UNIVERSIDADE DE SÃO PAULO

An and/or graph approach to the container loading problem

REINALDO MORÁBITO

MARCOS N. ARENALES

Nº 001

NOTAS

Instituto de Ciências Matemáticas de São Carlos

ISSN - 0103-2577

An and/or graph approach to the container loading problem

REINALDO MORÁBITO

MARCOS N. ARENALES

Nº 001

N O T A S D O I C M S C

Série Computação

São Carlos

abr. / 1993

N O T A S D O I C M S C

SERIE: Estatistica

Nº 001/93 - ACHCAR, J.A. - Some aspects of reparametrization for statistical models

SERIE: Computacao

Nº 001/93 - MORABITO, R.; ARENALES, M.N. - An and/or graph to the container loading problem

SERIE: Matematica

Nº 001/93 - MICALI, A.; CHIBLOUN, R. - Dimension de Krull des algebres graduees II

Nº 002/93 - MICALI, A.; DIAS, I. - Singularites et algebres symetriques

Nº 003/93 - CARVALHO, A.N.; OLIVEIRA, L.A.F. - Delay-partial differential equations with some large diffu sions

Nº 004/93 - CARVALHO, A.N. - Infinite dimensional dynamics described by ordinary differential equations

AN AND/OR GRAPH APPROACH TO THE CONTAINER LOADING PROBLEM

Reinaldo Morábito
Depto de Engenharia de Produção
Universidade Federal de São Carlos
13565-905 São Carlos, São Paulo, Brazil
UFSCAREP@BRFAPESP.BITNET

Marcos N. Arenales
Depto de Computação e Estatística
Universidade de São Paulo
13560 São Carlos, São Paulo, Brazil
ARENALES@ICMSC.USP.ANSP.BR

Abstract: The container loading problem consists of packing boxes of various sizes into available containers in such a way as to optimise an objective function. In this paper we deal with the case where there is just one available container and the objective is to maximize the total volume (or utility value) of the loaded boxes. We present three solution methods (all of them are heuristics). Two of them solve the problem by layers or by stacks reducing the three dimensional nature of the problem into several problems with lower dimension. The third method consists of representing possible loading as a complete path 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 constrained problems by the AND/OR graph approach. In addition computational experiments are presented by solving a number of randomly generated examples and from the literature.

1. Introduction

The *container loading problem* can be seen as a special case of the *cutting and packing problems* involving the three spacial dimensions. Parallelepipeds (i.e. boxes) of various sizes have to be packed into (or cut from) bigger parallelepipeds (i.e. containers) in such a way to optimize certain functions. In addition to the geometric constraints (no overlap is allowed), other constraints have to be held, for example, the cargo loading stability and density.

Figure 1a illustrates a set of boxes to be loaded in a set of containers. Suppose that the containers are enough to arrange all the boxes. Consider initialy the following combinatorial optimization problem:

load all the boxes into the containers in such a way
the total volume of the used containers is minimized.

(1)

A subset of containers is chosen to load all the boxes. Figure 1b illustrates a possible solution. Note that if the containers are identical problem (1) can be stated 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:

load the maximum volume of boxes into the available containers.

(2)

In this way a subset of boxes have to be chosen and placed into the available containers. Figure 1c illustrates a possible solution.

Figure 1 - Loading Boxes Into Containers

Since the 1950's various authors have proposed different approaches to the one-dimensional or two-dimensional cutting and packing problems. A few of

them considered the three dimensions. Surveys are given in Dyckhoff (1990), Dowsland and Dowsland (1992) and Dyckhoff and Finke (1992). Sweeney (1992) and Dyckhoff and Finke gathered 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 the container size in terms of the cargo density permits a better use of its volumetric capacity. Haessler and Talbot (1990) have studied a three-dimensional loading problem considering only cargo of low density. The authors have also argued that in several situations the volumetric capacity limits the quantity of boxes, before weight constraints were reached (see also Gehring et al, 1990).

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

2. Problem Formulation

Consider a set of boxes which are grouped into m types. All boxes type i, $i=1,..,m$, 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, $k=1,..,n$, 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:

$$\textit{maximize} \sum_{i=1}^{m} v_i a_i$$

subject to the condition that the m-vector $(a_1, a_2, \ldots, a_m)$ *corresponds to a three-dimensional packing pattern.*

(3)

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 $\lfloor z \rfloor$ denote the largest integer number less than or equal to z. If the quantity b_i is not large enough (i.e., $b_i < \lfloor L/l_i \rfloor \lfloor W/w_i \rfloor \lfloor H/h_i \rfloor$), then the variable a_i has to be superiorly bounded by b_i. In this case the problem (3) is called a *constrained problem*, given by:

$$\textit{maximize} \sum_{i=1}^{m} v_i a_i$$

subject to the condition that the m-vector $(a_1, a_2, \ldots, a_m)$ *corresponds to a three-dimensional packing pattern and* $a_i \le b_i$, $i=1,\ldots,m$.

(4)

The purpuse of this work is to present solution methods to problems (3) and (4) and to show their computational performance. The additional constraints $a_i \le b_i$, i=1,..,m in problem (4) impose considerable difficulties in solving the problem. This will be discussed in section 5, where an *AND/OR-graph approach* is proposed to solve the problem. The methods in sections 3 and 4 are limited to deal with problem (3).

3. Solving the Problem by Layers

In order to get a three-dimensional packing pattern we simplify the

possibilities in two phases. Firstly, horizontal layers 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 (i.e., an one-dimensional knapsack problem is solved).

3.1. *Phase I*

In the first phase, the boxes with height h_j are chosen and arranged into layers of (L,H,h_j) (Figure 2a). Let λ_{ij} be the number of boxes of type i in the layer j and

$$\mathbb{H}_j = \{\ i \mid h_i = h_j\ ,\ i=1,..,m\ \}$$

(if $h_k=h_j$ with $k\neq j$, then $\mathbb{H}_k$ can be eliminated). Let V_j be defined by

$$V_j = \textit{maximum} \sum_{i\in\mathbb{H}_j} v_i\lambda_{ij}$$
$$\textit{subject to the condition that the } m\textit{-vector } (\lambda_{1j},\lambda_{2j},\ldots,\lambda_{mj})$$
$$\textit{corresponds to a two-dimensional packing pattern}$$
$$\textit{where the rectangles } (l_i,w_i),\ i=1,..,m,\ \textit{are arranged into } (L,W) \tag{5}$$

Figure 2 - Packing Boxes by Layers

3.2. *Phase II*

The layers are finally chosen and piled up to the height H of the container (Figure 2b). Let μ_j be the number that layer j is used. So, we have the knapsack problem to be solved:

$$\textit{maximize} \sum_{j=1}^{m} V_j\mu_j$$
$$\textit{subject to:} \sum_{j=1}^{m} w_j\mu_j \leq H$$
$$\mu_j\geq 0 \textit{ and integer } j=1,..,m. \tag{6}$$

The final value to variable a_i in (3) is obtained by:

$$a_i = \sum_{j=1}^{m} \lambda_{ij}\mu_j \quad i=1,..,m$$

This scheme of solving problem by layers generalizes the method of Gilmore and Gomory (1965) for two-dimensional problems with 2-staged cutting pattern. Of course, the problem (5) which is to build a layer can be solved imposing the constraint of two stages and so only one one-dimensional knapsack problem should be solved. In the next section we give another way to generalize the method of Gilmore and Gomory.

