A Simulated Annealing algorithm for real-world 2-D Cutting Stock Problem with Setup Cost

Stephane Bonnevay[1], Philippe Aubertin[2], Thibault Lazert[1]

[1] ERIC, Universite de Lyon, 5 av. Pierre Mends France, 69676 Bron Cedex, France
stephane.bonnevay@univ-lyon1.fr

[2] AXOPEN, 149 bd Stalingrad, 69100 Villeurbanne, France
philippe.aubertin@axopen.com

Abstract

This article deals with the Two-Dimensional Cutting Stock Problem with Setup Cost (2CSP-S). This work was conducted for a paper industry company on their real data. The 2CSP-S combines three optimization sub-problems: a 2-D Bin Packing (2BP) problem (to place images on patterns), a Linear Programming (LP) problem (to find for each pattern the number of stock sheets to be printed) and a combinatorial problem (to find the number of each image on each pattern). We propose a method to solve the 2CSP-S focusing on this third sub-problem. Our approach is a simulated annealing algorithm that aims to automatically determine the proper number of each image on each pattern. It is important to notice that the proposed method is not a new packing technique. Our algorithm was tested to solve the 2CSP-S on real and artificial datasets.

1 Introduction

The problem of cutting stock materials (paper, steel, glass, ...) in order to satisfy the customer demand while minimizing the trim loss is known as the Cutting Stock Problem (CSP). In CSP, a sufficient number of setups/patterns are used to solve the problem. The CSP is the same problem as the Bin Packing Problem where a set of items must be grouped into bins. In the past, several different methods were developed in order to solve this couple of problems, including linear programming, heuristics, and metaheuristics [2, 4, 5, 9, 13, 19, 12]. In some real-world applications, optimizing the number of patterns is important because the manufacturing cost of one pattern can be expensive in regard to the cost of one stock sheet. The Pattern Minimization Problem (PMP) aims to minimize the number of different cutting patterns while satisfying the demand [1, 14]. Another problem, called the Cutting Stock Problem with Setup Cost (CSP-S), takes into account the costs of the setup and the stock sheet print in order to minimize the total production cost. Most of existing papers deal with one dimensional problems [3, 6, 8, 15, 18].

This paper deals with the Two-Dimensional Cutting Stock Problem with Setup Cost (2CSP-S) in order to solve a real-world application of the paper industry[1]. According to the 2CSP-S, customer demands correspond to the number of prints of several rectangular images. The aim is to satisfy customer demands while minimizing the total production cost (stock sheet cost and setup cost) by automatically fixing: the number of patterns, the number of stock sheets to be printed for each pattern, the number and the position of each rectangular image on patterns. Our method is not a new cutting (packing) technique (where the number of objects to be placed is known): in this work the number of objects (rectangular images) to be set on each pattern is unknown (only the demands are known). Our work focuses on the problem of finding the proper number of each image on each pattern (and to find the optimal number of patterns). For this purpose, we developed a simulated annealing algorithm. 2BP and LP sub-problems of the 2CSP-S are solved by well-known algorithms. In our case, images could be rotated by $90°$. This positioning consists in placing images without overlap. Positions of images have to satisfy the guillotine cut constraint meaning that the cutting must be a sequence of edge-to-edge cuts parallel to the edges of the pattern.

Section 2 presents a mathematical formulation of the 2CSP-S and sets out an example as way of explanation. Our algorithm, with all other used methods are detailed in Section 3. Experiments on real and artificial datasets are presented in Section 4.

[1] This is a cooperation with the Seripress company (http://www.seripress.com/)

2 Problem definition

2.1 Mathematical formulations

Let I be the set of n rectangular images. An image i is defined by its width w_i, its height h_i and its demand d_i (the number of prints of image i). Let J be the set of m patterns with the same size $H \times W$. The number m is unknown. Let p_{ji} be the number of image i in the pattern j. We note $p_j = (p_{j1}, \ldots, p_{jn})$. Let π_j be a placement (rectangle packing) of all the images of the pattern j (π_j includes exact position and rotation information). The number of possible positions is limited by the algorithm (see Section 3.2). Thereby, a pattern j could be defined as the couple (p_j, π_j). In the following of this paper, we shall only consider feasible couples (p_j, π_j) (i.e. with feasible placement). Let x_j be the number of stock sheets to be printed for the pattern j. The decision variables are: m, p_j, x_j and π_j $\forall j \in J$.

