New Pretreatment and Resolution using the Bee Colony Optimization for the Two-Dimensional Bin Packing Problem

Aida Kenza AMARA* and Bachir DJEBBAR

Department of Computer Science, Faculty of Mathematics and Computer Science, University of Science and Technology of Oran Mohamed BOUDIAF
USTO MB, B.p.1505 El-M'nouer, 31000, Oran, Algeria.

*Corresponding author: amara2008kenza@yahoo.fr, aida.amara@univ-usto.dz

Keywords: Bin Packing; Pretreatment; Heuristic Algorithm; Bee Colony Optimization.

Abstract The two-dimensional bin packing problem involves packing a given set of rectangles into a minimum number of larger identical rectangles called bins. In this paper, we propose and develop mathematically a new pretreatment for the oriented version of the problem in order to reduce its size, identify and value the lost spaces by increasing the size of some objects. A heuristic method based on the first-fit strategy adapted to this problem is proposed. We present an approach of resolution using the bee colony optimization. The computational results show the effectiveness of the pretreatment in reducing the number of bins.

1. Introduction

The two-dimensional bin packing problem (BPP-2D) is defined as follows: given a set of m rectangular objects \( A = \{a_1, \ldots, a_m\} \) and a number of larger identical rectangles called bins, the problem is to find the minimum number of bins that must be used to pack this set of objects without overlapping. This problem has many industrial applications, including the cutting of standardized stock units in the wood, steel and glass industries, packing on shelves or truck beds in transportation and warehousing, and the paging of articles in newspapers. Many real problems can be modeled as bin packing problems. However, each real problem has its own specificities. The one-dimensional bin packing problem is strongly NP-hard. This also applies in BPP-2D, since it is a generalization of the one-dimensional version [1]. Many papers deal with BPP-2D where the objects have a fixed orientation. Lodi et al. [3] have proposed an extensive survey. We quote Berkey and Wang [4], who have presented a heuristic version for the problem. Meta-heuristic procedures have been presented in Lodi et al. [2] and Faroe et al. [5].

Lower bounds have been presented in Carlier et al. [6]. They dominate the bounds introduced by Boschetti and Mingozzi [7], Martello and Vigo [8] and Fekete and Schepers [9]. Few papers deal with the version of BPP-2D where objects may be rotated by 90°. A heuristic method and new lower bounds have been introduced by Boschetti and Mingozzi [10]. Lodi et al. [2] have presented a meta-heuristic algorithm based on a Tabu search method. Another tabu search method has been proposed in Harwig and Barnes [11] where objects are packed in bottom left stable positions using the method presented in Chazelle [12].

We are interested in the case where the objects have a fixed orientation. This version is known as 2BP|O|F according to the classification proposed by Lodi et al. [2], where the field denoted O means that the objects are not rotated by 90° and the field denoted F means that guillotine packing are not imposed.

Throughout this paper, we consider that each object \( a_i = (L_i, l_i) \) in the set of the objects \( A \) has a width \( L_i \) and a height \( l_i \). We denote as \( L \) and \( l \) the width and the height of the bin \( B = (L, l) \). An instance \( I \) of a BPP-2D|O|F is then defined by the pair \( (A, B) \). \( L_i \) is the greatest dimension of the object \( a_i \), and \( l_i \) the other dimension. The largest and smallest dimensions of the bin are, respectively, \( L \) and \( l \).

This paper is organized as follows: In section 2, we propose a new pretreatment for the two-dimensional bin packing problem with fixed orientation. The strategy used to represent a solution based on a first-fit algorithm adapted to this problem is described in section 3. In section 4, we are
interested in modeling the problem using the bee colony optimization. The implementation is to test
the effectiveness of pretreatment; indeed it is to compare the number of bins used on the same
instance, with and without pretreatment. The principle of creating the sets of tests is given in Section
5. The simulation results of the bee colony optimization algorithm are shown in section 6. A
conclusion follows in section 7.