4. Solving the Problem by Stacks

Similarly to the method in section 3, we first define stacks of height H by piling up boxes (i.e., various one-dimensional knapsack problems are solved) and then we chose which stacks should build the loading pattern by arranging them on the floor of the container (i.e., a two-dimensional packing problem is solved). This procedure was proposed by Gilmore and Gomory (1965) but we did not find computational experiences using it in the literature.

4.1. *Phase I*

We define stacks (ℓ_j, w_k, H) and then we fill in these stacks with boxes (ℓ_i, w_i, h_i) such that $i \in \mathbb{LW}_{jk}$ where

$$\mathbb{LW}_{jk} = \{\ i \mid l_i \le l_j \text{ e } w_i \le w_k,\ i \in M\ \}, \quad j=1,..,m \text{ e } k=1,..,j$$

(If the number of stacks are too large, we can reduce them by considering only m stacks (ℓ_j, w_j, H), j=1,..,m (Figure 3a)).

Note that they can be built in identical sets and in this case only one of them should be kept. For example, if $w_k \le w_j$, $j \ne k$, or if $l_j \ne l_{j+1}$, $j=1,..,m-1$, then $\mathbb{LW}_{jk}$ can be disconsidered.

Let λ_{ijk} be the number of boxes of type i in the stack jk (i.e., a stack

with the dimensions (ℓ_j, w_k, H)). We define V_{jk} by

$$V_{jk} = \text{maximum} \sum_{i \in \mathbb{L}\mathbb{H}_{jk}} v_i \lambda_{ijk}$$
$$\text{subject to:} \sum_{i \in \mathbb{L}\mathbb{H}_{jk}} h_i \lambda_{ijk} \leq H$$
$$\lambda_{ijk} \geq 0 \text{ and integer, } i=1,\ldots,m. \tag{7}$$

Figure 3 - Packing Boxes by Stacks

4.2. *Phase II*

In a second phase, the stacks jk valued V_{jk} are chosen and arranged on the floor of the container that is a rectangle (L,W) (Figure 3b). Let μ_{jk} denote the number of stacks jk to be used in this arrangement. Then we have the two-dimensional packing problem to be solved:

$$\text{maximize} \sum_{j=1}^{m} \sum_{k=1}^{j} V_{jk} \mu_{jk}$$
$$\text{subject to the condition that the } m\text{-vector } (\mu_{11}, \mu_{21}, \mu_{22} \ldots, \mu_{mm})$$
$$\text{corresponds to a two-dimensional packing pattern where}$$
$$\text{the rectangles } (l_j, w_k),\ j=1,..,m \text{ and } k=1,..,j \text{ are arranged into } (L,W). \tag{8}$$

The value of the variable a_i is finally given by:

$$a_i = \sum_{j=1}^{m} \sum_{k=1}^{j} \lambda_{ijk} \mu_{jk}, \quad i=1,..,m$$

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. Note that these two methods deal with unconstrained problems only. 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. In addition, this approach can deal with the constrained problems.

5. AND/OR-Graph Approach

In the following, we describe a procedure to generate all possible unconstrained and constrained three-dimensional guillotine cutting patterns.

5.1. *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 demanded boxes (l_i,w_i,h_i), i=1,..,m. Figure 4a shows a sequence of cuts performed on the container (L,W,H) (A in the figure), producing the cutting pattern illustrated in Figure 4b.

Figure 4 - Sequence of Cuts and Corresponding Cutting Pattern

Initially a guillotine cut is done on the length L of container A (cut 1 in Figure 4b), 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 figure 4b) 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 Figure 4b (note that the intermediary boxes B and C are not indicated in this pattern but only the final boxes).

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 (see Figure 4a) 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 $l<min\{l_i, i=1,..,m\}$, $w<min\{w_i, i=1,..,m\}$ or $h<min\{h_i, i=1,..,m\}$ 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 anymore 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 Figure 4a). 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 a demanded box (l_i,w_i,h_i) 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 in 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. In section 10 we present a search strategy to solve container loading problems. Morabito 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

infeasible. 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 Figure 5). In section 6 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 section 8 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 infeasible. In section 9 we show how the bounds can also be used to define heuristics that enable us to find good and computationally feasible solutions.

Figure 5 - Three Equivalent Cutting Patterns

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.

5.2. *Constrained Pattern*

If there exists a limit b_i on the number of boxes type i (available to be packed) in a pattern (i.e., in the container) and $b_i < \lfloor L/l_i \rfloor \lfloor W/w_i \rfloor \lfloor H/h_i \rfloor$, $i=1,..,m$, then the problem is called constrained (problem (4)). 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 $\bar{b}_i(N)$ be the maximum quantity of boxes type i that can be packed in the box at N. (If N is the initial node, then $\bar{b}_i(N) = b_i$, $i=1,..,m$). Let (N_1, N_2) be a pair of successors from N, obtained by a guillotine cut. The following problem has to be solved.

$$\text{maximize} \sum_{i=1}^{m} v_1 a_i^1 + v_1 a_1^2$$

$$\text{subject to: } (a_1^1, a_2^1, \ldots, a_m^1) \text{ is a guillotine packing pattern for } N_1,$$

$$(a_1^2, a_2^2, \ldots, a_m^2) \text{ is a guillotine packing pattern for } N_2,$$

$$a_i^1 + a_i^2 \le \bar{b}_i(N), \quad i=1,\ldots,m. \tag{9}$$

It is not a trivial task to solve problem (9). In section 9 we provide a heuristic solution for it. Note that if $\bar{b}_i(N)$, $i=1,\ldots,m$ were large enough (unconstrained problem), problem (9) could be decomposed into two independent problems for solving N_1 and N_2.

6. Avoiding Equivalent Patterns

In this section we present some rules to avoid equivalent patterns.

