An Efficient Real-Time Algorithm for Placing Electronic Components on Panel

Catherine Huyghe, Stephane Negre, Melanie Fontaine

An Efficient Real-Time Algorithm for Placing Electronic Components on Panel

CATHERINE HUYGHE[1], STEPHANE NEGRE[2], MELANIE FONTAINE[3]
[1]Department of MIS, "Modélisation, Information & Systèmes"
University of Picardie Jules Verne, Amiens
FRANCE

[2]Department of EPROAD, "Eco-PRocédés, Optimisation & Aide à la Décision"
University of Picardie Jules Verne, Amiens
FRANCE

[3]Department of LTI, "Laboratoire des Technologies Innovantes"
University of Picardie Jules Verne, Amiens
FRANCE

Abstract: - In this paper, we present an industrial challenge faced by electronic component manufacturers. From each order, the components need to be placed on panels for shipping. Panels incur significant costs. To control expenses, companies are compelled to optimize the placement of components on panels. Every day, thousands of orders have to be shipped. For each order, they have to pack between a hundred and a thousand rectangular electronic components of different sizes on panels with specifications such as free component orientation and tight time constraints. This packaging must be optimized to minimize the number of panels used and associated costs. The industrial challenge lies not only in space optimization but also in the speed of the process, with solutions needing to be found in less than a minute to meet the dynamic and continuous demands of production and shipping.

Key-Words: - bin packing problem, heuristic, dynamical programming algorithm, pseudo-polynomial algorithm, lower bound, operational research, combinatorial problem.

Received: May 23, 2023. Revised: December 26, 2023. Accepted: January 19, 2023. Published: February 28, 2024.

1 Introduction

An electronic components company handles thousands of orders daily, each order consisting of up to a thousand components. The electronic components from each order need to be placed on panels for shipping. These panels have an expensive cost. The objective is to optimize the placement of electronic components on panels to minimize the number of panels used and, consequently, the associated costs.

Certain constraints exist, such as the fact that the components can be freely or fixed-oriented, meaning they can be rotated in 90-degree increments or not before being placed on a panel.

This problem can be framed as a two-dimensional bin packing problem (2D-BPP), where electronic components are rectangular objects to be packed, and the panels are the rectangular bins intended to contain them. Notably, some components have free orientation, placing the scenario within the realm of the two-dimensional bin packing problem with free orientation. As a characteristic NP-complete problem, finding an exact algorithm in polynomial time for this task is deemed impractical in the worst-case scenario.

To be able to process the thousands of orders per day within a reasonable timeframe, it is necessary, for each order, to be able to handle the placement of electronic components on panels in less than one minute while remaining close to the optimal solution.

2 State of the Art

In operational research and combinatorial optimization, the bin packing problem plays a crucial role in various sectors such as production, transportation, scheduling management, etc. The simplest version of the problem involves finding the minimal number of bins that contain the whole set of

objects, respecting certain placement rules. The main objective is to maximize the filling of the bins and, therefore, minimize wasted space, [1], [2].

This problem, classified NP-complete, [3], presents particular challenges, especially for instances involving numerous objects. The literature addresses various types of bin-packing problems tailored to specific applications. A typology considering the number of dimensions, bin, and object characteristics is described in [4]. A more recent classification is proposed in [5], aiming to include recent issues in cutting and arrangement.

The one-dimensional bin-packing problem (1D-BPP) is a well-known combinatorial optimization problem closely related to the one-dimensional cutting stock problem, [6]. Various approaches have been proposed to solve this NP-hard problem, including the use of metaheuristic algorithms, [7], symmetry-breaking constraints in integer linear programming, [8], branch-and-price-and-cut algorithms, [9], and hybrid intelligent algorithms for problems with multiple size restrictions, [10]. These studies have significantly advanced the field by providing effective methods for solving the 1D-BPP.

The two-dimensional bin packing problem (2D-BPP) is a challenging optimization problem with numerous applications. In [11], a classification of 2BP problems considering orientation and guillotine constraints is proposed. An algorithm for the determination of the guillotine restrictions is proposed in [12]. A first model dedicated to 2D-BPP is introduced in [13], as an extension of their 1D-BPP approach, [14], [15]. A model based on graph theory is proposed in [16]. The study in [17], provides a comprehensive analysis, including its time complexity and NP-hardness. A solution using combinatorial Benders decomposition, significantly improving on previous algorithms, is presented in [18]. A model considering slit distance and free rotation of pieces, demonstrating its competitiveness on benchmark instances, is introduced in [19] and [20], addresses just-in-time 2D-BPP, combining bin-packing and single-machine scheduling, proposing an integrated constraint program as a solution.

The three-dimensional bin-packing problem (3D-BPP) has been addressed in several studies with a unique focus. A solution, developed in [21], proposes an approach for pallet loading problems, considering practical constraints such as vertical support and load-bearing. A variation accounting for the compressibility factor of deformable objects, significantly enhancing space utilization is considered in [22]. A column generation-based heuristic for the problem with rotation, outperforming existing techniques in solution quality is proposed in [23] and [24], tackled a multi-objective version, aiming to minimize the number of bins while achieving balanced bins in terms of total weight. A topology order for 3D-BPP is proposed in [25].

We could deal with N-dimensional bin packing problem, where objects and bins have dimensions N. One-dimensional bin packing is NP-complete, and its generalization to N dimensions maintains this complexity, [26].

3 Proposed Approach

In this section, we proposed a dynamic and efficient approach to ensure a rapid response with the large number of components to be processed. A dynamic programming approach presents itself as a viable solution. This technique provides a systematic method for managing complexity by breaking down the overall problem into smaller sub-problems, allowing for a step-by-step optimization strategy. Essentially, global optimization is achieved through a sequential string of incremental enhancements that can be quickly calculated.

In this section, we first recall the exact algorithm to solve the partition problem of [3]. Then, we explain how we adapt this method to 1D-BPP. Subsequently, we introduce 2D-BPP and we expose the efficient algorithm based on this method.

3.1 Exact Method of the Partition Problem

A exact algorithm to solve the 2-partition problem, a well-known NP-complete problem, is introduced in [3].

In the 2-partition problem, given a finite set A of n elements a, each having a size $s(a) \in \mathbb{Z}^+$ for $a \in A$. The objective is to determine if there exists a subset A' of A such that: $\sum_{a\in A'} s(a) = \sum_{a\in A-A'} s(a)$.

They introduce the definition $B = \sum_{a\in A} s(a)$. If B is odd, the answer is immediately "False". If B is even, they construct a matrix $M(i,j)$ with $n+1$ rows and $(B/2)+1$ columns, where $0 \le i \le n$ and $0 \le j \le B/2$.

The first row of this matrix is initialized by: $M(0,0) =$ True, $M(0,j) =$ False, $\forall j > 0$. After, for all rows i from 1 to n, the matrices are sequentially constructed by setting $M(i,j) =$ True if and only if either $M(i-1,j) =$ True or $M\big(i-1, j-s(a_i)\big) =$ True with $j \in [s(a_i), B/2]$.

