A Genetic Algorithm to Solve a Real 2-D Cutting Stock Problem with Setup Cost in the Paper Industry

Stéphane Bonnevay
ERIC, Université de Lyon
5 av. Pierre Mendès France
69676 Bron Cedex, France

Philippe Aubertin
AXOPEN
149 bd Stalingrad
69100 Villeurbanne, France

Gérald Gavin
ERIC, Université de Lyon
5 av. Pierre Mendès France
69676 Bron Cedex, France

stephane.bonnevay@univ-lyon1.fr

ABSTRACT

This paper deals with the Two-Dimensional Cutting Stock Problem with Setup Cost (2CSP-S). This problem is composed of 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). In this article, we solve the 2CSP-S focusing on this third sub-problem. A genetic algorithm was developed to automatically find the proper number of each image on patterns. It is important to notice that our approach is not a new packing technique. This work was conducted for a paper industry company and experiments were realized on real and artificial datasets.

Keywords

Two-dimensional cutting stock problem, Setup cost, Combinatorial optimization, Genetic algorithms, Paper industry.

1. INTRODUCTION

The classical Cutting Stock Problem (CSP) deals with the problem of cutting stock materials (paper, steel, glass, etc.) in order to satisfy the customer demand while minimizing the trim loss [13, 4, 2, 19, 12]. The Bin Packing Problem, where a set of items must be grouped into bins, and the CSP belong to the same family of problems. In literature, several methods were developed in order to solve this couple of problems, including linear programming, heuristics, and meta-heuristics [2, 4, 5, 9, 13, 19, 12]. In CSP, a sufficient number of setups/patterns are used to solve the problem. 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) tries to minimize the number of different cutting patterns while satisfying the demand [1, 14]. The Cutting Stock Problem with Setup Cost (CSP-S) takes into account the cost of the setup and the cost of the stock sheet printed in order to minimize the total production cost [10, 15, 18]. Most of existing papers deal with one dimensional problems [8, 3, 6].

In this paper, we give a solution to the Two-Dimensional Cutting Stock Problem with Setup Cost (2CSP-S). In this problem, customer demands are the number of prints of several rectangular images. The goal consists of satisfying 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. Here, we do not solve a 2-D bin cutting (or packing) problem (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 objective is to find the best number of each image on each pattern (and to find the optimal number of patterns). For this purpose, we developed a genetic algorithm combined with different other techniques and heuristics. 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 the mathematical formulation of the 2CSP-S and sets out an example case as way of explanation. Our algorithm with all specific methods are detailed in Section 3. Results are presented in Section 4. This work was conducted for the Seripress[1] company, which is a paper industry company.

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

[1] http://www.seripress.com/

GECCO '15, July 11 - 16, 2015, Madrid, Spain

DOI: http://dx.doi.org/10.1145/2739480.2754660

exact position and rotation information). The set 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:

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

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:

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

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

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.

In order to take into account the overproduction in our model, we define r_i as the quantity of images i overproduced and $Over\mathsf{P}$ the total overproduction:

$$r_i = \sum_{j \in J} p_{ij} x_j - d_i \ \forall i \in I \text{ and } Over\mathsf{P} = \sum_{i \in I} r_i \quad (3)$$

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 $Over\mathsf{P}$ 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 $Over\mathsf{P}$.

2.2 Example

In order to illustrate the 2CSP-S problem, a simple example is displayed with 4 customers demands given in Table 1. The pattern size is equal to 60×40.

Table 1: Width, height and demand of the 4 images.

Image	Height	Width	Demand
1	30	24	246
2	56	13	562
3	22	14	1000
4	23	9	3498
pattern	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 one pattern ($m = 1$) and with two patterns ($m = 2$). Results are displayed in Figures 1 and 2. In all the figures of this paper, a star (*) next to the image number indicates that the corresponding image was rotated 90°. As shown in Table 2, the best solution is got with two patterns (see Figure 2) even if the manufacturing cost of one setup is 20 times greater than the printing cost of one stock sheet.

