# A tabu search algorithm for the pallet loading problem

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

[†]University of Valencia, Department of Statistics and Operations Research, 46100 Burjassot, Valencia, Spain
[‡]University of Castilla-La Mancha. Departamento de Informatica, E.Politecnica Superior, 02071 Albacete, Spain

**Abstract**

This paper presents a new heuristic algorithm for the pallet loading problem, the problem of packing the maximum number of identical rectangular boxes onto a rectangular pallet. The problem arises in distribution and logistics and has many practical applications. We have developed a tabu search algorithm based on new types of moves. Instead of moving individual boxes, we propose moving *blocks*, sets of boxes with the same orientation. We have tested our algorithm on the whole sets Cover I and Cover II, usually taken as a reference for this problem, and we obtain excellent results in very short computing times.

## 1 Introduction

In the Pallet Loading Problem, PLP, the maximum number of identical rectangular boxes have to be packed onto a rectangular pallet. The problem arises at factories when large quantities of one product must be shipped onto pallets, and it is sometimes called the Manufacturer's Loading Problem (MLP) to distinguish it from the Distributor's Loading Problem (DLP) in which in boxes of several sizes are packed together on one pallet. Though the problem is initially three-dimensional, practical considerations of stability and handling usually mean that the boxes must be placed orthogonally with respect to the edges of the pallet and in the layers in which the vertical orientation of the boxes is fixed. With these restrictions the 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. Though the problem is very easy to solve in many cases, it cannot be considered to have been completely solved. For example, for the instance (89,75,10,7), a state-of-the-art commercial code for integer programming such as *CPLEX* took 89 hours of CPU time to obtain an optimal solution with 99 boxes. The problem has therefore continued to attract a lot of research in recent decades.

The exact algorithms proposed to date 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 [2],[5],[8],[9], [11], [15]. These exact algorithms have been able to solve problems of only moderate size, up to 50 boxes. Several upper bounds have also been proposed [6],[7],[16],[19], which consider the geometric structure of the problem and linearly relax integer programming formulations.

Many heuristic methods have been developed. On the one hand, there are constructive methods of increasing complexity, from simple structures in which the pallet is divided into blocks [3],[21], [22], to recursive procedures [17],[18], [20],[23]. On the other hand, there are metaheuristics based on tabu search, genetic algorithms and strategic oscillation [1],[10],[13].

The best results to date have been obtained by recursive procedures that exhaustively study special structures leading to good but not necessarily optimal solutions. The algorithm $G-4$ of Scheithauer and Terno [20] finds, in very short computing times, the best *G4-structures* of each problem, structures which produce the optimal solution for all the instances of set $CoverI$ (up to 50 boxes) and for most of the instances of set $CoverII$ (up to 100 boxes). The algorithm proposed by Lins et al.[17] finds the best L-decomposition. This structure contains $G-4$ as a special case and obtains more optimal solutions, but it is computationally much more expensive. The metaheuristic algorithms described so far present computational results limited to small subsets of problems and they have not yet obtained results comparable to those sophisticated recursive procedures.

The purpose of the present paper is to develop an algorithm, based on tabu search, that can compete with the best recursive procedures in terms of the quality of the solutions and the computing times. 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 Section 3

we present the main components of the algorithm, defining the moves, the tabu list and the intensification and diversification strategies. In Section 4 we present the computational results. Finally, concluding comments are made in Section 5.

# 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 $(L, W, a, b)$. Positions $(x, y)$ of the boxes are given with respect to a coordinate system which originates in the bottom left corner of the pallet.

Initially each box can be placed at any position on the pallet, that is, $x \in X = \{u | 0 \leq u \leq L - b, integer\}$, $y \in Y = \{v | 0 \leq v \leq W - b, integer\}$.

The set of positions for the boxes can be first reduced to the *normal sets* proposed by Herz[14] and Christofides and Whitlock[4]:

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

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

These sets can be further reduced by using dominance considerations, as proposed by Scheithauer and Terno[20]. If we denote:

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

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

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

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

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.*

If $(n, m)$ satisfy $0 = S - n * a - m * b$, they are a *perfect partition.*

Dowsland[6] 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[9]: *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:

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

The definition of sets Cover I and Cover II is subject to some ambiguity. Sometimes one 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. In order to avoid this ambiguity, we have followed the constructive method proposed by Dowsland[9] 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.

The *G-4 structure* was defined by Scheithauer y Terno[20] and it is a very efficient packing structure for the PLP. Some examples appear in Figures 6, 7, 8(a). Most of the optimal solutions for the problem contain some type of *G-4 structure*.