6.1. *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 pieces. This can be extended to three-dimensional problems. That is, we can reduce the cuts along the length L to the set $\mathbb{X}$:

$$\mathbb{X} = \{ x \mid x = \sum_{i=1}^{m} \alpha_i l_i,\ 1 \le x \le L - l_0,\ 0 \le \alpha_i \le b_i \text{ and integer} \}$$

where $l_0 = \text{minimum}\{ l_i,\ i=1,\ldots,m \}$, the cuts along the width W to the set $\mathbb{Y}$:

$$\mathbb{Y} = \{ y \mid y = \sum_{i=1}^{m} \beta_i w_i,\ 1 \le y \le W - w_0,\ 0 \le \beta_i \le b_i \text{ and integer} \}.$$

where $w_0 = \text{minimum}\{ w_i,\ i=1,\ldots,m \}$, and the cuts along the height H to the set $\mathbb{Z}$:

$$\mathbb{Z} = \{ z \mid z = \sum_{i=1}^{m} \gamma_i h_i,\ 1 \le z \le H - h_0,\ 0 \le \gamma_i \le b_i \text{ and integer} \}.$$

where $h_0 = \text{minimum}\{ h_i,\ i=1,\ldots,m \}$.

The sets $\mathbb{X}$, $\mathbb{Y}$ and $\mathbb{Z}$ are called *discretization sets*. Later on we extend the Christofides and Whitlock formulae to generate the discretization sets.

6.2. *Symmetry*

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

$$\mathbb{X}(N) = \{ x_1 \mid x_1 = \sum_{i=1}^{m} \alpha_i l_i,\ 1 \le x_1 \le \lfloor x/2 \rfloor,\ 0 \le \alpha_i \le b_i \text{ and integer} \} \tag{10}$$

(analogously, sets $\mathbb{Y}(N)$ and $\mathbb{Z}(N)$). Note in (10) that the set $\mathbb{X}(N)$ does not depend on y nor z.

6.3. *Exclusion*

Consider again a node N representing a box (x,y,z). If a 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 of consideration in order to produce $\mathbb{X}(N)$ (see Figure 6). Similarly with $\mathbb{Y}(N)$, if $l_i>x$ or $h_i>z$ and with $\mathbb{Z}(N)$, if $l_i>x$ or $w_i>y$)). So,

$$\mathbb{X}(N) = \left\{ x_1 \mid x_1 = \sum_{i=1}^{m} \alpha_i l_i \text{ and } (\text{if } w_i>y \text{ or } h_i>z \Rightarrow \alpha_i=0),\ 1 \le x_1 \le \lfloor x/2 \rfloor,\ 0 \le \alpha_i \le b_i \text{ and integer} \right\} \tag{11}$$

(similarly with $\mathbb{Y}(N)$ and $\mathbb{Z}(N)$). Observe that $\mathbb{X}(N)$ now depends on y and z in (11)).

Figure 6 - $x_1 \notin \mathbb{X}(N)$

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 $\mathbb{X}(N)$, $\mathbb{Y}(N)$ and $\mathbb{Z}(N)$ are easily found for any node N.

Suppose $l_1 \le l_2 \le \ldots \le l_m$ 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,

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

$$l_i\le x_1\le\lfloor L/2\rfloor,$$

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

where $F_0(x_1)=\infty,\ x_1=1,\ldots,\lfloor L/2\rfloor$, and $F_i(0)=0,\ i=0,1,\ldots,m.$

The same can be done to the heights of the boxes. Let $G_i(x_1)$ be the minimum of the maximum heights of the boxes $1,2,\ldots,i$ whose lengths can be combined to sum x_1. So,

$$G_i(x_1)=min\left\{G_{i-1}(x_1);\ max\left\{h_i;\ \min_{\rho}\left\{G_{i-1}(x_1-\rho l_i),\ 1\le\rho\le min\{\lfloor x_1/l_i\rfloor,b_i\}\right.\right.\right.$$
$$\left.\left.\left.\text{and } \rho \text{ integer}\right\}\right\}\right\}$$

$$l_i\le x_1\le\lfloor L/2\rfloor,$$

$G_i(x_1) = G_{i-1}(x_1),\ x_1<l_i,$

where $G_0(x_1)=\infty,\ x_1=1,\ldots,\lfloor L/2\rfloor$, and $G_i(0)=0,\ i=0,1,\ldots,m.$

If $F_i(x_1)<\infty$ and $G_i(x_1)<\infty$ then $x_1=\sum_{j=1}^{i}\alpha_j l_j,\ 0\le\alpha_j\le b_j$ and α_j 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 $F_m(x_1)\le y$ and $G_m(x_1)\le z$, then $x_1\in X(N)$.

The discretization set $X(N)$ (N representing the box (x,y,z)) in (11) is

rewritten as

$$\mathbb{X}(N) = \{\ x_1 \mid F_m(x_1) \le y \text{ and } G_m(x_1) \le h,\ 1 \le x_1 \le \lfloor x/2 \rfloor\ \} \tag{12}$$

(analogously with $\mathbb{Y}(N)$ and $\mathbb{Z}(N)$).

6.4. *Cut Ordering*

Suppose that the node N representing the box (x,y,z) is cut at x_1, $x_1 \in \mathbb{X}(N)$ yielding (x_1,y,z) and $(x-x_1,y,z)$. Then suppose $(x-x_1,y,z)$ is cut at x_2, $x_2 \in \mathbb{X}(N)$, 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. Figure 7 illustrates these equivalent cutting patterns (similarly along the width y and height h).

Figure 7 - Equivalent Cutting Patterns

Christofides and Whitlock 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 in section 9, to arrive at a satisfatory solution to the problem (9).

7. Stability of the Loading

It is also necessary to consider the stability of the boxes which are piled up. Figure 8 depicts an unstable loading due to the nonoccupied space under box 3. Note that the methods in sections 3 and 4 tend to yield stable loadings.

Figure 8 - Unstable Loading

An additional condition can be imposed to the search graph in order to avoid unstable loading. For this, after a cut along the height, the new 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 section 8). 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 Figure 8 since after the cut, indicated by zz', the lower box should be cut at xx' to produce boxes 1 and 2. Figure 9a illustrates an intermediary box after cuts along the height and an unstable loading. After exchanging layers, figure 9b shows the same pattern but stable. Of course, this procedure, as well as the earlier ones, is not free of 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.

Figure 9 - Stable Loading

8. 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

$$\mathbb{B}(N) = \{\ i \mid l_i \le x,\ w_i \le y,\ h_i \le z \quad i=1,\ldots,m\ \}$$

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

8.1. *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:

$$\mathcal{L}^0(N) = \max_{i \in \mathbb{B}(N)} \{ v_i \min \{ \lfloor x/l_i \rfloor \lfloor y/w_i \rfloor \lfloor z/h_i \rfloor, \bar{b}_i(N) \} \}$$
$$= v_j \min \{ \lfloor x/l_j \rfloor \lfloor y/w_j \rfloor \lfloor z/h_j \rfloor, \bar{b}_j(N) \} \quad (13)$$

If $\bar{b}_j(N)$ is much smaller than $\lfloor x/l_j \rfloor \lfloor y/w_j \rfloor \lfloor z/h_i \rfloor$, the homogeneous cutting will produce a large waste which can be used to pack other boxes (see Figure 10).

