Alma Mater Studiorum University of Bologna

Ph.D. in Control System Engineering and Operational Research

MAT/09

XXIII Cycle

The Cutting Stock Problem in the Wood Industry

Rosa Daniela Medina Durán

Coordinator	**Relators**
Prof. Paolo Toth	Prof. Paolo Toth

Final exam 2011

Contents

Acknowledgments

First of all I would like to express my sincere gratitude to Prof. Paolo Toth, for his continuing support, his helpful suggestions and important advice, and to Ing. Enrico Malaguti, for his instructions and patience, I would have been lost without them. I really appreciate their direct cooperation on the development of this study.

I would like to thank Nirvanatec, TMachines, Optimalon Software and Soft Consult who agreed to compare their software with the algorithms proposed.

I would like to extend my thanks to those who have helped me during the doctorate program. In particular, to Prof. Lorena Pradenas for her confidence that led me to start the program. I also wish to express my appreciation to the Operational Research Group DEIS - UNIBO, specially my officemates, Albert E. Fernandes M., Fabio Furini and Alfredo Persiani with whom I have shared academic activities and support.

Finally, I would like to thank my family and friends, specially those friendly people I have met during my stay in Bologna, thank you for enriching my experience.

This work was partially supported by MIUR, Italy and the Beca Presidente de la República - CONICYT, Government of Chile, 2008.

Keywords

Combinatorial Optimization, Two-Dimensional Cutting Stock Problem, Column Generation.

Abstract

This thesis proposes a solution for board cutting in the wood industry with the aim of usage minimization and machine productivity. The problem is dealt with as a Two-Dimensional Cutting Stock Problem and specific Combinatorial Optimization methods are used to solve it considering the features of the real problem.

The thesis is organized as follows:

Chapter 1 The *Introduction* contains a detailed overview of the problem and a theoretical and methodological review of the literature.

Chapter 2 The second chapter, *Constructive Heuristic Algorithms*, presents detailed heuristic algorithms to solve the problem.

Chapter 3 The third chapter, *Column Generation*, presents a column generation approach to solve the linear relaxation of the problem and strategies to obtain an integer solution.

Chapter 4 The fourth chapter, *Post Optimization*, presents two techniques to improve the solutions.

Chapter 5 The fifth chapter, *Experiments*, presents the comparison of the developed algorithm with commercial software packages.

Chapter 6 The last chapter, *Summary*, provides some concluding remarks on the study.

List of Figures

List of Tables

As this emotion represents a drive to know new things,
curiosity is the fuel of science and all other disciplines of human study.
"Curiosity", Wikipedia, The Free Encyclopedia

Chapter 1

Introduction

In the wood industry, the raw material is present as rectangular wood boards that must be cut to obtain the required dimensions of the rectangular items used in further processes. The boards are all of the same material in different sizes, called classes of boards, and are available in a large number so as to consider them infinite. The rectangular items to be obtained are of different dimensions, smaller than the boards, and have specific demands that must be satisfied. Some items have a fixed orientation, according to the grain of the wood, while others can be orthogonally rotated, this means that the dimensions of the item can be swapped.

Before starting the cutting process, it is necessary to decide the appropriate layout of the items on the board, namely cutting patterns, for an appropriate use of the resources: the raw material, the cutting machine and the processing time. Consequently, the cutting patterns and their repetition must be determined considering the characteristics of the cutting machine in order to satisfy the demand for the items.

Unlike other cutting machines, wood cutting machines make it possible to stack the boards, in order to cut more than one board at a time, all of them with the same pattern, thus improving machine productivity. Note the difference between the number of boards to be cut (the required stocks to satisfy the demand), the number of different patterns and the number of processes of the machine, namely cycles. At each cycle, the machine follows a single pattern at least once. The maximum number of boards simultaneously cut depends on the boards' thickness and the machine maximum load.

The wood cutting machine performs only straight cuts from edge to edge, parallel to the board edges, called guillotine cuts. Some cutting machines can cut in two directions, i.e., the machine automatically turns the board 90° to perform a new cut; but usually, it is necessary for a worker to turn the board in the other direction, increasing the workload and processing time, and consequently the maximum number of cutting directions, called stages, is usually restricted. In the first stage, the board is cut by its length, or width, obtaining strips (horizontal or vertical respectively). Second orthogonal cuts are performed to obtain the items in the second stage. Third stages can be performed on the items to obtain smaller items or remove waste.

Another feature of the cutting machine is the possibility to perform a pre-cut in the board and process each part independently; in particular, one part can have horizontal strips while the other can have vertical strips.

For some boards, a trimming cut is necessary to polish the edges. Moreover, each time the blade of the machine performs a cut, a thin strip of board is removed. The machines have a precision of 10^{-4} meters, but for most instances the dimensions are expressed in millimeters.

We dealt with this problem as a Combinatorial Optimization Problem and used specific techniques to solve it with the aim of minimizing the stocks usage and maximizing the machine productivity. The problem is formally defined in Section 1.1, and the theory and methodology of the literature are reviewed in Sections 1.2 and 1.3.

1.1 Problem Description

In a formal definition, an instance of board cutting in the wood industry has:

A set $\mathbb{B}$ of b classes of rectangular boards. Each board class k ($k = 1, \ldots, b$) has dimensions (L_k, W_k), length and width respectively. All the board classes have the same thickness and there are an infinite number of them. Knowing the thickness of the boards and the maximum load of the machine, we define κ as the maximum number of boards that the machine can cut simultaneously.

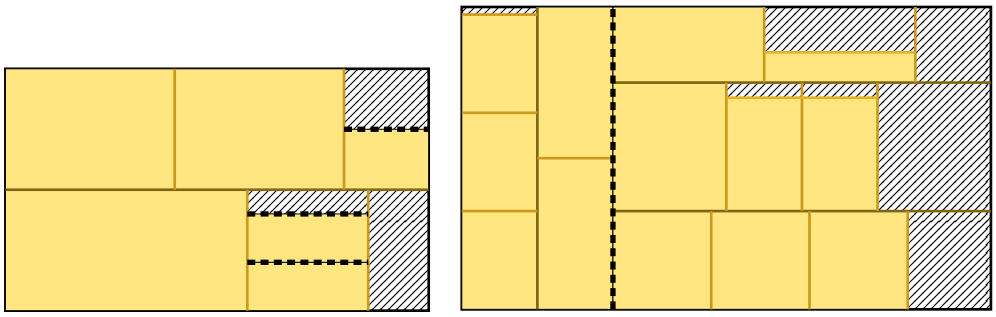
A set $\mathbb{I}$ of m classes of rectangular items. Each item class i ($i = 1, \ldots, m$) has dimensions (l_i, w_i), length and width respectively, demand d_i and a binary attribute r_i that indicates if the item class has a fixed orientation, i.e., if $r_i = 1$ the item class can not be orthogonally rotated. The item classes can be cut from at least one board class and all the item classes have different dimensions. If there are item classes with the same dimensions and rotation attribute, in a preprocessing step, the classes are merged, thus decreasing the number of classes and increasing the corresponding demand.

With no loss of generality, we do not consider blade width or trim cut. For the blade width, it is sufficient to add the blade width to each item dimension and to each board dimension vertically and horizontally. For the trim cut, it is sufficient to subtract the trim to the board dimension twice vertically and horizontally.

A solution for the problem has:

A set $\mathbb{P}^*$ of n^* different feasible cutting patterns, see Figure 1.1 where the lines represent the cuts in a board to obtain the items. A feasible cutting pattern has only guillotine cuts parallel to the board edges, generating strips. If the first cut is parallel to the board length, it generates horizontal strips; otherwise, a first cut parallel to the board width generates vertical strips. Subsequent guillotine cuts are necessary to obtain the items, with a maximum of three cuts. The third stage is allowed for trimming, i.e., to obtain the desired item dimensions (black dashed line in the strip at the top of Figure 1.1a); or to separate two small items (black dashed lines in the strip at the bottom of Figure 1.1a). A pre-cut in the board is also admissible to combine two feasible patterns of different directions (black dashed line in Figure 1.1b). Besides the layout to obtain the items, for each cutting pattern j ($j = 1, \ldots, n^*$), the board class used (B_j), the repetition of the pattern in the solution (R_j), the number of third-stage cuts in the pattern (Z_j) and the set of items in the pattern (I_j) are also known.

Different solutions for a given instance are compared considering the stock usage and machine productivity. The stock usage is evaluated with the total used area, while machine productivity is evaluated considering the number of cycles and the number of third-stage cuts at each cycle. We minimize a weighted function (called *index*) of the area (A), machine cycles (C) and third-stage cuts (Z), (see Equation 1.1).



(a) third-stages (b) pre-cut

Figure 1.1: Cutting Patterns Examples

$$index = \left\lfloor \frac{10}{3} \cdot A \right\rfloor + \frac{8}{3} \cdot C + 0.3 \cdot Z \tag{1.1}$$

$A = \sum_{j=1}^{n^*} A_j^k$ is the sum of the area of all the boards used in the solution, calculated in square meters, with A_j^k the area of board $B_j = k$. $C = \sum_{j=1}^{n^*} \lceil \frac{R_j}{\kappa} \rceil$ are the machine cycles, calculated as the number of times that the machine starts a process, i.e., whether it reaches κ boards or it changes the cutting pattern. $Z = \sum_{j=1}^{n^*} \lceil \frac{R_j}{\kappa} \rceil \cdot Z_j$ are the third-stage cuts, calculated as the sum of the of third-stage cuts of the cycles.

1.2 Literature Review

According to the Combinatorial Optimization literature, the problem belongs to the family of Cutting Stock Problems specifically, the Two-Dimensional Cutting Stock Problems (2DCSP) where rectangular stocks (raw material) might be cut to obtain the required dimensions. These problems are NP-hard [9], this means that all algorithms currently known for finding optimal solutions require a number of computational steps that may grow exponentially with the problem size rather than according to a polynomial function. Following the literature nomenclature, the wood industry problem can be further classified as Multiple Stock, because the stock boards have different dimensions, and Two-Stages Non Exact Guillotine because only guillotine cuts are admissible and a third guillotine cut is feasible to separate an item from waste. In practice, a third cut is also admissible to separate two small items, but as far as we know this feature has not been described in the literature.

To solve the 2DCSP, exact and heuristic approaches have been proposed. The heuristic methods are more flexible for considering specifics constraint of the problem and to have a good trade-off between the quality of a solution and its computational effort. One of the most common approaches to deal with the 2DCSP is the column generation approach proposed by Gilmore and Gomory [10], [11], [12]; they proposed the $k - staged$ pattern version and also considered the version with bins of different sizes. This approach is very effective in

minimizing the number of cutting patterns or the area. The solution of this approach is fractional, and to obtain an integer solution many techniques have been developed. Alvarez-Valdes et al. [2] developed and compared several heuristic methods for solving the 2DCSP, based on column generation. Riehme et al. [22] considered the two staged version of the problem in the case where boards of different sizes are available and the item demands differ in a large range. Vanderbeck [26] considered the 2DCSP with three stages, whereby unused parts of some stock can be used later as new stock. The author developed an approximate solution method based on a nested decomposition of the problem. Each subproblem is solved by column generation and then a rounding heuristic finds an integer feasible solution. Cintra et al. [4] considered several 2DCSPs with guillotine cuts and their variants in which orthogonal rotations are allowed and boards of different sizes are available. They presented dynamic programming algorithms for the Rectangular Knapsack Problem, which were then used to generate patterns in approaches for the 2DCSP based on column generation.

A preliminary work was presented In Furini et al. [7] and in Furini et al. [8], where we developed heuristic procedures based on a column generation approach to solve the general 2DCSP having bins of different sizes with the objective of minimize the used area. A further contribution consists in the definition of a Mixed Integer Linear Programming Model for the solution of this Knapsack Problem, as well as a heuristic procedure based on dynamic programming. In Malaguti et al. [17], we considered the heuristics propose in this study and presented a complete method for solving the real world 2DCSP optimizing a weighted function of the used area, cycles and third stage cuts. We give here a more detailed description of the algorithms.

For the objective of minimizing Equation 1, we noted that minimizing the stock usage is partially in contrast with the maximization of the machine productivity. While minimizing the stock usage requires an effective use of the boards (minimum waste), machine productivity requires an efficient distribution of the items in the patterns in order to minimize the cycles and the third-stage cuts. To minimize the number of cycles, it is desirable to have a reduced number of patterns with a high repetition to take advantage of cutting more than one pattern each time. A repetition of the pattern near to, although smaller than, a multiple of κ is also desirable.

To generate two stages guillotine patterns, we used the strip approach, considering first grouping the items into strips and then grouping the strips to create a pattern. This approach can be generalized to generate three-stage guillotine patterns by considering, in the first step, items of different sizes.

The development of the algorithms is done in an empiric manner, The components are added one at a time and the parameters are set on the basis of systematic and thorough testing. Many combinatorial optimization techniques are used. First, a Constructive Heuristic Algorithm is developed and used as a starting solution for a Column Generation procedure. Strategies to obtain integer solutions are studied and Local Search techniques are used to improve it. Many of these procedures solve other classic combinatorial optimization problems as subproblems, in Section 1.3 we present the algorithms to solve these subproblems that are used later by the proposed algorithm.

1.3 Heuristics for solving the subproblems

The Bounded Knapsack Problem (BKP) aims to solve the following problem:

$$max\{\dot{p}\dot{x}\colon \dot{w}\dot{x} \leq \dot{c}, 0 \leq \dot{x} \leq \dot{d}, \dot{x} \in \mathbb{Z}^{\dot{n}}\} \tag{1.2}$$

where $\dot{p}, \dot{w} \in \mathbb{R}_+^{\dot{n}}$, $\dot{d} \in \mathbb{Z}_+^{\dot{n}}$ and $\dot{c} \in \mathbb{R}_+$ are given. Each object i ($i = 1, \ldots, \dot{n}$) has a weight $\dot{w}_i$ and a profit $\dot{p}_i$ and is available in $\dot{d}_i$ copies. The variable $\dot{x}_i$ denotes the number of objects of class i in the solution. The BKP can be interpreted as the selection among $\dot{n}$ objects that maximizes the profit without exceeding the capacity $\dot{c}$. The BKP is a generalization of the classic 0-1 Knapsack Problem, which is known to be NP-hard. In the literature, we find many algorithms to solve this problem (see, e.g., [18] and [14]). We implemented the Greedy Algorithm which is fast and generally gives good solutions.