The main objective (fitness) is to minimize the total cost:

$$f(m, x_1, \ldots, x_m) = mC_{su} + C_{ss} \sum_{j \in J} x_j \tag{1}$$

where C_{su} is the manufacturing cost of one setup, and C_{ss} is the printing cost of one stock sheet.

To satisfy the demand d_i requested for each image i, there are n constraints:

$$\sum_{j \in J} p_{ji} x_j \geq d_i \ \forall i \in I \tag{2}$$

These n constraints and the objective function f (defined by Equation 1) define an integer linear program (where the decision variables are x_j $\forall j \in J$).

In order to simplify the problem, our algorithm is run with several fixed values of m (see line 5 of Algorithm 3). Thus, a solution of 2CSP-S is composed of a tuple $((p_1, \pi_1), \ldots, (p_m, \pi_m))$ and a tuple $(x_1, \ldots, x_m)$ which is the solution of the integer linear program defined above. Let S be the set of all possible solutions.

2.2 Example

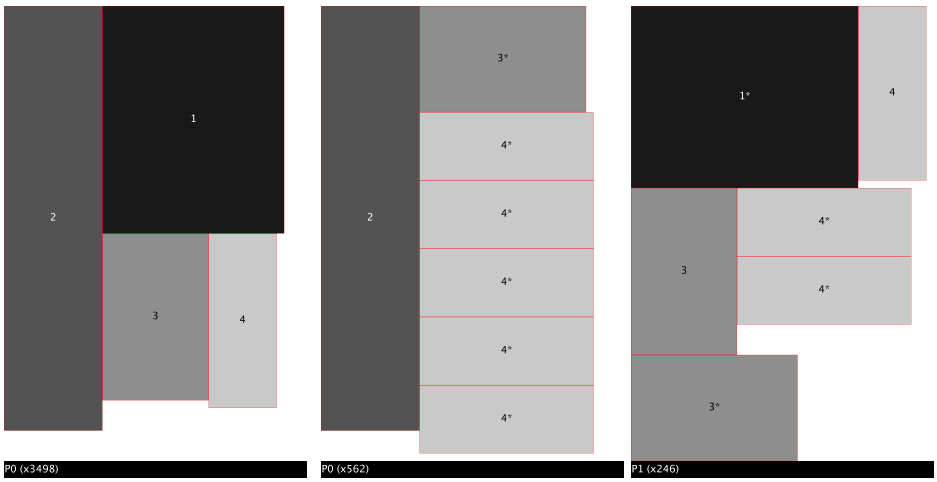
In order to illustrate the 2CSP-S problem, we display a simple example with 4 customers demands: image 1 (size: 30×24, demand: 246), image 2 (size: 56×13, demand: 562), image 3 (size: 22×14, demand: 1000) and image 4 (size: 23×9, demand: 3498). The pattern size is equal to 60×40.

The objective is to reduce the total material cost depending exclusively on C_{su} and C_{ss} (see Equation (1)). Here, $C_{su} = 20\$$ and $C_{ss} = 1\$$. As an example, calculations have been performed with 1 pattern ($m = 1$) and with 2 patterns ($m = 2$). Results are displayed in Figure 1. In all the figures of this paper, a star (*) next to the image number indicates that the corresponding image has been rotated $90°$. As shown in Table 1, the best solution is obtained with 2 patterns (see Figure 1b) even if the manufacturing cost of one setup is 20 times greater than the printing cost of one stock sheet.

Table 1: Costs corresponding to Figure 1, with $C_{su} = 20\$$ and $C_{ss} = 1\$$

Nb of patterns	Nb of stock sheets printed	Total cost
1	3498	$1 \times 20\$ + 3498 \times 1\$ = 3518\$$
2	808 (246 + 562)	$2 \times 20\$ + 808 \times 1\$ = 848\$$

It is important to understand that the best result is not obtained by fully completing all the patterns. This means that we cannot use the blank area rate to define the quality of a solution. According to overproduction, with one pattern, the overproduction is very large: 3498 copies of the images 1, 2 and 3 are printed, instead of respectively, 246, 562 and 1000 copies. With 2 patterns, only 54 copies of the image 3 and 50 copies of the image 4 are overly printed.



(a) Solution with 1 pattern (3498 prints)

(b) Solution with 2 patterns (246 + 562 = 808 prints)

Figure 1: Solutions found by our algorithm with one (a) and two (b) patterns.