Figure 10 - Layer Arrangement for a Homogeneous Cutting

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

$$\theta_j = \min \{ \lfloor z/h_j \rfloor , \bar{b}_j(N)/\lfloor x/l_j \rfloor \lfloor y/w_j \rfloor \} \quad (14)$$

(given that $\theta_j \le \lfloor z/h_j \rfloor$ and $\theta_j \lfloor x/l_j \rfloor \lfloor y/w_j \rfloor \le \bar{b}_j(N)$) which can be a non-integer number. By convenience, we approximate it to the nearest integer number, that is, $\theta_j \leftarrow \lfloor \theta_j + 0.5 \rfloor$. 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 (13) excluding box j, that is,

$$\mathcal{L}^0(x, y, z-\theta_j h_j) = \max_{i \in \mathbb{B}(N)-\{j\}} \{ v_i \min \{ \lfloor x/l_i \rfloor \lfloor y/w_i \rfloor \lfloor (z-\theta_j h_j)/h_i \rfloor, b_i(N) \} \}.$$

Figure 11 illustrates a possible solution provided by $\mathcal{L}^1$.

Figure 11 - A Composed Homogeneous Solution Provided by $\mathcal{L}^1$

If box type k is chosen to fill the left over box $(x, y, z-\theta_j h_j)$ we determine the approximated integer value to θ_k (see (14)), 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 left over box allows to 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.

8.2. *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 areas. So, we have an upper bound:

$$\begin{aligned} \mathcal{U}(N) = \text{maximum} & \sum_{i\in\mathbb{B}(N)} v_i a_i \\ \textit{subject to:} & \sum_{i\in\mathbb{M}(N)} (l_i w_i h_i) a_i \le (xyz) \\ & 0 \le a_i \le \bar{b}_i(N), \quad i\in\mathbb{B}(N). \end{aligned} \tag{15}$$

If we eliminate the condition $a_i \le \bar{b}_i(N)$, $i\in\mathbb{B}(N)$, 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 $\mathbb{B}(N) = \varnothing$.

The solution to problem (15) can be obtained in the following way: Suppose that $\mathbb{B}(N) = \{ i_1, i_2, \ldots, i_r \}$ and

$$\left[v_{i_1} \Big/ \left[l_{i_1} w_{i_1} h_{i_1}\right]\right] \ge \left[v_{i_2} \Big/ \left[l_{i_2} w_{i_2} h_{i_2}\right]\right] \ge \ldots \ge \left[v_{i_r} \Big/ \left[l_{i_r} w_{i_r} h_{i_r}\right]\right]$$

$$\mathcal{U}^1(N) = \sum_{j=1}^{r} v_{i_j} \bar{a}_{i_j}$$

where

$$\bar{a}_{i_1} = \min\{ \bar{b}_{i_1}(N),\ (xyz)/(l_{i_1} w_{i_1} h_{i_1}) \}$$

$$\bar{a}_{i_k} = \min\{ \bar{b}_{i_k}(N),\ [xyz - \sum_{j=1}^{k-1} (l_{i_j} w_{i_j} h_{i_j}) \bar{a}_{i_j}] \ /\ [l_{i_k} w_{i_k} h_{i_k}] \},\ k=2,\ldots,r$$

8.3. *A Branch & 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 homogenous packing (or composed homogenous packing) and so it is valued $\mathcal{L}(N)$.

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

Note that if $V(N)\geq\mathcal{U}(N_1)+\mathcal{U}(N_2)$, then N_1 and N_2 need not be explicitly considered. Also observe that if $V(N)=\mathcal{U}(N)$ then $V(N)$ 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 in section 5). 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 section 10 we describe one of them which we implemented. Before we describe the search strategy we provide other heuristics to discard nonpromissing paths.

9. Heuristics

The bounds defined in the previous section can also be used in order to produce heuristics that reduce the search space. These heuristics bet that some branchings do not lead to an optimal solution and so they are discarded. A *greedy* heuristic is also proposed to find good solutions to problem (9).

9.1. *Heuristic* 1 (*H*1)

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 section 9. 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 λ_1 be a previously defined fraction. We define heuristic *H*1 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, λ_1 will be taken approximately to zero.

9.2. *Heuristic* 2 (*H*2)

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, which becomes harder 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 λ_2 be a previously defined fraction. Heuristic *H*2 is defined as:

If: $\lambda_2 \mathcal{L}(N) \geq \mathcal{L}(N_1) + \mathcal{L}(N_2)$

Then: abandon the branching leading to N_1 and N_2.

Note that if $\lambda_2=0$, the above procedure to discard branchings is not a heuristic anymore. In the algorithm, λ_2 will be taken to approximatly one.

9.3. *Heuristic 3 (H3)*

This heuristic deals with problem (9), since it is not an easy problem to be solved. We propose a *greedy* heuristic in order to get a good solution. Let $\bar{b}_i(N)$ 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 $\bar{b}_i(N)$, $i=1,..,m$, and after determining a cutting pattern for the box represented by N_1, we consider node N_2 with the limit $\bar{b}_i(N)-\hat{a}_i^1$, $i=1,..,m$, where $\hat{a}_i^1$ is the quantity of boxes type i already used in N_1.

Observe that symmetry and ordering cut rules together with *H3* consist in a new heuristic, since the order to get solution by the greedy heuristic is important. In computational results (secton 11) we consider the use of symmetry rule or not.

10. A Search Strategy

In this section we present the search strategy used to traverse the graph described in section 5. It is a hybrid strategy combining two basic strategies:

(*i*) *BackTracking* (*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 *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 in sections 5, 6 and 7, the branch and bound method in section 8 and the heuristics discussed in section 9.

(*ii*) For simplicity, it was implemented an AND/OR-tree instead of an AND/OR-graph, that is, there is no checking test if a newly generated node has already been generated. Indeed, only the most valuable path is kept in memory. This turns the BT-HC algorithm almost independent of memory limitation, quite the opposite to dynamic programming. Note also that in this implementation step 4 is implicitly considered in step 3, we specify it to cause clarity of using *HC* strategy.

(*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 optmization in each section, does not ensure finding the most valuable path (i.e., optimum packing pattern to problem (3)) even in absent of heuristics from section 9.

11. Computational Results

The three methods described in sections 3, 4 and 5-10 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 ones are heuristic methods to solve the unconstrained three-dimensional problem (3) but they do not solve the constrained three-dimensional problem (4). On the other hand, the last method (AND/OR-graph approach) solves 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 in (6) and (7) and the BT-HC algorithm presented in Morábito et al (1992) to the two-dimensional packing problems in (5) and (8). Note that we considered only guillotine packing patterns and therefore the solutions of the non-guillotine packing problems (5) and (8) can be non-optimal.