Thus, $M(i,j)$ represents the truth value of the statement: "there exists a subset of $a_1, a_2, \dots, a_i$ for

which the sum of the item sizes is exactly j". The initialization is verified for the first row: we have the value True only in column 0. We can obtain the sum of 0 with the empty subset. The recursion is based on the following executions:

- If there exists a subset of the first $(i-1)$ objects such that their sum is exactly equal to j, then the same subset exists when adding object i. So, $M(i,j) =$ True if $M(i-1,j) =$ True.
- If $M\big(i-1, j-s(a_i)\big) =$ True, we can construct a new subset containing the subset A among the $i-1$ elements such that $\sum_{a_i \in A} s\,(a_i) = j - s(a_i)$ and we concatenate it with the a_i element in order to obtain the sum exactly equal to j.

To summary, the matrix is calculated as follows:

- $M(0,0) =$ True
- $M(i,j) =$ True if and only if $M(i-1,j) =$ True or $M\big(i-1, j-s(a_i)\big) =$ True.

Once the entire matrix has been filled in, the instance of the 2-partition problem is solved because the answer is "yes" if and only if $M(n, B/2) =$ True. In this case, there exists of subset A' of the n objects for which the sum is equal to $B/2$. We can obtain the second partition by constructing $A - A'$.

3.2 Adaptation of the Exact Partition Method to the 1D-BPP

In 1D-BPP, we have a finite set $O = o_1, o_2, \dots, o_n$ of objects. Each object $0 < i < n$ has a rational size $s(o_i)$. The objective in this problem, is to place all O objects in a minimum number bin of size b (all bins have the same size). We define the decision variables (which constitute the solution to the problem):

- $z_k = 1$ if bin k is used (0 otherwise)
- $f_{ik} = 1$ if object i is in bin k (0 otherwise)

The objective function is: Min $\sum_{k=1} z_k$
The model has the following feasible constraints:

- Each object must be placed exactly in one bin: $\sum_{k=1} f_{ik} \geq 1$ and $f_{ik} \leq z_k$
- The capacity of each bin must not be exceeded: $\sum_{i=1} f_{ik} * s(o_i) \leq b$

The partition problem and the bin packing problem are two classical combinatorial problems in theoretical computer science and optimization. The bin packing problem is a polynomial reduction of the partition problem. This means that there is a polynomial algorithm (i.e. a transformation whose cost is polynomial in terms of input size) to transform partition problem into 1D-BPP.

We study this polynomial reduction in order to propose an efficient algorithm for 1D-BPP. Each object size $s(o_i)$ in the bin packing instance is associated with a positive integer $s(a_i)$ in the partition instance.

In the 1D-BPP, we need to determine in which bin to place each object to minimize the total number of bins used. To do this, we need to maximize the filling of all the bins used. The main idea of our approach is to measure, at each decision step, for each feasible set of an object to a bin, the minimum space that will inevitably be lost at the end of the resolution. This measure allows us to anticipate all the residual spaces that will be lost in each bin. To minimize the space in the bins that will inevitably be lost at the end, we rely on the resolution method with the partition problem.

One approach is classically to place the most constraining objects first, i.e. the largest ones. To do this, we start by sorting the objects in ascending order of size. After we can construct matrix $M(i,j)$ with $n+1$ rows and $b+1$ columns, where $1 \leq i \leq n$ and $0 \leq j \leq b$. In this way, each row corresponds to an object except the first line which corresponds to the empty set of objects and each column to a bin size.

We can consider that the maximum number of necessary bins is n (each object is affected to a different bin). For each placement of an object o_i in a bin k, we calculate the residual space quantity : $P_k = b - \sum_{o_i \in O} s\,(o_i) \times f_{ik}$. Then, in the matrix, we look for the largest value j on line $i-1$ (so as not to take into account the object to be placed), such that $M(i-1,j) =$ True, $\forall j \in [0, P_k]$. The space that will inevitably be lost at the end in bin k is then $b - j$.

We perform this calculation for each bin that can accommodate object o_i. Then we choose the bin in which we place the object in to minimize the space that will be lost at the end of the treatment among the set of all the available bins:

$$\min_{k \in [1,n]} \left(P_k - \max_{j \in [0,P_k]} j \; s.t. M(i-1,j) = True \right).$$

This matrix allows us to detect, every decision to place, the quantity that will inevitably be lost at the end without knowing the next objects to be placed. It allows us to anticipate the losses we will have at the end of the packing regardless of the remaining objects.

For example, let's take an instance consisting of a bin of size 9 and 4 objects of sizes 3, 3, 3 and 4. The matrix M associated with this instance is visible in Figure 1. We start by placing object 4 (the largest one) in bin k. Following the placement of this object

4, $P_k = 9 - 4 = 5$ and $\left(P_k - \max_{j \in [0,P_k]} j \; s.t. M(i-1,j) = True\right) = 5 - 3 = 2$. This means that there is no subset of residual objects that can fill this quantity 2. This quantity (grey zone in the bin of Figure 1) is therefore inevitably and permanently lost because, in the third line of the matrix M, we have False values from column 5 to column 3 (red arrow in the matrix M of Figure 1). That means that no subset of remaining objects can be summed to give 5 or 4. Therefore, it is detected that these 2 units available in the bin will be lost.

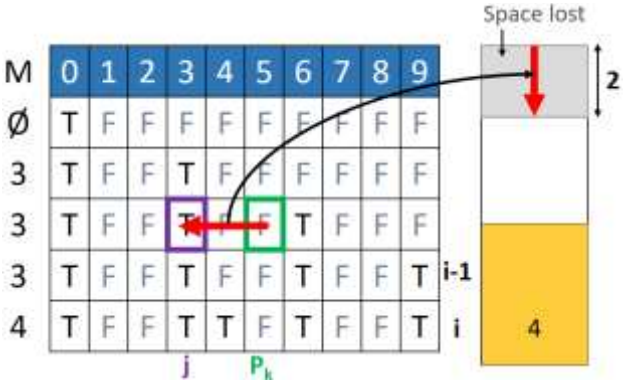

Fig. 1: Calculation of the area that will necessarily be lost, in grey, after placing object i of size 4 in a bin of size 9

Let's take this example and add an object of size 2. We now have an instance composed of a bin of size 9 and 5 objects of sizes 4, 3, 3, 3 and 2. The matrix M associated with this instance is visible in Figure 2. In this example, we know as soon as we place object 4 that there will be no loss generated at the end in this bin. Indeed,

$\left(P_k - \max_{j \in [0,P_k]} j \; s.t. M(i-1,j) = True\right) = 5 - 5 = 0$.

To summary, our criterion for choosing the bin in which to place object i is the bin that minimizes the final loss such that:

$$\min_{k \in [1,n]} (b - \sum_{o_i \in O} s(o_i) \times f_{ik} - \max_{j \in \left[0, b - \sum_{o_i \in O} s(o_i) \times f_{ik}\right]} j \; s.t. M(i-1,j) = True).$$

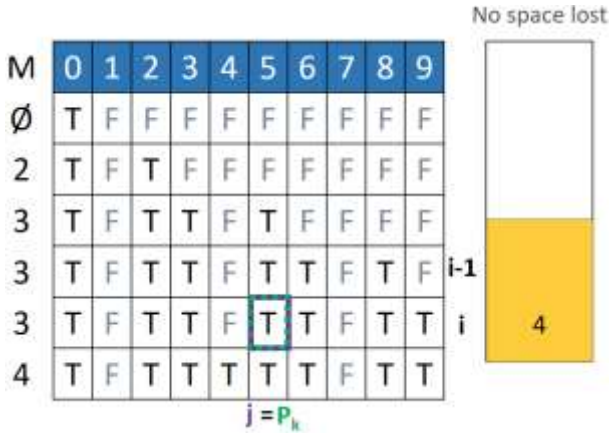

Fig. 2: Calculation of the area that will necessarily be lost after placing object i of size 7 in a bin of size 11

3.3 Adaptation to 2D-BPP

The 1D-BPP involves packing objects of different sizes into bins of fixed capacity, to minimize the total number of bins used. Each object has a size, and the aim is to find the packing configuration that uses the fewest bins possible.