2. Pretreatment

The pretreatment when applied to an instance \( I \), increase the size of some objects within the
instance. If an object takes the size of a bin, its optimal packing is then known and it can be treated
separately. This allows the cardinality of the set of the objects to be reduced without modifying the
optimal value of the problem. This way, the pretreatment help to increase the total area of the
objects and also simplify the instance, therefore the application of the pretreatment is interesting
before a resolution procedure. The proposed pretreatment is based on the concept of identically
feasible function (IFF) defined in Carlier and al. [6] as follows: \( I = (A, B) \) is an instance of the BPP-
2D the function \( f \) is an IFF associated with the instance \( I \) if and only if the instance \( f(I) = (A', B) \)
obtained by applying \( f \) to all objects of \( A \) is such that \( \text{OPT}(I) = \text{OPT}(f(I)) \).

The pretreatment consists in reclaming lost areas in the bins. Applying pretreatment to the initial
instance allows some objects to be enlarged. If the size of an object is updated to the size of a bin, its
optimal packing is then known and it can be treated separately.

We classify the objects into three categories: large, small and intermediate objects. For the latter,
we introduce a new concept called equivalence class. It involves creating subsets of objects. Each
subset is characterized by objects of the same height and classified in order of decreasing width.

The symbols and abbreviations used to define the pretreatment are:

- \( m \): number of objects \((m \in \mathbb{N}^*)\)
- \( n \): number of bins \((n \in \mathbb{N}^*)\)
- \( A = \{a_i = (L_i, l_i), \forall i=1\ldots m\}\)
- \( B \): all the bins of the same size as \( B = (L, l) \)
- \( L \): width of each bin
- \( l \): height of each bin
- \( L_i \): width of the object \( a_i \)
- \( l_i \): height of the object \( a_i \)
For \( a_i \in A: L_i \geq l_i (\forall i=1, 2\ldots, m) \)

**Definition 1**

The dimensions of the objects and bins are integers. An instance \( I \) of BPP-2D oriented is then
defined by the pair \((A, B)\). The optimal value of the number of bins needed to store all the objects of
an instance \( I \) is denoted \( \text{OPT}(I) \).

Each object is stored in a bin and positioned such that the edge of \( L_i \) dimension is parallel to the
width of the bin \((L)\) and that the edge of \( l_i \) dimension is parallel to the height of the bin \((l)\). (Fig. 1)

![Fig. 1. Orientation of an object in a bin](image-url)
Definition 2

Each object \( a_i = (L_i, l_i) \) in \( A \) has:
\[ L_i \leq L \text{ and } l_i \leq l \]
otherwise the feasibility is not possible.

The set \( A \) is composed of the subsets of objects following \( A_G, A_I, A_P \), such as:
\[ A = A_G \cup A_I \cup A_P \]

\( A_G \): all large objects
\( A_I \): all objects defined by identical properties (depending on height)
\( A_P \): all sufficiently small objects

Definition 3

\( \mathcal{A}_i \) is the equivalence class containing objects that have their height equal to \( l_i \)
\[ \mathcal{A}_i = \{a_i \in A_I : \text{having the small dimension equal to } l_i \} \]
\( \mathcal{A}_i = \text{cl}\{a_i\} = \{a_i \in A_I : l_i = l_i\} \)
\( A_I = \bigcup_{i=1}^{k} \mathcal{A}_i \) such as \( \forall i \neq j : \mathcal{A}_i \cap \mathcal{A}_j = \emptyset \)
\( l(\mathcal{A}_i) \): height of the objects of the equivalence class \( \mathcal{A}_i \) with: \( \forall i = 1, \ldots, k : l(\mathcal{A}_i) > l(\mathcal{A}_{i+1}) \)

2.1 Mathematical formulation

The following model minimizes the number of bins used by a solution:

\[ \min \sum_{z=1}^{n} y_z \quad y_z \in \{0,1\} \quad (z = 1, \ldots, n) \quad (1) \]
\[ \sum_{z=1}^{n} x_{iz} = 1 \quad x_{iz} \in \{0,1\} \quad (\forall a_i \in I, z = 1, \ldots, n) \quad (2) \]
\[ \sum_{a_i \in I} L_i x_{iz} f_{itz} \leq L y_z \quad (z = 1, \ldots, n) \quad (3.1) \]
\[ \sum_{a_i \in I} l_i x_{iz} f_{itz} \leq l y_z \quad (z = 1, \ldots, n) \quad (3.2) \]
\[ \sum_{a_i \in I} f_{itz} = 1 \quad (4) \]

\( y_z \): integer variable that takes the value 1 if the bin \( z \) is used and 0 otherwise.
\( x_{iz} \): integer variable that takes the value 1 if the object \( a_i \) is placed in the bin \( z \), and 0 otherwise.
\( f_{itz} \): integer variable that takes the value 1 if the object \( a_i \) is placed in the level \( t \) of the bin \( z \), and 0 otherwise.