11.1. *Unconstrained Problem*

We generated 80 random examples which were grouped in the sets S1, S2,..., S8 of 10 examples in each one (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 $[\alpha,\beta]$ (see the table). To simplify, the utility values v_i, i=1,...,m, were defined as $v_i = l_i w_i h_i / LWH$, which corresponds in problem (4) to maximize the occupied volume in the container. In all these examples, the boxes must be loaded into the container following a fixed orientation but no loading stability is required.

	S1	S2	S3	S4	S5	S6	S7	S8
m	5	5	5	10	5	10	20	30
$L=W=H$	100	100	100	100	1000	1000	1000	1000
α	0.15	0.15	0.25	0.05	0.05	0.05	0.05	0.05
β	0.85	0.50	0.75	0.50	0.50	0.50	0.50	0.50

Table 1 - Sets S1, S2, ..., S8

In Tables 2, 3 and 4, the column **method** specifies the solution method used: **layers** (section 3), **stacks** (section 4) and **BT/HC** (section 10). For each set of 10 solved examples, the column **solution (sd)** shows the mean value of the objetive function (with the standard deviation in parenthesis), the column **time** shows the mean computational time and the column **number of nodes** (concerned with BT/HC algorithm) shows the mean number of generated nodes 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 the heuristics are changed. In Table 2, we fixed $\lambda_1=0$ and $\lambda_2=0$ (i.e., the heuristics *H1* and *H2* do not discard optimal solution) and we initially varied the parameter *DB* (depth bound) (see section 10).

set	method	*DB*	solution (sd)	time	number of nodes
S1	layers		0.6985 (0.0772)	<0.1	
	stacks		0.7528 (0.0813)	0.5	
	BT-HC	∞	0.7798* (0.0756)	143.8	1,188,377
		4	0.7798 (0.0756)	0.6	7,210
		3	0.7798 (0.0756)	0.2	1,658
		2	0.7769 (0.0769)	0.1	331
S2	layers		0.8138 (0.0326)	<0.1	
	stacks		0.8569 (0.0483)	1.1	
	BT-HC	∞	0.8688* (0.0448)	275.8+	2,162,047
		4	0.8688 (0.0448)	8	77,374
		3	0.8683 (0.0448)	1	11,385
		2	0.8654 (0.0460)	<0.1	1,127
S3	layers		0.6207 (0.1141)	<0.1	
	stacks		0.6694 (0.0702)	<0.1	
	BT-HC	∞	0.7070* (0.0752)	0.3	2,025
		4	0.7070 (0.0752)	0.1	1,165
		3	0.7070 (0.0752)	0.1	476
		2	0.7067 (0.0750)	0.1	136

* optimum solution

\+ examples that were not solved in at least 600 seconds were changed. There was 1/3 of rejection

Table 2 - Solution of Sets S1, S2, S3

Remarks on Table 2

(*i*) *DB*=4 was able to find optimal solutions for all the examples. The number of generated nodes was strongly reduced by *DB*. A few optimal solutions 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 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 (l_i, w_i, h_i) were, in general, smaller and "closer to a cube" (i.e., $l_i=w_i=h_i$). This resulted in more difficult problems to solve, but with a better ocupation 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 by 3 and λ_1 and λ_2 are changed (see section 9). The column (λ_1,λ_2) shows the mean values for λ_1 and λ_2.

set	method	λ_1	λ_2	solution (sd)	time	number of nodes
S4	layers			0.9374 (0.0339)	0.4	
	stacks			0.9581 (0.0309)	23.2	
	BT-HC*	0	0	0.9697 (0.0217)	125.4	871,604
		0.001	0	0.9697 (0.0216)	121.3	844,414
		0.01	0	0.9696 (0.0216)	76.3	529,801
		0.05	0	0.9562 (0.0152)	11.4	79,417
		0	0.90	0.9697 (0.0217)	91.4	510,193
		0	0.95	0.9694 (0.0220)	45.2	177,808
		0	1.00	0.9632 (0.0234)	2.0	1,336
		0.01	0.95	0.9694 (0.219)	24.4	94,530

*DB=3

Table 3 - Solution of Set S4

Remarks on Table 3

(*i*) Now there is no guarantee of optimality given that *DB*=3. For λ_1=0 and λ_2=0 or λ_1=0.001 and λ_2=0.90 we obtained the same solutions, but the search was

considerably reduced. Good values for these parameters were λ_1=0.01 and λ_2=0.95 and so we decided to use them.

(*ii*) The volumetric ocupation was better than those examples in Table 1 given that the interval [0.05, 0.50] allowed generating smaller boxes and m=10.

The number of elements in the discretization sets $\mathbb{X}$, $\mathbb{Y}$ and $\mathbb{Z}$ (see section 6) is relatively small (less than 100) from S1 to S4 and so forth, 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 Morábito et al, 1992). Let $|\mathbf{A}|$ denoting the number of elements in the set $\mathbf{A}$. The new parameter M was defined to limit the size of the discretization sets, such that $|\mathbb{X}| \leq M$, $|\mathbb{Y}| \leq M$ and $|\mathbb{Z}| \leq M$.

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 DB=∞, λ_1=λ_2=0 in such a way the heuristics were annulled).

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

* DB=3, λ_1=0.01 and λ_2=0.95

Table 4 - Solution of Sets S5, S6, S7, S8

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.

11.2. *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 demanded boxes and their quantities to be packed into one container with avaiable dimensions $(L,W,H)=(5793,2236,2261)$ 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).

i	l_i	w_i	h_i	b_i
1	785	139	273	400
2	901	185	195	160
3	901	195	265	40
4	1477	135	195	40
5	614	480	185	8
6	400	400	135	16
7	264	400	400	80
8	385	365	290	40
				784 available boxes

Table 5 - George and Robinson Example

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 how the performance of BT/HC algorithm is when the symmetry rule is used or not and the orientation of boxes are required or not. The columns **symmetry** and **orientation** are specified with **T** (true) or **F** (false), respectively. In all the following solutions, we used the rule in section 7 to get loading stability.

symmetry	orientation	solution	number of boxes	volume (m^3)	time (sec)	number of nodes
T	T	0.8948	781	26.206	81	147,217
T	F	0.8975	783	26.286	51	51,087
F	T	0.8951	782	26.215	1201	2,129,875
F	F	0.8989*	784	26.325	3529	3,679,397

* optimal solution since all boxes were packed

DB=3, λ_1=0.01, λ_2=0.95 and M=100

Table 6 - Solution of George and Robinson Example

The best solution obtained by George and Robinson algorithm was to pack 783 boxes (0.8974, 26.283 m^3) leaving out one box type 7 (0.0422 m^3). For the BT-HC algorithm, when 783 boxes (0.8975, 26.286 m^3) were packed, the box left out was type 4 (0.0389 m^3). 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 m^3). This same solution can be also obtained with *M*=50 (Morabito, 1992) but the number of nodes is still large.