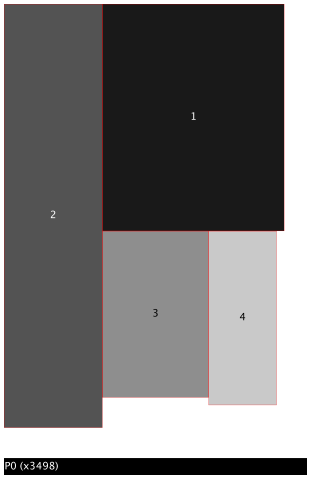

Figure 1: Solution with 1 pattern (3498 prints).

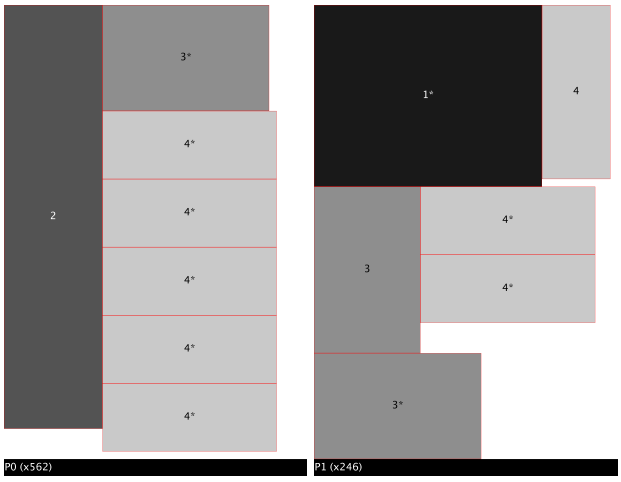

Figure 2: Solution with 2 patterns (246 + 562 = 808 prints).

Table 2: Costs of solutions of Figures 1 and 2

Nb patterns	Total cost
1	$1 \times 20\$ + 3498 \times 1\$ = 3518\$$
2	$2 \times 20\$ + 808 \times 1\$ = 848\$$

It is important to understand that the best result is not achieved by fully completing all the patterns. This means that we cannot use the blank area rate to define the quality of a solution or the quality of a pattern (unlike what is done in [5] to select parts of solutions in their genetic algorithm). According to the 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 two patterns, only 54 copies of the image 3 and 50 copies of the image 4 are overly printed.

2.3 Bounds

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

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

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, $m \geq L_0$.

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

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

$L_{max}(i)$ 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:

i) 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,

ii) 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,

iii) a combinatorial problem which consists of finding the proper tuple $(p_1, \ldots, p_m)$ where $p_j = (p_{j1}, \ldots, p_{jn})$ and p_{ji} is the number of image i in the pattern j (and find the number of patterns m).

In this paper, we present our genetic algorithm used to solve the sub-problem iii): 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 sub-problems i) and ii). The following subsections detail each part of our genetic algorithm to solve the sub-problem iii).

One important constraint of this work is the limitation of the computation time. Indeed, this work was developed in order to be applied to a real-world problem in the paper industry. Therefore, in the following we always took into account this time constraint in our choices. By construction, our algorithm has a very small memory footprint.

3.1 Stock sheets prints

The fitness, $f(s)$, of a solution $s \in S$ is a combination of the total cost given by Equation 1 and overproduction quantities. In order to compute this fitness, the number of stock sheets to be printed $(x_1, \ldots, x_m)$ have to be computed. 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 genetic algorithm), the aim is to find $(x_1, \ldots, x_m)$ minimizing Equation 1 under constraints given by Equation 2.

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, the number of stock sheets to be printed $x_j = \lceil x_j^* \rceil \ \forall j \in J$.

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 by $cpaf(n, m)$.

3.2 2-D bin packing algorithm

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, this method is called *2DBinPacking*(q), where $q = (q_1, \ldots, q_n)$ 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. As, this algorithm is deterministic, a tuple $(q_1, \ldots, q_n)$ always corresponds to the same placement (and then, on the same number of patterns).

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

3.3 Neighborhood of a solution

To create some random initial solutions, to build the mutation operator or to improve a solution with an hill-climbing process, we define a neighborhood of any solution s by modifying the tuple $(p_1, \ldots, p_m)$.