The Constraint (2) controls whether each object is placed in a single bin. The constraints (3.1) and (3.2) satisfy respectively that the sum of the widths (heights) of the objects placed in a bin does not exceed the width (height) of the bin. The Constraint (4) controls whether each object is placed in one and only one level within a single bin.
2.2 Feasibility constraints

This pretreatment is designed to store objects oriented in their large size. The objects, sorted in descending order of their width, are classified into three categories:

- The large objects: noted \( A_G \), are the objects that occupy more than the half of the bin in the width and height.
- The objects defined by identical properties (depending on height), noted \( A_I \) and are represented by the equivalence classes.
- The sufficiently small objects: noted \( A_P \) are the objects that have the large dimension less than a threshold established. They are placed with large and with objects defined by identical properties in order to reduce their size and get an optimal storage.

The objects of \( A_I \) are in conflict with those of \( A_G \) hence the expression:

\[
\forall a_i \in A_I, \forall a_j \in A_G : X_{iz} + X_{jz} \leq Y_z \quad (i,j=1,...,m, z=(1,...,n))
\]  

(5)

A conflict between two objects is a restriction prohibiting storing these objects in the same bin.[13] A conflict also exists between the objects belonging to different equivalence classes in the same level, such as:

\[
\forall a_i^t \in A_i, \forall a_j^t \in A_j : f_{iz}^t + f_{jz}^t \leq 1 \quad (a_i^t, a_j^t \in A_i, i \neq j \text{ and } i,j \in \{1,...,k\} , \forall z,t)
\]  

(6)

The objects \( a_i \) and \( a_j \) cannot be stored in the same level of a particular bin.

In the following, we will clarify the magnitudes concepts of the large objects, objects defined by identical properties (according to height) and the sufficiently small objects by dimensional constraints into three categories:

**Category 1**

\( A_G \) is the set of large objects, such as:

\[
A_G = \{ a_i \in A: L_i > \frac{1}{2} L \text{ et } l_i > \frac{1}{2} l \}
\]

Two objects of the subset \( A_G \) cannot be placed in the same bin. Indeed for any pair of objects \((a_i, a_j)\in A_G\) we have:

\[
\forall a_i, a_j \in A_G: L_i + L_j > L \text{ and } l_i + l_j > l
\]

If a large object takes the size of a bin, its optimal storage becomes trivial and it can be separated from the instance to solve. Otherwise, objects belonging to \( A_P \) are stored with the large object in the bin.

The number of bins required to store all the objects of \( A_G \) is equal to the cardinality of the subset \( A_G (| A_G |) \). Thus, the total number of bins \( n \) is such that \( n > | A_G | \).

**Category 2**

the subset \( A_I \) is defined as follows:

\[
A_I = \{ a_i \in A: \frac{1}{4} L < L_i \leq \frac{1}{2} L \}
\]

In the subset \( A_I \) the relation \( R \) is defined by:

\[
\forall a_i, a_j \in A_I, a_i R a_j \iff l_i = l_j
\]

Two Objects \( a_i, a_j \in A_I \) are bound by the relation \( R \) if and only if \( a_i \) and \( a_j \) have the same height.
Category 3
the subset $A_P$ represents all sufficiently small objects, such as:

$$A_P = \{ a_i \in A : L_i \leq \frac{1}{4} L \}$$

These are the objects that have a sufficiently small size so they can be stored with the objects of $A_G$ and/or $A_I$.

2.3 Storage of objects

The storage of the objects must respect the following strategy:
The large objects are sorted according to their decreasing width, followed by the intermediate objects (the majority) who themselves are divided into equivalence classes and sorted according to their decreasing height (recall that in each equivalence class the objects are sorted in descending order of width). And finally, the sufficiently small objects, such as large objects, are sorted in descending order of width.

Example of an instance illustrating this sort:

$A = (12,10); (11,7); (10,6); (9,7); (7,6); (6,6); (6,5); (5,5); (5,5); (5,5); (5,4); (5,4); (4,4); (4,4); (6,3); (6,3); (5,3); (5,3); (4,3); (4,3); (6,2); (6,2); (5,2); (4,2); (4,2); (6,1); (6,1); (5,1); (4,1); (3,3); (3,2); (2,2); (1,1).

