iii) In step 4, each chosen path from $s$ corresponds to a section (with depth at most equal to $DB$) of the complete path from the initial node to the final nodes. Observe that the HC strategy, based upon local optimization in each section, does not ensure finding the most valuable path [i.e. optimum packing pattern to problem (3)] even in the absence of heuristics (see previous section).

COMPUTATIONAL RESULTS

The three methods previously described were implemented in Turbo-Pascal 5.5 and ran on an IBM-PC-486 compatible microcomputer (33 MHerz clock, 640 kbytes RAM, DOS 5.0). Remember that the first two are heuristic methods to solve the unconstrained three-dimensional problem but they do not solve the constrained three-dimensional problem. On the other hand, the last method (AND/OR-graph approach) is a heuristic method to solve both the unconstrained and the constrained problems. We used the lexicographic algorithm described in Gilmore and Gomory (1963) to find optimal solutions to the one-dimensional knapsack problems and the BT-HC algorithm presented in Morabito et al. (1992) to the two-dimensional packing problems. Note that we considered only guillotine packing patterns and therefore the solutions of the non-guillotine packing problems can be non-optimal.

Unconstrained problem

We generated 80 random examples which are grouped in the sets $S_1, S_2, \ldots, S_8$ of 10 examples in each (Table 1). The generation of these examples was done as following: for each set we fixed the values to $m, L, W, H$ and then the values of $l_i, w_i, h_i$ were randomly chosen such that $l_i/L, w_i/W, h_i/H$ belong to a uniform distribution in the interval $[\beta, \gamma]$ (see the table). To simplify, the utility values $v_i, i = 1, \ldots, m$, were defined as $v_i = l_i w_i h_i / L W H$, which corresponds in problem (3) to maximize the used volume (in percentage) of the container. In all these examples, the boxes must be loaded into the container following a fixed orientation but no loading stability is required.

In Tables 2, 3 and 4, the column Method specifies the solution method used: Layers (Solving the Problem by Layers), Stacks (Solving the Problem by Stacks) and BT/HC (A Search Strategy). For each set of 10 solved examples, the column Solution (s.d.) shows the mean value of the percentual used volume of the container (with the standard deviation in parentheses), the column Time shows the mean computational time (in seconds) and the column Number of nodes (concerned with BT/HC algorithm) shows the mean number of nodes generated by the graph search when a particular parameter is changed.

The sets are arranged to observe the behavior of the BT–HC algorithm when some parameters of
Table 2. Solution of sets S1, S2, S3

<table>
<thead>
<tr>
<th>Set</th>
<th>Method</th>
<th>DB</th>
<th>Solution (s.d.)</th>
<th>Time (sec.)</th>
<th>Number of nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>layers</td>
<td>0.7528 (0.0813)</td>
<td>0.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>stacks</td>
<td>0.6985 (0.0772)</td>
<td>&lt;0.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BT–HC</td>
<td>0.7798* (0.0756)</td>
<td>143.8</td>
<td>1,188,377</td>
<td></td>
</tr>
<tr>
<td>S2</td>
<td>layers</td>
<td>0.8569 (0.0483)</td>
<td>1.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>stacks</td>
<td>0.8138 (0.0326)</td>
<td>&lt;0.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BT–HC</td>
<td>0.8688* (0.0448)</td>
<td>275.8†</td>
<td>2,162,047</td>
<td></td>
</tr>
<tr>
<td>S3</td>
<td>layers</td>
<td>0.6694 (0.0702)</td>
<td>&lt;0.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>stacks</td>
<td>0.6207 (0.1141)</td>
<td>&lt;0.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BT–HC</td>
<td>0.7070* (0.0752)</td>
<td>0.3</td>
<td>2,025</td>
<td></td>
</tr>
</tbody>
</table>

*Optimal guillotine patterns.
†In 1/3 of examples the method failed to solve in 600 sec. and thus an alternate example was generated.