Let $s \in S$ be a solution composed of a vector $(\pi_1, \ldots, \pi_m)$ according to a tuple $(p_1, \ldots, p_m)$, and a vector $(x_1, \ldots, x_m)$ which is the solution of the relaxed linear program (see Section 3.1). A neighbor of s is built from one of these four elementary operators: *add*, *remove*, *move* and *swap*. The operator *add* adds one image i in the pattern j involves incrementing p_{ij} if p_{ij} is less than $L_{max}(i)$. The operator *remove* consists of removing one image i from the pattern j involves decrementing p_{ij} if the total number of image i in the whole solution s remains strictly positive. The operator *move* moves one image from a pattern to another one, and the operator *swap* swaps two images from two different patterns.

A neighbor of a solution 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*(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.

The time complexity of this method only depends on the *2DBinPacking* complexity.

3.4 Initial solution generator

To create the initial population, some feasible solutions have to be generated. The first step of our process is inspired

by [10]. The objective is to place only one copy of each image i on k patterns. Thus, the method *2DBinPacking*$(q_1, \ldots, q_n)$ is run with $q_i = 1\ \forall i \in I$. The objective is to place only one copy of each image i in the solution. The method returns a feasible placement $(\pi_1, \ldots, \pi_k)$. An initial feasible solution s_0 is created from this placement. Then, from s_0, a random walk, *RandomWalk*(s_0), 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`.

A method *CreateInitialPopulation*() returns a list of $NbPop$ random initial solutions. The time complexity is $O(\texttt{NbPop} \times (\texttt{NbWalk} \times n_{max}^2 + cpaf(n,m)))$, where n_{max} is the total number of images in q during the random walk.

3.5 Main algorithm

Our algorithm, *GA-2CSP-S* (see Algorithm 1), combines all the previous methods: the *Crossover* and the *Mutation* operators (see Sections 3.6 and 3.7) in a classical genetic algorithm.

There are `NbGen` generations (loop line 5). The population at the k^{th} generation is denoted by P_k. Each generation is composed of `NbPop` solutions.

Line 6, the *RouletteWheelReproduction*(P_{k-1}) method reproduces a population composed of `NbPop` solutions according to the classical fitness proportionate selection (roulette-wheel selection). Thus, it returns an intermediate population $P^\star_{k-1}$ selected from P_{k-1}. In order to create a new generation P_k, the `NbBest` best solutions of P_{k-1} are first reproduced (elitist strategy) in P_k (line 7). Then P_k is filled up to the full size by generating solutions either through our crossover method *Crossover*$(P^\star_{k-1})$ (line 10) or our mutation method *Mutation*$(P^\star_{k-1})$ (line 12). *Crossover* and *Mutation* operators are detailed in Sections 3.6 and 3.7. At each step, the best found solution is updated from the set of all generated solutions (line 15).

Algorithm 1 GA-2CSP-S

```
 1: function GA-2CSP-S
 2:     BestKnown ← new Solution()
 3:     P_0 ← CreateInitialPopulation()
 4:     BestKnown ← UpdateBestSolution(P_0)
 5:     for k ← 1 to NbGen do
 6:         P*_{k-1} ← RouletteWheelReproduction(P_{k-1})
 7:         P_k ← BestSolutionsReproduction(NbBest, P_{k-1})
 8:         for i ← NbBest+1 to NbPop do
 9:             if Random(0,1) < ProbaCross then
10:                 add Crossover(P*_{k-1}) to P_k
11:             else
12:                 add Mutation(P*_{k-1}) to P_k
13:             end if
14:         end for
15:         BestKnown ← UpdateBestSolution(P_k)
16:     end for
17:     HillClimbing(BestKnown)
18:     DeleteOverproduction(BestKnown)
19:     return BestKnown
20: end function
```

At the end of this algorithm, a local improvement is applied on the best found solution with a hill-climbing process followed by an overproduction removal (line 18) from the population $P^\star_{k-1}$.