The 2D-BPP extends this problem to two-dimensional considerations. To give a formal definition of the problems we have based ourselves on [27], [28], [29]. In this section, we will use the "." operator as a relation between an instance of a class and an attribute. For example, $o_i.h$ represents attribute h (height) for object o_i. For a complete comprehension of all the attributes of all the considered classes, we can refer to Figure 7 in section 3.5.

Given a finite set $O = o_1, o_2, \ldots, o_n$ of objects, each o_i object has a height $o_i.h$ and width $o_i.w$. They can have a fixed or free orientation. If the object has a free orientation, this means that it can be rotated by 90 degrees before being placed in a bin. The objective in this problem is to place all n objects in a minimum k of bins. All the bins have the same height $b.h$ and width $b.w$. We define the decision variables (which constitute the solution to the problem):

- $l_{ij} = 1$ if object i is to the left of j (0 otherwise).
- $a_{ij} = 1$ if object i is above j (0 otherwise).
- $f_{ik} = 1$ if object i is in bin k (0 otherwise).
- $z_k = 1$ if bin k is used (0 otherwise).
- $o_i.x$ is the x (left) coordinate of object i in a bin and $o_i.y$ is its y (bottom) coordinate. (x, y) represents the coupling of the x and y attributes. $o_i.(x, y)$ is the bottom left coordinate of object i in a bin.

The objective function is: Min $\sum_{k=1} z_k$
The model has the following feasible constraints:

- If object i is neither to the left, nor to the right, nor above, nor below object j, then they are in two distinct bins: $l_{ij} + l_{ji} + a_{ij} + a_{ji} + (1 - f_{ik}) + (1 - f_{jk}) \geq 1, \quad \forall i < j$
- Objects don't overlap: $o_i.x - o_j.x + b.w \times l_{ij} \leq b.w - o_i.w$ and $o_i.y - o_j.y + b.h \times a_{ij} \leq b.h - o_i.h$
- Objects don't exceed the boundaries of the bin: $o_i.x \leq b.w - o_i.w + (1 - f_{ik}) \times b.w$ and $o_i.y \leq b.h - o_i.h + (1 - f_{ik}) \times b.w$ and $o_i.x, o_i.y \geq 0$
- Each object is placed in at least one bin: $\sum_{k=1} f_{ik} \geq 1$ and $f_{ik} \leq z_k$

3.4 Adaptation of the Exact Partition Method for the 2D-BPP

After having presented how we adapt exact method of partition resolution to the 1D-BPP using the properties of polynomial reduction, we now explain the extension to 2D-BPP.

To facilitate the approach, we consider two types of surfaces inside the bins: maximum surfaces and minimum surfaces.

3.4.1 Maximum Surface

A surface SM is maximum if and only if there is no other maximum surface that can include it in its entirety. Every maximum surface SM, is characterized by a distinctive bottom-left point ($SM.x$, $SM.y$), which specifies its position coordinate in bin.

For example, in Figure 3, placing the object at the bottom-left (0,0) position produces the two maximum surfaces SM_1 (rectangle defined by the 4 blue dotted lines) and SM_2(pink dotted lines).

Intersections of maximum surfaces are non-empty. Maximum surfaces allow us to know all the possible placements of each object in each bin. They are therefore used to determine all the possible placements of an object in bins. Each object is located in the bottom-left corner of a maximum surface. When an object is placed, we update the list of maximum surfaces L_{SM}. All maximum surfaces SM impacted by its placement are subdivided into residual maximum surfaces, while retaining the maximum surfaces attribute.

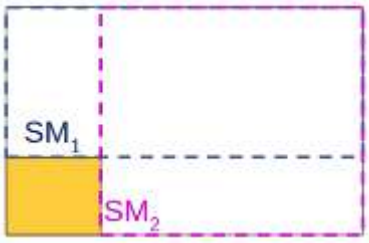

Fig. 3: Example of maximum surfaces. The maximum surfaces SM1 and SM2 are obtained after placing the first object in the bin

3.4.2 Minimum Surface

A minimum surface Sm is a surface resulting from the projection of all object segments placed on all squares along an axis. We have minimum surfaces along the x-axis Smx and minimum surfaces along the y-axis Smy.

For example, in Figure 4 we can see the minimum surfaces on the x and y axes created by the five object placements (yellow).

All the intersections between minimal surfaces are empty. These will be used to calculate the surfaces that will inevitably be lost at the end of each decision to place an object in a bin (i.e. in a maximum surface).

Each time an object is placed in a maximum surface, we update the list of minimum surfaces for the x-axis L_{Smx} and the y-axis L_{Smy}. To do this, we determine the surfaces impacted by the object's placement to deduce the new minimum surfaces.

Thus:

$$L_{Smx} = \cup_{Sm \in Smx} \{Sm, Sm \cap o_i.(x, y, w, h) \neq \varnothing\}$$
$$L_{Smy} = \cup_{Sm \in Smy} \{Sm, Sm \cap o_i.(x, y, w, h) \neq \varnothing\}$$

Where Smx is the set of minimum surfaces on the x-axis, Smy is the set of minimum surfaces on the y-axis, $o_i.(x, y, w, h)$ represents the combination of attributes x, y, w, and h of object i. This represents the surface area of object i at position (x, y) in the bin.

Note that initially, i.e. when no object is placed, the list of surfaces minimum and maximum are both initialized to the sizes of all the available bins. UB is a trivial upper bound of the bin number, i.e. the number of objects.

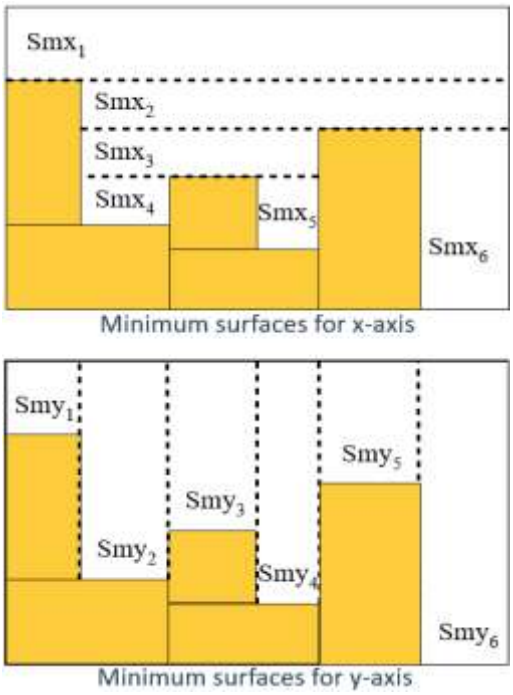

Fig. 4: Example of minimum surfaces resulting from the projection of all object segments placed in the bin