12. Conclusions

In this paper we dealed with the *container loading problem* (a special case of the three-dimensional cutting and packing problem). We presented three 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 Morabito 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.

References

Beasley, J. (1985), "Algorithms for Unconstrained Two Dimensional Guillotine Cutting", *Journal of the Operational Research Society* 36, pp.297-306.

Bischoff, E. and Marriott ,M. (1990), "A Comparative Evatuation of Heuristic for Container Loading", *European Journal of Operational Research* 44, pp.267-276.

Christofides, N. and Whitlock, C. (1977), "An Algorithm for Two Dimensional Cutting Problems", *Operations Research* 25, pp.30-44.

Dowsland, K. and Dowsland, W. (1992), "Packing Problems", *European Journal of Operational Research* 56, pp.2-14.

Dyckhoff, H. (1990), "A Typology of Cutting and Packing Problems", *European Journal of Operational Research* 44, pp.145-159.

Dyckhoff, H. and Finke, U. (1992), *Cutting and Packing in Production and Distribution: Typology and Bibliography*, Springer-Verlag Co, Heildelberg.

Gehring, H., Menschner, K. and Meyer, M. (1990), "A Computer Based Heuristic for Packing Pooled Shipment Containers", *European Journal of Operational Research* 44, pp.277-288.

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

Gilmore, P. and Gomory, R. (1963), "A Linear Programming Approach to the Cutting Stock Problem - Part II", *Operations Research* 11, pp.863-888.

Gilmore, P. and Gomory, R. (1965), "MultiStage Cutting Stock Problems of Two and More Dimensions", *Operations Research* 14, pp.1045-1074.

Haessler, R. and Talbot,F. (1990), "Load Planning for Shipments of Low Density Products", *European Journal of Operational Research* 44, pp.289-299.

Herz, J. (1972), "Recursive Computational Procedure for Two Dimensional Stock Cutting", *IBM Journal of Research and Development* 16, pp.462-469.

Morábito, R., (1992) "Uma Abordagem em Grafo-E/OU para o Problema do Empacotamento: Aplicação ao Carregamento de Paletes e Contêineres", *Tese de Doutorado*, Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, Brazil.

Morábito, R. and Arenales, M. (1992), "Staged and Constrained Two-Dimensional Guillotine Cutting Problems: A New Approach", *Notas* 126, ICMSC, Universidade de São Paulo, Brazil (*presented at the ORSA/TIMS Conference*, San Francisco, *November* 1992 *and submitted to EJOR*).

Morábito, R., Arenales, M. and Arcaro, V. (1992), "An And/Or-Graph Approach for Two-Dimensional Cutting Problems", *European Journal of Operational Research* 58(2), pp.263-271.

Nilsson, N. (1971), *Problem Solving Methods in Artificial Intelligence*. McGraw-Hill, New York.

Pearl, J. (1984), *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley Publishing Company, Reading, MA.

Sweeney, P. and Paternoster, E. (1992), "Cutting and Packing Problems: A Categorized, Application-Oriented Research Bibliography", *Journal of the Operational Research Society* 43, pp.691-706.