2.3 Overproduction

In order to take into account the overproduction in our model, we define r_i as the quantity of images i overproduced and $OverP$ the total overproduction:

$$r_i = \sum_{j \in J} p_{ji} x_j - d_i \; \forall i \in I \text{ and } OverP = \sum_{i \in I} r_i \tag{3}$$

In our model, the main objective is the minimization of the fitness f (see Equation 1). But if two solutions s and s' of S have the same fitness f then the one which minimize $OverP$ is chosen. Thereby, a method called *BestSolution*(s,s') is used to return the best solution between s and s'. This method combines f and $OverP$.

2.4 Bounds

We first calculate a lower bound L_0 of the number of patterns:

$$L_0 = \left\lceil \frac{\sum_{i \in I} h_i w_i}{H \times W} \right\rceil \tag{4}$$

L_0 corresponds to the minimum number of patterns when positioning only one copy of every image with no unoccupied areas on the patterns. Thus, $m \geq L_0$.

In the following, we note L_1 to be the minimum number of patterns found by our placement algorithm. Thus, $L_1 \geq L_0$. L_1 will be evaluated by the *InitialSolution*(L_0) method (see Section 3.4).

We define the upper bound $L_{max}(i)$ of the number of images i which can be placed on one pattern as:

$$L_{max}(i) = \left\lfloor \frac{H \times W}{h_i w_i} \right\rfloor \tag{5}$$

$L_{max}(i)$ is the theoretical maximum number of image i in one pattern. It is used to limit the process of adding an image in a pattern when building the neighborhood of a solution.

3 Our algorithm

2CSP-S is a combination of three optimization sub-problems: the first one is a 2-D Bin Packing (2BP) problem which consists of finding the placement $(\pi_1, \ldots, \pi_m)$ of all images on the m patterns; the

second one is a Linear Programming (LP) problem which consists of finding the number $(x_1, \ldots, x_m)$ of stock sheets to be printed for each pattern; the third one is a combinatorial problem which consists of finding the proper tuple $(p_1, \ldots, p_m)$ of each image on m patterns (and find the number of patterns m). In this section, we present our simulated annealing algorithm used to solve this third sub-problem: find the proper number of each image on each pattern. A well-known algorithm (see Section 3.2) and a classical linear programming solver were used to solve the first (2BP) and the second (LP) sub-problems. The following subsections detail each part of our approach.

3.1 Stock sheet prints

The fitness, *Fitness*(s), of a solution $s \in S$ is a combination of the total cost given by Equation 1 and overproduction quantities (see Section 2.3). But in order to compute $(x_1, \ldots, x_m)$ according to a fixed number m of patterns, a simplified fitness definition is given as follows:

$$f'(s) = \sum_{j \in J} x_j \tag{6}$$

As explained in Section 2.1, $f'(s)$ is the result of an integer programming problem. Indeed, given a tuple $(p_1, \ldots, p_m)$ which represents the numbers of all images in the m patterns (this tuple will be found by our simulated annealing algorithm), the aim is to find $(x_1, \ldots, x_m)$ minimizing Equation 6 under constraints given by Equation 2. As m is fixed at each step (see line 5 of Algorithm 3), all patterns are useful, thus:

$$x_i \geq 1 \; \forall i \in I \tag{7}$$

As we must evaluate the fitness of each solution during our algorithm, it is crucial to optimize this phase. Thereby, the corresponding relaxation problem (real-LP) is solved and the solution $(x_1^*, \ldots, x_m^*)$ rounded to upper whole numbers $(\lceil x_1^* \rceil, \ldots, \lceil x_m^* \rceil)$ is returned. Thus, $x_j = \lceil x_j^* \rceil \; \forall j \in J$.

The method which calculates the number of stock sheets to be printed and the fitness of the solution s is called *ComputePrintsAndFitness*(s). The time complexity of this method is denoted $cpaf(n, m)$.

3.2 2-D bin packing method

In order to deal with only feasible solutions, a 2-D Bin Packing (2BP) algorithm has to be used to build the placement $(\pi_1, \ldots, \pi_k)$ on k patterns according to a list of images $(q_1, \ldots, q_n)$, where q_i is equal to the number of image i. There are several methods to make a 2BP from a list of images [11, 13]. In our case, we slightly adapted an existing algorithm, named *maximal rectangles best short side fit*, which satisfies the guillotine cut constraint and which is one of the best constructive algorithms of [11].

In the following of this paper, this method is called *2DBinPacking*(k,q), where k is the number of patterns used and $q = (q_1, \ldots, q_n)$ is a n-tuple of the total number q_i of each image i. The output is a list $(\pi_1, \ldots, \pi_k)$ of placements on k patterns or *null* if there is no feasible packing. Unfeasible packing appears if the number of patterns k is too small to place all images given by q.