3.4.3 Partition Matrix for 2D-BPP

We construct two matrices, Mx for the x-axis and My for the y-axis. Mx will have as column number $b.w + 1$ and My will have as column number $b.h + 1$. Mx and My have $n + 1$ row number (the number n of objects $+1$). Objects are sorted in ascending order of surface area, so that the most constraining objects are placed first. If an object is freely oriented, we consider that we have two distinct objects. One for the width direction ($o_i.h$ x $o_i.w$) and one for the length direction ($o_i.w$ x $o_i.h$). We then place one or exclusive (XOR) between the two objects. In this way, the placement of one object results in the placing of the other. The matrices are then calculated in the same way as for the one-dimensional bin packing problem, i.e.:

- $Mx(0,0) = True$
- $My(0,0) = True$
- $Mx(i,j) = True$ if and only if $Mx(i-1,j) = True$ or $Mx(i-1, j-o_i.w) = True$.
- $My(i,j) = True$ if and only if $My(i-1,j) = True$ or $My(i-1, j-o_i.h) = True$.

Thus, $Mx(i,j) = True$ if there exists, among the first i elements, a combination such that $o_i.w = j$ and $My(i,j) = True$ if there exists, among the first i elements, a combination such that $o_i.h = j$.

3.4.4 Dynamical Criteria

Using matrices, we determine which surfaces, associated to all minimal surfaces, will be irretrievably lost at the end of the process.

When an object is placed in a maximum surface, it produces a new lists of minimum surfaces. L_{Smx} (resp. L_{Smy}) denotes the list of the minimal surfaces projected on x-axis (resp. y-axis). The i[th] minimum surfaces are respectively denoted Smx_i and Smy_i. We then calculate the loss associated with each minimum surface, (noted $\text{Smx}_i.SP$ and resp. $Smy_i.SP$) such that:

$$Smx_i.SP = (Smx_i.\text{w} - \max_{j\in[0,Smx_i.w]}(j, Mx(i-1,j) = True)) \times Smx_i.h$$

$$Smy_i.SP = (Smy_i.\text{h} - \max_{j\in[0,Smy_i.h]}(j, My(i-1,j) = True)) \times Smy_i.w$$

The sum of these losses associated with each minimum surface gives us the minimum loss SP of the placement of object i such that:

$$\text{SP} = \sum_{Smx\in L_{Smx}} (Smx.w - \max_{j\in[0,Smx.w]} j\ s.t.\ M(i-1,j) = \text{True}) \times Smx.h + \sum_{Smy\in L_{Smy}} (Smy.h - \max_{j\in[0,Smy.h]} j\ s.t.\ M(i-1,j) = \text{True}) \times Smy.w$$

Example 1: let's take a bin (10,12) and 4 objects of different sizes ((4,5), (4,5), (4,8) and (5,8)). The object (5,8) is freely oriented. Initially, we sort the objects in ascending order based on the size of their surface. The freely oriented object is duplicated, and a bitwise XOR ensures that placing the object in one orientation will lead to place the object in its other orientation.

Then, we construct two matrices Mx and My visible in Figure 5.

We aim to place the most constraining object first. So we start by probing the freely oriented object. This object can be placed in either orientation, i.e. in the width orientation (8,5) or the height one (5,8).

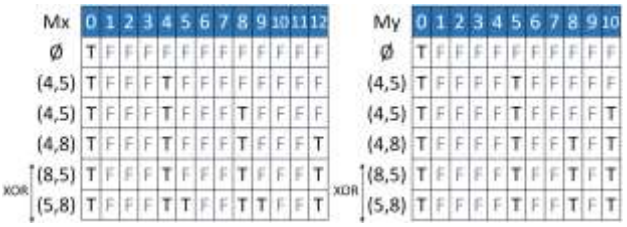

Figure 5: Example of Mx and My for the bin (12,10) and 4 objects: (4,5), (4,5), (4,8) and (5,8) with orientation free.

Placing this object in its height orientation (5,8) generates the two minimum surfaces on the x-axis : Smx_1 of size (7,8) and Smx_2 of size (12,2). These two minimum surfaces are shown in Figure 6.

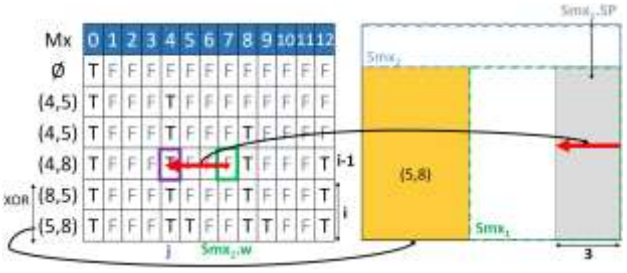

Fig. 6: Calculation of $Smx_1.SP$

Smx_1 is a surface on the x-axis, generated by placing the object i with orientation (5,8) in the bin of size (12,10). The surface in grey is the surface area of Smx_1, which will be lost.

For each minimum surface, we calculate its lost surface SP when the object is placed. Each minimum surface has a height of $Smx.h$ (resp. Smy.h) and width $Smx.w$ (resp. Smy.w).

Let's take the example of calculating the loss for minimum surface Smx_1. For this minimum surface Smx_1 of size (7,8), the surface that will be lost $Smx_1.SP$ is calculated as follows $Smx_1.SP = (Smx_1.w - \max_{j\in[0,Smx_1.w]}(j, Mx(i-1,j) = True)) \times Smx_1.h$ so $Smx_1.SP = (7-4) * 8 = 24$.

For Smx_2, the surface that will be lost is $Smx_2.SP = (12-12) * 2 = 0$.

We make the same calculation for the minimum surface on the y axis. Placing this object in its height orientation (5,8) generates the two minimum surfaces on the y-axis : Smy_1 of size (5,2) and Smy_2 of size (7,10). So, $Smy_1.SP = (2-0) * 5 = 10$ and $Smy_2.SP = (10-10) * 7 = 0$.

Placing this object in its height orientation (5,8) gives us: $SP = 24 + 0 + 10 + 0 = 34$.

Placing this object in its width orientation (8,5) generates the two minimum surfaces s on the x-axis : Smx_1 of size (4,5) and Smx_2 of size (12,5). These two minimum surfaces are shown in Figure 7.

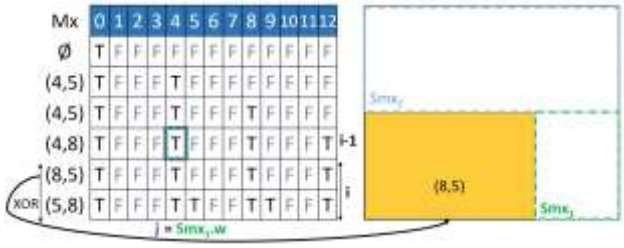

Fig. 7: Calculation of $Smx_1.SP$

Smx_1 is a surface on x-axis was generated by placing the object i with orientation (8,5) in a bin of size (12,10). Smx_1 and Smx_2 lead to no space lost because we have $Smx_1.SP = (4-4) * 5 = 0$ and $Smx_2.SP = (0-0) * 12 = 0$.