# 3 The Tabu Search algorithm

*Tabu Search* is now a well-established optimization algorithm (for an introduction, refer to the book by Glover and Laguna[12]. The basic elements of the algorithm are described in the next subsections.

## 3.1 Definition of a move

The solution space in which we move is composed of feasible solutions only. In this space we will define several moves to go from one solution to another. Analyzing different solutions for the problem, we observe that, rather than individual boxes, there are *blocks*, sets of boxes with a common orientation. Boxes not included in a block appear very seldom, unless they are included in a *G-4 structure*. Therefore, the moves we use are based on blocks and G-4

structures. This is a completely different approach from those already used in other metaheuristic algorithms.

The move will consist of changing the size of a block of the solution. We distinguish two types of moves:

- ***Block reduction***

  The size of a block is reduced horizontally or vertically by eliminating some of its rows or columns. That includes the possibility that the block may disappear completely.

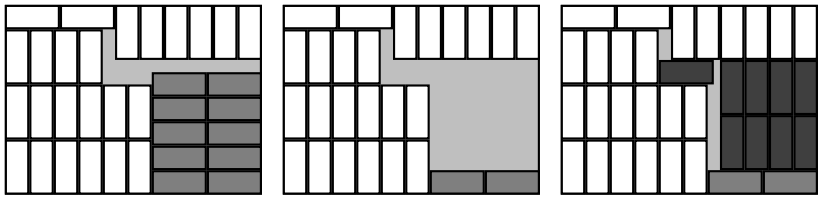
  The move starts by eliminating some of the rows or columns of the selected block from the current solution, then resulting in the appearance of a new region of *waste*, space not occupied by boxes. We move then the remaining blocks towards the nearest corners of the pallet so that waste is located mostly towards the center.

  In order to reutilize the new waste, the first step is to *merge the waste*, putting together the existing waste rectangles, new and old, in such a way that the new space can be filled by the maximum number of boxes.

  *The waste rectangles are then filled*, considering the horizontal or vertical block or the G4-structure that best fills each of them . Finally, *adjacent blocks with the same structure are merged* to simplify the solution. Examples of reduction moves appear in Figures 1 and 2. In Figure 1 a block is selected and reduced. None of the blocks is moved to the corners, the wastes are merged and then filled in two steps to produce a solution with more boxes. In Figure 2 the selected block disappears, another block is moved to the bottom right corner, the waste is merged and filled thereby producing a new solution which is optimal.

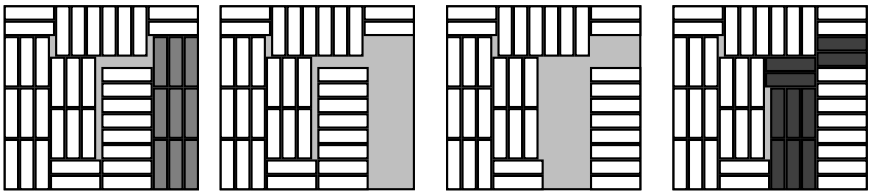
- ***Block augmentation*** The size of a block is increased horizontally or vertically by adding some rows or columns. The enlarged block may intersect with other blocks in which case the overlapping boxes are eliminated from the other blocks.

  Once we have enlarged the block and eliminated overlapping, we *merge the waste*, then fill *the waste rectangles* and finally *adjacent blocks with the same structure are merged* to simplify the solution, following the same steps of block reduction. Some examples of block augmentation appear in Figure 3.



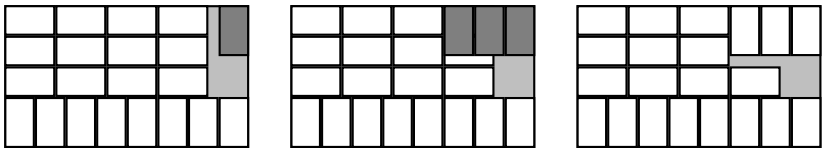
(a) Selection (b) Reduction (c) Filling

Figure 1: *Block reduction. Instance (42,31,9,4)*



(a) Selection (b) Reduction (c) Moving to the corner (d) Filling

Figure 2: *Block reduction. Instance (38, 36, 10, 3)*



(a) Selection (b) Increase (c) Elim. overlap.