This *HillClimbing*(s) method locally improves a solution s. This algorithm searches the best neighbor s' of s; if s' is better than s, it repeats this search from s' and so on, until no further improvements can be found. Instead of building all the neighbors of the curent solution, we randomly generate a given number of neighbors in the neighborhood of the current solution and select the best one. This method is used in Algorithm 1 line 17 and in our crossover and mutation operators.

The *DeleteOverproduction*(s) method was developed in order to delete some overproduced images i of s (where $r_i > 0$, see Equation 3) which do not affect the computation of stock sheet prints (see Section 3.1). An image i can be deleted on a pattern j if $r_i \geq x_j$.

According to the complexities of *Crossover* and *Mutation* operators (see Sections 3.6 and 3.7), the time complexity of our main algorithm is equal to $O(\texttt{NbGen} \times \texttt{NbPop} \times (n_{max}^2 + cpaf(n,m)))$.

3.6 Crossover operator

The crossover method, *Crossover*(P) (described by Algorithm 2), builds a new offspring from two different parent solutions (lines 3-4). At first the offspring is initialized to empty (0 pattern). A subset of patterns is randomly selected in the first parent and another one in the second parent (lines 5-6). The number of patterns selected in each parent is a random number, it is a percentage of the parents' patterns numbers, taken in $[25\%, 50\%]$. These patterns are transferred in the offspring (line 7). As explained in Section 2.2, the choice of patterns in parents was realized as in [5] because the blank area rate can not be used as a quality of a pattern. Then, the offspring is completed with the help of the *2DBinPacking*(q) with $q_i = 1$ for all missing items i (line 8).

The total offspring's patterns number could be different from its parents, then a larger part of space solutions could be explored.

Algorithm 2 Crossover process

```
 1: function CROSSOVER(P)
 2:     OffSpring ← new Solution()
 3:     Parent1 ← choose a solution ∈ P
 4:     Parent2 ← choose a different solution ∈ P
 5:     SetPattern1 ← choose some patterns ∈ Parent1
 6:     SetPattern2 ← choose some patterns ∈ Parent2
 7:     add SetPattern1 ∪ SetPattern2 to OffSpring
 8:     complete OffSpring with 2DBinPacking(q)
            ▷ with q_i = 1 for all missing item i in OffSpring
 9:     HillClimbing(OffSpring)
10:     return OffSpring
11: end function
```

The complexity of *Crossover*(P) is $O(n_q m + cpaf(n,m))$ (where $n_q = \sum_{i \in I} q_i$).

3.7 Mutation operator

The method *Mutation*(P) (see Algorithm 3) selects a parent at random in the population P (line 3) and applies a random walk from this parent (line 4) as done in Section 3.4 when generating initial solutions.

The complexity of *Mutation*(P) is $O(\texttt{NbWalk} \times n_{max}^2 + cpaf(n,m))$.

Algorithm 3 Mutation process

```
1: function MUTATION(P)
2:     Offspring ← new Solution()
3:     Parent ← choose a solution ∈ P
4:     Offspring ← RandomWalk(Parent)
5:     HillClimbing(Offspring)
6:     return Offspring
7: end function
```

4. EXPERIMENTS

A *Java* application was developed, using a Java Simplex Solver package to compute the number of stock sheets (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 $O(cpaf(n,m))$ and it is equal to $O(\texttt{NbGen} \times \texttt{NbPop} \times cpaf(n,m))$. Computations were run on a MacBook Pro 2.2GHz Intel Quad Core i7 16Go.

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

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 88×59 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 $C_{ss} = 1\$$. An example of this kind of data, composed of 40 images, is displayed in Table 3.