$B = (12, 10)$.

For this instance, we have:

- 5 large objects: $\{(12, 10); (11,7); (10,6); (9,7); (7,6)\}$
- 6 equivalence classes:
  - $\mathcal{A}_1 = \{(6, 6); (6, 6)\}$
  - $\mathcal{A}_2 = \{(6, 5); (6, 5); (5, 5); (5, 5); (5, 5)\}$
  - $\mathcal{A}_3 = \{(6, 4); (6, 4); (5, 4); (5,4); (4,4); (4,4)\}$
  - $\mathcal{A}_4 = \{(6,3); (6,3); (5,3); (5,3); (4,3); (4,3)\}$
  - $\mathcal{A}_5 = \{(6, 2); (6, 2); (5, 2); (4, 2); (4, 2)\}$
  - $\mathcal{A}_6 = \{(6, 1); (6, 1); (5, 1); (4, 1); (4, 1)\}$

- 4 sufficiently small objects: $\{(3,3); (3,2); (2,2); (1,1)\}$

2.4 Storage of large objects

For each object $a_i \in A_G$ placed in a bin, two new bins $B_i^1$ and $B_i^2$ are considered:

$B_i^1 = (L - L_i, l_i)$ and $B_i^2 = (L, 1 - l_i)$.

![Fig.2. Storage of a large object in a bin](image)

The two bins $B_i^1$ and $B_i^2$ are filled with the subset $A_P \subset A_P$.

The instance is then reduced to the optimality and the dimensions of the large object in the bin can be updated as follows:

$P_{r1} : a_i \rightarrow a_i'$
The function $Pr_1$ is a valid pretreatment procedure if and only if $|A_G| \leq |A_P' \cup A_P|$.

2.5 Storage of objects defined by identical properties (depending on height)

We consider the different equivalence classes in descending order of height. Conflict procedure detection is applied. It involves separating the equivalence classes in the same level of a given bin. The aim is to store objects belonging to the same equivalence class to form a first layer in the bin.

If the lost width obtained after placement a set of objects belonging to $\mathcal{A}_i (i=1 \ldots k)$ is less than $L_i$, $\forall L_i \in \mathcal{A}_i$, objects of the subset $A_p \setminus A_p'$ are added. For each equivalence class $\mathcal{A}_i$ we seek to build the subset $\mathcal{A}_i^{r,q}$, such that:

$$\mathcal{A}_i^{r,q} = \{ a_i \in \mathcal{A}_i : \sum L_i = p \text{ and } l_i = q : p \leq L \text{ and } q \leq 1 \}$$

Construction of the subsets $\mathcal{A}_i^{r,q}$

Initially, the equivalence classes are organized in descending order of height. The first subject of the first equivalence class is deposited on the bottom left position in an empty bin and the remaining objects are placed at the right until no space can contain the current object.

A first level is formed with the height of the equivalence class considered. The sum of the widths of all objects stored in this level is calculated by the function $S$:

$$S(\mathcal{A}_i^{r,q},L_i,L_i) = \max \sum X_{iz} L_i \text{ such that:}$$

$$\max \sum X_{iz} L_i \leq L \quad X_{iz} \in \{0,1\}$$

$$L - \max \sum X_{iz} L_i < L_i \quad \forall a_i \in \mathcal{A}_i \setminus \mathcal{A}_i^{r,q} (i=1,\ldots,k)$$

We set:

$$m_i = \max \sum X_{iz} L_i = r \text{ and } l_i = q \quad \forall a_i \in \mathcal{A}_i$$

$m_i$: cardinality (card) of the subset $\mathcal{A}_i^{r,q}$

Other subset belonging to the same equivalence class or to the equivalence class with lower height can be layered while: $l-q \geq l(\mathcal{A}_i) \forall i \in \{1,\ldots,k\}$. This procedure is repeated until an optimal packing in the bin is obtained.

Note: To ensure optimality, it is necessary to put a maximum number of objects of the same equivalence classes in a level (if the cardinality of the equivalence class permitting). Otherwise we choose the next class.