Placing this object in its width orientation (8,5) generates the two minimum surfaces on the y-axis : Smy_1 of size (8,5) and Smy_2 of size (4,10). So, $Smy_1.SP = (5-5) * 8 = 0$ and $Smy_2.SP = (10-10) * 4 = 0$.

So, placing this object in its width orientation (8,5) gives us: $SP = 0 + 0 + 0 + 0 = 0$.

In this example, placing the object in the direction of its height results in a minimum loss SP of 34. Placing the object in the direction of its width results in a minimum loss SP of 0. As we aim to minimize space lost, we detect that placing the object in the direction of its width leads to no lost and placing it in the direction of its height will lead to lose 34 units of space (according to the remaining objects). Then, we detect that the placement with the width orientation is the best one and chose it.

Example 2: let's take a bin (8,5) and 3 objects of different sizes ((1,2), (1,3) and (3,4)). The object (3,4) is freely oriented. Initially, we sort the objects in ascending order based on the size of their surface. The freely oriented object is duplicated, and a bitwise XOR ensures that placing the object in one orientation will lead to placing the object in its other orientation.

Then, we construct two matrices Mx and My visible in Figure 8.

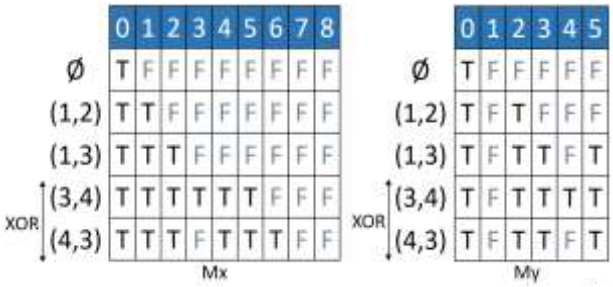

Fig. 8: Example of Mx and My for the bin (8,5) and 3 objects: (1,2), (1,3) and (3,4) with orientation free

We aim to place the most constraining object first. So we start by probing the freely oriented object. This object can be placed in either orientation, i.e. width (4,3) or height (3,4).

Placing this object in its width orientation (3,4) generates the two minimum surfaces s on the x-axis : Smx_1 of size (5,4) and Smx_2 of size (8,1). These two minimum surfaces are shown in Figure 9.

For each minimum surface, we calculate its lost surface SP when the object is placed. Each minimum surface has a height of $Smx.h$ (resp. Smy.h) and width $Smx.w$ (resp. Smy.w).

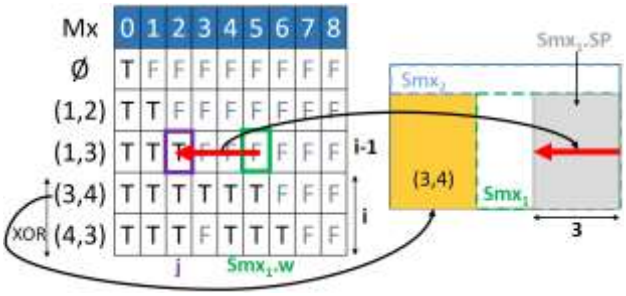

Fig. 9: Calculation of $Smx_1.SP$

Smx_1 is a surface on x-axis was generated by placing the object i with orientation (3,4) in a bin of size (8,5). The surface in grey is the surface area of Smx_1, which will be lost.

Let's take the example of calculating the loss for minimum surface Smx_1. For this minimum surface Smx_1 of size (5,4), the surface that will be lost $Smx_1.SP$ is calculate as following:
$Smx_1.SP = (Smx_1.w - \max_{j\in[0,Smx_1.w]}(j, Mx(i-1,j) = True)) \times Smx_1.h$ so $Smx_1.SP = (5-2) * 4 = 12$.

This calculation is performed for all minimum surfaces on the x-axis and the y-axis.

3.4.5 Algorithm

In the bin packing problem, for each object, we have to decide in which bin to pack it, in which location and in which orientation. In our case, this means determining in which maximum surface each object should be packed, and in which orientation if the object has a free orientation.

For each object, we look for the set of maximum surfaces in which we can place the object under consideration. For each of these maximum surfaces, we define a new list of maximum and minimum surfaces. So for each object, in each area that can accommodate it, we calculate the areas that will be irretrievably lost in the end. Then we choose to place the object in the maximum surface that will minimize the area lost at the end, such that:

$$\min(SP) = \min\Big(\sum_{Smx\in L_{Smx}} (Smx.w - \max_{j\in[0,Smx.w]} j\ s.t.\ M(i-1,j) = \text{True}) \times Smx.h + \sum_{Smy\in L_{Smy}} (Smy.h - \max_{j\in[0,Smy.h]} j\ s.t.\ M(i-1,j) = \text{True}) \times Smy.w\Big)$$

In our example, placing the object in the width direction (4,3) will result in a minimum loss of 21. Placing the object in the height direction (3,4) will result in a minimum loss of 18. It is this loss that allows us to choose the location and orientation of the object. We place the object in the location and orientation that minimize loss. So, this object is placed in the orientation of its height (3,4).

Consequently, every time an object is placed on a maximum surface, we can accurately estimate the additional space that will inevitably be lost by the end of the algorithm. It is this criteria of final loss that will guide the choice of placement for each object. If tie, we place the object i in the maximum surface SM such that: $min(SM.w - o_i.w)$ or $min(SM.h - o_i.h)$.

Our algorithm for solving two dimensional bin packing problem is detailed below:

Algorithm 1 pack(vector<Object>, vector<Bin>)

Require: List of objects *o* and their characteristics
Require: List of bins *b* and their characteristics
Ensure: List of object locations in bins
Step 1: Initializing the algorithm
Initialize list of maximum surface L_{SM}, list of minimum surfaces in x $\boldsymbol{L_{Smx}}$, list of minimum surfaces in y $\boldsymbol{L_{Smy}}$ for all available bins
Sort objects in ascending order of surface area.
Calculate the matrices Mx and My with objects sorted in ascending order of surface area.
Step 2: Algorithm
for each *i* object in descending order of surface area **do**
  for each maximum surface $SM \in L_{SM}$ **do**
    if $SM.w \geq o_i.w$ and $SM.h \geq o_i.h$ **then**
      Calculate L_{Smx} and L_{Smy}
      Calculate SP
      Calculate $t_x = SM.w - o_i.w$ and $t_y = SM.h - o_i.h$
    end if
  end for
  Choose maximum surface SM such that: $min(SP)$
  If tie, choose maximum surface SM such that: $min(t_x)$ or $min(t_y)$
  Place object *i* in surface *SM*, i.e. store the bin

number and bottom-left coordinates of the object
i in bin, update L_{Smx} and L_{Smy}, calculate new L_{SM}.
17: **end for**

Our algorithm avoids combinatorial exploration by making decisive object-to-position assignments in the bins at each step. To prevent making expensive decisions (far from the optimum), it integrates dynamic programming elements, ensuring both fast execution in terms of computing time and efficiency in terms of optimization. The algorithm's key strength lies in its comprehensive and enduring perspective on the impact of each decision made.

3.5 Industrial Problem

In the industrial context, companies need adaptable and modifiable solutions. The solution was modeled as a class diagram depicted in Figure 10.

