A Branch and Cut Algorithm for the Pallet Loading Problem

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

[a] *University of Valencia, Department of Statistics and Operations Research, 46100 Burjassot, Valencia, Spain*

[b] *University of Castilla-La Mancha. Departamento de Informatica, E.Politecnica Superior, Campus Universitario, 02071 Albacete, Spain*

Abstract

We propose a branch and cut algorithm for the Pallet Loading Problem. The 0-1 formulation proposed by Beasley for cutting problems is adapted to the problem, adding new constraints and new procedures for variable reduction. We then take advantage of the relationship between this problem and the maximum independent set problem to use the partial linear description of its associated polyhedron. Finally, we exploit the specific structure of our problem to define the solution graph and to develop efficient separation procedures. We present computational results for the complete sets Cover I (up to 50 boxes) and Cover II (up to 100 boxes).

Key words: Pallet loading; Packing; Maximum independent set;

1 Introduction

The Pallet Loading Problem (PLP) arises whenever identical rectangular boxes have to be packed on a rectangular pallet. Though the problem is initially three-dimensional, practical considerations usually mean that the boxes must be placed orthogonally with respect to the edges of the pallet, and in layers in which the vertical orientation of the boxes is fixed. With these restrictions the

⋆ This work has been partially supported by the Spanish Ministry of Science and Technology, DPI2002-02553

* Corresponding author.
Email addresses: ramon.alvarez@uv.es (R. Alvarez-Valdes), fparreno@info-ab.uclm.es (F. Parreño), jose.tamarit@uv.es (J.M. Tamarit).

problem becomes the two-dimensional problem of packing a large rectangle, a *pallet*, with the maximum number of small identical rectangles, *boxes*.

The problem has many practical applications in distribution and logistics. An increase in the number of boxes which can be shipped on a pallet directly leads to a decrease in costs. The problem has therefore attracted a lot of research in recent decades and many heuristic methods have been developed. On the one hand, constructive methods of increasing complexity, from simple structures in which the pallet is divided into blocks (Steudel[33], Smith and De Cani[32], Bischoff and Dowsland[6],Young-Gun and Maing-Kyu[37]), to the recursive procedures proposed by Morabito and Morales[25], by Scheithauer and Terno [31], based on G-4 structures, and more recently by Lins et al.[23], based on L-shaped structures. On the other hand, there are metaheuristics based on tabu search, genetic algorithms and strategic oscillation (Amaral and Wright[1], Dowsland[16], Herbert and Dowsland[19]). Several upper bounds have also been proposed (Nelissen[27], Dowsland[12],[13], Letchford and Amaral[24]), which consider the geometric structure of the problem and linearly relax integer programming formulations.

The exact algorithms proposed so far are basically tree search procedures in which at each node a partial layout of boxes on the pallet has been built. Different ways of adding boxes, extending the partial solution, and different bounding procedures define the different algorithms (De Cani[11], Iserman[22], Exeler [18], Bhattacharya et al.[5]). Special mention should be made of the work by Dowsland [14],[15] who also follows a constructive approach, but based on a very interesting equivalence to the maximum independent set on a *pallet loading* graph. These exact algorithms have been able to solve problems of only moderate size, that is of up to 50 boxes.

The purpose of the present paper is to develop an exact algorithm, based on branch and cut, to solve larger problems, of up to 100 boxes. The structure of the paper is as follows. In Section 2, we formally define the problem and review the main concepts, especially the idea of equivalence classes. In Sections 3 to 8 we present the main components of the algorithm, introducing the integer formulation of the problem, describing the separation procedures, heuristic algorithms based on rounding LP solutions, branching strategies and logical fixing of variables. In Section 9 we present the computational results. Finally, concluding comments are made in Section 10.

2 Problem formulation and basic concepts

The problem can be formulated as follows: Given a large rectangle, a *pallet*, with given length L and width W, and a small rectangle, a *box*, with length a

and width b, the number of $a*b$ boxes packed onto the $L*W$ pallet has to be maximized. Each instance of the PLP is denoted by a quadruple (L, W, a, b). A horizontally oriented box is referred to as an *H-box*, while a vertically oriented is referred to as a *V-box*. The positions of the boxes are given with respect to a coordinate system which originates in the bottom left corner of the pallet.

Given an instance (L, W, a, b), a pair (n, m) of non-negative integers is a *feasible partition* of S (L or W) if $n * a + m * b \leq S$. If (n, m) satisfy $0 \leq S - n * a - m * b < b$, they are an *efficient partition*. Dowsland[12] showed that two instances of the PLP are equivalent, that is, have the same feasible solutions, if they have the same efficient partitions for both the length and the width of the pallet. Hence, if we solve one instance of an equivalence class, in particular that with the lowest dimensions, we have solved all the other instances of the class.

The concept of equivalence class also determines the sets of test instances used for the PLP. Randomly generating and solving a set of instances would suppose solving equivalent problems more than once. It seems more reasonable to work with equivalence classes. Two sets have been proposed[15]: *Cover I*, the set of equivalence classes of instances satisfying:

$$1 \leq \frac{L}{W} \leq 2, \qquad 1 \leq \frac{a}{b} \leq 4, \qquad 1 \leq \frac{L*W}{a*b} < 51$$

and *Cover II*, the set of equivalence classes of instances satisfying:

$$1 \leq \frac{L}{W} \leq 2, \qquad 1 \leq \frac{a}{b} \leq 4, \qquad 51 \leq \frac{L*W}{a*b} < 101$$

Most of the authors mentioned above have used these sets totally or partially, especially Cover I. If we refer to exact procedures, Dowsland[14] tested her algorithm on the whole of Cover I, while Bhattacharya et al.[5] only used one subset of it and one instance of Cover II. In fact, not even heuristic algorithms have been tested on all the classes of Cover II. In this paper, we will test our branch and cut algorithm on the whole set Cover II, thus providing a complete picture of the easy and hard elements of the set.

The definition of sets Cover I and Cover II is subject to some ambiguity. Sometimes an instance satisfies the conditions defining the set but another equivalent instance does not. In these cases, the inclusion of the equivalence class in the set is not clear and depends on the user. In order to avoid this ambiguity, we have followed the constructive method proposed by Dowsland[15] to generate all the equivalence classes for which the lowest dimensional instance strictly satisfies the defining conditions of the set. Cover I produced in this way has 7827 classes, and Cover II has 40609 classes.

3 An integer formulation

A branch and cut algorithm is a branch and bound algorithm which may call a cutting plane algorithm at each node of the search tree. At each node we have an LP-relaxation of the problem. We solve this relaxation and if the solution is integer, we stop. Otherwise, we call the separation algorithms based on the families of valid inequalities we consider. If any violated inequalities are found, we add them to the LP relaxation and solve it again. If no violated inequalities are found, we have to branch from this node (for further details, see the book by Wolsey[36]). Therefore, the main elements of a branch and cut algorithm are the formulation, the separation procedures and the branching strategies, which we will describe in the next subsections. We also include some additional features, such as heuristics based on rounding LP-solutions and procedures for fixing variables and constraints.

3.1 Beasley's formulation

The PLP can be formulated as a particular case of Beasley's[3] 0-1 formulation for the two-dimensional non-guillotine cutting problem. We use two types of variables, defining the positions of the bottom left corners of the boxes:

$$h_{kj} = \begin{cases} 1 & \text{if a H-box is placed in position (k,j),} \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

$$v_{kj} = \begin{cases} 1 & \text{if a V-box is placed in position (k,j),} \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

The problem is:

$$\max \sum_{k=0}^{L-a} \sum_{j=0}^{W-b} h_{kj} + \sum_{k=0}^{L-b} \sum_{j=0}^{W-a} v_{kj} \tag{3}$$

subject to:

$$\sum_{k=max\{0,r-a\}}^{min\{r,L-a\}} \sum_{j=max\{0,s-b\}}^{min\{s,W-b\}} h_{kj} + \sum_{k=max\{0,r-b\}}^{min\{r,L-b\}} \sum_{j=max\{0,s-a\}}^{min\{s,W-a\}} v_{kj} \leq 1$$
$$(r = 0, \ldots, L-1; s = 0, \ldots, W-1), \tag{4}$$

$$h_{kj} \in \{0,1\} \qquad (0 \leq k \leq L-a; 0 \leq j \leq W-b) \tag{5}$$

$$v_{kj} \in \{0,1\} \qquad (0 \leq k \leq L-b; 0 \leq j \leq W-a) \tag{6}$$

Constraints (4) are *covering constraints* avoiding box overlapping. For each pair (r, s) the constraint (4) guarantees that the corresponding unit square is covered by one box at most.