The Greedy Algorithm considers the ratio $\dot{p}_i/\dot{w}_i$ ($i = 1, \ldots, \dot{n}$), measuring the convenience of an object, and sorts the objects according to this ratio in a non increasing way. The items are inserted following this order until the capacity is reached, i.e., no other item can be inserted without exceeding the capacity. If the items are ordered, the algorithm provides a good solution to the problem in $O(\dot{n})$ time.

The Bin Packing Problem (BPP) aims to find the minimum number of bins of capacity $\ddot{c}$ that contains all the given $\ddot{m}$ objects. Each object i ($i = 1, \ldots, \ddot{m}$) has a weight $\ddot{w}_i$. The BPP is NP-hard. Solving the BPP to optimality may be a very difficult and time consuming task [6]. We found good solutions by solving it heuristically with the Best-Fit Decreasing Heuristic (BFD) [25].

The BFD considers the objects ordered by non-increasing weight and all the bins empty, i.e., with residual capacity $\ddot{c}$. Following the order of the objects, each object is inserted in the bin with smallest residual capacity from among those that can accommodate this object.

The Two-Stage Two-Dimensional Knapsack Problem (2TKP) [15] determines a cutting pattern for a rectangle $\ddot{S}$, maximizing the global profit of the rectangular pieces obtained with two cuts and allowing a third-cut for trimming, i.e., separating a piece from waste. The rectangle $\ddot{S}$ has dimensions $(\ddot{L}, \ddot{W})$, length and width respectively. The rectangular pieces are selected from $\ddot{m}$ classes, each class i having dimensions $(\ddot{l}_i, \ddot{w}_i)$, length and width respectively, profit $\ddot{p}_i$ and a positive availability $\ddot{d}_i$. The dimensions of the pieces are feasible for the rectangle, i.e., $\ddot{l}_i \leq \ddot{L}$ and $\ddot{w}_i \leq \ddot{W} \quad \forall \quad i = 1, \ldots, \ddot{m}$. The cuts must be of the guillotine type, i.e., edge to edge straight cuts parallel to one of the rectangle edges. We found many heuristic and exact algorithms to solve this problem.

A Greedy Randomized Adaptive Search Procedure (GRASP) and a Path Relinking Procedure were developed by Alvarez-Valdes et al. [1] for the 2TKP. Briefly, the GRASP is an iterative process in which each iteration consists of two phases. The construction phase builds a feasible solution, whose neighborhood is explored until a local optimum is found after application of the local search phase. The procedure can be "strip-oriented" or "piece-oriented".

The construction phase of the "strip-oriented" procedure (GRASP_Strip), solves a series of BKP for each different $\bar{\ddot{w}}_s \in \bar{\ddot{\mathbb{W}}} = \{\bar{\ddot{w}}_s\colon \exists \ddot{w}_i = \bar{\ddot{w}}_s, i = 1, \ldots, \ddot{m}\}$ with pieces having a smaller or equal width, using $\ddot{l}_i$ as weight and $\ddot{L}$ as capacity. The solution of the BKP determines the strip profit. A strip is randomly selected between the strips with profits larger than a threshold and the strip is added to the rectangle, the piece availability is updated and the cycle is repeated until the remaining rectangle width is too small or no piece is available.

The threshold depends on the current largest strip profit, ensuring at least one strip. The improvement phase selects a strip to be removed from the current rectangle and the emptied space, merged with the strip of waste if it exists, is filled with a greedy procedure which inserts the pieces, ordered by their non increasing widths, on the most possible bottom-left corner. The procedure checks that the removed strip is not regenerated.

The construction phase of the "piece-oriented" procedure (GRASP_Pieces) has two stages. The first stage selects a suitable width, i.e., width not greater than the current free width of the rectangle, to create a strip while the second stage fills the strip with pieces. Each suitable width has a value defined as the average profit of the pieces with width smaller than or equal to the suitable width. A width is randomly selected among the suitable widths with a value larger than a threshold. The threshold depends on the current largest width value, ensuring at least one width. In the second stage, a list is created with pieces having lengths smaller than or equal to the remaining strip length and profits larger than a threshold (defined by the current largest piece profit). A piece on this list is randomly selected to be added to the strip and the piece availability is updated. The second stage is repeated until the remaining strip length is too small or no piece is available. Then a new strip is created with the first stage, repeating the stages until no strip can be created because the rectangle width is too small or no piece is available. The improvement phase selects each piece to be considered for a move. In detail, for each piece in the solution and each piece not in the solution but with positive demand and feasible dimensions (width smaller than or equal to the strip width and length smaller than or equal to the selected piece plus the remaining strip length) an improvement index defined by the difference of the pieces profits is calculated. The piece that provides the largest positive improvement index is selected for the replacement; if no such piece exists (all improvement indexes are negative) the piece in the solution is not replaced.

The Path Relinking Procedure generates new solutions by exploring trajectories that connect high quality solutions. In particular, starting from a solution, called an initiating solution, the strips of a better solution, called a guiding solution, are added one by one. The strips in the initiating solution that exceed the rectangle capacity are eliminated. Moreover, the strips of the initiating solution containing pieces that produce an excess of the availability are also eliminated. All the empty spaces produced in these moves are merged into a waste strip that is filled with a greedy procedure which inserts the remaining pieces, ordered by their non increasing widths, on the most possible bottom-left corner. The combinations of strips from GRASP_Strip or GRASP_Pieces can be very fruitful in the search for solutions of even higher quality.

Chapter 2

Constructive Heuristic Algorithms

In this chapter we propose heuristic algorithms to solve the Two-Dimensional Cutting Stock Problem. The algorithms consider the stack of the boards and third-stage guillotine cuts in order to furnish a suitable solution for the problem in the wood industry. The aim of the algorithms is to minimize the area of used boards, the number of cutting cycles and the number of third-stage cuts.

Minimizing the area of used boards is equivalent to reducing the waste of the solution. However, waste minimization could imply a large number of patterns which affects the number of cycles as it depends on the number of cutting patterns and the pattern repetition in the solution. To minimize the number of cutting cycles, a small number of cutting patterns with a repetition near to, although not larger than, a multiple of the maximum stack capacity of the machine would be desirable .

The minimization of the number of third-stage cuts is achieved through a strip approach. In the strip approach a pattern is built by joining strips that contain the items. In this manner, the guillotine cut type is ensured and a third-stage cut is needed only to remove waste from an item or to obtain smaller items.

Briefly, given an instance of the 2DCSP and the maximum number of stack boards, the *stripGenerator* procedure creates a list of strips for each board and direction, horizontal or vertical. The strips are created by combining the items, and those strips with a filling percentage above a specified threshold are selected. The items in the selected strips are removed from the set of items. With each list, the *patternGenerator* procedure fills the boards. Within this procedure, the *patternFilling* procedure tries to insert some of the remaining items in the empty space of the boards. All the patterns, with a waste below a specified threshold, are selected for the final solution. The other patterns are unpacked, i.e., the items in the pattern are included in the set of remaining items. If there are still remaining items after the termination of the *patternGenerator* procedure, the *remainingItemsInsertion* procedure tries to insert the remaining items in any pattern in the solution. If there are still remaining items, a *residualProblem* procedure combines those items into patterns.

To increase the pattern repetition, the *stripGenerator* and *patternGenerator* procedures are iteratively solved by limiting the item class demand to a fictitious number that depends on the maximum stack boards in the machine, as described in Algorithm 1. The pattern repetition is set to the maximum number that does not generate overproduction, defined as: $O_j = min_{i \in I_j}\{\lfloor \frac{d_i}{p_j^i} \rfloor\}$ where p_j^i is the number of items of class i in pattern j.

Algorithm 1 Constructive Heuristic

Input: Instance of 2DCSP and maximum stack κ
Output: Set of cutting patterns
  $t = \kappa$
  repeat
    Call *stripGenerator* procedure with $\tilde{d}_i = \lfloor \frac{d_i}{t} \rfloor$ for items with $d_i \geq t$
    for board and direction **do**
      Call *patternGenerator* procedure
    end
    Unpack patterns not used and increase d_i for the items in those patterns
    $t \leftarrow t - 1$
  until $t \leq 0$
  if $\exists d_i > 0 (i = 1, \ldots, m)$ **then**
    Call *remainingItemsInsertion* procedure
  end
  if $\exists d_i > 0 (i = 1, \ldots, m)$ **then**
    Call *residualProblem* procedure
  end

In this chapter, we describe the procedures presented before. Section 2.1 describes the procedure to generate the strips. Section 2.2 describes the procedure to make the patterns. Section 2.3 describes the procedure that fills an empty rectangle. Section 2.4 describes the iterative cycle used to increase the pattern repetition. Section 2.5 describes how the items are inserted into patterns in the solution. Section 2.6 describes the heuristic used to solve the remaining problem, i.e., a small 2DCSP. Finally, Section 2.7 presents computational results and the study of the parameters of the algorithm.

2.1 Strip Generation

Given the set of items, the *stripGenerator* Procedure creates a list of strips for each board class and direction, vertical or horizontal. The strips in the list have a waste below a prescribed threshold β. The waste is calculated as a percentage of the strip area. Given a board with length L and width W; in the horizontal direction, L determines the length of the strips, this means that the items, placed end to end, can not exceed it. To determine the set of items in the strip, we solve a BKP [3].

In the BKP, the capacity corresponds to L. The objects are the items using its length (width for rotated items) as weight and a function of its area as profit. As larger items are more difficult to place later in the algorithm, we privilege these items by using as profit the α-power of their percentage area with respect to the area of the board, with α a parameter.

First, a list of strip widths is determined with all different item widths, or item lengths for items that can be rotated. The cycle starts by solving a BKP for each direction, board type and strip width using the items that do not exceed the dimensions of the board and the strip width. For example, considering a horizontal direction, the item length must be not larger than L and the item width must be equal to the actual strip width, or, if the item can be rotated, the item width must be not larger than L and the item length must be equal to the

actual strip width. By solving the BKP, a strip is created, and its waste and repetition are computed. The strip with minimum waste is selected among all the strips, and, if its waste is below the threshold β, it is added to the strip list corresponding to the board and direction. The items in the selected strip are no longer available according to the strip repetition. All the other strips are discarded. The described cycle stops as soon as the minimum strip waste is larger than or equal to the threshold.

The BKP is solved several times and many of its solution are discarded; we use the Greedy Algorithm to reduce the computation time, see Section 1.3.

2.2 Pattern Generator

As stated before, the *patternGenerator* Procedure creates patterns with the strip list generated by the *stripGenerator* Procedure (Section 2.1) associated with a board class and a direction, vertical or horizontal. Beside the strip list, the procedure requires the board class to be filled and the γ parameter for the pattern waste threshold. In order to use the least number of boards and minimize the number of patterns, a BPP is solved.

Consider a list of horizontal strips for a board with width W. In the BPP, the capacity of the bins is the board width and the objects are the strips, where the strip width is used as weight. The BPP is heuristically solved using the BFD, see Section 1.3, and the solution is improved through the tabu search algorithm proposed by Fernandes Muritiba et al. [19].

Briefly, the tabu search algorithm starts by choosing one bin, say h and, iteratively, tries to empty the bin by randomly choosing a strip which is moved from this bin to another bin where it fits. If the strip cannot fit in any bin, it is included in the bin, say l, with the largest empty space. Feasibility is maintained by moving strips from l to h. An anti-cycling rule forbids one strip to enter a bin it left during the last T iterations (where T is a parameter). Even if bin h is not emptied, its filling is minimized, and the empty space can be filled with the available items through the *patternFilling* Procedure. Only the patterns with waste below γ are kept, calculating their maximum repetition and the items placed by the *patternFilling* Procedure are no longer available. The *patternFilling* Procedure is described in the next section.

2.3 Pattern Filling

The *patternFilling* Procedure tries to fill an empty rectangle with strips by using the remaining items. The strips can be horizontal or vertical, generating a preCut when the direction is different to the pattern direction (black dashed line in Figure 2.1). To minimize the waste, we solve a 2TKP using the empty rectangle of the patterns, the items area as profits and maximizing the sum of the area of the items in the rectangle.

To solve the 2TKP, we use the Greedy Randomized Adaptive Search Procedure (GRASP) and the Path Relinking Procedure developed by Alvarez-Valdes et al. [1], see Section 1.3.

We modify the procedures, making it possible to stack in the strips more than one item of the same length (width if the item is rotated) when the strip dimension admits it, dark items in Figure 2.1. In this case, the third-stage cut is used to separate the items and each third-stage cut is counted in the objective function. Also, in order to place more items in the strip, the feasible orientation of the items with minimum length is kept, i.e., if the item

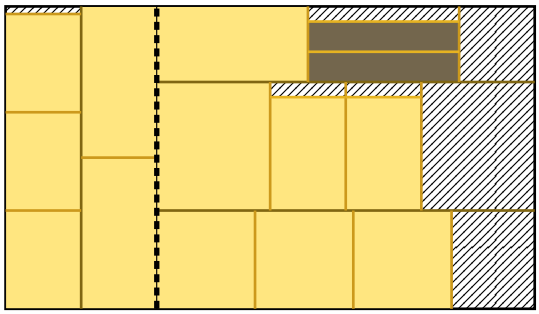


Figure 2.1: PreCut and stack items

can be rotated and the item length is larger than the item width, the item is rotated when its new dimensions are feasible for the rectangle dimensions. We obtain five solutions with the GRASP_Strip and keep the best three, also for the GRASP_Pieces. The six solutions set is ordered by non increasing profit and the solutions with the same profit are discarded. The solution with largest profit is memorized as the best solution. Each solution is used in the Path Relinking Procedure as a guiding solution for all the initiating solutions after it in the ordered set. If a better solution is found, the best solution is updated. Finally, the best solution is returned and is added to the original pattern.

2.4 Cycle Reduction

As mentioned before, the previous procedures are solved by bounding the item demands so as to increase the repetition of the resulting patterns and reduce the number of cycles.

At the beginning, the maximum number of boards that the machine is able to cut simultaneously is considered, $t = \kappa$. The procedures are invoked with a fictitious demand $\tilde{d}_i = \lfloor d_i/t \rfloor$ $(i = 1, \ldots, m)$, the patterns obtained are fixed to the maximum repetition that does not exceed the demand of the items and the items are removed accordingly. As a result, the obtained pattern j has a repetition R_j at least equal to the current t but no larger than $min_{i \in I_j}\{\lfloor \frac{d_i}{p_i^j} \rfloor\}$, where p_i^j is the number of items of item class i in pattern j. The procedures are repeated for $t = t - 1$, until $t = 1$ and the original or remaining demand of the items is considered.