According to [11], the time complexity of this method is $O(n_q^2 + n_q k)$ (where $n_q = \sum_{i \in I} q_i$). As $k < n$, it is equal to $O(n_q^2)$.

3.3 Neighborhood of a solution

To create a simulated annealing algorithm to automatically determine the proper number of each image on patterns, a neighborhood of any solution s is defined by modifying the tuple $(p_1, \ldots, p_m)$.

Let $s \in S$ be a solution composed of a tuple $((p_1, \pi_1), \ldots, (p_m, \pi_m))$ and a tuple $(x_1, \ldots, x_m)$ which is the solution of the relaxed linear program. A neighbor of s is built from one of these four operators: *add* one image in one pattern, *remove* one image from one pattern, *move* one image from a pattern to another one, *swap* two images from two different patterns. These operators only modify the tuple $(p_1, \ldots, p_m)$. The operator *add* add one image i in the pattern j which involves incrementing p_{ji}

if p_{ji} is less than $L_{max}(i)$. The operator *remove* removes one image i from the pattern j which involves decrementing p_{ji} if the number of image i in the whole solution s remains strictly positive. And so on.

During the simulated annealing process, only one neighbor is randomly chosen at each step. This neighbor is generated by randomly selecting one elementary operator, and then by randomly selecting one (*add* and *remove*) or two (*move* and *swap*) patterns, and one (*add*, *remove* and *move*) or two (*swap*) images in these patterns. Of course, these elementary operators can lead to an unfeasible solution (unfeasible placement). So, the method *2DBinPacking*(1,p_j) (see Section 3.2) is applied on each modified pattern j in order to check the placement feasibility and to build π_j (if feasible).

Thereby, the method *ChooseNeighbor*(s) chooses a neighbor of s, checks the feasibility on each modified pattern and returns either the solution corresponding to this neighbor, or *null* if there is no feasible packing (see Algorithm 1). Its time complexity only depends on the *2DBinPacking* complexity.

Algorithm 1 Choose a neighbor

```
1:  function CHOOSENEIGHBOR(s)
2:      s' ← s                                   ▷ copy s in s'
3:      Choose an elementary operator and apply it on s'
4:      for each modified p'_j do                ▷ one modified pattern in add and remove
                                                 ▷ two modified patterns in move and swap
5:          π ← 2DBinPacking(1,p'_j)
6:          if π != null then π'_j ← π           ▷ copy the new placement π in π'_j of s'
7:          else return null
8:          end if
9:      end for
10:     return s'                                ▷ return the neighbor of s
11: end function
```

3.4 Initial solution generator

To begin the simulated annealing process, an initial feasible solution has to be generated. The first step of this initialization is inspired by [10]. The objective is to place only one copy of each image i on k patterns. Thus, the method *2DBinPacking*(k,($q_1,\ldots,q_n$)) is run with $q_i = 1\ \forall i \in I$. If it returns *null*, the method is repeated with $k+1$ patterns and so on, until a feasible placement $(\pi_1,\ldots,\pi_k)$ is found. At the beginning, this method is run with $k = L_0$ (the lower bound of patterns number, see Section 2.4). At the end of this first step, an initial feasible solution s_0 is created. This first step also gives the minimum number of patterns L_1 (according to the performance of our *2DBinPacking* method); L_1 is the number of patterns of the solution s_0.

Experiments show that this initial solution s_0 is often very bad. Moreover, as *2DBinPacking* is a deterministic method, even if this first step is performed several times, s_0 will be the same. That is the reason why a second step is performed in order to obtain a random initial solution. Thus, from s_0, a random walk is performed: a solution s_1 is chosen in the neighborhood of s_0, a solution s_2 is chosen in the neighborhood of s_1, and so on, until a fixed iterations number `NbWalk`.

The complexity of *InitialSolution*(k) (see Algorithm 2) is $O(\texttt{NbWalk} \times n_{max}^2 + cpaf(n,k))$, where n_{max} is the total number of images in q during the random walk.

3.5 Main algorithm

Our algorithm, *SA-2CSP-S* (see Algorithm 3), combines all the previous methods in a simulated annealing process.