Table 3. Solution of set S4

<table>
<thead>
<tr>
<th>Set</th>
<th>Method</th>
<th>$\lambda_1$</th>
<th>$\lambda_2$</th>
<th>Solution (s.d.)</th>
<th>Time (sec.)</th>
<th>Number of nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>S4</td>
<td>layers</td>
<td>0.9581 (0.0309)</td>
<td></td>
<td>23.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>stacks</td>
<td>0.9374 (0.0339)</td>
<td></td>
<td>0.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BT–HC*</td>
<td>0.9697 (0.0217)</td>
<td></td>
<td>125.4</td>
<td>871,604</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.001</td>
<td>0.9697 (0.0216)</td>
<td></td>
<td>121.3</td>
<td>844,414</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.01</td>
<td>0.9696 (0.0216)</td>
<td></td>
<td>76.3</td>
<td>529,801</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.05</td>
<td>0.9562 (0.0152)</td>
<td></td>
<td>11.4</td>
<td>79,417</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0.9697 (0.0217)</td>
<td></td>
<td>91.4</td>
<td>510,193</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.95</td>
<td>0.9694 (0.0220)</td>
<td></td>
<td>45.2</td>
<td>177,808</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.001</td>
<td>0.9632 (0.0234)</td>
<td></td>
<td>2.0</td>
<td>1,336</td>
<td></td>
</tr>
</tbody>
</table>

*DB = 3.

The heuristics are changed. In Table 2, we fixed $\lambda_1 = 0$ and $\lambda_2 = 0$ (i.e. the heuristics H1 and H2 were annulled) and we initially varied the parameter DB (depth bound) (see previous section).

Remarks on Table 2

(i) DB = 4 was able to find optimal guillotine patterns for all the examples. The number of generated nodes was strongly reduced by DB. A few optimal guillotine patterns were found with DB = 2, but most of them were found with DB = 3. We decided to fix DB = 3 since there is a good trade-off between the computational time and the solution quality.

(ii) Solving the problem by stacks was better than solving it by layers. But the BT–HC algorithm produced better solutions than those two.

(iii) When the interval [0.15, 0.85] in S1 was reduced to [0.15, 0.50] in S2, the generated boxes (li, wi, hi) were, in general, smaller and 'closer to a cube' (i.e. li = wi = hi). This resulted in more difficult problems to solve, but with a better occupation of the available space in the container. However, when the interval was reduced to [0.25, 0.75], we had easier problems to solve.

In Table 3, the parameter DB is fixed at 3 and $\lambda_1$ and $\lambda_2$ are changed (see previous section). The column ($\lambda_1, \lambda_2$) shows the values for $\lambda_1$ and $\lambda_2$. 
Table 4. Solution of sets S5, S6, S7, S8

