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 optimization 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 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 $p_{ij}$ be the number of image $i$ in the pattern $j$. We note $p_j = (p_{j1}, \ldots, p_{jm})$. Let $\pi_j$ be a placement (rectangle packing) of all the images of the pattern $j$ (\pi_j includes
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 \((p_j, \pi_j)\). In the following of this paper, we shall only consider feasible couples \((p_j, \pi_j)\) (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 \( \pi_j \) \( \forall j \in J \).

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 \( x_j \) \( \forall j \in J \)).

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 \( \text{OverP} \) the total overproduction:

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

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 \( \text{OverP} \) is chosen. Thereby, a method called \( \text{BestSolution}(s, s') \) is used to return the best solution between \( s \) and \( s' \). This method combines \( f \) and \( \text{OverP} \).

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 \( \times \) 40.

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

<table>
<thead>
<tr>
<th>Image</th>
<th>Height</th>
<th>Width</th>
<th>Demand</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>30</td>
<td>24</td>
<td>246</td>
</tr>
<tr>
<td>2</td>
<td>56</td>
<td>13</td>
<td>562</td>
</tr>
<tr>
<td>3</td>
<td>22</td>
<td>14</td>
<td>1000</td>
</tr>
<tr>
<td>4</td>
<td>23</td>
<td>9</td>
<td>3498</td>
</tr>
</tbody>
</table>

The objective is to reduce the total material cost depending exclusively on \( C_{su} \) and \( C_{ss} \) (see Equation 1). Here, \( C_{su} = 208 \) and \( C_{ss} = 18 \). 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 (\( \star \)) next to the image number indicates that the corresponding image was rotated 90\(^\circ\). 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).](image1.png)

![Figure 2: Solution with 2 patterns (246 + 562 = 808 prints).](image2.png)

Table 2: Costs of solutions of Figures 1 and 2

<table>
<thead>
<tr>
<th>Nb patterns</th>
<th>Total cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(1 \times 208 + 3498 \times 1$ = 3518$)</td>
</tr>
<tr>
<td>2</td>
<td>(2 \times 208 + 808 \times 1$ = 848$)</td>
</tr>
</tbody>
</table>

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:

$$L_0 = \left\lfloor \frac{\sum_{i \in I} h_i w_i}{H \times W} \right\rfloor$$

$L_0$ 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 $L_{\text{max}}(i)$ of the number of images $i$ which can be placed on one pattern as:

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

$L_{\text{max}}(i)$ 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 $([x_1^1, \ldots, x_m^1])$ is returned. Thus, the number of stock sheets to be printed $x_j = [x_j^1] \forall j \in J$.

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

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 $\text{2DBinPacking}(q)$, 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 $O(n^2)$ where $n_q = \sum q_i$.

3.3 Neighborhood of a solution

To create the initial population, some feasible solutions are computed (null or if there is no feasible packing).

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 $p_{ji}$ if $p_{ji}$ is less than $L_{\text{max}}(i)$. The operator remove consists of removing one image $i$ from the pattern $j$ involves decrementing $p_{ji}$ if the total number of image $i$ in the whole solution stays 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 $\text{2DBinPacking}(p_j)$ (see Section 3.2) is applied on each modified pattern $j$ in order to check the placement feasibility and to build $\pi_j$ (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 $\text{2DBinPacking}$ 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 \( \text{NbWalk} \).

A method CreateInitialPopulation() returns a list of \( \text{NbPop} \) random initial solutions. The time complexity is \( O(\text{NbPop} \times (\text{NbWalk} \times \text{n}_{\text{max}} + \text{cpaf}(n, m))) \), where \( \text{n}_{\text{max}} \) 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 \( \text{NbGen} \) generations (loop line 5). The population at the \( k \)-th generation is denoted by \( P_k \). Each generation is composed of \( \text{NbPop} \) solutions.

Line 6, the RouletteWheelReproduction\((P_{k-1})\) method reproduces a population composed of \( \text{NbPop} \) solutions according to the classical fitness proportionate selection (roulette-wheel selection). Thus, it returns an intermediate population \( P'_{k-1} \) selected from \( P_{k-1} \). In order to create a new generation \( P_k \), the \( \text{NbBest} \) 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\((P'_{k-1})\) (line 10) or our mutation method Mutation\((P'_{k-1})\) (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 \( \text{NbGen} \) do
6: \( P'_{k-1} \) ← RouletteWheelReproduction\((P_{k-1})\)
7: \( P_k \) ← BestSolutionsReproduction\((\text{NbBest}, P'_{k-1})\)
8: for \( i ← \text{NbBest}+1 \) to \( \text{NbPop} \) do
9: if Random\((0,1) < \text{ProbaCross} \) 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\((\text{BestKnown})\)
18: DeleteOverproduction\((\text{BestKnown})\)
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 \( P'_{k-1} \).