From experiments on the real datasets of the company, we observed that the optimal number of patterns is not so far from the real minimum number of patterns L_1 (probably because the cost of one pattern creation is much higher than the sheet print cost). In this way and to simplify the problem of

Algorithm 2 Initial solution generator

```
1:  function INITIALSOLUTION(k)                          ▷ k is the initial number of patterns
2:      q_i ← 1 ∀i ∈ I                                   ▷ one copy of each image
3:      π ← 2DBinPacking(k,(q_1, ..., q_n))
4:      while π == null do
5:          k ← k + 1
6:          π ← 2DBinPacking(k,(q_1, ..., q_n))
7:      end while
8:      s ← new Solution(π)                              ▷ create a solution from π
9:      ComputePrintsAndFitness(s)                       ▷ compute prints and fitness
10:     for l ← 1 to NbWalk do                           ▷ random walk
11:         s' ← ChooseNeighbor(s)
12:         if s' != null then s ← s'
13:         end if
14:     end for
15:     ComputePrintsAndFitness(s)                       ▷ compute prints and fitness
16:     return s
17: end function
```

2CSP-S, the number of patterns is first fixed to L_1. L_1 corresponds to the number of patterns of the initial solution found by the method *InitialSolution*(L_0) (line 4). Then, this number is increased twice to test two more number of patterns. From a computational point of view, the loop line 5 has been parallelized; thus the computation time of this loop is independent of the number of iterations.

Our algorithm is a classical simulated annealing process. There is NbT temperature decreases (loop line 9), and NbM moves in the space of solution S at each temperature value (loop line 10). A neighbor s' of a solution s is accepted, either if s' is better than s (line 16), or if the ratio between Δ and the current temperature is not too large (line 20). At the end of each search, a local improvement is applied on the best solution found with a hill-climbing process (line 27) followed by an overproduction removal (line 28). The time complexity of our main algorithm is $O(\texttt{NbT} \times \texttt{NbM} \times (n_{max}^2 + cpaf(n,m)))$.

4 Experiments

A *Java* application was developed, using *Java.Stream* to parallelize the loop line 5 of *SA-2CSP-S* (see Algorithm 3) and a Java Simplex Solver package to compute $(x_1, \ldots, x_m)$. Thereby, the complexity $O(cpaf(n,m))$ depends on this Simplex Solver package; this complexity is not in $O(n^2)$. Thus, the global time complexity depends primarily on $O(cpaf(n,m))$. Calculations were run on a MacBook Pro 2.2GHz Intel Quad Core i7 16Go.

There are several existing modeling approaches to manipulate and code a solution like the graph-theoretical characterization [7], the binary tree [17] or the sequence pair [16]. Here, the *2DBinPacking*(k,q) method stores only the coordinates of each image on each pattern.

A thorough study of computation time highlighted that 42% of time is passed in the *ComputePrintsAndFitness*() method line 13 and 33% of time is passed in the *ChooseNeighbor*() method line 11. This is in accordance with the time complexity study.

4.1 Real-world datasets

The objective of this work was to deal with a real-world application. Every day the company gets a dataset composed of 40 to 50 images. The size of patterns is always equal to $88 \times 59\ cm^2$. The spacing between two side by side images must be equal to 1.6 cm. Thus, we added 0.8 cm to height and width of each image. The costs are $C_{su} = 20\$$ and $C_{ss} = 1\$$. Parameters for this calculation are: NbT = 40, NbM = 400000, the probability to accept some "bad" solutions at the beginning = 0.75, $\mu = 0.9$ and

Algorithm 3 SA-2CSP-S

```
1:  function SA-2CSP-S
2:      Pretreatments()                                   ▷ compute pretreatments (L_0, ...)
3:      BestKnown ← new Solution()                        ▷ create an empty solution
4:      L_1 ← NbPatternsOf(InitialSolution(L_0))          ▷ extract L_1
5:      for m ← L_1 to (L_1 + 2) do                       ▷ m = number of patterns
6:          s ← InitialSolution(m)
7:          BestSol ← s
8:          T ← InitializeTemperature()
9:          for k ← 1 to NbT do                           ▷ NbT changes of temperature
10:             for l ← 1 to NbM do                       ▷ NbM moves at temperature T
11:                 s' ← ChooseNeighbor(s)
12:                 if s' != null then
13:                     ComputePrintsAndFitness(s')
14:                     Δ ← f(s') − f(s)
15:                     if Δ ≤ 0 then
16:                         s ← BestSolution(s,s')        ▷ s ← s' if Δ < 0
17:                         BestSol ← BestSolution(s,BestSol)
18:                     else
19:                         p ← random(0, 1)
20:                         if p < e^(−Δ/T) then s ← s'
21:                         end if
22:                     end if
23:                 end if
24:             end for
25:             T ← μT                                    ▷ temperature decrease, with μ < 1
26:         end for
27:         HillClimbing(BestSol)                         ▷ Local improvement
28:         DeleteOverproduction(BestSol)                 ▷ Delete overproduction
29:         BestKnown ← BestSolution(BestKnown,BestSol)
30:     end for
31:     return BestKnown
32: end function
```

`NbWalk` = 200000 (random walk parameter in *InitialSolution* method). These parameters have been obtained from some experimental tests and analysis.