The number of variables, $(L-a)*(W-b)+(L-b)*(W-a)$, and constraints, $(L-b)*(W-b)$, involved in this formulation can be very high for large problems, but it can be significantly reduced, as is shown in the next subsection.

3.2 Reducing the number of variables and constraints

The set of positions for the boxes can first be reduced to the *normal sets* proposed by Herz[20] and Christofides and Whitlock[10]

$$S(L) = S(L, a, b) = \{r : r = \alpha a + \beta b, r + b \leq L, \alpha, \beta \in \mathcal{Z}_+\} \tag{7}$$

$$S(W) = S(W, a, b) = \{r : r = \alpha a + \beta b, r + b \leq W, \alpha, \beta \in \mathcal{Z}_+\} \tag{8}$$

Note that the elements of these normal sets correspond to the points at which there would be a box in a feasible partition. As all the equivalent instances have the same feasible partitions, they will have the same normal sets and then the number of variables would be the same for all of them.

These sets can be further reduced using dominance considerations, as proposed by Scheithauer and Terno[31]. If we denote:

$$\langle s \rangle_L := max\{r \in S(L) : r \leq s\}$$

the new set of positions, *raster points*, are:

$$\widetilde{S}(L) = \widetilde{S}(L, a, b) = \{\langle L - r \rangle_L : r \in S(L)\} \tag{9}$$

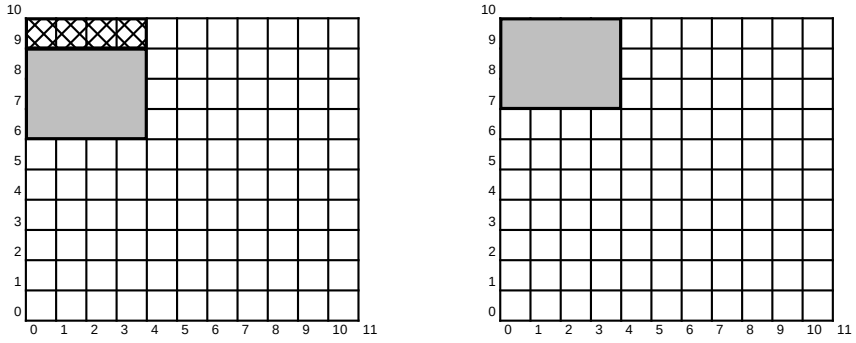
$$\widetilde{S}(W) = \widetilde{S}(W, a, b) = \{\langle W - r \rangle_W : r \in S(W)\} \tag{10}$$

Finally, we propose further reducing the number of variables by considering h_{kj} and v_{kj} variables separately. Not all the positions in sets $\widetilde{S}(L)$ and $\widetilde{S}(W)$ are non-dominated for both types of variables. For example, if we consider the instance $(11, 10, 4, 3)$ we have:

$$\widetilde{S}(W, a, b) = \widetilde{S}(10, 4, 3) = \{0, 3, 4, 6, 7\}$$

$$\widetilde{S}(L,a,b) = \widetilde{S}(11,4,3) = \{0,3,4,7,8\}$$

According to $\widetilde{S}(L,a,b)$ y $\widetilde{S}(W,a,b)$, $(0,6)$ is a non-dominated position for a variable. This is true for a V-variable, $v_{0,6}$, but not for the H-variable $h_{0,6}$ which would be dominated by $h_{0,7}$, as can be seen in Figure 1: Given a solution in which $h_{0,6} = 1$, we can obtain a new solution with an equal or greater number of boxes changing $h_{0,6} = 1$ for $h_{0,7} = 1$. Therefore, the raster points are different for H-variables and V-variables.



(a) Variable $h_{0,6}$ (b) Variable $h_{0,7}$

Figure 1. *Reduction of variables. Instance (11,10,4,3)*

The number of constraints can also be reduced. When considering the constraint corresponding to a unit square (r,s), if all the variables involved in the constraint have appeared in a *previous* constraint (r',s') (to the left and/or to the bottom of (r,s)), that is, if going up and/or right from (r',s') to (r,s) no new variable has appeared, then the constraint at (r,s) is redundant. In this case, the constraint avoiding the overlapping of square (r',s') is also responsible for avoiding the overlapping of square (r,s).

The effect of the reductions on this small instance appears at Table 1.

	H-variables	V-variables	Total variables	Constraints
Initial	49	48	97	56
Normal sets	25	24	49	30
Raster points	20	20	40	25
New reduction	16	9	25	23

Table 1
Reductions for instance (11,10,4,3)

Comparing only the effect of the raster points of Scheithauer and Terno[31]

and the new reduction on the complete sets Cover I and Cover II, we obtain the average results in Table 2.

	Raster points		New reduction	
	Variables	Constraints	Variables	Constraints
Cover I	373,46	212,34	266,37	211,89
Cover II	1406,35	753,03	1041,25	752,49

Table 2
Comparing raster points and the new reduction

3.3 An upper bound constraint

As was mentioned in the Introduction, there are good heuristic algorithms and good upper bounds for the PLP. As the objective function is the number of boxes packed onto the pallet, the optimal solution value has a narrow range of possible values. In fact, given n_{heur}, the value of a feasible solution, and n_{upper}, the value of an upper bound, we are looking for a solution with n_{opt} boxes, where $n_{heur} < n_{opt} \leq n_{upper}$. Taking advantage of this fact, in our search tree we define a first level of branching in which each node has a fixed number of boxes, decreasing from n_{upper} to $n_{heur} + 1$. We start searching the branch with the maximal number of boxes and explore it until a feasible, and hence optimal, solution is found or the branch has been completely studied. In this case, we proceed to the next branch with one box less and so on.

Therefore, when we solve the linear relaxation of the integer formulation at any node of the tree, we add the corresponding upper bound constraint:

$$\max \sum_{k=0}^{L-a} \sum_{j=0}^{W-b} h_{kj} + \sum_{k=0}^{L-b} \sum_{j=0}^{W-a} v_{kj} \leq B \tag{11}$$

where B is the number of boxes corresponding to the current first level branch.

As a consequence the *waste*, that is the non-used surface of the pallet corresponding to each branch, is also known. In a branch with n boxes, the waste is $U = L * W - n * a * b$. That can be used to improve the formulation. When we described the reduction of constraints, we mentioned that a constraint can be in charge of several unit squares. Hence, if all its variables are zero, not only one, but several squares will be empty. If the number of squares associated to a constraint is greater than the waste U, this constraint must be equal to one. This change in the status of the constraint is not necessary in the integer formulation, but is relevant in the linear relaxation.

4 Reducing the PLP to a graph problem

The relationship between the PLP and the maximum independent set on a conflict graph was introduced by Dowsland in 1987[14]. She used it to develop an exact algorithm, adapting an existing tree search procedure. Here we will take advantage of this relation and the extensive knowledge of the polyhedron of the maximum independent set in order to develop efficient separation procedures adapted to the PLP.

At each node, when the solution of linear relaxation is fractional, we define a graph $G_{PLP} = (V, E)$. Initially, each vertex $v \in V$ corresponds to a fractional variable $0 < h_{kj} < 1$ (or $0 < v_{kj} < 1$) in the linear solution. In the implementation of the algorithm, only variables with values above a threshold of 0.005 define a vertex. In that way, we reduce the size of the graph to be explored by eliminating variables which add very little when looking for violated inequalities.

We include an edge between each pair of vertices if the corresponding boxes are in conflict, that is, if they cannot be together in a solution with the given number of boxes. A first type of edges involves pairs of boxes that would overlap. But beyond these evident conflicts, there are some others which can be used to define new edges and to identify more violated inequalities in the graph.

- *Conflicts related to waste*

  Two boxes are in conflict if the waste that they would produce if they appeared together in a solution is greater than the waste U associated to the solution.

  There are many situations in which these conflicts appear:

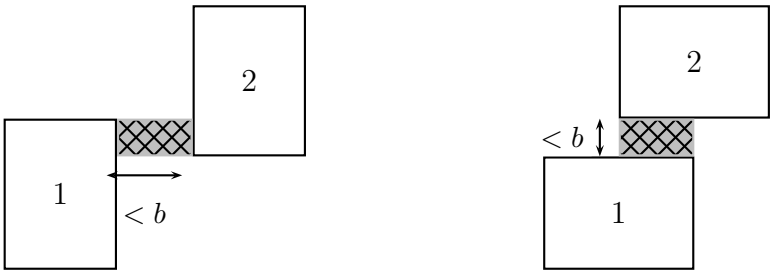
  · Direct waste between boxes. In Figure 2, the shadowed region shows the unavoidable waste associated to the presence of boxes 1 and 2.