Figure 3: *Block augmentation. Instance (24,14,5,3)*

We now describe the procedures used in the moves:

1. *Merging waste rectangles*

   We need a procedure to merge the waste rectangles which will be appearing in the process. This procedure is essential for the quality of the move. When we merge 2 rectangles, at most 3 new rectangles may appear, usually one large rectangle and 2 small ones (see Figure 4). Among the several alternatives of merging we try to select the most promising one. With this objective in mind, we impose some conditions. First, the rectangles to be merged must be adjacent and the common side must have a length of at least $b$ to accommodate at least one box. Second, after the merge, the new rectangles must accommodate more boxes than before the merge.

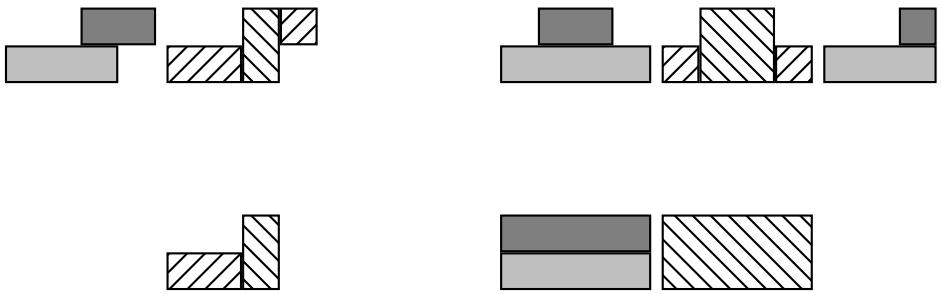


Figure 4: *Merging 2 waste rectangles*

2. *Moving blocks to the nearest corners*

   Every time a block is subject to some modification, we try to move it to its nearest corner, so that waste is concentrated in the middle, where it can be more easily merged (see Figure 5).

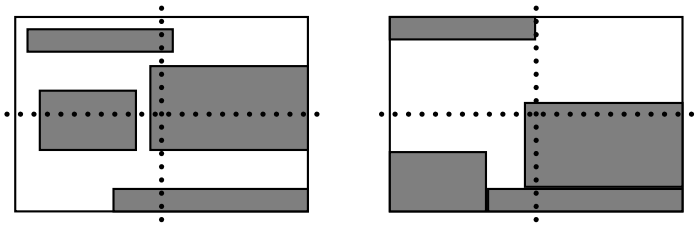


Figure 5: *Moving blocks towards the corners*

3. *Filling waste rectangles*

   We need a fast procedure to study efficient ways of filling each element of the list of waste rectangles. For each rectangle, we consider three possibilities: a horizontal block, a vertical block or a G4-structure and take the one accommodating the largest number of boxes. If there is a tie, we use blocks rather than G4-structures. If the waste has been produced by the reduction of a horizontal block, we prefer to fill it with a vertical block and conversely.

   The procedure works iteratively. The rectangle may be partially filled and the unfilled part is a new waste rectangle which is considered again for filling until its surface is smaller than that of a box.

   As was mentioned above, the G4-structure introduced by Scheithauer and Terno [20] is a very efficient packing structure for this problem and it appears very often in good solutions for the problem. We therefore consider it as an alternative to fill the rectangles. We do not study all the possible G4-structures but only what we call the *natural G4-structure*, that with the same number of horizontal and vertical boxes and producing the minimum inner waste. For example, for a $7 * 3$ box, the *natural G4-structure* will be that with 2 horizontal and 2 vertical boxes, as in Figure 6(a), leaving an inner waste of $1 * 1$. However, there may be alternative structures, maybe not so efficient in terms of the inner waste, but fitting on the pallet better. In the example, a new structure could be considered (Figure 6(b)), producing an inner waste of $2 * 2$. We leave the study of these alternatives to the diversification strategies described later.

   When choosing the *natural G4-structure* we also take into account whether its horizontal and vertical boxes would belong to *perfect parti-*

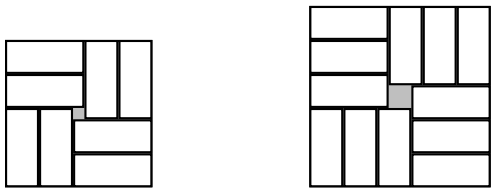
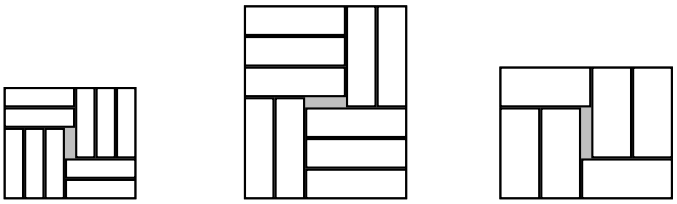


Figure 6: *Natural G4 structures for a* $7 * 3$ *box*

*tions* of the length and width of the pallet. If this is not the case, we do not choose the G4-structure. Proceeding in that way, it is possible that no G4-structure with the same number of horizontal and vertical boxes is left to be chosen. In this case, we consider structures with different numbers of horizontal and vertical boxes, as appears in Figure 7.