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 current 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 \( O(\text{NbGen} \times \text{NbPop} \times (\text{n}_{\text{max}} + \text{cpaf}(n, m))) \).

#### 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 \( \in P \)
4: Parent2 ← choose a different solution \( \in P \)
5: SetPattern1 ← choose some patterns \( \in \text{Parent1} \)
6: SetPattern2 ← choose some patterns \( \in \text{Parent2} \)
7: add SetPattern1 \u222a SetPattern2 to OffSpring
8: complete OffSpring with 2DBinPacking\((q)\)
9: \( \triangleright \) with \( q_i = 1 \) for all missing item \( i \) in OffSpring
10: HillClimbing\((\text{OffSpring})\)
11: return OffSpring
12: end function

The complexity of Crossover\((P)\) is \( O(n_q m + \text{cpaf}(n, m)) \) (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 \( O(\text{NbWalk} \times \text{n}_{\text{max}} + \text{cpaf}(n, m)) \).
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 \(O(NbGen \times NbPop \times cpaf(n,m))\). 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 \(\times\) 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 \(C_{ss} = 1\). An example of this kind of data, composed of 40 images, is displayed in Table 3.

<table>
<thead>
<tr>
<th>i</th>
<th>(h_i)</th>
<th>(w_i)</th>
<th>(d_i)</th>
<th>i</th>
<th>(h_i)</th>
<th>(w_i)</th>
<th>(d_i)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>13.2</td>
<td>8.9</td>
<td>52</td>
<td>21</td>
<td>13.2</td>
<td>8.9</td>
<td>530</td>
</tr>
<tr>
<td>2</td>
<td>13.2</td>
<td>8.9</td>
<td>51</td>
<td>22</td>
<td>19.4</td>
<td>13.2</td>
<td>100</td>
</tr>
<tr>
<td>3</td>
<td>18.5</td>
<td>14.9</td>
<td>500</td>
<td>23</td>
<td>40.4</td>
<td>28.1</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>40.4</td>
<td>28.1</td>
<td>10</td>
<td>24</td>
<td>28.1</td>
<td>19.4</td>
<td>25</td>
</tr>
<tr>
<td>5</td>
<td>21.9</td>
<td>17.2</td>
<td>250</td>
<td>25</td>
<td>40.4</td>
<td>28.1</td>
<td>31</td>
</tr>
<tr>
<td>6</td>
<td>4.9</td>
<td>2.0</td>
<td>250</td>
<td>26</td>
<td>31.0</td>
<td>22.0</td>
<td>400</td>
</tr>
<tr>
<td>7</td>
<td>13.2</td>
<td>8.9</td>
<td>200</td>
<td>27</td>
<td>40.4</td>
<td>28.1</td>
<td>10</td>
</tr>
<tr>
<td>8</td>
<td>40.4</td>
<td>28.1</td>
<td>45</td>
<td>28</td>
<td>40.4</td>
<td>28.1</td>
<td>10</td>
</tr>
<tr>
<td>9</td>
<td>20.0</td>
<td>18.2</td>
<td>500</td>
<td>29</td>
<td>40.4</td>
<td>28.1</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>28.1</td>
<td>19.4</td>
<td>10</td>
<td>30</td>
<td>19.4</td>
<td>13.2</td>
<td>110</td>
</tr>
<tr>
<td>11</td>
<td>28.1</td>
<td>19.4</td>
<td>45</td>
<td>31</td>
<td>40.4</td>
<td>28.1</td>
<td>25</td>
</tr>
<tr>
<td>12</td>
<td>19.4</td>
<td>13.2</td>
<td>160</td>
<td>32</td>
<td>40.4</td>
<td>28.1</td>
<td>50</td>
</tr>
<tr>
<td>13</td>
<td>22.7</td>
<td>22.0</td>
<td>100</td>
<td>33</td>
<td>40.4</td>
<td>28.1</td>
<td>20</td>
</tr>
<tr>
<td>14</td>
<td>9.0</td>
<td>5.0</td>
<td>10</td>
<td>34</td>
<td>33.0</td>
<td>22.0</td>
<td>74</td>
</tr>
<tr>
<td>15</td>
<td>13.2</td>
<td>8.9</td>
<td>25</td>
<td>35</td>
<td>20.0</td>
<td>4.7</td>
<td>510</td>
</tr>
<tr>
<td>16</td>
<td>13.2</td>
<td>8.9</td>
<td>25</td>
<td>36</td>
<td>28.1</td>
<td>19.4</td>
<td>300</td>
</tr>
<tr>
<td>17</td>
<td>13.2</td>
<td>8.9</td>
<td>25</td>
<td>37</td>
<td>13.2</td>
<td>8.9</td>
<td>120</td>
</tr>
<tr>
<td>18</td>
<td>13.2</td>
<td>8.9</td>
<td>25</td>
<td>38</td>
<td>28.1</td>
<td>19.4</td>
<td>15</td>
</tr>
<tr>
<td>19</td>
<td>13.2</td>
<td>8.9</td>
<td>25</td>
<td>39</td>
<td>13.2</td>
<td>8.9</td>
<td>35</td>
</tr>
<tr>
<td>20</td>
<td>13.2</td>
<td>8.9</td>
<td>50</td>
<td>40</td>
<td>40.4</td>
<td>28.1</td>
<td>100</td>
</tr>
</tbody>
</table>