For each subset $\mathcal{A}_i^{r,q}$ created, two new bins $B_i^3$ and $B_i^4$ are considered: $B_i^3 = (L-r,q)$ such as: $L-r < L_i$ ($\forall a_i \in \mathcal{A}_i : i=1,\ldots,k$) and $B_i^4 = (L,l-q)$.
Two decision problems to solve:
1° is it possible to store objects of \( A_P \setminus A_{P'} \) in the bin \( B_i^3 \)?
2° is it possible to consider the bin \( B_i^4 \) with the principle of the storage of objects defined by identical properties? For the second decision problems, there are two cases:

**Case 1:** \( 1 - q \geq 1 (A_i) \) \( i=1 \ldots k \), build a new level above the previous.

**Case 2:** \( 1 - q < 1 (A_i) \) \( i=1 \ldots k \), sufficiently small objects of \( A_P \setminus A_{P'} \) are stored in the bin \( B_i^4 \).

If the answers are yes then the following function is an IFF associated with the instance \( I \) considered:

\[
A_i^{r,q} \subset A_i \rightarrow A_i^{r,q} \subset A_i'
\]

\[
\begin{cases}
    r' = L \text{ and } q' = q \text{ if } (a_i \in A_i \text{ and } L - r < L_i \text{ and } (1 - q \geq 1 (A_i), i=1,\ldots,k)) \\
    r' = L \text{ and } q' = 1 \text{ if } (a_i \in A_i \text{ and } L - r < L_i \text{ and } (1 - q < 1 (A_i), i=1,\ldots,k)) \\
    L_i = 0 \text{ and } l_i = 0 \text{ if } (a_i \in A_P \setminus A_{P'}) \\
    L_i = L_i \text{ and } l_i = l_i \text{ otherwise}
\end{cases}
\]

\[A_i' = \bigcup A_i^{r,q}\]

3. **Proposition of a First-Fit Decreasing Width (FFDW) adapted to the BPP-2D**

The objects are sorted according to their decreasing width and oriented as follows:

Each object stored in a bin is positioned such that the edge of \( L_i \) dimension is parallel to the width of the bin (L) and that the edge of \( l_i \) dimension is parallel to the height of the bin (l). The objects are placed according to their widths, successively in the bins for forming different levels. Initially, only one bin is considered and the objects are sorted in descending order of width. The first object (the largest) is placed at the bottom left position in the bin and other objects are placed (depending on the sort order) to the right of the first object until no space can contain the current object.

A first level is thus formed and its height is determined by the highest object belonging to this level. If the remaining height in the bin (after creating the first floor), is greater than or equal to the height of the candidate object, a new level is created over the previous. Otherwise, a new bin is considered without closing the first one.

In an intermediate stage where we have opened \( k \) bins numbered 1 to \( k \) in the order of their first use, a current object is stored in the level of the bin of lower numbers having enough space to contain it. If no bin can contain the object \( a_i \), a new bin (\( k + 1 \)) is opened without closing the others. The general procedure is repeated until all the objects of the instance considered will be stored.

4. **Optimization by bee colony algorithm**

The algorithm is based on the natural bee behavior:

a. The bee colony can spread over long distances in multiple directions (over 10 km). The copious amounts of nectar or pollen should be visited by more bees.

b. The bees scouts looking at random from one plot to another food source.

c. The bees returning to the hive evaluate different food sources in function of a certain quality threshold (measured by a combination of some elements, such as the sugar content)
d. The bees deposit the nectar or pollen and go to the dance floor to perform a waggle dance.
e. The bees communicate through the waggle dance for giving the following information: the
direction of the source (angle between the sun and the source), the distance from the hive
(duration of the dance) and the quality assessment (fitness: Frequency dance)
f. Thus the bees can collect food effectively and quickly
g. This source will be announced in a new waggle dance to determine the current food level. If
this level is suitable more bees will be recruited for this source.
h. Depending on the physical condition, sources with copious amounts of nectar or pollen can
be visited by more bees or may be abandoned.