  · Indirect waste: Between the boxes being considered, some other boxes can be placed, but even with the fullest usage of the space there would still be some waste. In Figure 3, between boxes 1 and 2 there may be some vertical or horizontal boxes, but the shadowed region is the minimum waste involved.

  · Waste in a corner of the pallet. In Figure 4, the space in the corner may be filled in two different ways, each producing a waste. The lesser of the two is the unavoidable waste involved.

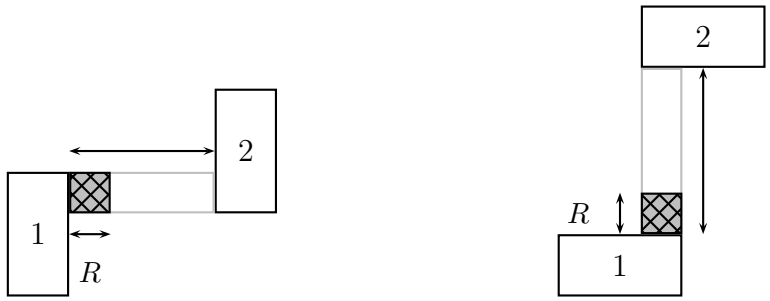
- *Conflicts related to dominance*

  We say that a pair of boxes is *dominated* by another pair if this second pair uses a strictly smaller region of the pallet, leaving the rest of the layout unchanged. A dominated pair of boxes can be eliminated from the solution



(a) Horizontal (b) Vertical

Figure 2. *Direct waste*



(a) Horizontal (b) Vertical

Figure 3. *Indirect waste*

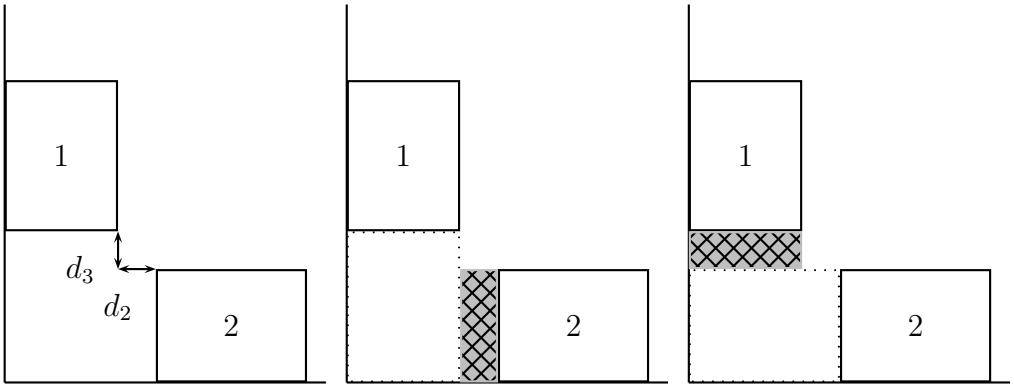


Figure 4. *Waste in the corners*

by adding an edge between them.

There are many situations in which these dominance conflicts appear. In Figure 5, a simple dominance situation arises when one pair is dominated by another in which a box of the original pair has been moved. Figure 6 shows two cases of dominance in which the dominant pair has a different

layout. Figure 7 shows a dominance situation in a corner of the pallet.

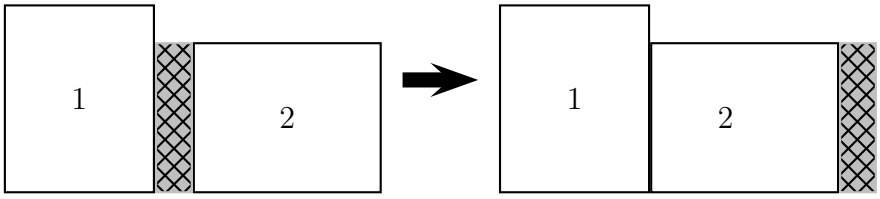
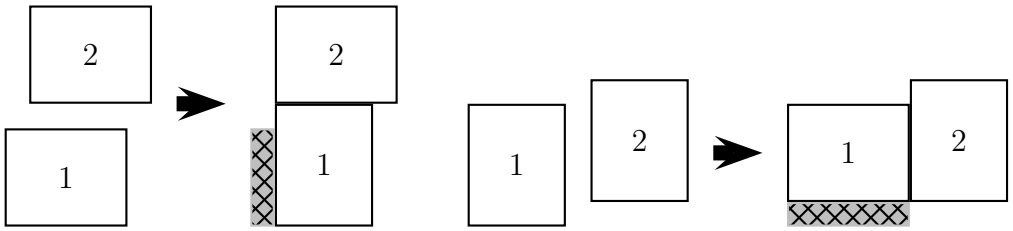


Figure 5. *Simple dominance*



(a) Horizontal (b) Vertical

Figure 6. *Dominance with a change in the layout*

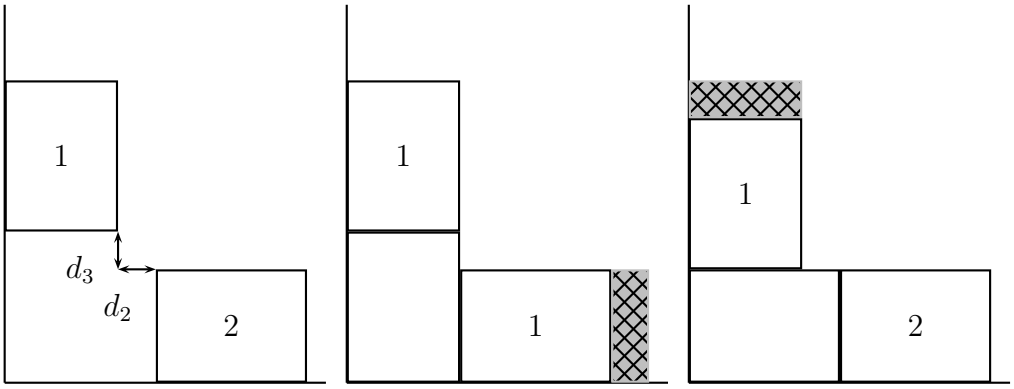


Figure 7. *Dominance in the corner*

- *Symmetry considerations*

  Very often we have feasible solutions that are symmetric to other solutions, with respect to a vertical axis passing through the center of the pallet, to a horizontal axis, or to both of them. This symmetry multiplies the number of solutions to explore. We can simplify the search by developing some procedures that eliminate symmetric solutions, considering only solutions in which the waste tends to be towards the top and the right of the pallet.

  At each node of the tree, we know from the first level of branching the number of boxes in the solution and the associated waste U. If we consider the pallet divided vertically into two halves, we can restrict our study to

solutions with $U_{left} \leq U/2$. In a similar way, considering the pallet divided horizontally, we can only study solutions for which $U_{bottom} \leq U/2$.

If we divide the pallet into four quarters, we can bound the maximum admissible waste for each of them (See Appendix and Figure 8) :

$U_B \leq \frac{U}{2}$	$U_D \leq U$
$U_A \leq \frac{U}{4}$	$U_C \leq \lceil \frac{3}{4}U - 1 \rceil + \frac{1}{4}$

Figure 8. *Maximum waste at each quarter.*

These bounds can be applied when adding edges related to waste. For instance, if we are considering a direct waste between boxes in the first quarter A, it suffices that this waste is greater than $U/4$ to include an edge between them.