2.5 Remaining Items Insertion

The *remainingItemsInsertion* Procedure tries to insert items with positive demands into a pattern in the solution in order to reduce the waste of the pattern and complete the demand of the items. In the following, we describe the insertion of an item in a horizontal strip; the vertical case is analogous.

Whenever possible, we try to insert the item using the least length of the remaining strip length; thus, if the item length is larger then the item width, the item is inserted rotated, if it fits in the strip. Moreover, if the item can not be inserted in any strip, but the item width is smaller than the residual width of the pattern, a new strip with the item is created and added to the pattern.

2.6 Residual Problem

As anticipated, the *residualProblem* Procedure is called when there are still remaining items after trying to insert items into the patterns in the solution, *remainingItemsInsertion* Procedure. These items come from unselected strips or unpacked patterns. It is necessary to solve a further 2DCSP to place these items, but this time the input size is significantly smaller. We solve the small 2DCSP using three heuristics, and choose the set of patterns with minimum waste; we then merge these patterns with the previous patterns to obtain a global solution.

Iterative Heuristic A The first heuristic solves the 2TKP iteratively until all the items are placed. The area of the items is considered as profit in order to minimize the waste. For each board class and direction, the internal procedure generates a pattern. The pattern with minimum waste, among all the board classes, is memorized and fixed to the maximum repetition that does not exceed the items availability; the items in the pattern are removed from the available set accordingly.

The internal procedure generates four patterns for a given board class (L, W) and direction, and the pattern with minimum waste is returned. We present the version for the horizontal direction; the version with vertical strips is obtained similarly by swapping the board length and width.

The internal procedure starts removing the infeasible items for the board class and appropriately setting the orientation of the available item classes. When the items can be rotated, the items are oriented with its narrower dimension to reduce the use of capacity, i.e., if the item length is larger than the item width, the item is rotated if it fits in the board. For each procedure, we consider a local availability of the item classes.

The procedure for the first pattern, based on the BKP, starts by determining all the feasible strip widths according to the item classes. While there are unplaced items, the strips are obtained by solving a BKP for each strip width with capacity L using the items of the same width (length for rotated items). The strip with maximum profit (the sum of the area of the items in the strip) is selected and its repetition is fixed to the maximum number that does not generate overproduction; the items in the selected strip are removed accordingly. Once all the items are placed in a strip, a further BKP is solved with capacity W using the strips generated before considering the strip width as weight, to obtain the pattern. The second pattern is obtained by repeating the previous procedure, but this time the third stage cut is allowed, i.e., the items in the strip can have a width (length for rotated items) smaller than or equal to the strip width, but at least one item must have the same width (length for rotated items) as the strip width.

The procedure for the third pattern starts by ordering the items in non increasing width (length for the rotated items). The strips are created by solving the BPP with capacity L using the item length (width for the rotated items) as weight and banning the third stage cuts, i.e., all the items in the strip have the same width. To obtain the pattern, a BKP is solved with capacity W and using the strips generated before with its width as weight and the sum of the area of the items in the strip as profit. The fourth pattern is obtained by repeating the procedure for the third pattern but allowing the three-stage cuts when solving the BPP, i.e. the items in the strip can have different widths (length for rotated items) and the first item inserted in the strip defines the strip width.

The BKPs are solved using the Greedy Algorithm [18], see Section 1.3.

The BPP is solved using the Best-Fit Decreasing Heuristic (BFD) [25] with capacity L and the items as objects, using the item length (width for rotated items) as weight. We keep the non increasing width order of the items and for those having the same width, we order them by their length. The first item inserted in a strip determines the strip width. As the items are ordered in non increasing width, they can be inserted in any strip created before if the capacity is enough.

As mentioned, the internal procedure selects and returns the pattern with minimum waste between these four patterns. The procedures are then repeated for the other directions and boards. Between all these patterns (two for each board class) the one with minimum waste is fixed to its maximum repetition that does not generate overproduction and the items are removed accordingly. The whole process is repeated until the demand of all the items is satisfied.

Iterative Heuristic B The second heuristic solves the 2DCSP for each direction and the solution with minimum waste is returned. Inspired in the third patterns obtained during the internal procedure of Heuristic A, a solution is created for a given direction by solving two nested BPP.

The internal procedure starts by setting the item classes appropriate to a feasible orientation that improves the strip usage, i.e., if the item length is larger than the item width, the item is rotated if it fits in at least one board.

Once the item classes are set, the procedure orders the item classes according to non increasing width (length for the rotated items), then the first BPP is solved, using BFD (see Section 1.3), considering the length of the boards as capacity of different bins and the item class length (width for rotated items) as weight, obtaining a set of feasible strips for different board classes. The first item class inserted in a strip, determines the strip width. The strip length is determined by the board length.

To create the patterns a further BPP is solved, using BFD, considering the width of the boards as capacity of the bins and the strip width as weight. The strips can be placed in any board class with length not smaller than the strip length if it minimizes the residual capacity of the bin. Finally, the procedure checks the repetition of the patterns and calculates their trim loss.

Iterative Heuristic C The last heuristic considers each item independently, i.e. having demand equal to 1; and calls the *stripGenerator* Procedure and *patternGenerator* Procedure. If necessary, it calls Iterative Heuristic A at the end.

The solution of the heuristics for the small 2DCSP having minimum waste is selected and merged with the set of patterns to obtain the global solution of the problem.

2.7 Computational Results

The aim of this section is to select the group of parameters that improves the performance of the algorithms described in the previous sections. All the algorithms were implemented in C and run on one core of an Intel Core 2 Quad 2.50 GHz, with 7.7 GB shared memory, under

Linux Ubuntu 9.04 operating system. Below we present the set of instances used to test the algorithms and the computational results for different sets of parameters.

2.7.1 Instances

We create 30 instances considering the dimensions and characteristics of real instances representative of real problems from the wood industry. The item classes length and width are in the range [108, 3512] and [65, 1606] respectively and the board classes length and width are in the range [1844, 5300] and [1020, 2200] respectively. All the items can be rotated and κ is set to 6.

Each instance is defined by the total number of items (t), the number of item classes (m) and the number of board classes (b). For the total number of items t the values are: 25, 50, 75, 100, 150, 200, 300, 400, 600 and 800. For the number of item classes m the values are: 10, 40 and 80. The number of board classes b is set to two for each combination; for the larger combinations we also create instances with three board classes (40x600, 40x800, 80x600 and 80x800). Unfeasible combinations are not considered, for example 25 total number of items and 40 item classes. The name of the instance defines its features: `instance`mxt or 3`instance`mxt for the instances with 3 board classes.

Below we describe how the instances are generated using the dimensions of the real instances. In order to have item classes with related dimensions, we create an adjacent list with the items that are not in the instance and have at least one dimension in common with an item class in the instance. The first new item class is selected randomly among the real item dimensions and the adjacent list is initialized with real items having a common dimension to the selected item class. With 0.3 of probability, or if the adjacent list is empty, the following new item class is selected randomly among the real items that have not been selected yet. The new item class is removed from the adjacent list, when it corresponds, and the real items having a common dimension to the new class are added to the adjacent list. With 0.7 of probability, the following item class is selected randomly among the items in the adjacent list. As before, the new item class is removed from the adjacent list and the real items adjacent to the new one are added. The cycle is repeated until all the required item classes are created. The demand of the item classes is given by the random distribution of the total number of items in the item classes. In detail, first we assign one item to each item class. Then, until the total number of items is reached, an item class is randomly selected and its demand is increased by one item. The dimensions of the board class are selected randomly among the real boards and it is verified that all the item classes fix into at least one of them.

Table 2.1 describes the area of the instances. The first column is the instance name, the second column is the total area of the items, the third column is the area of the smaller board class and the last column is the area of the largest board class. All the areas are expressed in square millimeters. Note that, on average, the area of the items is 56.36 and 38.72 times larger than the smallest and largest board, respectively; this means that, on average, at least 39.2 boards are needed with a minimum of 3 boards for `instance`10x25 and a maximum of 106 boards for `instance`40x800.

Table 2.1: Instance Area

Instance Name	Items Total Area	Board Class Min Area	Board Class Max Area
instance10x25	16318798.00	5264900.00	7585000.00
instance10x50	32611165.00	2972897.28	5692960.00
instance10x75	49044502.00	2972897.28	8487000.00
instance10x100	67039086.00	4407500.00	6232000.00
instance10x150	100132569.00	4407500.00	5264900.00
instance10x200	133307896.00	3721000.00	5692960.00
instance10x300	201648432.00	4732381.00	6232000.00
instance10x400	268079317.00	3672000.00	4262074.60
instance10x600	401779496.00	3721000.00	6282450.00
instance10x800	535119701.00	4407500.00	7215000.00
instance40x50	30515946.00	2852668.00	5692960.00
instance40x75	48686155.00	5692960.00	6232000.00
instance40x100	62687851.00	4262074.60	8487000.00
instance40x150	91291288.00	5264900.00	5299200.00
instance40x200	122173479.00	4554000.00	6282450.00
instance40x300	193671225.00	4732381.00	4761771.02
instance40x400	246017180.00	3066445.48	3672000.00
instance40x600	373084474.00	4732381.00	5299200.00
instance40x800	502610017.00	3672000.00	4761771.02
instance80x100	63821867.90	3791768.20	8487000.00
instance80x150	92195705.90	4554000.00	7215000.00
instance80x200	122551279.80	4554000.00	6232000.00
instance80x300	184769095.40	4761771.02	5264900.00
instance80x400	250513251.40	5299200.00	6251400.00
instance80x600	381326953.20	7585000.00	8487000.00
instance80x800	512594067.70	5692960.00	8487000.00
3instance40x600	373084474.00	3791768.20	5299200.00
3instance40x800	502610017.00	3672000.00	5299200.00
3instance80x600	381326953.20	2852668.00	8487000.00
3instance80x800	512594067.70	3066445.48	5692960.00

2.7.2 Constructive Heuristic Parameters

The parameters of the heuristic algorithms are the power of the item classes α, the strip acceptance threshold β and the pattern acceptance threshold γ. As mentioned before, we presumed that a high value of α privileges the insertion of large items in the first phases of the algorithm and, therefore, leaves a small area of remaining items for the final 2DCSP. In relation to the threshold parameters, we expect that smaller values lead to smaller areas of used boards and larger computation times; as well as higher values lead to larger areas of used boards but shorter computation times.

With the objective of studying the behavior of the algorithm to changes in the parameters and to find the set of parameters that improves the performance of the algorithm in terms of the Equation 1.1, we study three values for each parameter: $\alpha \in \{1.0, 1.1, 1.2\}$, $\beta \in \{0.05, 0.10, 0.15\}$ and $\gamma \in \{0.05, 0.10, 0.15\}$.

Table 2.2 shows the average of the solutions of the experiments. The first three columns show the value of each parameter α, β and γ respectively. Column 4 is the average of the computational time $\overline{T}$, in seconds, to solve each instance. Column 5 is the average of the percentage area, in square meters, of the items placed with the *residualProblem* procedure, $\overline{items}$. Column 6 is the average of the board used area $\overline{A}$ in square meters. Column 7 is the average of the number of cycles $\overline{C}$. Column 8 is the average of the number of third stage cuts $\overline{Z}$. The last column is the average of Equation 1.1, $\overline{index}$ of each solution.

We observe that when α increases and the other parameters remain invariable, on average, the area of the items placed with the *residualProblem* Procedure decrease for the parameters $\{\beta = 0.05, \gamma = 0.15\}$, $\{\beta = 0.10, \gamma = 0.05\}$, $\{\beta = 0.10, \gamma = 0.10\}$ and $\{\beta = 0.10, \gamma = 0.15\}$ as we expected; for the parameters $\{\beta = 0.15, \gamma = 0.05\}$ the algorithm presents the opposite behavior. Note that the computational time is directly related to the percentage area of the items placed with the *residualProblem* Procedure.

We observe that when β increases and the other parameters remain invariable, on average, the area of used boards increases for the parameters $\{\alpha = 1.0, \gamma = 0.10\}$ and $\{\alpha = 1.1, \gamma = 0.15\}$; instead, for the parameters $\{\alpha = 1.0, \gamma = 0.05\}$, $\{\alpha = 1.2, \gamma = 0.05\}$, $\{\alpha = 1.2, \gamma = 0.10\}$ and $\{\alpha = 1.2, \gamma = 0.15\}$ the area of used boards decreases. In relation to the computational time, when β increases and the other parameters remain fixed, on average, the computational time decreases for the parameters $\{\alpha = 1.0, \gamma = 0.05\}$, $\{\alpha = 1.0, \gamma = 0.10\}$, $\{\alpha = 1.0, \gamma = 0.15\}$, $\{\alpha = 1.1, \gamma = 0.10\}$, $\{\alpha = 1.1, \gamma = 0.15\}$, $\{\alpha = 1.2, \gamma = 0.10\}$ and $\{\alpha = 1.2, \gamma = 0.15\}$. Surprisingly, we observe that, on average, the number of third stage cuts decreases whenever β increases and the other parameters remain invariable; this fact is explained because a larger β increases the number of strips selected during the *stripGenerator* Procedure, where the third stage cuts are banned.