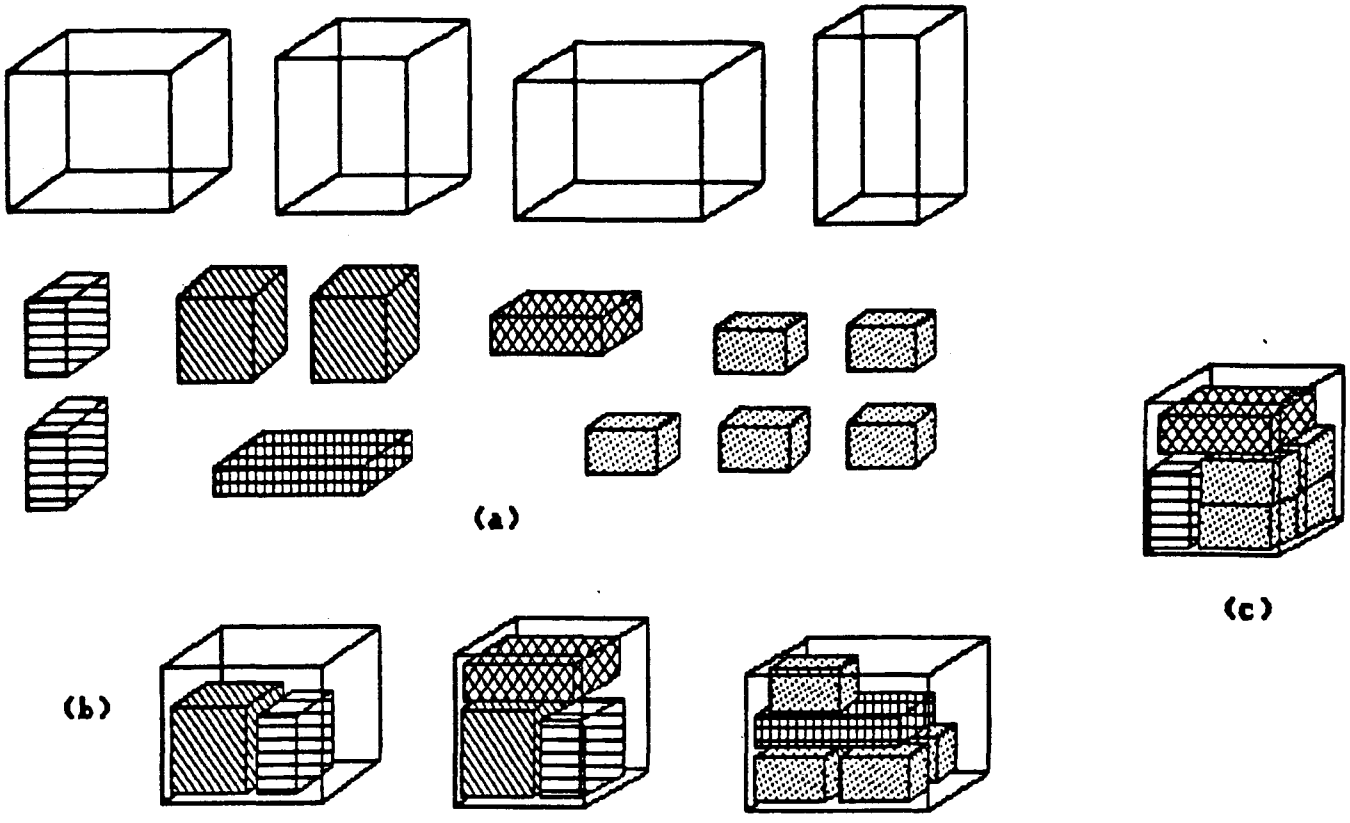


Figure 1 - Loading Boxes Into Containers

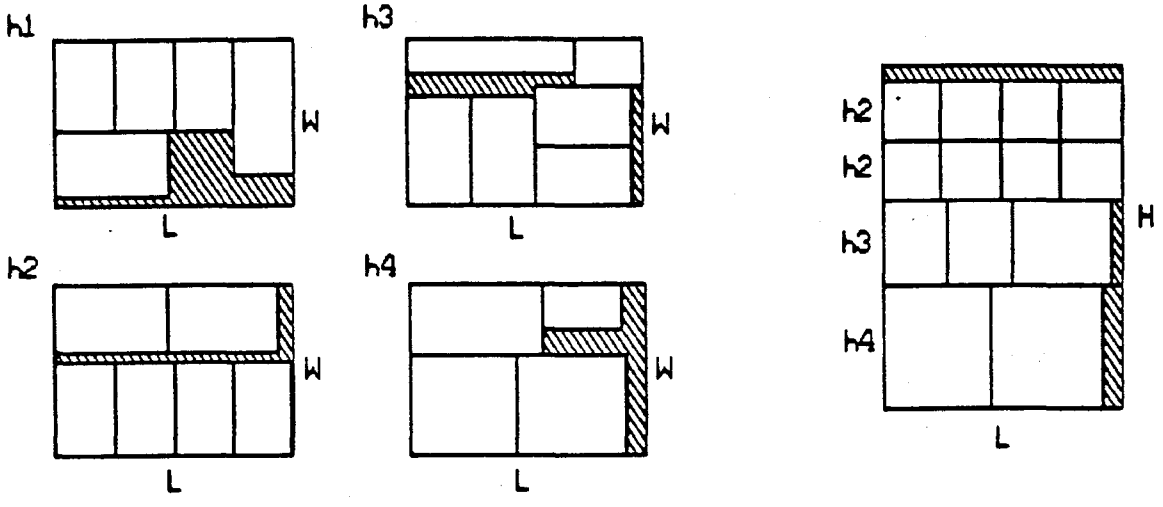
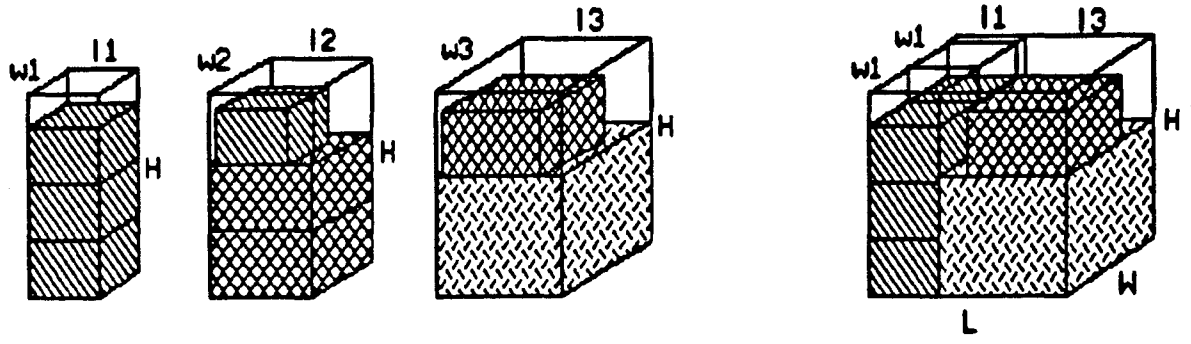