The industrial problem (Shipping) is composed of several bin packing problems, one for each order. Each order is composed of several components and several panels. The problem can generate solution. A solution is the list of x, y coordinates for each component in a panel. A component has a width and a height. In the industrial problem, an object can be freely oriented or fixed. A panel type is characterized by its width and height. In the industrial context, they also have a peripheral margin in which no object can be positioned, and a DS spacing must be maintained between components on the same panel. However, for peripheral components, the components can be placed against the edges of the panel's technical margins.

To deal with this spacing constraint, you can:

- Artificially increase the dimensions of components (i.e. objects) by $DS/2$.
- Decrease the panel's technical margins by $DS/2$.
- Position components with an inter-components spacing of 0.

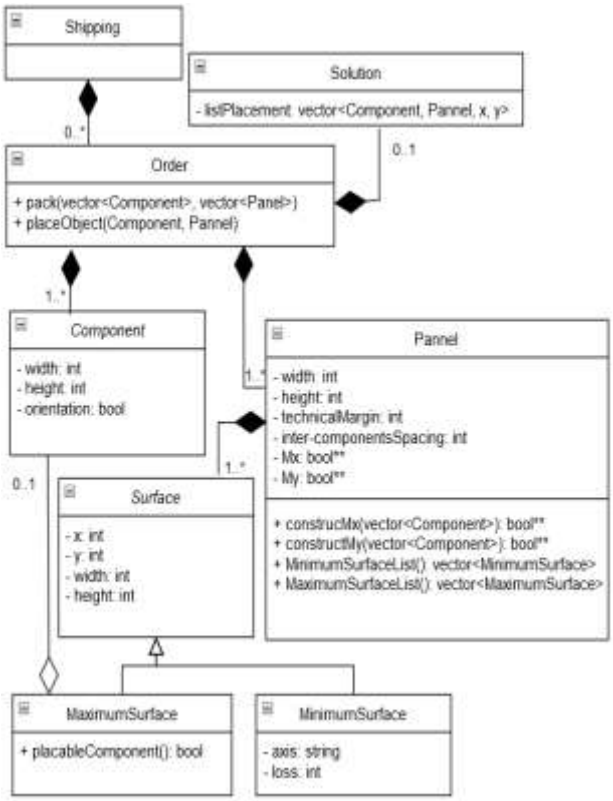

Fig. 10: Implemented class diagram for industrial problem

For each panel, we construct Mx and My matrices. Each panel has its list of maximum and minimum surface. The maximum surfaces have a function to determine whether it is possible to place an object in that area. If feasible, it helps us know the x, y position of the object in the panel. The minimum surfaces have an x or y axis and know their lost area for each placement of an object in a maximum surface.

Establishing relationships between panels and surfaces, as well as between surfaces and components, summarizes both the nature of the problem and the structure of the solution.

Depending on the industrial problem, there may be different types of constraints. This model is adaptable based on various constraints.

Incompatibilities between components and panels can be considered by adding a relationship between panels and components (component i forbidden in panel j).

Incompatibilities between components and zones in panels (the component must (or must not) be placed in such and such a zone of the panel, between 2 x coordinates and/or between 2 y coordinates) can also be considered. We could integrate this with our concept of surfaces.

Incompatibility between components can also be considered by adding a pair of component relationships.

4 Results

The C++ language was used to run the algorithm.

On some instances, it can be difficult to find the optimal solution and a solution close to the optimum. This is the case on instances where objects vary greatly in size. This complexity is even greater when the optimal arrangement of objects completely fills the bins. In this case, a single placement error can have a considerable influence on the addition of further containers. The solutions can then be very far from the optimal solution.

We tested our algorithm on several difficult extreme instances.

4.1 Instance Typology 1

To test our algorithm, we relied on the difficult extreme instances constructed in [3], for 1D-BPP.

The first instance typology in [3], is:

$$U = \{u_1, u_2, \dots, u_{18m}\},$$

$$s(u_i) = \begin{cases} \frac{1}{7} + \varepsilon & 1 \le i \le 6m \\ \frac{1}{3} + \varepsilon & 6m < i \le 12m \\ \frac{1}{2} + \varepsilon & 12m < i \le 18m \end{cases}$$

We have adapted this instance typology for 2D-BPP. To achieve this, we have taken both dimensions into account. This two-dimensional instance typology is:

$$U = \{u_1, u_2, \dots, u_{64m}\},$$

$$s(u_i) = \begin{cases} \frac{1}{7}b.w + \varepsilon, \frac{1}{7}b.h + \varepsilon & 1 \le i \le 6m \\ \frac{1}{3}b.w + \varepsilon, \frac{1}{7}b.h + \varepsilon & 6m < i \le 12m \\ \frac{1}{2}b.w + \varepsilon, \frac{1}{7}b.h + \varepsilon & 12m < i \le 18m \\ \frac{1}{7}b.w + \varepsilon, \frac{1}{3}b.h + \varepsilon & 18m < i \le 24m \\ \frac{1}{3}b.w + \varepsilon, \frac{1}{3}b.h + \varepsilon & 24m < i \le 30m \\ \frac{1}{2}b.w + \varepsilon, \frac{1}{3}b.h + \varepsilon & 30m < i \le 36m \\ \frac{1}{7}b.w + \varepsilon, \frac{1}{2}b.h + \varepsilon & 36m < i \le 42m \\ \frac{1}{3}b.w + \varepsilon, \frac{1}{2}b.h + \varepsilon & 42m < i \le 58m \\ \frac{1}{2}b.w + \varepsilon, \frac{1}{2}b.h + \varepsilon & 58m < i \le 64m \end{cases}$$

The optimal placement of this instance typology can be seen in Figure 11.

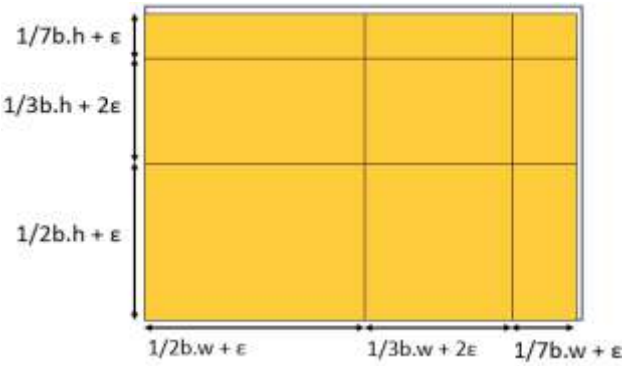

Fig. 11: Optimal placement of first instance typology of [3], implemented for two-dimensional bin packing problem

We can consider an infinite number of instances of this typology. The results are shown in Table 1.

Table 1. Results obtained on first instance typology of [3]

Nb objects	Opt	Our	Time (s)
64	6	6	0.05
128	12	12	0.07

4.2 Instance Typology 2

A second difficult extreme instances of typologies exist in [3], for 2D-BPP.

This second instances typologies in [3] is:

$$U = \{u_1, u_2, \dots, u_{30m}\},$$

$$s(u_i) = \begin{cases} \frac{1}{2} + \varepsilon & 1 < i \leq 6m \\ \frac{1}{4} + 2\varepsilon & 6m < i \leq 12m \\ \frac{1}{4} + \varepsilon & 12m < i \leq 18m \\ \frac{1}{4} - 2\varepsilon & 18m < i \leq 30m \end{cases}$$

As for the first instance typology of [3], we have adapted it to the 2D-BPP by taking into account the two dimensions.