The edges related to the waste which can be included depend on the quarter in which the waste is located and depend also on the instance we are considering as representative of the equivalence class. For instance, let us consider the instance (27,17,5,3) and suppose that we are looking for a solution with 30 boxes. The associated waste will be $U = 9$ and the waste relative to the box size:

$$\frac{U}{(a * b)} = \frac{9}{5 * 3} = 0,6$$

If we take another equivalent instance (40007,24005,8001,4001), the solution with the same number of boxes will have a waste $U = 8005$ and the waste relative to the box size will in this case be:

$$\frac{U}{(a * b)} = \frac{8005}{8001 * 40001} = 0,00025$$

In (27,17,5,3) in position:

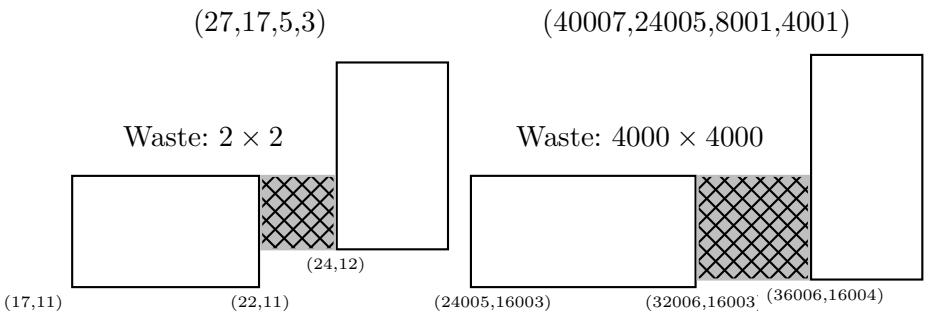
$(1 * a + 4 * b, 1 * a + 2 * b) = (17, 11)$ we can put an H-box
$(3 * a + 3 * b, 0 * a + 4 * b) = (24, 12)$ we can put a V-box

Similarly, in (40007,24005,8001,4001) in the corresponding position

$(1 * a + 4 * b, 1 * a + 2 * b) = (24005, 16003)$ an H-box can be placed
$(3 * l + 3 * w, 0 * l + 4 * w) = (36006, 16004)$ a V-box can be placed

In the first case, the pair of boxes would produce a direct waste of 4 units, which would not be greater than the maximum allowable waste for the

upper right quarter (9), and no edge between them could be added. (Figure 9(a)). But in the second case, the direct waste between the boxes would be 4000*4000=16000000 greater than the maximum allowable waste (8005) and an edge between them could be added (Figure 9(b)). Obviously, if the two boxes cannot appear together in a solution of 30 boxes for the second instance, they cannot appear together in a solution for the first instance, so the edge could also be used for it. In this example, if we build the graph corresponding to the instance (27,17,5,3) it has 279 edges, while the graph corresponding to the instance (40007,24005,8001,4001) has 341 edges.



(a) Small relative waste (b) Large relative waste

Figure 9. *Different behavior of equivalent instances*

Instance (40007,24005,8001,4001) exhibits such good properties because it is the element of the class with the smallest upper bound. Its relative waste is very low (0.0025) and it is possible to detect many pairs of boxes which are *badly placed* with respect to an optimal solution and to add many edges to eliminate them. The differences between relative wastes among instances of an equivalence class can be very large. When building the graph, we use the instance with the smallest upper bound, obtained by following the procedure proposed by Nelissen[26].

5 Valid inequalities and separation procedures

Given the graph $G_{PLP} = (V, E)$, we review the valid inequalities known for the maximum independent set problem which will be used in the separation procedures. The maximum independent set polyhedron is denoted by P_{IS}.

5.1 Cliques

A *clique* C in a graph G is a maximal complete subgraph of G. If $C \subseteq V$ is a clique, any independent set will have at most one node of C. Hence, the *clique inequality*:

$$\sum_{v \in C} x_v \leq 1 \tag{12}$$

is valid for P_{IS}. Padberg [29] showed that inequality (12) defines a facet of P_{IS} iff C is a clique of G.

In the PLP it can be shown that if in the graph G_{PLP} we consider only the edges avoiding overlapping between boxes, the only cliques are the covering constraints (4). However, if we add the edges related to waste and dominance, there are other cliques which can be violated by linear solutions. In order to identify them, we follow the strategy proposed by Hoffmann and Padberg[21], combining an exhaustive and a greedy procedure, depending on the degree of the vertex being explored.

Step 1. Take $v \in V$ with smallest degree ($d(v)$).
Step 2. If $d(v) = 0$, *Stop*.
Step 3.
- If $v \cup \{star(v)\}$ is a clique:
  If the sum of its variables is greater than 1, add the constraint to the linear formulation. Delete v.
- Otherwise, go to *Step 4*.

Step 4.
- If $d(v) \geq 25$. Use a *greedy* constructive procedure to find the maximum weight independent set.
  If the sum of its variables is greater than 1, add the constraint to the linear formulation. Delete v. Go to *Step 1*.
- If $d(v) < 25$, use the exhaustive procedure by Bron and Kerbosch[7] to find all the maximum cardinality independent sets. Each of them whose sum of variables is greater than 1 defines a new constraint. Delete v. Go to *Step 1*.

5.2 Odd cycles

If H is an odd cycle in G, the *odd cycle inequality* is

$$x(V(H)) \leq \lfloor |V(H)|/2 \rfloor. \tag{13}$$

Padberg[29] showed that these inequalities are valid for P_{IS}, but they do not usually define facets. In order to obtain facets, he proposes a sequential lift-

ing procedure, starting from an odd cycle without chords, a *hole*, satisfying inequality (13) and adding variables x_v, one at a time, with coefficients α_v, obtained by solving a series of, usually small, 0-1 problems, finally producing the inequality

$$x(V(H)) + \sum_{v \in V \setminus V(H)} \alpha_v x_v \leq \lfloor |V(H)|/2 \rfloor. \tag{14}$$

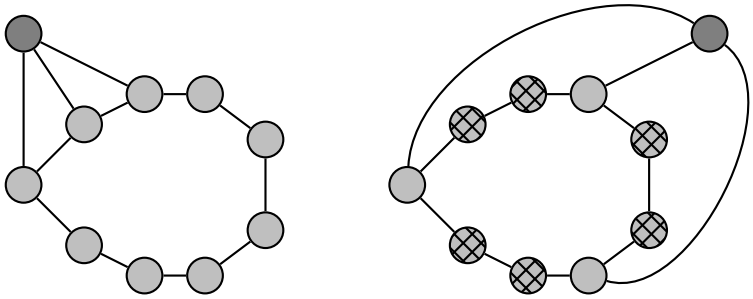
We will follow a sequential lifting procedure adapted to our problem. First, the solution graph may be non-connected. We start by identifying the connected components and then we apply the procedure to each component. Second, we cannot use a strategy like that used by Nemhauser and Sigismondi[28], who first identify violated odd hole constraints and then lift them, because in our problem the sum of the variables of the cycle produces a violated constraint on few, if any, occasions. Only after lifting do we obtain the violated constraints we are looking for. Therefore, our procedure has to find some odd cycles, lift them, and see if the sum of the variables involved in the lifted inequality is greater than the corresponding right hand side.

We have tested two different types of algorithm to find odd cycles:

- *Minimum spanning tree algorithm.*
  We build the minimum spanning tree G_{LWlw} and its co-tree. Then the cycles are generated by including the edges of the co-tree in the spanning tree, and we keep those of odd cardinality.
- *An algorithm for searching holes* (Hoffman and Padberg (1993)[21]).
  The procedure chooses a vertex v as the root node and builds a tree level by level. Each level is defined by the distance to the root, where distance is the number of arcs between vertices. Therefore, all the vertices adjacent to v are in level 1, the vertices adjacent to those of level 1, excluding v and those in level 1, are in level 2 and so on. Any pair of nodes of level k with disjoint paths to v define an odd hole containing v.

Our sequential lifting procedure follows a tree structure similar to that of Nemhauser and Sigismondi[28], but unlike theirs we do not solve exactly the maximum independent set problems corresponding to every coefficient because we are looking for a very fast procedure. We start by identifying the set of variables which can be lifted. These are (a) those adjacent to three or more consecutive vertices of the cycle (Figure 12.a) and (b) those that produce *even chains*, where by chain we mean consecutive vertices of the cycle (Figure 12.b). However, the second type of variables are much more complicated to use in a sequential procedure and we only use the first type of candidates.

Once the set of candidates has been defined, we use the following sequential process, in which we denote by $N_C(v)$ the set of vertices of C adjacent to v:



(a) Three or more consecutive adjacent vertices

(b) Even chains

Figure 10. *Candidates for lifting*

Step 1. *Initialization.*
$p = 0$.
$E = \emptyset$. Set of variables lifted in the node
L=List of candidates for lifting
Augment=0.

Step 2. *Selecting the variable to lift.*
If $L = \emptyset$, go to *Step* 4 (Backtrack).
Otherwise, take the first variable $x_k \in L$, set $L = L - \{x_k\}$

Step 3. *Test of variable* x_k.

$$\text{If, for all } x_v \in E \begin{cases} x_v \text{ and } x_k \text{ are adjacent} \\ \text{or} \\ |N_C(v) \setminus N_C(v')| \geq 2 \end{cases}$$

Go to *Step* 4 (Branching), *Augment*=1.
Otherwise, go to *Step* 1.

Step 4. *Branching* (Variable x_k is added to the constraint)
$p = p + 1$
$E = E \cup \{x_k\}$
Go to *Step* 2.