(a) (26,18,7,2)

(b) (36,28,10,3)

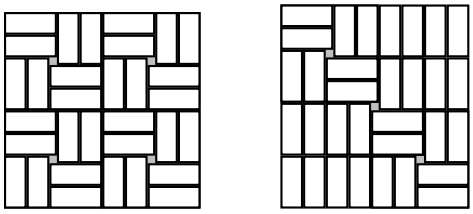
(c) (58,31,9,4)

Figure 7: *Other types of natural G4-structures*

Once we have chosen a G4-structure, we can obtain from it some other derived structures, repeating them or nesting them (Figures 8(a),8(b)).

4. *Merging blocks with the same structure*

   The number of moves to explore increases with the number of blocks. Therefore, at the end of each iteration we try to merge blocks with the same orientation and the same length or width if they are adjacent and have a common side, or if one of them can be moved to make them both adjacent to one common side (Figure 9).



(a) (b)

Figure 8: *Repeated and nested G4-structures*

(a) Before merging (b) Merged block

Figure 9: *Merging blocks*

## 3.2 Reducing the number of moves to explore

If we have a block of dimension $(n \times m)$, we can do $(n + m - 1)$ reductions and, depending on the size of the pallet, a number of augmentations. From all these possible moves, we are not going to study those considered not promising, according to several criteria. In that way, we increase the speed of the procedure, hopefully without a significant decrease in the quality of the solutions.

- If a block is not adjacent to any waste rectangle, it is not considered for moving. If the block is adjacent to a waste rectangle, we consider moves only in the direction in which there is waste.
- When we reduce or enlarge a block, we only explore the move if the new block can form part of an efficient partition of the length and width of the pallet.
- If the bottom left corner of the new block is not a rater point, we do not consider the move.
- If the new block will produce more waste than that associated with the best known solution, we do not explore the move.

## 3.3 Moves to be studied

At each iteration we apply the following three-step procedure to each block of the current solution:

1. Select a block adjacent to a waste rectangle.
2. Consider all types of moves, reductions and augmentations, in the directions adjacent to waste rectangles.
3. For each type of move, consider all possible numbers of rows and columns not eliminated by the criteria described above.

## 3.4 Selection of the move

The objective function consists only of maximizing the number of boxes packed onto the pallet. However, if the moves are evaluated according to that

function, there will be many moves with the same evaluation. Therefore, we have included other desirable characteristics of the solution in the evaluation function, which are listed below ordered in decreasing importance:

- *Symmetry.* We try not to explore symmetric solutions but prefer solutions in which waste is mostly concentrated to the right and to the top of the pallet.
- *Efficiency.* If possible, we prefer solutions in which the blocks are *efficient*. We say that a block is efficient if its length and width can make part of a perfect partition.
- *Centered and concentrated wastes.* We prefer solutions in which waste rectangles are centered and concentrated as much as possible, because that will make it easier to merge them and obtain spaces for more boxes.
- *Waste in pallet sides.* Solutions without waste in the sides of the pallet are preferred.
- *Improving.* We do not allow solutions with a block which cannot appear in a solution which is better than the best known solution.
- *New boxes.* We prefer a solution with some new boxes, rather than a solution which is the same as the previous solution but eliminates some of its boxes.

  These criteria are added to the evaluation function with some weights reflecting their relative importance, according to the results of a preliminary computational experience on a subset of problems.

## 3.5 Tabu list

At each iteration the block selected for the move is added to the tabu list. A move is then tabu if the new solution contains a block in the tabu list. For a given number of iterations we cannot visit a solution with a block with the same size and at the same position of a selected block.

The tabu list size varies dynamically. After a given number of iterations without improving the solution, the length is randomly chosen from $[0.5m^*, 1.5m^*]$, where $m^* = \lfloor \frac{L*W}{l*w} \rfloor$ (the upper bound based on the pallet area).

The *aspiration criterion* allows us to move to a tabu solution if it improves the best solution obtained so far.