Table 3: Example of a daily dataset of the company.

i	h_i	w_i	d_i	i	h_i	w_i	d_i
1	13.2	8.9	62	21	13.2	8.9	530
2	13.2	8.9	31	22	19.4	13.2	100
3	18.5	14.9	500	23	40.4	28.1	80
4	40.4	28.1	10	24	28.1	19.4	25
5	21.9	17.2	250	25	40.4	28.1	31
6	4.9	2.0	250	26	31.0	22.0	400
7	13.2	8.9	200	27	40.4	28.1	10
8	40.4	28.1	45	28	40.4	28.1	10
9	20.0	18.2	500	29	40.4	28.1	10
10	28.1	19.4	10	30	19.4	13.2	110
11	28.1	19.4	45	31	40.4	28.1	25
12	19.4	13.2	160	32	40.4	28.1	50
13	22.7	22.0	100	33	40.4	28.1	20
14	9.0	5.0	10	34	33.0	22.0	74
15	13.2	8.9	25	35	20.0	4.7	510
16	13.2	8.9	25	36	28.1	19.4	300
17	13.2	8.9	25	37	13.2	8.9	120
18	13.2	8.9	25	38	28.1	19.4	15
19	13.2	8.9	25	39	13.2	8.9	35
20	13.2	8.9	50	40	40.4	28.1	100
				pattern	88.0	59.0	

In the company, creation of patterns (choice and placement of images) is done daily by an employee by hand taking more than one hour. From the dataset of Table 3, the solution found by the company was composed of 5 patterns and required 815 stock sheets printed. The total cost is equal to 915$. From the same dataset, our algorithm took 505 seconds to automatically find a solution with 6 patterns but with only 493 stock sheets printed. In our case, the total cost is equal to 613$, which is a gain of 33.0% (see Figure 3).

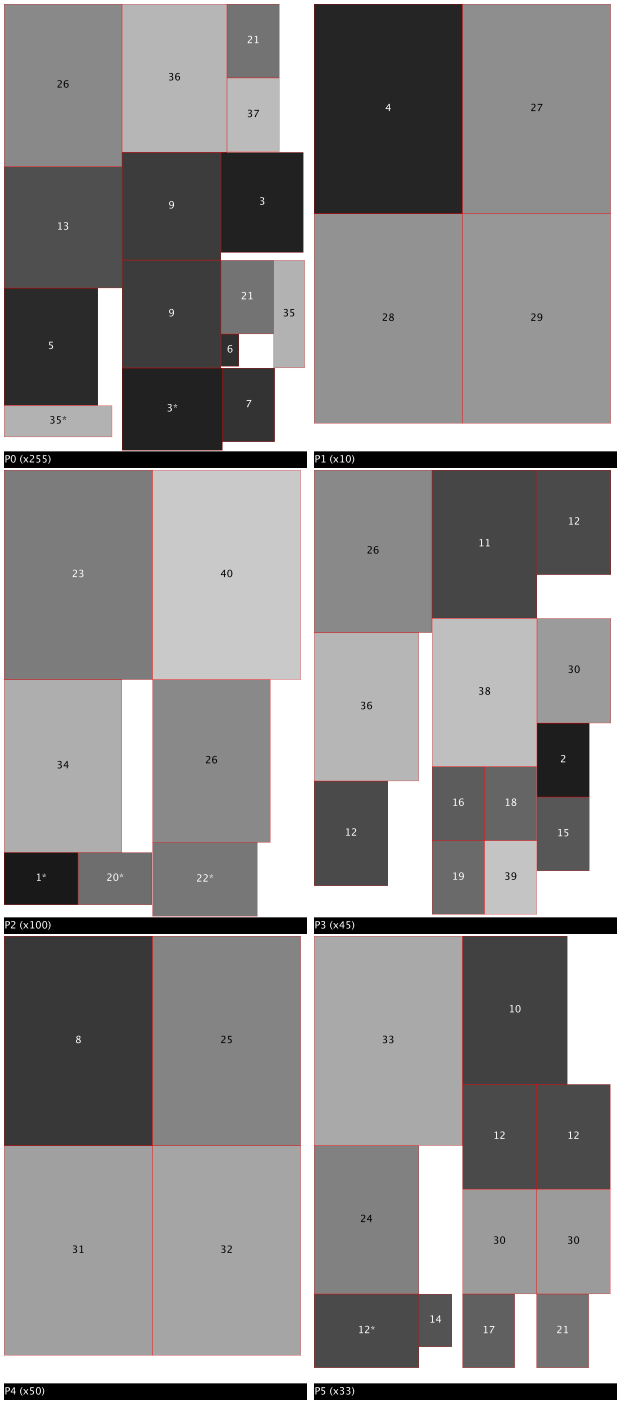

Figure 3: Best solution found by our algorithm from the dataset displayed in Table 3: 6 patterns and a total of 493 stock sheets printed (total cost: 613$).