Step 5. *Backtrack.*
If *Augment*=1, save the constraint (set E).
Set *Augment* $= 0$.
$p = p - 1$
If $p < 0 \rightarrow$ Stop (Enumeration completed).
Otherwise, $E = E - \{x_r\}$, where x_r is the last variable of E
$L = \{x_{r+1}, \ldots, x_n\}$
Go to *Step* 1.

Figure 13 shows the two possibilities for lifting. If variables x_1 and x_2 have already been included in the constraint, and variable x_3 is being tested for inclusion, then it will also be added because $|N_C(x_3) \setminus N_C(x_1)| \geq 2$ and x_3 is adjacent to x_2.

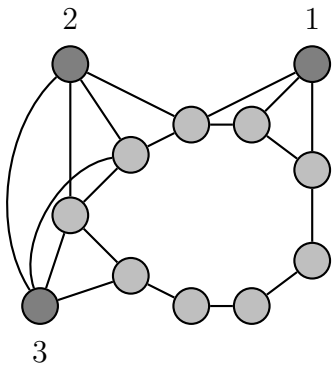


Figure 11. *Sequential lifting of vertices*

5.3 Wheels and other inequalities

In 1997, Cheng and Cunningham[9] introduced *wheels*, new classes of valid inequalities. They discuss the conditions in which wheels define facets and provide some separation procedures. However, as we are looking for fast and simple identification schemes, we have not included these procedures in our algorithm. Some other facet-defining inequalities have been described for the maximum independent sets: *antiholes*, *webs*, *antiwebs*[2], *fans*, *grilles*[8], but no separation procedures have been developed for them.

6 Rounding LP solutions

In the branch and cut procedure we solve many linear problems and obtain fractional solutions. Every time we have a fractional solution we try to get an integer solution from it by using some kind of rounding procedure. Sometimes this process improves the best feasible solution known so far. We have studied several rounding strategies:

- Simple deterministic and non-deterministic procedures

  A simple procedure consists of ordering the fractional variables in a non-increasing order of values. We take the first variable on the list and round it up to 1, taking all its adjacent variables from the list, and repeat the step while there are still variables left. Usually there will be many ties, and it is reasonable to break ties at random and repeat the procedures a given number of iterations.

Bertsimas and Vohra[4] propose a non-deterministic method. Each fractional variable has a probability to be rounded up to 1:

$$P\{x_j = 1\} = f(x_j^*) = 1 - (1 - x_j^*)^k, k = log|V| \tag{15}$$

- Minimum regret rounding

  Strijk et al.[34] study the Map Labelling Problem and reduce it to a Maximum Independent Set Problem. They propose a new rounding strategy based on a regret function. Denote by $\mathcal{I}$ the collection of all independent sets in G_{PLP}. Suppose we have a solution $x \in P_{IS}$ with some fractional components. This solution is rounded by repeatedly choosing an independent set I$\in \mathcal{I}$ with $0 < x_I < 1$, rounding x_I up to 1 and $x_{N(I)}$ down to 0, until x is integral. $N(I)$ denotes the set of neighbors of I. This rounding operation defines a function f mapping x on a vector $x' \in P_E$ using I:

$$f : P_E \times I \rightarrow P_E : (x, I) \rightarrow x', \qquad \text{where } x'_u = \begin{cases} 1, & \text{if } u \in I; \\ 0, & \text{if } u \in N(I); \\ x_u, & \text{otherwise.} \end{cases} \tag{16}$$

  Given $x \in P_{IS}$ and I$\in \mathcal{I}$, then $f(x, I) \in P_{IS}$. The $regretfunction$ is the difference in the objective function when we go from x to x' and we look for the minimum decrease, *minimum regret*, when choosing the set I for rounding.

The best results have been obtained by using the *minimum regret* rounding algorithm and it has been used in the current implementation of our procedures.

7 Branching strategies

We have studied several strategies:

- Branching on variables

  The fractional variable with its value nearest to 0.5 is chosen as the branching variable and in the next level two new nodes are created. On the left branch, the variable is fixed to 1, on the right branch the variable is fixed to 0.
- Branching on constraints

  The covering constraint whose slack is nearest to 0.5 is chosen as the branching constraint. On the left branch, the constraint is set equal to 1, on the right branch it is set equal to 0. It may happen that a solution is fractional but all the covering constraints have a slack of 1 or 0. In that case, we switch to branching on variables.

- Other branching strategies

  We have considered the branching scheme proposed by Rossi and Smriglio[30] in their work on the Maximum Independent Set Problem. At any node t of the search tree the branching constraints fix a set $U^t \subseteq V (L^t \subseteq V)$ of variables to 1 (to 0). Hence, we are left with a problem with variables $V^t = V \setminus (U^t \cup L^t)$. Let $\bar{\alpha}$ be the best known solution and let $\bar{\alpha}^t = \bar{\alpha} - |U^t|$. If we can certify that the problem in the node has a solution which is lower than or equal to $\bar{\alpha}^t$, then the node can be fathomed. The branching rule then goes as follows: Let us consider a set $W^t \subseteq V^t$ for which it is possible to prove that the problem on that set has a solution which is lower than or equal to $\bar{\alpha}^t$, and let $Z^t = V^t \setminus W^t$. If the problem in the node has a solution greater than $\bar{\alpha}^t$, every solution must contain at least one variable in Z^t. Each $v_i \in Z^t$ defines a branch. Rossi and Smriglio propose a procedure to build W^t and reference other proposals for obtaining Z^t.

  Our preliminary computational experience showed that the application of Rossi and Smriglio's [30] strategy to our problem resulted on sets Z^t which were too large and enumeration trees which were too wide, with respect to those obtained by branching on variables or by branching on constraints. Therefore, in the implementation of our algorithms we do not consider that alternative.

  For $CoverI$, the search trees are very small and therefore branching on variables and branching on constraints produce similar results. For the instances of $CoverII$, the strategy of branching on constraints produces smaller trees and is used in the final implementation.

8 Variable and constraint setting by logical implications

Depending on the branching strategy, some variables or constraints can be set for the remaining subtree being explored.

- *Branching on variables*

  If a variable x_j is set to 1, its corresponding box is put into the solution, so all the variables corresponding to boxes which cannot appear with it in a solution with the required number of boxes are set to 0. That involves boxes which would overlap with it, but also those involved in the considerations of waste and dominance and in the symmetry considerations described in Section 4.
- *Branching on constraints*

  Suppose we are in a branch with a known waste U. If a covering constraint is set to 0, it would induce a waste of u. Therefore, the remaining covering constraints which, if set to 0, would produce a waste greater than $U - u$, can be set to 1.

9 Computational results

9.1 Implementation details

The branch and cut algorithm has been implemented using *ABACUS (A Branch And CUt System)* developed by Thienel[35] for the implementation of *Branch and Bound* algorithms which use linear relaxations of integer formulations and which can be complemented with the dynamic generation of cutting planes (*Branch and Cut*), column generation (*Branch and Price*), or both (*Branch and Cut and Price*). For a revision of the work done with that software, refer to Elf et al.[17]. *ABACUS* provides a system of abstract $C++$ classes from which the necessary classes for a specific problem can be derived.

The algorithms have been coded in $C++$ and run on a $PCPentiumIII$ at $850Mhz$.

9.2 Results on Cover I

Set $CoverI$ contains 7827 equivalence classes corresponding to instances with solutions up to 50 boxes. These problems have already been solved by other procedures and we have used them as a test bench to study different strategies to be applied later to larger problems.

Many of the problems in $CoverI$ are very easy, because the solution of a simple constructive heuristic, the five-block heuristic, matches the value of a simple lower bound based on the restricted pallet area. In fact, that is the case for 6736 classes and they are not considered further.

We apply our algorithm to the remaining 1091 problems. In 592 cases the solution of the first linear relaxation provides the optimal solution, either because the solution value is equal to the heuristic solution or because the linear solution is integer. Therefore, we are left with 499 problems in which some cutting planes and/or branching phases are required. In the remainder of this subsection we will refer only to that subset.

The Tables will show the results of the different strategies according to the number of nodes ($\#nodes$) and CPU time ($Time$). Table 3 shows the performance of different separation procedures, called only at root node:

- *Cliques*: looking only for clique inequalities
- *Cycles*: looking only for odd cycles obtained from a minimum weight spanning tree and lifting them

- *Holes*: looking only for holes and lifting them
- *Fast*: looking for cliques and for odd cycles from the minimum spanning tree, but lifting only *promising* cycles, that is, cycles for which the sum of variables before lifting is at least 0.95 of the corresponding right hand side of the constraint.