In the company, creation of patterns (choice and placement of images) is done daily by an employee by hand taking more than one hour. We tested our algorithm on 20 datasets of the company and our results were consistently better than those from their existing method in terms of costs. In 100% of cases, these better results are obtained by reducing the number of stock sheets printed, and in 20% of cases by increasing the number of patterns. The mean gain of stock sheets printed was equal to 20.8%, and the mean percentage of cost improvement was equal to 17.4%.

Overall, the three main advantages are the automation of the process compared to the existing time consuming human efforts, the reduction of the manufacturing time and the global cost reduction.

4.2 Artificial datasets

The best evaluation of our algorithm was realized in the previous section on some real datasets of the Seripress company. However, it would be interesting to compare our algorithm to those already published in the literature. The difficulty is to perform a real comparison because most of other studies deal with some slightly different problems (in one-dimension, with a fixed orientation, with no guillotine

constraint...). Moreover, papers dealing with a similar problem to ours do not compare their results to other algorithms. Indeed, they build a kind of "best" solution, which is never displayed in their papers, and calculate the gap between this solution and solutions given by their algorithm. The "best" solution is either a lower bound, which is not relevant in 2CSP-S problem, or a solution calculated by their own algorithm during a very long time (2, 3 or 4 hours of calculations).

We uploaded some datasets created by [10] and tested our algorithm on these datasets. These datasets are composed of four different numbers of images (20, 30, 40 and 50 images), two different types of demands (type S randomly taken from $[1, 25]$ and type L taken from $[100, 200]$), and two different pattern sizes (α for 1400×700 and δ for 2800×1400). Figure 2 displays a solution given by our algorithm from the dataset named 50Lδ, with the same parameters as the previous one (all results of this paper are available from *http://stephanebonnevay.kyvos.net/download/2cspsResults.zip*).

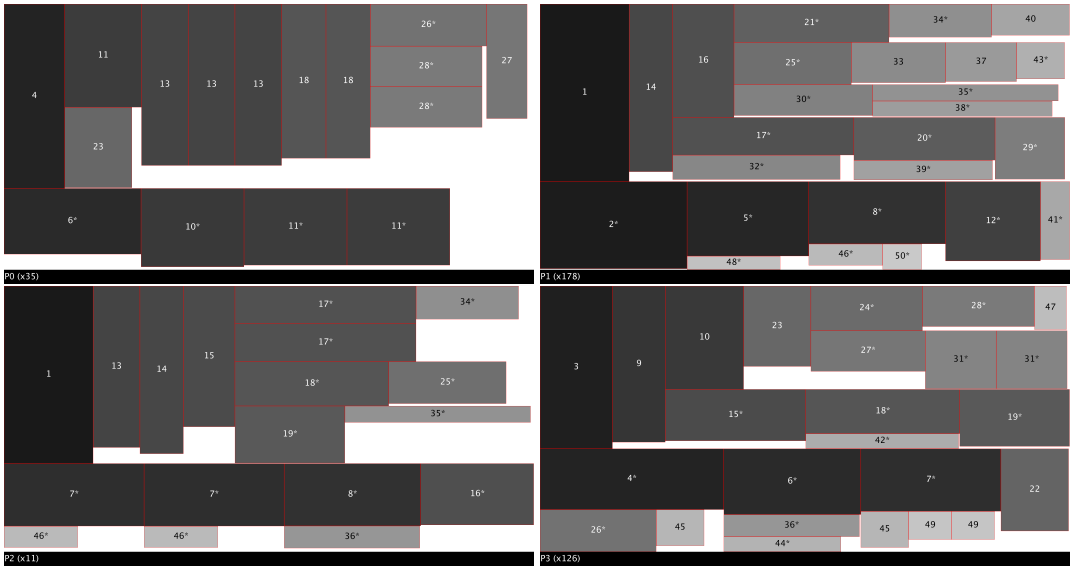


Figure 2: Best solution found by our algorithm (a total of 350 stock sheet printed) from the dataset 50Lδ.