<table>
<thead>
<tr>
<th>Set</th>
<th>Method</th>
<th>M</th>
<th>Solution (s.d.)</th>
<th>Time (sec.)</th>
<th>Number of nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>S5</td>
<td>layers</td>
<td>∞</td>
<td>0.8621 (0.0705)</td>
<td>0.1</td>
<td>11,602</td>
</tr>
<tr>
<td></td>
<td>stacks</td>
<td>100</td>
<td>0.8879 (0.0441)</td>
<td>0.7</td>
<td>35,516</td>
</tr>
<tr>
<td></td>
<td></td>
<td>200</td>
<td>0.8911 (0.0475)</td>
<td>1.4</td>
<td>46,182</td>
</tr>
<tr>
<td></td>
<td></td>
<td>300</td>
<td>0.8959 (0.0705)</td>
<td>2.2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BT–HC*</td>
<td>100</td>
<td>0.9054 (0.0438)</td>
<td>2.2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>200</td>
<td>0.9104 (0.0472)</td>
<td>5.4</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>300</td>
<td>0.9140 (0.0469)</td>
<td>7.1</td>
<td></td>
</tr>
<tr>
<td>S6</td>
<td>layers</td>
<td>∞</td>
<td>0.8892 (0.0415)</td>
<td>0.2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>stacks</td>
<td>100</td>
<td>0.9203 (0.0299)</td>
<td>2.1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>200</td>
<td>0.9243 (0.0281)</td>
<td>7.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>300</td>
<td>0.9262 (0.0269)</td>
<td>14.9</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BT–HC*</td>
<td>100</td>
<td>0.9375 (0.0173)</td>
<td>7.8</td>
<td>31,684</td>
</tr>
<tr>
<td></td>
<td></td>
<td>200</td>
<td>0.9392 (0.0166)</td>
<td>20.6</td>
<td>80,655</td>
</tr>
<tr>
<td></td>
<td></td>
<td>300</td>
<td>0.9419 (0.0168)</td>
<td>56.0</td>
<td>217,773</td>
</tr>
<tr>
<td>S7</td>
<td>layers</td>
<td>∞</td>
<td>0.9350 (0.0190)</td>
<td>0.7</td>
<td></td>
</tr>
<tr>
<td></td>
<td>stacks</td>
<td>100</td>
<td>0.9464 (0.0088)</td>
<td>3.9</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>200</td>
<td>0.9475 (0.0102)</td>
<td>15.6</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>300</td>
<td>0.9524 (0.0102)</td>
<td>38.6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BT–HC*</td>
<td>100</td>
<td>0.9542 (0.0069)</td>
<td>16.2</td>
<td>39,672</td>
</tr>
<tr>
<td></td>
<td></td>
<td>200</td>
<td>0.9567 (0.0070)</td>
<td>73.5</td>
<td>176,531</td>
</tr>
<tr>
<td></td>
<td></td>
<td>300</td>
<td>0.9597 (0.0082)</td>
<td>161.6</td>
<td>385,741</td>
</tr>
<tr>
<td>S8</td>
<td>layers</td>
<td>∞</td>
<td>0.9500 (0.0125)</td>
<td>1.1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>stacks</td>
<td>100</td>
<td>0.9515 (0.0092)</td>
<td>7.2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>200</td>
<td>0.9560 (0.0094)</td>
<td>26.9</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>300</td>
<td>0.9585 (0.0085)</td>
<td>75.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BT–HC*</td>
<td>100</td>
<td>0.9586 (0.0093)</td>
<td>36.9</td>
<td>71,822</td>
</tr>
<tr>
<td></td>
<td></td>
<td>200</td>
<td>0.9642 (0.0080)</td>
<td>153.7</td>
<td>289,738</td>
</tr>
<tr>
<td></td>
<td></td>
<td>300</td>
<td>0.9660 (0.0078)</td>
<td>456.5</td>
<td>834,259</td>
</tr>
</tbody>
</table>

*DB = 3, λ1 = 0.01 and λ2 = 0.95.

Remarks on Table 3

(i) Now there is no guarantee of optimality given that DB = 3. For λ1 = 0 and λ2 = 0 or λ1 = 0.001 and λ2 = 0.90 we obtained the same solutions, but the search was considerably reduced. Good values for these parameters were λ1 = 0.01 and λ2 = 0.95 and so we decided to use them.

(ii) The volumetric occupation was better than those examples in Table 2 given that the interval [0.05, 0.50] allowed generating smaller boxes and m = 10.

The number of elements in the discretization sets X, Y and Z (see Avoiding Equivalent Patterns) is relatively small (less than 100) from S1 to S4 and so all elements were generated. On the other hand, this number is large (greater than 100) from S5 to S8 and we had to use a heuristic due to Beasley (1985) to reduce it (see also Morabito et al., 1992). Let | A | denote the number of elements in a set A. The new parameter M was defined to limit the size of the discretization sets, such that |X| ≤ M, |Y| ≤ M and |Z| ≤ M.

In Table 4, the column M shows the values of the parameter M. Note that the parameter M interferes in the three methods since the discretization sets are used to solve two-dimensional cutting problems (in the search graph for two-dimensional problems it was taken that DB = ∞, λ1 = λ2 = 0 in such a way the heuristics were annulled).

When M is increased from 100 to 300 we can improve the solution about 1%, but the computational time is significantly increased. We chose M = 100.