We observe that when γ increases and the other parameters remain invariable, on average, the area of used boards behaves contrary to expectations, i.e., the area of used boards decreases for the parameters $\{\alpha = 1.0, \beta = 0.15\}$, $\{\alpha = 1.1, \beta = 0.05\}$, $\{\alpha = 1.1, \beta = 0.10\}$, $\{\alpha = 1.1, \beta = 0.15\}$, $\{\alpha = 1.2, \beta = 0.05\}$, $\{\alpha = 1.2, \beta = 0.10\}$ and $\{\alpha = 1.2, \beta = 0.15\}$. Instead, on average, the computational time decreases when γ increases and the other parameters remain invariable as we expected. Unexpectedly, we observe that when γ increases and the other parameters remain fixed, on average, the area of the items placed with the *residualProblem* Procedure decreases for the parameters $\{\alpha = 1.0, \beta = 0.05\}$, $\{\alpha = 1.0, \beta = 0.10\}$, $\{\alpha = 1.0, \beta = 0.15\}$, $\{\alpha = 1.1, \beta = 0.10\}$, $\{\alpha = 1.1, \beta = 0.15\}$, $\{\alpha = 1.2, \beta = 0.05\}$, $\{\alpha = 1.2, \beta =$

Table 2.2: Constructive Heuristic Parameters

α	β	γ	$\overline{T}$	$\overline{items}$	$\overline{A}$	$\overline{C}$	$\overline{Z}$	$\overline{index}$
1.0	0.05	0.05	9.76	75.80	273.20	25.63	28.00	986.82
1.0	0.05	0.10	7.79	66.97	274.26	29.33	28.23	1000.39
1.0	0.05	0.15	7.17	65.14	273.52	28.87	28.90	996.81
1.0	0.10	0.05	8.77	74.56	273.11	26.57	25.77	988.34
1.0	0.10	0.10	6.49	65.56	275.87	29.17	23.47	1003.88
1.0	0.10	0.15	6.01	59.59	274.44	28.73	24.37	998.20
1.0	0.15	0.05	8.24	72.61	272.28	28.47	22.97	989.83
1.0	0.15	0.10	3.47	54.58	276.09	26.70	19.90	997.04
1.0	0.15	0.15	2.79	44.76	276.17	27.57	18.97	999.37
1.1	0.05	0.05	10.29	56.24	272.71	28.40	28.03	992.68
1.1	0.05	0.10	6.95	66.92	274.64	28.60	24.10	998.46
1.1	0.05	0.15	5.92	62.69	276.33	28.87	24.27	1004.89
1.1	0.10	0.05	8.18	74.28	274.51	29.03	23.67	998.99
1.1	0.10	0.10	5.18	64.29	274.91	27.63	23.03	996.43
1.1	0.10	0.15	4.19	58.15	275.57	28.30	23.30	1000.62
1.1	0.15	0.05	9.68	73.93	273.27	26.43	23.53	987.88
1.1	0.15	0.10	3.04	52.77	274.41	28.10	19.60	994.98
1.1	0.15	0.15	2.03	43.85	276.65	28.30	19.17	1002.92
1.2	0.05	0.05	10.99	76.65	273.98	27.67	28.80	995.15
1.2	0.05	0.10	7.02	67.17	275.67	28.67	26.23	1002.65
1.2	0.05	0.15	5.72	62.23	277.84	26.70	25.67	1004.57
1.2	0.10	0.05	8.59	73.63	273.40	28.67	25.30	994.80
1.2	0.10	0.10	5.34	63.77	275.32	27.67	23.67	998.14
1.2	0.10	0.15	4.16	57.92	276.98	28.00	21.57	1003.94
1.2	0.15	0.05	9.50	74.00	272.32	28.00	24.43	989.16
1.2	0.15	0.10	3.25	53.82	273.78	28.23	20.33	993.52
1.2	0.15	0.15	2.14	44.39	276.48	29.97	19.50	1006.83

0.10} and $\{\alpha = 1.2, \beta = 0.15\}$; this fact is explained because a larger γ means a larger waste in the patterns selected during the first phase of the algorithm, therefore, the number of patterns accepted and consequently the number of items placed increase.

In the next experiments, we choose the configuration of minimum index $\{\alpha = 1.0, \beta = 0.05, \gamma = 0.05\}$ for the Constructive Heuristic which has a computation time of 9.76 seconds on average. Table 2.3 presents the results of this configuration for all the instances. The first column is the name of the instance. Column 2 is the computational time in seconds, (T). Column 3 is the area of the used boards, (A). Column 4 is the number of cycles, (C). Column 5 is the number of third-stage cuts, (Z). The last column is the value of Equation 1.1, $(index)$. The last row reports average values.

Table 2.3: Constructive Heuristic: $\alpha = 1.0, \beta = 0.05, \gamma = 0.05$

instance	T	A	C	Z	*index*
instance10x25	0.06	26.32	4	0	97.67
instance10x50	0.15	43.58	10	3	172.57
instance10x75	0.15	65.79	11	3	249.23
instance10x100	0.20	89.83	14	0	336.33
instance10x150	0.70	133.46	8	1	465.63
instance10x200	2.03	179.95	12	1	631.30
instance10x300	4.10	245.25	12	10	852.00
instance10x400	11.29	322.67	20	0	1128.33
instance10x600	27.46	515.25	24	2	1781.60
instance10x800	18.40	815.30	24	0	2781.00
instance40x50	0.20	42.72	8	5	164.83
instance40x75	0.58	60.16	10	8	229.07
instance40x100	0.58	76.53	13	11	292.97
instance40x150	1.18	105.57	19	31	410.97
instance40x200	1.11	138.68	21	25	525.50
instance40x300	2.92	241.68	23	12	869.93
instance40x400	8.98	329.99	30	23	1185.90
instance40x600	6.56	419.63	47	38	1534.73
instance40x800	61.55	573.50	39	36	2025.80
instance80x100	1.08	70.78	10	44	274.87
instance80x150	1.50	101.37	17	46	396.13
instance80x200	2.14	136.62	23	49	531.03
instance80x300	5.85	206.03	40	63	811.57
instance80x400	5.83	275.72	44	49	1051.03
instance80x600	12.75	412.46	45	92	1521.60
instance80x800	16.76	555.71	64	106	2054.47
3instance40x600	8.03	446.09	35	12	1582.93
3instance40x800	34.20	567.50	37	27	1997.77
3instance80x600	16.50	410.35	50	81	1524.63
3instance80x800	40.01	587.65	55	62	2123.27
average	9.76	273.20	25.63	28.00	986.82

In this chapter, we have presented Constructive Heuristic Algorithms to solve the 2DCSP considering the requirements of the wood industry. The algorithms are based on the construction of strips that are subsequently grouped into patterns. We study the parameters of the algorithms, and set them to the value that improves the performance of the algorithm. We are interested in obtaining a fast solution with these algorithm, and use it as the initial solution in more complex procedures described in the next Chapters.

Chapter 3

Column Generation

Given an instance for the Cutting Stock Problem, the classic approach to find a good solution is based on the model of Gilmore and Gomory [10] [12], (given by Equations (3.1)–(3.3)) that implicitly considers all the feasible cutting patterns $\mathbb{P}$, which are exponentially many. Each pattern $j \in \mathbb{P}$ $(j = 1, \ldots, \eta)$ is characterized by a cost E_j and a vector $P_j = [p_j^1 \ldots p_j^i \ldots p_j^m] \in \mathbb{Z}^m$ representing the items in the pattern, where p_j^i is the number of items of class i $(i = 1, \ldots, m)$ that are in the pattern j. A variable x_j (column in the model) is associated with each pattern and its value indicates the number of times that the pattern is in the solution.

$$O.F. := \quad min \sum_{j=1}^{\eta} E_j x_j \tag{3.1}$$

$$subject \quad to \quad \sum_{j=1}^{\eta} p_j^i x_j \geq d_i \quad i = 1, \ldots, m \tag{3.2}$$

$$x_j \in \mathbb{Z}^+, \qquad j = 1, \ldots, \eta \tag{3.3}$$

Relaxing the integrality constraint, equation (3.3), allows us to find the optimal solution, possibly fractional, without considering all the feasible patterns by using the column generation approach.

In the column generation approach, the relaxed model that considers a small set of patterns is called Restricted Model, and the patterns needed to find the optimal solution of the continuous relaxation must be determined and added. These patterns, needed to improve the solution, have a negative reduced cost, i.e., the profit of the pattern is larger than the pattern cost, where the profit of the pattern depends on the value of the current optimal dual solution π_i associated with each item class i $(i = 1, \ldots, m)$. Each time a new pattern is added to the Restricted Model, the profit of the patterns changes because of the new optimal dual solution. If the pattern with the maximum profit has a positive reduced cost, the current solution of the Restricted Model corresponds to the optimal solution of the continuous relaxation of the model (3.1)–(3.3). Otherwise, not all the patterns to solve the relaxed problem are known. To find these patterns a pricing subproblem is solved in order to identify the feasible pattern with maximum profit.

In particular, for the 2DCSP in the wood industry, the pricing subproblem corresponds to the 2TKP, allowing a third stage for trimming or to separate smaller items and using the dual variables, associated with each item class, as profits.

Note that model (3.1)–(3.3) does not consider the machine cycles. In Appendix A we propose two ILP models that explicitly consider the cycles, but these models are difficult to

solve, therefore we use the model of Gilmore and Gomory, considering the usage minimization and machine productivity in the objective function. In detail, each pattern is assigned a cost that represents the contribution of each pattern to the performance index (Equation 1.1). Unlike the area, the cycles as well as the third-stage cuts do not depend linearly on the value of the x variable, i.e., the contribution to the cost changes according to κ; therefore we appropriately estimate the number of cycles for each pattern and consider the contribution of each pattern in the cycle in the objective function. This means that for each pattern the cost is given by Equation 3.4, where $w_1 = \frac{10}{3}$, $w_2 = \frac{8}{3}$ and $w_3 = 0.3$ are the weights of the performance index (Equation 1.1), A_j^k is the area of the board $B_j = k$, Z_j is the number of third-stage cuts and $\tilde{C}_j$ is an approximation of the number of cycles for that pattern.

$$E_j = w_1 * A_j^k + \frac{w_2 + w_3 * Z_j}{\tilde{C}_j} \tag{3.4}$$

To estimate the number of cycles, we consider the items in the pattern and set $\tilde{C}_j$ to the maximum number of patterns that does not exceed the item classes demand (Equation 3.5), consequently the patterns that are more appropriate for high repetition have a larger $\tilde{C}_j$ and, therefore, a smaller value in the second part of Equation 3.4.

$$\tilde{C}_j = min_{i \in I_j} \left\{ \left\lfloor \frac{d_i}{p_j^i} \right\rfloor \right\} \tag{3.5}$$

We initialize the Restricted Model with a feasible solution found by the Constructive Heuristic (Chapter 2). Using the optimal dual solution of the Restricted Model π_i ($i = 1, \ldots, m$), the pricing subproblem generates the patterns. All the patterns with negative reduced cost are candidates to improve the objective function. Since each pattern is associated with a board, we need to solve the pricing subproblem for each board class.

As solving the 2TKP to optimality may be very time consuming and as the solution is only optimal in the marginal sense [5], we solve the subproblem quickly by using a nested BKP, similar to the procedure presented in [12], and a Modified GRASP [1] procedure.

Before adding a candidate column, we check that it is not in the restricted master problem by using a hashing function that considers the area of the items in the column. The generation process is stopped when no column is found for any board class or when the time limit for the column generation is reached. The solution of the relaxed model is, in general, fractional; we propose a heuristic procedure based on this solution to find an integer solution for the problem.

In order to find an integer solution, a rounding procedure followed by the solution of the remaining problem is usually used. In the context of column generation, the rounding heuristic destroys the structure of the subproblem and is doomed to fail because the columns in the Restricted Master Problem typically do not contain an optimal integer solution [25]. We propose an approach guided by the continuous relaxation of the model to obtain an integer solution

In the following we explain the procedure to solve the pricing subproblem and the heuristics to find an integer solution.

3.1 Nested BKP

The nested BKP solves two BKPs for a given board class and direction (horizontal or vertical). The inner BKP creates the strips while the second join the strips to make a pattern. We present the version of the procedure for a given board class of dimensions (L, W) and the horizontal direction. The vertical direction is obtained by swapping the board length and width and the items dimensions.

Before calling the procedure, a set of all the feasible strip widths $\bar{W} = \{\bar{w}_s : \exists w_i = \bar{w}_s \wedge w_i \leq W, i = 1, \ldots, m\}$, for the given board class and direction is calculated according to the item classes. Moreover, the set of feasible items for these widths $I_{\bar{w}_s}$ is calculated, i.e., the item classes with item width no larger than $\bar{w}_s$ and length no larger than L, or items classes that can be rotated and have widths no larger than L and lengths no larger than $\bar{w}_s$.

The inner BKP is solved for each $\bar{w}_s$ considering L as capacity and the set of feasible item classes $I_{\bar{w}_s}$ as objects. The optimal dual solution π_i associated with each item class i is used as profit and the item class length, or item class width for the rotated items, is used as the weight. The solution of the inner BKP is a feasible strip for the board with maximum profit. Note that the items in the strip do not exceed the item class demand. The profit of each strip is the value of objective function of the BKP, i.e., the sum of the profit of the items in the strip.

In the outer BKP, the aim is to make a pattern with maximum profit using the previous strips. The capacity is the board class width W, the object weights are the strip widths and the object profits are the profit of the strips. The solution is a pattern with feasible dimensions. The BKPs are solved using the Greedy Algorithm presented in Section 1.3.

Note that the pattern solution might be infeasible because the items in the pattern might exceed the item class demand. We replace the extra items with available items and update the profit of the pattern. In order to improve the pattern profit, a procedure tries to insert some available items in the pattern allowing third-stage cuts.

3.2 Modified GRASP

The GRASP [1] as explained in Section 1.3 heuristically solves the 2TKP, i.e., it returns a pattern with a high profit. Internally, it uses the GRASP_Strip and GRASP_Pieces to create patterns and the Path Relinking Procedure to generate new solutions.

As explained in Section 2.3, we modified the procedures in order to allow third-stage cuts for trimming or to separate small items when the strip width allows it. The orientation of the items is also selected in order to place the maximum number of them.

Moreover, in the procedure used during the column generation, we add all the intermediate patterns generated by the GRASP_Strip and GRASP_Pieces having negative reduced cost to the set of columns of the Restricted Model.

The GRASP is also invoked a second time, banning the third-stage cuts.

3.3 Diving Heuristic

We propose a Diving Heuristic Algorithm guided by the continuous relaxation of the model (3.1)–(3.3). The algorithm begins with the fractional solution found with the column gener-

ation approach. New columns are added to the Restricted Master Problem at each node and the integer solution is obtained by fixing variable bounds.

Starting from the relaxed model and the original vector of demands, each node is examined in three main steps: obtain the relaxed solution, solve the remaining problem and select the branching variable.

For a general node, in the first step, we solve the master problem and obtain the primal and dual solutions. If the primal solution is integer, the procedure stops returning the best memorized solution. Otherwise we continue with the second step in which we use the constructive heuristic, Chapter 2, with the dual solution as profit of the item classes and find a solution for the remaining problem, i.e., the items that are not contained in the bounded variables. Joining this partial solution with the bounded variables of the model we obtain a new solution for the problem. In the last step, we consider the values of the primal solution different from zero. We select a variable for branching and fix its value appropriately. The demand of the items in the variable is updated accordingly.