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).
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 (\(\alpha\) for \(1400 \times 700\) and \(\delta\) for \(2800 \times 1400\)).

Table 4: Dataset 30S\(\alpha\) created by [10].

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

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 \( \text{prints} \)), the average total cost (column \( \text{cost} \)) and the average CPU time in seconds (column \( \text{time} \)) 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 \( C_{su} = 208 \), 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 \( C_{su} = 208 \) 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 \( \delta \)) the number of patterns is logically smaller than for datasets with a smaller pattern size (type \( \alpha \)). Even if the number of patterns is lower, the running time is greater because the number of images on patterns of type \( \delta \) is much larger than the number of images in the small patterns (type \( \alpha \)).

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 computed from 10 runs on the dataset 30V\(_\alpha\) (the price of the printing cost of one stock sheet \( C_{su} \) is constant and equal to 18$). It shows that the number of patterns of the best found solution decreases (from 9.2 to 7.2) when the price of the manufacturing cost of one setup increases (from 1$ to 25$). A decrease of \( m \) is observed with all the datasets of this study.

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


![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 30V\(_\alpha\) (C\(_{ss}\) is constant and equal to 18$).](image-url)
Table 5: Results from artificial datasets given by [10].

<table>
<thead>
<tr>
<th>Data</th>
<th>Average</th>
<th>Best</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$L_0$</td>
<td>$m$</td>
</tr>
<tr>
<td>20Sα</td>
<td>4</td>
<td>4.0</td>
</tr>
<tr>
<td>30Sα</td>
<td>5</td>
<td>5.0</td>
</tr>
<tr>
<td>40Sα</td>
<td>7</td>
<td>7.4</td>
</tr>
<tr>
<td>50Sα</td>
<td>8</td>
<td>9.0</td>
</tr>
<tr>
<td>20Vα</td>
<td>4</td>
<td>5.4</td>
</tr>
<tr>
<td>30Vα</td>
<td>6</td>
<td>7.8</td>
</tr>
<tr>
<td>40Vα</td>
<td>8</td>
<td>10.0</td>
</tr>
<tr>
<td>50Vα</td>
<td>10</td>
<td>11.8</td>
</tr>
<tr>
<td>20Lα</td>
<td>3</td>
<td>4.2</td>
</tr>
<tr>
<td>30Lα</td>
<td>6</td>
<td>8.0</td>
</tr>
<tr>
<td>40Lα</td>
<td>7</td>
<td>8.4</td>
</tr>
<tr>
<td>50Lα</td>
<td>8</td>
<td>9.6</td>
</tr>
<tr>
<td>20Vδ</td>
<td>1</td>
<td>1.0</td>
</tr>
<tr>
<td>30Vδ</td>
<td>2</td>
<td>2.1</td>
</tr>
<tr>
<td>40Vδ</td>
<td>2</td>
<td>2.3</td>
</tr>
<tr>
<td>50Vδ</td>
<td>2</td>
<td>3.0</td>
</tr>
<tr>
<td>20Lδ</td>
<td>1</td>
<td>1.0</td>
</tr>
<tr>
<td>30Lδ</td>
<td>2</td>
<td>2.0</td>
</tr>
<tr>
<td>40Lδ</td>
<td>3</td>
<td>3.0</td>
</tr>
<tr>
<td>50Lδ</td>
<td>3</td>
<td>3.8</td>
</tr>
<tr>
<td>20Lδ</td>
<td>1</td>
<td>1.0</td>
</tr>
<tr>
<td>30Lδ</td>
<td>2</td>
<td>2.4</td>
</tr>
<tr>
<td>40Lδ</td>
<td>2</td>
<td>2.5</td>
</tr>
<tr>
<td>50Lδ</td>
<td>2</td>
<td>3.0</td>
</tr>
</tbody>
</table>