Parameters for this calculation are: `NbGen` = 30, `NbPop` = 60, `NbBest` = 10, `ProbaCross` = 0.75 and `NbWalk` = 10000. These parameters were obtained from some experimental tests. For example, these tests showed that a large population was better than a higher number of generations.

We tested our algorithm on 20 datasets of the company and our results were consistently better (in terms of costs) than those from its currently used method. In 100% of cases, these better results are obtained by reducing the number of stock sheets printed, and in 20% of cases by modifying the number of patterns. The mean gain of stock sheets printed was equal to 26.1%, and the mean percentage of cost improvement was equal to 22.2%.

Overall, the three main advantages are: the automatization of the process, the reduction of the manufacturing time and the global cost reduction.

4.2 Artificial datasets

The best evaluation of our algorithm was presented 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, in these papers, a kind of "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). According to this "best" solution, authors calculate the gap between this solution and solutions given by their algorithm. Thus, no comparison could be performed.

However, we uploaded some datasets created by [10] and we tested our algorithm on these datasets. These datasets are composed of four different numbers of images (20, 30, 40 and 50 images), three different types of demands (type S randomly taken from $[1, 25]$, type L taken from $[100, 200]$ and type V taken from either $[1, 25]$ or $[100, 200]$) and two different pattern size (α for 1400×700 and δ for 2800×1400).

Table 4: Dataset 30Sα created by [10].

i	h_i	w_i	d_i	i	h_i	w_i	d_i
1	894	417	25	16	344	318	7
2	763	433	13	17	385	263	8
3	624	430	19	18	260	380	17
4	949	262	2	19	778	118	3
5	778	318	22	20	663	126	5
6	593	415	23	21	158	489	1
7	960	256	7	22	810	83	20
8	907	248	17	23	248	229	5
9	462	468	4	24	259	179	6
10	873	234	15	25	233	177	1
11	595	333	16	26	143	281	10
12	673	246	23	27	384	87	15
13	844	185	12	28	165	173	11
14	894	174	23	29	162	175	24
15	422	323	24	30	208	98	24
				pattern	1400	700	

Figure 4 displays a solution given by our algorithm from the dataset named 30Sα (see Table 4), with the same parameters as the previous ones.