Its two-dimensional adaptation is:

$$U = \{u_1, u_2, \dots, u_{75m}\},$$

$$s(u_i) = \begin{cases} \frac{1}{2}b.w + \varepsilon, \frac{1}{2}b.h + \varepsilon & 1 \leq i \leq 3m \\ \frac{1}{4}b.w + \varepsilon, \frac{1}{2}b.h + \varepsilon & 3m < i \leq 6m \\ \frac{1}{4}b.w - 2\varepsilon, \frac{1}{2}b.h + \varepsilon & 6m < i \leq 9m \\ \frac{1}{4}b.w + 2\varepsilon, \frac{1}{4}b.h + 2\varepsilon & 9m < i \leq 21m \\ \frac{1}{4}b.w - 2\varepsilon, \frac{1}{4}b.h + 2\varepsilon & 21m < i \leq 33m \\ \frac{1}{2}b.w + \varepsilon, \frac{1}{4}b.h + \varepsilon & 33m < i \leq 36m \\ \frac{1}{4}b.w + \varepsilon, \frac{1}{4}b.h + \varepsilon & 36m < i \leq 39m \\ \frac{1}{4}b.w - 2\varepsilon, \frac{1}{4}b.h + \varepsilon & 39m < i \leq 42m \\ \frac{1}{2}b.w + \varepsilon, \frac{1}{4}b.h - 2\varepsilon & 42 < i \leq 45m \\ \frac{1}{4}b.w + 2\varepsilon, \frac{1}{4}b.h - 2\varepsilon & 45m < i \leq 57m \\ \frac{1}{4}b.w + \varepsilon, \frac{1}{4}b.h - 2\varepsilon & 57m < i \leq 60m \\ \frac{1}{4}b.w - 2\varepsilon, \frac{1}{4}b.h - 2\varepsilon & 60m < i \leq 75m \end{cases}$$

The optimal placement of this instance type can be seen in Figure 12.

We test our algorithm on numerous kind of instances based on this typology. The results are shown in Table 2.

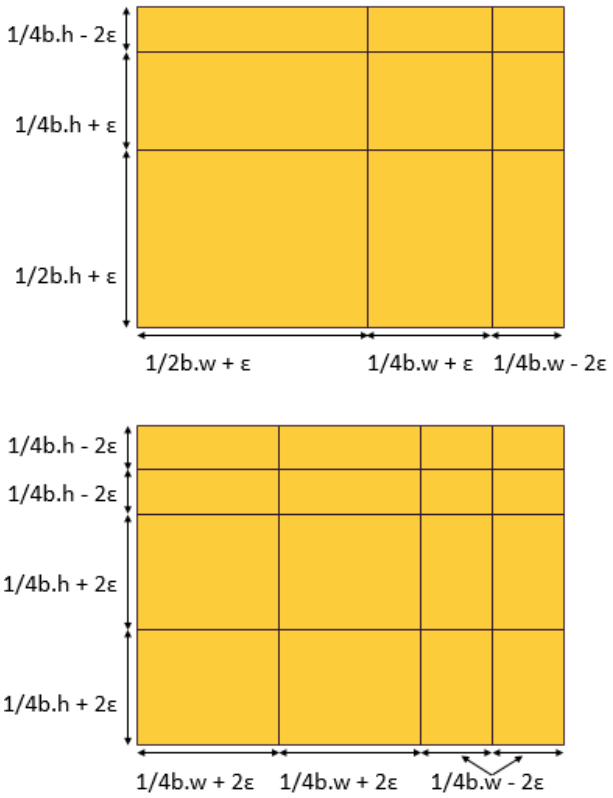

Fig. 12: Optimal placement of the second instance typology of [3], implemented for two-dimensional bin packing problem

Table 2. Results obtained on the second instance typology of [3]

Nb objects	Opt	Our	Time (s)
75	6	6	0.05
150	8	8	0.08

4.3 Instance Typology 3

To test the robustness of the algorithm in extreme scenarios, we generated numerous examples by randomly partitioning the bins to define the objects. This is done in such a way that the objects have a wide range of sizes. In these instances, the bins are full. Randomly slicing bins without loss to define the objects to be placed makes it possible to construct a perfect package layout design. This never happens in reality but allows us to evaluate the worst-case performance of the algorithm. The optimal solution tolerates no losses, and the slightest error in the algorithm takes us farther away from the optimal solution. To add complexity, all objects are freely oriented.

We built instances of reasonable size (between 10 and hundred objects). An example is visible in Figure 13. To check the complexity and computation times of our algorithm, we've built large instances (between 200 and a thousand

objects). An extract of the results obtained is given in Table 3.

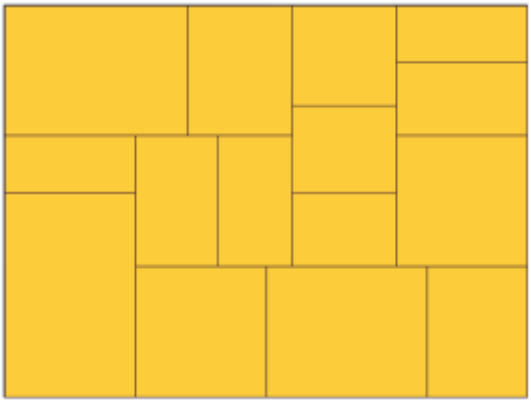

Fig. 13: Illustration of optimal placement for one bin of instance typology 3

Table 3. Results obtained on our instances

Nb objects	Opt	Our	Time (s)
44	4	4	0.05
47	3	4	0.06
51	3	4	0.07
57	3	3	0.08
66	6	7	0.07
440	40	40	5.5
470	30	31	6.5
510	30	31	7
570	30	30	9
660	60	61	28
1000	100	101	45

4.4 Discussion

All the results we give are obtained on extremely difficult instances (difficult topology, no losses allowed). On extremely difficult instances of , our algorithm finds the optimum solution for all these instances with a computation time of less than 1 second.

With our difficult extreme instances (instance typology 3), our algorithm offers an optimal solution or an approximation very close to the optimum in less than a minute. The most notable discrepancy we find is one bin more than the optimal solution. We have examined the reasons for this disparity and identified that it occurs when the evaluation functions (minimum loss at the end of processing and best saturation) are equal for several different surfaces. This situation, although rare, can occur, particularly when the number of bins is high, indicating the presence of many different maximum surfaces.

To prevent such cases, it would be sufficient to explore the decisions of both equalities, but this would risk making the algorithm combinatorial for minimal gains, i.e. just winning one bin. In all experiments, the difference does not exceed one extra bin.

On real results from the industrial context, our algorithm generates solutions that, at worst, are 97% of the optimal solution. On average, we obtain results over 99% of the optimal solutions, with a computation time of less than 1 minute for very large instances.

Unlike exact methods and metaheuristics, which take a long time to compute, and very fast heuristics, which in the worst case are far from optimum, our solution gives us an excellent compromise between solution quality (problem optimization) and computation time. Furthermore, it can handle very large instances on an industrial scale.

5 Conclusion

Our industrial objective was to place between a hundred and thousand freely rotating components (objects) in panels (bins) in less than one minute.

In this paper, we present an efficient algorithm designed to solve this problem, which is similar to the two-dimensional bin packing problem with free orientation.