## 3.6 Local improvement

At the end of each iteration we try a local improvement. We put a G-4 structure at every corner of the pallet, one at a time, eliminate the overlapping boxes, and see if the new solution is better than the current solution.

## 3.7 Intensification and diversification strategies

The moves we have defined involve a high level of diversification. However, we have included two more diversification strategies:

- Strategic oscillation

  Our evaluation function has several terms whose relative importance is controlled by a set of weights. These weights have been chosen for having a good average behavior, but maybe they are not the most appropriate for some particular instances. Therefore, we make them oscillate in the following way:

  We use a coefficient $\beta_i$, $i = 1, \ldots 6$ which multiplies every weight. Initially every $\beta_i = 1$, $i = 1, \ldots 6$. After a given number of iterations without improving the best known solution, the coefficient oscillates between 0 and 1.

- Long term memory

  Along the search process, we keep in memory the frequency of each block appearing in the solutions, where a block is defined by its size and orientation.

  This information is used for diversification and for intensification purposes. When used for diversification, we favor the moves of blocks appearing very frequently in the solutions, then inducing new blocks to appear. When used for intensification, we keep in memory only blocks corresponding to high quality solutions and then we favor these blocks appearing again in the new solutions.

# 4 Computational results

We have coded our algorithm in C++ and made our tests on a *Pentium 4* at 2.0GHz. After every 100 iterations without improving the best known solution we change the *natural G-4 structure* to be considered, if there is more than one alternative. Also, the length of the tabu list changes every 100 iterations without improvement. After 10000 iterations without improving we perform a strategic oscillation of the objective function coefficients and after 9000 non-improving iterations we do a diversification phase based on long term frequencies over 250 iterations and then an intensification phase over 250 iterations.

All the 7827 classes of *Cover I* were solved to optimality in less than 250 iterations. That supposes less than 1 second of CPU time. The results for the complete set *Cover II*, 40609 classes, appear in Table 4, which shows the number of instances not optimally solved. A limit of 100000 iterations has been imposed in order to keep the CPU time below the 10 minutes.

Table 1: *Cover II (40609 classes).*

| Iterations | Non-optimal |
|---|---|
| 500 | 229 |
| 1000 | 137 |
| 10000 | 38 |
| 50000 | 23 |
| 100000 | 15 |

Heuristic G-4 of Scheithauer and Terno [20] is available on Internet and we have used it to solve all the classes of *Cover II*. Only 45 of them were not solved to optimality because they do no not have a G-4 structure. Therefore, our Tabu Search algorithm, which solves all the instances solved by the G4 procedure and some others, is prepared to go one step further and find optimal solutions for problems not having G4-structure at a reasonable computational cost. Figure 10 shows an example of optimal solution with no G-4 structure.

The algorithm proposed more recently by Lins et al. [17] is more complex. It exhaustively studies L-structures, which include G-4 structures as particular cases. For simplicity the authors have restricted their study to a subset of *Cover II*, of more than 20000 instances, and it has solved all

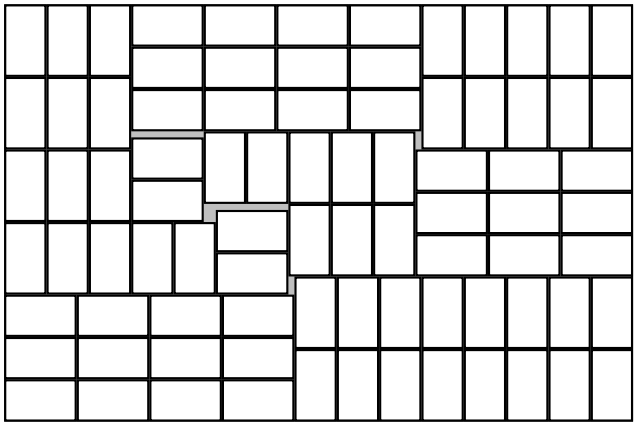


Figure 10: *Instance (104,69,12,7) of Cover II*

of them optimally, though in some cases the computational times are very high (more than 360 minutes on a Pentium III at 700 Mhz). In fact, they conjecture that their algorithm always finds optimal solutions for the pallet loading problem. If this conjecture could be proven, the algorithm would no longer be heuristic and its lengthy computational times should be compared with other exact approaches.

On this subset of instances our Tabu Search algorithm has found the optimal solution for all but one instance, (74,46,7,5), within the imposed limit of 10 CPU minutes. Therefore, our algorithm seems to achieve a good balance between solution quality and computational cost.