We propose three strategies to select the variable and fix its value:

First Branching Select the variable with the largest fractional part. The variable value is fixed to its value rounded up, if the column generation of the master problem has finished. Otherwise, the variable value is fixed to the largest integer that does not exceed the current demand of the items in the pattern.

Second Branching Select the largest fractional variable. The variable value is fixed to its value rounded up, if the column generation of the master problem has finished. Otherwise, the variable value is fixed to the largest integer that does not exceed the current demand of the items in the pattern.

We improve the Second Branching by bounding the variables with larger value. In detail, for all the variables with a value larger than or equal to κ, we fix the lower bound of the variable to the largest multiple of κ that does not exceed the variable value and update the demand according to the lower bound. We called this strategy Second Branching + Big Variables.

Third Branching Select the variable with largest fractional part and fix it to the variable value rounded up, if the current demand of the items in the pattern is not exceeded or if there is no column generation. Otherwise, select the variable with the smallest fractional part and fix it to the variable value rounded down.

We improve the Third Branching by fixing the value of the integer variables and bounding the variables with a value larger than or equal to κ to the largest multiple of κ that does not exceed the variable value. We update the vector of demand accordingly. We called this strategy Third Branching + Integer Variables.

We also propose the version in which the value of the variable with the largest fractional part is not fixed but its lower bound is set to the variable value rounded up. We called this strategy Third Branching Bounding and it can be combined with the Integer Variables strategy.

Note that fixing the variable to a value smaller than its value would regenerate the variable in the next iteration of column generation and thus the algorithm could go in a loop. To

prevent the regeneration of a pattern, whenever a bound is fixed, the availability of the items in the pattern is updated and the heuristics used to solve the pricing subproblem considers only the current availability of the items. In addition, we verify the regeneration with a hashing function that considers the area of the items in the column and does not add the column if it matches any of the other columns in the Restricted Model. Moreover, the second time that the variable is regenerated, the hashing function forbids bounding the variable to its value rounded down. Furthermore, if the time for the column generation has been reached, this procedure could generate infeasibility, i.e., some items may not be covered by any column. To avoid this, we allow the rounding down of the variables only when the time limit for the column generation has not yet been reached.

3.4 Column Generation and Diving Heuristic Settings

The purpose of this section is to study the Column Generation procedure and the approaches to obtain an integer solution and determine which configuration achieves the best performance of the algorithm. We used the instances presented in Section 2.7.1. All the algorithms were implemented in C and run on one core of an Intel Core 2 Quad 2.50 GHz, with 7.7 GB shared memory, under Linux Ubuntu 9.04 operating system. The linear programs were solved using the callable library GLPK (GNU Linear Programming Kit) Version 4.34.

We initialize the Restricted Model with a feasible solution founded with the Constructive Heuristic Chapter 2, with the parameter states in Section 2.7. For solving the remaining problem at each node, we select the fastest configuration of the Constructive Heuristic, $\{\alpha = 1.1, \beta = 0.15, \gamma = 0.15\}$.

The first approach to obtain an integer solution consists in solving the integer model (3.1)–(3.3) using the ILP solver of GLPK with the columns obtained to solve the continuous relaxation of the model during the Column Generation procedure. In order to respect the 10 minutes of the wood industry maximum computational time, we set the Column Generation time limit to 500 seconds and the ILP solver time limit to 100 seconds.

Table 3.1 shows the average of the results of the experiments. We present the different strategies to obtain an integer solution in Column 1. Column 2 is the average of the computational time $\overline{T}$, in seconds, to solve each instance. Column 3 is the average of the board used area $\overline{A}$ in square meters. Column 4 is the average of the number of cycles $\overline{C}$. Column 5 is the average number of third stage cuts $\overline{Z}$. The last column is the average of Equation 1.1, $\overline{index}$ of each solution.

Note that all the Diving strategies have an average index smaller than the average index obtained with the ILP Solver.

The Second Branching achieves the best performance of the Diving Heuristic, improving the starting solution given by the Constructive Heuristic. On average, the area is improved in 9.10% and the cycles are improved in 18.49%, instead the third-stage cuts are increased in 34.17%; however, the index is improved in 9.51% with a 93.65% increasing in the computational time. Table 3.2 presents the results of this configuration for all the instances. The first column is the name of the instance. Column 2 is the computational time to solve the continuous relaxation of the model (T_0). Column 3 is the value of the continuous relaxation of the model (LB). Column 4 is the number of columns used to solve the continuous relaxation of the model ($NCol$). Column 5 is the computational time in seconds, (T). Column 6 is the

area of the used boards, (A). Column 7 is the number of cycles, (C). Column 8 is the number of third-stage cuts, (Z). The last column is the value of Equation 1.1, ($index$) The last row reports average values.

We observe that the time limit is reached only for the largest instance. In the following chapter we present two techniques of local search, with the aim of improving these solutions, using as time limit the remaining time to complete the 600 seconds computational time limit of the wood industry.

In this chapter, we have used the Column Generation approach to improve the solution obtained with the Constructive Heuristic proposed in Chapter 2. The model used in the column generation considers the main features of the wood industry and the required objective of stock usage and machine productivity. Moreover, several strategies to find an integer solution were studied. As the computational time limit of wood industry is not reached for all the instances, in the next chapter, we will use local search techniques until the time limit is reached, to further improve the solutions.

Table 3.1: Column Generation and Diving Heuristic Settings

Integer Strategy	$\overline{T}$	$\overline{A}$	$\overline{C}$	$\overline{Z}$	$\overline{index}$
ILP Solver	304.79	256.58	28.07	39.43	941.37
First Branching	329.66	254.50	24.73	34.53	924.18
Second Branching	321.43	252.67	22.30	31.90	910.67
Second Branching + Big Variables	319.65	252.63	22.70	31.80	911.57
Third Branching	336.46	253.39	26.97	36.03	926.82
Third Branching + Integer Variables	295.98	255.76	27.20	37.80	935.91
Third Branching + Bounding	335.13	253.77	26.47	34.87	926.40
Third Branching + Bounding + Integer Variables	304.53	252.57	25.83	36.10	921.15

Table 3.2: Column Generation and Diving Heuristic: Second Branching

instance	T_0	LB	$NCol$	T	A	C	Z	$index$
instance10x25	1.46	63.89	160	3.99	23.38	3	4	86.20
instance10x50	3.23	124.48	81	7.61	40.36	6	7	152.10
instance10x75	4.11	181.33	71	9.85	57.73	7	9	213.37
instance10x100	1.88	244.56	52	6.69	74.78	8	11	273.63
instance10x150	1.60	404.22	64	7.24	124.64	10	0	441.67
instance10x200	3.55	529.75	107	12.33	162.64	11	5	572.83
instance10x300	6.01	732.10	122	23.24	221.82	12	15	775.50
instance10x400	5.99	987.55	56	49.21	302.87	19	3	1060.57
instance10x600	6.33	1462.21	51	64.78	440.93	20	17	1527.43
instance10x800	7.88	2323.00	72	57.94	704.26	23	18	2413.73
instance40x50	29.93	108.28	690	121.50	34.16	6	17	134.10
instance40x75	101.47	167.06	927	213.94	53.93	9	10	206.00
instance40x100	272.76	215.59	932	503.90	67.97	10	39	264.37
instance40x150	93.48	315.18	751	290.50	100.17	17	30	387.33
instance40x200	115.22	415.78	734	344.93	130.67	19	24	492.87
instance40x300	155.89	678.72	621	469.65	208.46	23	43	768.23
instance40x400	69.07	919.90	553	484.05	277.98	29	34	1013.53
instance40x600	99.55	1288.71	440	388.40	390.67	27	41	1386.30
instance40x800	250.57	1745.21	665	575.24	533.27	40	47	1897.77
instance80x100	503.59	220.08	1542	505.34	69.88	11	28	269.73
instance80x150	511.77	315.37	1651	516.08	99.47	16	38	385.07
instance80x200	501.94	417.40	1620	512.52	131.59	22	54	512.87
instance80x300	505.50	630.51	1390	530.37	195.27	30	71	751.30
instance80x400	518.28	849.86	1387	548.37	270.43	43	34	1025.87
instance80x600	503.71	1300.78	1088	548.43	404.88	38	64	1469.53
instance80x800	500.10	1742.66	898	599.72	541.74	49	71	1956.97
3instance40x600	222.37	1288.65	755	515.57	393.52	29	38	1399.73
3instance40x800	264.19	1726.56	745	548.65	528.45	37	43	1872.57
3instance80x600	509.78	1296.92	1163	576.47	406.67	40	80	1485.67
3instance80x800	516.38	1755.07	1437	606.36	587.65	55	62	2123.27
average	209.59	815.05	694.17	321.43	252.67	22.30	31.90	910.67

Chapter 4

Post Optimization

In Chapters 2 and 3, we have developed a Constructive Heuristic, a Column Generation Approach and a Diving Heuristic to find a good solution for the 2DCSP in wood industry. Briefly, once the continuous relaxation of the model is solved, through the column generation approach, the Diving Heuristic searches for an integer solution in a search tree where each node is defined by a partial solution obtained after bounding some fractional variables. The search tree is explored in a depth-first fashion until an integer solution is found, because the exhaustive exploration may be very expensive from the computational viewpoint; however, a better solution could exist in the unexplored nodes of the search tree. With this idea, in this chapter, we present two approaches to improve the solution obtained after the Diving Heuristic, considering the Restricted Model and the previously generated patterns.

The approaches are independent from each other and can be used in parallel to obtain different solutions. Both approaches depend on a time limit that is calculated in order to not exceed the industry maximum computational time of 600 seconds. The first approach is a Local Search (Section 4.1) that continues the exploration of the search tree using the Diving Heuristic. The second approach is a Tabu Search (Section 4.3) that uses the columns of the Restricted Master Problem to find a better solution in a set covering fashion.

4.1 Local Search

The Local Search approach (LS) destroys part of a given solution in order to explore new nodes of the search tree. Each iteration is composed of two phases. Given a solution, the first phase selects the patterns to be unpacked, while the second phase explores the search tree with the Diving Heuristic, see Section 3.3.

During the selection phase of the LS, the bounds of the variables (patterns) not selected are fixed, i.e., the lower bound is set to the value of the variable in the solution and the items in those patterns are removed from the available items. The variables selected and the other variables in the Restricted Model have a lower bound equal to zero. The upper bound for all the variables is equal to the total number of items.

With this partial solution, the exploration phase uses the Diving Heuristic Algorithm to find an integer solution guided by the continuous relaxation of the model (3.1)–(3.3). At each node of the Diving Heuristic, the Constructive Heuristic solves the remaining 2DCSP with the current set of available items. The best solution is updated whenever a solution of less

index, Equation 1.1, is found by the Constructive Heuristic or at the bottom of the Diving Heuristic.

When the exploration phase finishes, a new iteration starts with the selection of the variables. The iterations stop when the time limit is reached or when the maximum number of iterations is reached and the best solution is returned.

To select the variables in the solution, we propose the following criteria with different parameters.

LS1 Assign a random probability to each variable with value larger than zero in the best solution and destroy all the variables with a probability below a threshold.

LS2 Calculate the total waste for each variable with value larger than zero in the best solution as the rate of the total unused area over the area of the used boards and destroy the variables with a total waste above a threshold.

LS3 Calculate a similarity rate for each variable with value larger than zero in the best solution. The similarity rate $\mathcal{S}_{i,j}$ for patterns i and j is defined as the number of equal items over the number of total items in the patterns.

$$\mathcal{S}_{i,j} = \frac{\mid I_i \cap I_j \mid}{\mid I_i \cup I_j \mid} \tag{4.1}$$

A perturbation is randomly added to the similarity rate, specifically, with 0.3 of probability the rate is increased by 0.1. Order the variables by non increasing similarity rate and, following this order, destroy a percentage of columns.

LS4 Calculate the total waste of each variable with value larger than zero in the last solution and order the variables by non increasing waste. Following this order, destroy a percentage of variables.

LS5 For each variable with value larger than zero in the last solution calculate its reduced cost using the optimal dual solution found at the end of the column generation. Order the variables by no increasing reduced cost and, following this order, destroy a percentage of variables.

4.2 Local Search Settings

The objective of this section is to study the Local Search performance and find the selection criteria and the parameters that improve the solution found with the diving heuristic. The procedures were implemented in C and run on one core of an Intel Core 2 Quad 2.50 GHz, with 7.7 GB shared memory, under Linux Ubuntu 9.04 operating system. The linear programs were solved using the callable library GLPK (GNU Linear Programming Kit) Version 4.34. Starting from the Restricted Model at the end of the Diving Heuristic, see Chapter 3, the time limit is set to complete the 600 seconds of computational time, with a minimum of 50 seconds. The maximum number of iterations is set to 1000.

The parameters of the selection criteria are set according to preliminary tests to 0.3, 0.5 and 0.7. For LS1, the parameter is the probability threshold. For LS2, the parameter is

the waste threshold. For LS3, LS4 and LS5, the parameter is the percentage of destroyed patterns.

Table 4.1 presents the average values for the selection criteria. Column 1 is the selection criteria name with the parameter value. Column 2 is the average of the total computational time $\overline{T}$, in seconds, to solve each instance. Column 3 is the average of the board used area $\overline{A}$ in square meters. Column 4 is the average of the number of Cycles $\overline{C}$. Column 5 is the average of the number of third stage cuts $\overline{Z}$. The last column is the average of Equation 1.1, $\overline{index}$ of each solution.

Table 4.1: Local Search Selection Criteria

Criteria	$\overline{T}$	$\overline{A}$	$\overline{C}$	$\overline{Z}$	$\overline{index}$
$LS1_{0,3}$	369.00	255.46	23.37	26.37	921.22
$LS1_{0,5}$	601.08	252.08	21.80	30.03	906.81
$LS1_{0,7}$	601.22	251.97	21.57	31.13	906.15
$LS2_{0,3}$	405.68	260.99	22.07	26.40	936.16
$LS2_{0,5}$	561.13	252.31	22.23	31.97	909.31
$LS2_{0,7}$	379.27	260.72	21.73	25.53	934.15
$LS3_{0,3}$	378.88	255.27	23.47	26.63	920.87
$LS3_{0,5}$	601.94	252.06	22.13	30.10	907.65
$LS3_{0,7}$	467.76	254.03	22.47	27.70	914.45
$LS4_{0,3}$	587.98	252.45	22.30	31.50	909.82
$LS4_{0,5}$	412.18	258.72	24.07	27.77	934.27
$LS4_{0,7}$	471.18	256.02	23.17	28.13	922.98
$LS5_{0,3}$	605.93	252.16	22.10	32.70	908.64
$LS5_{0,5}$	411.56	259.51	23.60	26.63	935.39
$LS5_{0,7}$	452.13	258.18	22.60	26.40	928.15

The higher computational time is due to a few outlier run times caused by particularly difficult instances.