Our algorithm produces results very close to the optimum (in the worst case, it generates just one more bin) in acceptable computation times (less than one minute) for large-scale instances (thousand objects) even taking into account the free orientation of the objects.

Although there is potential for further improvement by implementing a complete branch-and-cut algorithms incorporating all the lower bounds of the existing literature, such an extension may lead to a considerable increase in computation time for little additional result (just saving one bin).

According to our computation times and our object model, the perspectives of this work are very rich. We could envisage adapting our solution to other types of industrial problems with other types of constraints, such as cutting problems. Furthermore, our approach is easily adaptable to other industrial constraints such as:

- Different types of bins of different sizes. Simply create two matrices for each type of bin;
- Incompatibilities between objects and bins; The matrices only take into account objects compatible with selected surfaces in the bin;
- Etc.

We could also adapt it to 3D-BPPs such as packing, container and truck loading, palletization, etc. To do this, you need to add a depth attribute to the panel and the component and introduce a new attribute z for the surfaces. Then, we can easily modify our algorithm to consider this additional dimension. The introduction of an extra dimension will significantly increase calculation times but within reasonable limits.

References:

[1] M. Quiroz-Castellanos, L. Cruz-Reyes, J. Torres-Jimenez, C. Gómez, H. J. F. Huacuja, and A. C. Alvim, A grouping genetic algorithm with controlled gene transmission for the bin-packing problem, *Computers and Operations Research*, vol. 55, pp. 52-64, 2015.

[2] M. M. Baldi, T. G. Crainic, G. Perboli, and R. Tadei, Branch-and-price and beam search algorithms for the variable cost and size bin-packing problem, *Annals of Operations Research*, vol. 222, no. 1, pp. 125-141, 2014.

[3] M. R. Garey and D. S. Johnson, *Computer and intractability: A Guide to the Theory of -Completeness*, 1979.

[4] H. Dyckhoff, A typology of cutting and packing problems, *European Journal of Operational Research*, vol. 44, no. 2, pp. 145-159, 1990.

[5] G. Wäscher, H. Haußner, and H. Schumann, An improved typology of cutting and packing problems, *European Journal of Operational Research*, vol. 183, no. 3, pp. 1109-1130, 2007.

[6] G. Scheithauer, One-Dimensional Bin Packing, in Introduction to Cutting and Packing Optimization, International Series in Operations *Research and Management Science*, vol. 263, Springer, Cham, 2018.

[7] C. Munien and A. Ezugwu, Metaheuristic algorithms for one-dimensional bin-packing problems: A survey of recent advances and applications, *Journal of Intelligent Systems*, vol. 30, no. 1, pp. 636-663, 2021.

[8] K. H. Salem and Y. Kieffer, An experimental study on symmetry breaking constraints impact for the one-dimensional bin packing problem, in *2020 15th Conference on Computer Science and Information Systems (Fed-CSIS)*, IEEE, 2020.

[9] L. Wei, Z. Luo, R. Baldacci, R and A. Lim, A new branch-and-price-and-cut algorithm for one-dimensional bin-packing problems, *INFORMS Journal on Computing*, vol. 32, no. 2, pp. 428-443, 2020.

[10] K. Huang and Y. Dai, An intelligent algorithm integrated with fit algorithms for solving the one-dimensional bin packing problem under multiple length restrictions, in *Second International Conference on Algorithms, Microchips, and Network Applications (AMNA 2023)*, SPIE, 2023.

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

[12] D. Varsamis, and F. Chanlioglou, A parallel approach of best fit decreasing algorithm, *WSEAS Transactions on Computers*, 17(9), 79-85, 2018.

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

[14] P. Gilmore and R. Gomory, A linear programming approach to the cutting stock problem, *Operations Research*, vol. 9, pp. 849–859, 1961.

[15] P. Gilmore and R. Gomory, A linear programming approach to the cutting stock problem - part II, *Operations Research*, vol. 11, pp. 863–888, 1963.

[16] S. P. Fekete and J. Schepers, A combinatorial characterization of higher-dimensional orthogonal packing, Mathematics of *Operations Research*, vol. 29, no. 2, pp. 353-368, 2004.

[17] A. Pandey, *An analysis of solutions to the 2D bin packing problem and additional complexities*, Authorea Preprints, 2023.

[18] J.-F. Côté, M. Haouari, and M. Iori, Combinatorial benders decomposition for the two-dimensional bin packing problem, *INFORMS Journal on Computing*, vol. 33, no. 3, pp. 963-978, 2021.

[19] Z. Wang, D. Chang, and X. Man, Optimization of two-dimensional irregular bin packing problem considering slit distance and free rotation of pieces, *International Journal of Industrial Engineering Computations*, vol. 13, no. 4, pp. 491-506, 2022.

[20] S. Polyakovskiy and R. M'Hallah, Just-in-time two-dimensional bin packing, *Omega*, vol. 102, p. 102311, 2021.

[21] F. Gzara, S. Elhedhli, and B. C. Yildiz, The pallet loading problem: Three-dimensional bin packing with practical constraints, *European*

Journal of Operational Research, vol. 287, no. 3, pp. 1062-1074, 2020.

[22] Q. Zuo, X. Liu, L. Xu, L. Xiao, C. Xu, J. Liu, and W. K. V. Chan, The Three-dimensional Bin Packing Problem for Deformable Items, in *2022 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*, IEEE, 2022.

[23] B. Mahvash, A. Awasthi, and S. Chauhan, A column generation-based heuristic for the three-dimensional bin packing problem with rotation, *Journal of the Operational Research Society*, vol. 69, no. 1, pp. 78-90, 2018.

[24] J. Hasan, J. Kaabi, and Y. Harrath, Multi-objective 3D bin-packing problem, in 2019 *8th International Conference on Modeling Simulation and Applied Optimization (ICMSAO)*, IEEE, 2019.

[25] M. Daniela, I. Paul, and B. Alexandra. A topological order for a rectangular three-dimensional Bin Packing problem. In Proceedings of the *12th WSEAS international conference on Computers*, pp. 285-290, 2008.

[26] S. Hong, D. Zhang, H. C. Lau, X. Zeng, and Y. W. Si, A hybrid heuristic algorithm for the 2D variable-sized bin-packing problem, *European Journal of Operational Research*, vol. 238, no. 1, pp. 95-103, 2014.

[27] C. S. Chen, S. M. Lee, and Q. S. Shen, An analytical model for the container loading problem, *European Journal of Operational Research*, vol. 80, pp. 68-76, 1995.

[28] H. Onodera, Y. Taniguchi, and K. Tmaru, Branch-and-bound placement for building block layout, *28th ACM/IEEE Design Automation Conference*, pp. 433–439, 1991.

[29] D. Pisinger and M. Sigurd, The two-dimensional bin packing problem with variable bin sizes and costs, *Discrete Optimization*, vol. 2, no. 2, pp. 154-167, 2005.

Contribution of Individual Authors to the Creation of a Scientific Article (Ghostwriting Policy)

The authors equally contributed to the present research, at all stages from the formulation of the problem to the final findings and solution.

Sources of Funding for Research Presented in a Scientific Article or Scientific Article Itself

No funding was received for conducting this study.

Conflict of Interest

The authors have no conflicts of interest to declare.