Computational results are summarized in Table 2. Our algorithm was applied 10 times to each dataset. For each dataset, three results are displayed corresponding to the three different numbers of patterns (column m). The first number of patterns is equal to L_1 (the lower bound of the patterns number according to a feasible solution). Column *Average prints* corresponds to the average number of stock sheets printed, and column *Average time* corresponds to the average CPU time in seconds. Number of stock sheets printed of the best solution found among the ten runs is given by the column *Best prints*, and the corresponding global cost is calculated in the next column with $C_{su} = 20\$$ and $C_{ss} = 1\$$. Gray cells correspond to best global solutions. Even though the loop in line 5 of the Algorithm 3 is parallelized, the average computation time of each number of patterns is reported in Table 2.

According to each dataset, the number of stock sheets printed decreases along with the number of patterns. Similarly, the computation time increases along with the number of patterns, because the number of feasible neighbors grows along with the number of patterns. The average number of stock sheets printed is always close to those of the best solution, thus, even though it is a non deterministic process, our algorithm is fairly stable. For datasets with some small demands (type S), the best solutions (gray cells) are always those with the minimum number of patterns because in our case the setup cost of one pattern ($C_{su} = 20\$$) is greater than the difference between two total numbers of stock sheets printed (for two consecutive solutions). On the other hand, for datasets with some large demands (type L), the gap between prints of solutions is larger than 20, thus the best solution has not always been obtained with the minimum number of patterns. The more demands there are, the more the addition of a pattern influences the best solution. As we fix the number of patterns at each calculation, the best solution depends only on the number of stock sheets printed. Then, if the setup cost changes, it is not useful to recalculate solutions, just modify the global cost of solutions and choose the best one.

Table 2: Results from some artificial datasets given by [10].

		Average		Best				Average		Best	
Data	**m**	**prints**	**time**	**prints**	**cost**	**Data**	**m**	**prints**	**time**	**prints**	**cost**
20Sα	4	63.0	97	**63**	143$	20Lα	4	540.6	135	**526**	606$
20Sα	5	58.1	181	**58**	158$	20Lα	5	508.6	215	**500**	600$
20Sα	6	56.3	243	**56**	176$	20Lα	6	498.6	283	**486**	606$
30Sα	6	83.7	174	**82**	202$	30Lα	8	1252.0	235	**1252**	1412$
30Sα	7	80.9	267	**79**	219$	30Lα	9	1244.0	309	**1244**	1424$
30Sα	8	79.6	346	**78**	238$	30Lα	10	1244.0	380	**1244**	1444$
40Sα	8	104.0	205	**101**	261$	40Lα	8	1254.5	115	**1247**	1407$
40Sα	9	100.5	260	**99**	279$	40Lα	9	1228.2	209	**1213**	1393$
40Sα	10	100.0	359	**97**	297$	40Lα	10	1209.5	297	**1186**	1386$
50Sα	9	126.7	204	**125**	305$	50Lα	9	1470.3	121	**1447**	1627$
50Sα	10	124.7	323	**119**	319$	50Lα	10	1441.0	222	**1431**	1631$
50Sα	11	121.2	441	**119**	339$	50Lα	11	1422.7	325	**1405**	1625$
20Sδ	1	22.0	85	**22**	42$	20Lδ	1	179.0	120	**179**	199$
20Sδ	2	13.8	446	**13**	53$	20Lδ	2	131.4	443	**126**	166$
20Sδ	3	13.0	522	**12**	72$	20Lδ	3	132.4	534	**121**	181$
30Sδ	2	26.0	458	**23**	63$	30Lδ	2	280.0	284	**272**	312$
30Sδ	3	21.0	557	**20**	80$	30Lδ	3	247.4	484	**244**	304$
30Sδ	4	22.0	678	**21**	101$	30Lδ	4	253.2	604	**242**	322$
40Sδ	2	35.1	293	**33**	73$	40Lδ	2	346.5	230	**340**	380$
40Sδ	3	29.6	490	**27**	87$	40Lδ	3	314.1	407	**311**	371$
40Sδ	4	27.7	649	**26**	106$	40Lδ	4	307.9	548	**295**	375$
50Sδ	3	36.2	542	**34**	94$	50Lδ	3	403.3	459	**373**	434$
50Sδ	4	34.6	721	**32**	112$	50Lδ	4	361.9	689	**350**	430$
50Sδ	5	34.8	877	**33**	133$	50Lδ	5	367.6	789	**346**	446$

5 Conclusion

In this work, a simulated annealing algorithm was developed, combined with different other techniques and heuristics, especially designed to solve the Two-Dimensional Cutting Stock Problem with Setup Cost (2CSP-S). Our algorithm was applied to real-world applications in paper industry. The obtained results are very significant for the company: they improve the manufacturing time and the global cost. Moreover, it is an anytime algorithm which is an important issue for the company.