The random selection ($LS1$) achieves the best performance for these parameters. The selection based on waste ($LS2$ and $LS4$) achieves the worst performance, although it is very fast. The $LS3_{0.5}$, that considers the similarity rate improves the number of cycles, on average, by 1.4%. We observe that $LS1_{0,7}$ achieves the best performance, improving the area 0.86%, the machine cycles 5.83%, the third-stage cuts 22.85% and the index 1.17% with respect to the Diving Heuristic procedure. Table 4.2 presents the results of this configuration for all the instances. The first column is the name of the instance. Column 1 is the computational time in seconds, (T). Column 2 is the area of the used boards, (A). Column 3 is the number of cycles, (C). Column 4 is the number of third-stage cuts, (Z). The last column is the value of Equation 1.1, ($index$) The last row reports average values. Note that 19 instances improve their index value with respect to the solution found with the Diving Heuristic procedure.

Table 4.2: Local Search: $LS1_{0.7}$

instance	T	A	C	Z	*index*
instance10x25	600.07	20.43	3	5	77.50
instance10x50	600.03	40.36	6	2	150.60
instance10x75	600.02	54.76	6	4	199.20
instance10x100	600.03	74.78	7	3	268.57
instance10x150	600.19	124.64	6	2	431.60
instance10x200	600.06	160.90	11	5	566.83
instance10x300	600.06	221.35	12	10	772.00
instance10x400	599.54	297.43	16	2	1034.27
instance10x600	600.02	439.77	18	20	1519.00
instance10x800	600.14	699.86	20	12	2388.93
instance40x50	601.12	34.16	6	17	134.10
instance40x75	617.53	53.93	9	10	206.00
instance40x100	599.23	67.97	9	24	257.20
instance40x150	600.47	100.17	17	30	387.33
instance40x200	600.05	127.84	18	36	484.80
instance40x300	600.98	208.46	23	43	768.23
instance40x400	600.99	277.98	29	34	1013.53
instance40x600	608.39	390.67	27	41	1386.30
instance40x800	600.33	532.87	40	45	1896.17
instance80x100	600.01	69.88	11	27	269.43
instance80x150	600.23	99.47	16	33	383.57
instance80x200	600.04	131.59	22	54	512.87
instance80x300	600.22	195.27	30	67	750.10
instance80x400	600.75	271.38	38	55	1021.83
instance80x600	600.23	405.78	37	59	1468.37
instance80x800	600.18	541.74	49	71	1956.97
3instance40x600	600.23	392.95	29	38	1397.73
3instance40x800	600.68	528.45	37	43	1872.57
3instance80x600	600.14	406.67	40	80	1485.67
3instance80x800	604.58	587.65	55	62	2123.27
average	601.22	251.97	21.57	31.13	906.15

4.3 Tabu Search

The Tabu Search tackles the problem in a set covering fashion using the columns of the Restricted Master problem, where a covering means that the demand of each item class is satisfied, and looks for a minimal solution that minimize the objective function. A minimal solution is a set of columns such that if one of them is removed the demand of at least one of the rows, item classes, is uncovered.

The algorithm starts with a greedy minimal cover of the rows. At each iteration a column in the solution is randomly selected and its value is reduced according to the number of cycles, i.e., the new value is a multiple of κ. In order to prevent the algorithm from cycling, once a column is randomly selected and removed from the current solution, it is not considered for insertion for a specified number of iterations (tenure), unless it is needed to obtain feasible solutions. Then the algorithm reconstructs a complete cover by iteratively choosing the column j and the number of repetitions R_j such that the ratio $\mathcal{R}_{R_j}$ between the cost and the profit of covered rows is minimized, see Equation 4.2.

$$\mathcal{R}_{R_j} = \frac{\left\lceil \frac{R_j}{\kappa} \right\rceil \cdot f_j + R_j \cdot v_j}{\sum_{i \in I_j} l_i \cdot w_i \cdot min\{R_j \cdot p_j^i, d_i\}} \quad 0 \leq R_j \leq ub_j, \tag{4.2}$$

where: p_j^i is the number of items of class i in the pattern j; $ub_j = max_{i \in I_j}\{\lceil \frac{d_i}{p_j^i} \rceil\}$ is the minimum number that exceeds the demands of all the items in the pattern; $f_j = \frac{10}{3} \times A_j^k$ is the component of the cost associated with the area A_j^k of the used board $B_j = k$; and, $v_j = \frac{8}{3} + 0.3 \times Z_j$ is the component of the cost associated with each cycle.

When the current solution is not improved for a specified number of iterations, a crossover operator finds a new solution. The crossover operator selects two parent solutions from a pool of greedy solutions created at the beginning of the procedure. The parent solutions are different from the current solution and different from each other. The child solution is built by keeping the smallest variable value of the parent solutions; consequently, some item classes are not covered and a complete cover is reconstructed. The parent solution with largest value is replaced by the current solution and the tabu iterations continue with the child solution found.

4.4 Tabu Search Parameters

The parameters of the Tabu Search are the tenure, the number if iterations between two crossovers and the number of solutions for the pool of the crossover. We study three different values for each parameter: tenure $\in \{5, 10, 15\}$, iterations between two crossovers $\in \{5, 15, 35\}$ and number of solutions for the pool of the crossover $\in \{5, 15, 20\}$. All the algorithms where implemented in C and run on one core of an Intel Core 2 Quad 2.50 GHz, with 7.7 GB shared memory, under Linux Ubuntu 9.04 operating system. The linear programs were solved using the callable library GLPK (GNU Linear Programming Kit) Version 4.34. The input of the algorithm is the restricted model after the Diving Heuristic.

Table 4.3 presents the average values for the different parameter sets. Column 1 is the tenure value, *tenure*. Column 2 is the number of iterations between two crossovers, *ItCross*. Column 3 is the number of solutions, *nSol*. Column 4 is the average of the board used area $\overline{A}$ in square meters. Column 5 is the average of the cycles $\overline{C}$. Column 6 is the average of the number of third stage cuts $\overline{Z}$. The last column is the average of Equation 1.1, $\overline{index}$ of each solution.

Table 4.3: Tabu Search Parameters

tenure	*ItCross*	*nSol*	$\overline{A}$	$\overline{C}$	$\overline{Z}$	$\overline{index}$
5	5	5	251.20	21.10	29.53	901.89
5	5	15	250.99	21.07	30.37	901.35
5	5	20	251.01	21.10	30.00	901.40
5	15	5	251.14	21.27	30.40	902.36
5	15	15	250.98	21.30	30.10	901.83
5	15	20	251.40	21.00	29.10	902.20
5	35	5	251.18	21.70	30.73	903.79
5	35	15	251.39	21.37	30.50	903.53
5	35	20	251.33	21.43	30.83	903.57
10	5	5	251.16	20.87	30.03	901.29
10	5	15	251.07	21.17	29.93	901.76
10	5	20	251.04	21.13	29.97	901.58
10	15	5	251.34	21.13	30.10	902.62
10	15	15	251.24	21.07	30.37	902.15
10	15	20	251.13	21.20	29.90	902.04
10	35	5	251.16	21.37	30.87	902.90
10	35	15	251.44	21.23	30.33	903.29
10	35	20	251.21	21.40	30.73	903.05
15	5	5	251.22	21.07	29.50	901.83
15	5	15	251.07	20.90	30.13	901.11
15	5	20	251.07	21.07	30.07	901.53
15	15	5	251.29	21.13	30.30	902.48
15	15	15	251.25	21.23	30.63	902.75
15	15	20	251.22	21.07	29.90	901.95
15	35	5	251.09	21.63	31.30	903.45
15	35	15	251.58	21.10	30.00	903.30
15	35	20	251.55	21.03	29.90	902.93

In general, the solutions for all the parameter sets are similar, therefore, the sensitivity of the algorithm to changes in the parameters seems negligible. However, we observe an increase in the average index value when the number of iterations between two crossovers grows; this fact implies that a diversification of the solution, given by the increased number of crossovers, leads to better solutions. The best performance is obtained with tenure 15, iterations between two crossovers 5 and number of solutions 15; improving the area 0.77%, the cycles 8.00%, the third-stage cuts 18.43% and the index 1.11%, with respect to the solution

of the Diving Heuristic. In Appendix A, the solutions of the best configuration are compared with ILP models that explicitly consider area, cycles and third stage cuts. The results show the efficiency of the Tabu Search to find a good solution.

Table 4.4 presents the results of the best configuration for all the instances. The first column is the name of the instance. Column 2 is the time limit for the Tabu Search in seconds, (T_{TS}). Column 3 is the computational time in seconds, (T). Column 4 is the area of the used boards, (A). Column 5 is the number of cycles, (C). Column 6 is the number of third-stage cuts, (Z). The last column is the value of Equation 1.1, ($index$). The last row reports average values. Note that 14 instances improve their index values with respect to the solution found with the Diving Heuristic procedure.

Table 4.4: Tabu Search: {15,5,15}

instance	T_{TS}	T	A	C	Z	$index$
instance10x25	596.01	600.00	20.43	3	3	76.90
instance10x50	592.39	600.00	40.60	5	5	149.83
instance10x75	590.15	600.00	57.72	6	6	209.80
instance10x100	593.31	600.00	74.78	8	9	273.03
instance10x150	592.76	600.00	124.64	7	2	434.27
instance10x200	587.67	600.00	160.90	10	3	563.57
instance10x300	576.76	600.00	223.32	10	6	772.47
instance10x400	550.79	600.00	296.23	16	3	1030.57
instance10x600	535.22	600.00	440.93	16	13	1515.57
instance10x800	542.06	600.00	699.86	19	6	2384.47
instance40x50	478.50	600.00	34.16	6	17	134.10
instance40x75	386.06	600.00	53.93	9	10	206.00
instance40x100	96.10	600.00	67.97	10	39	264.37
instance40x150	309.50	600.00	100.17	17	30	387.33
instance40x200	255.07	600.00	130.67	19	24	492.87
instance40x300	130.35	600.00	208.46	23	43	768.23
instance40x400	115.95	600.00	277.98	29	34	1013.53
instance40x600	211.60	600.00	390.67	27	41	1386.30
instance40x800	50.00	625.24	535.30	32	48	1883.73
instance80x100	94.66	600.00	69.88	11	28	269.73
instance80x150	83.92	600.00	99.47	16	38	385.07
instance80x200	87.48	600.00	131.59	22	54	512.87
instance80x300	69.63	600.00	195.27	30	71	751.30
instance80x400	51.63	600.00	270.43	43	34	1025.87
instance80x600	51.57	600.00	404.88	38	64	1469.53
instance80x800	50.00	649.72	541.74	49	71	1956.97
3instance40x600	84.43	600.00	396.17	26	31	1398.63
3instance40x800	51.35	600.00	528.93	33	32	1860.60
3instance80x600	50.00	626.47	406.67	40	80	1485.67
3instance80x800	50.00	656.36	548.37	47	59	1970.03
average	283.83	605.26	251.07	20.90	30.13	901.11

Chapter 5

Experiments

The purpose of this chapter is to compare the results obtained by the developed algorithms with those obtained by commercial software packages available for the solution of 2DCSP. The software packages are optimization tools used to obtain efficient patterns considering technological and organizational parameters of production. Efficient patterns are critical to reduce production cost, since small improvements in cutting patterns can result in major savings in raw materials, workload and processing times.

When comparing our results with commercial software packages, however, one must consider that:

- Commercial software packages optimize an objective function similar to the one that we consider, but weights are not given explicitly, thus some software packages may give more importance to material usage, and some others to cycles or third cut minimization;
- Commercial software packages may optimize further features of the cutting patterns that we do not consider in the objective function, e.g., cuts within a pattern may be optimized in order to reduce the processing time.

The software tools descriptions, based on the work of Macedo et al. [16], are presented in the following sections 5.1 to 5.4. In section 5.5, we compare the results of our algorithms and the results of the software packages.

5.1 GNCutter32

GNCutter32 [21], by Optimalon Software with head office in Canada, is a Windows - based software library written on C# for Microsoft .NET Framework. The prices for the license per computer are in the range of 149 USD to 497 USD according to the features and size of the problems that the package can hold and each additional license costs 100 USD. The optimization objective of the package can be set to minimize the waste (default) or minimize the number of different layouts. The package supports guillotine cuts and free cutting (including cuts without the guillotine constraints), rotated or fixed orientations of the items, maximum cut length, blade width consideration, reuse of cuts, safety margins, first cut orientation and sheets of different sizes. The results can be output in CNC code for cutting machines. In the solution, the coordinates of each item are indicated for the corresponding sheet, also the

number of different saw cuts and, optionally, the lines that indicate the cutting-out sequence of each layout can be visualized.

Among these features we selected those admissible to the problem and similar to our objectives:

- UseLayoutMinimization = true :"The GNCutter32 library tries to minimize the number of different layouts during the calculations, which means that several sheets share the same layout. This is very important for wood cutting because the sheets with the same layout can be cut all together and hence the productivity can be increased dramatically."
- SetAutoCutDirection() : "The calculation engine will automatically detect what direction produces the best results."
- GuillotineSheet_Z : "The calculation produces the layout shown in Figure 5.1."

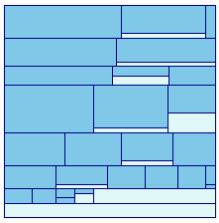


Figure 5.1: GNCutter XYZ-cutting

We run the program on a Pentium 4 2.8 Ghz 496 MB of RAM under Microsoft Windows XP Professional

5.2 CutLogic2D

CutLogic2D [24], by TMachines, s.r.o, with head office in Slovakia, is a package specialized in panel cutting optimization. The prices for the license per computer are in the range of 499 USD to 1499 USD according to the features and size of the problems that the package can hold. The optimization objective of the package can be set to minimize the waste (default) or minimize the number of different layouts. The package supports guillotine cuts and free cutting (including cuts without the guillotine constraints), maximum cut length, blade width consideration, reuse of cuts, safety margins, first cut orientation and sheets of different sizes. The input data can be imported from CSV files and the results can be output in CSV files. It is possible to calculate the total price of a project, based on a variety of inputs of direct or indirect costs. In the solution, the coordinates of each item are indicated for the corresponding sheet.