		Cliques	Cycles	Holes	Fast
Time	Maximum	31,04	835,03	15188,22	104,9
	Average	3,14	28,34	179,91	8,45
	St. deviation	4,2	71,74	905,61	12,85
#Nodes	Maximum	33	17	165	35
	Average	3,77	1,93	2,17	2,45
	St. deviation	3,56	2,27	7,82	3,04
Solved at the Root Node		209	399	422	349

Table 3
Cover I. *Separation procedures*

In Table 3 we can observe that the procedure based on identifying and lifting holes solves most of the problems without resorting to branching, but it is too time consuming. The strategy of looking only for *Cliques* is very fast on these small problems, but for larger problems we think that we will need more powerful separation procedures. The strategy we have called *Fast* seems to achieve a reasonable balance between the separation and branching phases of the algorithm. Using this strategy, in Table 4 we investigate the effect of different types of edges in the solution graph:

- *Overlapping*: Edges avoiding overlapping between boxes
- *Waste*: Edges based on allowable waste in the solution
- *Dominance*: Edges based on dominance considerations
- *All*: All types of edges

We can see in Table 4 that the computational effort increases as we add new types of edges. If we were looking for an efficient algorithm for $CoverI$ problems we would not include these new edges. However, the second half of the Table, and particularly its last line, shows that many more problems are solved either without branching or with search trees which are smaller on average, and this will be very useful in larger problems.

Table 5 shows the results of three versions of the complete algorithm, using all types of edges and the *Fast* separation procedure.

- *Branch&Cut*: Separation procedures are invoked at each node of the tree

		Overlapping	Waste	Dominance	All types
Time	Maximum	16,09	71,4	98,7	104,9
	Average	1,38	4,48	5,48	8,45
	St. deviation	1,63	6,37	8,92	12,85
#Nodes	Maximum	25	27	25	35
	Average	4,18	3,38	3,41	2,45
	St.deviation	3,7	3,6	3,6	3,04
Solved at the Root Node		185	257	271	349

Table 4
Cover I. *Different types of edges*

while some violated inequalities are found.

- *B&C2Iter*: Separation procedures are invoked at each node of the tree, but at most twice at each node.
- *CuttingPlanes*: Separation procedures are used only at the root node, while some violated inequalities are found.

		Branch&Cut	B&C2iter	CuttingPlanes
Time	Maximum	344,83	137,1	104,9
	Average	12,07	4,73	8,45
	St.deviation	27,16	8,3	12,85
#Nodes	Maximum	27	29	35
	Average	1,95	2,51	2,45
	St.deviation	2,09	2,19	3,04
Solved at the Root Node		349	259	349

Table 5
Cover I. Comparing algorithms

As Table 5 shows, it seems more efficient to limit the number of times the separation procedures are called at each node, or to limit their use to the root node. Both algorithms, *B&C2iter* and *CuttingPlanes*, are very efficient and will be used for solving problems in $CoverII$.

9.3 Results on Cover II

This set of 40609 equivalence classes has not been completely explored before, either by heuristic or by exact techniques. As was done with $CoverI$, we first

separate those easy problems for which the results of the five-block heuristic is proven to be optimal by a simple upper bound. That leaves us with 10764 problems. In a second step, we solve these problems with an elaborated heuristic, G4 [31], and Nelissen's upper bound [27] and only for 2433 problems is the heuristic solution not proved to be optimal.

We then apply our algorithm to those 2433 non-easy problems. 2192 of them are solved by the first linear solution, either because the solution value is equal to the heuristic solution, or because the linear solution is integer. There are then 241 hard problems left in which some cutting planes and/or branching phases are necessary.

Table 6 contains the CPU times required by *B&C2iter*, *CuttingPlanes* and *CPLEX* 7.0, a state-of-the-art commercial code for linear and integer programming, on these 241 hard problems. Figure 12 allows a direct comparison between *B&C2iter* and $CPLEX7.0$. Both algorithms share the same formulation, described in Section 3, and the differences correspond to the effect of the specific separation procedures we have developed for the problem with respect to the powerful, but general purpose, problem-solving strategies included in *CPLEX*. The last column of the Table shows that there are still some very hard problems exceeding the time limit of 54000 seconds. It has been necessary to impose a limit because preliminary computational tests showed that in some isolated cases the time required could be extremely long. For example, *CPLEX* took 1423508 seconds for solving instance (89,75,10,7). Taking averages on the computing times and number of nodes is not possible because not all the problems have been solved. Besides that, the averages would be severely distorted by the existence of a few extremely high values. A measure which can be calculated and gives a robust indication of the central values of computing times and number of nodes is the median, which appears in Table 7.

	B&C2iter	*CuttingPlanes*	*CPLEX*
$T < 10$	137	127	85
$10' < T < 1h$	57	77	84
$1h < T < 2h$	18	10	24
$2h < T < 5h$	15	15	16
$5h < T < 10h$	8	5	12
$10h < T < 15h$	1	0	1
$T > 15h$	3	7	19

Table 6
Cover II. Comparing algorithms

If the limit of 54000 CPU seconds (15 hours) seems too high for practical

purposes, Table 8 shows the number of unsolved problems for several upper limits. Note that, in general, *unsolved* would mean that the optimality of the best current solution has not been proven, because the search has not been able to completely discard the existence of a solution with one more box than that provided by the best current solution.

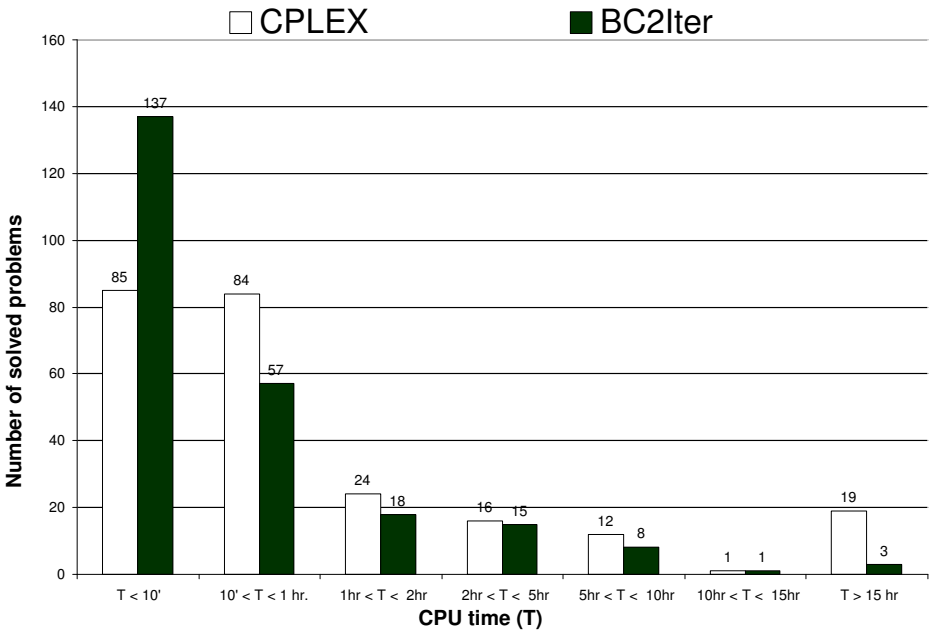


Figure 12. *Comparing CPU times on Cover II*

	Median	
	CPU Time	Number of nodes
B&C2iter	436,44	15
CuttingPlanes	506,91	21
CPLEX	1268,62	82

Table 7
Cover II. Medians for the algorithms

	Unsolved problems at the limit				
	1 hour	2 hours	5 hours	10 hours	15 hours
B&C2iter	48	30	15	7	3
CuttingPlanes	37	27	12	7	7
CPLEX	72	48	32	20	19

Table 8
Cover II. Unsolved problems

10 Conclusions

We have developed a branch and cut algorithm for solving optimally PLP instances of up to 100 boxes. Problems of this size had not been exhaustively studied to date by exact methods. The proposed versions of the algorithm are shown to be able to solve almost all the 40609 equivalence classes contained in $CoverII$, though a small subset of very hard problems would need extremely long solution times. The algorithm combines previous knowledge of the problem with some new features. Beasley's 0-1 formulation is adapted to the PLP, adding new constraints and new reduction procedures for variables and constraints. The separation procedures are based on known valid inequalities for the maximum independent set on the solution graph, but the definition of this graph includes new types of edges, characteristic of the PLP, and the procedures applied to odd cycle identification are tailored to this problem. This combination of known and new elements proves to work efficiently for the target set of the study. In the near future, we will try to apply some of these ideas to more general packing problems.

11 Appendix: Maximum waste at each quarter

1) The waste in quarter A, $U_A \leq \frac{\mathcal{U}}{4}$.