Constrained problem

In order to evaluate the performance of the BT–HC algorithm to solve constrained problems, we chose one example from 'real life' published by George and Robinson (1980). Table 5 presents the ordered boxes and their quantities to be packed into one container with available dimensions.
Table 5. George and Robinson (1980) example

<table>
<thead>
<tr>
<th>i</th>
<th>l_i</th>
<th>w_i</th>
<th>h_i</th>
<th>b_i</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>785</td>
<td>139</td>
<td>273</td>
<td>400</td>
</tr>
<tr>
<td>2</td>
<td>901</td>
<td>185</td>
<td>195</td>
<td>160</td>
</tr>
<tr>
<td>3</td>
<td>901</td>
<td>195</td>
<td>265</td>
<td>40</td>
</tr>
<tr>
<td>4</td>
<td>1477</td>
<td>135</td>
<td>195</td>
<td>40</td>
</tr>
<tr>
<td>5</td>
<td>614</td>
<td>480</td>
<td>185</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>400</td>
<td>400</td>
<td>135</td>
<td>16</td>
</tr>
<tr>
<td>7</td>
<td>264</td>
<td>400</td>
<td>400</td>
<td>80</td>
</tr>
<tr>
<td>8</td>
<td>385</td>
<td>365</td>
<td>290</td>
<td>40</td>
</tr>
</tbody>
</table>

784 available boxes

Table 6. Solution of George and Robinson (1980) example

<table>
<thead>
<tr>
<th>Symmetry</th>
<th>Orientation</th>
<th>Solution</th>
<th>Number of boxes</th>
<th>Volume (m³)</th>
<th>Time (sec.)</th>
<th>Number of nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>0.8948</td>
<td>781</td>
<td>26.206</td>
<td>81</td>
<td>147,217</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>0.8975</td>
<td>783</td>
<td>26.286</td>
<td>51</td>
<td>51,087</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>0.8951</td>
<td>782</td>
<td>26.215</td>
<td>1201</td>
<td>2,129,875</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>0.8989*</td>
<td>784</td>
<td>26.325</td>
<td>3529</td>
<td>3,679,397</td>
</tr>
</tbody>
</table>

*Optimal solution since all boxes were packed.

DB = 3, λ_1 = 0.01, λ_2 = 0.95 and M = 100.

(L, W, H) = (5793, 2236, 2261) mm. The utility values are v_i = l_i w_i h_i / LWH, i = 1, ..., 8. The resulting loading pattern must be stable and the boxes can be loaded on anyone of their six faces (no orientation is required).

We used those values of the parameters that ran well to solve unconstrained problems in the above section, that is, DB = 3, λ_1 = 0.01, λ_2 = 0.95 and M = 100. Table 6 shows the performance of the BT/HC algorithm when the symmetry rule is used or not and the orientation of boxes is required or not. The columns Symmetry and Orientation are specified by T (true) or F (false), respectively. In all the following solutions, we used the rule in the section on Stability of Loading to obtain loading stability.

The best solution obtained by the George and Robinson (1980) algorithm was to pack 783 boxes (0.8974, 26.283 m³) leaving out one box type 7 (0.0422 m³). For the BT–HC algorithm, when 783 boxes (0.8975, 26.286 m³) were packed, the box left out was type 4 (0.0389 m³). This explains the different values in the objective function. Note that the optimal solution of the problem was found by our algorithm (0.8989, 26.325 m³). This same solution can be also obtained with M = 50 (Morábito, 1992) but the number of nodes is still large.

CONCLUSIONS

In this paper we dealt with the container loading problem (a special case of the three-dimensional cutting and packing problem). We presented three heuristic methods to solve the so called unconstrained case, two of them reduce the three-dimensional problem into a number of one-dimensional and two-dimensional problems. The third method generalizes the AND/OR-graph approach proposed in Morábito and Arenales (1992) and can also solve the constrained case. Computational experiences with randomly generated and published examples show that the AND/OR-graph approach produce better solutions than other methods described in the literature.

Acknowledgement – The work of Reinaldo Morábito in this paper was supported in part by CNPq.

REFERENCES