4.1 Basic algorithm of bee’s method

The pseudo code of the basic bee’s algorithm is presented in the simplified form:

- Initialize the population randomly
- Evaluate the fitness of the population
- While the stop criterion is not met
  - Select m solutions amongst n to perform local search
  - Recruit bees for selected solutions and evaluate their fitness
  - Select the best bee in each neighborhood
  - Assign the remaining bees randomly and evaluate their fitness
- End while

4.2 The parameters of the simulation

Objects parameters:
- Number of objects to be stored
- Maximum dimensions ($L_i$ and $l_i$) of each objects $a_i \in A$

Bins parameters:
- Number of bins
- Size of each bin ($L$, $l$)

Bee algorithm parameters:
- Number of bees
- Percentage of each type of bees (Scouts, active and inactive)
- Number of visits
- Number of iterations

To select and set the parameters of the bee algorithm, various tests were performed. The best
combination providing an optimal solution was maintained. Indeed, we varied the number of:
- Active bees (between 30% and 60%)
- Scout bees (between 20% and 40%)
- Inactive bees (between 10% and 30%)
- Iterations from 5 to 50

4.3 The coding parameters

- The nectar (or pollen) represents the object to be placed in the bin
- The hive represents the bin to fill with objects
- The Quality, proximity and ease of extraction of the food source is the fitness of the solution
5. Implementation tests

We varied the number of objects in the instances (I₁, I₂ and I₃): 60, 100, and 240. Each instance is characterized by three categories of objects with a probability of existence. The size of the bins in each instance is equal to (12, 10).

<table>
<thead>
<tr>
<th>Category</th>
<th>Aₐ</th>
<th>Aᵢ</th>
<th>Aᵰ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Probability</td>
<td>10%</td>
<td>50%</td>
<td>40%</td>
</tr>
</tbody>
</table>

Table 1. Categories and probabilities of the objects in each instance

| the interval of the width L | [7, 12] | [4, 6] | [1, 3] |

6. Computational results

The algorithm is implemented in JAVA programming language on processor intel (R) Core (TM) i3-3120M with a speed of 2.5 GHz and 4GB memory capacity of RAM in Windows 8. The choice of integrated development environment NetBeans 7.4 was guided by the benefits of object oriented programming.

The objective is to demonstrate the impact of the introduction of the concept of equivalence classes and conflicts between objects and subsets of objects in minimizing the number of bins used. The implementation is to test the effectiveness of the pretreatment; indeed it is to compare the number of bins used on the same instance, with and without pretreatment. We begin with a resolution using an adaptation of the first-fit heuristic to our problem (FFDW). Then we proceed to the application of our pretreatment and compare the results obtained in terms of number of bins used.

The results of the simulation of the three instances I₁, I₂ and I₃ with the pretreatment are shown in the following table:

<table>
<thead>
<tr>
<th>Set</th>
<th>Number of objects</th>
<th>optimal solution</th>
<th>Number of bins occupied by Aₐ</th>
<th>Number of bins occupied by Aᵢ</th>
</tr>
</thead>
<tbody>
<tr>
<td>I₁</td>
<td>240</td>
<td>52 bins</td>
<td>24</td>
<td>28</td>
</tr>
<tr>
<td>I₂</td>
<td>100</td>
<td>23 bins</td>
<td>10</td>
<td>13</td>
</tr>
<tr>
<td>I₃</td>
<td>60</td>
<td>14 bins</td>
<td>6</td>
<td>8</td>
</tr>
</tbody>
</table>

The results of the simulation of I₁, I₂ and I₃, with the pretreatment are satisfactory because the various constraints imposed have been met. It is therefore necessary to validate this pretreatment, to make a comparison with FFDW:

<table>
<thead>
<tr>
<th>Set</th>
<th>Number of objects</th>
<th>Optimal solution with pretreatment OPT (I)</th>
<th>Optimal solution with FFDW</th>
<th>Gain in number of bins</th>
</tr>
</thead>
<tbody>
<tr>
<td>I₁</td>
<td>240</td>
<td>52</td>
<td>59</td>
<td>7</td>
</tr>
<tr>
<td>I₂</td>
<td>100</td>
<td>23</td>
<td>25</td>
<td>2</td>
</tr>
<tr>
<td>I₃</td>
<td>60</td>
<td>14</td>
<td>14</td>
<td>0</td>
</tr>
</tbody>
</table>

In each instance I₁, I₂ and I₃ listed the optimal solution is given by our pretreatment. This is specially the case when the number of objects increases. These good results are mainly due to the introduction of the concept of equivalence classes.

7. Conclusion

In this paper, we have proposed new pretreatment for the two-dimensional bin packing problem, the oriented case. We have described an adaptation of the First Fit heuristic for the resolution of the problem. We have also compared the numbers of bins used to pack all objects of the instances with and without pretreatment and the optimal solution is giving by our pretreatment.
References