We used the trial version of CutLogic2D v.3.5.1.322 and set the type of cutting to 3-stage exact as shown in figure 5.2.

We run the program on a Pentium 4 2.8 Ghz 496 MB of RAM under Microsoft Windows XP Professional

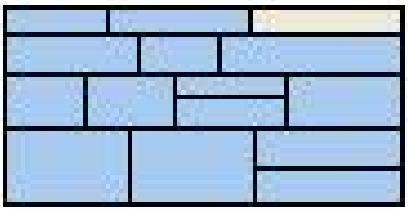


Figure 5.2: CutLogic2D 3-stage exact

5.3 Plus 2D

Plus2D [20], by Nirvana Technologies Private Limited, with head office in India, is a package for generating optimized cutting plans and maximizing panel yields. The price for the license per computer is €1400. The optimization objective of the package can be set to minimize the waste (default) or minimize the cost, the package can also be used to determine the best sheet dimensions for a given set of items. The package supports guillotine cuts, rotated or fixed orientations of the items, maximum cut length, adjustable optimization level, blade width consideration, reuse of cuts, safety margins, first cut orientation, sheets of different sizes and edge banding calculation. The input data can be imported from CSV, TXT or DXF files and the results can be output in DXF, CSV or XML files. The results can be output in CNC code for cutting machines. It is possible to create a material database, keep have track of the inventory, automatically create a document to present to potential customers and calculate the total price of a project, based on a variety of inputs of direct or indirect costs. In the solution, the coordinates of each item are indicated for the corresponding sheet, also the number of different saw cuts and, optionally, the lines that indicate the cutting-out sequence of each layout can be visualized.

The results were obtained with the version 8.82 The system requirements are a 1 GHz 32-bit (x86) processor, at least 2 GB of free Hard Disk space, at least 516 MB of RAM, 256 of colors display 800x600 and Windows XP (or higher) operating system.

5.4 Merick Calc 3000

Merick Calc 3000 [23], by Soft Consult, with head office in Czech Republic. The price for the license per computer is €199 and each additional license cost €90. The optimization objective of the package can be set to minimize the waste (default) or minimize the cost. The package supports guillotine cuts, rotated or fix orientations of the items, maximum cut length, adjustable optimization level, blade width consideration, reuse of cuts, first cut orientation, sheets of different sizes and edge banding calculation. The results can be output in CNC code for cutting machines. It is possible to create a material database, keep track of the inventory and calculate the total price of a project, based on a variety of inputs of direct or indirect costs.

The instances were tested using the version 1.1.27.1269 in a AMD Athlon 64 X2 Dual Core Processor 5200+, 2GB RAM 2.7 GHz under Windows 7 operating system. The accuracy of calculation was set to: 1, 3, 6 and 10.

5.5 Computational Results

Considering that the Local Search approach and the Tabu Search approach are independent and can be invoked in parallel, we present the best solution between them as solution of the Proposed Algorithm. These algorithms were run on one core of an Intel Core 2 Quad 2.50 GHz, with 7.7 GB shared memory, under Linux Ubuntu 9.04 operating system.

Table 5.1 presents the results for our algorithms and the four software packages. The first column of the table is the name of the instance. Then, the table is divided into five parts, and for each part the columns represent the computational time T, the area A, the cycles C, the third-stage cut Z and the index $index$, Equation 1.1, for the solutions. The first part is the best solution between the Local Search and Tabu Search procedures presented in Chapter 4. The next parts are the solutions of the software packages: GNCutter32, CutLogic2D, Plus 2D and Merick Calc 3000, respectively. For the last software package, Merick Calc 3000, the number of third stage cuts was not provided, and, we therefore present only the computational time T, the area A and the cycles C for the accuracy of calculation parameter equal to 3. The last row of the table reports average values.

The results show that, on average, the proposed algorithms have the minimum number of cycles, third-stage cuts and indexes compared with the software packages. The average area obtained with Plus 2D is smaller than the area required for the algorithms, but the average number of third stage cuts of the package is 3.66 times more than the average number of third-stage cuts of the algorithms. The algorithms have an appropriated computational time, although it is larger than the time required by Plus 2D, GNCutter32 and MerickCalc3000.

Table 5.1: Comparison of algorithms to commercial packages

instance	Proposed Algorithm					Plus2D					CutLogic2D					GNCutter32					Merick Calc 3000		
	T	*A*	*C*	*Z*	*index*	*T*	*A*	*C*	*Z*	*index*	*T*	*A*	*C*	*Z*	*index*	*T*	*A*	*C*	*Z*	*index*	*T*	*A*	*C*
instance10x25	600.00	20.43	3	3	76.90	3	20.43	3	16	80.80	69.0	21.06	4	8	83.07	1.45	20.43	3	3	76.90	3	21.06	4
instance10x50	600.00	40.60	5	5	149.83	2	43.08	7	18	167.07	201.0	40.35	7	7	154.77	1.33	40.35	5	0	147.33	2	40.35	8
instance10x75	600.02	54.76	6	4	199.20	7	63.25	7	30	237.67	76.8	57.72	7	7	212.77	3.25	75.13	9	2	274.60	7	54.75	6
instance10x100	600.03	74.78	7	3	268.57	4	74.78	7	33	277.57	310.2	74.78	7	9	270.37	2.23	87.25	7	0	308.67	7	79.19	12
instance10x150	600.19	124.64	6	2	431.60	3	124.64	14	41	464.63	270.6	124.64	11	14	448.53	3.38	131.62	9	0	462.00	4	125.50	15
instance10x200	600.00	160.90	10	3	563.57	3	161.12	15	36	587.80	154.2	162.87	13	13	580.57	2.89	176.23	12	1	619.30	3	169.23	16
instance10x300	600.06	221.35	12	10	772.00	4	224.59	16	39	802.37	88.2	224.02	15	21	792.30	2.17	242.62	12	2	840.60	8	226.55	16
instance10x400	600.00	296.23	16	3	1030.57	1	299.92	19	15	1054.17	65.4	304.89	20	19	1075.03	3.95	313.28	21	28	1108.40	1	310.46	24
instance10x600	600.00	440.93	16	13	1515.57	3	439.77	21	75	1543.50	82.8	444.41	20	41	1546.63	4.03	506.89	29	6	1768.13	7	453.50	22
instance10x800	600.00	699.86	19	6	2384.47	5	699.86	25	78	2422.07	72.0	699.86	20	29	2394.03	6.02	776.81	26	0	2658.33	11	699.86	28
instance40x50	600.00	34.16	6	17	134.10	60	34.16	6	38	140.40	487.2	34.16	6	20	135.00	3.64	42.72	9	2	166.60	16	34.16	6
instance40x75	600.00	53.93	9	10	206.00	75	53.93	8	37	211.43	441.6	52.31	9	20	204.00	1.83	56.09	7	28	213.07	37	53.93	9
instance40x100	599.23	67.97	9	24	257.20	66	67.97	10	63	271.57	514.2	72.16	7	41	270.97	2.58	76.68	11	17	289.43	79	68.04	12
instance40x150	600.00	100.17	17	30	387.33	54	100.27	18	87	408.10	1332.6	100.07	17	51	393.63	3.66	111.01	13	18	410.07	67	100.21	17
instance40x200	600.05	127.84	18	36	484.80	50	127.84	20	76	502.13	679.2	127.84	22	40	496.67	3.09	137.42	15	16	502.80	91	131.30	20
instance40x300	600.00	208.46	23	43	768.23	40	200.87	28	135	784.17	508.8	208.53	23	59	774.03	3.75	222.42	21	68	817.40	93	208.50	30
instance40x400	600.00	277.98	29	34	1013.53	46	287.18	51	94	1121.20	453.6	291.40	27	45	1056.50	3.56	317.81	37	23	1164.57	27	292.59	41
instance40x600	600.00	390.67	27	41	1386.30	56	387.26	36	161	1434.30	604.2	394.83	38	70	1438.33	4.22	432.29	31	59	1540.37	116	394.66	38
instance40x800	625.24	535.30	32	48	1883.73	44	525.12	51	199	1945.70	745.2	541.56	43	52	1935.27	3.00	560.74	47	99	2024.03	112	558.02	54
instance80x100	600.01	69.88	11	27	269.43	227	70.78	10	57	278.77	918.6	69.88	11	40	273.33	2.61	71.69	8	2	259.93	125	68.97	12
instance80x150	600.23	99.47	16	33	383.57	259	97.58	15	88	391.40	1291.8	98.71	17	53	390.23	2.67	110.48	12	19	405.70	156	99.47	16
instance80x200	600.00	131.59	22	54	512.87	247	128.71	21	113	518.90	1808.4	129.43	21	56	503.80	2.70	138.54	16	35	514.17	185	129.91	23
instance80x300	600.22	195.27	30	67	750.10	238	193.02	33	137	772.10	2287.8	195.74	28	100	756.67	2.41	204.68	26	95	779.83	304	195.24	36
instance80x400	600.75	271.38	38	55	1021.83	278	257.38	36	159	1000.70	1397.4	261.32	43	98	1015.07	2.92	276.39	28	93	1023.57	425	268.52	44
instance80x600	600.23	405.78	37	59	1468.37	225	392.94	41	279	1502.03	1463.4	396.92	44	126	1478.13	3.17	410.45	20	81	1445.63	963	404.51	43
instance80x800	649.72	541.74	49	71	1956.97	253	526.72	48	303	1973.90	2241.6	538.42	48	146	1965.80	3.86	563.60	30	133	1997.90	948	536.46	56
3instance40x600	600.23	392.95	29	38	1397.73	64	356.81	37	190	1344.67	684.6	396.57	31	78	1427.07	4.13	436.30	38	63	1574.23	141	396.03	39
3instance40x800	600.00	528.93	33	32	1860.60	57	521.42	48	191	1923.30	742.8	531.03	54	113	1947.90	5.84	560.26	37	69	1986.37	246	525.96	56
3instance80x600	626.47	406.67	40	80	1485.67	171	391.14	44	251	1495.63	1218.6	400.96	37	92	1462.27	23.78	475.67	47	46	1724.13	1080	399.92	47
3instance80x800	656.36	548.37	47	59	1970.03	207	529.66	66	295	2029.50	3741.6	543.10	68	136	2032.13	4.66	585.24	65	81	2147.63	459	538.98	73
average	605.30	250.77	20.73	30.33	899.69	91.73	246.74	25.37	111.13	922.92	831.78	251.32	23.93	53.47	917.16	3.96	272.03	21.70	36.30	975.06	190.77	252.86	27.43

Chapter 6

Summary

This work is devoted to the problem of cutting rectangular boards in the wood industry, to obtain the desired items, and to the description of heuristic algorithms to solve the problem, regarding usage minimization and machine productivity. Usage minimization is intended as the minimization of the total raw material used, while the machine productivity is achieved by considering special features of the cutting machine, like the number of stages performed to obtain an item and the possibility of simultaneously cutting more than one board. The development of the algorithms is done in an empirical manner. The components are added one at a time and the heuristic parameters are set on the basis of systematic and thorough testing. Moreover, the results are examined in order to find why certain configurations work better than others.

In Chapter 2, Constructive Heuristic algorithms were presented to find a solution for the problem in a short time. The algorithms are based in the construction of strips that are subsequently grouped into patterns. We study the parameters of the algorithms, and set them to the values that improve their performance.

In Chapter 3, the Gilmore and Gomory model and the Column Generation approach were used to improve the solution found by the Constructive Heuristic algorithms. A Diving Heuristic procedure to find an integer solution is proposed. Based on the continuous relaxation of the model, the Diving Heuristic procedure explores a branch of the search tree by performing a depth-first search. Several strategies for bounding the variables at the nodes of the search tree are studied. The procedures results very effective improving the average performance of the Constructive Heuristic solution of 9,51%.

In Chapter 4, two post optimization techniques were proposed to further improve the solution. The Local Search approach continues the exploration of the search tree using the Diving Heuristic. In detail, given a solution, some patterns are selected and removed, and different branches of the search tree are examined. Different criteria for selecting the patterns are studied. The Tabu Search approach tackles the problem in a set covering fashion. Given a minimal covering solution, some patterns are added to the tabu list and a new solution is built without considering the patterns in the tabu list. Moreover, when the solution is not improved in a given number of iterations, a crossover operator is used to change the explored region. We study the parameters of the algorithm and set them to the values that improve its performance.

Finally, in Chapter 5, we compared the results obtained by the developed algorithms

with those obtained by commercial software packages available for the solution of 2DCSP. Although commercial software packages may also consider other features of the production process, and therefore a straight comparison can not be carried out, the results show the effective performance of the algorithm for usage minimization and machine productivity.

Bibliography

[1] R. Alvarez-Valdes, R. Marti, J. Tamarit, and A. Parajon. Grasp and path relinking for the two-dimensional two-stage cutting-stock problem. *INFORMS Journal on Computing*, (2):261–272, 2008.

[2] R. Alvarez-Valdes, A. Parajon, and J. Tamarit. A computational study of lp-based heuristic algorithms for two-dimensional guillotine cutting stock problems. *OR Spectrum*, (2):179–192, 2002.

[3] N. Christofides. An algorithm for two-dimensional cutting problems. *Operations Research*, (1):30–44, 1977.

[4] G. Cintra, F. Miyazawa, Y. Wakabayashi, and E. Xavier. Algorithms for two-dimensional cutting stock and strip packing problems using dynamic programming and column generation. *European Journal of Operational Research*, (1):61–85, 2008.

[5] A. Farley. Practical adaptations of the gilmore-gomory approach to cutting stock problems. *OR Spectrum*, Jan 1988.

[6] S. Fekete and J. Schepers. New classes of fast lower bounds for bin packing problems. *Mathematical Programming*, (1):11–31, 2001.

[7] F. Furini, A. E. Fernandes Muritiba, R. Medina Durán, and A. Persiani. Column generation approach for 2d guillotine cutting stock problems with bins of different shapes, 2009. In OPTIMA, Congreso Chileno de Investigación Operativa, 2009, Chillán. OPTIMA 2009.