Some works, which attempt to improve the computation time and the quality of solutions, are currently in progress. For example, in this work, we used the same parameters for all experiments, but it will be interesting to adapt the number of temperature changes `NbT` and the number of moves `NbM` according to the number of patterns because a large number of patterns increases the number of feasible solutions in a neighborhood. Moreover, a genetic algorithm is currently tested to generate solutions without a fixed number of patterns; this process automatically find the best number of patterns.

The use of some better 2-D Bin Packing algorithms should improve the placement process and thus improve the quality of the final solution, but it should increase the computation time too.

Finally, the Seripress company has some new colorimetric constraints: images with same color must be set side by side. We are going to adapt our algorithm to take into account these new constraints.

6 Acknowledgments

We would like to acknowledge the Seripress company for this collaboration.

References

[1] A. Aloisio, C. Arbib, and F. Marinelli. On lp relaxations for the pattern minimization problem. *Networks*, 57(3):247–253, 2011.

[2] G. Belov and G. Scheithauer. A branch-and-cut-and-price algorithm for one-dimensional stock cutting and two-dimensional two-stage cutting. *EJOR*, 171(1):85–106, 2006.

[3] G. Belov and G. Scheithauer. Setup and open-stacks minimization in one-dimensional stock cutting. *INFORMS Journal on Computing*, 19(1):27–35, 2007.

[4] H. Ben Amor and J. Valerio de Carvalho. Cutting stock problems. In G. Desaulniers, J. Desrosiers, and M. Solomon, editors, *Column Generation*, pages 131–161. Springer US, 2005.

[5] A. Bortfeldt. A genetic algorithm for the two-dimensional strip packing problem with rectangular pieces. *EJOR*, 172:814–837, 2006.

[6] Y. Cui, X. Zhao, Y. Yang, and P. Yu. A heuristic for the one-dimensional cutting stock problem with pattern reduction. *Proc. of the Institution of Mechanical Engineers*, 222(6):677–685, 2008.

[7] S.P. Fekete and J. Schepers. On more-dimensional packing i: Modeling, 2000.

[8] R.W. Haessler. One-dimensional cutting stock problems and solution procedures. *Mathematical and Computer Modeling*, 16(1):1–8, 1992.

[9] E. Hopper and Turton B.C.H. An empirical investigation of meta-heuristic and heuristic algorithms for a 2d packing problem. *EJOR*, 128:34–57, 2001.

[10] S. Imahori, M. Yagiura, S. Umetani, S. Adachi, and T. Ibaraki. Local search algorithms for the two-dimensional cutting stock problem with a given number of different patterns. In *Metaheuristics International Conference, Lyon (France)*, volume 35, pages 1–6, september 2003.

[11] J. Jylanki. A thousand ways to pack the bin - a practical approach to two-dimensional rectangle bin packing. research report, 2010.

[12] J. Kallrath, S. Rebennack, J. Kallrath, and R. Kusche. Solving real-world cutting stock-problems in the paper industry: Mathematical approaches, experience and challenges. *European Journal of Operational Research*, 238(1):374–389, 2014.

[13] A. Lodi, S. Martello, and D. Vigo. Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems. *INFORMS Journal on Computing*, 11(4):345–357, 1999.

[14] C. McDiarmid. Pattern minimisation in cutting stock problems. *Discrete Appl. Math.*, 98(1-2):121–130, 1999.

[15] A. Mobasher and A. Ekici. Solution approaches for the cutting stock problem with setup cost. *Computers and OR*, 40(1):225–235, 2013.

[16] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani. Vlsi module placement based on rectangle-packing by the sequence pair. *IEEE TRANS. ON CAD*, 15(12):1518–1524, 1996.

[17] B. Preas and W.M. van Cleemput. Placement algorithms for arbitrarily shaped blocks. In D.W. Hightower, editor, *DAC*, pages 474–480. ACM, 1979.

[18] F. Vanderbeck. Exact algorithm for minimizing the number of setups in the one-dimensional cutting stock problem. *Operations Research*, 45(5):915–926, 2000.

[19] G. Wäscher, H. Haussner, and H. Schumann. An improved typology of cutting and packing problems. *European Journal of Operational Research*, 183(3):1109–1130, 2007.