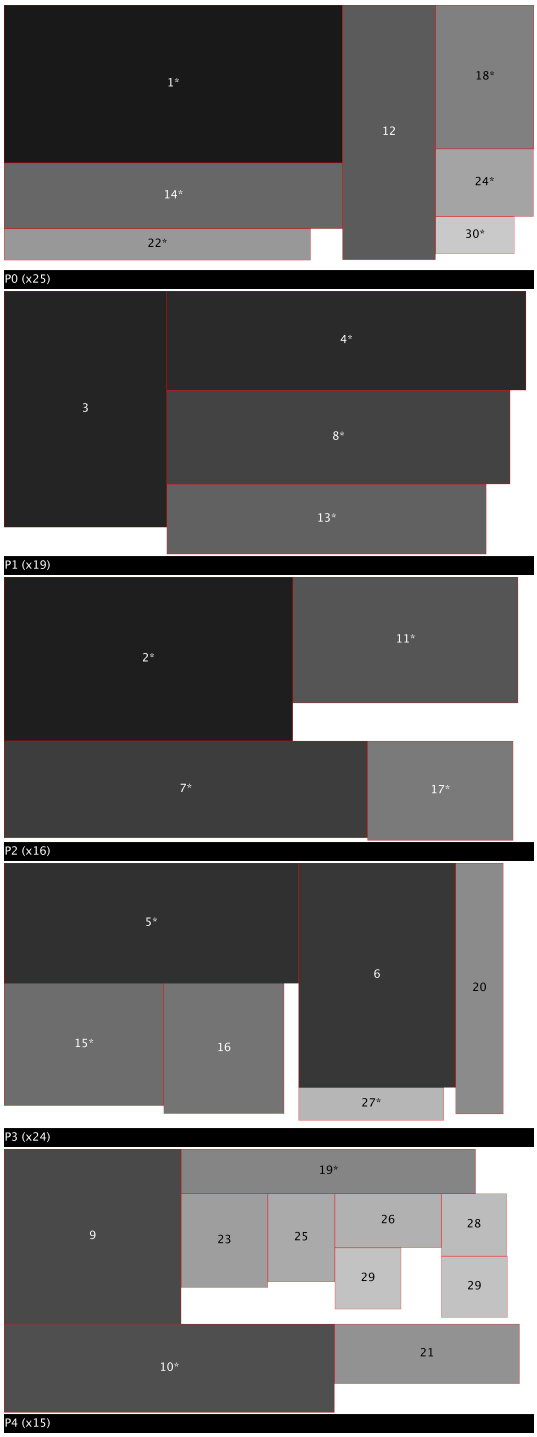

Figure 4: Best solution found by our algorithm from the dataset 30Sα: 5 patterns and a total of 99 stock sheets printed (total cost: 199$).

The averages of running time are summarized in the left side of Table 5. We applied our algorithm 10 times to each dataset. For each dataset, the theoretical lower bound L_0,

the average number of patterns (column m), the average number of stock sheets printed (column prints), the average total cost (column cost) and the average CPU time in seconds (column time) are displayed. As in the previous subsection, the manufacturing cost of one setup C_{ss} is equal to 1$ and the printing cost of one stock sheet C_{su} is equal to 20$. The best solutions found from these 10 runs on each dataset are displayed in right side of Table 5.

We can logically observe that as the number of images increases (from 20 to 50), m becomes larger and the computational time increases.

For datasets with small demands (type S) the total numbers of stock sheets printed are small in regard to the setup cost of one pattern ($C_{su} = 20$$), thus adding a pattern penalizes the total cost of solutions. That is the reason why solutions are composed of a small number of patterns. The gap between the setup cost of one pattern ($C_{su} = 20$$) increases with the total demand of datasets of type V and L. The more demands there are, the more the addition of a pattern influences the best solution (see Figure 2). For datasets with a large pattern size (type δ) the number of patterns is logically smaller than for datasets with a smaller pattern size (type α). Even if the number of patterns is lower, the running time is greater because the number of images on patterns of type δ is much larger than the number of images in the small patterns (type α).

The average values (m, prints, cost and time) are always close to those of the best solution; thus, even if it is a non-deterministic process, our algorithm is fairly stable.

As explained at the begin of this section, it is difficult to compare our results to existing ones. It is all the more so difficult as the results depend on the prices of the manufacturing cost of one setup C_{ss} and the printing cost of one stock sheet C_{su}. Figure 5 displays the influence of prices on the average number of patterns computes from 10 runs on the dataset 30Vα (the price of the printing cost of one stock sheet C_{su} is constant and equal to 1$). It shows that the number of patterns of the best found solution decreases (from 9.2 to 7.2) when the price of the manufacturing cost of one setup increases (from 1$ to 25$). A decrease of m is observed with all the datasets of this study.

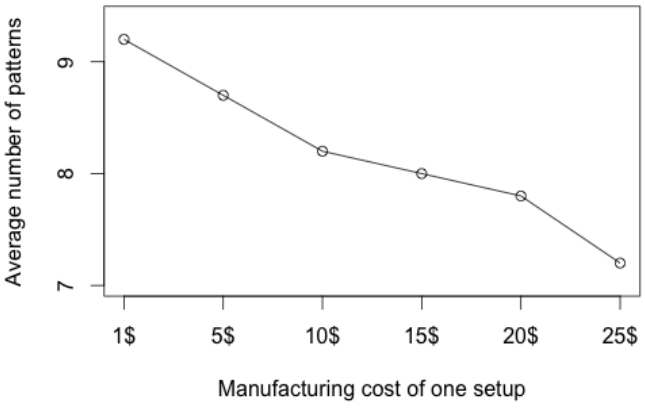

Figure 5: The influence of the manufacturing cost of one setup (C_{ss}) on the average number of patterns. These average numbers are obtained from 10 runs on the dataset 30Vα (C_{ss} is constant and equal to 1$).