[8] F. Furini, E. Malaguti, R. Medina Durán, A. Persiani, and P. Toth. A column generation heuristic for the two-dimensional two-staged guillotine cutting stock problem with multiple stock size, 2011. submitted to an International Journal.

[9] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)*. W. H. Freeman & Co Ltd, first edition edition, January 1979.

[10] P. Gilmore and R. Gomory. A linear programming approach to the cutting-stock problem. *Operations Research*, (6):849–859, 1961.

[11] P. Gilmore and R. Gomory. A linear programming approach to the cutting stock problem-part ii. *Operations Research*, (11):863–888, 1963.

[12] P. Gilmore and R. Gomory. Multistage cutting stock problems of two and more dimensions. *Operations Research*, (13):94–120, 1965.

[13] IBM ILOG CPLEX v12.1. User's Manual for CPLEX, 2011. ftp://ftp.boulder.ibm.com/software/websphere/ilog/docs/optimization/cplex/ps_usrmancplex.pdf.

[14] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack problems*. Springer, Berlin, 2004.

[15] A. Lodi and M. Monaci. Integer linear programming models for 2-staged two-dimensional knapsack problems. *Mathematical Programming*, (94):257–278, 2003.

[16] R. Macedo, E. Silva, C. Alves, F. Pereira e Alvelos, J. M. V. de Carvalho, C. Arbib, F. Marinelli, F. Pezzella, L. de Giovanni, and L. Gambella. 2d cutting stock optimization software survey, 2010. http://www.scoop-project.net/DOCUMENTI/File/MacedoEtAlii_080715.pdf.

[17] E. Malaguti, R. Medina Durán, and P. Toth. Algorithms for the cutting stock problem with multi-objective function and their application in the wood industry, 2011. submitted to an International Journal.

[18] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, Chichester, 1990.

[19] A. F. Muritiba, M. Iori, E. Malaguti, and P. Toth. Algorithms for the bin packing problem with conflicts. *INFORMS Journal on Computing*, (22):401–415, 2010.

[20] Nirvana Technologies Private Limited. Plus2D. http://www.nirvanatec.com/panel_optimization_wood.html.

[21] Optimalon Software. GNCutter32. http://www.tmachines.com/cutlogic-2d.htm.

[22] J. Riehme, G. Scheithauer, and J. Terno. The solution of two-stage guillotine cutting stock problems having extremely varying order demands. *European Journal of Operational Research*, (91):543–552, 1996.

[23] Soft Consult. Merick Calc 3000. http://www.softconsult.cz/us/mc3k/index.htm.

[24] TMachines, s.r.o. CutLogic2D. http://www.optimalon.com/cutting_optimization_library.htm.

[25] F. Vanderbeck. Computational study of a column generation algorithm for bin packing and cutting stock problems. *Mathematical Programming*, (86):565–594, 1999.

[26] F. Vanderbeck. A nested decomposition approach to a three-stage, two-dimensional cutting-stock problem. *Management Science*, Jan 2001.

Appendix A

ILP Models for 2DCSP

The model of Gilmore and Gomory, Equations (3.1)–(3.3), used before for the Column Generation, does not explicitly consider the cycles of the machines. We propose two ILP Models for 2DCSP that explicitly consider the cycles of the machines. The first model considers a variable to define the number of cycles of each pattern; while the second model considers each feasible repetition of the patterns in the solution. The details of the models are given in the following.

A.1 Model 1

The first model, Equations (A.1-A.4), considers an integer variable x_j for each feasible pattern j ($j = 1, \ldots, \eta$). The value of the variable x_j indicates the number of times that pattern j is in the solution. In addition, for each pattern j, a variable y_j determines its number of cycles .

$$min \quad \sum_{j=1}^{\eta}(f_j \times x_j + v_j \times y_j) \tag{A.1}$$

$$s.t. \quad \sum_{j=1}^{\eta} p_j^i x_j \geqslant d_i \quad i = 1, \ldots, m \tag{A.2}$$

$$y_j \geqslant \frac{x_j}{\kappa} \quad j = 1, \ldots, \eta \tag{A.3}$$

$$x_j \in \mathbb{Z}^+, \quad y_j \in \mathbb{Z}^+ \quad j = 1, \ldots, \eta \tag{A.4}$$

As before, the vector $P_j = [p_j^1 \ldots p_j^i \ldots p_j^m] \in \mathbb{Z}^m$ represents the items in the pattern, where p_j^i is the number of items of item class i ($i = 1, \ldots, m$) that are in the pattern j; the cost associated with each variable x_j depends on the area of the board $B_j = k$, calling A_j^k the area of the board class k, the cost f_j is defined as:

$$f_j = \frac{10}{3} \times A_j^k \tag{A.5}$$

the cost v_j associated with each variable y_j is defined as:

$$v_j = \frac{8}{3} + 0.3 \times Z_j \tag{A.6}$$

where Z_j is the number of third-stage cuts in the pattern j.

In order to strengthen the model, we add the following constraints that impose a minimum *number of variables* to cover each item class.

$$\sum_{j:i\in I_j} x_j \geqslant min_j\left\{\left\lceil\frac{d_i}{p_j^i}\right\rceil\right\} \qquad i=1,\ldots,m \tag{A.7}$$

A.2 Model 2

The second model considers an integer variable x_j^t for each feasible pattern j ($j=1,\ldots,\eta$) and each possible number of boards simultaneously cut in a cycle $t=1,\ldots,\kappa$.

$$\begin{aligned}
min \quad & \textstyle\sum_{j=1}^{\eta}\sum_{t=1}^{\kappa} C_j^t \times x_j^t & & & (A.8)\\
s.t. \quad & \textstyle\sum_{j=1}^{\eta}\sum_{r=1}^{\kappa} tp_j^i x_j^t \quad \geqslant \quad d_i & i=1,\ldots,m & & (A.9)\\
& x_j^r \in \{0,1\} \quad j=1,\ldots,\eta & r=1,\ldots,\kappa-1 & & (A.10)\\
& x_j^\kappa \in \mathbb{Z}^+ & j=1,\ldots,\eta & & (A.11)
\end{aligned}$$

Each variable has a cost C_j^t that expresses the repetition cost and the cycles cost, calling A_j^k the area of the board $B_j=k$, the cost is defined as:

$$C_j^t = t\frac{10}{3}A_j^k + \frac{8}{3} + 0.3Z_j \tag{A.12}$$

where Z_j is the number of third-stage cuts in the pattern. Again, the vector $P_j=[p_j^1\ldots p_j^i\ldots p_j^m]\in\mathbb{Z}^m$ represents the items in the pattern, where p_j^i is the number of items of item class i ($i=1,\ldots,m$) that are in the pattern.

In order to strengthen the model, we add the following constraints that impose a minimum *number of variables* to cover each item class.

$$\sum_{j:i\in I_j}\sum_{t=1}^{\kappa} x_j^t \geqslant min\left\{\left\lceil\frac{d_i}{p_j^i}\right\rceil\right\} \qquad i=1,\ldots,m \tag{A.13}$$

Also for the binary variables, we add the following constraints to avoid the symmetry:

$$\sum_{t=1}^{\kappa-1} x_j^t \leqslant 1 \quad j=1,\ldots,n \tag{A.14}$$

A.3 Computational Results

Solving the models to optimality may be very difficult for non trivial instances, because of the exponential number of patterns.

To study the models, we used the patterns generated after the Diving Heuristic procedure, and compared the results of the models with the solution of the Tabu Search, see 4.3, that use the same set of columns. To solve the models we used the ILP Solver of CPLEX 12 [13], taking advantage of the advanced routines embedded in the ILP Solver.

Table A.1 presents the results of the best configuration of the Tabu Search, Model 1 and Model 2. The first column in the table is the name of the instances followed by the computational time limit in the second column. Then, the Table has three main parts. The first one reports the results of the Tabu Search, presenting the area of the solution (A), the number of cycles (C), the third-stage cuts (Z) and the value of Equation 1.1, ($index$). The second part of the Table reports the results of Model 1 and the last part reports the results of Model 2. For the models, the columns present the computational time (T), the objective function (obj), the optimality gap (gap), the area of the solution (A), the number of cycles (C), the third-stage cuts (Z) and the value of Equation 1.1, ($index$). The last row presents average values.

The Table shows that the Tabu Search is more efficient than the proposed models when considering the average $index$ value. However, Model 2 seems to be more effective in the reduction of cycles and third-stage cuts; we observe that the average of cycles (C) and third-stage cuts (Z) are smaller. With respect to the computational time of the models, Model 2 is faster than Model 1, but for many instances the time limit is reached. For those instances where the time limit is not reached, the solutions of the models are the same and, for the majority of the instances, equal to the solution of the Tabu Search.

Table A.1: Comparison of Tabu Search to Models

instance	*Time Limit*	Tabu Search			Model 1						Model 2					
		A	*C*	*Z*	*index*	*T*	*obj*	*gap*	*A*	*C*	*Z*	*index*	*T*	*obj*	*gap*	*A*	*C*	*Z*	*index*
instance10x25	596.01	20.43	3	3	76.90	1.82	77.02	0.00	20.43	3	3	76.90	0.07	77.02	0.00	20.43	3	3	76.90
instance10x50	592.39	40.60	5	5	149.83	82.85	150.17	0.01	40.60	5	5	149.83	0.99	150.18	0.00	40.60	5	5	149.83
instance10x75	590.15	57.72	6	6	209.80	191.37	210.20	0.01	57.72	6	6	209.80	0.82	210.22	0.00	57.72	6	6	209.80
instance10x100	593.31	74.78	8	9	273.03	32.03	273.29	0.01	74.78	8	9	273.03	3.24	273.30	0.00	74.78	8	9	273.03
instance10x150	592.76	124.64	7	2	434.27	592.76	429.87	1.12	124.64	7	2	434.27	435.57	434.70	0.01	124.64	7	2	434.27
instance10x200	587.67	160.90	10	3	563.57	587.67	562.15	0.25	160.90	10	2	563.27	587.67	559.01	0.86	160.90	10	3	563.57
instance10x300	576.76	223.32	10	6	772.47	576.76	763.65	0.67	221.82	10	9	768.37	576.76	762.24	2.10	223.09	12	10	778.00
instance10x400	550.79	296.23	16	3	1030.57	0.84	1027.94	0.01	296.23	15	2	1027.60	7.69	1028.04	0.00	296.23	15	2	1027.60
instance10x600	535.22	440.93	16	13	1515.57	535.22	1505.49	0.42	439.77	15	20	1511.00	535.22	1503.33	0.92	440.93	16	16	1516.47
instance10x800	542.06	699.86	19	6	2384.47	542.06	2371.77	0.38	699.86	18	0	2380.00	542.06	2370.63	0.43	699.86	18	0	2380.00
instance40x50	478.50	34.16	6	17	134.10	478.50	128.56	10.99	37.01	7	8	144.07	96.43	134.36	0.00	34.16	6	15	133.50
instance40x75	386.06	53.93	9	10	206.00	386.06	189.51	16.16	59.09	9	17	225.10	386.06	197.17	4.64	53.39	9	16	205.80
instance40x100	96.10	67.97	10	39	264.37	96.10	237.18	19.66	76.53	12	27	295.10	96.10	247.12	11.14	72.19	10	36	277.47
instance40x150	309.50	100.17	17	30	387.33	309.50	342.66	14.51	105.54	15	30	400.00	309.50	362.55	7.76	105.54	13	22	392.27
instance40x200	255.07	130.67	19	24	492.87	255.07	442.79	13.57	136.95	18	26	511.80	255.07	461.98	7.49	135.22	16	20	498.67
instance40x300	130.35	208.46	23	43	768.23	130.35	710.97	9.66	213.21	24	41	786.30	130.35	731.29	6.96	217.88	19	30	785.67
instance40x400	115.95	277.98	29	34	1013.53	115.95	967.40	4.16	279.75	26	25	1008.83	115.95	976.33	4.87	285.88	25	22	1025.27
instance40x600	211.60	390.67	27	41	1386.30	211.60	1335.10	4.90	395.96	27	40	1403.00	211.60	1347.14	4.46	399.00	27	27	1409.10
instance40x800	50.00	535.30	32	48	1883.73	50.00	1811.50	4.25	532.72	38	48	1890.73	50.00	1815.32	6.91	559.10	28	38	1949.07
instance80x100	94.66	69.88	11	28	269.73	94.66	251.42	16.20	76.56	14	25	299.83	94.66	256.57	6.97	70.78	10	44	274.87
instance80x150	83.92	99.47	16	38	385.07	83.92	357.73	22.07	118.46	19	45	458.17	83.92	365.17	8.02	101.37	17	46	396.13
instance80x200	87.48	131.59	22	54	512.87	87.48	467.93	21.01	151.00	28	48	592.07	87.48	484.18	8.89	136.62	23	49	531.03
instance80x300	69.63	195.27	30	71	751.30	69.63	683.01	18.64	216.80	36	69	838.70	69.63	713.09	13.39	219.54	29	47	822.43
instance80x400	51.63	270.43	43	34	1025.87	51.63	893.38	20.53	299.24	41	58	1123.73	51.63	942.83	12.63	294.48	34	23	1078.57
instance80x600	51.57	404.88	38	64	1469.53	51.57	1338.57	16.97	437.92	45	108	1611.40	51.57	1390.51	12.33	441.90	37	48	1585.07
instance80x800	50.00	541.74	49	71	1956.97	50.00	1787.72	16.42	586.76	59	86	2138.13	50.00	1835.34	10.43	578.48	40	47	2048.77
3instance40x600	84.43	396.17	26	31	1398.63	84.43	1333.19	5.25	393.92	31	38	1407.07	84.43	1344.97	6.90	406.60	30	31	1444.30
3instance40x800	51.35	528.93	33	32	1860.60	51.35	1785.68	4.74	529.48	37	36	1873.47	51.35	1794.24	7.02	551.34	31	30	1928.67
3instance80x600	50.00	406.67	40	80	1485.67	50.00	1338.65	18.70	444.81	51	93	1645.90	50.00	1390.75	12.52	439.17	40	64	1588.87
3instance80x800	50.00	548.37	47	59	1970.03	50.00	1824.49	9.88	551.92	61	73	2023.57	50.00	1865.92	10.01	580.38	46	54	2072.87
average	283.83	251.07	20.90	30.13	901.11	196.71	853.30	9.04	259.35	23.17	33.30	935.70	168.86	867.52	5.59	260.74	19.67	25.50	928.60