Proof:

Let us suppose that there is a solution with $U_A > \frac{\mathcal{U}}{4}$

Then $\mathcal{U} - U_A < \frac{3\mathcal{U}}{4} \Rightarrow \frac{\mathcal{U} - U_A}{3} < \frac{\mathcal{U}}{4} \Rightarrow \exists$ quarter with waste $< \frac{\mathcal{U}}{4}$

By symmetry A will be this quarter.

1.1) If L even and W even: $U_A \leq \left\lfloor \frac{\mathcal{U}}{4} \right\rfloor$

Proof:

Let us suppose that $U_A > \left\lfloor \frac{\mathcal{U}}{4} \right\rfloor$

As L even and W even, U_A is integer and then $U_A \geq \left\lfloor \frac{\mathcal{U}}{4} \right\rfloor + 1$

Hence, $\mathcal{U} - U_A \leq \mathcal{U} - \left\lfloor \frac{\mathcal{U}}{4} \right\rfloor - 1 < \mathcal{U} - \frac{\mathcal{U}}{4} = \frac{3\mathcal{U}}{4}$

As $\mathcal{U} - U_A$ is integer, $\mathcal{U} - U_A \leq \left\lfloor \frac{3\mathcal{U}}{4} \right\rfloor \rightarrow \frac{\mathcal{U} - U_A}{3} \leq \left\lfloor \frac{3\mathcal{U}}{4} \right\rfloor \Big/ 3 \Rightarrow$

we can write: $\mathcal{U} = 4\text{a} + \text{b}$, $\text{b} < 4$ where $\text{a} = \left\lfloor \frac{\mathcal{U}}{4} \right\rfloor$

Therefore, $3\mathcal{U} = 12a + 3b$

Dividing by 4 and rounding down to the nearest integers: $\left\lfloor \frac{3\mathcal{U}}{4} \right\rfloor = 3a + \left\lfloor \frac{3b}{4} \right\rfloor$

If we divide by 3:

$$\left\lfloor \frac{3\mathcal{U}}{4} \right\rfloor \Big/ 3 = a + \left\lfloor \frac{3b}{4} \right\rfloor \Big/ 3 = \begin{cases} a & \text{if } b = 0 \\ a & \text{if } b = 1 \\ a + \frac{1}{3} & \text{if } b = 2 \\ a + \frac{2}{3} & \text{if } b = 3 \end{cases}$$

Hence $\left\lfloor \left\lfloor \frac{3\mathcal{U}}{4} \right\rfloor \Big/ 3 \right\rfloor = \left\lfloor \frac{\mathcal{U}}{4} \right\rfloor$ and $\frac{\mathcal{U} - U_A}{3} \leq \left\lfloor \frac{3\mathcal{U}}{4} \right\rfloor \Big/ 3$

There is at least 1 quarter with $\left\lfloor \frac{\mathcal{U} - U_A}{3} \right\rfloor \leq \left\lfloor \left\lfloor \frac{3\mathcal{U}}{4} \right\rfloor \Big/ 3 \right\rfloor = \left\lfloor \frac{\mathcal{U}}{4} \right\rfloor$

By symmetry, this quarter will be A.

1.2) If L even or W even: $U_A = \left\lfloor \frac{\mathcal{U}}{4} \right\rfloor_{0.5}$

(where $\lfloor \rfloor_{0.5}$ means rounding down to the nearest multiple of de 0.5)

Proof:

If $U_A > \left\lfloor \frac{\mathcal{U}}{4} \right\rfloor_{0.5} \rightarrow U_A \geq \left\lfloor \frac{\mathcal{U}}{4} \right\rfloor_{0.5} + \frac{1}{2}$

$\mathcal{U} - U_A \leq \mathcal{U} - \left\lfloor \frac{\mathcal{U}}{4} \right\rfloor_{0.5} - \frac{1}{2} < \mathcal{U} - \frac{\mathcal{U}}{4} = \frac{3\mathcal{U}}{4}$

As $\mathcal{U} \in Z$ and $U_A = \lfloor U_A \rfloor_{0.5}$ because L or W even $\rightarrow \mathcal{U} - U_A \leq \left\lfloor \frac{3\mathcal{U}}{4} \right\rfloor_{0.5}$

$\frac{\mathcal{U} - U_A}{3} \leq \left\lfloor \frac{3\mathcal{U}}{4} \right\rfloor_{0.5} \Big/ 3$

At least one quarter with $\left\lfloor \frac{\mathcal{U} - U_A}{3} \right\rfloor_{0.5} \leq \left\lfloor \left\lfloor \frac{3\mathcal{U}}{4} \right\rfloor_{0.5} \Big/ 3 \right\rfloor_{0.5} = \left\lfloor \frac{\mathcal{U}}{4} \right\rfloor_{0.5}$

2) The waste in the second quarter B, $U_B \leq \frac{\mathcal{U}}{2}$.

This waste corresponds to the left side of the pallet.

3) The waste in the third quarter C, $U_C \leq \left\lceil \frac{3}{4}\mathcal{U} - 1 \right\rceil + \frac{1}{4}$

Proof:

If $U_C > \left\lceil \frac{3}{4}\mathcal{U} - 1 \right\rceil + \frac{1}{4} \rightarrow U_C \geq \left\lceil \frac{3}{4}\mathcal{U} - 1 \right\rceil + \frac{1}{2}$

Therefore, $\mathcal{U} - U_C \leq \mathcal{U} - \left\lceil \frac{3}{4}\mathcal{U} - 1 \right\rceil - \frac{1}{2} = \Delta$

Let us see the value of Δ:

Let $\mathcal{U} = 4a + \text{b}$ (b < 4), where $a = \left\lfloor \frac{\mathcal{U}}{4} \right\rfloor$

$3\mathcal{U} = 12a + 3b$

$\frac{3\mathcal{U}}{4} = 3a + \frac{3}{4}b \rightarrow \frac{3\mathcal{U}}{4} - 1 = 3a - 1 + \frac{3}{4}b \rightarrow \left\lceil \frac{3\mathcal{U}}{4} - 1 \right\rceil = 3a - 1 + \left\lceil \frac{3}{4}b \right\rceil$

$\left\lceil \frac{3\mathcal{U}}{4} - 1 \right\rceil + \frac{1}{2} = 3a - \frac{1}{2} + \left\lceil \frac{3}{4}b \right\rceil$

$\Delta = \mathcal{U} - \left(\left\lceil \frac{3\mathcal{U}}{4} - 1 \right\rceil + \frac{1}{2} \right) = (4a + b) - \left(3a - \frac{1}{2} + \left\lceil \frac{3}{4}b \right\rceil \right) =$

$$= a + \frac{1}{2} + \left(b - \left\lceil \frac{3}{4}b \right\rceil \right) \begin{cases} b - \left\lceil \frac{3}{4}b \right\rceil = 0 \rightarrow \text{if } b = 0 \\ b - \left\lceil \frac{3}{4}b \right\rceil = 0 \rightarrow \text{if } b = 1 \\ b - \left\lceil \frac{3}{4}b \right\rceil = 0 \rightarrow \text{if } b = 2 \\ b - \left\lceil \frac{3}{4}b \right\rceil = 0 \rightarrow \text{if } b = 3 \end{cases}$$

Therefore, $\Delta = a + \frac{1}{2} = \left\lfloor \frac{\mathcal{U}}{4} \right\rfloor + \frac{1}{2}$ and, $\mathcal{U} - U_C \leq \Delta = \lfloor \frac{\mathcal{U}}{4} \rfloor + \frac{1}{2}$

The waste in the other 3 quarters is lower than or equal to $\lfloor \frac{\mathcal{U}}{4} \rfloor + \frac{1}{2}$

3.1) If L and W even, the waste in C must be integer.

Therefore, $\mathcal{U} - U_C \leq \left\lfloor \frac{\mathcal{U}}{4} \right\rfloor$

that is, the waste in the other 3 quarters would be at most $\lfloor \frac{\mathcal{U}}{4} \rfloor$

By horizontal symmetry: C $\leftrightarrow$ D and A $\leftrightarrow$ B,

we can obtain a valid solution with $U_A \leq \lfloor \frac{\mathcal{U}}{4} \rfloor$ and $U_C \leq \left\lceil \frac{3}{4}\mathcal{U} - 1 \right\rceil + \frac{1}{4}$

3.2) L odd, W even (there may be a fractional waste of 0.5 in C).