Finally, in order to explore the possible extension of our algorithm to larger problems, we have generated what we have called *Cover III*, the set of classes satisfying:

$$1 \leq \frac{L}{W} \leq 2, \qquad 1 \leq \frac{a}{b} \leq 4, \qquad 101 \leq \frac{L * W}{a * b} < 151$$

extending the structure of Cover I and Cover II to instances of up to 150 boxes. There are 98016 equivalence classes for which the optimal solution is not known, but for which several upper bounds can be computed. For that set, we are able to compare the solutions obtained by our heuristic and the upper bounds. Table 4 shows the number of problems for which the solution obtained by the heuristic does not match the corresponding upper bound. These results allow us to conclude that the tabu search algorithm optimally

solves most of the classes of *Cover III*. Figure 11 shows a solution with 149 boxes.

Table 2: *Cover III (98016 Instances).*

| Iteration | Non-optimal |
|---|---|
| 1000 | 1770 |
| 10000 | 1263 |
| 25000 | 1202 |
| 50000 | 1173 |
| 100000 | 1122 |

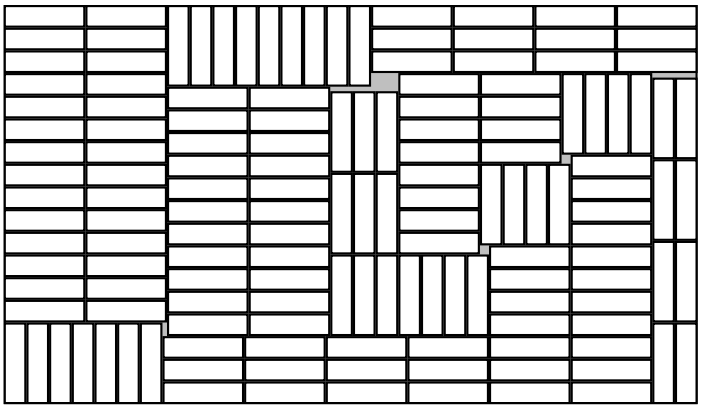


Figure 11: *Instance (153, 88, 18, 5) of Cover III*

# 5 Conclusions

We have developed a new Tabu Search algorithm which produces excellent results for the whole sets *Cover I* and *Cover II* and which can be extended to a new set *Cover III* of equivalence classes for problem instances of up to 150 boxes.

New types of moves have been defined. On the one hand, increasing and decreasing blocks and on the other hand those which explicitly consider the inclusion of G4-structures which have been shown to be very efficient for this particular packing problem. These moves are embedded in an algorithmic

scheme combining diversification and intensification strategies. The results show that these metaheuristic procedures can obtain good results in reasonable computing times for large-size pallet problems. We are currently trying to apply this type of ideas to more complex packing problems, involving boxes of different sizes.

# References


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

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

[3] Bischoff EE, Dowsland WB (1982) An Application of the Micro to Product Design and Distribution. Journal of the Operational Research Society 33: 271-280

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

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

[6] Dowsland KA (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

[7] Dowsland KA (1985) Determining an upper bound for a class of rectangular packing problems. Computers and Operations Research 12: 201-205

[8] Dowsland KA (1987) An exact algorithm for the pallet loading problem. European Journal of Operational Research 31: 78-84

[9] Dowsland KA (1987) A combinated data-base and algoritmic approach to the pallet-loading problem. European Journal of Operational Research 38: 341-345

[10] Dowsland KA (1996) Simple tabu thresholding and the pallet loading problem. In: Osman IH, Kelly JP(Eds) Metaheuristics: theory and applications, Kluwer Academic Publishers, pp 379-406

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

[12] Glover F, Laguna M (1997) Tabu Search, Kluwer Academic Publishers, Boston.

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

[14] Herz JC (1972) A recursive computational procedure for two-dimensional stock cutting. IBM Journal of Research Development 16: 462-469

[15] Iserman H (1987) Ein Planungsssytem zur Optimierung der Palettenbeladung kongruenten rechteckigen Versangebinden. OR Spektrum 9: 235-249

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

[17] Lins L, Lins S, 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

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

[19] Neli$\beta$en 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

[20] Scheithauer G, Terno J (1996) The G4-Heuristic for the Pallet Loading Problem. European Journal of Operational Research 46: 511-522

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

[22] Steudel JH (1979) Generating pallet loading patterns: A special Case of the two-Dimensional Cutting Stock problem. Management Science 25: 997-1004

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