5. CONCLUSION

We developed a genetic algorithm, combined with several other techniques and heuristics, especially designed to deal with the Two-Dimensional Cutting Stock Problem with Setup Cost. Our algorithm was applied to real-world applications in paper industry. The obtained results are very significant and very useful for the company. They improve the manufacturing time (solutions are automatically built in few minutes) and the global cost. Moreover, it is an algorithm which is able to return a feasible solution at any time of the process.

Some additional work is currently in progress. These works attempt to improve the computation time and the quality of solutions. 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.

In the crossover process developed in [5], patterns (layers) are selected according to the filling rate. As the filling rate makes no sense in 2CSP-S, patterns are randomly selected in our *Crossover* operator. To overcome this, a new quality definition of a pattern or of a subset of patterns should be defined to improve our algorithm.

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.

Finally, some parallelization processes could be added in the main algorithm. For example, initial solutions and offsprings could be generated in parallel in order to decrease the running time.

6. ACKNOWLEDGMENTS

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

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

[8] R. Haessler. One-dimensional cutting stock problems

Table 5: Results from artificial datasets given by [10].

		Average				Best		
Data	L_0	**m**	**prints**	**cost**	**time**	**m**	**prints**	**cost**
20Sα	4	4.0	63.0	143.0 $	53.4	4	63	143 $
30Sα	5	5.0	98.4	198.4 $	144.3	5	97	197 $
40Sα	7	7.4	117.6	265.6 $	216.6	7	116	256 $
50Sα	8	9.0	146.4	326.4 $	374.7	9	143	323 $
20Vα	4	5.4	402.0	510.0 $	132.2	5	405	505 $
30Vα	6	7.8	601.4	757.4 $	253.6	8	591	751 $
40Vα	8	10.0	495.0	695.0 $	504.3	10	491	691 $
50Vα	10	11.8	1110.4	1346.4 $	582.9	11	1081	1301 $
20Lα	3	4.2	541.6	625.6 $	161.7	4	528	608 $
30Lα	6	8.0	1252.0	1412.0 $	239.2	8	1252	1412 $
40Lα	7	8.4	1292.8	1460.8 $	221.6	9	1268	1448 $
50Lα	8	9.6	1536.0	1728.0 $	349.0	10	1514	1714 $
20Sδ	1	1.0	22.0	42.0 $	24.1	1	22	42 $
30Sδ	2	2.1	26.8	65.8 $	299.1	2	24	64 $
40Sδ	2	2.3	34.6	74.2 $	311.4	2	33	73 $
50Sδ	2	3.0	38.8	98.4 $	541.2	3	38	98 $
20Vδ	1	1.0	197.0	217.0 $	32.2	1	197	217 $
30Vδ	2	2.0	192.6	232.2 $	286.6	2	191	231 $
40Vδ	2	3.0	132.9	192.9 $	568.4	3	132	192 $
50Vδ	3	3.8	291.2	362.8 $	1046.6	4	284	364 $
20Lδ	1	1.0	179.0	199.0 $	37.7	1	179	199 $
30Lδ	2	2.4	264.0	314.2 $	407.9	3	257	317 $
40Lδ	2	2.5	342.0	382.2 $	399.3	3	322	382 $
50Lδ	2	3.0	396.7	456.0 $	789.1	3	395	455 $

and solution procedures. *Mathematical and Computer Modeling*, 16(1):1–8, 1992.

[9] E. Hopper and T. 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. van Cleemput. Placement algorithms for arbitrarily shaped blocks. In D. 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.