If $U_C > \left\lceil \frac{3}{4}\mathcal{U} - 1 \right\rceil + \frac{1}{4}$, then $\mathcal{U} - U_C \leq \Delta = \lfloor \frac{\mathcal{U}}{4} \rfloor + \frac{1}{2}$

The waste in the other 3 quarters is lower than or equal to $\lfloor \frac{\mathcal{U}}{4} \rfloor + \frac{1}{2}$.

If $\mathcal{U} - U_C = \lfloor \frac{\mathcal{U}}{4} \rfloor$ we are in case (3.1).

If $\mathcal{U} - U_C = \lfloor \frac{\mathcal{U}}{4} \rfloor + \frac{1}{2}$ there is at least a waste of $\frac{1}{2}$ in quarter A.

Therefore, $U_B \leq \lfloor \frac{\mathcal{U}}{4} \rfloor$. With an horizontal symmetry we obtain

a solution with $U_C \leq \lfloor \frac{3}{4}\mathcal{U} - 1 \rfloor + \frac{1}{4}$ satisfying $U_A \leq \lfloor \frac{\mathcal{U}}{4} \rfloor$.

3.3) L even, W odd.

The same argument of previous case with a waste of $\frac{1}{2}$ in quarter D.

3.4) L, W odd (there may be a fractional waste of 0.25 in C)

If $U_C > \left\lceil \frac{3}{4}\mathcal{U} - 1 \right\rceil + \frac{1}{4}$, then $\mathcal{U} - U_C \leq \Delta = \lfloor \frac{\mathcal{U}}{4} \rfloor + \frac{1}{2}$

The waste in the other 3 quarters is lower than or equal to $\lfloor \frac{\mathcal{U}}{4} \rfloor + \frac{1}{2}$

If $\mathcal{U} - U_C = \lfloor \frac{\mathcal{U}}{4} \rfloor + \frac{1}{2}$ there is either a waste of $\frac{1}{2}$ in quarter A or D or a waste of $\frac{1}{4}$ in quarter A and $\frac{1}{4}$ in quarter D.

In both cases, $U_B \leq \lfloor \frac{\mathcal{U}}{4} \rfloor$. With an horizontal symmetry we obtain a solution with $U_C \leq \lceil \frac{3}{4}\mathcal{U} - 1 \rceil + \frac{1}{4}$ satisfying $U_A \leq \lfloor \frac{\mathcal{U}}{4} \rfloor$

4) Obviously, in the last quarter D, $U_D \leq \mathcal{U}$.

References

[1] Amaral, A. and Wright, M. (2001) Experiments with a strategic oscillation algorithm for the pallet loading problem, *International Journal of Production Research* **39-11**, 2341-2351.

[2] Balas, E. and Padberg, M. (1976) Set partitioning: a survey, *SIAM Review* **18**, 710-760.

[3] Beasley, J. E. (1985) An exact two dimensional non-guillotine cutting tree search procedure, *Operations Research* **33**, 49-64.

[4] Bertsimas, D. and Vohra, R. (1998) Rounding algorithms for covering problems, *Mathematical Programming* **80**, 63-89.

[5] Bhattacharya, R., Roy, R. and Bhattacharya, S. (1998) An exact depth-first algorithm for the pallet loading problem, *European Journal of Operational Research* **110**, 610-625.

[6] Bischoff, E. and Dowsland, W.B. (1982) An Application of the Micro to Product Design and Distribution, *Journal of the Operational Research Society* **33**, 271-280.

[7] Bron, C. and Kerbosch, J. (1973) Algorithm 457. Finding all cliques of an Undirected Graph[H], *Communications of the CACM* **16**, 575-577.

[8] Cánovas, L. and Landete, M. and Marín, A. (2000) New facets for the set packing polytope, *Operational Research Letters* **27**, 153-161.

[9] Cheng, E. and Cunningham, W.H (1997) Wheel inequalities for stable set polytopes, *Mathematical Programming* **77**, 389-421.

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

[11] De Cani, P. (1979) Packing problems in theory and practice, *Ph.D.Thesis* Department of Engineering Production, University of Birmingham.

[12] Dowsland, K.A. (1984) The three-dimensional pallet chart: an analysis of the factors affecting the set of feasible layouts for a class of two-dimensional packing problems, *Journal of the Operational Research Society* **35**, 895-905.

[13] Dowsland, K.A. (1985) Determining an upper bound for a class of rectangular packing problems, *Computers and Operations Research* **12**, 201-205.

[14] Dowsland, K.A. (1987) An exact algorithm for the pallet loading problem, *European Journal of Operational Research* **31**, 78-84.

[15] Dowsland, K.A. (1987) A combined database and algorithmic approach to the pallet loading problem, *Journal of the Operational Research Society* **38**, 341-345.

[16] Dowsland, K.A. (1996) Simple tabu thresholding and the pallet loading problem. In: Osman, I.H. and Kelly, J.P. (Eds). *Metaheuristics: theory and applications*, Kluwer Academic Publishers, pp. 379-405.

[17] Elf, M. and Gütwenger, C. and Jünger, M. and Rinaldi, G. (2001) Branch-and-Cut Algorithms for Combinatorial Optimization and Their Implementation in ABACUS, in Jünger, M. and Naddef, D. (Eds.) *Computational Combinatorial Optimization: optimal or provably near optimal solutions* Lectures Notes in Computer Science **2241**, Springer, Berlin, pp. 1157-1223.

[18] Exeler, H. (1988) Das homogene Packproblem in der betriebswirtschaftlichen Logistik, Physica-Verlag, Heidelberg.

[19] Herbert, A and Dowsland, W. (1996) A family of genetic algorithms for the pallet loading problem, *Annals of Operations Research* **63**, 415-436.

[20] Herz, J.C. (1972) A recursive computational procedure for two-dimensional stock cutting, *IBM Journal of Research Development* **16**, 462-469.

[21] Hoffman, K. L. and Padberg, M. (1993) Solving Airline Crew Scheduling Problems by Branch-and-Cut, *Management Science* **39**, 657-682.

[22] Iserman, H. (1987)Ein Planungsssytem zur Optimierung der Palettenbeladung mit kongruenten rechteckigen Versangebinden, *OR Spektrum* **9**, 235-249.

[23] Lins, L., Lins, S. and Morabito, R. (2003) An L-approach for packing (l,w)-rectangles into rectangular and L-shaped pieces, *Journal of the Operational Research Society* **54**, 777-789.

[24] Letchford, A. and Amaral, A. (2001) Analysis of upper bounds for the Pallet Loading Problem, *European Journal of Operational Research* **132**, 582-593.

[25] Morabito, R and Morales, S. (1998) A simple and effective recursive procedure for the manufacturer's pallet loading problem, *Journal of the Operational Research Society* **49**, 819-828.

[26] Neliβen, J. (1993) New Approaches to the Pallet Loading Problem, *Schriften zur Informatik und Angewandten Mathematik* **155**; obtainable by the author or via anonymous ftp as /pub/reports/pallet.ps.qz on ftp.informatik.rwth-aachen.de

[27] Neliβen, J. (1995) How to use structural constraints to compute an upper bound for the pallet loading problem, *European Journal of Operational Research* **84**, 662-680.

[28] Nemhauser, G. and Sigismondi, G. (1992) A strong cutting plane/branch and bound algorithm for node packing, *Journal of the Operational Research Society* **43**, 443-457.

[29] Padberg, M. W. (1973) On the facial structure of set packing polyhedra, *Mathematical Programming* **5**, 199-215.

[30] Rossi, F. and Smriglio, S. (2001) A branch-and-cut algorithm for the maximum cardinality stable set problem, *Operations Research Letters* **28**, 63-74.

[31] Scheithauer, G. and Terno, J. (1996) The G4-Heuristic for the Pallet Loading Problem, *European Journal of Operational Research* **47**, 511-522.

[32] Smith, A. and De Cani,P. (1980) An algorithm to optimize the layout of boxes in pallets, *Journal of the Operational Research Society* **31**, 573-578.

[33] Steudel, H.J. (1979) Generating pallet loading patterns: A special case of the two-dimensional cutting stock problem, *Management Science* **25**, 997-1004.

[34] Strijk, T., Verweij, B. and Aardal, K. (2000) Algorithms for Maximum Independent Set Applied to Map Labelling, *Technical Report*, University of Utrecht, UU-CS-2000-22.

[35] Thienel, S. (1995) ABACUS A Branch And CUt System, Version 2.3, *Optimization Research And Software, OREAS*

[36] Wolsey, L.A (1998) Integer Programming, John Wiley and Sons.

[37] Young-Gun, G. and Maing-Kyu, K. (2001) A fast algorithm for two-dimensional pallet loading problems of large size, *European Journal of Operational Research* **134**, 193-202.