Figure 2 - Packing Boxes by Layers



(a) Phase I

(b) Phase II

Figure 3 - Packing Boxes by Stacks

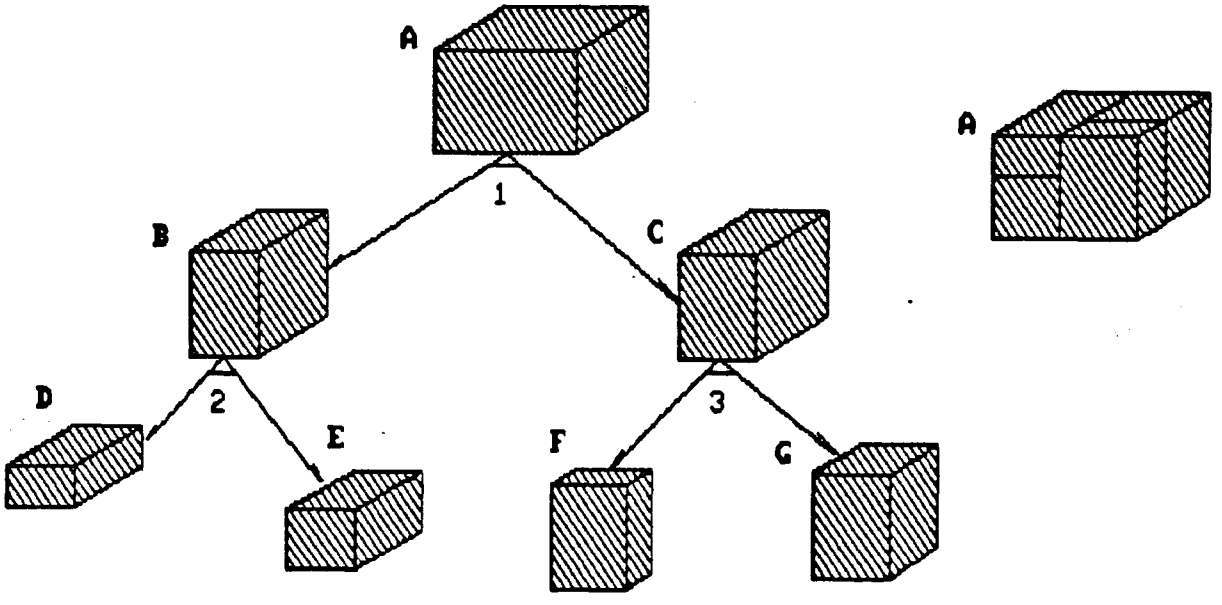


Figure 4 - Sequence of Cuts and Corresponding Cutting Pattern

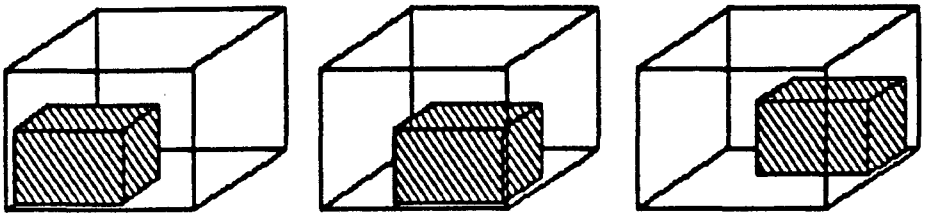


Figure 5 - Three Equivalent Cutting Patterns

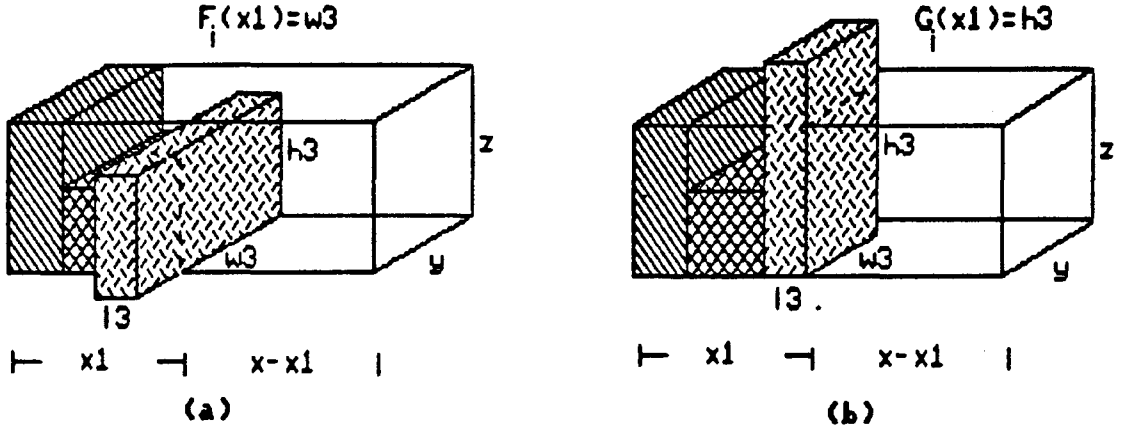


Figure 6 - Exclusion

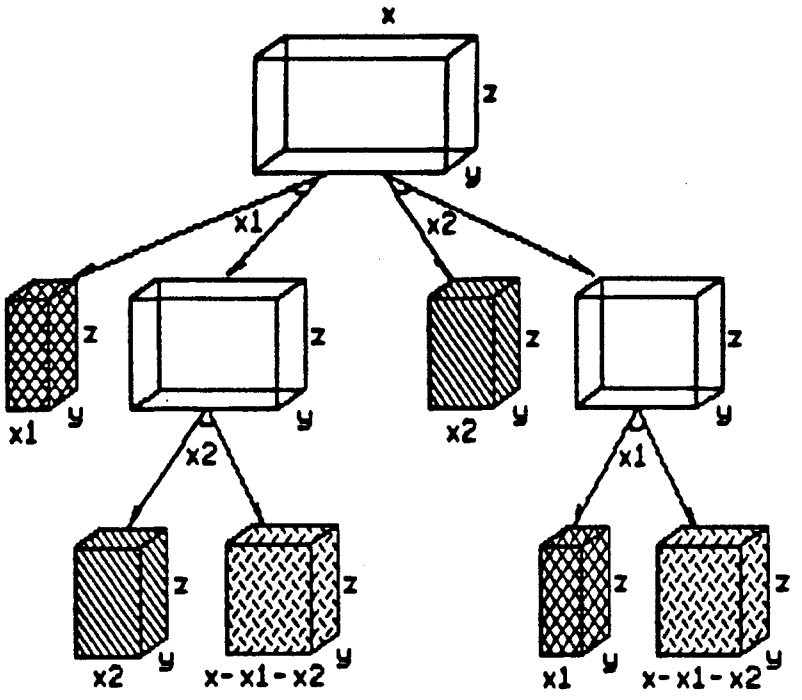


Figure 7 - Equivalent Cutting Patterns

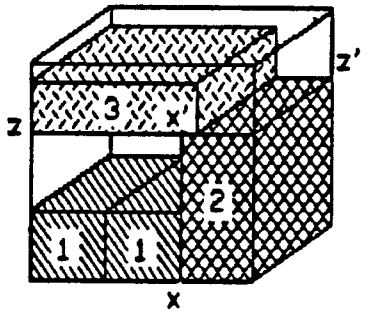


Figure 8 - Unstable Loading

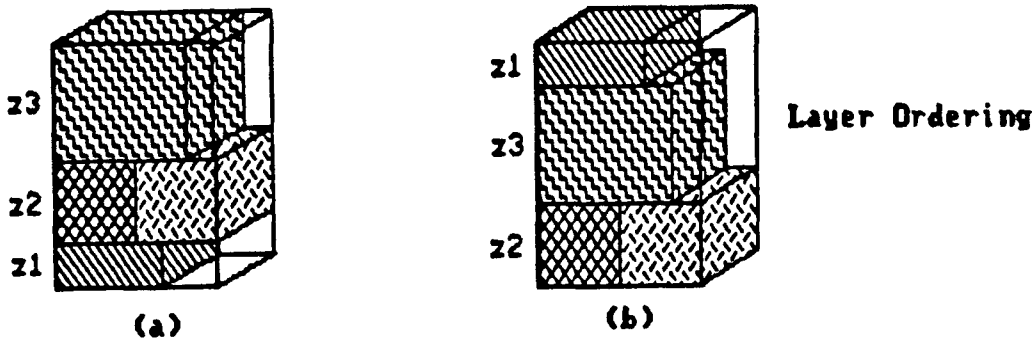


Figure 9 - Unstable and Stable Loading

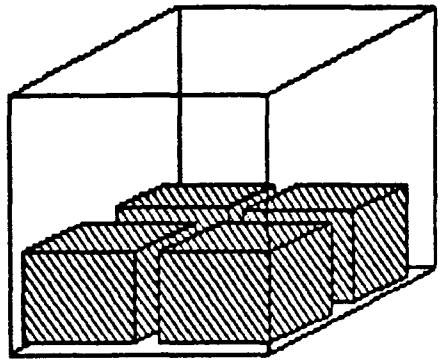


Figure 10 - Layer Arrangement for a Homogeneous Cutting

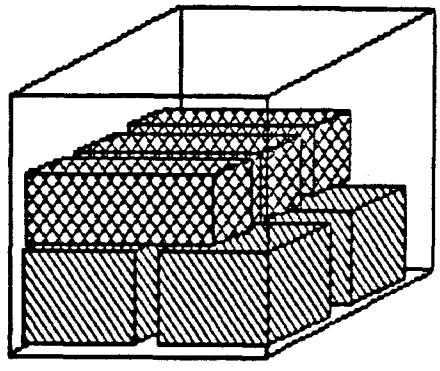


Figure 11 - A Composed Homogeneous Solution